Heun’s Method(Modified Euler Method) Report

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 $w(t) = w(t) + 100$

Heun’s Method

- its a second-order numerical method
- it incorporates a predictor-corrector approach which gives higher accuracy with an error of order $O(h^2)$
- less complex than RK4

How to Use

- Heun’s Method is designed to solve a first-order ODE of the form:
  $\frac{dy}{dt} = f(t, y), \quad y(t_0) = y_0$
  the flow of the method is to
- Compute an initial estimate using the Euler method (predictor step)
- Refine the estimate by averaging the derivatives at the current and predicated points

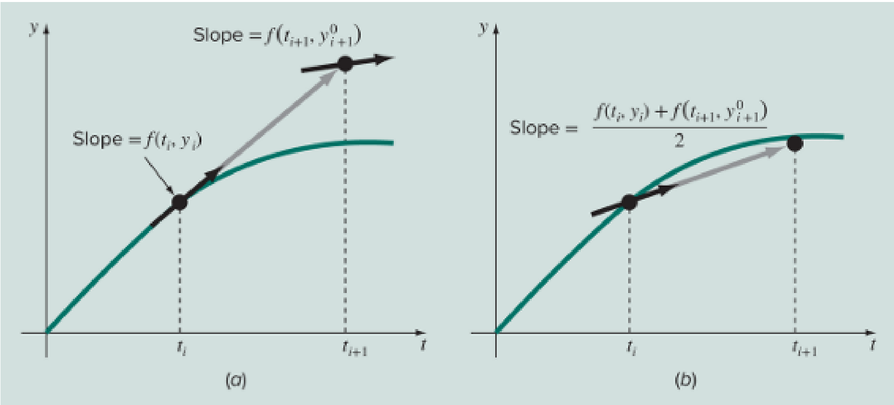
Formula of Heun's Method

$y_{n+1} = y_n + \frac{h}{2}[f(t_n, y_n) + f(t_{n+1}, \tilde{y}_{n+1})]$
where :

- y_n is the solution at time t_n
- h is the step size
- $f(t_n, y_n)$ is the derivative at the current point
- $\tilde{y}_{n+1}$ is the predicted solution at t_{n+1} which is calculated by
  $\tilde{y}_{n+1} = y_n + h \cdot f(t_n, y_n)$
- $f(t_{n+1}, \tilde{y}_{n+1})$ is the derivative at the predicted point.

So basically you first apply the Euler Method to predict where the next y_b will be then you use the (y_b, t_{n+1}) and the (y, t_n) to make get the slope for both of them and then we use those two slopes $f(t_n, y_n)$,$f(t_{n+1}, \tilde{y}_{n+1})$ to get an average estimation of where would the next y really be(y_{n+1}).

FIGURE 22.4 Graphical depiction of Heun's method. (*a*) Predictor and (*b*) corrector.



Solving Systems of ODE

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

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

  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

- $f_1(t, v, w) = \frac{k(v-v_r)(v-v_t)-w+\text{In}[i]}{C}$
- $f_2(t, v, w) = a\left[b(v-v_r)-w\right]$
  and after computing the derivative we will calculate the predictor for each dependent variable
- $\tilde{v}[i+1] = v[i] + h \cdot f_1(t_i, v[i], w[i])$
- $\tilde{w}[i+1] = w[i] + h \cdot f_2(t_i, v[i], w[i])$
  then calculate the corrector for each dependent variable (which corresponds to the solution at the next time step)

- $v[i+1] = v[i] + \frac{h}{2}[f_1(t_i, v[i], w[i]) + f_1(t_{i+1}, \tilde{v}[i+1], \tilde{w}[i+1])]$
- $w[i+1] = w[i] + \frac{h}{2}[f_2(t_i, v[i], w[i]) + f_2(t_{i+1}, \tilde{v}[i+1], \tilde{w}[i+1])]$
  and at the end check for spike and reset if needed

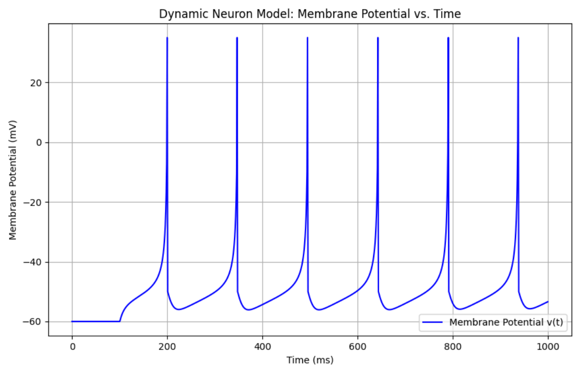
The Python Code

```
import numpy as np

import matplotlib.pyplot as plt

"""

Here we intialize the parameters

"""

C = 100

k = 0.7

vr = -60

vt = -40

a = 0.03

b = -2

c = -50

d = 100

vpeak = 35

h = 1

In = np.zeros(1001)

In[101:]=70

print(In)

v= np.zeros(1001)

w = np.zeros(1001)

##Initial Conditions

v[0] = vr

w[0] = 0



def f(i, y):

    """

    this method is used to calculate the derivative of dependent variable
```

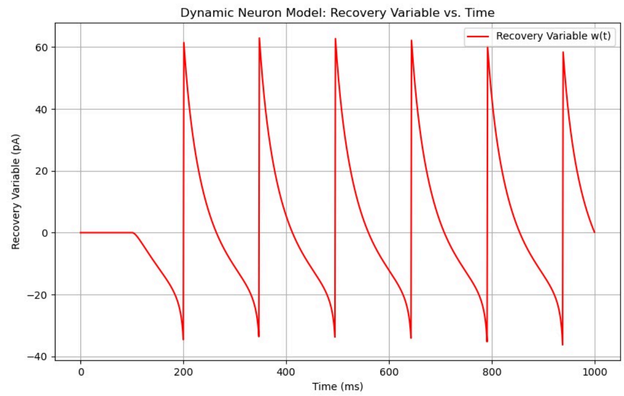
```
the methods gets an array of values for v, and w as an inputs and returns an array of the
calculated derivatives for each variable(numpy array)
    """

    v = y[0]

    w = y[1]

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

    dwbydt = a * (b * (v - vr) - w)

    return np.array([dvbydt, dwbydt])

### Heun's Method Implementation

for i in range(0, 1000):

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

    derivatives = np.zeros(2)

    derivatives = f(i,y)

    y_predicted = y + h*derivatives

    y_corrected = y + (h/2)*(f(i,y)+f(i+1,y_predicted))

    v[i+1] = y_corrected[0]

    w[i+1] = y_corrected[1]

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

        v[i] = vpeak

        v[i + 1] = c

        w[i + 1] = w[i + 1] + d



t = np.arange(0, 1001, h)

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()
```

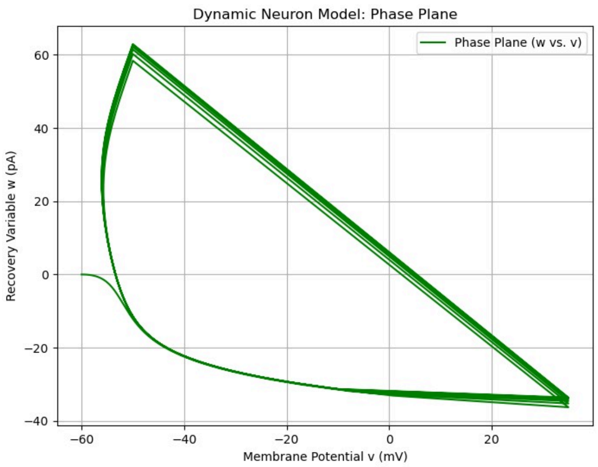
Output of the Code

V versus T



W versus T






```
Values of v and w from index 96 to 105
 Index |      v (mV) |      w (pA)
--------------------------------
    96 |      -60.00 |        0.00
    97 |      -60.00 |        0.00
    98 |      -60.00 |        0.00
    99 |      -60.00 |        0.00
   100 |      -60.00 |        0.00
   101 |      -59.65 |        0.00
   102 |      -59.04 |       -0.04
   103 |      -58.50 |       -0.11
   104 |      -58.02 |       -0.21
   105 |      -57.60 |       -0.34
```

Disadvantages of Heun's method

- because Heun's method is an explicit method it does not fare well with stiff ode's like with the model we are using
- because the Hodgkin Huxley model is stiff with sharp changes, As a second-order method $O(h^2)$, it is less accurate than higher-order methods like RK4 and other adaptive methods