Machine Learning CMPU 395

Assignment 1: Hodgkin-Huxley Modeling

1 Task

The task is to simulate the activity in an actively firing neuron using the Hodgkin-Huxley equations. The necessary equations and suitable parameters are listed below. Minimum requirement is that you produce a voltage trace of the neuron membrane potential when it is firing spikes. For highest grade you should also measure the output frequency for different input currents and present this as a diagram.

2 Software Tools

For this assignment you will use Python extended with SciPy. Python is a modern all-purpose programming language which has gained a lot of interst in the science community. A large set of utility packages make it easy to use Python for various tasks. Here, we are mainly interested in the packages for solving differential equations and for plotting graphs. These are both part of a set of packages known as Scientific Python (SciPy).

Sets of ordinary differential equations (ODEs) can be integrated (solved) using a function called `odeint`. This function is part of a package called `scipy.integrate`. Plotting can be done using the `plot` function from a package called `pylab`.

3 Template code

Below is a sample program which illustrates how `odeint` and `plot` can be used. This template code solves this set of differential equations:

$$\frac{du}{dt} = v \qquad \frac{dv}{dt} = -u$$

Before you run the program, try to envision what these equations mean. The first equation tells us that u will increase when v is positive and decrease when v is negative. The second equation tells us that v will move in the opposite direction compared to u due to the minus sign.

As you can see, it can be quite challenging to grasp what will happen by just staring at the equations. With a computer it becomes easy to simulate what will happen to the *state variables* (u and v in this case), as time proceeds. The process of simulating a set of differential equations is also referred to as *integration*.

The function `odeint` takes three arguments: a function specifying which differential equations to integrate, a list of initial values (for u and v), and an

array of time points when the state variables should be calculated. We use the function `arange` from the `numpy` package to generate an array of 100 equidistally placed time points.

The last two lines of the template program show how the result of the simulation can be plotted. The expression `y[:,0]` refers to a *slice* from the two-dimensional array `y`. Colon here means "all rows" while 0 means only column zero. This means that we are plotting u. Note that indices always start from zero in Python.

```
import numpy                          # Numerical library
import pylab                          # Plotting routines
from scipy.integrate import odeint    # ODE integration


# Integrate this system of differential equations
# du/dt = v
# dv/dt = -u

# Our choice of representation: array x contains [u, v]

def f(x, t):
    # Compute right hand side of ODE system
    return [x[1], -x[0]]


# Time runs from 0 to 10 in 0.1 increments
t = numpy.arange(0.0, 10, 0.1)

# Starting from u=0.0 and v=1.0, integrate over time
y = odeint(f, [0.0, 1.0], t)

# y is now a 2-dimentional array,
# column zero contains u values for all
# times in t, column one contains v

# Plot the u values as a function of time
pylab.plot(t, y[:,0])
pylab.show()
```

Copy this program into a file `odesolver.py` and run it from the command line by typing `python odesolver.py`. You should see a graphical window showing how u evolves over time.

Note that you can save the diagram in a file by clicking on the rightmost button below the graph. This will be handy when you want to include your graphs in your lab reports.

Make a small change of the equation system. You can e.g. use these equa-

tions instead:

$$\frac{du}{dt} = v - 0.1u \qquad \frac{dv}{dt} = -u - 0.1v$$

Change the plotting to show v as a function of u. Save this plot and include it in your lab report.

4 Equations

Now you should be ready to take on the challange of integrating the famous Hodgin-Huxley equations. Here we have four state variables which interact: the membrane voltage (V), the sodium channel activation (m), the sodium channel inactivation (h), and the potassium channel activation (n).

Your program will be easier to both write and read if you make use of variables for the parameters involved, and helper functions for the equations below which are not differential equations.

4.1 Membrane Voltage Equations

The voltage over the cell membrane v changes due to ions passing through channels in the membrane. We will only take three such currents into account: the sodium current (I_{Na}), the potassium current (I_K), and an unspecific leak current (I_L).

$$C \cdot \frac{dv}{dt} = I_L + I_{Na} + I_K$$

$$I_L = G_L \cdot (E_L - v)$$

$$I_{Na} = m^3 h G_{Na} \cdot (E_{Na} - v)$$

$$I_K = n^4 G_K \cdot (E_K - v)$$

The three state variables m, h, and n denote the activation, and inactivation of sodium channels, and the activation of potassium channels, respectively.

Parameter	Value	Description
C	$0.3 \cdot 10^{-9}\,\mathrm{F}$	Membrane capacitance
G_L	$0.03 \cdot 10^{-6}\,\Omega^{-1}$	Leak conductance
E_L	$-0.070\,\mathrm{V}$	Leak equilibrium potential
G_{Na}	$10 \cdot 10^{-6}\,\Omega^{-1}$	Sodium conductance
E_{Na}	$0.050\,\mathrm{V}$	Sodium equilibrium potential
G_K	$2 \cdot 10^{-6}\,\Omega^{-1}$	Potassium conductance
E_K	$-0.090\,\mathrm{V}$	Potassium equilibrium potassium

4.2 Channel Activation Equations

$$\frac{dm}{dt} = \alpha_m \cdot (1 - m) - \beta_m \cdot m$$

$$\frac{dh}{dt} = \alpha_h \cdot (1 - h) - \beta_h \cdot h$$

$$\frac{dn}{dt} = \alpha_n \cdot (1 - n) - \beta_n \cdot n$$

$$\alpha_m = \frac{A(v - B)}{1 - e^{-\frac{v-B}{C}}} \qquad \beta_m = \frac{-A(v - B)}{1 - e^{\frac{v-B}{C}}}$$

$$\alpha_h = \frac{-A(v - B)}{1 - e^{\frac{v-B}{C}}} \qquad \beta_h = \frac{A}{1 + e^{-\frac{v-B}{C}}}$$

$$\alpha_n = \frac{A(v - B)}{1 - e^{-\frac{v-B}{C}}} \qquad \beta_n = \frac{-A(v - B)}{1 - e^{\frac{v-B}{C}}}$$

Function	A	B	C
α_m	$0.2 \cdot 10^6$	-0.040	0.001
β_m	$0.06 \cdot 10^6$	-0.049	0.020
α_h	$0.08 \cdot 10^6$	-0.040	0.001
β_h	$0.4 \cdot 10^3$	-0.036	0.002
α_n	$0.02 \cdot 10^6$	-0.031	0.0008
β_n	$0.005 \cdot 10^6$	-0.028	0.0004

5 Running and Reporting

Assemble a program which simulates a neuron with the given equations and parameters. In order to activate the neuron to see some interesting behavior you need to "inject" a small electrical current into the cell. Do this by adding yet another current, I_{Stim} with a constant value. Note that all parameters are in *SI* units, so currents will be in Amperes (!). If you inject 1 A into a neuron it will probably evaporate before you even see it. Around 1 nA may be more appropriate.

This assignment should be reported in a short (two pages are enough) lab report. Include one diagram from your modified u, v equations, and one diagram showing the membrane voltage of your neuron when you inject current.

For the highest grade you should also include a diagram showing how the firing frequency of the neuron changes for different injected currents.