module1-ex1-12

April 9, 2023

1 Lab Session #1

1.1 Computational Neurophysiology [E010620A]

1.1.1 Dept of Electronics and Informatics (VUB) and Dept of Information Technology (UGent)

Jorne Laton, Lloyd Plumart, Talis Vertriest, Jeroen Van Schependom, Sarah Verhulst

Student names and IDs: César Zapata - 02213600, Academic Year: 2022-2023

1.1.2 General Introduction

In all the practical sessions of this course we will use python 3 and jupyter notebooks. Install Anaconda on your computer and open jupyter notebook by typing "jupyter notebook" or "jupyter lab" in the command line. Your browser will open a file explorer, from where you can navigate to the exercise.

The lab sessions consist of a jupyter notebook in which the different steps are described and explained, together with the tasks you are asked to complete.

You will form groups of two and submit one report per group. Reports should be formatted according to the guiding document. Make sure your answers stand out in the final submitted document!

Deadline: 2 weeks after lecture. Reports submitted after the deadline will not be graded.

1.1.3 Context and Goals

This lab session is focused on the Hodgkin-Huxley (HH) model, following and reproducing the theoretical chapter that can be found here: https://neuronaldynamics.epfl.ch/online/Ch2.S2.html. You will be asked to complete code scripts, make observations and explain the results of the different simulations, and contextualise the analyses with the HH model theoretical background.

2 Questions

2.1 Part 1: Hodgkin-Huxley model equations

2.1.1 Q1.1 HH equations

We are coding the HH equations by a single or by multiple functions, to reproduce the behaviour of a human pyramidal neuron when excited. This will be the function to be made to perform the

HH model is the following:

```
m, h, n, V, INa, IK, alpha_m, alpha_n, alpha_h, beta_m, beta_n, beta_h =
HH_model(T, I_input, dt)
```

The inputs are: 1. `T` -> Time window of simulation [ms] 2. `I_input` -> input current [µA] 3. `dt` -> the rate of update [ms]

The outputs are: 1. `V` -> the voltage of neuron [mV] 2. `m` -> activation variable for Na-current [u/cm^2] 3. `h` -> inactivation variable for Na-current [u/cm^2] 4. `n` -> activation variable for K-current [u/cm^2] 5. `t` -> the time axis of the simulation (useful for plotting) [ms] 6. `I_Na` -> Na current [µA/cm^2] 7. `I_K` -> K current [µA/cm^2] 8. `alpha_m, beta_m, alpha_n, beta_n, alpha_h, beta_h` -> gating parameters [1/ms]

You can use the functions that are already provided to retrieve some parameters. To complete the code, you can find the equations and parameter values in https://neuronaldynamics.epfl.ch/online/Ch2.S2.html.

Q1.1a Implement the update equations of the gating variables m, n and h as described in Table 2.1 of the online version of the book and reproduce Figure 2.3.

Q1.1b Make the plots and describe in your own words what you have plotted.

Q1.1c Unfortunately, when simulating the HH model with these parameters, you will not get a K-current. Please adjust (u-25) by (u+25) in the update equations for gating variable n.

- Import these modules
- Fill in answers here

2.1.2 Q1.2 Simulate the response to an impulse current

To try out whether the designed functions work, design a step function (A1.2a) that can be used to model the current input (A1.2b). Consider the following design parameters:

1. `I_input function` -> step function
2. `T` -> time of simulation: 100 ms
3. `dt` -> update time: 0.01 ms
4. Current impulse `I_input`: 20 µA between 1 and 2 ms.

- Fill in answers here

2.1.3 Q1.3. Plot V(t)

Plot the first 20 ms of $V(t)$.

Describe the dynamics of the neural voltage V. Does it make sense?

- Fill in answers here

2.1.4 Q1.4 Plot the model parameters

Plot the model parameters n, m and h in function of t and again limit the plots to the first 20 ms. Describe the dynamics of the model parameters m, n, and h. Does it make sense? Describe how the gates swing open and closed during the dynamics of a spike. * Fill in answers here

2.1.5 Q1.5. Plot I_Na and I_K

Plot I_Na and I_K in function of t (again only the first 20 ms).

Describe the dynamics of the currents. Does it make sense? Describe the currents flows during a spike.

- Fill in answers here

2.1.6 Q1.6. Plot the conductances g_Na and g_K

Plot the conductances g_Na, g_K in function of t (again only the first 20 ms).

How are the conductances evolving during a spike?

- Fill in answers here

2.2 Part 2: Package BRIAN

In the second part of the practical, you are going to use the Brian library to simulate the dynamics of a squid neuron when excited. To learn more about this module, read this paper: https://pubmed.ncbi.nlm.nih.gov/19115011/ . Before starting, the Brian module must be installed, together with the neurodynex3 module. Open the anaconda prompt, and install the brian2 package:

```
conda install -c conda-forge brian2
```

and then the neurodynex3 package:

```
pip install neurodynex3
```

2.2.1 Note

If you are working on Linux or one of the newest mac OS systems, you might check whether your pip command is actually pip3.

After installing, the packages are ready to use! If you need more information about the modules, you can find it here https://briansimulator.org/.

- Import these modules

2.2.2 Q2.1 Step current response

We study the response of a Hodgkin-Huxley squid neuron to different input currents.

Have a look at the documentation of the functions `HH.simulate_HH_neuron()` and `HH.plot_data()` and the module `neurodynex3.tools.input_factory`.

By using the mentioned functions, code the following steps: 1. Step function for the input current 2. Run the HH simulation (for 300ms) 3. Plot the results of this simulation

Vary the amplitude of the input current between 0.1 and 50 µA between 50 and 250ms. Describe the different dynamics of the spiking neuron.

What is the lowest step current amplitude to generate a spike or to generate repetitive firing? Discuss the difference between the two regimes.

- Fill in answer here

2.2.3 Q2.2 Slow and fast ramp current

The minimal current to elicit a spike does not just depend on the amplitude I or on the total charge Q of the current, but on the "shape" of the current. Let's investigate why.

Q2.2.1 Slow ramp current Inject a slow ramp current into a HH neuron. The current is 0 µA at t = 0 and starts at 5 ms, linearly increasing to an amplitude of 14.0 µA at t = t_ramp_end. At t > t_ramp_end, the current is set back to 0 µA. A slow ramp duration could be between 30-100 ms.

Experiment with different t_ramp_end values to discover the maximal duration of a ramp, such that the neuron does not spike. Make sure you simulate for at least 20ms after t_ramp_end.

Q2.2.1a Use the function `HH.plot_data()` to visualize the dynamics of the system.

Q2.2.1b What is the membrane voltage at the time when the current injection stops (t=slow_ramp_t_end)?

- Fill in answers here

Q2.2.2 Fast ramp current Q2.2.2a Do the same as before but this time for a fast ramp current. Start the linearly increasing input current again at t = 5 ms. The amplitude at t = t_ramp_end is 5.0 µA. Start with a duration of 5 ms and then increase the ramp duration it until no spike occurs. Use the function `HH.plot_data()` to visualize the dynamics of the system.

Q2.2.2b What is the membrane voltage at the time when the current injection stops (t=slow_ramp_t_end)?

- Fill in answers here

Q2.2.3. Differences Discuss the differences between the two situations. Why are the two "threshold" voltages different? Link your observation to the gating variables m, n and h.

Hint: have a look at Chapter 2, Figure 2.3.

- Fill in answers here

2.2.4 Q2.3 Rebound Spike

A HH neuron can spike not only if it receives a sufficiently strong depolarizing input current but also after a hyperpolarizing current. Such a spike is called a rebound spike.

Inject a hyperpolarizing step current I_input = -1 µA for a duration of 25 ms into the HH neuron. Simulate the neuron for 50 ms and plot the voltage trace and the gating variables. Repeat the simulation with I_input = -5 µA. What is happening here? To which gating variable do you attribute this rebound spike?

It may be difficult to to see which gating parameter is responsible for depolarisation (and a possible consequent rebound spike) after a negative current injection. Therefore, also plot the real ratio of n and m, together with the effect of h and use this plot to answer the above question.

- Fill in answers here

3 Answers

3.1 Part 1: Hodgkin-Huxley model equations

3.1.1 Imports

[1]:
```
# Import and add all the libraries you need throughout the code

import math
import numpy as np
import matplotlib.pyplot as plt

# Make your graphs colorblind-friendly
plt.style.use('tableau-colorblind10')
```

3.1.2 A1.1: HH Equations

- Go back to Q1.1

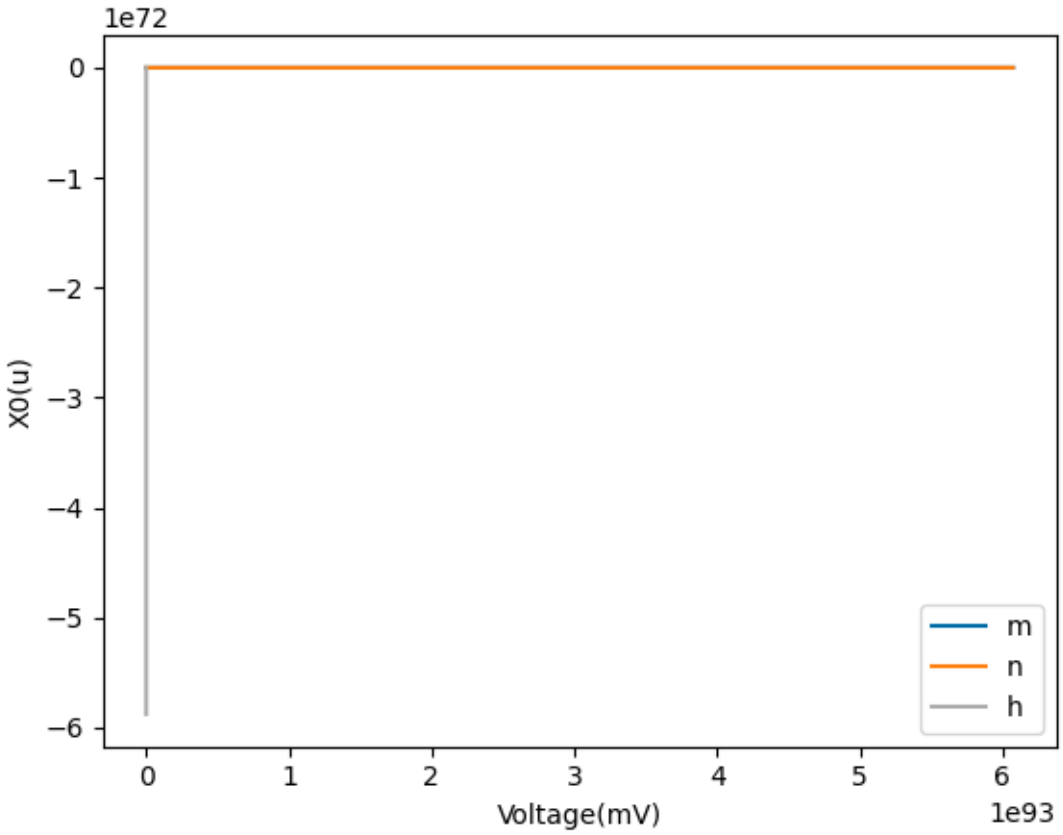
[16]:
```
# A1.1a Enter your code at the bottom of this cell

# Use the following variable names:

#T = 100 #ms. Given
#I_input
#dt # given
#alpha_m
#beta_m
#alpha_n
#beta_n
#alpha_h
#beta_h
#m
#h
#n
#V
#t
#I_Na
#I_K
#I_L

# Hints:
# Reversal potentials have unit mV
# Constant conductances have unit mS/cm2

# Hint: complete the following functions:
```

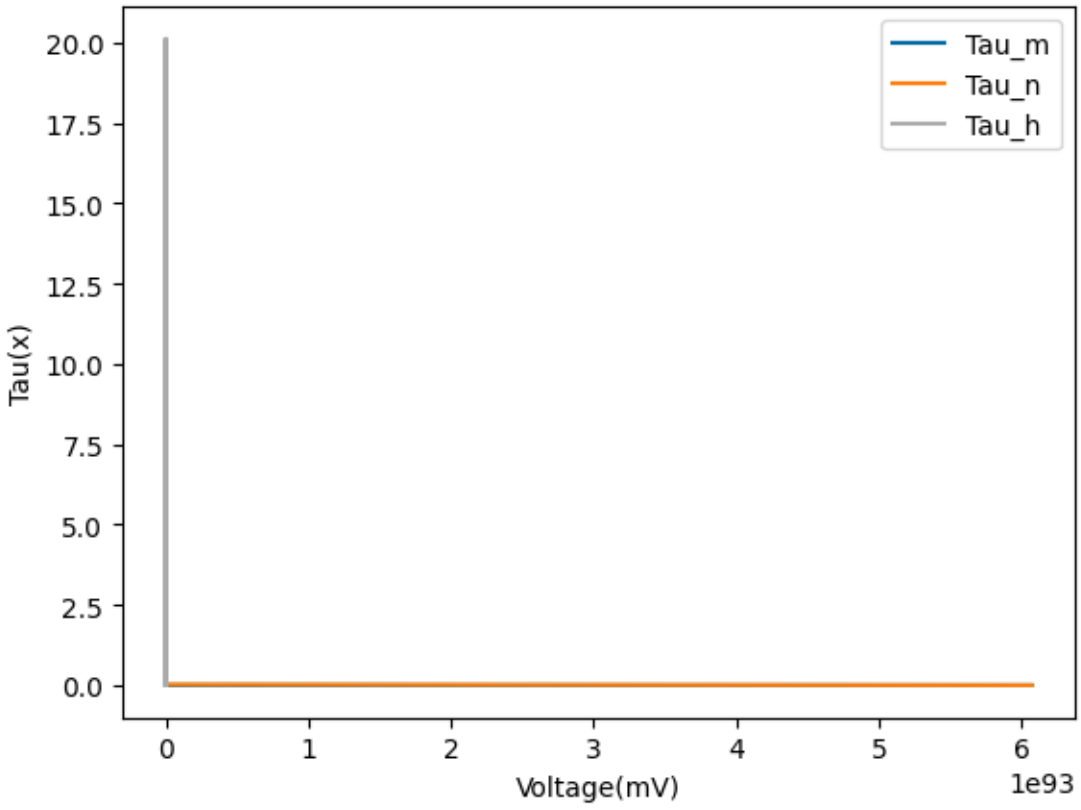
```
def gating_variable_n(u):
    alpha_n = (0.02 * (u-25)) / (1 - math.e ** (-(u-25) / 9) )
    beta_n = (-0.002 * (u-25)) / (1 - math.e ** ( (u-25) / 9) )

    return alpha_n, beta_n


def gating_variable_m(u):
    alpha_m = (0.182 * (u+35)) / (1 - math.e ** (-(u+35) / 9) )
    beta_m = (-0.124 * (u+35)) / (1 - math.e ** ( (u+35) / 9) )

    return alpha_m, beta_m


def gating_variable_h(u):
    alpha_h = 0.25 * (math.e ** (-(u+90) / 12) )
    beta_h = 0.25 * (math.e ** ((u+62) / 6)) / ((math.e ** ((u+90) / 12)) )

    return alpha_h, beta_h



def HH_model(T, I_input, dt):

    u_rest = -65 #(mV) -> membrane resting potential
    t = np.arange(0, T, dt)  # time axis of the simulation

    # gating variables
    alpha_m, beta_m = gating_variable_m(u_rest)
    alpha_n, beta_n = gating_variable_n(u_rest)
    alpha_h, beta_h = gating_variable_h(u_rest)

    # Table 2.1 in the document. Parameters for conductance and reverse␣
 ↪potentials
    g_Na = 40 #(mS/cm2)
    E_Na = 55 #(mV)
    g_K = 35 #(mS/cm2)
    E_K = -77 #(mV)
    g_L = 0.3 #(mS/cm2)
    E_L = -65 #(mV)
    C_m = 1 #(uF/cm2)


    # Defining initial conditions
    V = np.zeros(len(t)) # membrane potential vector (mV)
    m = np.zeros(len(t))
    h = np.zeros(len(t))
```

```
    n = np.zeros(len(t))
    Tau_m = np.zeros(len(t))
    Tau_n = np.zeros(len(t))
    Tau_h =np.zeros(len(t))

    # initial values
    V[0] = u_rest
    m[0] = (alpha_m) / (alpha_m + beta_m)
    n[0] = (alpha_n) / (alpha_n + beta_n)
    h[0] = (alpha_h) / (alpha_h + beta_h)
    Tau_m[0] = 1 / (alpha_m + beta_m)
    Tau_n[0] = 1 / (alpha_n + beta_n)
    Tau_h[0] = 1 / (alpha_h + beta_h)


    def Ic(v, m, n, h):
        # return  g_L*(E_L - v) + g_K*(n**4)*(E_K - v) + g_Na*(m**3)*h*(E_Na -␣
↪v)
        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)
        suma = I_L + I_k + I_Na
        return  suma


    # Simulating
    for i in range(1, len(t)):

        # updating gating variables
        alpha_m, beta_m = gating_variable_m(V[i-1])
        alpha_n, beta_n = gating_variable_n(V[i-1])
        alpha_h, beta_h = gating_variable_h(V[i-1])

        Tau_m[i] = 1 / (alpha_m + beta_m)
        Tau_n[i] = 1 / (alpha_n + beta_n)
        Tau_h[i] = 1 / (alpha_h + beta_h)

        V[i] = (dt * (-Ic(V[i-1], m[i-1], n[i-1], h[i-1]) / C_m)) + V[i-1]
        m[i] = (alpha_m * (1 - m[i-1]) - beta_m * m[i-1])
        h[i] = (alpha_h * (1 - h[i-1]) - beta_h * h[i-1])
        n[i] = (alpha_n * (1 - n[i-1]) - beta_n * n[i-1])



    # Plotting
    plt.plot(V, m, label="m")
    plt.plot(V, n, label="n")
```

```
    plt.plot(V, h, label="h")
    plt.legend()
    plt.xlabel('Voltage(mV)')
    plt.ylabel('X0(u)')
    plt.show()


    plt.plot(V, Tau_m, label="Tau_m")
    plt.plot(V, Tau_n, label="Tau_n")
    plt.plot(V, Tau_h, label="Tau_h")
    plt.legend()
    plt.xlabel('Voltage(mV)')
    plt.ylabel('Tau(x)')
    plt.show()

    return m, h, n, V, t, alpha_m, alpha_n, alpha_h, beta_m, beta_n, beta_h,␣
 ↪Tau_n, Tau_m, Tau_h


###################################
##   A1.1a solutions functions   ##
###################################

# m, h, n, V, t, I_Na, I_K, I_L, alpha_m, alpha_n, alpha_h, beta_m, beta_n,␣
 ↪beta_h, tau_n, tau_m, tau_h = HH_model(100, 0.02, 0.001)
HH_model(100, 0.02, 0.01)
```

```
C:\Users\cesar\AppData\Local\Temp\ipykernel_15032\548214621.py:39:
RuntimeWarning: overflow encountered in double_scalars
  beta_m = (-0.124 * (u+35)) / (1 - math.e ** ( (u+35) / 9) )
C:\Users\cesar\AppData\Local\Temp\ipykernel_15032\548214621.py:32:
RuntimeWarning: overflow encountered in double_scalars
  beta_n = (-0.002 * (u-25)) / (1 - math.e ** ( (u-25) / 9) )
C:\Users\cesar\AppData\Local\Temp\ipykernel_15032\548214621.py:46:
RuntimeWarning: overflow encountered in double_scalars
  beta_h = 0.25 * (math.e ** ((u+62) / 6)) / ((math.e ** ((u+90) / 12)) )
C:\Users\cesar\AppData\Local\Temp\ipykernel_15032\548214621.py:46:
RuntimeWarning: invalid value encountered in double_scalars
  beta_h = 0.25 * (math.e ** ((u+62) / 6)) / ((math.e ** ((u+90) / 12)) )
C:\Users\cesar\AppData\Local\Temp\ipykernel_15032\548214621.py:93:
RuntimeWarning: overflow encountered in double_scalars
  I_Na = g_Na*(m**3)*h*(v-E_Na)
C:\Users\cesar\AppData\Local\Temp\ipykernel_15032\548214621.py:94:
RuntimeWarning: overflow encountered in double_scalars
  I_k = g_K*(n**4)*(v - E_K)
```

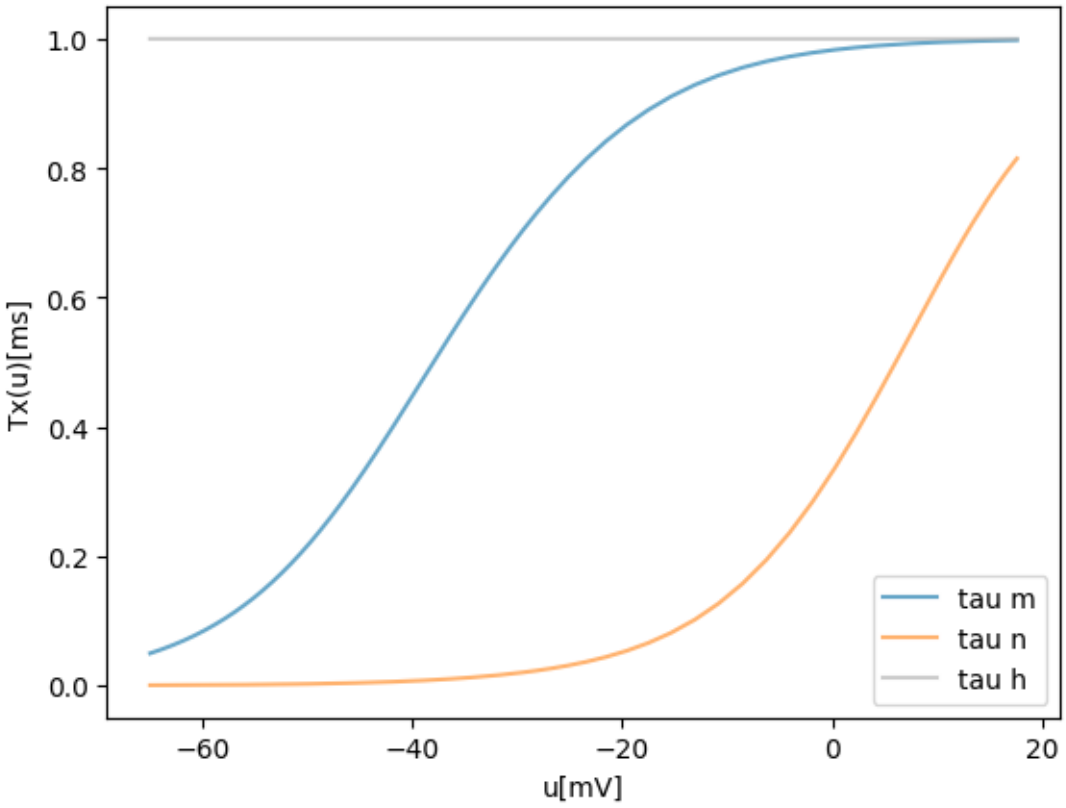

1e72
0
−1
−2
−3
−4
−5
−6
X0(u)
0
1
2
3
4
5
6
1e93
Voltage(mV)
m
n
h

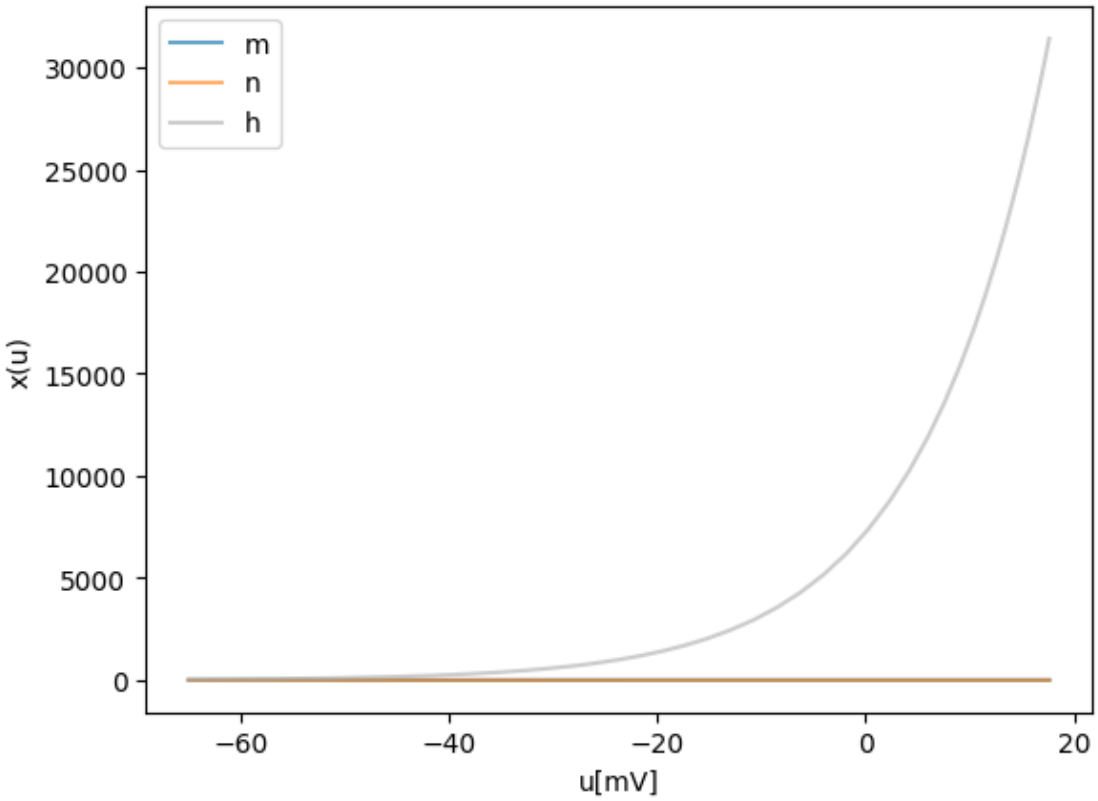


[16]: (array([0.04975503, 0.   , 0.20192478, …,   nan,   nan,
                nan]),
       array([6.22459331e-01, -1.73472348e-18, 3.11381658e-02, …,
                      nan,   nan,   nan]),
       array([4.53793276e-04, 0.00000000e+00, 8.16935124e-05, …,
                     nan,   nan,   nan]),
       array([-65.   , -65.00368013, -65.00369117, …,   nan,
                    nan,   nan]),
       array([0.000e+00, 1.000e-02, 2.000e-02, …, 9.997e+01, 9.998e+01,
              9.999e+01]),
       nan,
       nan,
       nan,
       nan,
       nan,
       nan,
       array([5.55278237, 5.55278237, 5.55255646, …,   nan,   nan,
                nan]),
       array([0.24632955, 0.24632955, 0.24630807, …,   nan,   nan,
                nan]),
       array([19.99636906, 19.99636906, 19.99486629, …,   nan,

```
                    nan,         nan]))
```

[85]:
```
# A1.1b Plot your graph below

# Hint: the first graph plots rate constants (n, m and h in function of␣
 ↪membrane voltage)
# Hint: the second graph plots the voltage dependent time constants (tau_n,␣
 ↪tau_m and tau_h in function of membrane voltage)


plt.figure(1)
plt.plot(V, m, alpha=0.6, label="tau m")
plt.plot(V, n, alpha=0.6, label="tau n")
plt.plot(V, h, alpha=0.6, label="tau h")
plt.xlabel("u[mV]")
plt.ylabel("Tx(u)[ms]")
plt.legend()
plt.show()


plt.figure(2)
plt.plot(V[:-1], Tau_n, alpha=0.6, label="n")
plt.plot(V[:-1], Tau_m, alpha=0.6, label="m")
plt.plot(V[:-1], Tau_h, alpha=0.6, label="h")
plt.xlabel("u[mV]")
plt.ylabel("x(u)")
plt.legend()
plt.show()

# What is your conclusion?

################################
##   A1.1b plots   ##
################################


#########################
##   A1.1b conclusion   ##
#########################

# Answer in green box below
```


1.0
0.8
0.6
0.4
0.2
0.0
Tx(u)[ms]
−60
−40
−20
0
20
u[mV]
tau m
tau n
tau h



A1.1b conclusion

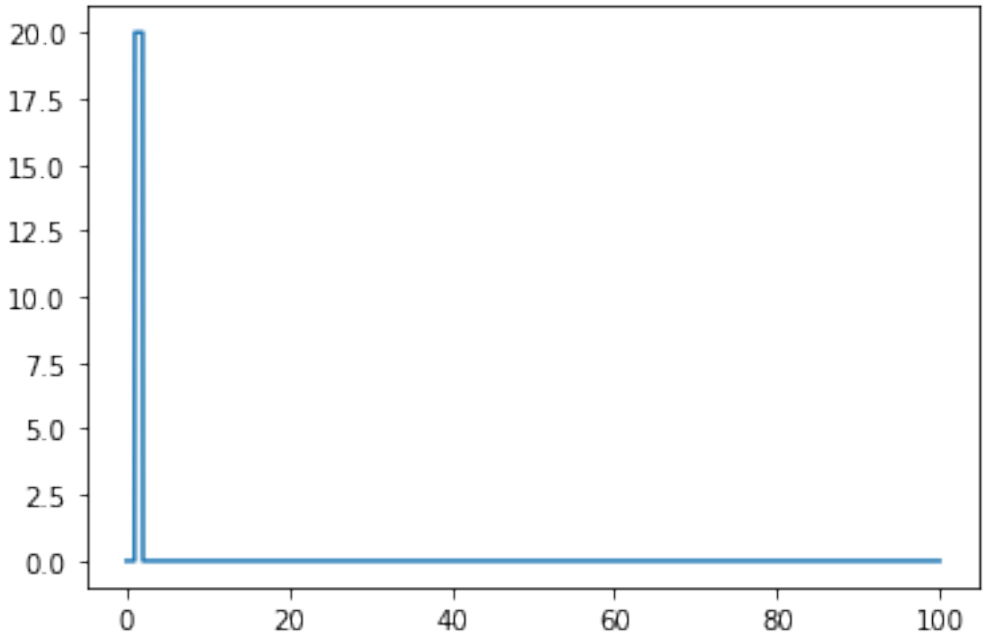
Your answer here

[4]:
```
# A1.1c Enter the updated HH model here

########################################
##   A1.1c updated HH model solution   ##
########################################
```

3.1.3 A1.2: Response simulation

- Go back to Q1.2

[14]:
```
# A1.2a Set up and plot your input current

########################################
##   A1.2a solutions input current   ##
########################################
t = np.arange(0, 100, 0.01)
I_input = [0 if x < 1 or x > 2 else 20 for x in t]
```



[6]:
```
# A1.2b Insert the input current into your HH_model function

########################################
##   A1.2b insert input in HH_model   ##
########################################
```

3.1.4 A1.3: Plot V(t)

- Go back to Q1.3

[7]:
```
# A1.3a Enter your answer below

############################
##   A1.3a solution plot  ##
############################
```

[8]:
```
# A1.3b Enter your interpretation and conclusion below

########################
##   A1.3b conclusion   ##
########################

# Answer in green box below
```

A1.3b conclusion

Your answer here

3.1.5 A1.4: Plot the model parameters

- Go back to Q1.4

[9]:
```
# A1.4a Plot your graph below

############################
##   A1.4a solution plot   ##
############################
```

[10]:
```
# A1.4b Enter your interpretation and conclusion answer below

########################
##   A1.4b conclusion   ##
########################

# Answer in green box below
```

A1.4b conclusion

Your answer here

3.1.6 A1.5: Plot I_Na and I_K

- Go back to Q1.5

[11]:
```
# A1.5a Plot your graph here

############################
##   A1.5a plot   ##
############################
```

[12]:
```
# A1.5b Enter your interpretation and conclusion

########################
##   A1.5b conclusion   ##
########################

# Answer in green box below
```

A1.5b conclusion

Your answer here

3.1.7 A1.6: Plot the conductances g_Na and g_K

- Go back to Q1.6

[13]:
```
# A1.6a Enter your code and plot
```

```
############################
##   A1.6a solution plot  ##
############################
```

[14]:
```
# A1.6b Enter your interpretation and conclusion

#########################
##   A1.6b conclusion   ##
#########################

# Answer in green box below
```

A1.6b conclusion

Your answer here

3.2 Part 2: Package BRIAN

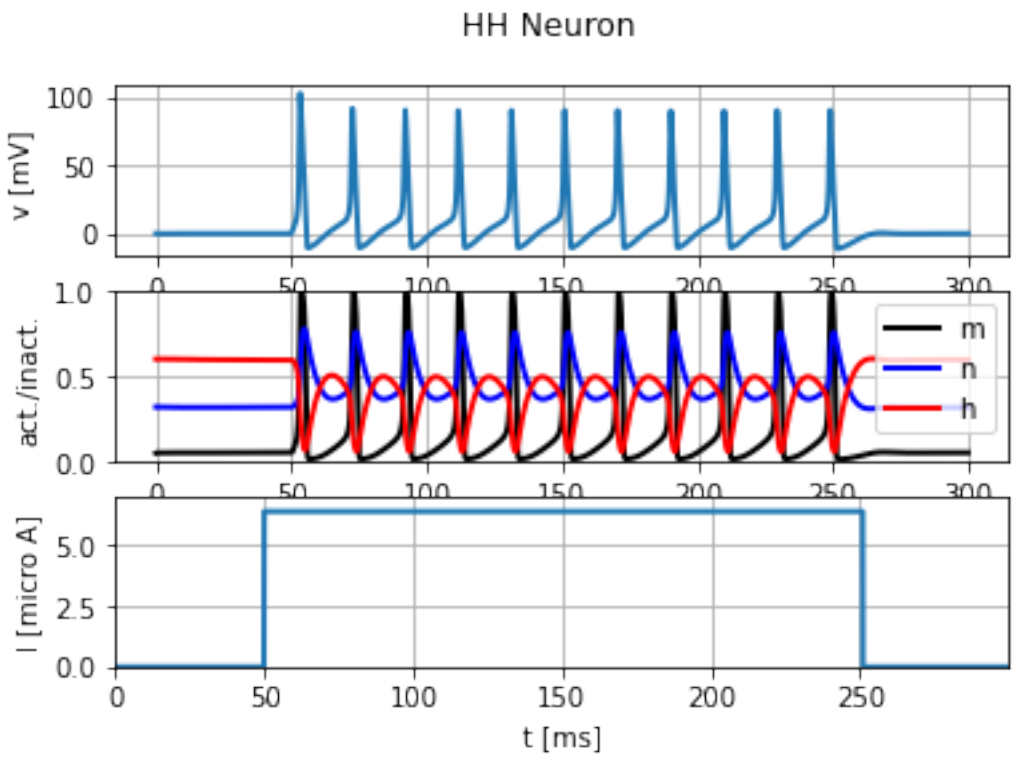
3.2.1 Import

[2]:
```
# The new libraries we need to add

%matplotlib inline
import brian2 as b2
import matplotlib.pyplot as plt
import numpy as np
from neurodynex3.hodgkin_huxley import HH
from neurodynex3.tools import input_factory
```

3.2.2 A2.1 Step current response

- Go back to Q2.1

[28]:
```
# A2.1a Enter your answer below

# Hint: The unit of the I_input current in the neurodynex3.hodgkin_huxley␣
 ↪module is µA (coded b2.uA)

########################
##   Q2.1. solution   ##
########################

I_input = input_factory.get_step_current(50, 250, b2.ms, 6.4 * b2.uA, 50 * b2.
 ↪uA)
state_monitor = HH.simulate_HH_neuron(I_input, 300 * b2.ms)
HH.plot_data(state_monitor, title="HH Neuron")
```



[17]:
```
# Enter your conclusion and interpretation

############################
##   Q2.1. conclusion   ##
############################

'''
The neuron will start showing a spike from a current with amplitude of 2.4uA,␣
 ↪which will then
convert to repetitive firing at 6.4uA and continue with this behavior up until␣
 ↪50uA, increasing
its firing frequency directly proportional to the amplitude of the step current.
'''


# Answer in green box below
```

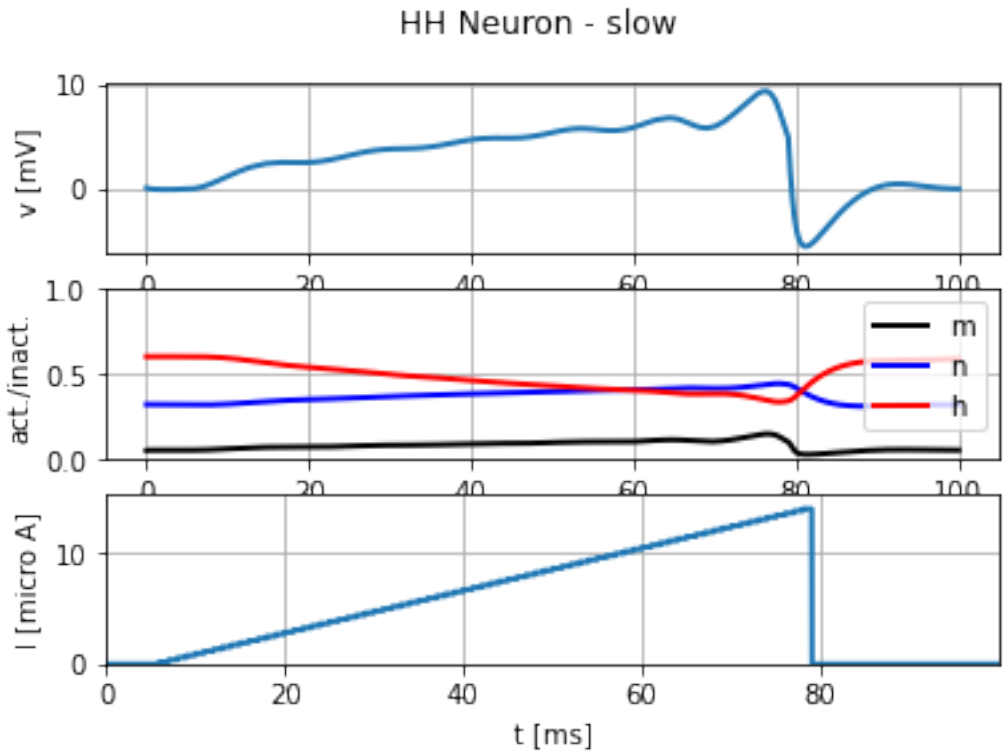
A2.1 conclusion

The lowest step current amplitude to generate a spike is: 2.4uA
 The lowest step current amplitude to generate repetitive firing is: 6.4uA

3.2.3 A2.2.1 Slow ramp current

- Go back to Q2.2.1

[44]:
```
# Enter your code below

#########################
##   Q2.2.1a solution   ##
#########################

b2.defaultclock.dt = 0.02*b2.ms
slow_ramp_t_end = 78  # no spike. make it shorter
slow_ramp_current = input_factory.get_ramp_current(5, slow_ramp_t_end, b2.ms, 0.
 ↪*b2.uA, 14.0*b2.uA)
state_monitor = HH.simulate_HH_neuron(slow_ramp_current, 100 * b2.ms)
idx_t_end = int(round(slow_ramp_t_end*b2.ms / b2.defaultclock.dt))
voltage_slow = state_monitor.vm[0,idx_t_end]
HH.plot_data(state_monitor, title="HH Neuron - slow")
print(f"voltage_slow={voltage_slow}")
```




```
voltage_slow=0.007214642935639053
```

[29]:
```
############################
##   Q2.2.1a conclusion   ##
############################

'''
The slow rising of the current prevents the neuron from reaching the threshold␣
 ↪to get a spike.
'''

# Answer in green box below
```

[29]:
```
'\nThe slow rising of the current prevents the neuron from reaching the
threshold to get a spike.\n'
```

A2.2.1a conclusion

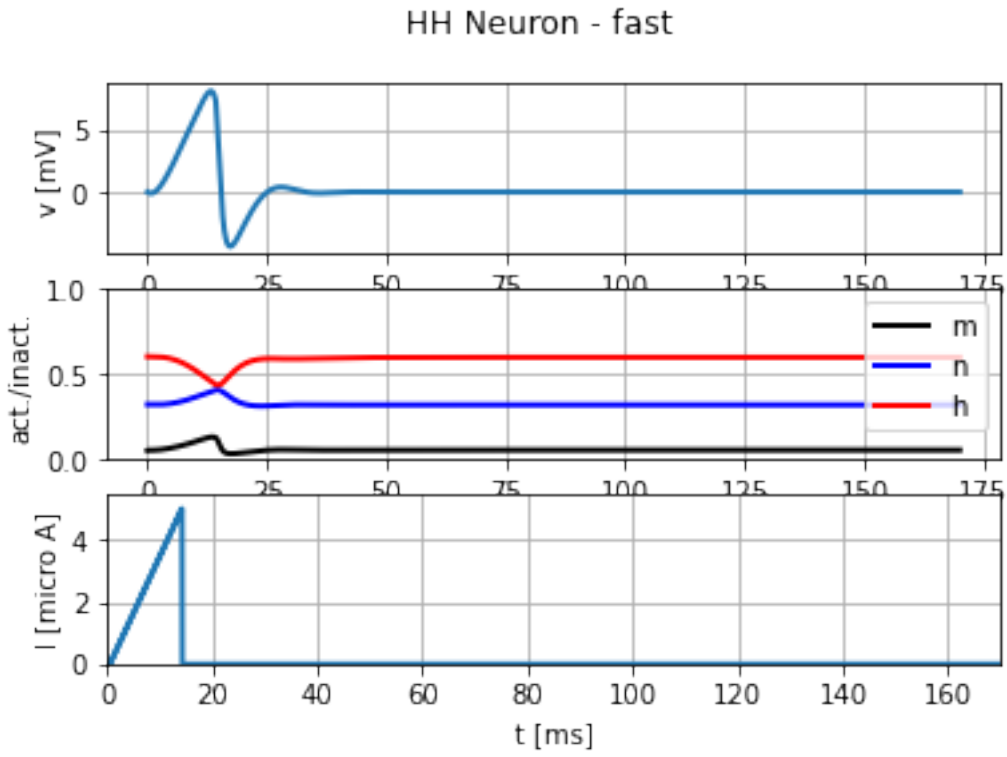
The membrane voltage at the time when the current injection stops is 7mV. At this time, the neuron
 shows a minimal rise in its voltage, but does not reach the threshold to be classified as a spike.

3.2.4 A2.2.2 Fast ramp current

- Go back to Q2.2.2

[64]:
```
# Enter your code below

# Hint: change the unit time to 0.1ms to get a smooth ramp current.

#########################
##   Q2.2.2a solution   ##
#########################

b2.defaultclock.dt = 0.02*b2.ms
fast_ramp_t_end = 142  # no spike. make it longer  -> voltage goes more than 10␣
 ↪times higher at 141
fast_ramp_current = input_factory.get_ramp_current(5, fast_ramp_t_end, 0.1*b2.
 ↪ms, 0.*b2.uA, 5*b2.uA)
state_monitor = HH.simulate_HH_neuron(fast_ramp_current, 170 * b2.ms)
idx_t_end = int(round(fast_ramp_t_end*0.1*b2.ms / b2.defaultclock.dt))
voltage_fast = state_monitor.vm[0,idx_t_end]
HH.plot_data(state_monitor, title="HH Neuron - fast")
print(f"voltage_fast={voltage_fast}")

###########################
##   Q2.2.2a conclusion   ##
###########################
```

```
# Answer in green box below
```




```
voltage_fast=0.007443206417119181
```

A2.2.2a conclusion

Your answer here

[23]:
```
#insert your code here

#########################
##   Q2.2.2b solution  ##
#########################



############################
##   Q2.2.2b conclusion   ##
############################


# Answer in green box below
```

A2.2.2b conclusion

The membrane voltage at the time when the current injection stops is 7mV.

3.2.5 A2.2.3 Differences

- Go back to Q2.2.3

[24]:
```
# Enter your answer below

#########################
##   Q2.2.3 solution   ##
#########################

# Answer in green box below
```

A2.2.3 conclusion

The final voltages for the slow ramp and the fast ramp are not significantly different,
 which means that the neuron is not responding to the velocity of the input, only to its value.

3.2.6 A2.3 Rebound spike

- Go back to Q2.3

[25]:
```
# Enter your answer below

######################
##   Q2.3 solution   ##
######################
```

A2.3 conclusion

Your answer here

[ ]:
```

```