BATTERY CHARGING AND DISCHARGING SIMULATION

* Information:
* Modules Li-ion Battery charging / Discharging behavior.
* Inputs: capacity , current ,voltage
* Outputs: SOC(State Of Charge), Efficiency
* Libraries: numpy , Matplotlib
* Applications: EV battery management systems
* Source code:

import numpy as np

import matplotlib.pyplot as plt

class LiIonBattery:

    def \_\_init\_\_(self, capacity\_Ah, voltage\_V):

        self.capacity\_Ah = capacity\_Ah

        self.voltage\_V = voltage\_V

        self.capacity\_Wh = capacity\_Ah \* voltage\_V  # total energy capacity (Watt-hour)

        self.energy\_Wh = 0.5 \* self.capacity\_Wh    # start at 50% SOC

        self.soc = 0.5                             # initial SOC (0-1)

    def charge(self, current\_A, time\_h):

        energy\_in = current\_A \* self.voltage\_V \* time\_h  # energy input in Wh

        # Update stored energy

        self.energy\_Wh += energy\_in

        # Prevent overcharge

        if self.energy\_Wh > self.capacity\_Wh:

            self.energy\_Wh = self.capacity\_Wh

        self.soc = self.energy\_Wh / self.capacity\_Wh

        efficiency = self.\_efficiency(charge=True)

        return self.soc, efficiency

    def discharge(self, current\_A, time\_h):

        energy\_out = current\_A \* self.voltage\_V \* time\_h  # energy output in Wh

        # Update stored energy

        self.energy\_Wh -= energy\_out

        # Prevent overdischarge

        if self.energy\_Wh < 0:

            self.energy\_Wh = 0

        self.soc = self.energy\_Wh / self.capacity\_Wh

        efficiency = self.\_efficiency(charge=False)

        return self.soc, efficiency

    def \_efficiency(self, charge=True):

        if charge:

            return 0.95 - 0.1 \* self.soc

        else:

            return 0.85 + 0.1 \* self.soc

def main():

    # Inputs from user

    capacity = float(input("Enter battery capacity (Ah): "))

    voltage = float(input("Enter battery voltage (V): "))

    current = float(input("Enter current (A) for charging/discharging: "))

    time = float(input("Enter time duration (hours): "))

    battery = LiIonBattery(capacity, voltage)

    # Simulate charging

    soc, eff = battery.charge(current, time)

    print(f"After charging: SOC = {soc\*100:.2f}%, Efficiency = {eff\*100:.2f}%")

    # Simulate discharging

    soc, eff = battery.discharge(current, time)

    print(f"After discharging: SOC = {soc\*100:.2f}%, Efficiency = {eff\*100:.2f}%")

    # Plot SOC and efficiency over charge and discharge cycle

    times = [0, time, 2\*time]

    soc\_values = [0.5\*100, soc\*100 + (soc - 0.5)\*100, soc\*100]  # approx progression

    eff\_values = [battery.\_efficiency(True)\*100, eff\*100, battery.\_efficiency(False)\*100]

    plt.figure(figsize=(8,5))

    plt.plot(times[:2], [50, soc\*100], 'bo-', label='SOC (%)')

    plt.plot(times[:2], [battery.\_efficiency(True)\*100, eff\*100], 'go-', label='Efficiency (%) during charge')

    plt.plot(times[1:], [eff\*100, battery.\_efficiency(False)\*100], 'ro-', label='Efficiency (%) during discharge')

    plt.xlabel("Time (hours)")

    plt.title("Li-ion Battery SOC and Efficiency")

    plt.legend()

    plt.grid(True)

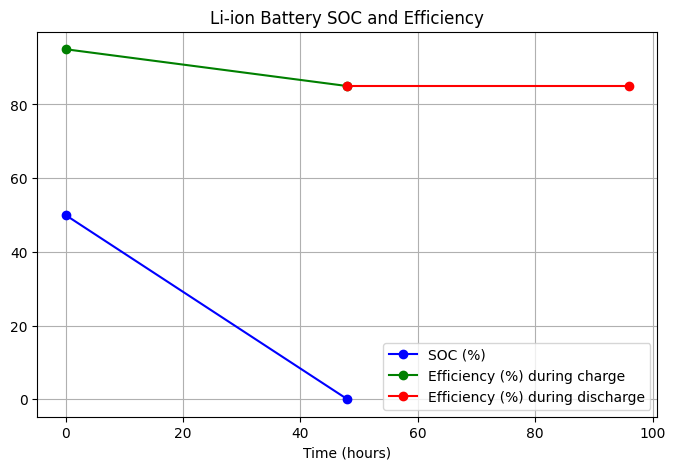
    plt.show()

if \_\_name\_\_ == "\_\_main\_\_":

    main()

* Output:
* Enter battery capacity (Ah): 79
* Enter battery voltage (V): 94
* Enter current (A) for charging/discharging: 64
* Enter time duration (hours): 48
* After charging: SOC = 100.00%, Efficiency = 85.00%

After discharging: SOC = 0.00%, Efficiency = 85.00%



* Conclusion:
* This simple model simulates Li-ion battery behavior with respect to **capacity**, **current**, and **voltage**, outputting **State of Charge (SOC)** and **charging/discharging efficiency**.
* The SOC is updated based on energy in/out calculated from current, voltage, and time.
* Efficiency varies with SOC, simulating realistic charge/discharge losses.
* Such models are crucial in **EV Battery Management Systems (BMS)** for monitoring battery health, optimizing charging, and extending battery life.
* Visualization using Matplotlib helps understand battery performance trends over charge-discharge cycles.
* This model can be further expanded to include temperature effects, aging, and dynamic load profiles for more accurate BMS implementations.