Explicit Euler Method

Ahmed Salem

The ODE used

$\frac{dv}{dt} = k(v - v_r)(v - v_t) - w + In$

$\frac{dw}{dt} = a[b(v - v_r) - w$

Variables and Parameters

- C: **Membrane capacitance** (value: $C = 100$, units: typically pF or scaled).
- v: **Neuron membrane potential** (in millivolts, mV).
- w: **Recovery current** (in picoamperes, pA, or scaled units).
- t: **Time** (in milliseconds, ms).
- vr: **Resting membrane potential** (value: $vr = -60mV$)
- k: **Scaling factor for the quadratic term** (value: $k = 0.7$).
- vt: **Threshold membrane potential** (value: $vt = -40mV$)
- In: **Input current** (values: $In = 0, for, t < 101ms, In = 70, for, t \geq 101ms$, units: pA)
- a: **Time scale of recovery** (value:$a = 0.03$, units: ms^-1)
- b: **Sensitivity of recovery to membrane potential** (value: $b = -2$, dimensionless or scaled).

Initial conditions

$v(t = 0) = v0$
$w(t = 0) = w0$
Where $v0$ and $w0$ are constants chosen based on the desired initial state of the neuron (e.g., resting potential).

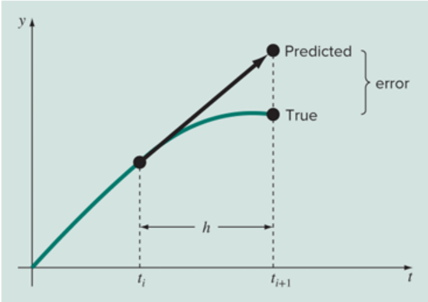
Reset mechanism

when the membrane potentials reaches a threshold $v_{\text{peak}} = 35mV$ we

- set $v(t) = -50mV$
- set $w(t) = w(t) + 100$

Euler Method

FIGURE 22.1 Euler's method.



- To calculate the y_{n+1} we use $y[i+1] = y[i] + h \cdot f(t_i, y)$ where where $f(t_i, y_i)$ is the differential equation evaluated at t_i and y_i

Advantages of Euler's method

- Simplicity and Ease of Implementation
- Low Computational Cost per Step
- Intuitive and Transparent
- Suitable for Initial Testing

Disadvantages of Euler's method

- Low Accuracy: as its a first-order method O(h)
- Unsuitable with stiff or rapidly changing systems : as the dynamic neuron model we are using shoes sharp variations in solutions and derivatives

---

Solving Systems of ODE

- Apply the method to each dependent variable simultaneously
  so the system will look like this

$$\frac{d}{dt}\begin{bmatrix} v \\ w \end{bmatrix} = \begin{bmatrix} \frac{k(v-v_r)(v-v_t)-w+\mathrm{In}}{C} \\ a\left[b(v-v_r)-w\right] \end{bmatrix} = f(t, \begin{bmatrix} v \\ w \end{bmatrix})$$

  we will use initial condition of
- $v(0) = vr = -60$
- $w(0) = 0.$

the Flow of the solution

first we will initialize the parameters and variables then for each step (In is an array where $In = 0, for, i < 101, and, In = 70, for, i >= 101$) there will be a function that calculates the derivative for both dependent variables

- $f_1(t, v, w) = \frac{k(v-v_r)(v-v_t)-w+\mathrm{In}[i]}{C}$
- $f_2(t, v, w) = a\left[b(v-v_r)-w\right]$
  and after computing the derivative we will calculate y_{n+1} using:
- $v[i+1] = v[i] + h \cdot f_1(t_i, v[i], w[i])$
- $w[i+1] = w[i] + h \cdot f_2(t_i, v[i], w[i])$

The flow of the program

1. setup the parameters
2. chose the initial values
3. in the main body (the for loop) we first call $euler$
4. in the $euler$ we then call $neuron$ to calculate the derivatives
5. use the calculated derivatives to calculate the next y
6. return the calculated y to the main body and save it in a numpy array
7. at the end of the for loop check for spike
    1. if true , reset
    2. else, continue
8. rinse and repeat

Code

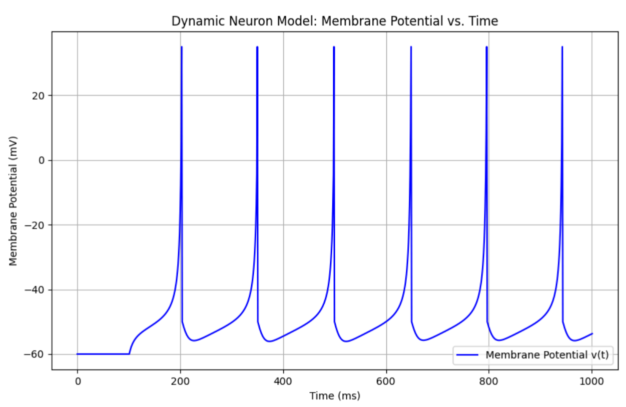
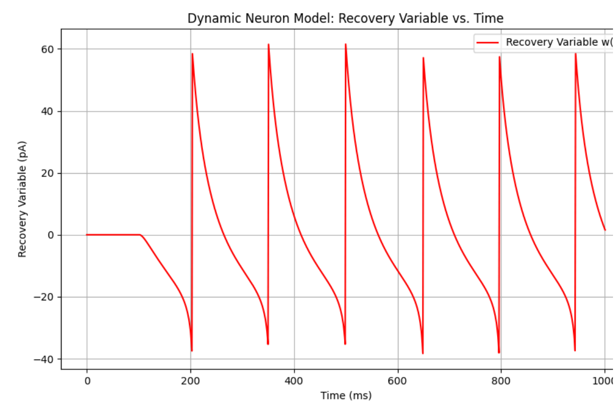
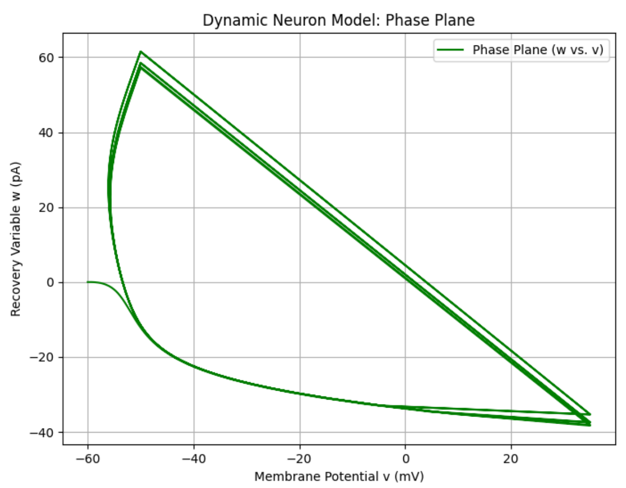
```
import numpy as np
import matplotlib.pyplot as plt
import time
#-------------------------
## Para

C=100
vr = -60
vt = -40
k = 0.7
an = 0.03
bn = -2
cn =-50
dn =100
vpeak = 35
ncall = 0
nout = 1000
h= 1
#---------------
In = np.zeros(nout+2)
In[101:] = 70
v = np.zeros(nout+2)
w = np.zeros(nout+2)
v[0]=vr
w[0]=0
t = np.arange(0,nout+2,h)
###-------------------
def neuron(t,y):

    global ncall

    v = y[0]

    w = y[1]

    dvbydt = (k*(v - vr) * (v - vt) - w + In[i]) / C

    dwbydt = an * (bn * (v - vr) - w)
```

```python
    ncall += 1

    return np.array([dvbydt, dwbydt])
###---------------
def euler(h,t,y):

    deriv = neuron(t,y)

    y = y + (deriv*h)

    return y

#----------

def get_time():

    return time.time() * 1000

###-----------------

start_time = get_time()

for i in range(0,nout+1):

    y = np.array([v[i],w[i]])

    yout=euler(h,t[i],y)

    v[i+1]=yout[0]

    w[i+1]=yout[1]

    if(v[i+1]>=vpeak):

        v[i]=vpeak

        v[i+1]=cn

        w[i+1]=w[i+1]+dn

###--------------------
end_time = get_time()
elapsed_time = end_time - start_time
print(f"elapsed time is : {elapsed_time}")
# Plotting v(t)
plt.figure(figsize=(10, 6))
plt.plot(t, v, 'b-', label='Membrane Potential v(t)')
plt.xlabel('Time (ms)')
plt.ylabel('Membrane Potential (mV)')
plt.title('Dynamic Neuron Model: Membrane Potential vs. Time')
plt.grid(True)
plt.legend()
plt.show()
# Plotting w(t)
plt.figure(figsize=(10, 6))
plt.plot(t, w, 'r-', label='Recovery Variable w(t)')
plt.xlabel('Time (ms)')
```

```
plt.ylabel('Recovery Variable (pA)')
plt.title('Dynamic Neuron Model: Recovery Variable vs. Time')
plt.grid(True)
plt.legend()
plt.show()
# Plotting w vs. v (Phase Plane)
plt.figure(figsize=(8, 6))
plt.plot(v, w, 'g-', label='Phase Plane (w vs. v)')
plt.xlabel('Membrane Potential v (mV)')
plt.ylabel('Recovery Variable w (pA)')
plt.title('Dynamic Neuron Model: Phase Plane')
plt.grid(True)
plt.legend()
plt.show()
print(ncall)
print(f"\nComparison with Table 4.3a:")
print(f"t=0.0:   v[0] = {v[0]:.4f}, w[0] = {w[0]:.4f}")
print(f"t=250.0: v[250] = {v[251]:.4f}, w[250] = {w[251]:.4f}")
print(f"t=500.0: v[500] = {v[501]:.4f}, w[500] = {w[501]:.4f}")
print(f"t=750.0: v[750] = {v[751]:.4f}, w[750] = {w[751]:.4f}")
print(f"t=1000.0: v[1000] = {v[1001]:.4f}, w[1000] = {w[1001]:.4f}")
```

Results

- Time take for the method to finish: **5.576904296875**

V versus T	W versus T
	
	

Reference values	Code Values
(see below)	(see below)

Reference values

```
     t          v          w
   0.0   -60.0000     0.0000
 250.0   -54.4819     6.2834
 500.0   -50.6154    59.0910
 750.0   -49.5530   -12.4763
1000.0   -53.6973     1.5649

ncall =  1000
```

Code Values

```
Comparison with Table 4.3a:
t=0.0:   v[0] = -60.0000, w[0] = 0.0000
t=250.0: v[250] = -54.4819, w[250] = 6.2834
t=500.0: v[500] = -50.6154, w[500] = 59.0910
t=750.0: v[750] = -49.5530, w[750] = -12.4763
t=1000.0: v[1000] = -53.6973, w[1000] = 1.5649
```