Electronic Supplementary Material (ESI) for Physical Chemistry Chemical Physics.
This journal is © the Owner Societies 2021

Supporting information on: "Double Layer Capacitances Analysed with Impedance Spectroscopy and Cyclic Voltammetry: Validity and Limits of the Constant Phase Element Parameterization"

Maximilian Schalenbach, Yasin Emre Durmus, Hermann Tempel, Hans Kungl, Rüdiger-A. Eichel

Content

1 Experimental Cell

Figure S1 shows a schematic illustration of the cell used for the electrochemical measurements.

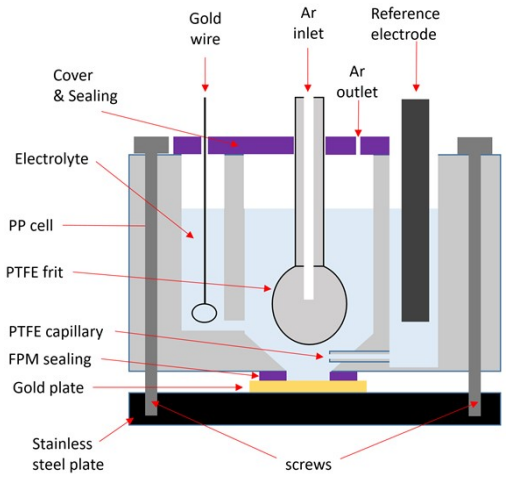

Figure S1: Schematic sketch of the in-house made electrochemical cell used in the experiments. PP: Polypropylene. PTFE: Polytetrafluoroethylene. FPM: Fluoroelastomer.

2 Reproduction measurement and discussion of measurement errors

Figure S2 and S3 show a reproduction measurement of the data that was presented in the article, for which the polishing procedure and measurement of the gold electrode (with a new 0.1 M $HClO_4$ electrolyte) were repeated.

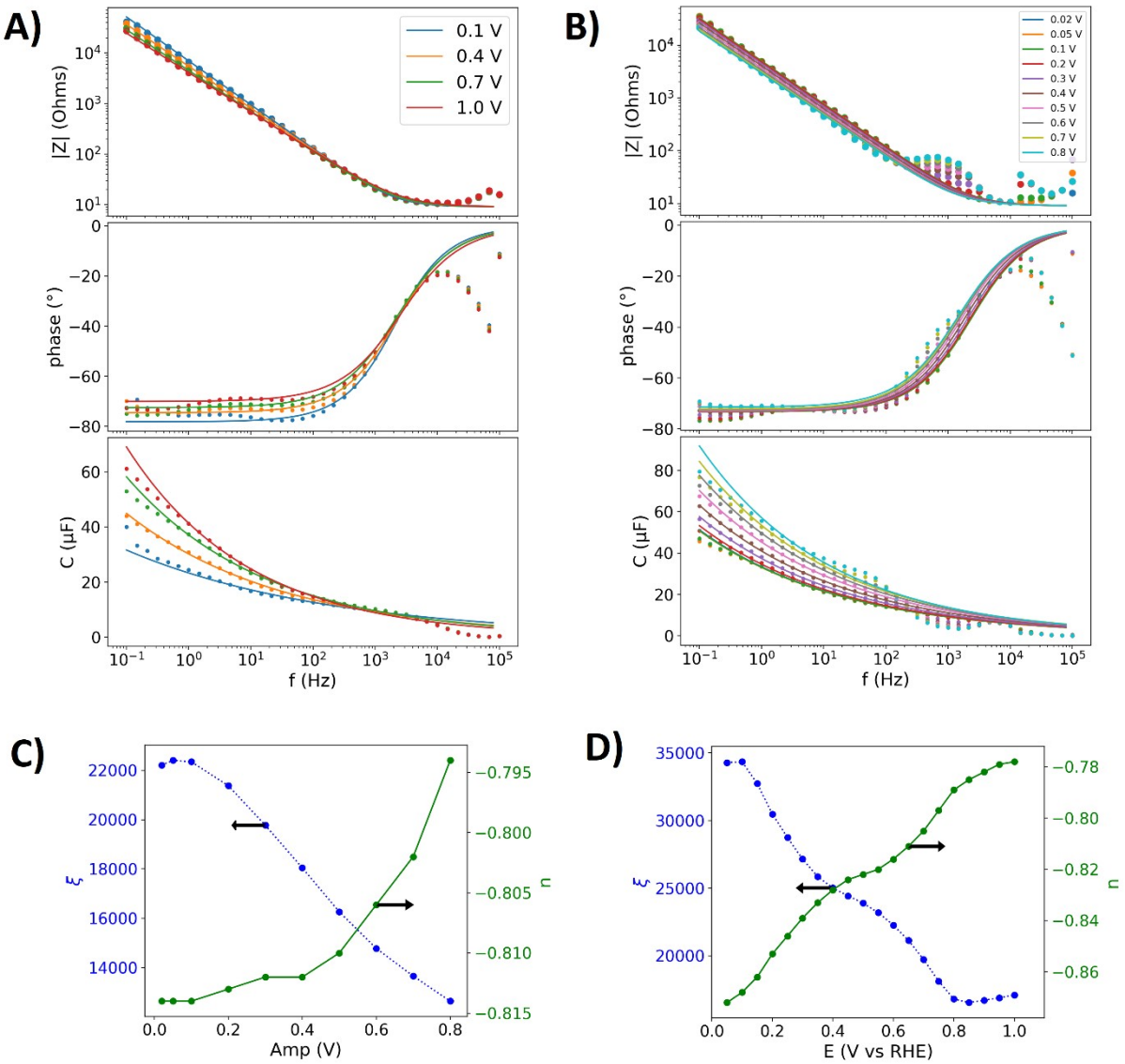

Figure S2: Impedance data of the reproduction measurement

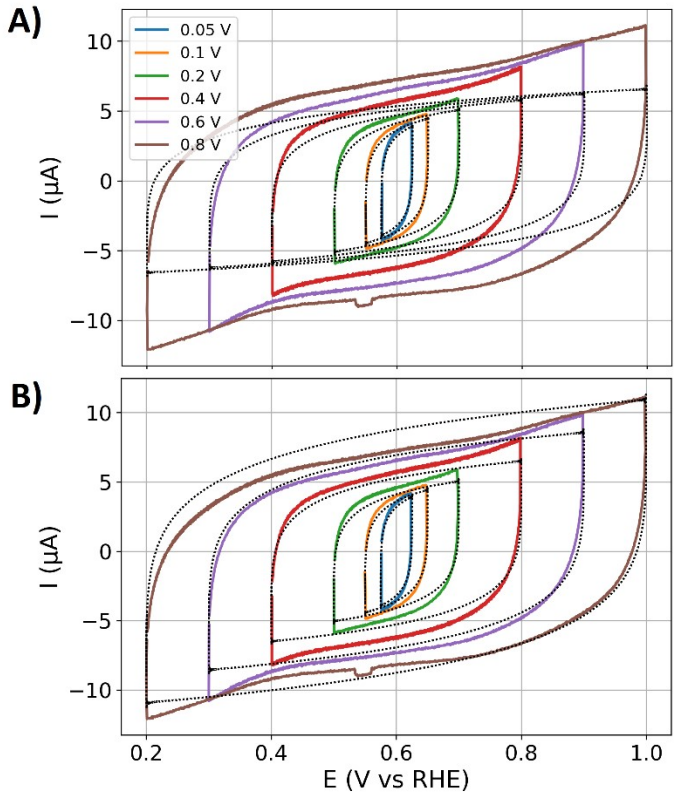

Figure S3: CV data of the reproduction measurement.

The polishing procedure of the sample comes with an error of approximately 30% that influences the absolute values of the impedance and the CV current. Relative errors in the shape of the CV data come from slightly different impurities, which include the amount of dissolved oxygen, impurities in the solution, and adsorbed species on the sample. With respect to the small currents at the polished surface, these impurities with contributions in the range of < 1µA can lead to perceivable contributions in the CV data. Moreover, the exposure at air during the cleaning procedure after polishing can lead to the adsorption of molecules and/or oxidation of the gold sample which can slightly change the CV data.

3 Full range CV data

Figure S4 shows the full range CV data from 0.05 to 1.05 V vs RHE of the polished gold electrode. Above 1.05 V vs RHE gold corrosion starts [1]. At 0.05 V vs RHE and 10^{-4} of the concentration of dissolved hydrogen under 1 bar absolute hydrogen pressure, the hydrogen electrode is in equilibrium (Nernst equation). Thus, up to 0.05 V vs RHE a significant contribution of the hydrogen evolution to the CV can be thermodynamically excluded. Thus, the CV data graphed in Figure S4 is dominated by the contributions of the double layer capacitances.

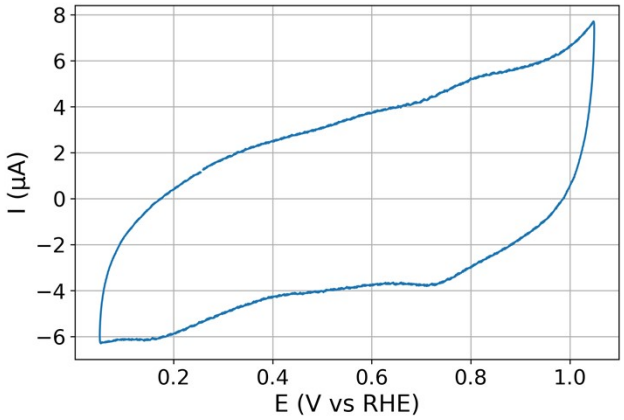

Figure S4: CV from 0.05 V to 1.05 V, showing the full electrochemical stable window.

4 Detailed impedance analysis

Figure S5 and S6 show the impedance data in the Nyquist representation. With reference the strongly increasing impedance towards lower frequencies, the Nyquist plots are dominated by the lower frequencies, for which the Bode representation was preferred in the article.

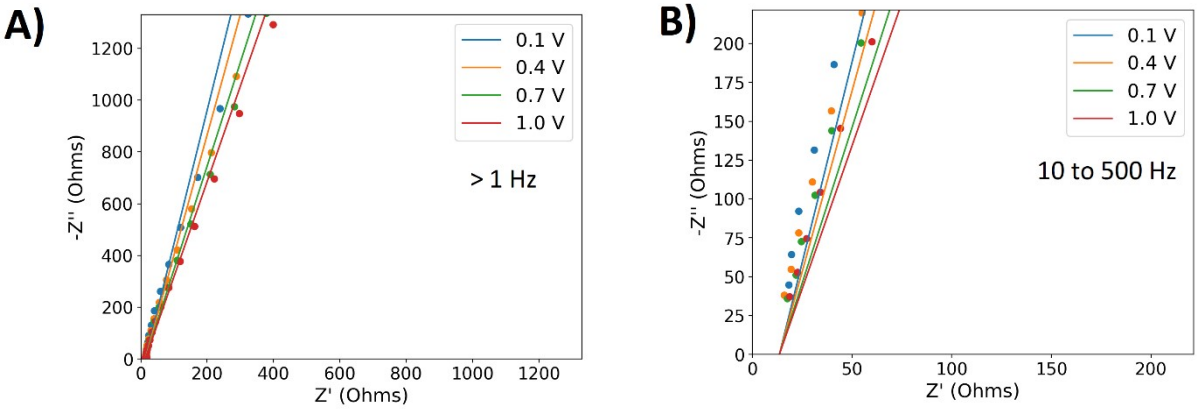

Figure S5: Nyquist plot of the impedance spectra obtained with the potential variation. A) Frequencies above 1 Hz. B) Frequencies between 10 and 500 Hz.

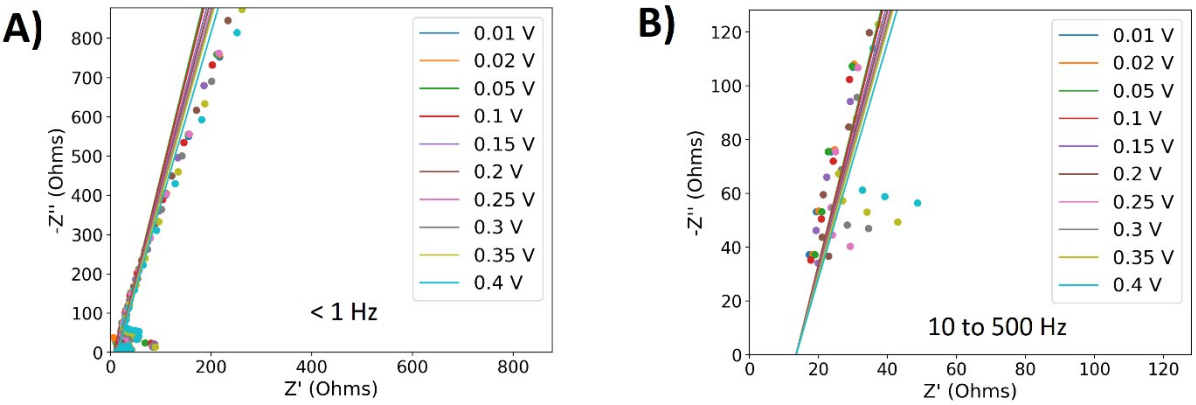

Figure S6: Nyquist plot of the impedance spectra that were obtained with amplitude variation. . A) Frequencies above 1 Hz. B) Frequencies between 10 and 500 Hz.

The fits of the impedance spectra were applied to the capacitance dispersion. To measure the fit error, the Chi-squared values were calculated with Python using the "Scipy" package with the "chisquare" module by comparing the measured values of capacitance dispersion to the calculated values of the fit. The Chi-squared values of the fits were calculated (unit of µF) for frequencies below 5 kHz as shown in Table S1 and S2. The Chi-squared values of the impedance measurements with the potential are between 0.6 and 2.1 µF. In the case of the amplitude variation, the Chi-squared values show a steadily increase from 0.1 V to 0.4 V with an increase from 2.82 at 39.4 µF, showing that the fit quality decreases as the potential dependence of the double layer response increasingly contributes to the response.

Table S1: χ^2 values for the fits to the capacitance dispersion of the impedance measurements with the potential variation.

E (V vs RHE)	$\chi^2\ (\mu F)$
0.05	2.04
0.1	1.01
0.15	1.67
0.2	1.14
0.25	1.29
0.3	0.85
0.35	0.67
0.4	0.64
0.45	0.62
0.5	0.84
0.55	1.04
0.6	0.77
0.65	0.91
0.7	0.96
0.75	1.19
0.8	1.24
0.85	1
0.9	0.95
0.95	1.02
1	1.16

Table S2: χ^2 values for the fits to the capacitance dispersion of the impedance measurements with the amplitude variation.

Amp (V)	$\chi^2\ (\mu F)$
0.02	2.82
0.05	3.07
0.1	2.99
0.2	1.7
0.3	2.23
0.4	5.97
0.5	12.12
0.6	21.45

0.7	29.47
0.8	39.36

5 Program code for CV simulation

The code runs in Python 3 on a personal computer without any advanced hardware specifications. The following text can

```
import pandas as pd
import numpy as np
import math
import matplotlib.pyplot as plt

def cpe_fourier(Amp_pp, scan_rate,f,xi,alpha, R_s):
    """
    Function that calculates the CV response.
    Parameters
    ----------
    Amp_pp : FLOAT
        The peak to peak amplitude of the CV (equals voltage range). Unit: V
    scan_rate : FLOAT
        The scan rate of teh CV. Unit: V/s.
    f : FLOAT
        Frequency. If !=0, the scan rate will be calculated from this value. Unit: Hz.
    xi : FLOAT
        Prefactor of the CPE. Unit: Ohm
    alpha : FLOAT
        Exponent of the CPE times -1 (Equals -n). Unit: Dimensionless
    R_s : FLOAT
        Serial resistance (dominated by electrolyte resistance): Unit: Ohm.

    Returns: Pandas DataFrame with the simulation results
    """
    # Amplitude around mean potential
    Amp_0 = Amp_pp/2

    # Define whether scan rate or frequency was defined
    if f == 0:
        print("calculating frequency")
        f = scan_rate/(Amp_pp*2)
    else:
        scan_rate = f*Amp_pp*2

    omega_0 = 2*math.pi*f #
    scan_rate = f*Amp_pp*2

    print("scan_rate ="+str(scan_rate))
    print("f ="+str(f))
    print("Amp ="+str(Amp_0*2))

    n = 500 # Amount of Fourier series terms

    def U_amp_phase(n):
        # Calculates the angular frequency, Amplitude and Phase of the Fourier Series representation of the potential variation
        omega = omega_0*(2*n-1)
        U_Amp = 8*Amp_0/((math.pi * (2*n-1))**2)
        U_phase = -3*math.pi/2
        return [omega, U_Amp, U_phase]

    def I_amp_phase(n, omega, U_Amp, U_phase):
        # calculates the impedance of an angular frequency array
        sigma = xi*math.sin(np.pi*(-alpha)/2)*omega**(-alpha) # substition of the real part of the CPE
        rho = xi*math.cos(np.pi*(-alpha)/2)*omega**(-alpha) # substition of the imaginary part of the CPE
        beta = rho/(rho**2 + sigma**2) # substition 1
        gamma = -sigma/(rho**2 + sigma**2)
        Z_real = R_s + beta/(beta**2+gamma**2)
        Z_imag = -gamma/(beta**2+gamma**2)
        I_phase = math.atan(Z_imag/Z_real) + U_phase
        I_Amp = U_Amp/((Z_real**2 + Z_imag**2)**0.5)
```

```
    return [I_Amp, I_phase]

# Dataframes that collects the angular frequencies and the amplitudes and phase angles of the excitation and the response
df_results = pd.DataFrame(columns = ["n","omega","U_Amp","U_phase", "I_Amp","I_phase"]).set_index("n")

print("Iterating through the fourier series")
for i in range(1,n+1):
    # calculate amplitude and impedance based on the impedance
    [omega, U_Amp, U_phase] = U_amp_phase(i)
    [I_Amp, I_phase] = I_amp_phase(i,omega, U_Amp, U_phase)
    # Append results to the results dataframe
    df_results.loc[i] = 0
    df_results.loc[i][["omega","U_Amp","U_phase","I_Amp","I_phase"]] = [omega, U_Amp, U_phase,I_Amp, I_phase]

# Calculate the excitation function and its response in the time domain
print("Starting time domain")
# Definition of the lattice
time_steps = 5000 # Amount of time steps of the calculated response
periods = 1.001 # Amount of periods of the response
total_time = periods/f
h = total_time/time_steps # time resolution

# Dataframes, that contain the information in the time domain
t_array = np.arange(0,total_time,h)
df_U_t = pd.DataFrame()
df_I_t = pd.DataFrame()
df_results_t = pd.DataFrame()
df_U_t["t_array"] = t_array
df_I_t["t_array"] = t_array
df_results_t["t_array"] = t_array
df_results_t["U"] = 0
df_results_t["I"] = 0

# Add the current waves in the time domain
for j in range(1,n+1):
    [omega, U_Amp, phase,I_Amp, I_phase] = df_results.loc[j][["omega","U_Amp","U_phase","I_Amp","I_phase"]]
    df_U_t[str(j)] = U_Amp*np.sin(omega*df_U_t["t_array"]- U_phase)
    df_I_t[str(j)] = I_Amp*np.sin(omega*df_I_t["t_array"]- I_phase)
    # Superposition of the individual waves
    df_results_t["U"] += df_U_t[str(j)]
    df_results_t["I"] += df_I_t[str(j)]

return df_results_t
```

References

1 S. Cherevko, A. a. Topalov, A. R. Zeradjanin, I. Katsounaros and K. J. J. Mayrhofer, Gold dissolution: towards understanding of noble metal corrosion, *RSC Adv.*, 2013, **3**, 16516.