Single Neuron Model (2):

Hodgkin-Huxley model coding

BrainPy Overview

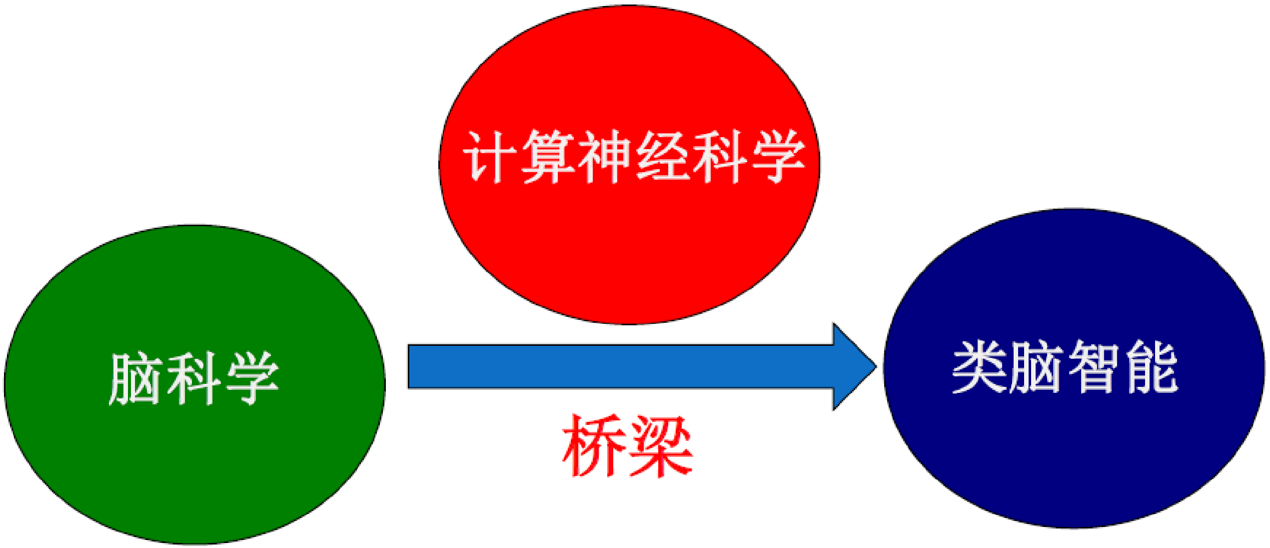
What is BrainPy?

BrainPy is a Python library designed for high-performance flexible brain modeling.

Among its key ingredients it supports:

- **General numerical solvers**
  - ☐ Ordinary differential equations
  - ☐ Stochastic differential equations
  - ☐ Delayed differential equations
  - ☐ Fractional differential equations
- **Neurodynamics simulation tools**
  - ☐ Support brain objects, such like neurons, synapses, networks, soma, dendrites, channels, and even molecular.
- **Neurodynamics analysis tools**
  - ☐ Support phase plane analysis and bifurcation analysis (future support continuation analysis).

What is Computational Neuroscience?



目标:

- 用数学方法/模型来阐明脑的工作原理
- 为类脑智能提供新思想和理论基础

Requirements for Research in Computational Neuroscience

1. Easy to learn and use

- Computational neuroscientists with the background of biology, mathematics, physics, etc. usually are not skilled at the programming.

2. Flexible and transparent

- Computational neuroscientists usually need create new dynamical systems, and has the requirement to control all the underlying things (make sure the support of framework match the updating logic of the new model).

3. Extensible and scalable

- New concepts, new models, and new methods are constantly emerging in computational neuroscience.

4. Efficient running speed

- Running speed should be efficient to speed up the model simulation.

	Low-level programming			Descriptive languages		
	NEURON	**NEST**	**CARLsim**	**Brian2**	**BMTK**	**GENN**
Host	Yale University	Blue Brain Project	UC, Irvine	Sorbonne Université	Allen Brain Institute	University of Sussex
编程语言	Hoc and NMODL, Python interface (PyNeuron)	SLI, Python interface (PyNEST)	C++, CUDA, Python Interface (PyCARL)	Python, Cython, C++	Python	C++ / CUDA, Python interface (PyGeNN)
开发历史	>24	>14	>11	>13	>3	>6
学习难度	高	高	高	低	低	高
灵活性	差	差	差	好	差	好
运行速度	较好	较好	高	较好	较好	高
透明程度	差	差	好	差	差	好
动力学分析	无	无	无	无	无	无
面向领域	multi-scale modeling	large-scale modeling	large-scale modeling	point models	multi-scale modeling	large-scale modeling

Why Create Another Neural Simulator?

Representative Neural Simulators : NEURON, NEST

	Low-level programming			Descriptive languages		
	NEURON	**NEST**	**CARLsim**	**Brian2**	**BMTK**	**GENN**
Host	Yale University	Blue Brain Project	UC, Irvine	Sorbonne Université	Allen Brain Institute	University of Sussex
编程语言	Hoc and NMODL, Python interface (PyNeuron)	SLI, Python interface (PyNEST)	C++, CUDA, Python Interface (PyCARL)	Python, Cython, C++	Python	C++ / CUDA, Python interface (PyGeNN)
开发历史	>24	>14	>11	>13	>3	>6
学习难度	高	高	高	低	低	高
灵活性	差	差	差	好	差	好
运行速度	较好	较好	高	较好	较好	高
透明程度	差	差	好	差	差	好
动力学分析	无	无	无	无	无	无
面向领域	multi-scale modeling	large-scale modeling	large-scale modeling	point models	multi-scale modeling	large-scale modeling

```python
import nest

nest.ResetKernel()
nest.SetKernelStatus(
    {"resolution": 0.05})
neuron_param = {"tau_m": Tau,
                "t_ref": TauR,
                "tau_syn_ex": Tau_psc,
                "tau_syn_in": Tau_psc,
                "C_m": C,
                "V_reset": E_L,
                "E_L": E_L,
                "V_m": E_L,
                "V_th": Theta}
neurons = nest.Create("iaf_psc_exp", 2)
neurons.set(neuron_param)
```

```cpp
#include "iaf_psc_exp.h"

// C++ includes:
#include <limits>

// Includes from libnestutil:
#include "dict_util.h"
#include "numerics.h"
#include "propagator_stability.h"

// Includes from nestkernel:
#include "event_delivery_manager_impl.h"
#include "exceptions.h"
#include "kernel_manager.h"
#include "universal_data_logger_impl.h"
```

```python
G = NeuronGroup(10, '''dv/dt = I_leak / Cm : volt
                        I_leak = g_L*(E_L - v) : amp''')
```

Code Generation

```cpp
for(int _idx=0; _idx<_num_idx; _idx++)
{
    double v = _array_neurongroup_v[_idx];
    double w = _array_neurongroup_w[_idx];
    const double _w = -(dt) * w / tau_w + w;
    const double _v = dt * (-(v) + w) / tau_v + v;
    w = _w;
    v = _v;
    _array_neurongroup_v[_idx] = v;
    _array_neurongroup_w[_idx] = w;
}
```

- **Not flexible enough.**

	NEURON	NEST	BRIAN	GENESIS
Neuron model without dynamics	M	M	Y	M
Neuron model with simplified and discontinuous dynamics *Examples: Leaky Integrate-and-Fire (LIF), Izhikevich or Quadratic LIF; Exponent Leaky Integrate-and-Fire (eLIF)*	M	M	Y	M
Neuron model with simplified and continues dynamics *Examples: FitzHugh–Nagumo, Morris–Lecar*	M	M	Y	M
Single compartment, conductance-based model—temporal integration (point neuron) *Examples: Single-Compartment Hodgkin–Huxley model*	YG	M	Y	Y
Can conductance-based descriptions of ion channels be added to the neuron model? *Example: h-channel*	YG/M	m	Y	M
Neuron model with simplify morphology (2-compartment model) *Example: Pinsky–Rinzel model*	YG	M	Y	M
New model of chemical synapse	M	m	Y	M
New model of electrical synapse	M	m	Y	M
New model of learning rule	m	M	Y	M

Tikidji-Hamburyan, Ruben A., et al. "Software for brain network simulations: a comparative study." Frontiers in Neuroinformatics 11 (2017): 46.

Representative Neural Simulator：BRIAN

	Low-level programming			Descriptive languages		
	NEURON	NEST	CARLsim	Brian2	BMTK	GENN
Host	Yale University	Blue Brain Project	UC, Irvine	Sorbonne Université	Allen Brain Institute	University of Sussex
编程语言	Hoc and NMODL, Python interface (PyNeuron)	SLI, Python interface (PyNEST)	C++, CUDA, Python Interface (PyCARL)	Python, Cython, C++	Python	C++ / CUDA, Python interface (PyGeNN)
开发历史	>24	>14	>11	>13	>3	>6
学习难度	高	高	高	低	低	高
灵活性	差	差	差	好	差	好
运行速度	较好	较好	高	较好	较好	高
透明程度	差	差	好	差	差	好
动力学分析	无	无	无	无	无	无
面向领域	multi-scale modeling	large-scale modeling	large-scale modeling	point models	multi-scale modeling	large-scale modeling

```python
import nest

nest.ResetKernel()
nest.SetKernelStatus(
    {"resolution": 0.05})
neuron_param = {"tau_m": Tau,
                "t_ref": TauR,
                "tau_syn_ex": Tau_psc,
                "tau_syn_in": Tau_psc,
                "C_m": C,
                "V_reset": E_L,
                "E_L": E_L,
                "V_m": E_L,
                "V_th": Theta}
neurons = nest.Create("iaf_psc_exp", 2)
neurons.set(neuron_param)
```

```cpp
#include "iaf_psc_exp.h"

// C++ includes:
#include <limits>

// Includes from libnestutil:
#include "dict_util.h"
#include "numerics.h"
#include "propagator_stability.h"

// Includes from nestkernel:
#include "event_delivery_manager_impl.h"
#include "exceptions.h"
#include "kernel_manager.h"
#include "universal_data_logger_impl.h"
```

```python
G = NeuronGroup(10, '''dv/dt = I_leak / Cm : volt
                       I_leak = g_L*(E_L - v) : amp''')
```

Code Generation

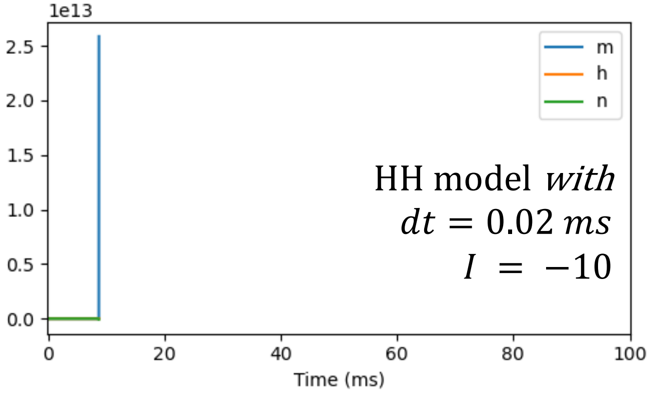
```cpp
for(int _idx=0; _idx<_num_idx; _idx++)
{
    double v = _array_neurongroup_v[_idx];
    double w = _array_neurongroup_w[_idx];
    const double _w = -(dt) * w / tau_w + w;
    const double _v = dt * (-(v) + w) / tau_v + v;
    w = _w;
    v = _v;
    _array_neurongroup_v[_idx] = v;
    _array_neurongroup_w[_idx] = w;
}
```

- *Code generation* approach has **intrinsic limitations on *Flexibility and Transparence*** .

1. String description is pseudo programming, greatly reducing the program expressive power

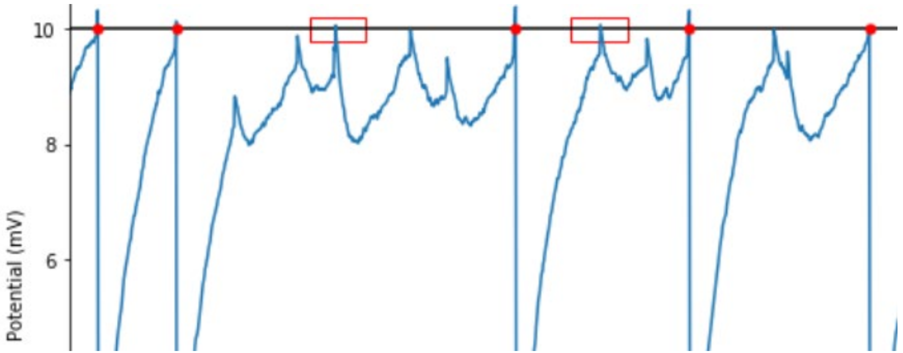
```
dm/dt = alpha_m*(1-m)-beta_m*m : 1
dn/dt = alpha_n*(1-n)-beta_n*n : 1
dh/dt = alpha_h*(1-h)-beta_h*h : 1
```

```
m = np.clip(int_m(ST['m'], _t_, ST['V']), 0., 1.)
h = np.clip(int_h(ST['h'], _t_, ST['V']), 0., 1.)
n = np.clip(int_n(ST['n'], _t_, ST['V']), 0., 1.)
```



2. 代码对用户隐藏，不支持debug，不知道是否生成用户想到的逻辑。一旦发现错误用户无法纠正代码。这往往导致用户定义新模型时捉襟见肘。

```
S = Synapses(source=neurons, target=neurons,
 model='w : 1; I_post=w*(V_pre-V_post): 1 (summed)',
on_pre=f'V_post += w * {k_spikelet}')
```



3. 对不满足假设与规定的模型不支持

Intrinsic limitations on extensibility and scalability

动力学分析

- A “brute force” simulation approach is hardly effective and accurate in practice
- Some complex dynamical equations are hard to analyze by hand, and can only be analyzed by computer optimization

(Meijer H., et al., 2009)

- New analysis methods for dynamical systems have been proposed.

新微分方程

- Delayed differential equations and fractional differential equations have been another frontiers in brain modeling.

(Lundstrom, et al., Nature Neuroscience, 2008; Gerhard Werner, frontiers, 2010)

深度网络模型

- Deep neural networks have been demonstrated as a very successful mothod for brain modeling.

(Daniel L. K. Yamins, et al., PNAS, 2014; Kohitij Kar, et al. Nature Neuroscience, 2018)

What is BrainPy?

BrainPy is a Python library designed for high-performance flexible brain modeling.

Among its key ingredients it supports:

- **General numerical solvers**
  - Ordinary differential equations
  - Stochastic differential equations
  - Delayed differential equations
  - Fractional differential equations
- **Dynamics simulation tools**
  - Support brain objects, such like neurons, synapses, networks, soma, dendrites, channels, and even molecular.
- **Dynamics analysis tools**
  - Support phase plane analysis, bifurcation analysis, numerical continuation, and Sensitivity analysis.
- **Seamless integration with deep learning models**

Install BrainPy

方法一：pypi

brainpy-simulator 1.0.2

```
pip install brainpy-simulator
```

方法二：conda

Installers

conda install

noarch v1.0.2

To install this package with conda run:

```
conda install -c brainpy brainpy-simulator
```

方法三：GitHub

github.com/PKU-NIP-Lab/BrainPy

PKU-NIP-Lab / **BrainPy**

Coding HH model with BrainPy

Brain modeling by using differential equations

- Neuronal activities can be described by a set of differential equations.

$$\frac{dx}{dt} = f(x) + g(x)dW$$

- **Basic question**: How to solve the differential equations?

$$x(t) = ?$$

- Single neuron modeling --- Hodgkin-Huxley equations

$$C_m \frac{dV}{dt} = -\bar{g}_{\mathrm{K}} n^4 (V - V_K) - \bar{g}_{\mathrm{Na}} m^3 h (V - V_{Na}) - \bar{g}_l (V - V_l) + I_{syn}$$
$$\frac{dm}{dt} = \alpha_m(V)(1 - m) - \beta_m(V) m$$
$$\frac{dh}{dt} = \alpha_h(V)(1 - h) - \beta_h(V) h$$
$$\frac{dn}{dt} = \alpha_n(V)(1 - n) - \beta_n(V) n$$

$$\boxed{V(t) = ?}$$

Methods to solve differential equations

- Get algebraic solution

$$\frac{dy}{dx} = x^2 - 3 \quad \Rightarrow \quad y = \frac{x^3}{3} - 3x + K$$

$$\frac{d\theta}{dt} = \frac{\sin(t + 0.2)}{\theta^2} \quad \Rightarrow \quad \frac{\theta^3}{3} = -\cos(t + 0.2) + K$$

- Numerical integration

Euler's Method

$$y(t + dt) \approx y(t) + dt\,y'(t) + \frac{dt^2\,y''(t)}{2!} + \frac{dt^3\,y'''(t)}{3!} + \frac{dt^4\,y^{\mathrm{iv}}(t)}{4!} + \ldots$$

$$y(t + dt) \approx y(t) + dt\,y'(t)$$

Solving HH neuron model by Numerical Method

Euler

$$m_t = m_{t-1} + \left[\alpha_m(V_{t-1})(1 - m_{t-1}) - \beta_m(V_{t-1})m_{t-1}\right] * dt$$

$$h_t = h_{t-1} + \left[\alpha_h(V_{t-1})(1 - h_{t-1}) - \beta_h(V_{t-1})h_{t-1}\right] * dt$$

$$n_t = n_{t-1} + \left[\alpha_n(V_{t-1})(1 - n_{t-1}) - \beta_n(V_{t-1})n_{t-1}\right] * dt$$

$$V_t = V_{t-1} + \left[\frac{-\bar{g}_K n_{t-1}^4(V_{t-1} - V_K) - \bar{g}_{Na} m_{t-1}^3 h_{t-1}(V_{t-1} - V_{Na}) - \bar{g}_l(V_{t-1} - V_l) + I_{syn}}{C_m}\right] * dt$$

RK4

$$k1 = f(t_n, y_n)$$

$$k_2 = f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_1\right)$$

$$k_3 = f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_2\right)$$

$$k_4 = f(t_n + h_n, y_n + hk_3)$$

$$y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

Exponential Euler

$$\frac{dy}{dt} = A - By$$

$$y(t + dt) = y(t)e^{-B*dt} + \frac{A}{B}(1 - e^{-B*dt})$$

Support for Ordinary Differential Equations

An ODE system

$$\frac{dx}{dt} = f_1(x, t, y, p_1)$$
$$\frac{dy}{dt} = f_2(y, t, x, p_2)$$

ODE as a Python function

- Can be a **scalar**
- Can be a **vector / matrix**

Variables

Parameters

```python
def diff(x, y, t, p1, p2):
    dx = f1(x, t, y, p1)
    dy = g1(y, t, x, p2)
    return dx, dy
```

- Can be a **system**: group of variables

```python
import numpy as np

def diff(xy, t, p1, p2):
    x, y = xy
    dx = f1(x, t, y, p1)
    dy = g1(y, t, x, p2)
    return np.array([dx, dy])
```

Simple decorator for numerical integration

```python
@bp.odeint(method='rk4', dt=0.01)
def diff(x, y, t, p1, p2):
    dx = f1(x, t, y, p1)
    dy = g1(y, t, x, p2)
    return dx, dy
```

Numerical method

Numerical precision

Supported ODE Numerical Methods

Runge-Kutta Methods	**Adaptive Runge-Kutta Methods**	**Other Methods**
Euler	Runge–Kutta–Fehlberg 4(5)	Exponential Euler
Midpoint	Runge–Kutta–Fehlberg 1(2)	
Heun's second-order method	Dormand–Prince method	
Ralston's second-order method	Cash–Karp method	
RK2	Bogacki–Shampine method	
RK3	Heun–Euler method	
RK4		
Heun's third-order method		
Ralston's third-order method		
Third-order Strong Stability Preserving Runge-Kutta		
Ralston's fourth-order method		
Runge-Kutta 3/8-rule fourth-order method		

Neuron Model: brainpy.NeuGroup

```python
[7]: class HH(bp.NeuGroup):
         target_backend = 'general'

         @staticmethod
         def diff(V, m, h, n, t, Iext, gNa, ENa, gK, EK, gL, EL, C):
             alpha = 0.1 * (V + 40) / (1 - bp.ops.exp(-(V + 40) / 10))
             beta = 4.0 * bp.ops.exp(-(V + 65) / 18)
             dmdt = alpha * (1 - m) - beta * m

             alpha = 0.07 * bp.ops.exp(-(V + 65) / 20.)
             beta = 1 / (1 + bp.ops.exp(-(V + 35) / 10))
             dhdt = alpha * (1 - h) - beta * h

             alpha = 0.01 * (V + 55) / (1 - bp.ops.exp(-(V + 55) / 10))
             beta = 0.125 * bp.ops.exp(-(V + 65) / 80)
             dndt = alpha * (1 - n) - beta * n

             I_Na = (gNa * m ** 3.0 * h) * (V - ENa)
             I_K = (gK * n ** 4.0) * (V - EK)
             I_leak = gL * (V - EL)
             dVdt = (- I_Na - I_K - I_leak + Iext) / C

             return dVdt, dmdt, dhdt, dndt
```

Set the target backend the model going to run.

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

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

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

$$C\frac{dV}{dt} = -\left(\bar{g}_{Na} m^3 h(V - E_{Na}) + \bar{g}_K n^4 (V - E_K) + g_{leak}(V - E_{leak})\right) + I(t)$$

```python
def __init__(self, size, ENa=50., EK=-77., EL=-54.387,
             C=1.0, gNa=120., gK=36., gL=0.03, V_th=20.,
             **kwargs):
    # parameters
    self.ENa = ENa
    self.EK = EK
    self.EL = EL
    self.C = C
    self.gNa = gNa
    self.gK = gK
    self.gL = gL
    self.V_th = V_th

    # variables
    self.V = bp.ops.ones(size) * -65.
    self.m = bp.ops.ones(size) * 0.5
    self.h = bp.ops.ones(size) * 0.6
    self.n = bp.ops.ones(size) * 0.32
    self.spike = bp.ops.zeros(size)
    self.input = bp.ops.zeros(size)

    self.integral = bp.odeint(f=self.diff, method='rk4', dt=0.01)
    super(HH, self).__init__(size=size, **kwargs)
```

Initialize Parameters

Initialize Variables

Initialize Base Class

```python
def update(self, _t):
    V, m, h, n = self.integral(self.V, self.m, self.h, self.n, _t,
                               self.input, self.gNa, self.ENa, self.gK,
                               self.EK, self.gL, self.EL, self.C)
    self.spike = (self.V < self.V_th) * (V >= self.V_th)
    self.V = V
    self.m = m
    self.h = h
    self.n = n
    self.input[:] = 0
```

Integrate variables in $[t - dt, t]$

Whether generate a spike?

Update variables

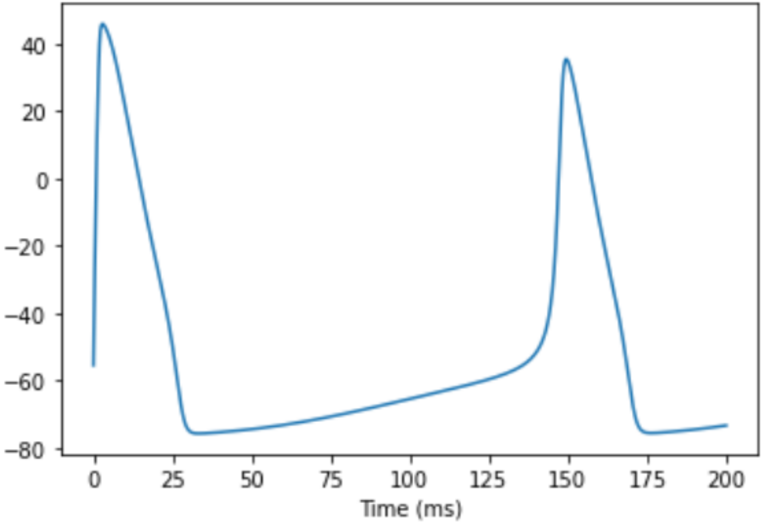
```
[9]: bp.backend.set('numpy')

     group = HH(100, monitors=['V'])

     group.run(200., inputs=('input', 10.))

     bp.visualize.line_plot(group.mon.ts, group.mon.V, show=True)
```

强大的inputs支持，(key, value, ops), 支持 +, -, *, /, = 赋值



Exercise

1. 安装Anaconda Python环境(https://docs.anaconda.com/anaconda/install/)

2. 安装 BrainPy == 1.0.2 (https://pypi.org/project/brainpy-simulator/1.0.2/)

3. 阅读HH模型：
   - https://brainmodels.readthedocs.io/en/latest/tutorials/neurons/HH_model.html
   - https://brainpy.readthedocs.io/en/latest/tutorials_advanced/ode_numerical_solvers.html#Exponential-Euler-methods

4. Implement HH model