Supporting Information 3: Sensitivity analysis

The simulation of interface characteristics and charge transfer dynamics for layered electrodes using cascade capacitance in supercapacitors by COMSOL software

Yixin Luo, Dongsheng Chen*, Chen Zhang, Jinkun Yang, Tian Gao*, Xiaojing Luo*

College of Mathematics and Physics, Shanghai University of Electric Power, Shanghai 200090, P. R. China

* Corresponding author

E-mail: cds78@shiep.edu.cn

Introduction:

Supporting information 3 was python code of sensitivity analysis, this code can be run in python (https://www.python.org/ org).

```
import numpy as np
import matplotlib.pyplot as plt

def monte_carlo_sensitivity_analysis(num_samples=10000):
    # Define the range of input parameters
    koh_concentration_range = (0.5, 3.0)  # mol/L
    go_content_range = (0, 18)  # %
    layers_range = (1, 6)

    # Generate random samples
    koh_concentration = np.random.uniform(koh_concentration_range[0],
koh_concentration_range[1], num_samples)
    go_content = np.random.uniform(go_content_range[0], go_content_range[1],
num_samples)
    layers = np.random.randint(layers_range[0], layers_range[1] + 1, num_samples)

    # Define model coefficients
    a = 0.5784  # A/m² per mol/L
    b = -0.4743  # A/m² per %
    c = 8.6158  # A/m² per layer
    d = 0.0167  # A/m² per %²
    e = 0  # Base value

    # Calculate peak current density with the quadratic term
    J = a * koh_concentration + b * go_content + c * layers + d * go_content**2 + e

    # Compute mean and variance
    J_mean = np.mean(J)
    J_var = np.var(J)

    # Plotting the histogram of J
    plt.figure(figsize=(10, 6))
    plt.hist(J, bins=50, density=True, alpha=0.75)
    plt.title('Probability Distribution of Peak Current Density')
    plt.xlabel('Peak Current Density(A/m²)')
    plt.ylabel('Probability Density')
    plt.grid(True)
    plt.show()

    # Compute sensitivity indices
    # Compute mean of input parameters
    mean_koh = np.mean(koh_concentration)
    mean_go = np.mean(go_content)
    mean_layers = np.mean(layers)
```

```
    # Sensitivity analysis
    # For KOH concentration
    koh_samples = np.random.uniform(koh_concentration_range[0], koh_concentration_range[1], num_samples)
    fixed_go = mean_go
    fixed_layers = mean_layers
    J_koh = a * koh_samples + b * fixed_go + c * fixed_layers + d * fixed_go**2 + e
    var_koh = np.var(J_koh)

    # For GO content
    go_samples = np.random.uniform(go_content_range[0], go_content_range[1], num_samples)
    fixed_koh = mean_koh
    fixed_layers = mean_layers
    J_go = a * fixed_koh + b * go_samples + c * fixed_layers + d * go_samples**2 + e
    var_go = np.var(J_go)

    # For number of layers
    layers_samples = np.random.randint(layers_range[0], layers_range[1] + 1, num_samples)
    fixed_koh = mean_koh
    fixed_go = mean_go
    J_layers = a * fixed_koh + b * fixed_go + c * layers_samples + d * fixed_go**2 + e
    var_layers = np.var(J_layers)

    # Calculate sensitivity indices
    sensitivity_koh = var_koh / J_var
    sensitivity_go = var_go / J_var
    sensitivity_layers = var_layers / J_var

    # Print results
    print("Sensitivity Indices:")
    print(f"KOH concentration: {sensitivity_koh:.3f}")
    print(f"GO content: {sensitivity_go:.3f}")
    print(f"Number of layers: {sensitivity_layers:.3f}")

# Run the sensitivity analysis
monte_carlo_sensitivity_analysis()
```