Assignment 1
Neuroscience of Learning, Memory, Cognition
Dr. Karbalaei Aghajan

Mohammad Hossein Shafizadegan
99104781

March 7, 2023

Contents

1 Part I : python practice

1.1 Questions 1

Q : Search about pylab package and describe it shortly?

A : Pylab is a procedural interface to the Matplotlib object-oriented plotting library. It is a module that gets installed alongside Matplotlib.
Actually pylab is a convenience module that bulk imports matplotlib.pyplot (for plotting) and NumPy (for mathematics and working with arrays) in a single name space.
This package provides us with a MATLAB-like namespace and is very useful for plotting multipurpose curves using a Python programme

Q: What operation does multiply() perform?

In order to see how python multiplication using * operator and *multiply*() function works, we calculate the products using the following code and print the results.

```
    c = a * b
    d = multiply(a, b)

    print("a * b = ", c)
    print("using multiply function : ", d)
```

results :

```
a * b =  [[ 0  8 21 24]]
using multiply function :  [[ 0  8 21 24]]
```

It can be seen that the 'multiply()' function, calculates the product of each same elements of arrays and the result is a vector so this operation is an element-wise multiplication.

1.2 Question 2

After simply completing the delivered function, by using the following code we check the performance of the function :

```
    arr = array([[4, 5, 2, 9]])
    c = 3
    arr2 = my_square_function(arr, c)

    print("The input array : ", arr)
    print("Constant added to squared array : ", c)
    print("result = ", arr2)
```

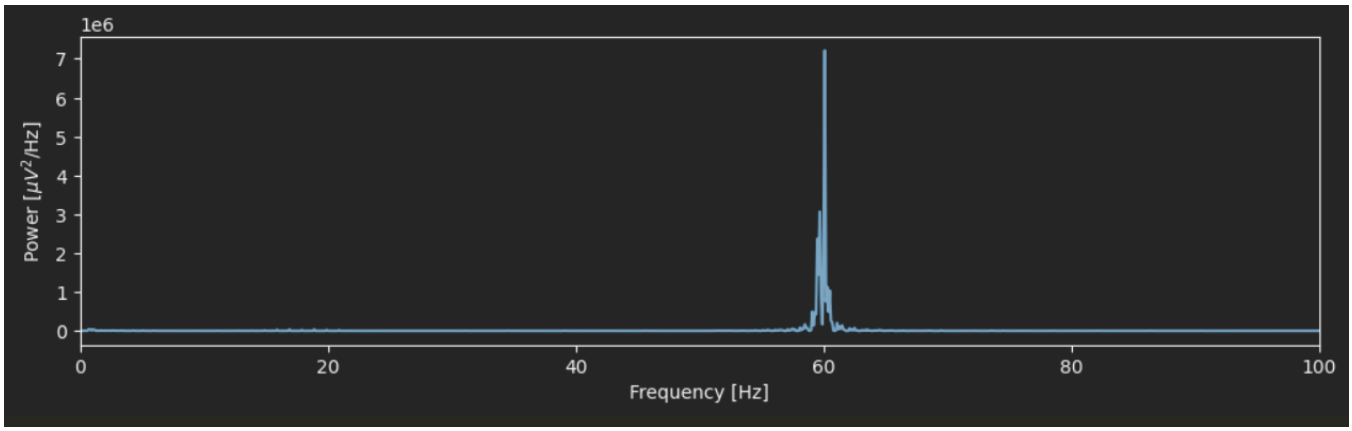
results :

```
The input array :  [[4 5 2 9]]
Constant added to squared array :  3
result =  [[19 28  7 84]]
```

1.3 Question 3

One simple way to calculate the fft of a signal is using the numpy $fft()$ function.
Regarding that for calculating Power Spectral Density we should just calculate the square of magnitude of the signal's

fft, so the code will be : Sxx = abs(xf)**2
Since the signal is real, the fft and PSD of the signal are even functions so the negative frequencies can be ignored.



2 PART II: NEURON MODELS

2.1 Coding Exercise 1: Python code to simulate the LIF neuron

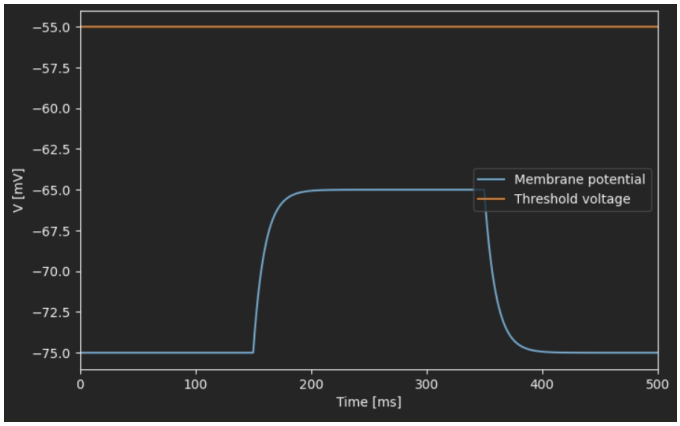
In the *run_LIF*() function, after setting and defining the differential equation parameters using the pars dictionary for simple access, we will create the output voltage and the input current vector with length of Lt.
This is how the code works :

- If the input argument "stop" equals to "True", we should set the beginning and end of the injected current to zero.
- In order to solve the differential equation, we will use an iterative process over time.
- In the iteration loop, first for simulating the refractory period which affects the voltage after a spike occurs, we check if the neuron is in refractory period or not. If yes, it sets the voltage to V_{reset} and decreases tr by 1.
- After that we check if the voltage exceeds V_{th} or not. If yes, it records a spike event in *rec_spikes*, sets the voltage to V_{reset} and sets tr to tref / dt.
- In process of solving the differential equation, after calculating the dv based on the equation, the value of the next iteration of the output voltage equals to summation of the previous iteration value and the corresponding dv so we update the value of voltage by adding the calculated dv.
- rec_spikes array is using index units format but we desire ms units. This can be achieved by multiplying the array with dt.

Now for using the defined function and seeing the results for a $100\,pA$ injected current and finally visualizing the results, we have used the following code :

```
    # Get parameters
    pars = default_pars(T=500)

    # Simulate LIF model
    v, sp = run_LIF(pars, Iinj=100)
    # Visualize
    t = np.arange(0,500,pars['dt'])
    plt.plot(t, v, label='Membrane potential') #plotting membrane voltage
    plt.xlim([0, 500])
```

```
    plt.xlabel('Time [ms]')
    plt.ylabel('V [$mV]')
    plt.plot(t,pars['V_th']*np.ones(v.shape[0]), label='Threshold voltage') # plotting the threshold
        voltage
    plt.legend(loc="center right")
    plt.show()
```



It can be seen that since the membrane voltage has not reached the threshold voltage, there is no spike at all. Also the effects of the time constant can be seen when the voltage is going to rise ($t \cong 150\ ms$) and when the voltage is going to decrease to the initial voltage ($t \cong 350\ ms$)

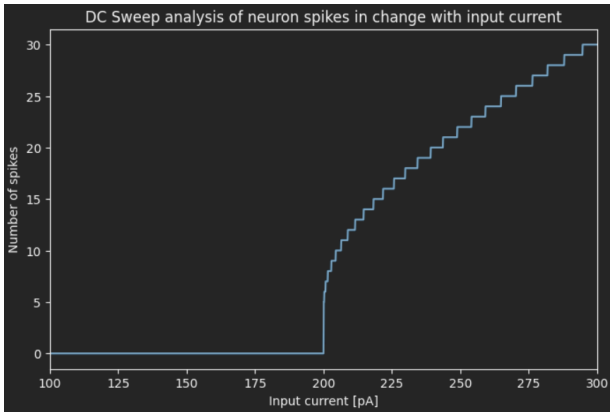
2.2 Response of an LIF model to different types of input currents

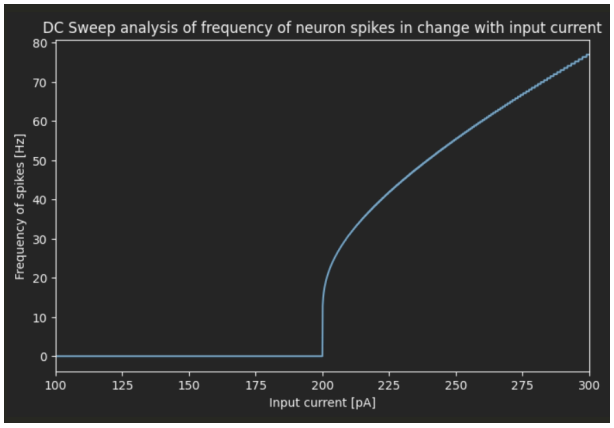
Parameter exploration of Direct current (DC) input amplitude

In order to observe the effects of DC input current, we'll check the number of spikes and frequency of them for different values of the injected current so we run a DC sweep analysis over the current.

```
    Iin = np.arange(100,300,0.1)
    sp_num = np.zeros(Iin.shape[0])
    sp_freq = np.zeros(Iin.shape[0])

    for i in range(Iin.size):

       v , sp = run_LIF(pars, Iinj=Iin[i])
       sp_num[i] = sp.shape[0]

       if(sp.shape[0] == 0):
          sp_freq[i] = 0
       else:
          sp_freq[i] = 1/(sp[1] - sp[0]) * 1000
```

Based on the following results, it can be seen that by increasing the input DC current, the number of spikes and their frequency will increase.
Note that the relation of number of spikes and their frequency with the input DC current is not linear.





For calculating the **rheobase current** we have developed the following function. Using this function, we increase the input current step by step and check if the neuron spikes or not and then report the minimum current.

```
def find_rheobase_current(pars, init_I):

I = init_I
finish = False
while(not finish):
   v , sp = run_LIF(pars, Iinj=I)
   sp_num = sp.shape[0]
   I = I + 1
   if(sp_num > 0):
      finish = True

return I-1
```

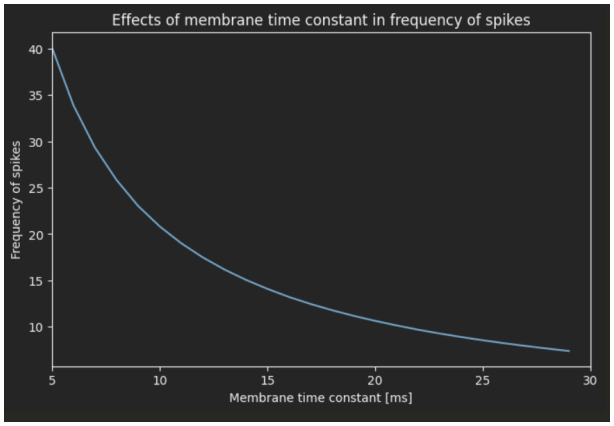
result :

```
The rheobase current equals to  201 pA.
```

It can be seen that for a DC current as input, the minimum current needed for neuron to spike is 201 pA.

Effects of membrane time constant

In a loop we change the values of the membrane time constant (tau_m), and calculate the spike frequency as we do before.



The smaller the time constant is, the more agile the neuron will be, so when the time constant is low the frequency of spikes is higher as we expected.

LIF neuron Explorer for noisy input

for completing the my_GWN function, we will use the following code.

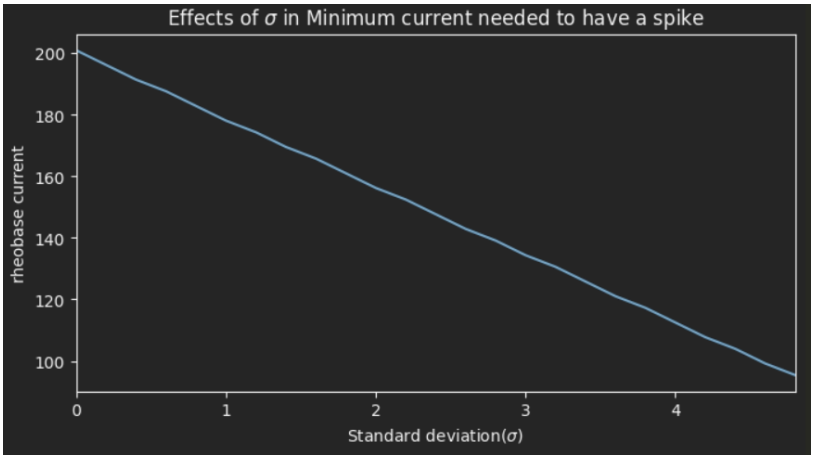
```
    # Set random seed
    np.random.seed(myseed)

    # Generate GWN and convert units to sec.
    I_gwn = mu + sig*100 * np.random.randn(Lt)
```

We should pay careful attention that unit of the sigma as the input argument of function is in seconds while in our LIF model, simulation time and time steps are all in unit of ms so if we add a noise with 100σ to the signal and then calculate the average in unit of second the resulting noise has parameter of σ because by averaging data the noise will be reduced. That is why we have multiplied σ by 100 in the code.

Q : how does the minimum input (i.e., μ) needed to make a neuron spike change with increase in σ ?

To check the effect of σ, we run a DC sweep analysis over the σ and calculate the minimum DC current needed for neuron to spike using the previously developed function "find_rheobase_current()" and finally visualize the results.

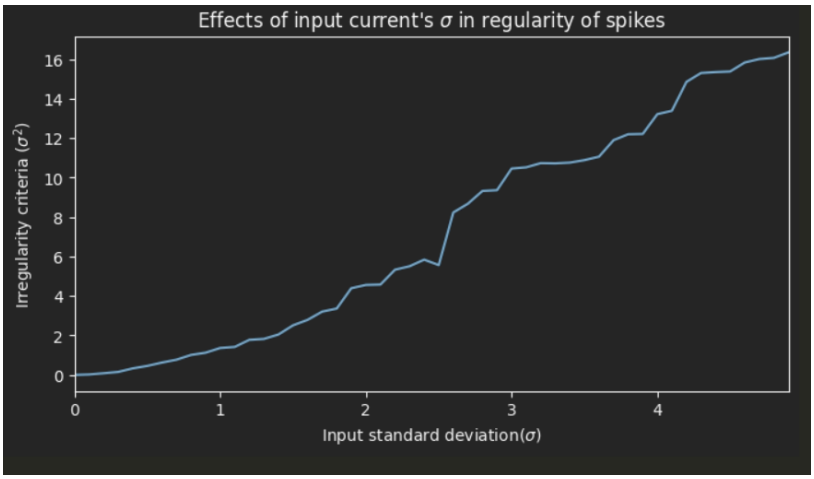
```
    sig = np.arange(0,5,0.2)
    myseed = 2020
    min_current = np.zeros(sig.shape[0])
    pars = default_pars()

    for i in range(sig.size):
    Iinj = my_GWN(pars,90,sig[i],myseed)
    min_current[i] = np.mean(find_rheobase_current(pars,Iinj))
```



As we increase the value of σ, the possibility to reach higher currents with a lower mean will increase so as we expected and as we can see in the above plot, the minimum DC input current needed to have a spike will decrease as we increase the σ.

Q : how does the spike regularity change with increase in σ ?
One way to see how irregular is the spikes behavior is to check the time difference of adjacent spikes and see how different they are. For representing this irregularity with only one variable, its recommended to use variance of the time difference array.

```
    sig = np.arange(0,5,0.1)
    mu = 250
    myseed = 2020
    irregularity = np.zeros(sig.shape[0])
    pars = default_pars()

    for i in range(sig.size):
    Iinj = my_GWN(pars,mu,sig[i],myseed)
    v , sp = run_LIF(pars, Iinj)
    irregularity[i] = np.var(np.diff(sp))
```



Its clear that when $\sigma = 0$ and therefor we have a DC input, the spikes are completely regular while they become more irregular as we increase the σ of input current.

Analyzing GWN Effects on Spiking

As we discussed and showed before, by in increasing the input DC current the number of spikes will also increase. It seems that the greater the input DC current will be, the more spikes we'll have but I think that the membrane time constant will set a limit in number of spikes in a specified time interval.
Also by increasing the mean of the GWN current, the number of spikes will also increase.
Also we discussed that by increasing the σ of the GWN current the spikes will become more irregular.

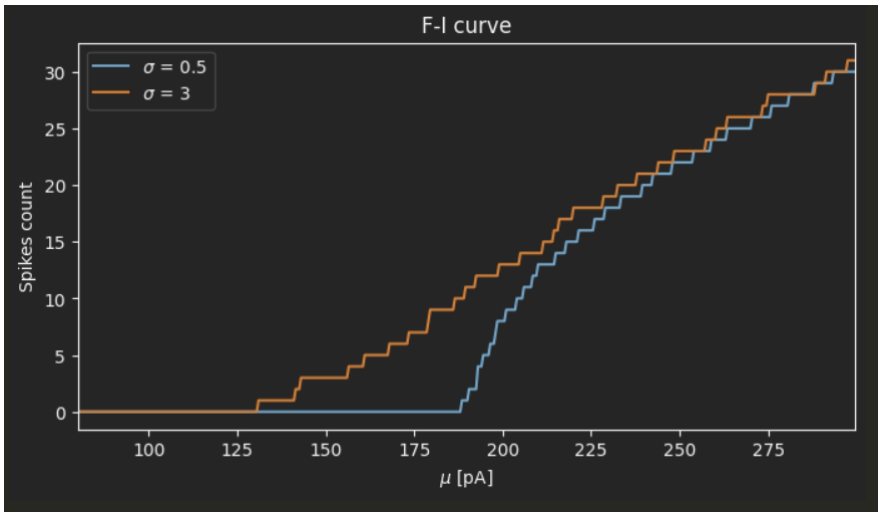
The Interspike interval (ISI)

The interspike interval is the time between subsequent action potentials (also known as spikes) of a neuron.
The interspike interval can be used to measure the firing rate and the variability of neuronal activity.
A common way to visualize the interspike interval is by using an interspike interval histogram, which shows the frequency distribution of different intervals.
The concept of interspike interval is based on the idea that the timing of action potentials can convey information about neuronal activity and communication.
They can also reflect the balance of excitatory and inhibitory inputs that affect neuronal drive

F-I Explorer for different 'sig_gwn'

F-I explorer In order to plot the F-I curve we have to find the firing rate for different values of μ. In this part we just plot the F-I curve for 2 different values of σ in a same figure.
Since for a GWN input current the spikes behave irregularly, the frequency of them is not a constant value so here in order to check the firing rate we just calculate the total number of spikes.

```
mu = np.arange(80,300,0.5)
sigma = [0.5, 3]
myseed = 2020
sp_freq_1 = np.zeros(mu.shape[0])
sp_freq_2 = np.zeros(mu.shape[0])
pars = default_pars()

for i in range(mu.size):
   Iinj_1 = my_GWN(pars,mu[i],sigma[0],myseed)
   v_1 , sp_1 = run_LIF(pars, Iinj_1)
   Iinj_2 = my_GWN(pars,mu[i],sigma[1],myseed)
   v_2 , sp_2 = run_LIF(pars, Iinj_2)

   sp_freq_1[i] = len(sp_1)
   sp_freq_2[i] = len(sp_2)
```



It can be clearly seen that for greater σ, spike fire rating starts in lower mean of current. Also the irregularity and stochastic behavior of spikes for a greater value of σ are vivid.

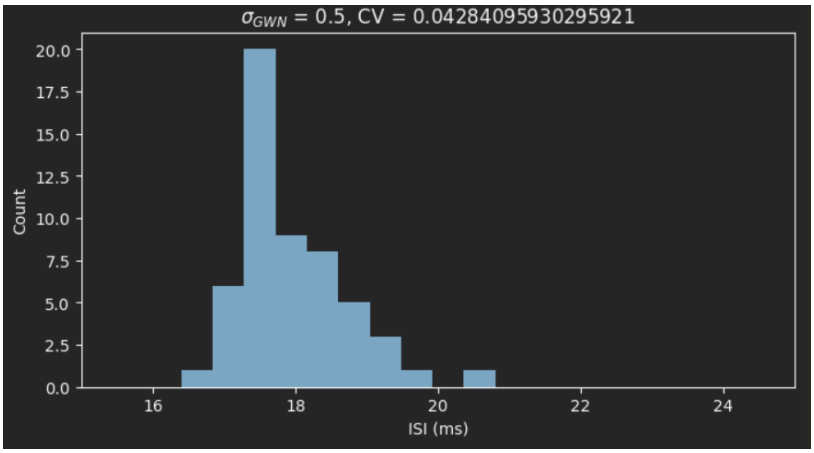
Compute CV_{ISI} values

In the $isi\_cv\_LIF()$ function, we should first calculate the time difference of the adjacent spikes and store them in an array. The best way to do so is to use $np.diff()$ built-in function.
Based on the formula used to calculate the CV_{ISI}. After calculating the standard deviation of isi array using $np.std()$ and also calculating the mean of it using $np.mean()$, we will calculate the value of CV_{ISI}.

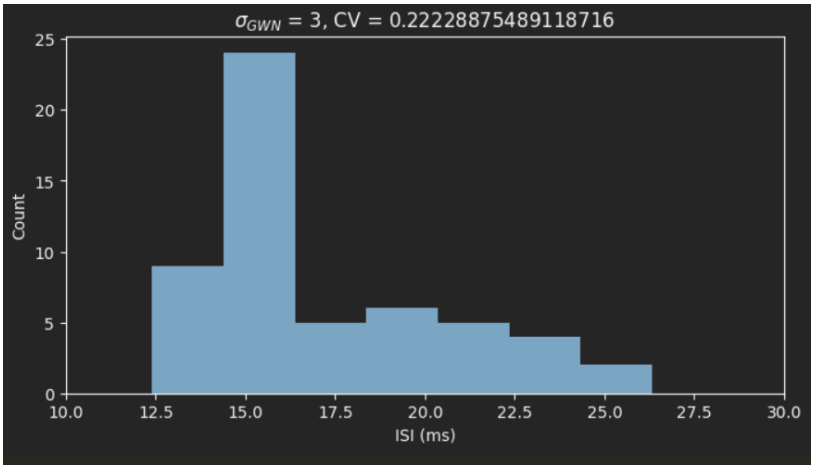
$$CV_{ISI} = \frac{std(ISI)}{mean(ISI)}$$

```
    if len(spike_times) >= 2:
        # Compute isi
        isi = np.diff(spike_times)
        # Compute cv
        cv = np.std(isi) / np.mean(isi)
    else:
        isi = np.nan
        cv = np.nan
```

Finally in order to visualize the result of the CV_{ISI} calculated for each value of standard deviation in a different figures, we will use $plt.hist()$ function and we will give the function the values of each ISI array and also we will report the calculated values of CV_{ISI} in plot's title.



In this case that $\sigma = 0.5$, different values of ISI are approximately in range of 16 ms to 20 ms which show us the fact that the neuron's spikes behavior is more irregular than their behavior for a DC input ($\sigma = 0$).



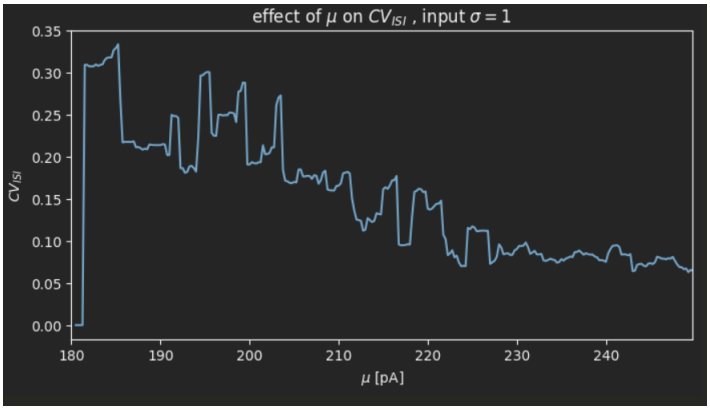
Now $\sigma = 3$ and consequently the behavior of spikes has become more irregular in comparison to the previous case as we expected. Different values of ISI are roughly in range of 12.5 ms to 26 ms.

Spike irregularity explorer for different 'sig_gwn'

1. As we saw here different values of standard deviation will highly affect the regularity of neuron's spike behavior. The greater the value of σ, the more irregular behavior. Also it can be seen that for greater values of σ, a smaller amount of average current is needed for neuron to spike.

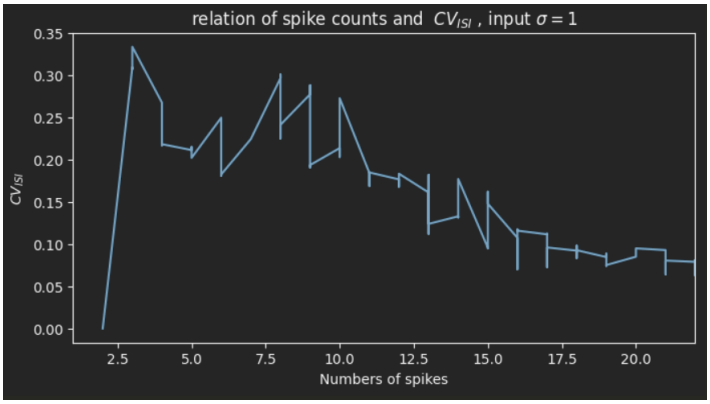
2. For observing the effects of the mean of the GWN input on CV_{ISI}, we have calculated the corresponding value of CV_{ISI} for different values of μ. Finally we have visualized the results by the following code :

```
        for i in range(mu.size):
        Iinj = my_GWN(pars,mu[i],sigma,myseed)
        v_1 , sp = run_LIF(pars, Iinj)
        _, cv = isi_cv_LIF(sp)
        cv_isi[i] = cv
```



It can be seen that generally by increasing the values of μ, The CV_{ISI} will decrease.
Increasing the mean input current of LIF model makes it fire more frequently and regularly. Also with greater values of μ, the neuron is less affected by the noise or variability of the input current, which reduces its firing irregularity

3. By plotting spike count (or rate) vs. CV_{ISI} for different input currents, we will see a generally inverse relationship: as spike count (or rate) increases, CV_{ISI} decreases.
This is because higher input currents make the neuron fire more and consequently since the fact that for higher μ noise will affect spikes less, neuron will fire more frequently and regularly. Conversely, lower input currents make the neuron fire less frequently and more irregularly



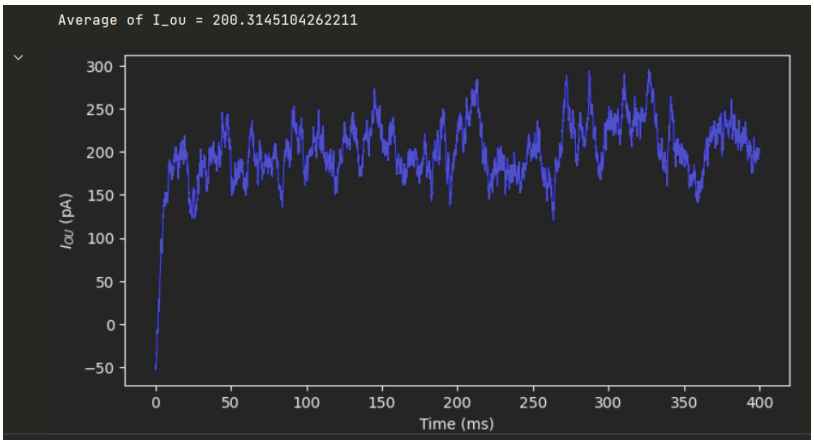
2.3 Ornstein-Uhlenbeck Process

According to the following code we will complete the $my\_OU()$ function based on the given differential equation which results in colored noise current.

```
    for it in range(Lt-1):
    dI = (dt/tau_ou) * (mu-I_ou[it]) + np.sqrt(2*dt/tau_ou)*sig*noise[it+1]
    I_ou[it+1] = I_ou[it] + dI
```

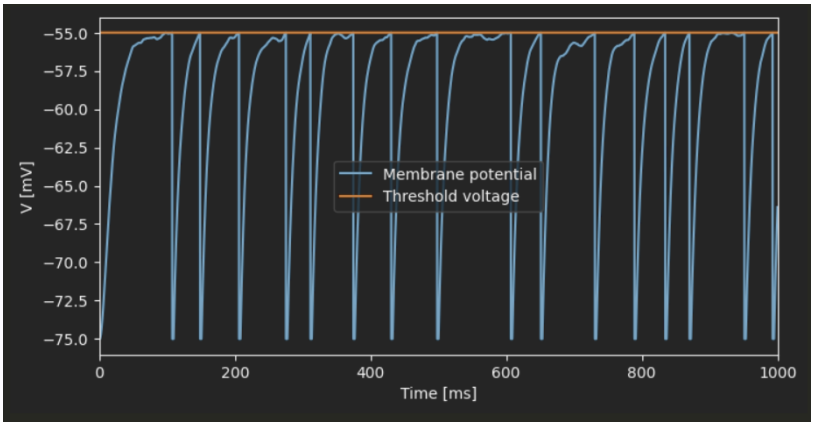
In order to have a sense that how this kind of noise behaves, we will plot the I_{ou} and also we will calculate it's average.

```
    pars = default_pars()
    pars['tau_ou'] = 5
    I_ou = my_OU(pars, 200, 30, 2020)
    print("Average of I_ou = " + str(np.mean(I_ou)))
    plt.plot(pars['range_t'], I_ou, 'b', lw=1.0)
    plt.xlabel('Time (ms)')
    plt.ylabel(r'$I_{OU}$ (pA)')
    plt.show()
```



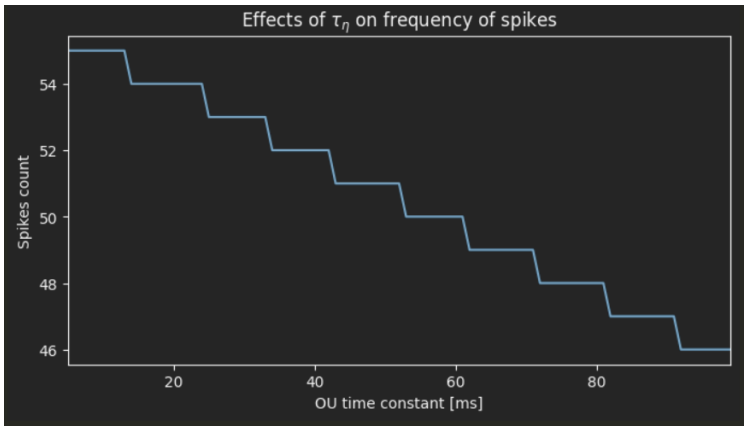
The τ_η parameter determines how much time it takes that amplitude of current reaches the value of μ.

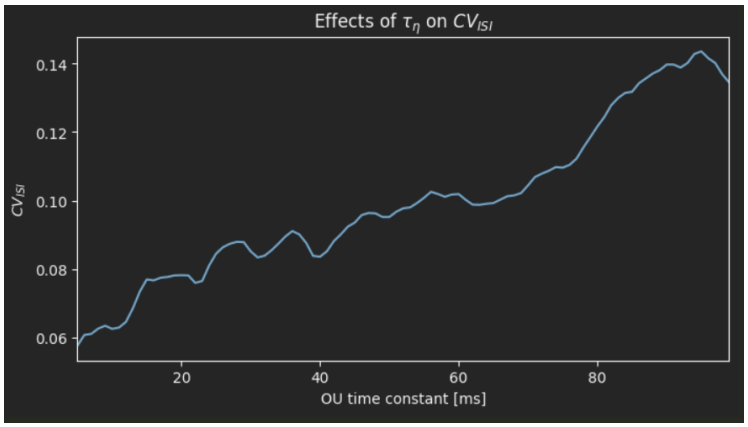
Now by using the $run\_LIF()$ function for a duration of 1 second, and $\mu = 200$, $\sigma = 5$, $\tau_\eta = 10$, we will plot the membrane potential and will see how the neuron behave.



Using the following code we will change the value of τ_η and then we will calculate the number of spikes and CV_{ISI} (which is a criteria for spikes irregularity) for each value of τ_η to see how τ_η will affect spike pattern and rate.

```
tau = np.arange(5,100,1)
sp_num = np.zeros(tau.shape[0])
cv_isi = np.zeros(tau.shape[0])

pars = default_pars(T=1000)
mu = 250
sig = 10

for i in range(tau.size):

pars['tau_ou'] = tau[i]
Iinj = my_OU(pars, mu, sig, 2020)
v , sp = run_LIF(pars, Iinj)
isi, cv_isi[i] = isi_cv_LIF(sp)
sp_num[i] = len(sp)
```





Based on the above diagrams, it can be seen that by increasing OU process time constant, it takes more time for the current to reach the values close to the average (μ) so behavior of spikes will change. For greater values of τ_η number of spikes and actually firing rate will decrease. Meanwhile based on the diagrams, spikes behavior will become more irregular. This can be even seen from the plot of membrane potential created by $run\_LIF()$ mode in the previous page. We should also remember that the CV_{ISI} depends of the σ of the input noise too.

2.4 Extensions to Integrate-and-Fire models

Generalized Integrate-and-Fire models are a type of neuron models that can capture some of the fundamental behaviors of real neurons. They consist of a voltage equation that describes how the membrane potential changes over time and a reset rule that determines when and how a spike is generated.
There are different variants of these models with different levels of complexity and accuracy. Some examples are :

1. **The exponential integrate-and-fire model (EIF) :**
   This model adds a nonlinear term to the voltage equation to account for the sharp rise of membrane potential near threshold voltage.
   The differential equation used to introduce membrane potential behavior is as follows :

$$\frac{dV}{dt} = -g_L(V - E_L) + g_L\Delta_T \exp(\frac{V - V_T}{\Delta_T}) + \frac{I}{C}$$

   where V is the membrane potential, g_L is the leak conductance, E_L is the leak reversal potential, Δ_T is the slope factor of the exponential term, V_T is the threshold potential, I is the input current and C is the membrane capacitance.
   The EIF model can capture some of the features of real neurons, such as adaptation and bursting3. However, it does not describe the biophysical processes that shape the time course of an action potential

2. **The adaptive exponential integrate-and-fire model (AdEx) :**
   This model incorporates an adaptation variable that modulates the threshold and the reset potential after each spike.The AdEx model is a spiking neuron model with two variables: voltage and adaptation1. It can simulate different types of neurons by adjusting its parameters.
   The differential equation for the variables are :

$$C\frac{dV}{dt} = -g_L(V - E_L) + g_L\Delta_T \exp\left(\frac{V - V_T}{\Delta_T}\right) + I - w$$
$$\tau_w \frac{dw}{dt} = a(V - E_L) - w$$

   where V is the membrane potential, w is the adaptation current, C is the membrane capacitance, g_L is the leak conductance, E_L is the leak reversal potential, Δ_T is the slope factor, V_T is the threshold potential, I is the input current, τ_w is the adaptation time constant and a is the subthreshold adaptation.

3. **The generalized exponential integrate-and-fire model (GEM) :** This model combines the exponential nonlinearity with a subthreshold variable that affects both the voltage and the adaptation dynamics.

$$C\frac{dV}{dt} = -g_L(V - E_L) + g_L\Delta_T \exp\left(\frac{V - V_T}{\Delta_T}\right) + I - g_x x$$
$$\tau_x \frac{dx}{dt} = -x + f(V)$$

   where V is the membrane potential, x is the subthreshold variable, C is the membrane capacitance,g_L is the leak conductance, E_L is the leak reversal potential, Δ_T is the slope factor,V_T is the threshold potential, I is the input current, g_x is a coupling conductance between V and x, τ_x is a time constant for x and $f(V)$ is a function of V.

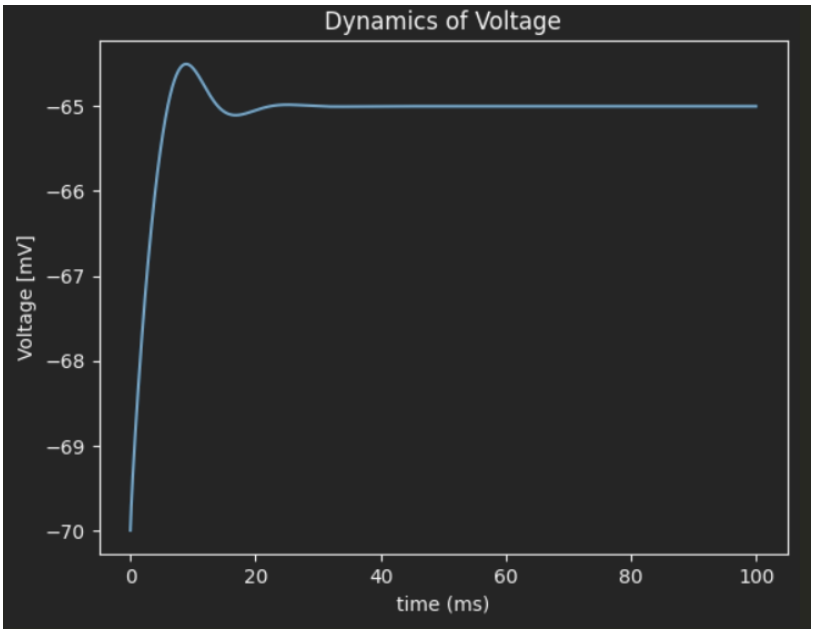
2.5 The Hodgkin-Huxley model

Explaining the code :
In order to obey clean code rules and more simplicity, we have defined functions for defining the formulas of $\alpha_N(v)$, $\alpha_M(v)$, $\alpha_H(v)$, $\beta_N(v)$, $\beta_M(v)$, $\beta_H(v)$.
After that in the HH function, first we will define the parameters and constants that we will use in our differential equations. Then we will create the desired variable v , t , m , n , h and set the initial values of them.
Using the given differential equations and the following formulas we will complete the iterative process used to solve these non-linear differential equation.

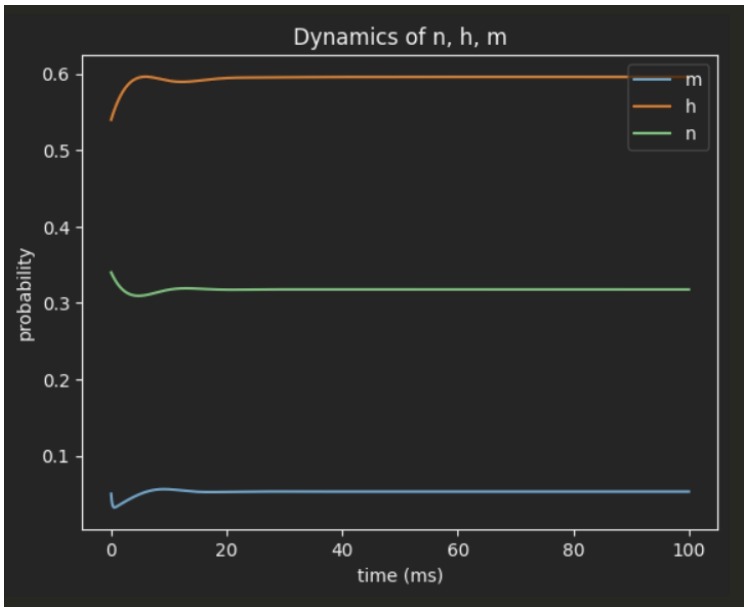
$$g_{Na} = g_{Na_0} m^3(v) h(v) \qquad , \qquad g_K = g_{K_0}\, n^4(v)$$

```
    for i in range(0,T-1):

      V[i+1] = V[i] + dt * (-1*gNa0*m[i]**3*h[i]*(V[i]+65 - ENa) - gK0*n[i]**4*(V[i]+65 - EK) -
          gL0*(V[i]+65 - EL) + I0)
      m[i+1] = m[i] + dt * (alphaM(V[i])*(1-m[i]) - betaM(V[i])*m[i])
      h[i+1] = h[i] + dt * (alphaH(V[i])*(1-h[i]) - betaH(V[i])*h[i])
      n[i+1] = n[i] + dt * (alphaN(V[i])*(1-n[i]) - betaN(V[i])*n[i])
```

For observing and visualizing the neuron membrane potential and gates behavior and dynamics, we will plot membrane potential and gate variables as a function of time.



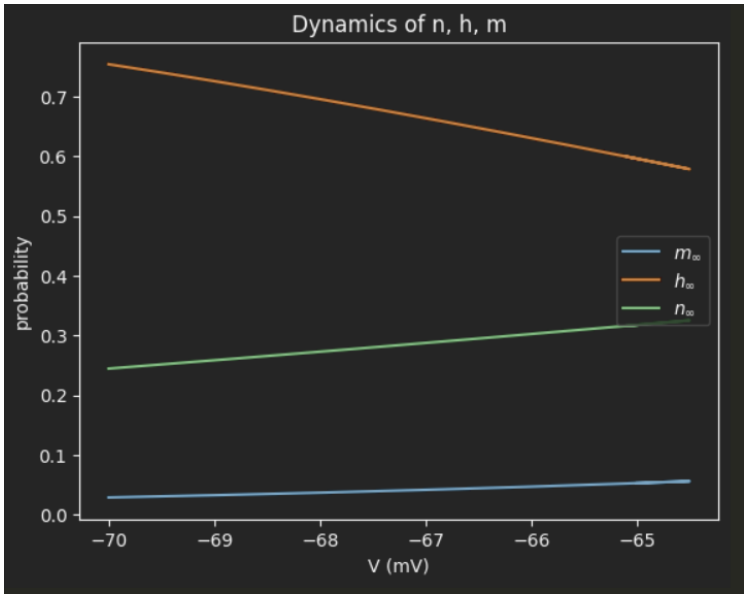
As we can see, since there is no external input current to the neuron, we don't expect the observe reasonable change in membrane potential but because of membrane leakage current and leakage conductance the potential of membrane will change a little. After some time, the potential will become stable and fixed. Also we notice that there is no spike.
The final value for the membrane potential in this case equals to $-65\ mV$ after $100\ ms$.
These behaviors can be seen in the above diagram.



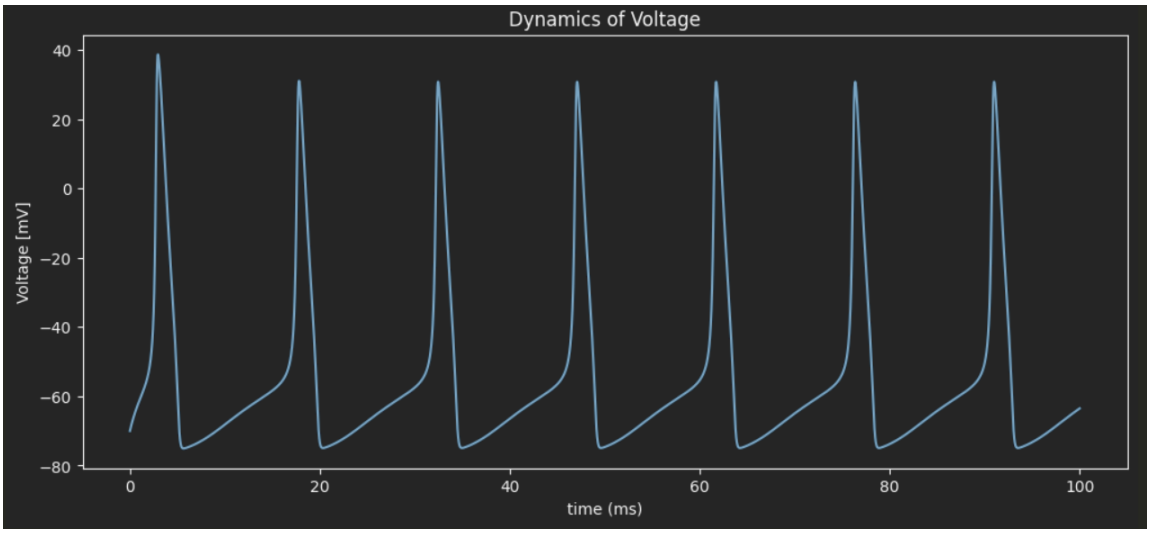
We know that the probability that the gating variable n,m and h be open, is also dependent to the membrane potential. Based on the previous diagram which shows the dynamic of membrane potential, there is no reasonable and noticeable change in membrane potential that results in spike firing because there is no stimuli or external current. Therefor, there is no reason that sodium ions enter the neuron via their channels and also there is no reason that the potassium ions exit the neuron so we don't expect any reasonable change in probability of being open for gating variables.
This kind of behavior can be completely seen in the above figure.
Also according to the above diagram the final values of n,m and h are roughly equal to their initial values.

For better visualization and comprehension we have also computed the steady values of gating variable as a function of membrane voltage and also plot the results using following formulas:

$$n_\infty(v) = \frac{\alpha_n(v)}{\alpha_n(v) + \beta_n(v)} \quad , \quad m_\infty(v) = \frac{\alpha_m(v)}{\alpha_m(v) + \beta_m(v)} \quad , \quad h_\infty(v) = \frac{\alpha_h(v)}{\alpha_h(v) + \beta_h(v)}$$

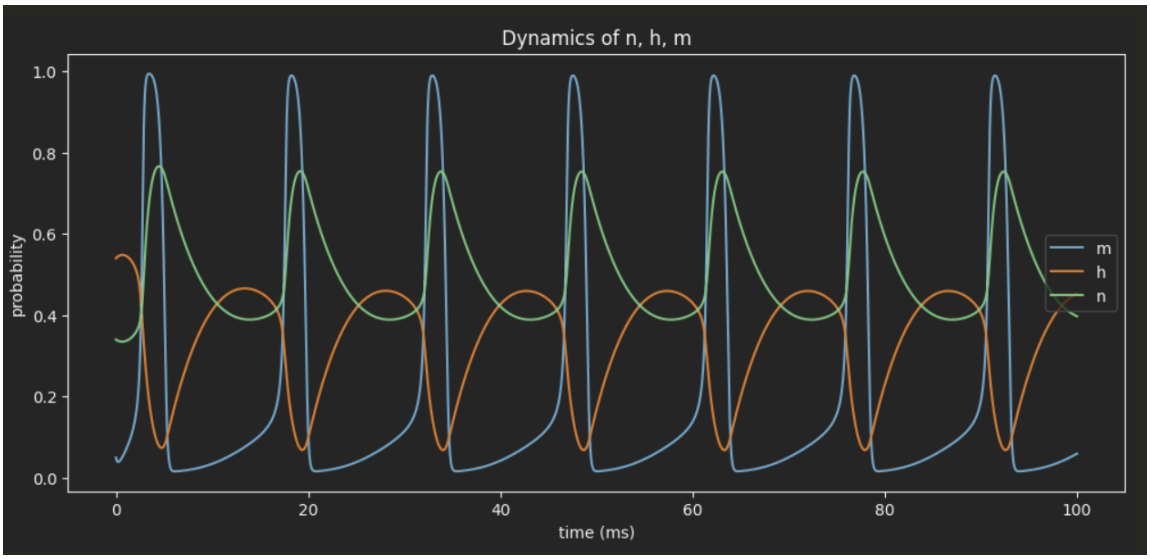


Now in this section we will apply a non-zero current to the neuron.



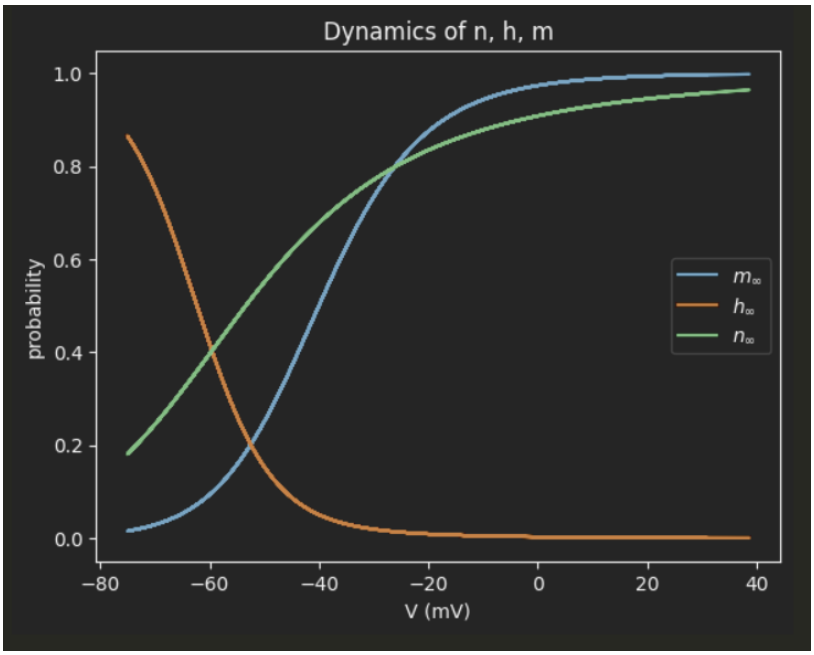
The stimuli current is big enough that the membrane potential has increased and reached the threshold voltage so as we can see in the above diagram, the neuron starts firing spikes periodically.
Different steps of a spike voltage behavior including the depolarization, re-polarization and hyperploarization can be seen vividly and perfectly.

The dynamic behavior of gating variable are as follows :



We will discuss the gating variables behavior during a single spike. Through time, the sodium channel will be opened thus the membrane potential will increase to E_{Na}. Increase in voltage further increases the current in a positive feedback so m will increase.
Meanwhile as voltage increases the value of h gate will decrease to close the sodium channel and stop the increase in membrane voltage. The sodium channel hence starts to close.
The potassium channel also starts to open with higher membrane voltage. As potassium ions flow and exit the neuron, the membrane voltage moves to E_K. Finally the voltage will raise the resting voltage and these steps will occur periodically for each spike.

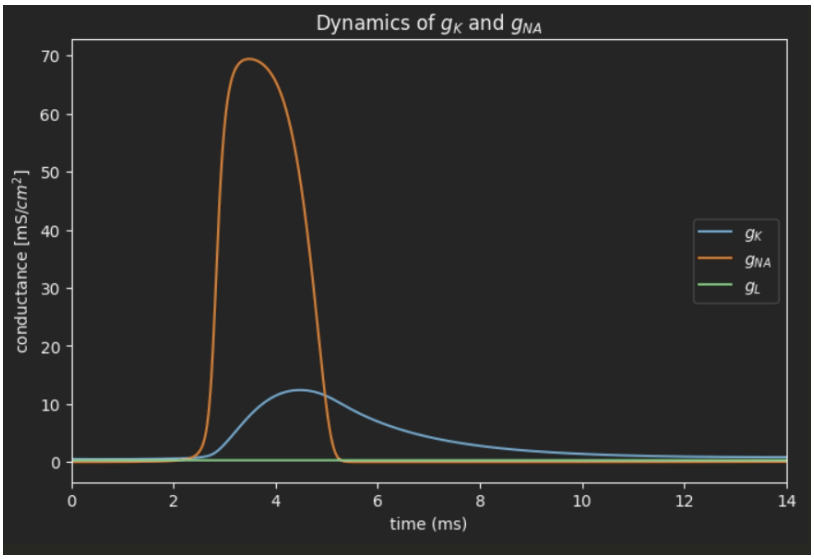
The diagram of steady values of gating variable as a function of voltage can be seen in the following :



Behavior of conductances

Using the formulas of conductance of sodium channel and conductance of potassium channel, we can visualize their behavior during a single spike using the following code :

```
gK = gK0 * n2**4
gNa = gNa0 * m2**3 * h
gL = gL0 * np.ones(t2.shape[0])

plt.plot(t2,gK, label="$g_K$")
plt.plot(t2,gNa, label="$g_{NA}$")
plt.plot(t2,gL, label="$g_{L}$")
plt.xlabel('time (ms)')
plt.xlim([0, 14])
plt.ylabel('conductance [mS/$cm^2$]')
plt.title('Dynamics of $g_K$ and $g_{NA}$')
plt.legend(loc = 'center right')
plt.rcParams['figure.figsize']=(8,5)
plt.show()
```



During a spike as we saw before, in the first step (depolarization) the membrane voltage starts to increase and we saw that the value of m will increase sharply with a small time constant, also in the first step and early times the value of h is not that small and the sodium channel won't get closed so its something noticeable. Also in this step, the potassium channels are close and will be opened gradually so n is small. Since that fact that $g_{Na} = m^3 h\ g_{Na_0}$ and $g_K = n^4\ g_{K_0}$, value of g_{Na} will sharply increase while g_K has small values.
In the following steps including re-polarization and hyper-polarization, since the fact that sodium channels will get closed and values of m will sharply decrease with power of 3 (m^3) and their values are too small hence the g_{Na} will also decrease consequently although the values of h will increase a bit.
Because the fact that values of n will rise with a greater time constant in comparison to m, the value of g_K will increase with more latency than g_{Na}. By decreasing n through time, g_K will also decrease.
Also we know that in Hodgkin and Huxley model the leakage conductance will be considered constant.