Midpoint (RK2) Method

© **Omar Gamal's Report**

- Izhikevich model and midpoint method

The Izhikevich model equations considered as Stiff ODEs where certain components of the solution change on a much faster timescale than others, leading to numerical instability or inefficiency when solved with standard methods like the explicit Euler or **midpoint (RK2) method** as the Explicit methods like **RK2** are generally not well-suited for stiff ODEs because they require extremely small step sizes (h) to remain stable, especially when the system exhibits rapid changes (e.g., the spikes in v when it exceeds 35 in your simulation)

- How to handle Stiff ODEs effectively (suggestions for improvements)
    1. Variable Step Size and Adaptive Methods: Use adaptive step-size control to adjust (h) dynamically based on the local stiffness

    2. Analyze and Adjust Model Parameters: Check the parameters (e.g., C=100 C = 100 C=100, k=0.7 k = 0.7 k=0.7, a=0.03 a = 0.03 a=0.03) and initial conditions (v0=−60 v_0 = -60 v0=−60, w0=0 w_0 = 0 w0=0) for sources of stiffness. Reducing h h h (e.g., to 0.1 ms) or adjusting a a a (recovery speed) might mitigate some stiffness

➢ The Problem solved using midpoint method

$$C\,dv/dt = k(v - v_r)(v - v_t) - w + I_n$$
$$dw/dt = a[b(v - v_r) - w]$$

➢ Parameters used:

- C (Capacitance) = 100
- k (Voltage scaling factor) = 0.7v
- Vr (Resting potential) = -60
- Vt (Threshold potential) = -40
- a (Recovery speed) = .03
- b (Coupling v to w) = -2
- d (Amount to increase w after spike) = 100
- Vpeak (Peak threshold) = 35
- In (Constant input current) -> if iteration < 101 , In = 0 , else In = 70
- v (neuron membrane potential)
- w (recovery current)
- W0 (initial value of w) = 0
- V0 (initial value of v) = -60

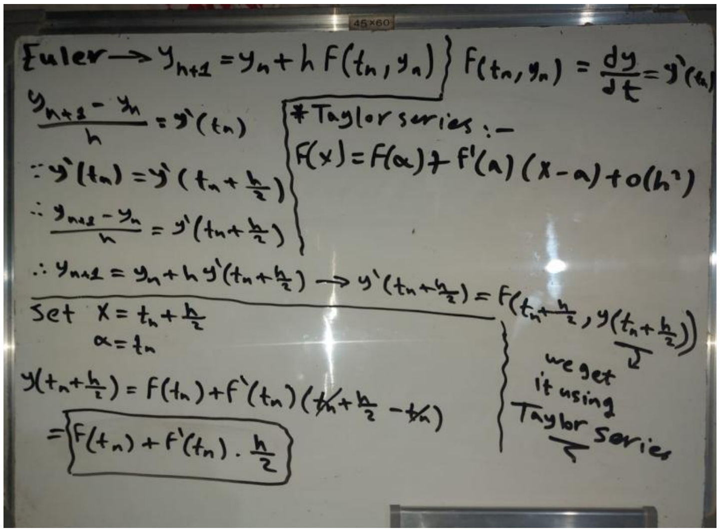
- Midpoint method (RK2) equations

1. $K_{1v} = F_v (W_n , V_n)$
2. $K_{1v} = F_v (W_n , V_n)$
3. $V_{mid} = V_n + (h/2) * K_{1v}$
4. $W_{mid} = W_n + (h/2) * K_{1w}$
5. $K_{2v} = F_v(V_{mid} , W_{mid})$
6. $K_{2w} = Fw(V_{mid} , W_{mid})$
7. $V_{n+1} = V_n + h * K_{2v}$
8. $W_{n+1} = W_n + h * K_{2w}$

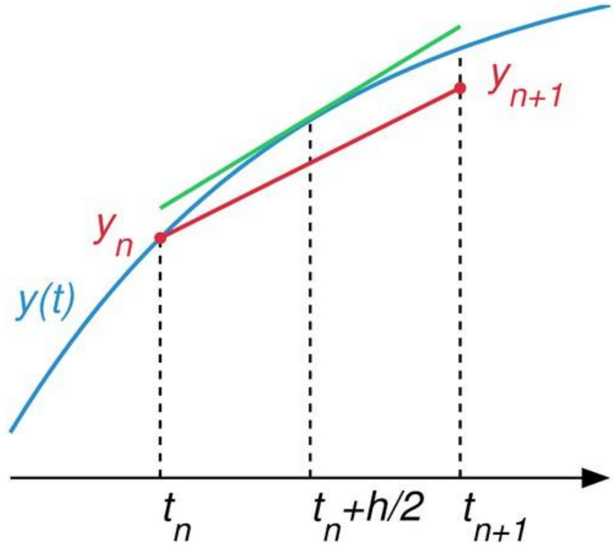
- How to use the method :
    1. Calculate K_{1v} and K_{1w} using the calculations mentioned above (Step 1)
    2. Calculate V_{mid} , W_{mid} using the calculations form Step 1 (Step 2)
    3. Calculate K_{2v} , K_{2w} using the calculations from Step 2 (Step 3)
    4. Calculate V_{n+1} , W_{n+1} using the calculations from step 3 (Step 4)
    5. Repeat the steps (starting from step 1) as $V_n = V_{n+1}$ and $W_n = W_{n+1}$, only if V_{n+1} didn't exceed the spike value but if it does then $V_n = c = -50$, $W_{n+1} = W_{n+1} + d$

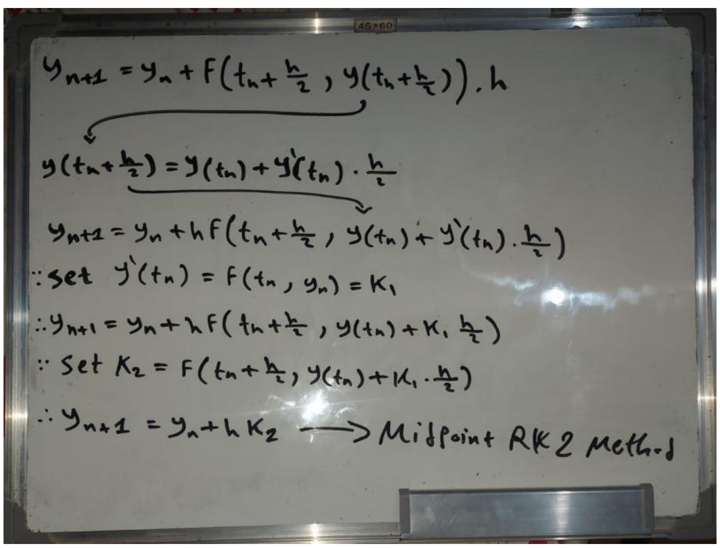
➢ The Derivation of the midpoint method

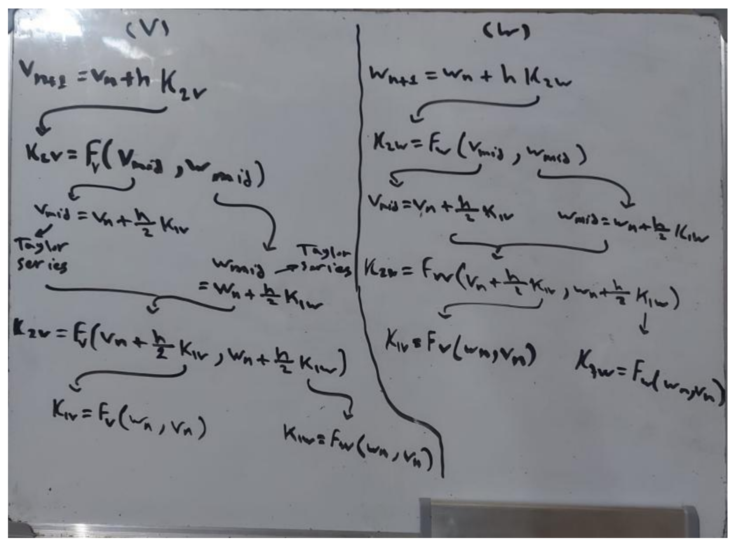
The midpoint method is a refnement of the **Euler Method**



Board 1







Board 2 **Board 3**

- **Board 1 and Board 2**

  In Board No.1 and Board No.2, We try to conclude the equation of Midpoint method in terms of one variable (Which is y)

- **Board 3**

  In Board No.3, We try to conclude the equation of Midpoint method in terms of 2 variables which are (v, w), these equations are the one used to solve the given problem

- Code of Izhikevich model using Midpoint method :

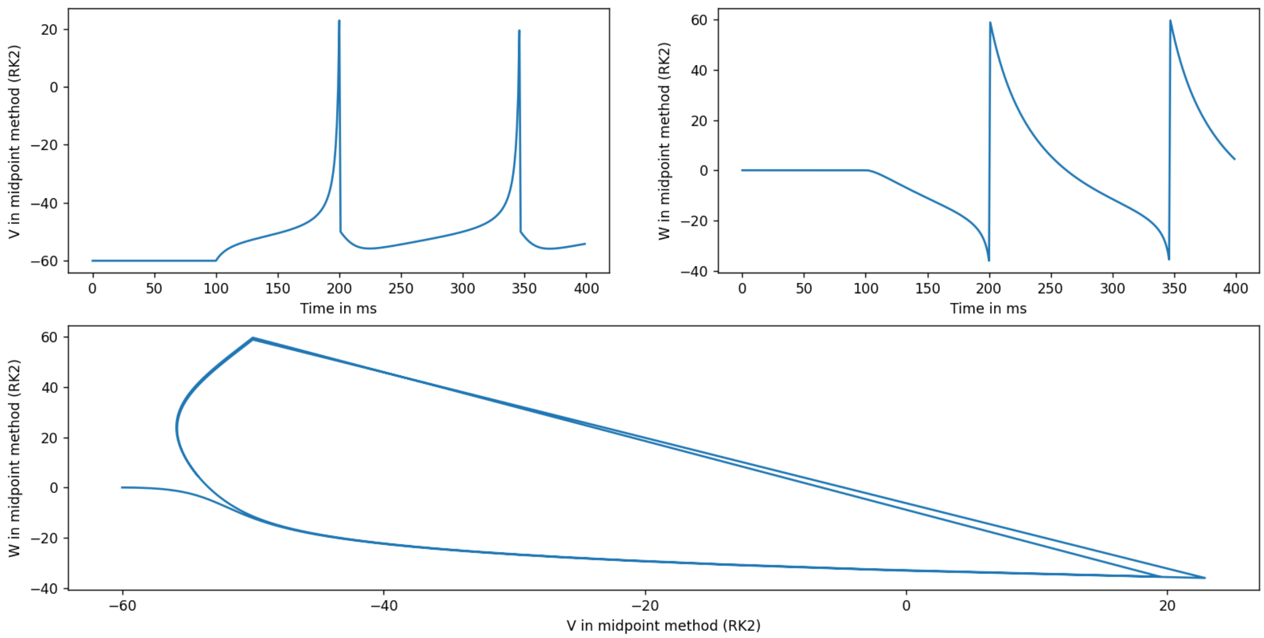
```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

def midpoint_method(v_old , w_old , h_value , num_iter):

    # defining the arrays for dataframe rendering

    v_old_values = []

    v_new_values = []

    w_old_values = []

    w_new_values = []

    k_1_v_values = []

    k_1_w_values = []

    k_2_v_values = []

    k_2_w_values = []

    iterations   = []

    # defining the parameters that will be used in equations

    c = 100

    k = .7

    v_r = -60

    v_t = -40

    a = .03

    b = -2

    d = 100

    I_N = 0

    # looping "num_iter" number of iterations

    for i in range(num_iter):

        # if the iteration was larger than or equal to 101 , I_N will equal to 70

        if i + 1 >= 101:

            I_N = 70

        # calculating the K1v and K1w (Step 1)

        print(f"\nIter : {i + 1}")

        k_1_v = (1/c)*(k*(v_old - v_r)*(v_old - v_t) - w_old + I_N)

        k_1_w = a*(b*(v_old - v_r) - w_old)

        # calculating the mid value of v using k_1_v and mid value of w using k_1_w (Step 2)

        v_mid = v_old + (h_value / 2) * k_1_v

        w_mid = w_old + (h_value / 2) * k_1_w
```

```
        # calculating the K2v and K2w using mid values from Step 2 (Step 3)

        k_2_v = (1 / c) * (k * (v_mid - v_r) * (v_mid - v_t) - w_mid + I_N)

        k_2_w = a * (b * (v_mid - v_r) - w_mid)

        # calculating the new values of v and w using values from step 3 (Step 4)

        v_new = v_old + h_value * k_2_v

        w_new = w_old + h_value * k_2_w

        # appending the values to the arrays which intially declared for rendering the values using dataframe

        k_1_v_values.append(k_1_v)

        k_2_v_values.append(k_2_v)

        k_1_w_values.append(k_1_w)

        k_2_w_values.append(k_2_w)

        v_old_values.append(v_old)

        w_old_values.append(w_old)

        v_new_values.append(v_new)

        w_new_values.append(w_new)

        iterations.append(i)

        # checking the spike value if V became bigger than the spike value , then V of next iteration = -50 and W of next
iteration = previous value of W + d

        if v_new >= 35:

            v_old = -50

            w_old = w_new + d

        else:

            v_old = v_new

            w_old = w_new

    # rendering all the values calculated using dataframe from pandas

    df = pd.DataFrame({"v old" : v_old_values , "w old" : w_old_values , "K1v" : k_1_v_values , "k1w" : k_1_w_values ,
"K2v" : k_2_v_values , "K2w" : k_2_w_values , "v new" : v_new_values , "w new" : w_new_values} , index = iterations)

    # plotting the value of V (y-axis) , time (x-axis) in ms

    ax_1 = plt.subplot(2 , 2 , 1)

    ax_1.plot(iterations , v_old_values )

    ax_1.set_xlabel("Time in ms")

    ax_1.set_ylabel("V in midpoint method (RK2)")

    # plotting the value of W (y-axis) , time (x-axis) in ms

    ax_2 = plt.subplot(2 , 2 , 2)

    ax_2.plot(iterations , w_old_values)

    ax_2.set_xlabel("Time in ms")

    ax_2.set_ylabel("W in midpoint method (RK2)")

    # plotting the value of W (y-axis) , V (x-axis)
```

```python
    ax_3 = plt.subplot(2 , 2 , (3 , 4))

    ax_3.plot(v_old_values , w_old_values)

    ax_3.set_xlabel("V in midpoint method (RK2)")

    ax_3.set_ylabel("W in midpoint method (RK2)")

    plt.show()

    print(df)

# initial values of Vo = -60 , Wo = 0 , time = 1 ms , number of iteraions = 1000

midpoint_method(-60 , 0 , 1 , 1000)
```

- Code output:

```
Iter : 1000
         v old      w old      K1v       k1w       K2v       K2w      v new      w new
0   -60.000000   0.000000 0.000000  0.000000  0.000000 -0.000000 -60.000000   0.000000
1   -60.000000   0.000000 0.000000 -0.000000  0.000000 -0.000000 -60.000000   0.000000
2   -60.000000   0.000000 0.000000 -0.000000  0.000000 -0.000000 -60.000000   0.000000
3   -60.000000   0.000000 0.000000 -0.000000  0.000000 -0.000000 -60.000000   0.000000
4   -60.000000   0.000000 0.000000 -0.000000  0.000000 -0.000000 -60.000000   0.000000
..         ...        ...      ...       ...       ...       ...        ...        ...
995 -37.672854 -25.922633 1.322936 -0.561950  1.442965 -0.593209 -36.229889 -26.515842
996 -36.229889 -26.515842 1.592470 -0.630731  1.753561 -0.669045 -34.476328 -27.184887
997 -34.476328 -27.184887 1.958740 -0.715874  2.181881 -0.763898 -32.294447 -27.948784
998 -32.294447 -27.948784 2.473894 -0.823870  2.795335 -0.885728 -29.499111 -28.834513
999 -29.499111 -28.834513 3.230350 -0.965018  3.717012 -1.047453 -25.782099 -29.881966
```



➢ Why would we use this method?

1. Easy to implement using python
2. It uses intermediate step (calculating the midpoint) which will reduce truncation error
3. It has more accurate than Euler method at same step size (h)
4. It is computationally cheaper than RK4 method

➢ Why wouldn't we use this method?

1. Not accurate as RK4
2. Twice as expensive as Euler method computationally
3. Not easy to be implemented manually (using paper and pen)