Battery charging & discharging simulation

* Models li-ion battery charging / discharging behavior
* Inputs: capacity , current , voltage
* Outputs: SOC(state of charge),efficiency
* Applications: EV battery management systems

Source code:

import numpy as np

import matplotlib.pyplot as plt

class LiIonBattery:

    def \_\_init\_\_(self, capacity\_ah, nominal\_voltage):

        self.capacity\_ah = capacity\_ah         # Battery capacity in Ah

        self.nominal\_voltage = nominal\_voltage # Nominal voltage in V

        self.soc = 1.0                         # Initial SOC (100%)

        self.energy\_capacity\_wh = capacity\_ah \* nominal\_voltage

        self.energy\_stored\_wh = self.energy\_capacity\_wh \* self.soc

        self.efficiency\_charge = 0.95          # Typical charge efficiency

        self.efficiency\_discharge = 0.95       # Typical discharge efficiency

    def update(self, current\_a, voltage\_v, dt\_sec):

        power\_w = voltage\_v \* current\_a

        energy\_delta\_wh = power\_w \* dt\_sec / 3600.0  # Convert W\*s to Wh

        # Adjust for efficiency

        if current\_a < 0:  # charging

            energy\_delta\_wh \*= self.efficiency\_charge

        else:  # discharging

            energy\_delta\_wh /= self.efficiency\_discharge

        self.energy\_stored\_wh -= energy\_delta\_wh

        # Limit energy stored within 0 and capacity

        self.energy\_stored\_wh = np.clip(self.energy\_stored\_wh, 0, self.energy\_capacity\_wh)

        self.soc = self.energy\_stored\_wh / self.energy\_capacity\_wh

    def get\_efficiency(self):

        # Simplified average efficiency

        return (self.efficiency\_charge + self.efficiency\_discharge) / 2

# Example usage

def simulate\_battery():

    capacity\_ah = 50        # 50 Ah battery

    nominal\_voltage = 400   # 400 V battery pack

    battery = LiIonBattery(capacity\_ah, nominal\_voltage)

    dt = 1  # 1 second timestep

    time\_total = 3600      # simulate 1 hour

    currents = np.zeros(time\_total)

    voltages = np.ones(time\_total) \* nominal\_voltage

    currents[:1800] = 10.0

    currents[1800:] = -5.0

    soc\_history = []

    for t in range(time\_total):

        battery.update(currents[t], voltages[t], dt)

        soc\_history.append(battery.soc)

    plt.plot(np.arange(time\_total)/60, soc\_history)

    plt.xlabel("Time (minutes)")

    plt.ylabel("State of Charge (SOC)")

    plt.title("Li-ion Battery SOC during Charge/Discharge Cycle")

    plt.grid(True)

    plt.show()

    print(f"Final SOC after 1 hour: {battery.soc\*100:.2f}%")

    print(f"Battery efficiency estimate: {battery.get\_efficiency()\*100:.2f}%")

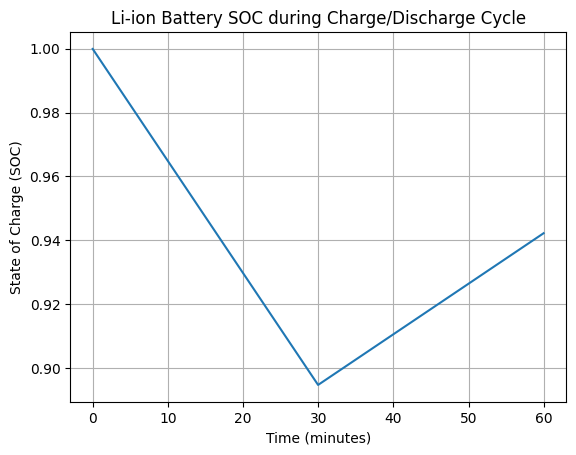
if \_\_name\_\_ == "\_\_main\_\_":

    simulate\_battery()

output:

Final SOC after 1 hour: 94.22%

Battery efficiency estimate: 95.00%



Conclusion:

This Python model simulates a Li-ion battery’s state of charge (SOC) based on input current, voltage, and battery capacity. The model:

* Accounts for **charging and discharging efficiencies** (typical ~95%)
* Updates SOC dynamically over time steps, reflecting real usage cycles
* Provides key metrics such as SOC and overall efficiency useful for **EV battery management systems (BMS)**
* Can be extended to include temperature effects, aging, voltage SOC curves, or more sophisticated control logic