Electrochemical Surface State Sensing Using Electrochemical Impedance Spectroscopy

Eric Viklund

Contents

1 Abstract

Using electrochemical impedance spectroscopy, we have devised a method of sensing the microscopic surface conditions on the surface of niobium as it is undergoing an electrochemical polishing (EP) treatment. The method uses electrochemical impedance spectroscopy (EIS) to gather information on the surface state of the electrode without disrupting the polishing reaction.

The EIS data is analyzed using a so-called distribution of relaxation times (DRT) method. Using DRT, the EIS data can be deconvolved into discrete relaxation time peaks without any apriori knowledge of the electrode dynamics. By analyzing the relaxation time peaks, we are able to distinguish two distinct modes of the EP reaction. As the polishing voltage is increased, the electrode transitions from the low voltage EP mode, characterized by two relaxation time peaks, to the high voltage EP mode, characterized by three relaxation time peaks. By analyzing EPed samples, we show that samples polished in the low voltage mode have significantly higher surface roughness due to grain etching and faceting. Samples polished in the high voltage mode obtain a smooth surface finish. This shows that EIS combined with DRT analysis can be use to predict etching on EPed Nb. This method can also be performed before or during the EP, which could allow for adjustment of polishing parameters to guarantee a smooth cavity surface finish.

2 Introduction

Electropolishing (EP) is commonly used to polish Nb SRF cavities to nanometer scale surface roughness.

EIS has been used to study the chemistry of niobium in HF electrolytes before**??**. However, these studies have only analyzed the spectrum using qualitative methods or traditional equivalent circuit fitting techniques. These methods have proven insufficient for explaining several important phenomenon of EP such as surface etching and spontaneous current oscillations.

In this study, we use a model-free method of analyzing the EIS spectrum called distribution of relaxation times (DRT) analysis. The advantage of this method is that the EIS spectrum can be easily characterized over a large range of polishing conditions without making any assumptions about the underlying chemical processes.

Using this method, we are able to observe the formation of the Nb_2O_5 layer as the polishing voltage is increased. We also show that the formation of the oxide corresponds with a reduction in surface etching by analyzing Nb samples using SEM and confocal laser microscopy.

3 Theory

An alternating voltage is applied to the niobium electrode of the form:

$$E = E_0 + E_{AC} \sin(\omega * t) \tag{1}$$

For small amplitudes of $\mathrm{E_{ac}}$ and assuming the system is operating at a steady-state, the electrode response to the alternating voltage can be described by a linear time-invariant system (LTI). Thus the form of the current must be:

$$I = I_0 + I_{AC}\cos(\omega * t + \phi) \tag{2}$$

The complex impedance of the electrode is determined by the phase difference, ϕ, and the ratio of the magnitudes of the AC component of the current and voltage:

$$Z = \frac{I_{AC}}{E_{AC}} * e^{j\phi} \tag{3}$$

$$or$$

$$Z = Z' + jZ'' \tag{4}$$

Here we use 'j' as the imaginary unit. Z' and Z" are the real and imaginary components of Z.

The impedance spectrum of the niobium was deconvolved using the distribution of relaxation times (DRT) method. We consider the electrode as a collection of infinitesimal discrete circuit elements. This is motivated by the fact that the electrode and its surrounding environment is a 3-dimensional object where each point on the elctrode acts independantly from every other part of the elctrode. This is in contrast with the classical view of electrochemical systems that treat the electrodes as homogeneous objects described by a set of discrete curcuit elements.

The fundamental elctrochemical circuit element is the RC circuit, a resistor and a capacitor in parallel, and can be described by it's time constant, τ=RC. Taking an infinite number of RC circuits in series we obtain what is known as a Voigt circuit. The impedance of an RC circuit and of an infinite Voight circuit is given by the equations:

$$Z_{RC} = \frac{R}{1 + j\omega\tau} \tag{5}$$

$$Z_{Voigt} = R + j\omega L + \int_0^\infty \frac{G(\tau)d\tau}{1 + j\omega\tau} \tag{6}$$

The function G(τ) is the distribution of relaxation times of the measured system.

It is more convenient to rewrite the integral in a log scale, since EIS measurements are typically performed over multiple orders of magnitude.

$$Z = R + j\omega L + \int_{-\infty}^{\infty} \frac{\gamma(ln\tau) dln\tau}{1 + j\omega\tau} \tag{7}$$

To solve for the function $\gamma(\ln\tau)$ numerically, we discretize the problem by introducing a test function.

$$\gamma(ln\tau) \approx \sum_{n=0}^{N} x_n \phi_n(ln\tau) \tag{8}$$

$$Z \approx R + j\omega L + \sum_{n=0}^{N} x_n \int_{-\infty}^{\infty} \frac{\phi_n(ln\tau) dln\tau}{1 + j\omega\tau} \tag{9}$$

or in matrix form:

$$Z = R\mathbf{1} + \mathbf{A}'\mathbf{x} + j(\omega L\mathbf{1} + \mathbf{A}''\mathbf{x}) \tag{10}$$

$$\mathbf{x} = [x_0, x_1, \ldots, x_N]^T \tag{11}$$

$$\mathbf{A}' = \int_{-\infty}^{\infty} \frac{\phi_n(ln\tau) dln\tau}{1 + \omega^2\tau^2} \tag{12}$$

$$\mathbf{A}'' = \int_{-\infty}^{\infty} \frac{-\omega\tau\phi_n(ln\tau) dln\tau}{1 + \omega^2\tau^2} \tag{13}$$

to solve for x we fit equation~**??** to the experimental impedance measurements by minimizing the square difference. The matrix M is a normalization term to prevent overfitting.

$$\min_{\mathbf{x},R,L} [||Z'_{exp} - (R\mathbf{1} + \mathbf{A}'\mathbf{x})||^2 + ||Z''_{exp} - (\omega L\mathbf{1} + \mathbf{A}''\mathbf{x})||^2 + |\mathbf{x}\mathbf{M}\mathbf{x}^T|] \tag{14}$$

M is calculated by integrating the function G(τ) and it's derivatives. The derivative of G is equal to the sum of the derivatives of the test functions.

$$\frac{d^k\gamma}{dln\tau^k} = \sum_{n=0}^{N} x_n \frac{d^k\phi_n}{dln\tau^k} \tag{15}$$

We want to penalize the magnitudes of the derivatives of γ, thus we calculate the square of the derivative and integrate.

$$(\frac{d^k\gamma}{dln\tau^k})^2 = \sum_{n=0}^{N} x_n \frac{d^k\phi_n}{dln\tau^k} \sum_{m=0}^{N} x_m \frac{d^k\phi_m}{dln\tau^k} \tag{16}$$

$$\int_0^\infty (\frac{d^k\gamma}{dln\tau^k})^2 dln\tau = \sum_{n=0}^{N}\sum_{m=0}^{N} x_n x_m \int_0^\infty \frac{d^k\phi_n}{dln\tau^k}\frac{d^k\phi_m}{dln\tau^k} dln\tau \tag{17}$$

$$(\mathbf{M}_k)_{n,m} = \int_0^\infty \frac{d^k\phi_n}{dln\tau^k}\frac{d^k\phi_m}{dln\tau^k} dln\tau \tag{18}$$

$$\mathbf{M} = \sum_{k=0}^{K} \lambda_k \mathbf{M}_k \tag{19}$$

The optimum values of λ_k are not trivial to find. Higher values lead to stronger smoothing of γ, which could lead to important details being ignored. If λ_k is too small, the procedure will overfit to any noise in the experimental data.

4 Experimental

Four samples were measured using the EIS method. To examine the effect of nitrogen doping on the electropolishing reaction, two of the samples were were exposed to nitrogen gas at 800˜ for two minutes. To test the effect of cold EP, the electrolyte temperature was lowered to 13˜ during two of the sample measurements.

:	Serial Number	Electrolyte Temperature	Nitrogen Doped

```
import numpy as np
import h5py
import matplotlib.pyplot as plt

mm = 0.03937

data = h5py.File('./Data/data.hdf5')

samples = data['samples']
```

```
sample_serial_numbers = samples.keys()

fig = plt.figure(figsize=(7,12))
subfigs = fig.subfigures(nrows=4,ncols=1)

for row, sample_serial_number in enumerate(sample_serial_numbers):
    sample_data = samples[sample_serial_number+'/data']


    subfig = subfigs[row]
    subfig.suptitle('Sample Serial Number: '+sample_serial_number)

    re_ax, im_ax = subfig.subplots(nrows=1, ncols=2)

    re_ax.set_xscale('log')
    im_ax.set_xscale('log')


    for cycle in sample_data.keys():
cycle_data = sample_data[cycle]

freq = np.array(cycle_data['freq'])
ReZ = np.array(cycle_data['ReZ'])
ImZ = np.array(cycle_data['ImZ'])

re_ax.scatter(freq,ReZ+int(cycle),marker='x',s=5)
im_ax.scatter(freq,ImZ+int(cycle),marker='x',s=5)

re_ax.set_ylim(0,40)
im_ax.set_ylim(-10,30)
```

5 Calculations

5.1 Test Function

To discretize the DRT function, we use a set of Gaussian test functions evenly spaced on the log scale.

$$\phi_n(ln\tau) = x_n e^{\frac{ln\tau - ln\tau_n}{\mu}} \tag{20}$$

For a series of impedance measurements measured at frequencies (f_1,f_2 f_m f_M) in a descending frequency order and equally spaced in on the log frequency scale, the centers of the gaussian test functions, $\ln\tau_n$ are chosen to be $1/f_m$. We note that this is an arbitrary decission chosen for convenience and the spacing between test functions and the total number of test functions used can take any value. However, changing the spacing to larger or smaller values or adding test functions outside the range of measured frequencies would have no real physical meaning.

The width, μ, of the gaussian function is set such that the full width at half maximum (FWHM) is equal to $\ln\tau_{n+1}$-$\ln\tau_{n-1}$. This ensures a good compromise between being able to fit rapidly changing regions of the DRT function and having enough overlapping regions between neighboring test functions.

```
import numpy as np
from scipy.special import hermite

def Gaussian_Func(x,sigma):
    y = np.exp(-x**2/(2*sigma**2)) / (2.5066*sigma)
    return y

def Gaussian_Derivative(x,n,sigma):
    hermite_poly = hermite(n)
    y = (-1)**n * Gaussian_Func(x,sigma) * hermite_poly(x/sigma) * (1/sigma)**n
    return y
```

5.2 Numerical Integration of A' and A"

To calculate the matrices A' and A", the integral~12 and~ 13 must be integrated numerically. This calculation is performed using the Gaussian quadrature method.

$$\int_a^b f(x)dx \approx \frac{b-a}{2}\sum_{i=1}^{n} w_i f(\frac{b-a}{2}\xi_i + \frac{b-a}{2}) \tag{21}$$

Here ξ are the roots of the n-th Legendre polynomial and w are the weights are calculated from the derivative of the n-th Legendre polynomial using the equation

$$w_i = -\frac{2}{(1-\xi_i^2)(P_n'(\xi_i))} \tag{22}$$

```
import numpy as np

def quad(f, a, b, n=5):
    # Generate the quadrature rule coefficients from the numbpy library
    x, w = np.polynomial.legendre.leggauss(n)

    # Scale the quadrature points and weights to the interval [a, b]
    x_scaled = 0.5 * (b - a) * x + 0.5 * (b + a)
    w_scaled = 0.5 * (b - a) * w

    # Evaluate the function at the quadrature points and sum up the weighted contributi
    integral = np.sum(w_scaled[:,None] * f(x_scaled),axis=0)

    return integral
```

The integrands of A', A", and M are given by the functions

```
def re_integrand(x, fn, fm, gaussian_sigma):
    x_diff = x[:,None] - fn[None,:] + fm[None,:]
    return Gaussian_Func(x, gaussian_sigma)[:,None] / (1 + np.exp(-2 * x_diff))

def im_integrand(x, fn, fm, gaussian_sigma):
    x_diff = x[:,None] - fn[None,:] + fm[None,:]
    return Gaussian_Func(x, gaussian_sigma)[:,None] * np.exp(-x_diff) / (1 + np.exp(-2

def norm_integrand(x, fn, fm, k, gaussian_sigma):
    x_diff = x[:,None] - fn[None,:] + fm[None,:]
    return Gaussian_Derivative(x_diff,k,gaussian_sigma)*Gaussian_Derivative(x,k,gaussia
```

This funtion performs the integrals on the integrands for each pair of experimental and test frequencies.

```
def integrate_test_functions(log_fm,log_fn,gaussian_sigma):

    #The measured frequencies fm and the test function frequencies fn are combined for
    fn_mesh, fm_mesh = np.meshgrid(log_fm, log_fn, indexing='ij')

    # reshape fn_mesh and fm_mesh into column vectors
    fn = fn_mesh.flatten()
    fm = fm_mesh.flatten()

    #Real integral
    integral = quad(lambda y: re_integrand(y,fn,fm,gaussian_sigma), -3*gaussian_sigma,

    A_re = integral.reshape(fn_mesh.shape)

    #Imag integral
    integral = quad(lambda y: im_integrand(y,fn,fm,gaussian_sigma), -3*gaussian_sigma,

    A_im = integral.reshape(fn_mesh.shape)

    return A_re, A_im
```

5.3 Cost Function

The cost function is defined by

```
def cost_func(x,f_m,A_re,A_im,b,norm_matrix):

    x_re = x[0:A_re.shape[1]]
    x_im = x[A_re.shape[1]:A_re.shape[1]+A_im.shape[1]]
    R = x[A_re.shape[1]+A_im.shape[1]]
    L = x[A_re.shape[1]+A_im.shape[1]+1]

    Z_sim_re = np.matmul(A_re,x_re)+R
    Z_sim_im = np.matmul(A_im,x_im)+np.exp(f_m)*L

    Z_exp_re = np.real(b)
    Z_exp_im = np.imag(b)
    Re_cost = 1.0*np.sum(np.abs(Z_exp_re-Z_sim_re))
    Im_cost = 1.0*np.sum(np.abs(Z_exp_im-Z_sim_im))
    Re_norm_cost = np.abs(np.matmul(x_re,np.matmul(norm_matrix,x_re.T)))
    Im_norm_cost = np.abs(np.matmul(x_im,np.matmul(norm_matrix,x_im.T)))

    cost = Re_cost + Im_cost + Re_norm_cost + Im_norm_cost

    return cost
```

5.4 Minimization Algorithm

```
from scipy.optimize import minimize
def DRT_Fit(measured_frequencies, measured_Z, fitting_frequencies, reg_params):

    f_m = measured_frequencies
    f_n = fitting_frequencies
    Z_m = measured_Z

    reg_order = reg_params.shape[0]
    gaussian_width = (np.max(fitting_frequencies)-np.min(fitting_frequencies))/fitting_



    #Generate parameter array
    #real, imag, resistance
    x_params = np.zeros((2*fitting_frequencies.shape[0]+2))

    #Integrate the test functions wrt the measured frequencies
    A_re, A_im = integrate_test_functions(measured_frequencies, fitting_frequencies, ga

    #Calculate the regularization matrix
    fn_mesh, fm_mesh = np.meshgrid(f_n, f_n, indexing='ij')

    fn = fn_mesh.flatten()
    fm = fm_mesh.flatten()

    M = np.zeros((fitting_frequencies.shape[0],fitting_frequencies.shape[0]))
    for k in range(reg_order):
```

```
integral = quad(lambda y: norm_integrand(y,fn,fm,k,gaussian_width/2.355), -3*gaussian_w
M += reg_params[k]*integral.reshape(fn_mesh.shape).T


    #Fit to experimental data
    b = Z_m

    #Solve for parameters
    res = minimize(cost_func,x_params,args=(f_m,A_re,A_im,b,M))
    x_params = res['x']

    #Calculate the gamma function coefficients
    x_re = x_params[:fitting_frequencies.shape[0]]
    x_im = x_params[fitting_frequencies.shape[0]:-2]
    R = x_params[-2]
    L = x_params[-1]

    return x_re, x_im, R, L
```

The minimization algorithm is applied to each of the EIS spectrum.

```
import numpy as np
import h5py
import time


data = h5py.File('./Data/data.hdf5')

samples = data['samples']

sample_serial_numbers = samples.keys()

#Generate regularization hyper-parameter array
reg_params = np.array((1e-3,2e-4,6e-5))

for row, sample_serial_number in enumerate(sample_serial_numbers):
    sample_data = samples[sample_serial_number+'/data']

    potential_steps = len(sample_data.keys())
    for cycle in sample_data.keys():
cycle_data = sample_data[cycle]

Freq = np.array(cycle_data['freq'])*2*np.pi
ReZ = np.array(cycle_data['ReZ'])
ImZ = np.array(cycle_data['ImZ'])

Z = ReZ + 1j*ImZ

#PARAMS
f_min = np.min(Freq)
f_max = np.max(Freq)
fitting_params_per_decade = 6

#Convert to log frequency
log_f_min = np.log(1e0)
log_f_max = np.log(1e6)

#Generate the fitting test function offsets
number_of_fitting_params = int(fitting_params_per_decade*(log_f_max-log_f_min))
log_f_n = np.linspace(log_f_min,log_f_max,number_of_fitting_params)
gaussian_width = (np.max(log_f_n)-np.min(log_f_n))/log_f_n.shape[0]

#Convert experimental frequencies to log space
log_Freq = np.log(Freq)

x_re = np.zeros((number_of_fitting_params))
x_im = np.zeros((number_of_fitting_params))
R = 0
L = 0

# start the timer
start_time = time.time()

x_re, x_im, R, L = DRT_Fit(log_Freq, Z, log_f_n, reg_params)


# end the timer
end_time = time.time()
```

```
# calculate the elapsed time
elapsed_time = end_time - start_time

print("Cycle: "+cycle+" Elapsed time: {:.2f} seconds".format(elapsed_time))

if 'R' in cycle_data.keys():
    del cycle_data['R']

cycle_data.create_dataset('R',data=R)

if 'L' in cycle_data.keys():
    del cycle_data['L']

cycle_data.create_dataset('L',data=L)

if 'x_re' in cycle_data.keys():
    del cycle_data['x_re']

cycle_data.create_dataset('x_re',data=x_re)

if 'x_im' in cycle_data.keys():
    del cycle_data['x_im']

cycle_data.create_dataset('x_im',data=x_im)
```

6 Results

```
import numpy as np
import h5py
import matplotlib.pyplot as plt
from matplotlib.colors import LogNorm, SymLogNorm

mm = 0.03937

data = h5py.File('./Data/data.hdf5')

samples = data['samples']
```

```
sample_serial_numbers = samples.keys()

fig = plt.figure(figsize=(7,12))
subfigs = fig.subfigures(nrows=4,ncols=1)


for row, sample_serial_number in enumerate(sample_serial_numbers):
    sample_data = samples[sample_serial_number+'/data']

    x_re = []
    x_im = []
    Current = []
    Current_min = []
    Current_max = []
    Electrode_Potential = []

    for cycle in sample_data.keys():
cycle_data = sample_data[str(cycle)]
x_re.append(cycle_data['x_re'])
x_im.append(cycle_data['x_im'])
Current.append(np.mean(cycle_data['I']))
Current_min.append(np.min(cycle_data['I']))
Current_max.append(np.max(cycle_data['I']))
Electrode_Potential.append(np.mean(cycle_data['Ewe']))

    x_re = np.array(x_re).T
    x_im = np.array(x_im).T
    Current = np.array(Current).T
    Current_min = np.array(Current_min).T
    Current_max = np.array(Current_max).T
    Electrode_Potential = np.array(Electrode_Potential).T



    x_pos = (x_re - x_im)/2
    x_neg = (x_re + x_im)/2
    """Plotting"""

    # Plot gamma functions
    #Params
```

```
plot_freq_min = 1e0
plot_freq_max = 1e6
plot_points_per_decade = 10


#Convert to log
log_plot_freq_min = np.log(plot_freq_min)
log_plot_freq_max = np.log(plot_freq_max)
plot_points = int(plot_points_per_decade*(log_plot_freq_max-log_plot_freq_min))

#Generate plotting frequencies
log_plot_freq = np.linspace(log_plot_freq_min,log_plot_freq_max,plot_points)

pos_plot_freq = np.exp(log_plot_freq)
plot_freq = np.concatenate((np.flip(-pos_plot_freq),pos_plot_freq))


gamma_pos = np.sum(x_pos[None,:,:]*Gaussian_Func(log_plot_freq[:,None,None]-log_f_
gamma_neg = np.sum(x_neg[None,:,:]*Gaussian_Func(log_plot_freq[:,None,None]-log_f_

gamma = np.concatenate((np.flip(gamma_neg),gamma_pos))


#ax1, ax2 = subfig.subplots(nrows=1, ncols=2)
fig, (ax1, ax2) = plt.subplots(ncols=2,dpi=300,sharey=True,gridspec_kw={'width_rat

X,Y = np.meshgrid(plot_freq,Electrode_Potential,indexing='ij')

mesh = ax2.pcolormesh(X,Y,gamma,shading='nearest', cmap='viridis')
#mesh.set_norm(SymLogNorm(linthresh=1e-2))
cbar = fig.colorbar(mesh)

ax2.set_xscale('symlog')
ax2.set_xlabel('Frequency [rad/s]')
ax2.set_xticks([-1e5,-1e3,-1e1,0,1e1,1e3,1e5])
ax2.set_ylim(0.5,1.0)

ax1.plot(Current,Electrode_Potential)
ax1.fill_betweenx(Electrode_Potential, Current_min, Current_max, alpha = 0.2)
ax1.set_ylabel('Polishing Voltage [V]')
```

```
    ax1.set_xlabel('Current [mA]')
    ax1.invert_xaxis()
    # ax1.set_xticks([0.1,0.2])
    ax1.set_yticks(np.arange(0,5,1))
    # ax1.set_xlim(0.2,0)

    fig.subplots_adjust(wspace = 0)
    #ax2.set_title(sample_serial_number)
```

7 Conclusion

This study shows that EIS measurements can be used to differentiate the eching and polishing regimes in niobium EP

8 Supplemental Information

The EIS measurements were performed using a BioLogic VSP-300 potentiostat

8.1 Data Pre-Processing

EIS data from the potentiostat is exported as a series of text files, one for each sample, containing the electrode potential, current, frequency, impedance, and run number, which indicates which voltage step the measurement was performed at. To make the data more manageable, we convert the data into a hierarchical data format (HDF5).

To accomplish this we use the numpy python library to load the text files into arrays, and the h5py library to create the HDF5 file.

```
import numpy as np
import h5py
import os
import pandas as pd
from tabulate import tabulate
```

First we create a new file to store all the data.

```
import os
import h5py
filename = 'Data/data.hdf5'
if os.path.exists(filename):
    os.remove(filename)
f = h5py.File(filename,'a')
print(f)
```

Then we create a group to contain each of the samples.

```
samples = f.require_group("samples") #
print(samples)
```

We create the metadata strings for each of the samples in the Data directory.

```
from tabulate import tabulate
import pandas as pd

d = {'Serial Number': ["S35","S68","S67","S75"],
     'Electrolyte Temperature': [21,13,13,21],
     'Nitrogen Doped': ['No','No','Yes','Yes']}

df = pd.DataFrame(data=d)

print(tabulate(df, headers="keys", showindex=False, tablefmt="orgtbl"))


import numpy as np

for i, sample in enumerate(df['Serial Number']):
    f.create_dataset('samples/'+sample+'/electrolyte_temperature', data=np.asarray(df[
    f.create_dataset('samples/'+sample+'/nitrogen_doped', data=np.asarray(df['Nitrogen
```

We load the text files containing the EIS data

```
import os
import re

# get the list of files and directories in the raw data  directory
directory = 'Data/text_files/'
```

```
text_files = os.listdir(directory)

# get the samples from the data file
samples = f['samples']

# add the data to each sample
for file in text_files:
    for sample in samples:
if file.startswith(sample):
    data = pd.read_csv(directory+file,delimiter='\t')
    data = data.drop('Unnamed: 6', axis=1)

    Ewe = np.asarray(data['<Ewe>/V'])
    I = np.asarray(data['<I>/mA'])
    ImZ = np.asarray(-data['-Im(Z)/Ohm'])
    ReZ = np.asarray(data['Re(Z)/Ohm'])
    freq = np.asarray(data['freq/Hz'])
    cycle_number = np.asarray(data['cycle number'],dtype='uint8')

    nonzero_measurements = np.where(freq!=0)

    Ewe = Ewe[nonzero_measurements]
    I = I[nonzero_measurements]
    ImZ = ImZ[nonzero_measurements]
    ReZ = ReZ[nonzero_measurements]
    freq = freq[nonzero_measurements]
    cycle_number = cycle_number[nonzero_measurements]

    data_group = samples[sample].create_group('data',track_order=True)

    for cycle in range(cycle_number.min(),cycle_number.max()):

cycle_measurements = np.where(cycle_number == cycle)

data_group.create_dataset(str(cycle)+'/Ewe',data=np.asarray(Ewe[cycle_measurements]))
data_group.create_dataset(str(cycle)+'/I',data=np.asarray(I[cycle_measurements]))
data_group.create_dataset(str(cycle)+'/ImZ',data=np.asarray(ImZ[cycle_measurements]))
data_group.create_dataset(str(cycle)+'/ReZ',data=np.asarray(ReZ[cycle_measurements]))
data_group.create_dataset(str(cycle)+'/freq',data=np.asarray(freq[cycle_measurements]))
```