Reduced Models

brain dynamics programming

张天秋
Peking University

目录 CONTENTS

北京大学 PEKING UNIVERSITY

北京大学
PEKING UNIVERSITY

01

LIF neuron models programming

The LIF neuron model

1.Define LIF `class`

2. Initialization

parameters

variables

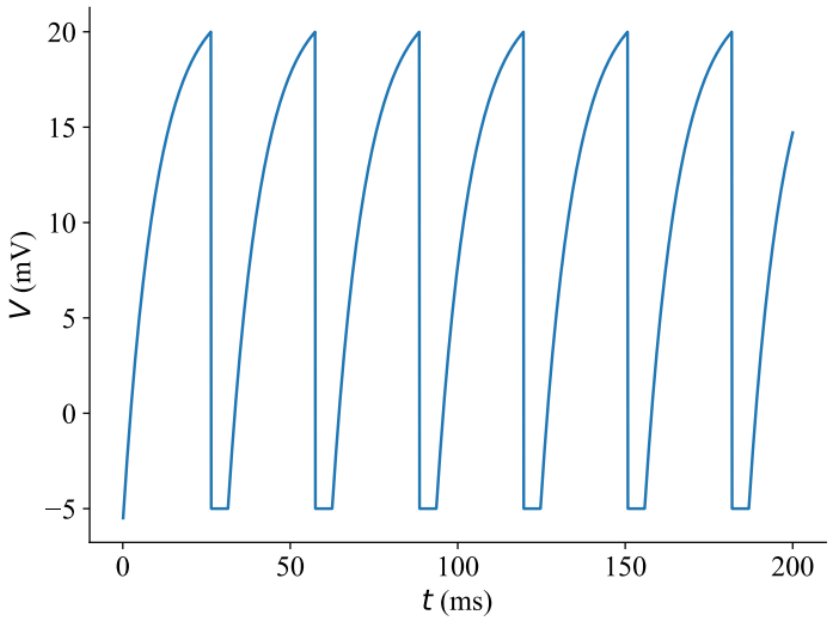
integral function

3. Define the derivative function

4. Complete the `update()` function

$$\tau \frac{dV}{dt} = -(V - V_{rest}) + RI(t)$$

$$\text{if } V > V_{th}, \quad V \leftarrow V_{reset} \text{ last } t_{ref}$$



The LIF neuron model

北京大学 PEKING UNIVERSITY

1.Define LIF `class`

2. Initialization

parameters

variables

integral function

```python
class LIF(bp.dyn.NeuDyn):
  def __init__(self, size, V_rest=0., V_reset=-5., V_th=20., R=1., tau=10., t_ref=5., **kwargs):
    # 初始化父类
    super(LIF, self).__init__(size=size, **kwargs)

    # 初始化参数
    self.V_rest = V_rest
    self.V_reset = V_reset
    self.V_th = V_th
    self.R = R
    self.tau = tau
    self.t_ref = t_ref  # 不应期时长

    # 初始化变量
    self.V = bm.Variable(bm.random.randn(self.num) + V_reset)
    self.input = bm.Variable(bm.zeros(self.num))
    self.t_last_spike = bm.Variable(bm.ones(self.num) * -1e7)  # 上一次脉冲发放时间
    self.refractory = bm.Variable(bm.zeros(self.num, dtype=bool))  # 是否处于不应期
    self.spike = bm.Variable(bm.zeros(self.num, dtype=bool))  # 脉冲发放状态

    # 使用指数欧拉方法进行积分
    self.integral = bp.odeint(f=self.derivative, method='exponential_euler')
```

The LIF neuron model

1.Define LIF `class`

2. Initialization

parameters

variables

integral function

3. Define the derivative function

```
# 定义膜电位关于时间变化的微分方程
def derivative(self, V, t, Iext):
    dvdt = (-V + self.V_rest + self.R * Iext) / self.tau
    return dvdt
```

$$\tau \frac{\mathrm{d}V}{\mathrm{d}t} = -(V - V_{\mathrm{rest}}) + RI(t)$$

$$\text{if } V > V_{\mathrm{th}}, \quad V \leftarrow V_{\mathrm{reset}} \text{ last } t_{\mathrm{ref}}$$

The LIF neuron model

北京大学 PEKING UNIVERSITY

1.Define LIF `class`

2. Initialization

   parameters

   variables

   integral function

```
def update(self):
  _t, _dt = bp.share['t'], bp.share['dt']
  # 以数组的方式对神经元进行更新
  refractory = (_t - self.t_last_spike) <= self.t_ref  # 判断神经元是否处于不应期
  V = self.integral(self.V, _t, self.input, dt=_dt)  # 根据时间步长更新膜电位
  V = bm.where(refractory, self.V, V)  # 若处于不应期，则返回原始膜电位self.V，否则返回更新后的膜电位V
  spike = V > self.V_th  # 将大于阈值的神经元标记为发放了脉冲
  self.spike[:] = spike  # 更新神经元脉冲发放状态
  self.t_last_spike[:] = bm.where(spike, _t, self.t_last_spike)  # 更新最后一次脉冲发放时间
  self.V[:] = bm.where(spike, self.V_reset, V)  # 将发放了脉冲的神经元膜电位置为V_reset，其余不变
  self.refractory[:] = bm.logical_or(refractory, spike)  # 更新神经元是否处于不应期
  self.input[:] = 0.  # 重置外界输入
```

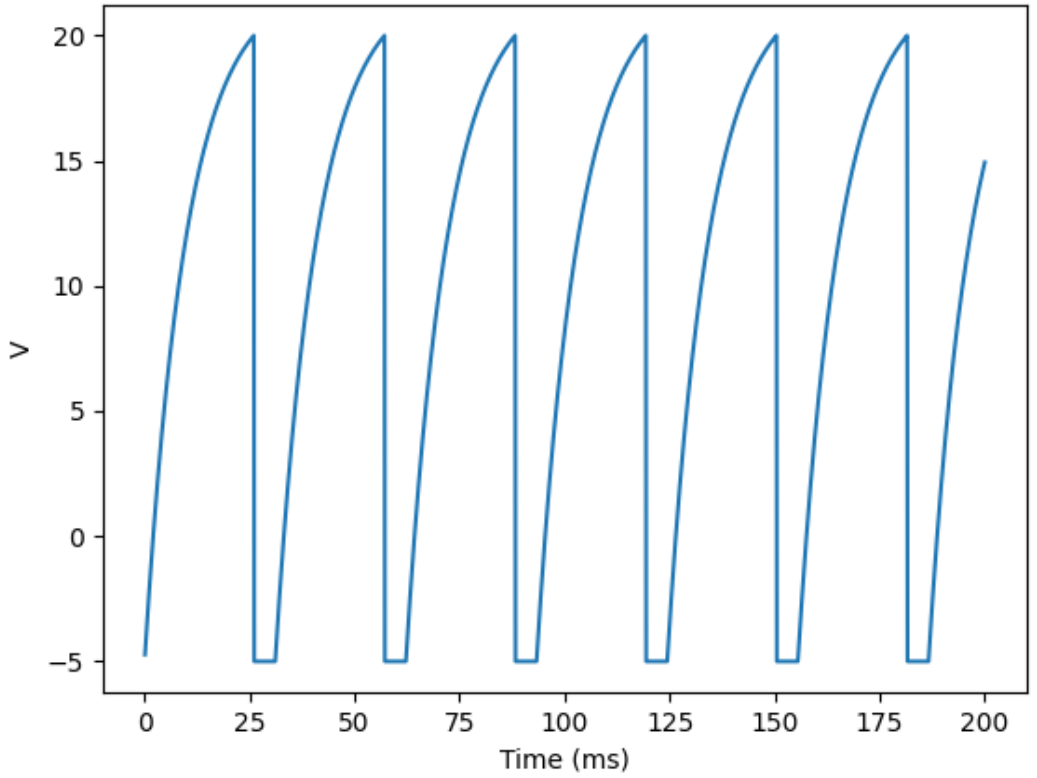
3. Define the derivative function

4. Complete the `update()` function

Simulation

北京大学 PEKING UNIVERSITY

```python
def run_LIF():
    # 运行LIF模型

    group = LIF(1)
    runner = bp.DSRunner(group, monitors=['V'], inputs=('input', 22.))
    runner(200)  # 运行时长为200ms

    # 结果可视化
    fig, gs = bp.visualize.get_figure(1, 1, 4.5, 6)
    ax = fig.add_subplot(gs[0, 0])
    plt.plot(runner.mon.ts, runner.mon.V)
    plt.xlabel(r'$t$ (ms)')
    plt.ylabel(r'$V$ (mV)')
    ax.spines['top'].set_visible(False)
    ax.spines['right'].set_visible(False)
    plt.show()
```



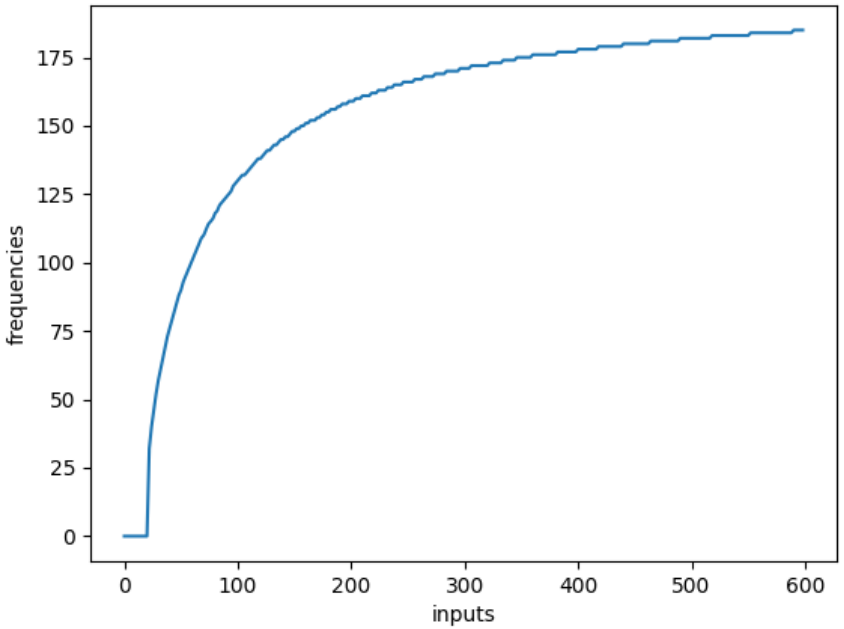
Input current & firing frequency

$$V(t) = V_{\text{reset}} + RI_{\text{c}}(1 - \mathrm{e}^{-\frac{t-t_0}{\tau}}).$$

$$T = -\tau \ln\left[1 - \frac{V_{\text{th}} - V_{\text{rest}}}{RI_{\text{c}}}\right]$$

$$f = \frac{1}{T + t_{\text{ref}}} = \frac{1}{t_{\text{ref}} - \tau \ln\left[1 - \frac{V_{\text{th}} - V_{\text{rest}}}{RI_{\text{c}}}\right]}$$

```
#输入与频率的关系-------------

current = bm.arange(0,600,2)
duration = 1000

LIF_neurons = LIF(current.shape[0])
runner_2 = bp.dyn.DSRunner(LIF_neurons,monitors=['spike'],inputs=('input',current),dt=0.01)

runner_2.run(duration)

freqs = runner_2.mon.spike.sum(axis=0) / (duration/1000)

plt.figure()
plt.plot(current,freqs)
plt.xlabel('inputs')
plt.ylabel('frequencies')
```



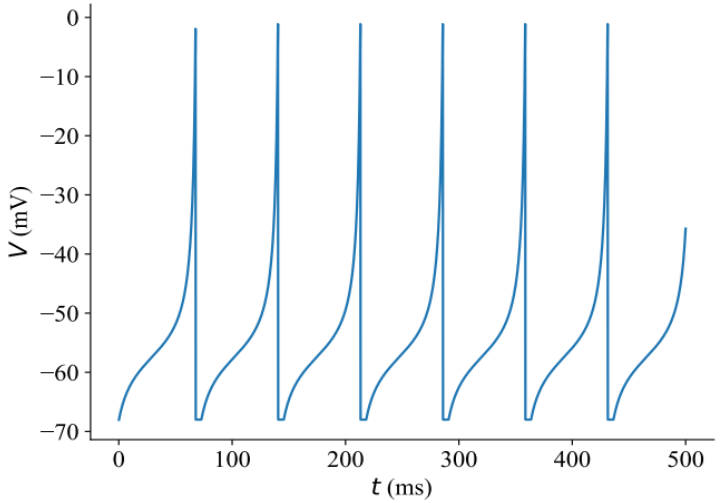
Other Univariate neuron models

- The Quadratic Integrate-and-Fire (QIF) model:

$$\tau \frac{\mathrm{d}V}{\mathrm{d}t} = a_0(V - V_{\mathrm{rest}})(V - V_{\mathrm{c}}) + RI(t)$$

$$\text{if } V > \theta, \quad V \leftarrow V_{\mathrm{reset}} \text{ last } t_{\mathrm{ref}}$$

```
def derivative(self, V, t, I):
  dVdt = (self.c * (V - self.V_rest) * (V - self.V_c) + self.R * I) / self.tau
  return dVdt
```

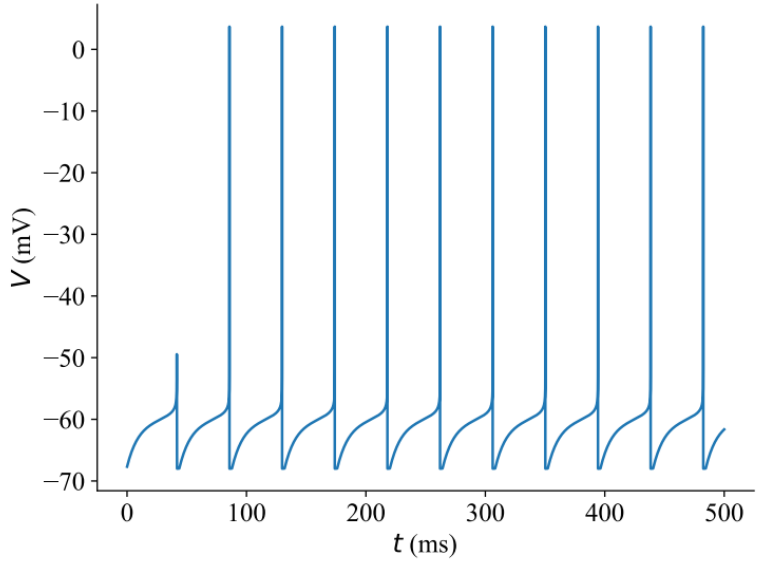


- The Exponential Integrate-and-Fire (ExpIF) model:

$$\tau \frac{\mathrm{d}V}{\mathrm{d}t} = -(V - V_{\mathrm{rest}}) + \Delta_T \mathrm{e}^{\frac{V - V_T}{\Delta_T}} + RI(t)$$

$$\text{if } V > \theta, \quad V \leftarrow V_{\mathrm{reset}} \text{ last } t_{\mathrm{ref}}$$

```
def derivative(self, V, t, I):
  exp_v = self.delta_T * bm.exp((V - self.V_T) / self.delta_T)
  dvdt = (- (V - self.V_rest) + exp_v + self.R * I) / self.tau
  return dvdt
```



北京大学
PEKING UNIVERSITY

02

AdEx neuron models programming

The AdEx neuron model

1.Define AdEx `class`

2. Initialization

   parameters

   variables

   integral function

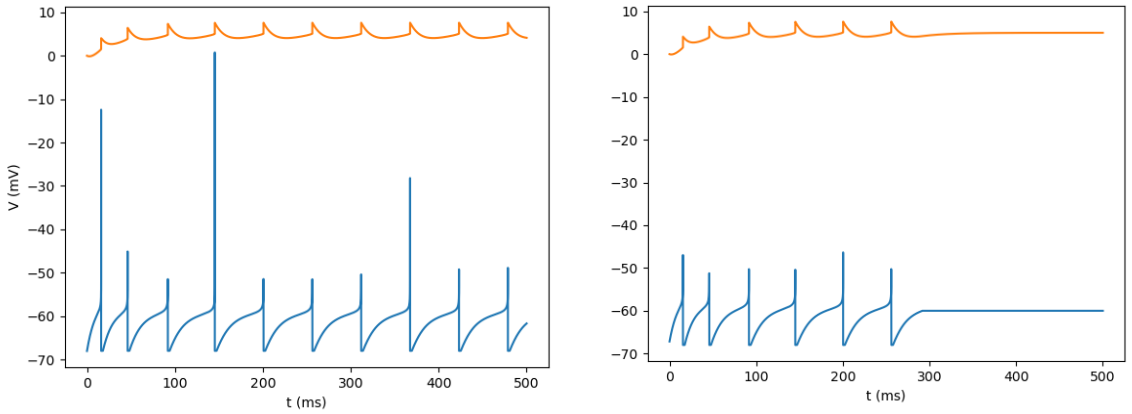
3. Define the derivative function

4. Complete the `update()` function

$$\tau_m \frac{\mathrm{d}V}{\mathrm{d}t} = -(V - V_{\mathrm{rest}}) + \Delta_T \mathrm{e}^{\frac{V - V_T}{\Delta_T}} - Rw + RI(t),$$

$$\tau_w \frac{\mathrm{d}w}{\mathrm{d}t} = a(V - V_{\mathrm{rest}}) - w + b\tau_w \sum_{t^{(f)}} \delta(t - t^{(f)})),$$

$$\text{if } V > V_{\mathrm{th}}, \quad V \leftarrow V_{\mathrm{reset}} \text{ last } t_{\mathrm{ref}}.$$



The AdEx neuron model

北京大学 PEKING UNIVERSITY

1.Define AdEx `class`

2. Initialization

- parameters
- variables
- integral function

```
class AdEx(bp.dyn.NeuDyn):
  def __init__(self, size,
               V_rest=-65., V_reset=-68., V_th=-30., V_T=-59.9, delta_T=3.48,
               a=1., b=1., R=1., tau=10., tau_w=30., tau_ref=0.,
               **kwargs):
    # 初始化父类
    super(AdEx, self).__init__(size=size, **kwargs)

    # 初始化参数
    self.V_rest = V_rest
    self.V_reset = V_reset
    self.V_th = V_th
    self.V_T = V_T
    self.delta_T = delta_T
    self.a = a
    self.b = b
    self.R = R
    self.tau = tau
    self.tau_w = tau_w

    self.tau_ref = tau_ref
```

The AdEx neuron model

北京大学 PEKING UNIVERSITY

1.Define AdEx `class`

2. Initialization

- parameters
- variables
- integral function

```
# 初始化变量
self.V = bm.Variable(bm.random.randn(self.num) - 65.)
self.w = bm.Variable(bm.zeros(self.num))
self.input = bm.Variable(bm.zeros(self.num))
self.t_last_spike = bm.Variable(bm.ones(self.num) * -1e7)  # 上一次脉冲发放时间
self.refractory = bm.Variable(bm.zeros(self.num, dtype=bool))  # 是否处于不应期
self.spike = bm.Variable(bm.zeros(self.num, dtype=bool))  # 脉冲发放状态

# 定义积分器
self.integral = bp.odeint(f=self.derivative, method='exp_auto')
```

```
# 定义积分器
self.integral = bp.odeint(f=self.derivative, method='exp_auto')
```

The AdEx neuron model

北京大学 PEKING UNIVERSITY

1.Define AdEx `class`

2. Initialization

parameters

variables

integral function

```
def dV(self, V, t, w, I):
  exp = self.delta_T * bm.exp((V - self.V_T) / self.delta_T)
  dVdt = (- V + self.V_rest + exp - self.R * w + self.R * I) / self.tau
  return dVdt

def dw(self, w, t, V):
  dwdt = (self.a * (V - self.V_rest) - w) / self.tau_w
  return dwdt

@property
def derivative(self):
  return bp.JointEq([self.dV, self.dw])
```

3. Define the derivative function

The AdEx neuron model

北京大学 PEKING UNIVERSITY

1.Define AdEx `class`

2. Initialization

   parameters

   variables

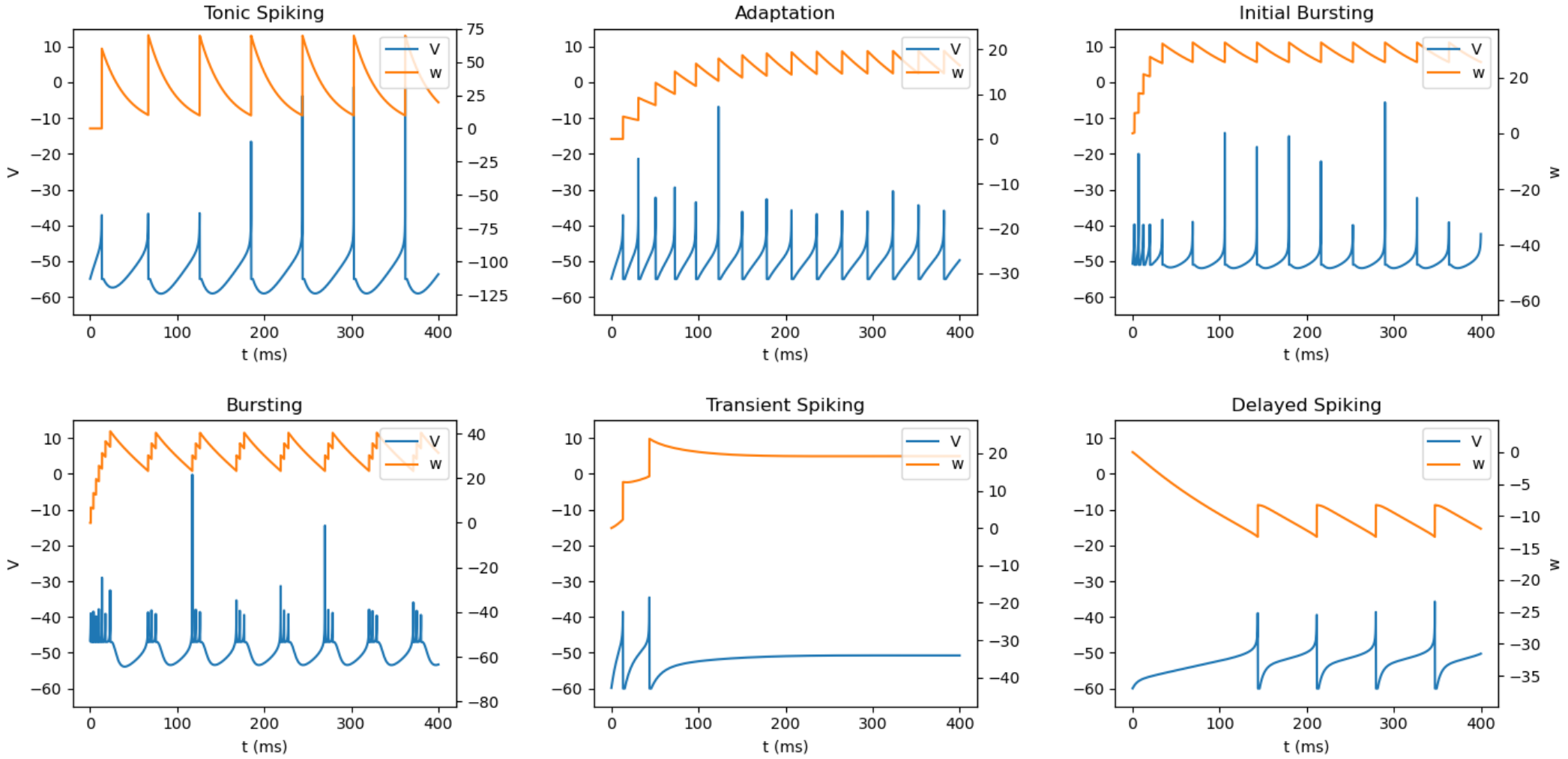
   integral function

```
def update(self):
  t, dt = bp.share['t'], bp.share['dt']
  V, w = self.integral(self.V.value, self.w.value, t, self.input, dt)
  refractory = (t - self.t_last_spike) <= self.tau_ref  # 判断神经元是否处于不应期
  V = bm.where(refractory, self.V, V)  # 若处于不应期，则返回原始膜电位self.V，否则返回更新后的膜电位V
  spike = V > self.V_th  # 将大于阈值的神经元标记为发放了脉冲
  self.spike[:] = spike  # 更新神经元脉冲发放状态
  self.t_last_spike[:] = bm.where(spike, t, self.t_last_spike)  # 更新最后一次脉冲发放时间
  self.V[:] = bm.where(spike, self.V_reset, V)  # 将发放了脉冲的神经元膜电位置为V_reset，其余不变
  self.w[:] = bm.where(spike, w + self.b, w)    # 更新自适应电流
  self.refractory[:] = bm.logical_or(refractory, spike)  # 更新神经元是否处于不应期
  self.input[:] = 0.  # 重置外界输入
```

3. Define the derivative function

4. Complete the `update()` function

Simulation

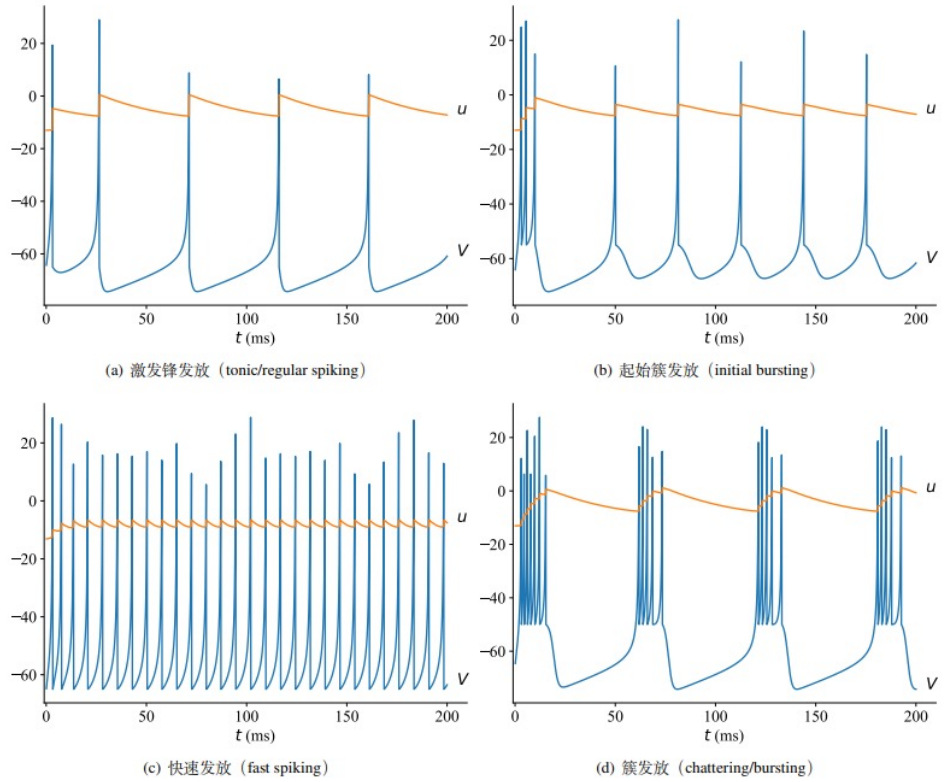

Tonic Spiking
Adaptation
Initial Bursting
Bursting
Transient Spiking
Delayed Spiking
V
w
t (ms)

Other multivariate neuron models

- The Izhikevich model:

$$\frac{dV}{dt} = 0.04V^2 + 5V + 140 - u + I$$
$$\frac{du}{dt} = a\,(bV - u)$$
$$\text{if } V > \theta, \quad V \leftarrow c,\ u \leftarrow u + d \ \text{ last } \ t_{\text{ref}}$$

```
def dV(self, V, t, u, I):
    dVdt = 0.04 * V * V + 5 * V + 140 - u + I
    return dVdt

 Brandon Zhang
def du(self, u, t, V):
    dudt = self.a * (self.b * V - u)
    return dudt
```



(a) 激发锋发放（tonic/regular spiking）　(b) 起始簇发放（initial bursting）

(c) 快速发放（fast spiking）　(d) 簇发放（chattering/bursting）

Other multivariate neuron models

北京大学 PEKING UNIVERSITY

- The Generalized Integrate-and-Fire (GIF) model:

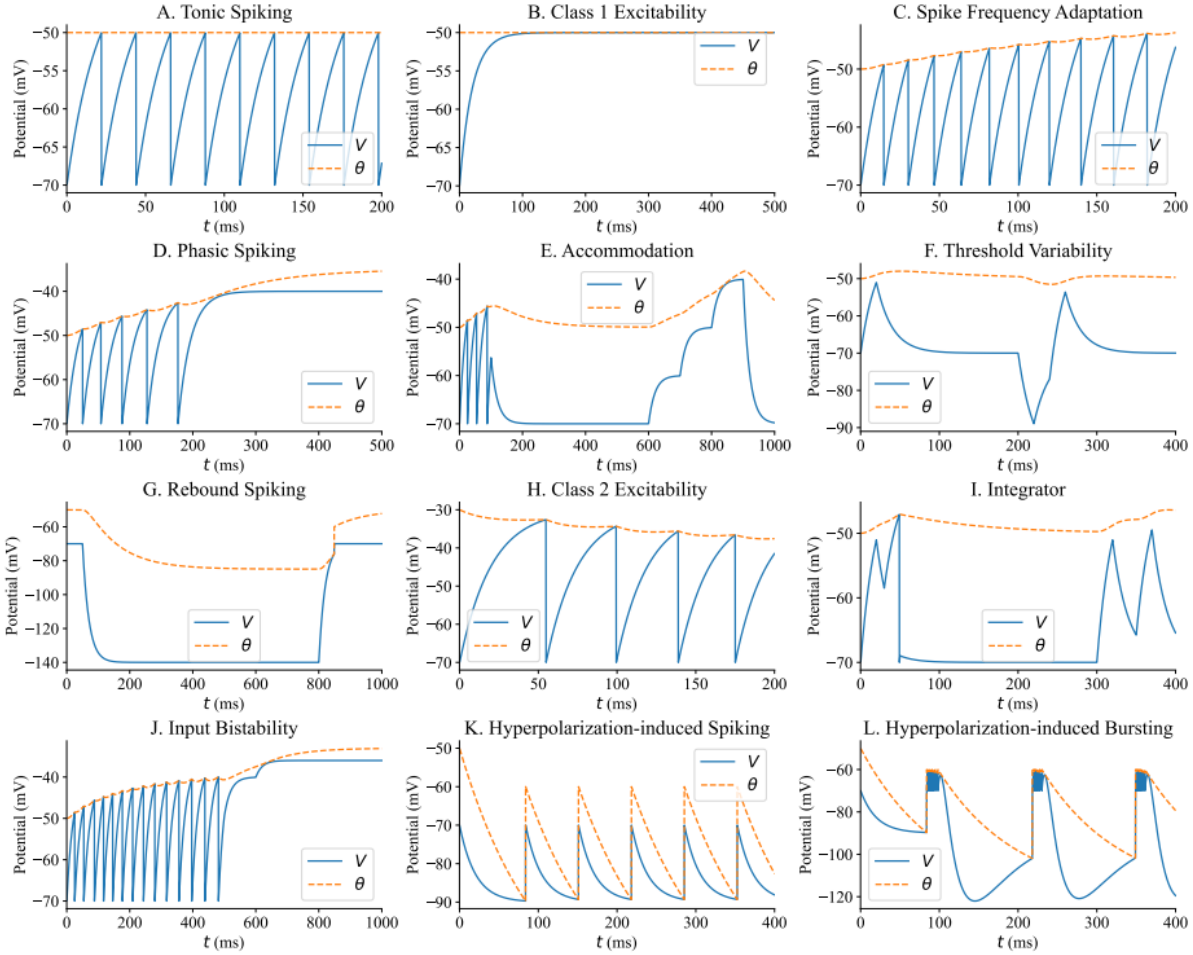
$$\tau \frac{\mathrm{d}V}{\mathrm{d}t} = -(V - V_{\mathrm{rest}}) + R\sum_j I_j + RI$$

$$\frac{\mathrm{d}\Theta}{\mathrm{d}t} = a(V - V_{\mathrm{rest}}) - b(\Theta - \Theta_\infty)$$

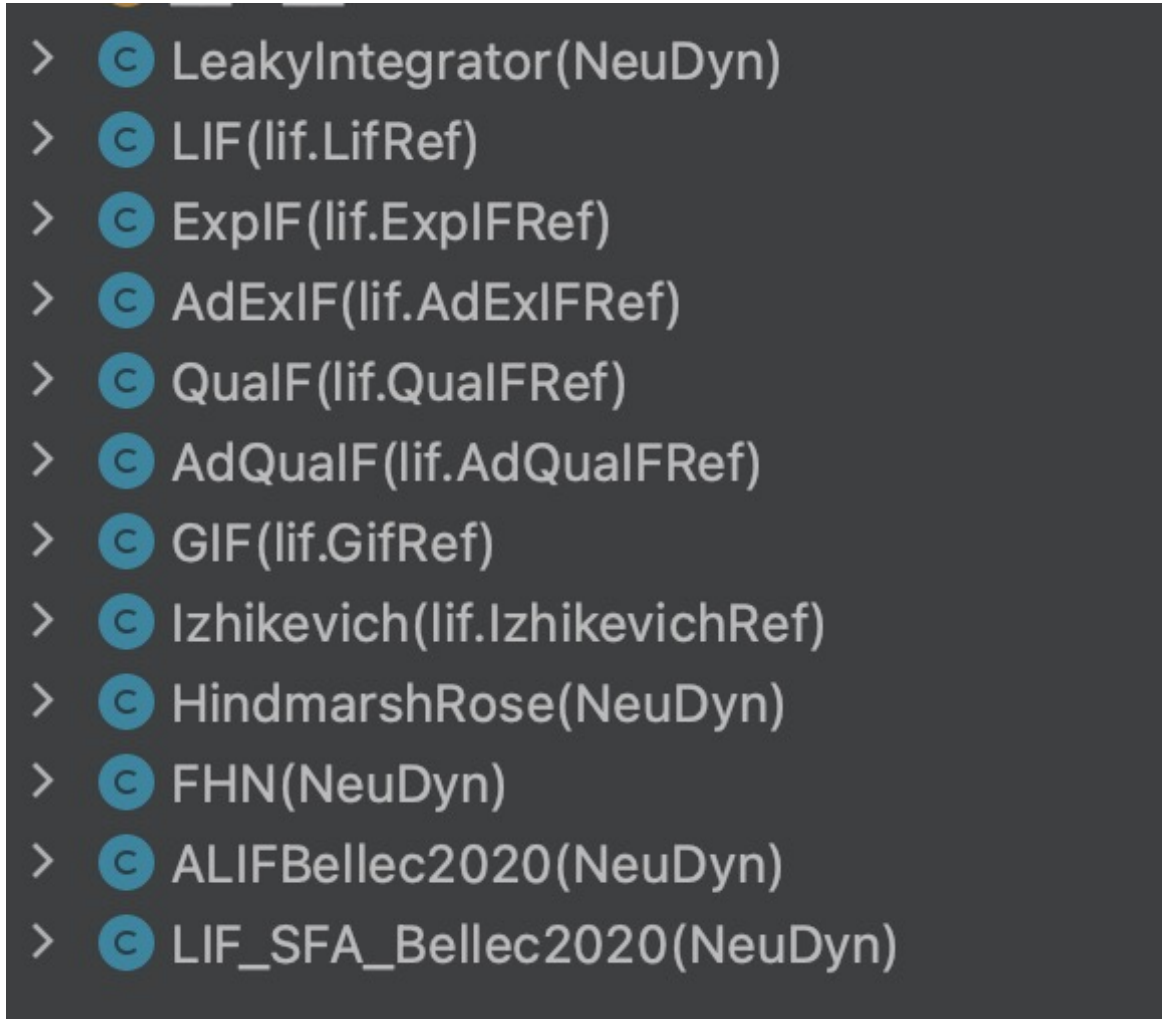
$$\frac{\mathrm{d}I_j}{\mathrm{d}t} = -k_j I_j, \quad j = 1, 2, \ldots, n$$

$$\text{if } V > \Theta, \quad I_j \leftarrow R_j I_j + A_j,\ V \leftarrow V_{\mathrm{reset}},\ \Theta \leftarrow max(\Theta_{\mathrm{reset}}, \Theta)$$

```
def dI1(self, I1, t):
  return - self.k1 * I1

Brandon Zhang
def dI2(self, I2, t):
  return - self.k2 * I2

Brandon Zhang
def dVth(self, V_th, t, V):
  return self.a * (V - self.V_rest) - self.b * (V_th - self.V_th_inf)

Brandon Zhang *
def dV(self, V, t, I1, I2, I):
  return (- (V - self.V_rest) + self.R * (I + I1 + I2)) / self.tau
```



Built-in reduced neuron models



北京大学
PEKING UNIVERSITY

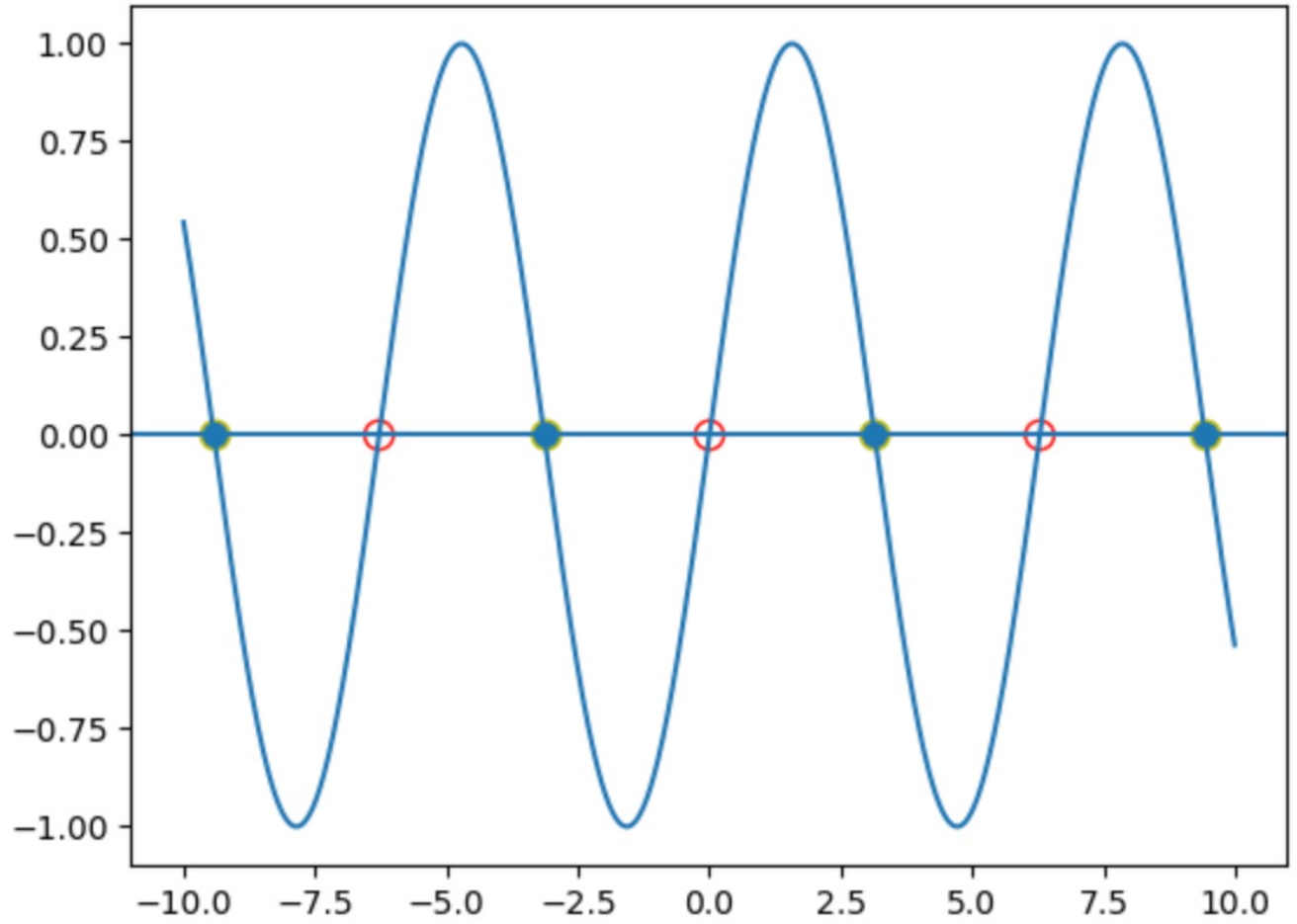
03

Dynamic analysis: phase-plane analysis

Simple case

$$\frac{dx}{dt} = \sin(x) + I,$$

```
@bp.odeint
def int_x(x, t, Iext):
    return bp.math.sin(x) + Iext
```

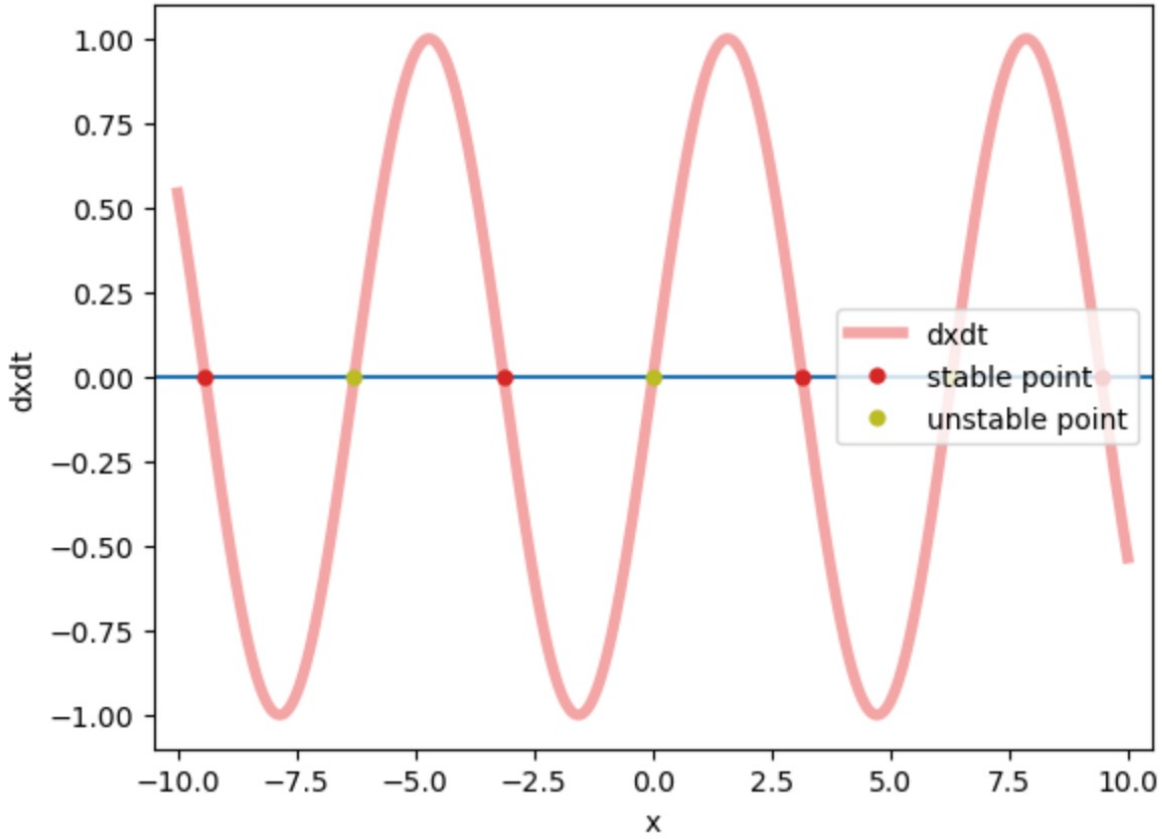


Simple case

$$\frac{dx}{dt} = \sin(x) + I,$$

```
I am creating the vector field ...
I am searching fixed points ...
Fixed point #1 at x=-9.424777960769386 is a stable point.
Fixed point #2 at x=-6.283185307179586 is a unstable point.
Fixed point #3 at x=-3.1415926535897984 is a stable point.
Fixed point #4 at x=3.552755127361717e-18 is a unstable point.
Fixed point #5 at x=3.1415926535897984 is a stable point.
Fixed point #6 at x=6.283185307179586 is a unstable point.
Fixed point #7 at x=9.424777960769386 is a stable point.
```

```
pp = bp.analysis.PhasePlane1D(
  model=int_x,
  target_vars={'x': [-10, 10]},
  pars_update={'Iext': 0.},
  resolutions={'x': 0.01}
)
pp.plot_vector_field()
pp.plot_fixed_point(show=True)
```



Phase plane analysis

北京大学 PEKING UNIVERSITY

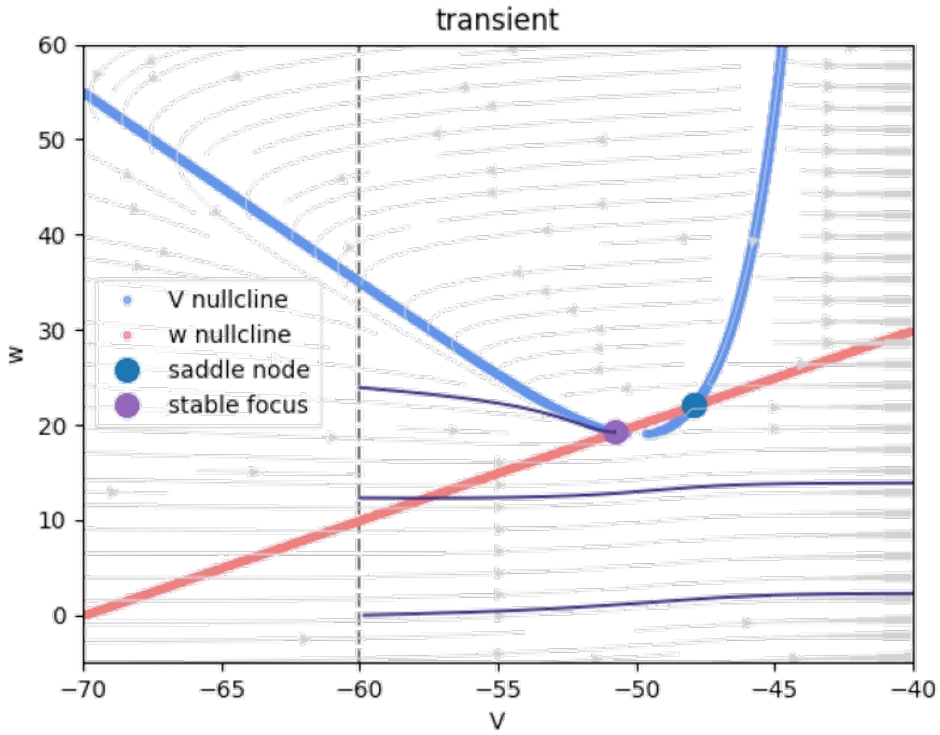
- **Nullcline**: The zero-growth isoclines, such as $g(x, y) = 0$ and $g(x, y) = 0$.
- **Fixed points**: The equilibrium points of the system, which are located at all the nullclines intersect.
- **Vector field**: The vector field of the system.
- **Limit cycles**: The limit cycles.
- **Trajectories**: A simulation trajectory with the given initial values.

Phase plane analysis for AdEx

北京大学 PEKING UNIVERSITY

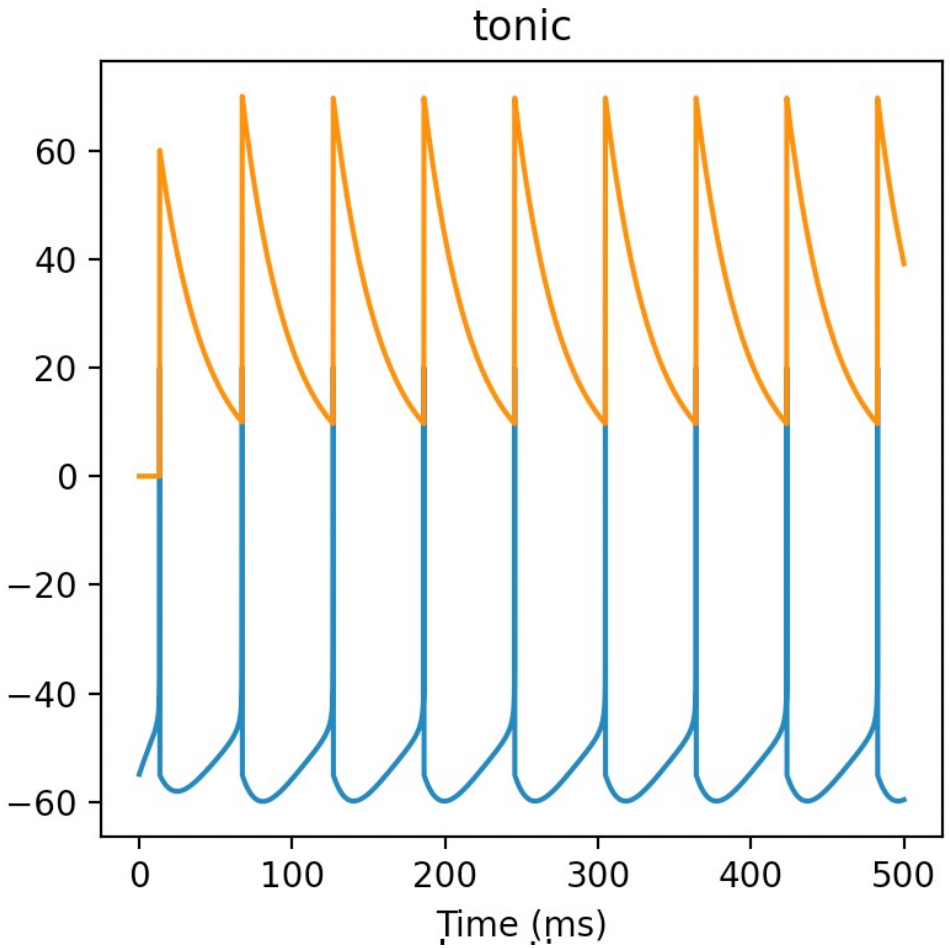
- 2-D phase plane analysis

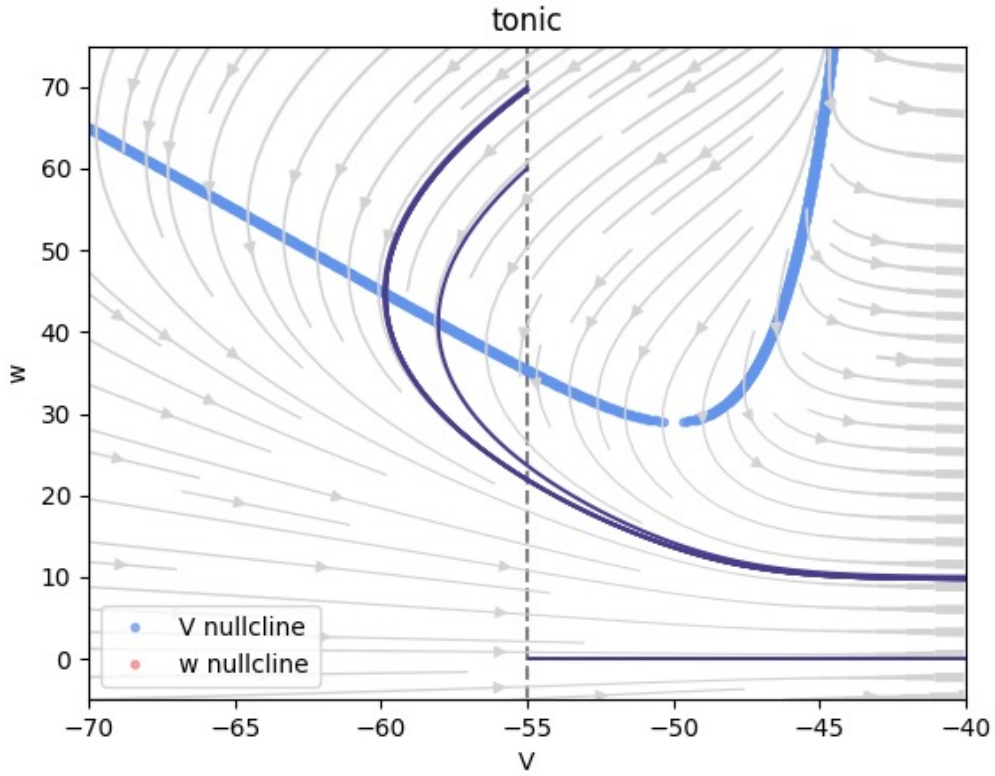
```python
def phaseplane(group, title, v_range=None, w_range=None, Iext=65., duration=400):
    #bm.clear_buffer_memory()
    plt.figure()
    v_range = [-70., -40.] if not v_range else v_range
    w_range = [-10., 50.] if not w_range else w_range

    # 使用BrainPy中的相平面分析工具
    phase_plane_analyzer = bp.analysis.PhasePlane2D(
        model=group,
        target_vars={'V': v_range ,'w': w_range}, # 待分析变量
        pars_update={'Iext': Iext}, # 需要更新的变量
        resolutions=0.05)

    # 画出V, w的零增长曲线
    phase_plane_analyzer.plot_nullcline()

    # 画出固定点
    phase_plane_analyzer.plot_fixed_point()
    # 画出向量场
    phase_plane_analyzer.plot_vector_field(plot_style=dict(color='lightgrey', density=1.))

    #phase_plane_analyzer.plot_trajectory(initials={'V':[group.V_reset],'w':[0]},duration=400,dt=0.01)

    runner = bp.dyn.DSRunner(group, monitors=['V', 'w', 'spike'], inputs=('input',Iext))
    runner.run(duration)

    spike = runner.mon.spike.squeeze()
    s_idx = bm.where(spike)[0] # 找到所有发放动作电位对应的index
    s_idx = bm.concatenate([bm.array([0]), s_idx, bm.array([len(spike) - 1])]) # 加上起始点和终止点的index
    # 分段画出V, w的变化轨迹
    for i in range(len(s_idx) - 1):
        plt.plot(runner.mon.V[s_idx[i]: s_idx[i + 1]], runner.mon.w[s_idx[i]: s_idx[i + 1]], color='darkslateblue')

    # 画出虚线 x = V_reset
    plt.plot([group.V_reset, group.V_reset], w_range, '--', color='grey', zorder=-1)
```



Phase plane analysis for AdEx

- Tonic firing

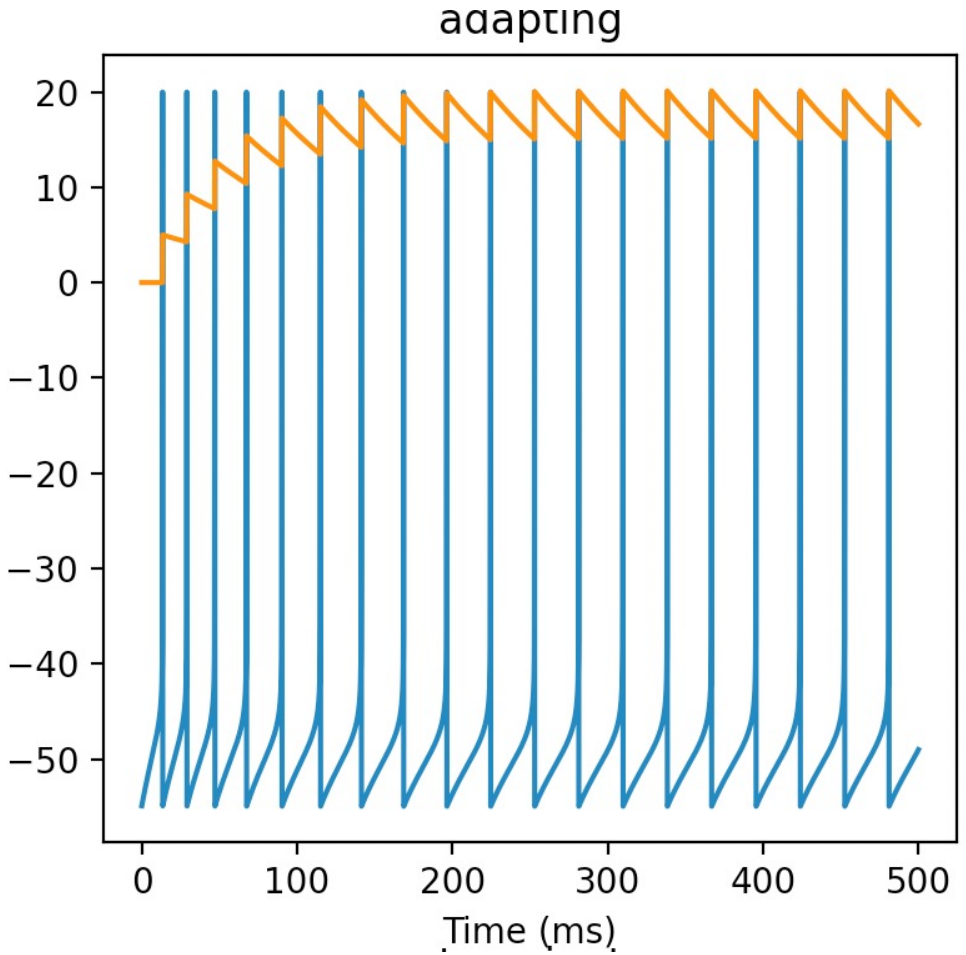


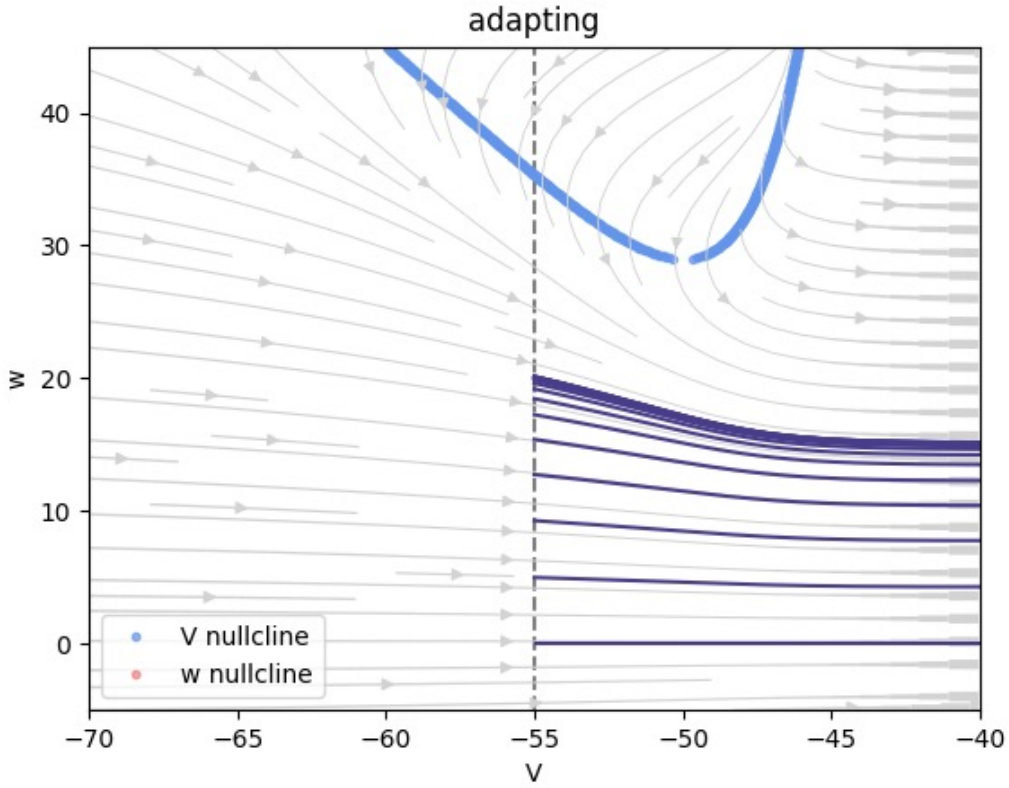


Phase plane analysis for AdEx

北京大学 PEKING UNIVERSITY

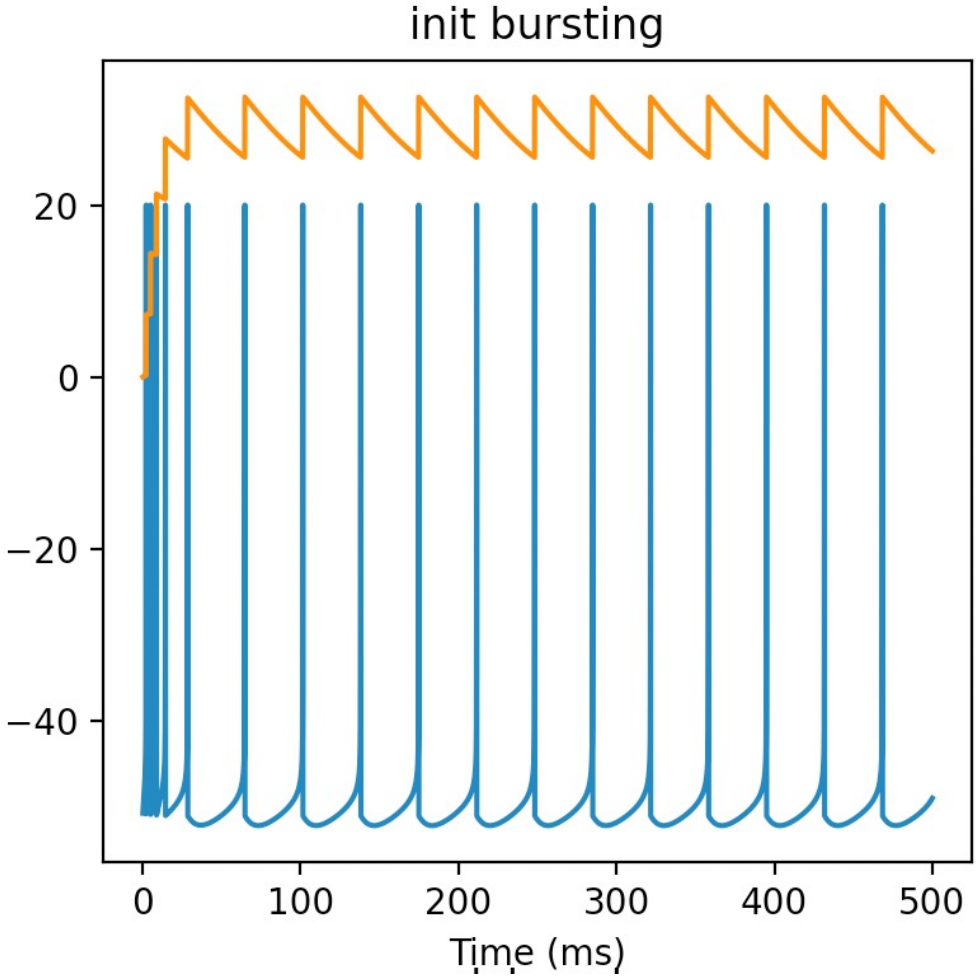
- Adapting

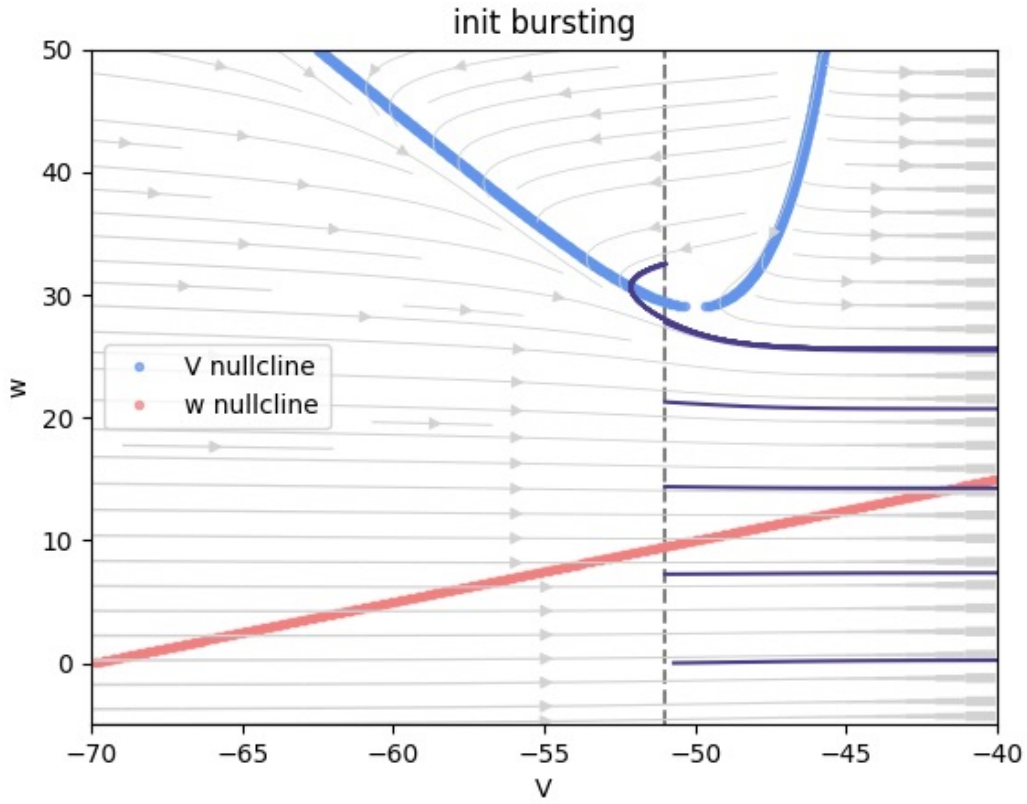




Phase plane analysis for AdEx

- Init bursting

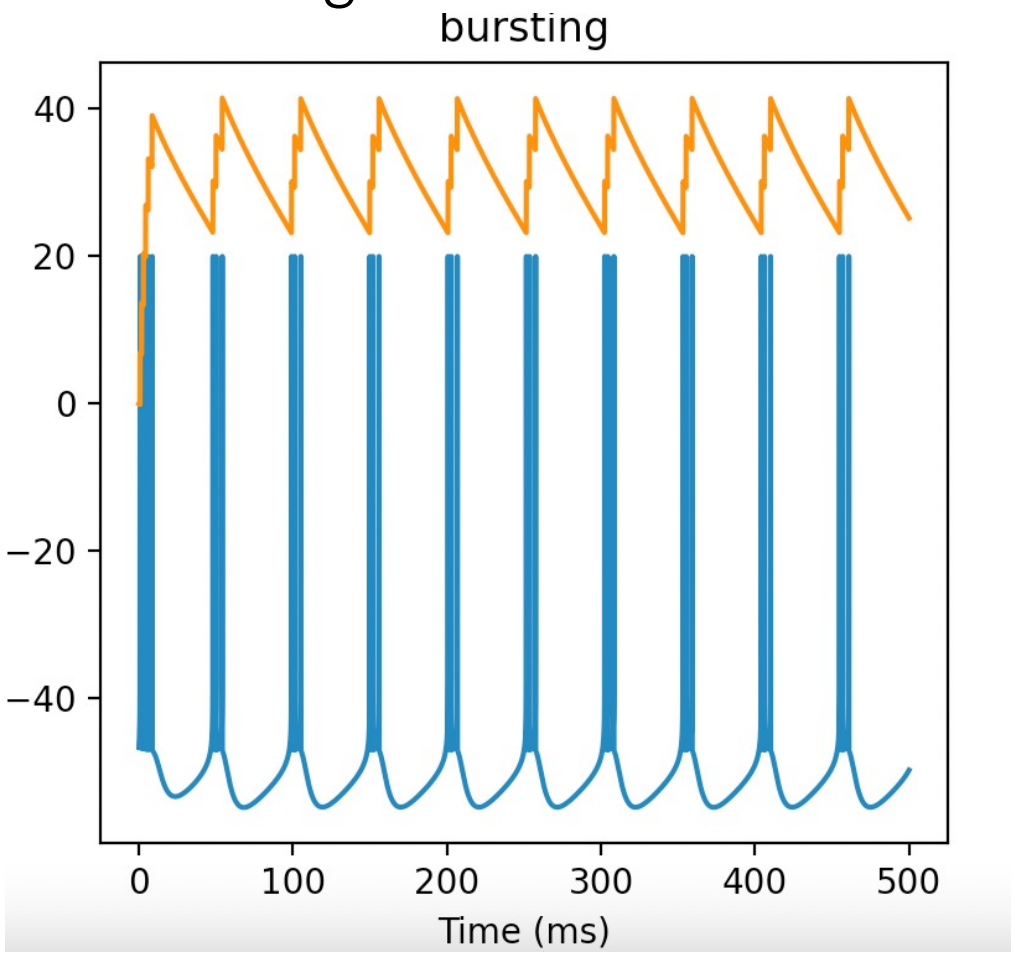


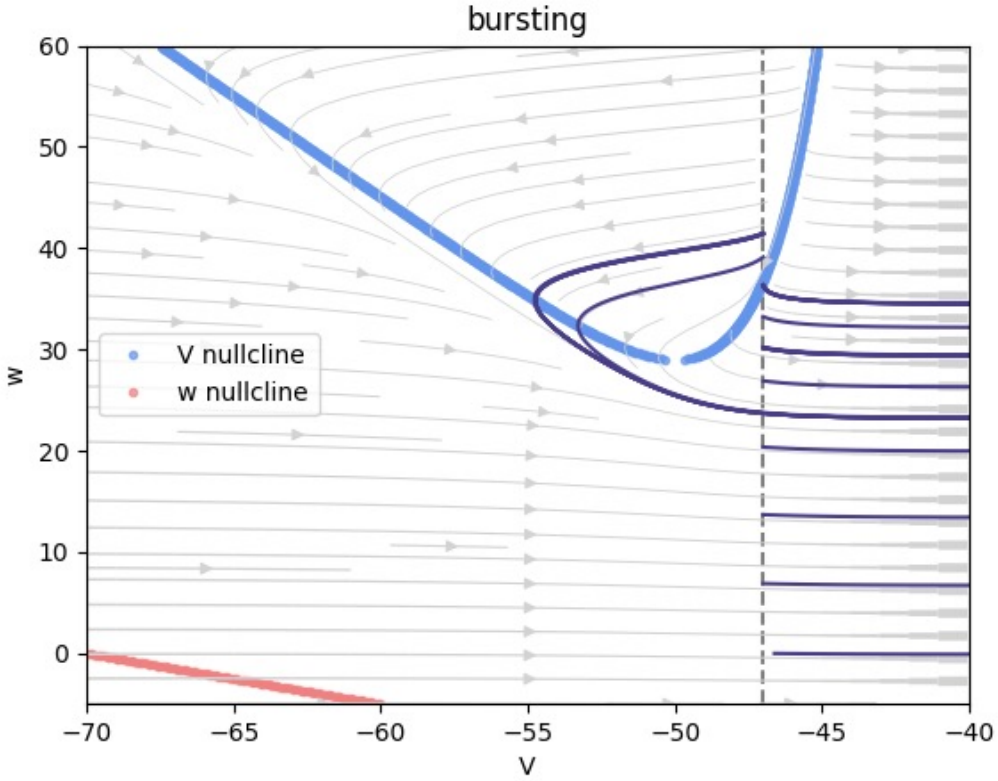


Phase plane analysis for AdEx

北京大学 PEKING UNIVERSITY

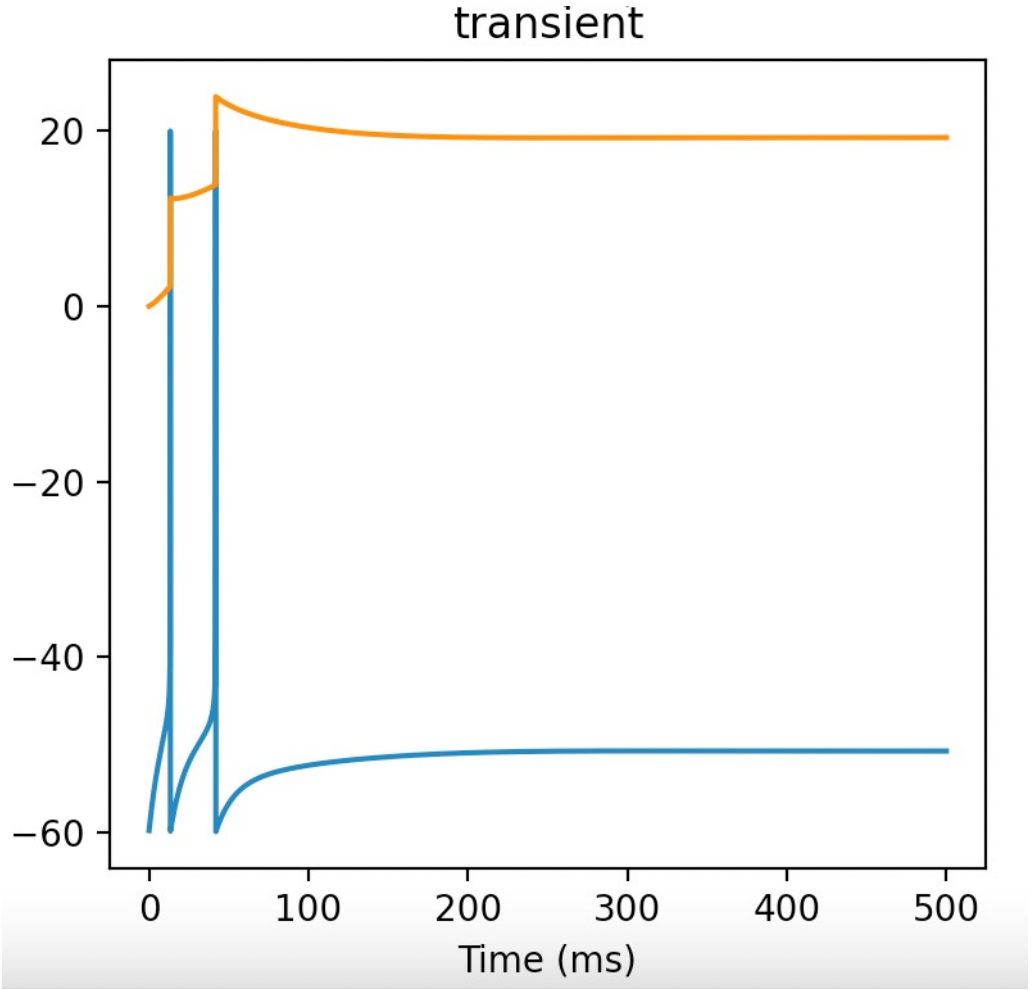
- bursting

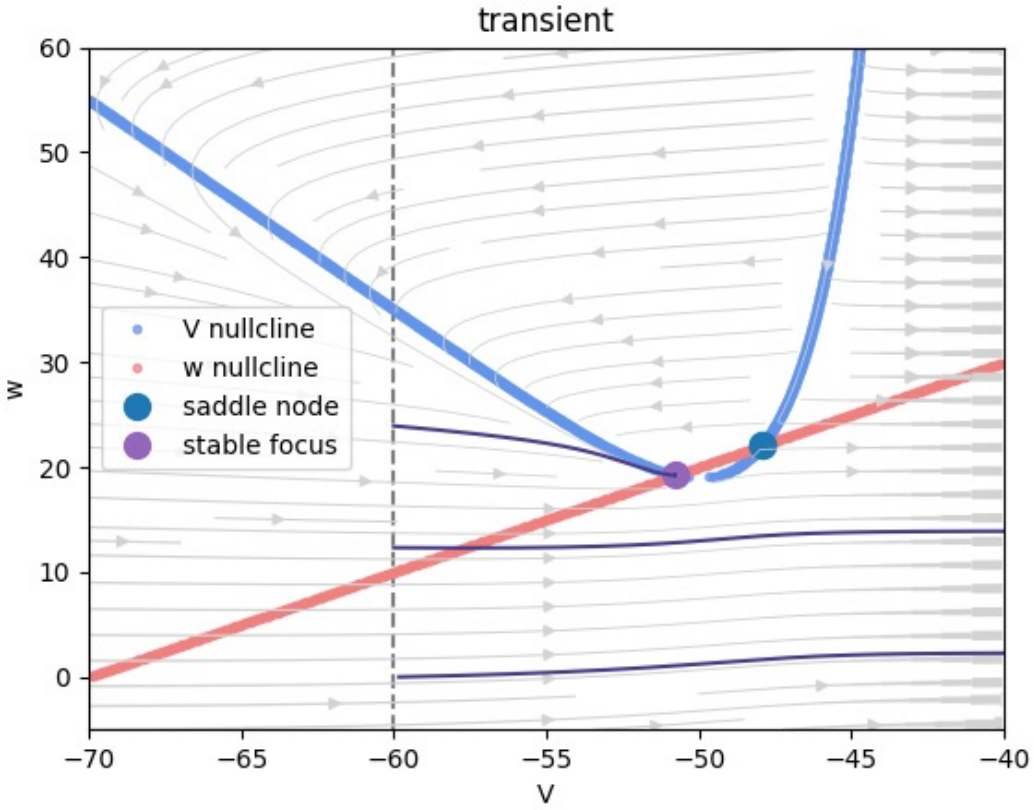




Phase plane analysis for AdEx

- transient

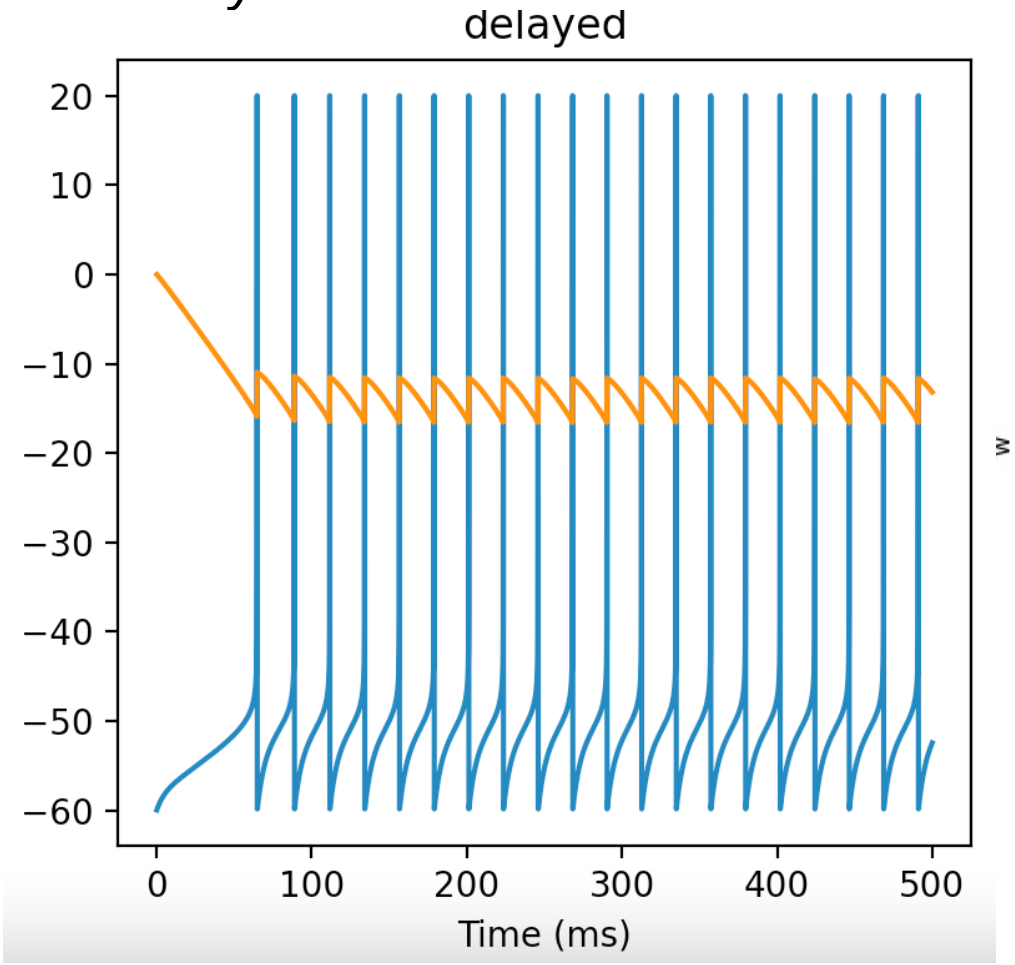


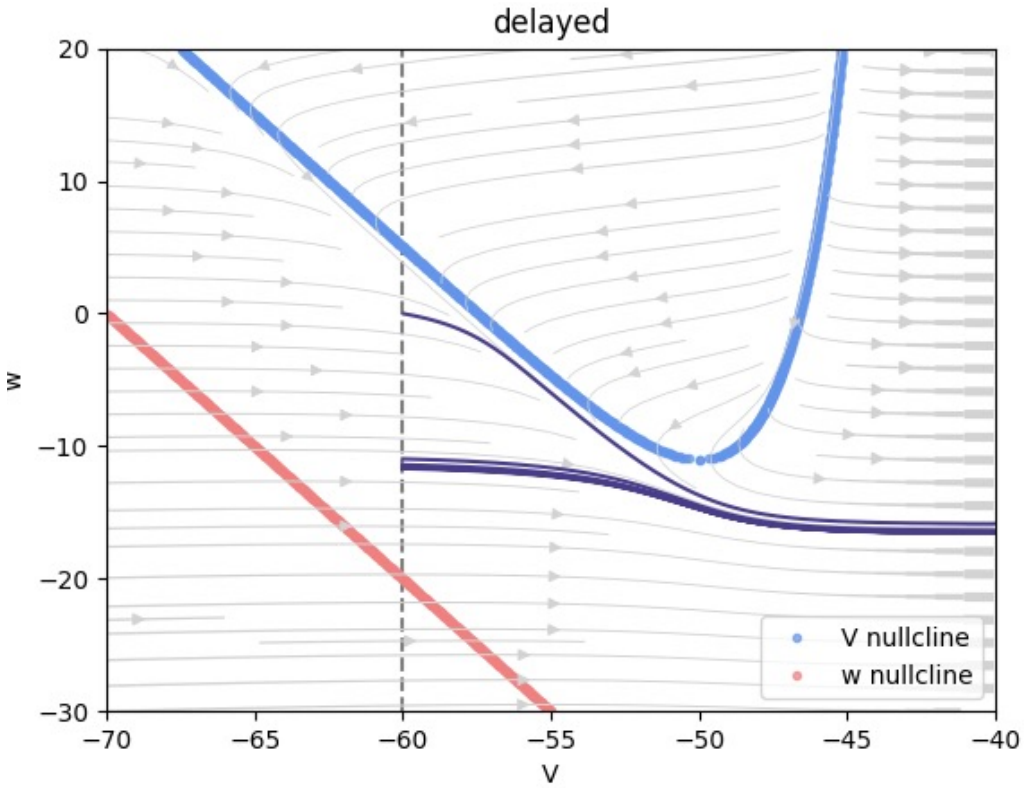


Phase plane analysis for AdEx

北京大学 PEKING UNIVERSITY

- delayed





北京大学
PEKING UNIVERSITY

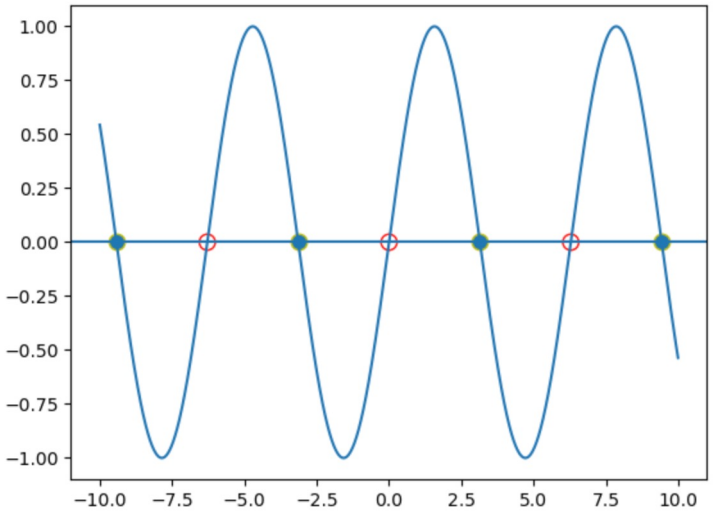
04

Dynamic analysis: bifurcation analysis

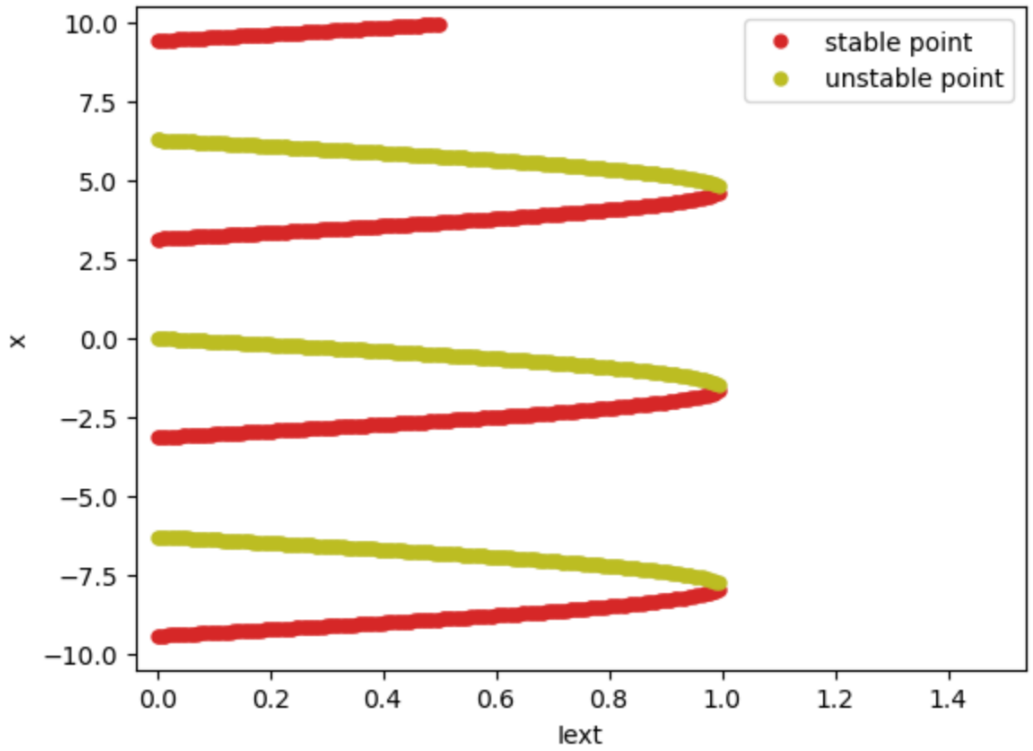
Simple case

$$\frac{dx}{dt} = \sin(x) + I,$$

```
bif = bp.analysis.Bifurcation1D(
    model=int_x,
    target_vars={'x': [-10, 10]},
    target_pars={'Iext': [0., 1.5]},
    resolutions={'Iext': 0.005, 'x': 0.05}
)
bif.plot_bifurcation(show=True)
```




```
I am making bifurcation analysis ...
```



Thanks!

北京大学
PEKING UNIVERSITY