Lab 2: Conductance-Based Models

This lab asks you to explore a Hodgkin–Huxley (HH) style action potential model. The intended learning outcomes are to (1) impart a deeper understanding of the HH model and the phases of the action potential (2) check that you can solve ODEs numerically using built in solvers with higher-order accuracy like Runge-Kutta, and (3) check that you can create or modify code to apply mathematical models to answer questions about neurophysiology. The exam may ask you questions about the mathematics or numerical implementation of HH, or fact-based questions about the role of the various ionic currents in the phases of the action potential.

In the lab 2 directory on github you will find the file `hhhelper.py`. This provides an implementation of the Hodgkin-Huxley model using the parameters from Chapter 2 of *Neuronal Dynamics* (see notes at the appendix at the end of this document). The `HH_model` class encapsulates the model parameters:

- `mymodel = HH_model()` creates a new instance. You can optionally pass keyword arguments to override parameter defaults, e.g. `HH_model(K_a=0.9)` speeds up the activation of the voltage-gated potassium channel.
- `mymodel.currents(v,m,n,h)` returns leak, voltage-gated potassium, and voltage-gated sodium currents (in that order), for a given voltage and gating state (m, n, h).
- `mymodel.ode(v,m,n,h,I)` calculates the time derivatives of the membrane voltage v, and the potassium opening, sodium opening, and sodium inactivation (m, n, h) gates, and applied inward current I (nanoamps).

Feel free to re-implement this logic as you'd like, and complete this coursework in any language you'd prefer. Programming exercises:

1. Use `scipy.integrate.solve_ivp()` function (or similar) with the `RK45` integrator to simulate a point-neuron's response to an inward applied current of $I = 0.7$ nA for 200 ms. (You may need to wrap the `ode(v,m,n,h,I)` in a function that provides the input current to interface with `solve_ivp()`.)

2. Use `mymodel.currents(v,m,n,h)` to compute and plot the current contributions of voltage-gated sodium, voltage-gated potassium, and leak channels and plot these as a separate traces. *You should see relatively little contribution from the potassium current.*

3. This hints that our model could make spikes without using the potassium current. Confirm that, indeed, there is little influence from this current by removing it by setting the maximal conductance for potassium to zero with `HH_model(gKmax=0)`. *Without potassium, what mechanism is stopping the spike from rising, and what causes the neuron to return to rest after a spike?*

4. Use `HH_model(K_a=0.9)` to make the voltage-gated potassium channel open faster. How has the spike shape changed? How has the potassium current during the spike changed? *Comment on the role of voltage-gated potassium channels in the downswing of the action potential.*

5. Give the potassium gate an intermediate opening rate, but a rather slow closing rate using `HH_model(K_a=0.2,K_b=0.0002)`. *Comment on the role of voltage-gated potassium channels in the relative refractory period.*

Appendix

Model:

$$
\begin{aligned}
C\dot{v} &= g_\ell(E_\ell - v) + \bar{g}_{\mathrm{K}} n^4 (E_{\mathrm{K}} - v) + \bar{g}_{\mathrm{Na}} m^3 h (E_{\mathrm{Na}} - v) + I \\
\dot{n} &= \alpha_n(v) \cdot (1 - n) - \beta_n(v) \cdot n \\
\dot{m} &= \alpha_m(v) \cdot (1 - m) - \beta_m(v) \cdot m \\
\dot{h} &= \alpha_h(v) \cdot (1 - h) - \beta_h(v) \cdot h.
\end{aligned}
\tag{1}
$$

Parameters:

The default parameters are for an excitatory pyramidal cell in rat cortex, taken from Table 2.1 in Gerstner et al. (2004), and are ultimately derived from Mainen et al. (1995), Huguenard et al. (1988), and Hamill et al. (1991).

x	E_x [mV]	$\bar{g}_x$ [mS/cm^2]
Na	55	40
K	-77	35
L	-65	0.3

	$\alpha(u)$	$\beta(u)$
n	$0.02(u-25)/\left[1-e^{-(u-25)/9}\right]$	$-0.002(u-25)/\left[1-e^{(u-25)/9}\right]$
m	$0.182(u+35)/\left[1-e^{-(u+35)/9}\right]$	$-0.124(u+35)/\left[1-e^{(u+35)/9}\right]$
h	$0.25e^{-(u+90)/12}$	$0.25e^{(u+62)/6}/e^{(u+90)/12}$

Removable singularity in the gating variables:

The gating expressions for n and m have a singularity at the threshold that we can remove using L'Hôpital's rule:

$$
\lim_{x\to 0} \frac{x}{1-e^{-x/r}} = \lim_{x\to 0} \frac{\frac{\mathrm{d}}{\mathrm{d}x}x}{\frac{\mathrm{d}}{\mathrm{d}x}(1-e^{-x/r})} = \lim_{x\to 0} \frac{1}{\frac{1}{c}e^{x/r}} = \frac{r}{e^0} = r \tag{2}
$$

We defined a singularity-free piecewise helper function for $\alpha(\cdot), \beta(\cdot)$:

$$
\text{gate\_nosingularity}(d; r) = \begin{cases} \frac{x}{1-\exp(-x/r)} & x \neq 0 \\ r & x = 0 \end{cases} \tag{3}
$$

Starter code:

```
# Create Matlab-like namespace
from pylab import *

class HH_model:
    # enclose parameters in an object without having to
    # unpack everything from `self` in instance methods.
    def __init__(self,
        # Potassium voltage gating (open/close) "n"
        KVth   = 25   , #  K  open threshold voltage (mV)
        Kr     = 9    , #  K  voltage gating scaling (mV)
        K_a    = 0.020, #  K  open  rate constant    (1/ms)
        K_b    = 0.002, #  K  close rate constant    (1/ms)
        # Sodium voltage gating (open/close) "m"
        NaVth  = -35  , #  Na open threshold voltage (mV)
        Na_a   = 0.182, #  Na open  rate constant    (1/ms)
        Na_b   = 0.124, #  Na close rate constant    (1/ms)
        Nar    = 9    , #  Na voltage gating scaling (mV)
        # Sodium voltage gating (inactivate/deinactivate) "h"
        HVin1  = -62  , #  Na inactivation threshold (mV)
        Hr1    =   6  , #  Na inactivation scale     (mV)
        HVin2  = -90  , #  Na deinactivation voltage (mV)
        Hr2    =  12  , #  Na deinactivation scale   (mV)
        H_k    = .25  , #  Na activation rate        (1/ms)
        # Specific maximal conductances
        gL     = 0.3  , # Leak conductance    (millisiemens/cm²)
        gKmax  = 35   , #  K  max conductance (millisiemens/cm²)
        gNamax = 45   , #  Na max conductance (millisiemens/cm²)
        # Reversal potentials
        EL     = -65  , # Leak reversal potential (mV)
        EK     = -77  , #  K   reversal potential (mV)
        ENa    =  55  , #  Na  reversal potential (mV)
        # Membrane specific capacitence
        C      = 1.0 ): # Membrane capacitance (microfarads /cm²)
        # exp clipped to avoid under/overflow
        exp  = lambda x  :np.exp(np.clip(x,-15,15))
        # Remove singularity from n,m gates
        gate = lambda x,r:np.where(x != 0, x / (1 - exp(-x / r)), r)
        #  / rate forms of n,m,h gates
        an   = lambda v: K_a *gate(  v-KVth  ,Kr )
        bn   = lambda v: K_b *gate(-(v-KVth ),Kr )
        am   = lambda v: Na_a*gate(  v-NaVth ,Nar)
        bm   = lambda v: Na_b*gate(-(v-NaVth),Nar)
        ah   = lambda v: H_k * exp(-(v-HVin2)/Hr2)
        bh   = lambda v: ah(v)*exp( (v-HVin1)/Hr1)
        # Current as a function of voltage, gates
        def currents(v,m,n,h):
            sK  = n**4
            sNa = m**3*h
            gK  = gKmax *sK
            gNa = gNamax*sNa
            IL  = gL *(EL -v)
            IK  = gK *(EK -v)
            INa = gNa*(ENa-v)
            return np.array([IL, IK, INa])
        # System ODE with input current I
        def ode(v,m,n,h,I):
            IL, IK, INa = self.currents(v,m,n,h)
            dv = (IK + INa + IL + I)/C
            dm = (1-m)*am(v) - bm(v)*m
            dn = (1-n)*an(v) - bn(v)*n
            dh = (1-h)*ah(v) - bh(v)*h
            return np.array([dv,dm,dn,dh])
        # Make everything above accessible in the object
        self.__dict__.update(locals())
```