Lecture 7

2025-10-15

Hodgkin–Huxley model; Exponential Euler method for numerical integration; Python classes for packaging code

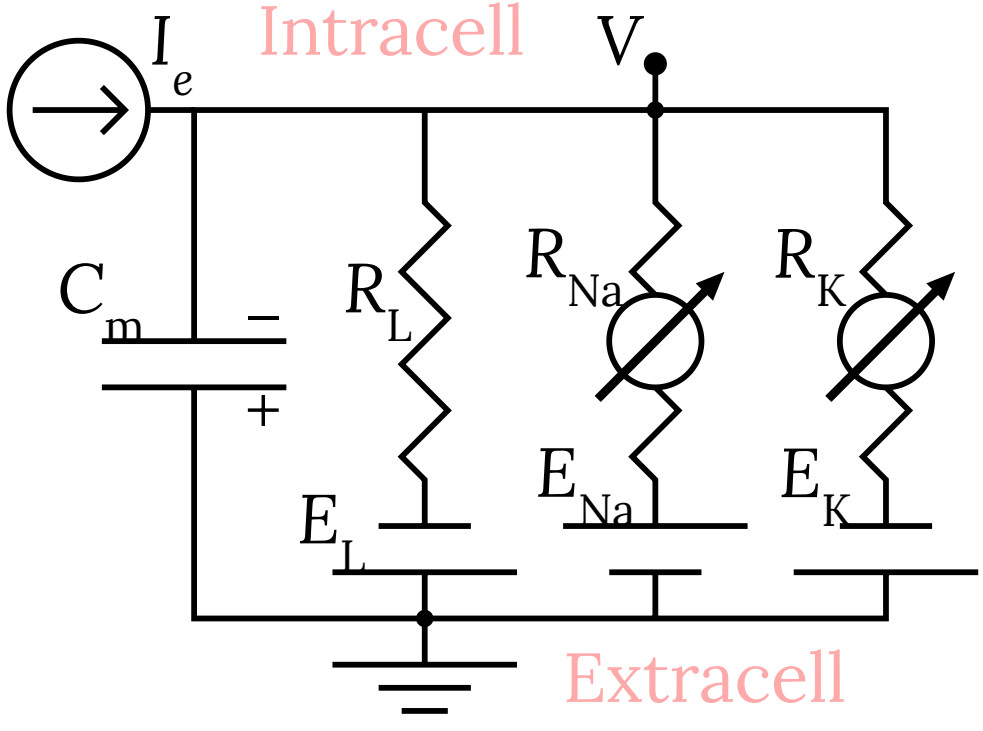
- Online slides: lec07-hodgkin_huxley.html
- Code: code07.ipynb
- Homework: hw07.pdf, hw07-data.npz, b2test.ipynb

Username: cns, Password: nycu2025

Hodgkin–Huxley model

$$i_{\mathrm{m}} = \bar{g}_{\mathrm{L}}(V - E_L) + \bar{g}_{\mathrm{K}} n^4 (V - E_K) + \bar{g}_{\mathrm{Na}} m^3 h (V - E_{\mathrm{Na}}) \tag{5.25}$$

with $$c_{\mathrm{m}} \frac{dV}{dt} = -i_{\mathrm{m}} + \frac{I_e}{A} \tag{5.6}$$



Gating variables:

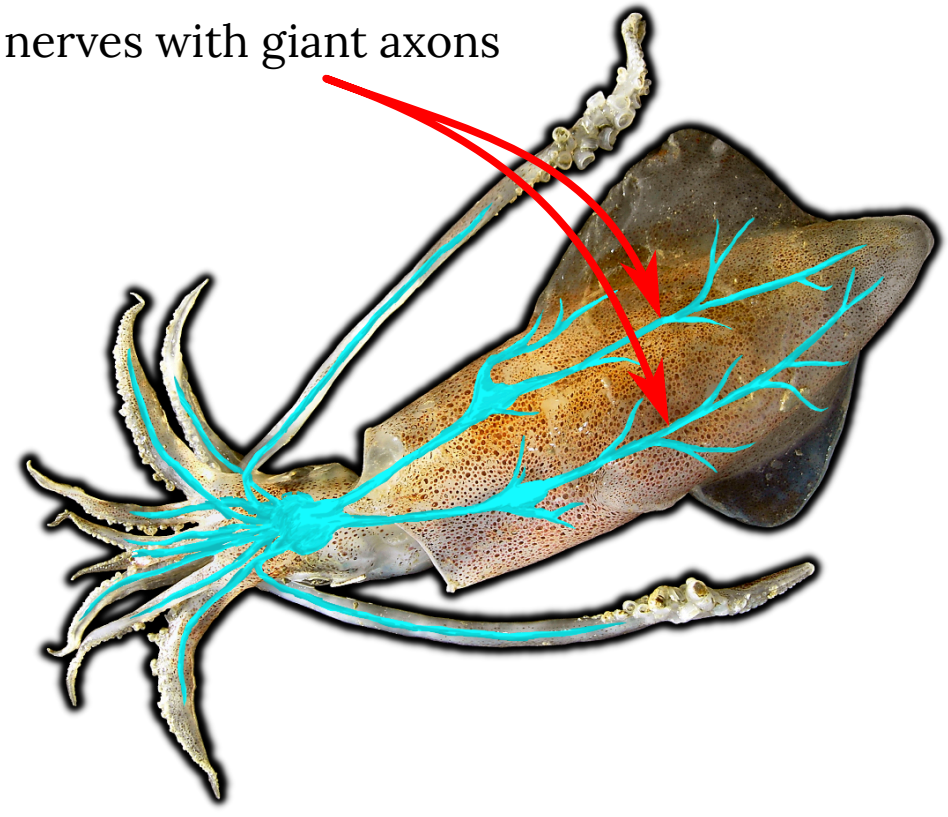
$$\frac{dn}{dt} = \alpha_n(1-n) - \beta_n n$$

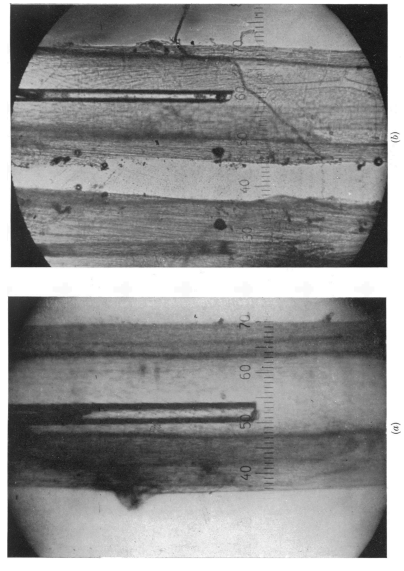
$$\frac{dm}{dt} = \alpha_m(1-m) - \beta_m m$$

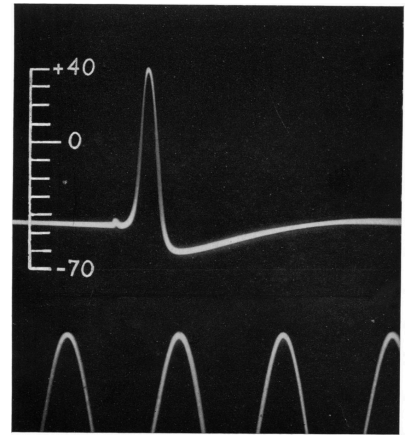
$$\frac{dh}{dt} = \alpha_h(1-h) - \beta_h h$$

Empirical equations from measurements

Giant axons of a squid



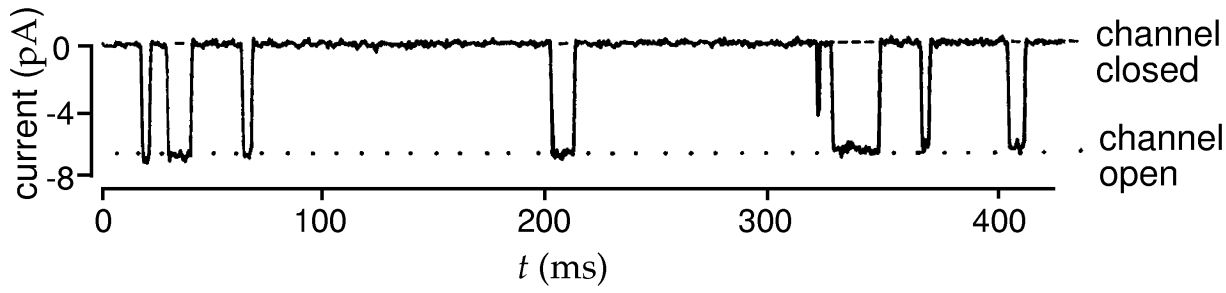




Hodgkin and Huxley (1952)

Voltage-dependent conductance

Acetylcholine-sensitive receptor channel (Dayan & Abbott 2007, Fig. 5.7) *"In the open state, the channel passes 6.6 pA at a holding potential of -140 mV. This is equivalent to more than 10^7 charges per second ... an open channel conductance of 47 pS. (From Hille, 1992)"*.



$$\text{Membrane Conductance} = \text{Channel Number} \times \text{Open Channel Conductance} \times \text{Open Channel Fraction}$$

$\bar{g}_i$: Maximal conductance P_i: Opening probability ... can be Voltage-, Transmitter-, or Ca^{2+}-dependent

⇒ Conductance of ion channel type i is $\bar{g}_i P_i$.

Stochasticity is taken care by the law of large number...

Synaptic conductance

- Ionotropic receptors: Fast

- Metabotropic receptors
  - Typically involves G-protein-mediated receptors
  - Also second messengers
  - Serotonin, dopamine, norepinephrine, acetylcholine

- Excitatory: Glutamate

  Receptors: AMPA (rapid), NMDA (slower, Ca^{2+}): Ionotropic

- Inhibitory: GABA (γ-aminobutyric acid)

  Receptors: $GABA_A$ (ionotropic Cl^-), $GABA_B$ (metabotropic K^+)

Gating of membrane channels

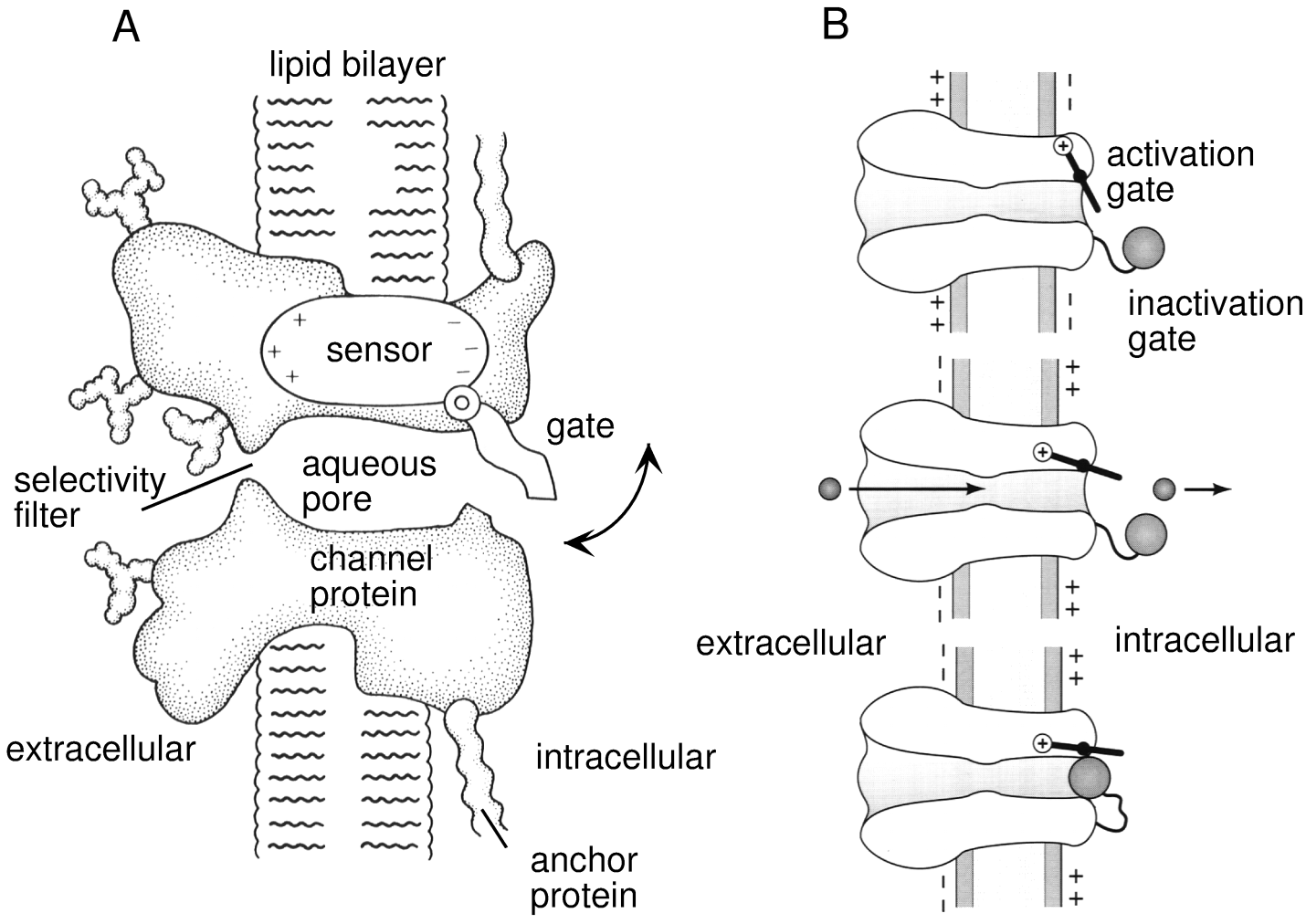


Figure 5.8, Dayan & Abbott (2007)

Ion channels in Hodgkin–Huxley model

Delayed-rectifier K^+ conductance

- Persistent conductance
- Open probability: $P_{\mathrm{K}} = n^k$, k: number of subunits.
- $0 \leq n \leq 1$: gating/activation variable

Fast Na^+ conductance

- Transient conductance
- Open probability: $P_{\mathrm{Na}} = m^k h$
- m: activation variable, h: inactivation variable

Channel kinetics

Opening rate $\alpha_n(V)$; Closing rate $\beta_n(V)$

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

$$\Rightarrow \quad \tau_n(V)\frac{dn}{dt} = n_\infty(V) - n$$

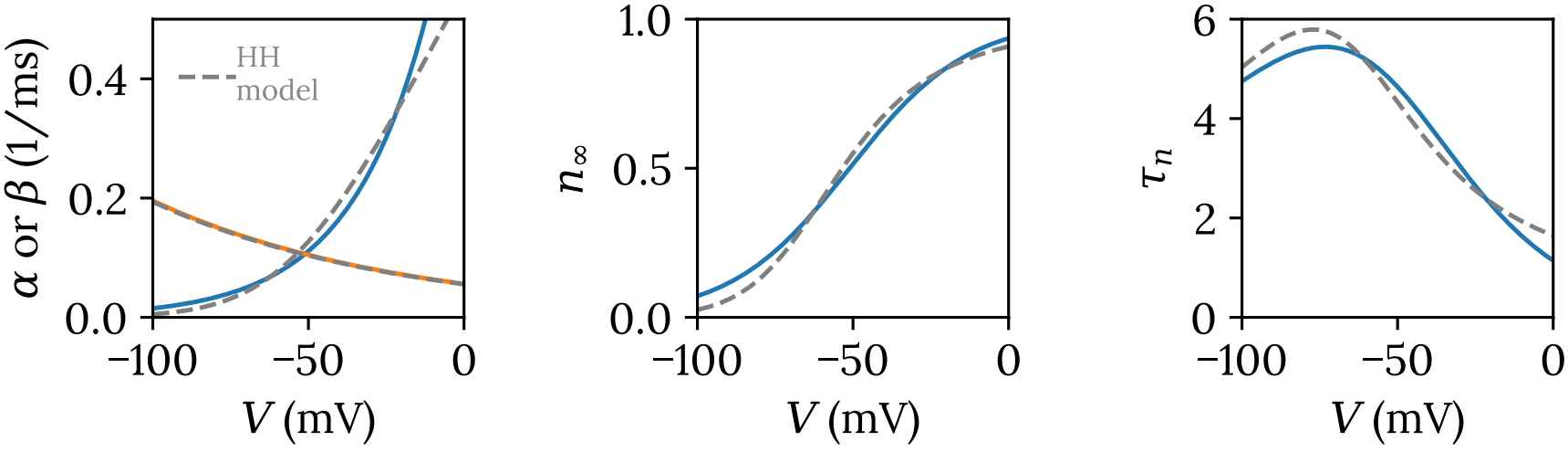
where $\tau_n(V) = \frac{1}{\alpha_n(V)+\beta_n(V)}$ and $n_\infty(V) = \frac{\alpha_n(V)}{\alpha_n(V)+\beta_n(V)}$.

For fixed V, n approaches $n_\infty(V)$ exponentially with time constant $\tau_n(V)$. From thermal fluctuation:

$$\alpha_n(V) = A_\alpha \exp(-qB_\alpha V/k_\mathrm{B}T) = A_\alpha \exp(-B_\alpha V/V_T)$$

And, similar for $\beta_n(V)$ with A_β and B_β.

Compare to Hodgkin–Huxley fits



Thermal consideration:

$A_\alpha = 0.82\ \text{ms}^{-1}$, $B_\alpha = -0.04\ \text{mV}^{-1}$, $A_\beta = 0.056\ \text{ms}^{-1}$, $B_\beta = 0.0125\ \text{mV}^{-1}$

Dashed lines: Hodgkin–Huxley

$$\alpha_n = \frac{0.01(V+55)}{1-\exp(-0.1(V+55))}$$

$$\beta_n = 0.125\exp(-0.0125(V+65))$$

Hodgkin–Huxley parameters (original)

x	E_x [mV]	g_x [mS/cm^2]
Na	−115	120
K	12	36
L	−10.613	0.3

Original paper from 1952 for squid giant axon

Making $V_\infty = 0$ as reference and "cm" as length unit.

x	$\alpha_x(V/\text{mV})$ [ms^{-1}]	$\beta_x(V/\text{mV})$ [ms^{-1}]
n	$.01(V+10)/\left[e^{(V+10)/10}-1\right]$	$.125e^{V/80}$
m	$.1(V+25)/\left[e^{(V+25)/10}-1\right]$	$4e^{-V/18}$
h	$.07e^{V/20}$	$1/\left[e^{(V+30)/10}+1\right]$

Hodgkin–Huxley 1952

Hodgkin–Huxley parameters (modern)

x	E_x [mV]	g_x [mS/mm^2]
Na	50	1.2
K	−77	0.36
L	−54.387	0.003

Redefine membrane potential: $V_{\mathrm{HH}} = V_{\mathrm{ref}} - V$ with $V_{\mathrm{ref}} = V_\infty = -65$ mV.
Use "mm" as length unit with $c_m = 0.1$ µF/mm², the specific membrane capacitance.

x	$\alpha_x(V/\mathrm{mV})$ [ms^{-1}]	$\beta_x(V/\mathrm{mV})$ [ms^{-1}]
n	$.01(V+55)/\left[1-e^{-.1(V+55)}\right]$	$.125e^{-.0125(V+65)}$
m	$.1(V+40)/\left[1-e^{-.1(V+40)}\right]$	$4e^{-.0556(V+65)}$
h	$.07e^{-.05(V+65)}$	$1/\left[1+e^{-.1(V+35)}\right]$

Dayan & Abbott 2007

Hodgkin–Huxley model for other neurons

x	E_x [mV]	g_x [mS/cm^2]
Na	55	40
K	−77	35
L	−65	0.3

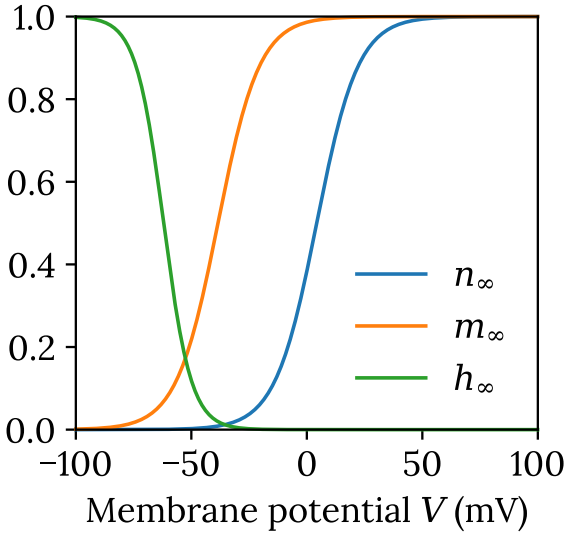
Cortical pyramidal neurons

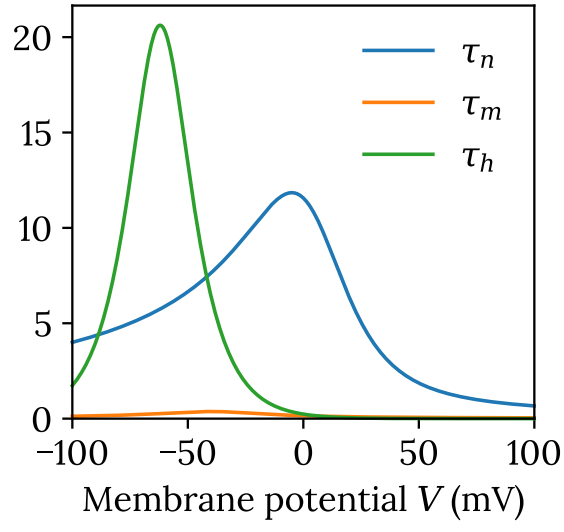
$c_m = 1\ \mu\text{F/cm}^2$

x	$\alpha_x(V/\text{mV})$ [ms^{-1}]	$\beta_x(V/\text{mV})$ [ms^{-1}]
n	$.02(V-25)/\left[1-e^{-(V-25)/9}\right]$	$-.002(V-25)/\left[1-e^{(V-25)/9}\right]$
m	$.182(V+35)/\left[1-e^{-(V+35)/9}\right]$	$-.124(V+35)/\left[1-e^{(V+35)/9}\right]$
h	$.25e^{-(V+90)/12}$	$.25e^{(V+62)/6}/e^{(V+90)/12}$

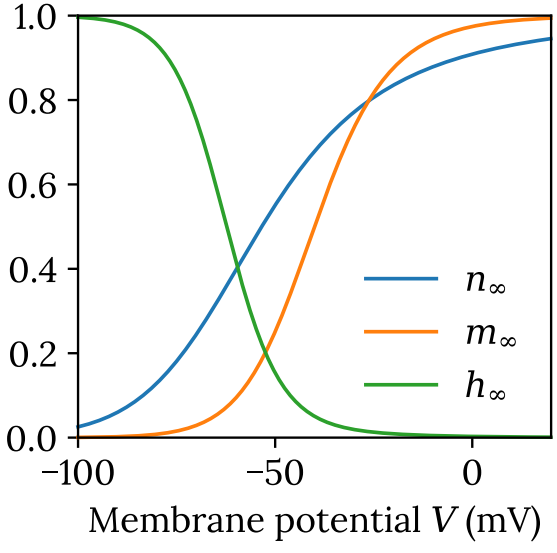
https://neuronaldynamics.epfl.ch/online/Ch2.S2.html

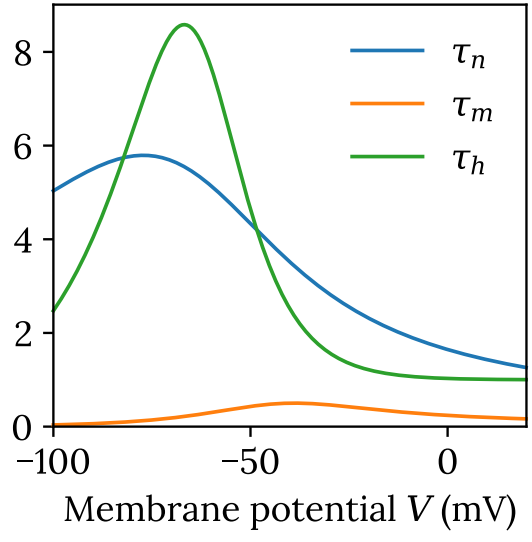
Compare gating variables





Cortical pyramidal cell





Squid giant axon

Integrating the Hodgkin–Huxley model

Vectorization: $y = (V, n, m, h)$. The same formulism

$$\frac{dy}{dt} = f(t, y)$$

remains applicable. Need to find $f(t, y)$, which is now also a vector.

```
def hh_f(t,y):
  [V,n,m,h] = y
  return np.array([
    (hh_g_L*(hh_E_L-V)+hh_g_K*n**4*(hh_E_K-V)
      +hh_g_Na*m**3*h*(hh_E_Na-V)+hh_ie(t))/hh_cm,
    hh_alpha_n(V)*(1-n)-hh_beta_n(V)*n,
    hh_alpha_m(V)*(1-m)-hh_beta_m(V)*m,
    hh_alpha_h(V)*(1-h)-hh_beta_h(V)*h
  ])
```

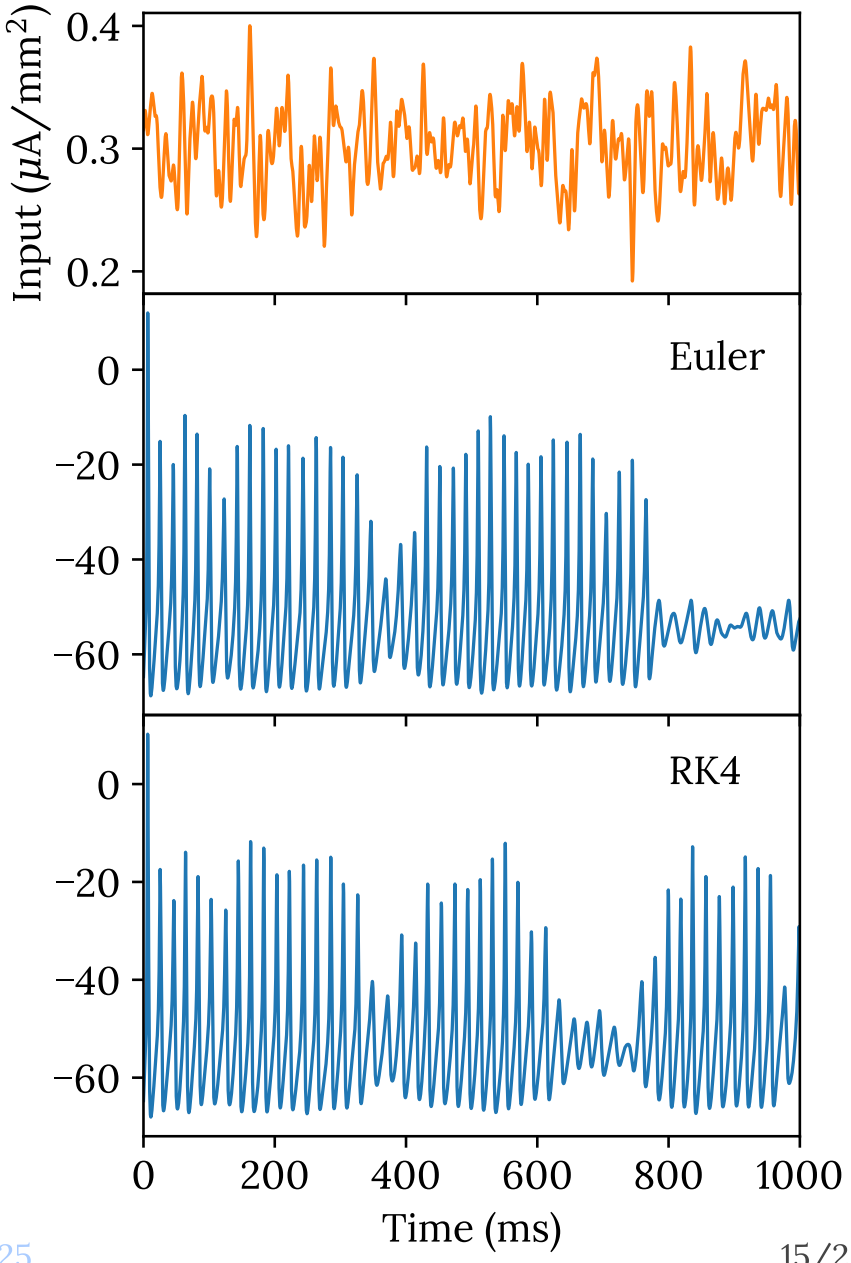
Numerical integration of ODEs

The same integration functions can be used directly with just a small modification.

```
def euler_step(t,y,f,dt):
  dyt = f(t,y)
  return y+dt*dyt

def runge_kutta(t,y,f,dt):
  k1 = f(t,y)
  k2 = f(t+dt/2,y+k1*dt/2)
  k3 = f(t+dt/2,y+k2*dt/2)
  k4 = f(t+dt,y+k3*dt)
  return y+dt*(k1+2*(k2+k3)+k4)/6
```

Results from some example protocol

```
T = 1000 # total time
dt = 0.1 # time between frames
nt = int(T/dt) # number of time frames
ts = np.arange(nt)*dt # time frames

# Some correlated random input for testing
wr = np.arange(-10,10,dt)
ww = np.exp(-wr**2/5)
ww /= np.sum(ww)
np.random.seed(123)
ii = np.convolve(np.random.normal(size=nt),
   ww,mode='same')
ii = np.convolve(ii,ww,mode='same')*.3+.3
ii = np.append(ii,ii[-1]) # for Runge-Kutta
ie = lambda t:ii[int(t/dt)]

# Initial condition
y0 = np.array([-65,.318,.053,.596])
```



Improving Euler for exponential decays

$$\frac{dx}{dt} = A - Bx \Rightarrow \tau_x \frac{dx}{dt} = x_\infty - x$$

where $x_\infty = A/B$ and $\tau_x = 1/B$.

Exact solution when x_∞ and τ are constant

$$x(t) = x_\infty - [x(t_0) - x_\infty]e^{-(t-t_0)/\tau_x}$$

Replace Euler incremental $x \to x + \Delta t(A - Bx)$ with

$$x(t) \to A/B + (x - A/B)e^{-\Delta t B}$$

$$= x_\infty + (x - x_\infty)e^{-\Delta t/\tau_x}$$

Re-factorize Hodgkin–Huxley

Casting membrane potential dynamics into the form:

$$\frac{dV}{dt} = A - BV,$$

we have

$$A = \frac{1}{c_{\mathrm{m}}}(\bar{g}_{\mathrm{L}}E_{\mathrm{L}} + \bar{g}_{\mathrm{K}}n^4E_{\mathrm{K}} + \bar{g}_{\mathrm{Na}}m^3hE_{\mathrm{Na}} + I_e/A)$$

$$B = \frac{1}{c_{\mathrm{m}}}(\bar{g}_{\mathrm{L}} + \bar{g}_{\mathrm{K}}n^4 + \bar{g}_{\mathrm{Na}}m^3h)$$

Implementing Euler exponential

```
def euler_step(t,y,f,dt):
  dyt = f(t,y)
  return y+dt*dyt

def euler_step_AB(t,y,A,B,dt):
  dyt = A(t,y)-B(t,y)*y
  return y+dt*dyt

def euler_step_ABi(t,y,A,B,dt):
  yinf = A(t,y)/(By:=B(t,y))
  return yinf+(y-yinf)*np.exp(-dt*By)

def euler_step_exp(t,y,yinf,tau,dt):
  yf = yinf(t,y)
  return yf+(y-yf)*np.exp(-dt/tau(t,y))
```

Whether to use (A, B) or (y_∞, τ) is a matter of convenience.

Packing model in a class

Some concepts in software engineering

- The problem of namespace pollution
- Modularization for reusability
- Code isolation, limiting interface
- Data encapsulation
- Object-oriented programming

Python class

```
class Person:
  def __init__(self, name, age): # constructor
    self.name = name
    self.age = age
```

```
p1 = Person("John", 36) # create object
print(p1.name) # access members
print(p1.age)
```

The HodgkinHuxley class

Constructor

```
class HodgkinHuxley: # default to Dayan-Abbott version
  def __init__(
    self,
    E_Na=50,g_Na=1.2,
    E_K=-77,g_K=0.36,
    E_L=-54.387,g_L=0.003,
    cm=0.1,
    alpha_n=lambda v:.01*(v+55)/(1-np.exp(-.1*(v+55))),
    beta_n= lambda v:.125*np.exp(-.0125*(v+65)),
    alpha_m=lambda v:.1*(v+40)/(1-np.exp(-.1*(v+40))),
    beta_m= lambda v:4*np.exp(-.0556*(v+65)),
    alpha_h=lambda v:.07*np.exp(-.05*(v+65)),
    beta_h= lambda v:1/(1+np.exp(-.1*(v+35)))
  ):
    self.E_Na = E_Na
    self.g_Na = g_Na
    self.E_K = E_K
    self.g_K = g_K
    self.E_L = E_L
    self.g_L = g_L
    self.cm = cm
    self.alpha_n = alpha_n
    self.beta_n = beta_n
    self.alpha_m = alpha_m
    self.beta_m = beta_m
    self.alpha_h = alpha_h
    self.beta_h = beta_h
    self.ie = lambda t:0
```

Member functions

```
def f(self,t,y):
  [V,n,m,h] = y
  return np.array([
    (self.g_L*(self.E_L-V)+
      self.g_K*n**4*(self.E_K-V)+
      self.g_Na*m**3*h*(self.E_Na-V)+
      self.ie(t))/self.cm,
    self.alpha_n(V)*(1-n)-self.beta_n(V)*n,
    self.alpha_m(V)*(1-m)-self.beta_m(V)*m,
    self.alpha_h(V)*(1-h)-self.beta_h(V)*h
  ])
def A(self,t,y):
  [V,n,m,h] = y
  return np.array([
    (self.g_L*self.E_L+
      self.g_K*n**4*self.E_K+
      self.g_Na*m**3*h*self.E_Na+
      self.ie(t))/self.cm,
    self.alpha_n(V),
    self.alpha_m(V),
    self.alpha_h(V)
  ])
def B(self,t,y):
  [V,n,m,h] = y
  return np.array([
    (self.g_L+self.g_K*n**4+
      self.g_Na*m**3*h)/self.cm,
    self.alpha_n(V)+self.beta_n(V),
    self.alpha_m(V)+self.beta_m(V),
    self.alpha_h(V)+self.beta_h(V)
  ])
```

Using a simulator

- NEURON https://neuron.yale.edu/neuron/
- GENESIS http://genesis-sim.org/
- NEST https://www.nest-simulator.org/
- Brian, Brian2 https://briansimulator.org/

Some technical notes on Python coding

- Positional, named, and default arguments for functions
- Assignment lists or arrays to variables

See code07.ipynb