

TBBL Impedance Data Analysis

By: Jian Ruan

Time: June 9, 2022

Goal: Automate IDE impedance data analysis process. Avoid copy & paste!

Follow these steps for Accelerated Aging Testing!

Data Collection (about 3min / device)

Step 1: Setup the IDE in the Ferrari Cage. (1min/IDE)

Step 2: Turn on Autolab and connect to the lab Dell PC.

Step 3: Open Nova2.1.4 software with the procedure "**FRA MUX 1ch 50mV 10k-10Hz - automated**".

Step 4: Change the export file-name to corresponding IDE.

E.g:

IDE-12-8-m means IDE-12, 12 μ m, mutual.

IDE-16-16-s means IDE-16, 16 μ m, shunt

Step 5: Click on the run button and wait for the test result. (1min/IDE)

Step 6: Nova2.1.4 will auto-generate a csv file for each IDE in the ASCII format. File location: Desktop/IDE-data.

Step 7: Edit the "**date.csv**" file to keep track of the experiment dates.

Step 8: Clean the IDE and put it back to the Lab Armor.

Data Analysis

Step 1: Open the Jupyter Notebook "**[20220609]TBBL-Impedance Data Analysis**".

Step 2: Make sure you change to the right file address so Jupyter can access the impedance data.

Step 3: Run Jupyter Notebook and get your awesome data graphs!

```
In [1]: # Step 0: Import Library & Check system requirement
import numpy as np
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
```

```
import datetime

mpl.rcParams['axes', labelsize=16, titlesize=20)

# Python Version Requirement
print("Matplotlib Version", mpl.__version__) #> 3.0.0
print("Seaborn Version", sns.__version__, "\n") #> 0.9.0

# Step 1: Define IDE as a class for recurring use.
class IDE:
    def __init__(self, name, miu_m, capa, data_format):
        """
            Built-in function to initialize the IDE object
            self: no need to put in parameters, built-in structures in Python Class
            name: a string, number of the IDE
            miu_m: a list, tested channels (E.g: [2,4,8,16])
            capa: a list, type of capacitance ("m": mutual, "s": shunt)
            data_format: a string, file format ("csv")
        """
        # Initialize IDE object
        self.name = name
        self.miu_m = miu_m
        self.capa = capa
        self.data_format = data_format
        self.data_list = []
        self.num_channels = 0

    def setData(self):
        """
            Auto-generate a list of data file for each IDE
            E.g:
                IDE_20 = IDE(20, [2,16], ["m","s"], "csv")
                Will generate
                data_list = [
                    "20-2-m.csv",
                    "20-2-s.csv",
                    "20-16-m.csv",
                    "20-16-s.csv"]
                Totally 4 data .csv files for IDE_20.
        """
        for m in self.miu_m:
            for c in self.capa:
                file_name = str(self.name) + "-" + str(m) + "-" + c + "." + self.data_format
                self.data_list.append(file_name)

        #Number of tested channels
        self.num_channels = len(self.data_list)

    def getDataFile(self):
        return self.data_list

    def getName(self):
        return "IDE-" + str(self.name)

    def cleanData(self, df):
        # Add one row for checking if the freq is in the first row

        for idx, row, in df.iterrows():
            # idx is the index of the row
            # row is a Series: Frequency f = row[0], Impedance Z = row[1], -Phase = row[2], Resistance Rs = row[3], Rct = row[4], Capacitance C = row[5]
```

```

if "Frequency" in row[0]:
    idx_stamp = idx

if (idx - idx_stamp) > 25:
    # For our impedance testing, only the first 25 data are important.
    df = df.drop(idx)

#Export cleaned data
df.to_csv(file, index=False)

def extractImped(self, df, freq):
    """
    freq: an int representing the frequency we want to extract
    df: IDE datafile - e.g 12-8-m.csv
    """
    impe_freq = []

    for idx, row, in df.iterrows():
        # idx is the index of the row
        # row is a Series: Frequency f = row[0], Impedance Z = row[1], -Phase =
        # Resistance Rs = row[3], Rct = row[4], Capacitance C = row[5]
        if "Frequency" not in row[0]:
            if float(row[0]) == freq:
                impe_freq.append(float(row[1]))
    return impe_freq

def plotImpedFreq(self, file, df, freq_list):
    # Plot impedance change with respect to time
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(21, 7), constrained_layout=True)
    ax1.set_xlabel("T (Day)", fontsize=20)
    ax1.set_ylabel("Z (\u03a9)", fontsize=20)
    ax1.grid(color='lightgrey', linestyle='-', linewidth=0.5)

    ax2.set_xlabel("T", fontsize=20)
    ax2.set_ylabel("Z / Z0", fontsize=20)
    ax2.grid(color='lightgrey', linestyle='-', linewidth=0.5)

    impe_freq_1K = []

    for freq in freq_list:
        impe_freq = self.extractImped(df, freq)
        if freq == 1000:
            impe_freq_1K = impe_freq
        ax1.plot(impe_freq, label = str(freq) + "Hz", marker='o')

    #1000Hz is our reference freq for failure criteria
    if 'm' in file:
        start_wet = impe_freq_1K[1]
    elif "s" in file:
        start_wet = impe_freq_1K[0]

    ratio = []

    for imped in impe_freq_1K:
        ratio.append(imped / start_wet)

    ax2.plot(ratio, label = "1000 Hz", marker='o')
    ax2.axhline(y = 1, label = "1", color = 'r', linestyle='dotted')
    ax2.axhline(y = 1.2, label = "1.2", color = 'b', linestyle='dotted')
    ax2.axhline(y = 0.33, label = "0.33", color = 'g', linestyle='dotted')

    ax1.legend(loc='upper right', fontsize = 20)

```

```

ax2.legend(loc='upper right', fontsize = 20)

plt.show()
print('\n \n')

def getGraph(self, IDE_name, df, df_date, date_idx, Rs, Rct, C):
    # iloc[1:27, 0] means row 1 to 27, and column 0

    date_list = df_date.iloc[date_idx, 1:]
    #datetime.date(2022, 5, 31)

    # Create the IDE Graph Canvas (1 row, 2 coluns)
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 10), constrained_layout=True)
    fig.suptitle('IDE-' + file[-4:], fontsize = 30)

    # Set the axis scales & both axes to log scale
    ax1.set_xlim=(10, 100000), ylim=(10000, 1000000000), xscale="log", yscale="log"
    ax1.set_xlabel("F (Hz)", fontsize=20)
    ax1.set_ylabel("Z (\u03a9)", fontsize=20)
    ax1.grid(color='lightgrey', linestyle='-', linewidth=0.5)

    ax2.set_xlim=(10, 100000), ylim =(0, 90), xscale = "log")
    ax2.set_xlabel("F (Hz)", fontsize=20)
    ax2.set_ylabel("-Phase (')", fontsize=20)
    ax2.grid(color='lightgrey', linestyle='-', linewidth=0.5)

    #Create the Rs, Rct, and C canvas (1 row, 3 columns)
    fig2, (ax3, ax4, ax5) = plt.subplots(1, 3, figsize=(21, 7), constrained_layout=True)
    ax3.set_xlabel("Index", fontsize=20)
    ax3.set_ylabel("Rs (\u03a9)", fontsize=20)
    ax3.grid(color='lightgrey', linestyle='-', linewidth=0.5)

    ax4.set_xlabel("Index", fontsize=20)
    ax4.set_ylabel("Rct (\u03a9)", fontsize=20)
    ax4.grid(color='lightgrey', linestyle='-', linewidth=0.5)

    ax5.set_xlabel("Index", fontsize=20)
    ax5.set_ylabel("C (F)", fontsize=20)
    ax5.grid(color='lightgrey', linestyle='-', linewidth=0.5)

    i = 0
    j = 0
    day_idx = 0

    #Distinguish mutual vs shunt (mutual has dry, shunt doesn't)
    if date_list[0] == "dry":
        start = date_list[1]
    else:
        start = date_list[0]

    start = self.getDateime(start)

    day_actual_list = []

    while j < len(df):
        i += 1 # ith is Title, Data starts from i+1
        j = i + 25
        x_freq = df.iloc[i:j,0].astype(float) #frequency
        y_imped = df.iloc[i:j,1].astype(float) #impedance
        y_phase = df.iloc[i:j,2].astype(float) #-phase
        date = date_list[day_idx]

        if date == "dry":

```

```

        prefix = "Day-0"
        day_actual_list.append(0)

    else:
        date = self.getDatetime(date)
        day_diff = date - start
        day_actual = self.getActualDates(day_diff.days, 70.5, 37.0)

        prefix = "Day-" + str(1 + day_diff.days) + "(" + str(1 + day_actual)
        date = date_list[day_idx]

        day_actual_list.append(1 + day_actual)

    # Graph 1: Impedance Z(Ω) vs Frequency(Hz) - Labeled by dates
    ax1.plot(x_freq, y_imped, 'o-', label = prefix + ": " + date)

    # Graph 2: -Phase(°) vs Frequency(Hz) - Labeled by dates
    ax2.plot(x_freq, y_phase, 'o-', label = prefix + ": " + date)

    i = j
    day_idx += 1

    ax3.plot(day_actual_list, Rs, marker='o')
    ax4.plot(day_actual_list, Rct, marker='o')
    ax5.plot(day_actual_list, C, marker='o')

    ax1.legend(loc='upper right', fontsize = 15)
    ax2.legend(loc='upper right', fontsize = 15)
    plt.show()
    return None

def getRC(self, df, value):
    """
    Get the table of R_s over time
    """
    data = []
    idx = 0

    if value == "Rs":
        idx = 3
    if value == "Rct":
        idx = 4
    if value == "C":
        idx = 5
    # remove strings

    j = 1
    while j <= len(df):
        data.append(float(df.iloc[j, idx]))

        j = j + 26
    return data

def getActualDates(self, day_diff, room_temp, device_temp):
    # Accelerated Aging Test Formula
    exp = (room_temp - device_temp)/10.0
    factor = 2**exp
    day_real = day_diff * factor
    return int(day_real)

def getDatetime(self, m_d_y):
    """
    m_d_y is a string in month, day, year format
    """
    time_list = m_d_y.split("/")

```

```

date = datetime.date(2022, int(time_list[0]), int(time_list[1]))
return date

def addNewIDE(start, end):
    """
    Create a list of new IDE with names.
    start: the start number of IDE
    end: the end number of IDE
    """
    L = []
    for i in range(start, end):
        L.append(str(i) + "-2-m")
        L.append(str(i) + "-2-s")
        L.append(str(i) + "-4-m")
        L.append(str(i) + "-4-s")
        L.append(str(i) + "-8-m")
        L.append(str(i) + "-8-s")
        L.append(str(i) + "-16-m")
        L.append(str(i) + "-16-s")

    df_new_IDE = pd.DataFrame(L, columns=['new IDE name'])

    #Export a csv file with new IDEs
    df_new_IDE.to_csv("new_IDE.csv", index=False)
    return df_new_IDE

```

Matplotlib Version 3.5.1
Seaborn Version 0.11.2

In [2]:

```

# Step 2: Initialize tested IDEs

# Read experiment date file.
date = "date.csv"
df_date = pd.read_csv(date)

# Initialize IDEs
#Old Devices
IDE_12 = IDE(12, [8], ["m","s"], "csv")
#IDE_14 = IDE(14, [16], ["m","s"], "csv")
IDE_16 = IDE(16, [2,4,8,16],[ "m", "s"], "csv")
IDE_20 = IDE(20, [2,16], ["m", "s"], "csv")

#New Devices
IDE_22 = IDE(22, [2,4,8,16],[ "m", "s"], "csv")
IDE_26 = IDE(26, [2,4,8,16],[ "m", "s"], "csv")
IDE_27 = IDE(27, [2,4,8,16],[ "m", "s"], "csv")
IDE_28 = IDE(28, [2,4,8,16],[ "m", "s"], "csv")
IDE_31 = IDE(31, [2,4,8,16],[ "m", "s"], "csv")

# Track device results
total_channels = 0
num_failed = 0
num_good_2 = 0
num_good_4 = 0
num_good_8 = 0
num_good_16 = 0

# List of all IDEs
IDE_list = [IDE_12, IDE_16, IDE_20, IDE_22, IDE_26, IDE_27, IDE_28, IDE_31]

# IDE_12 as an example

```

```
# Iterate through one IDE
date_idx = 0

# Graph Device Data
for ide in IDE_list:
    ide.setData()
    for file in ide.getDataFile():
        df = pd.read_csv(file)

    #First clean the data to only include the 25 items
    ide.cleanData(df)

    #Read the cleaned data
    df = pd.read_csv(file)

    #Extract Rs, Rct, and C
    Rs = ide.getRC(df, "Rs")
    Rct = ide.getRC(df, "Rct")
    C = ide.getRC(df, "C")

    #Get the summary graph for each IDE
    ide.getGraph(file, df, df_date, date_idx, Rs, Rct, C)

    ide.plotImpeFreq(file, df, [1000, 100, 215.44, 10000])
    #Extract the impedance based on frequency

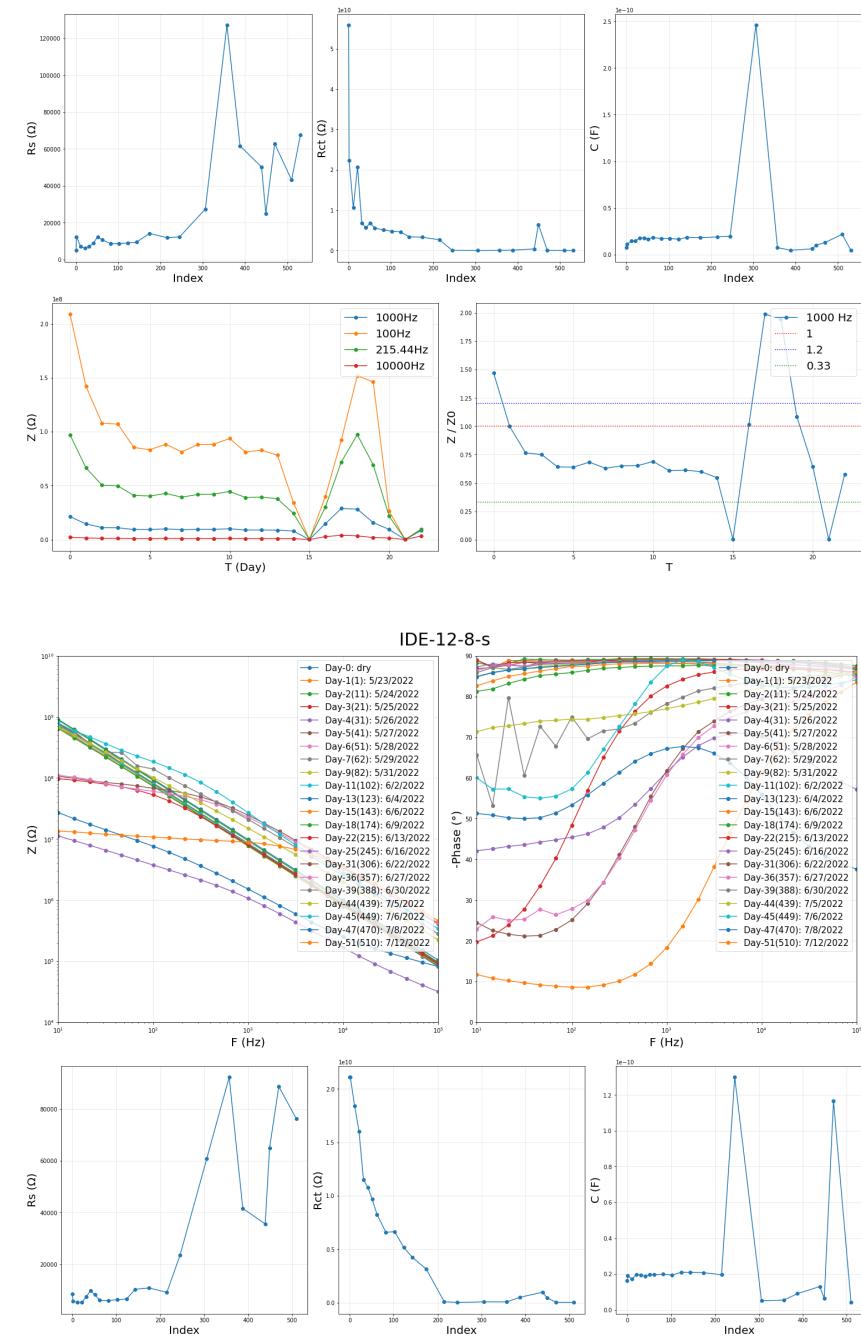
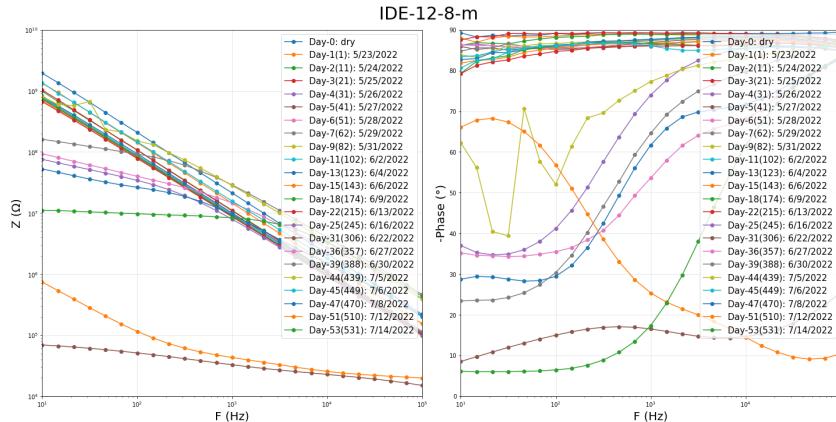
#Graph IDE failure mode - wisker plot

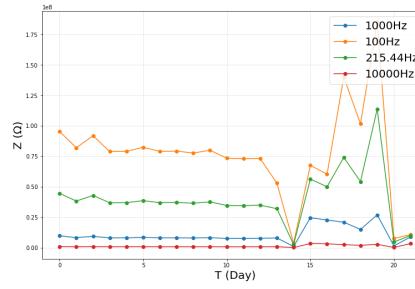
# Read IDE life span file.
life = "IDE_life_span.csv"
df_life = pd.read_csv(life)

# def graphLifeSpan(df):
#     fig1, ax1 = plt.subplots()
#     ax1.set_title('IDE Life span')
#     ax1.boxplot(df)
#     return None

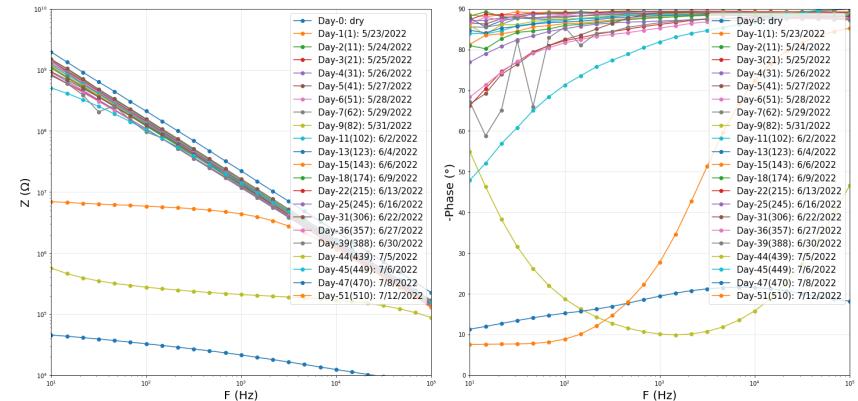
# graphLifeSpan(df_life)

# Extract the data for only 10000, 1000, 215.44, 100 Hz
```

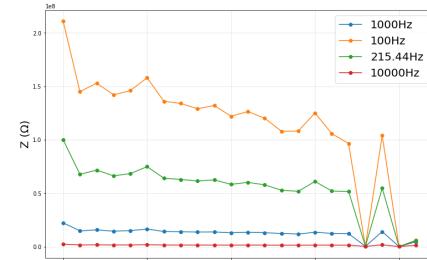
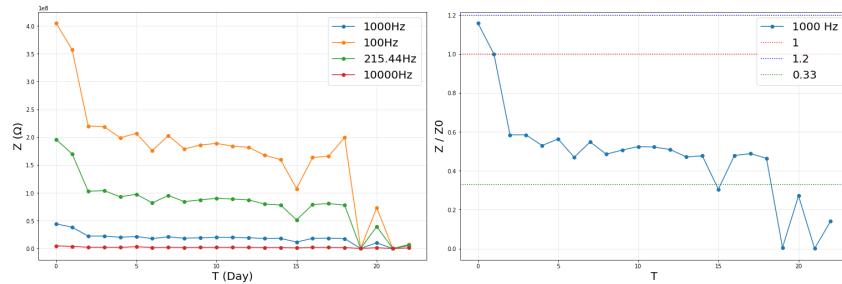
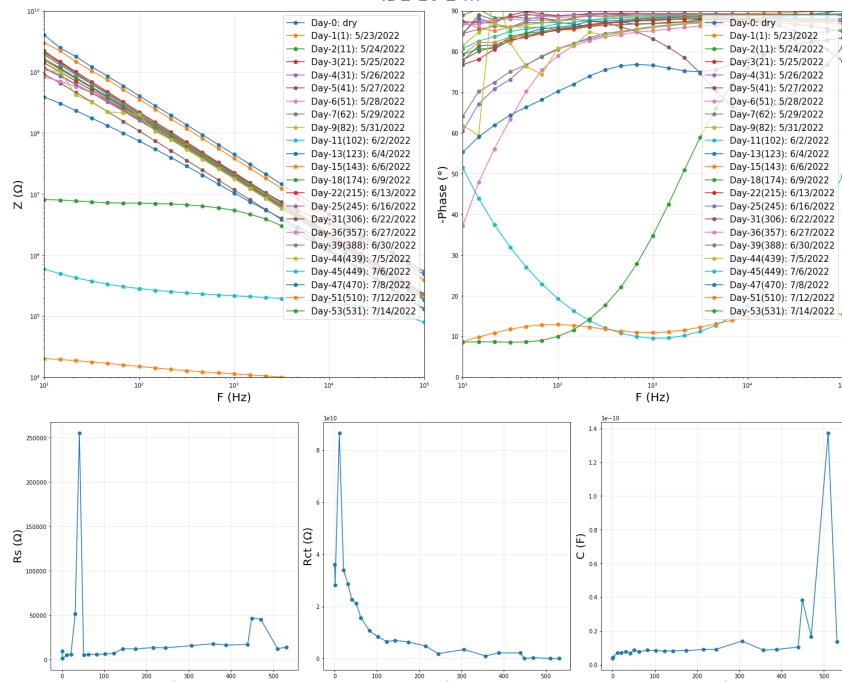


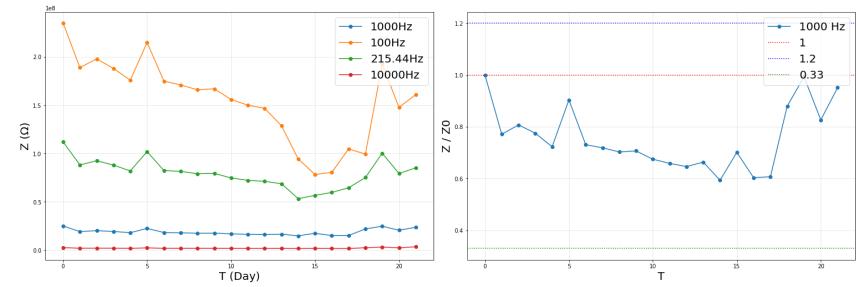
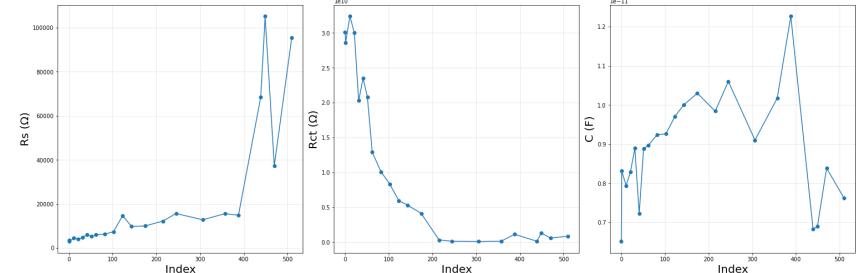
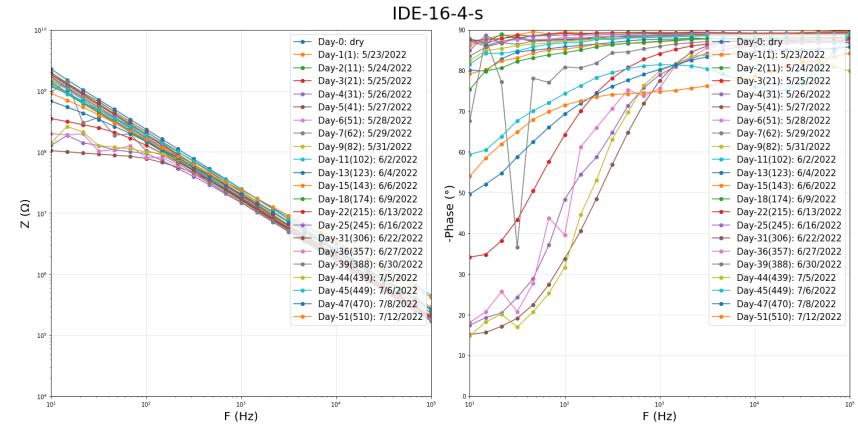
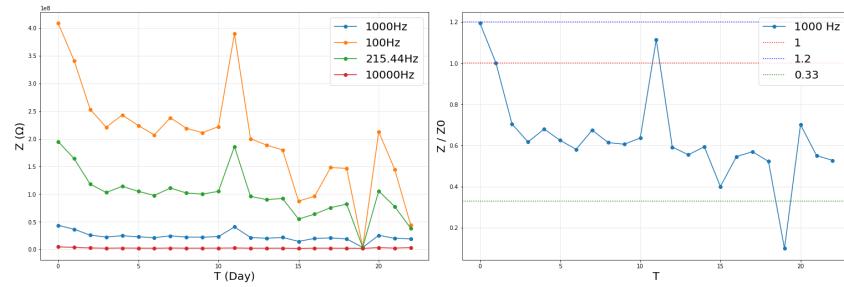
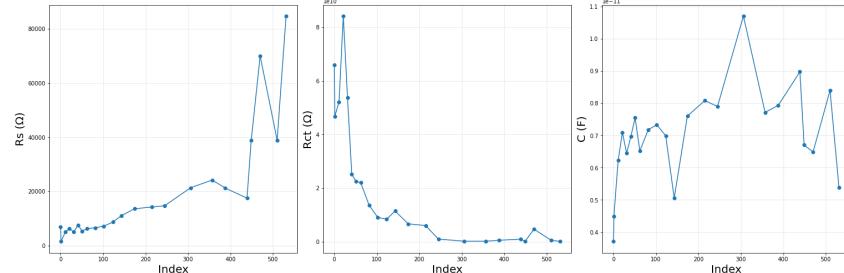
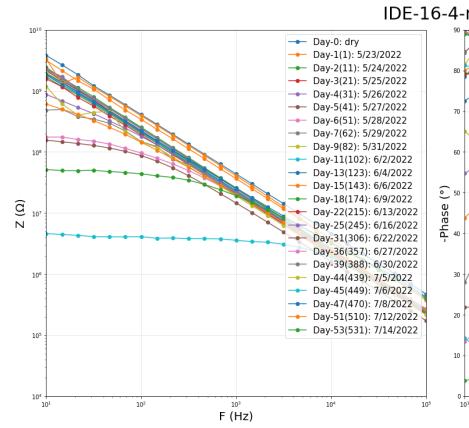


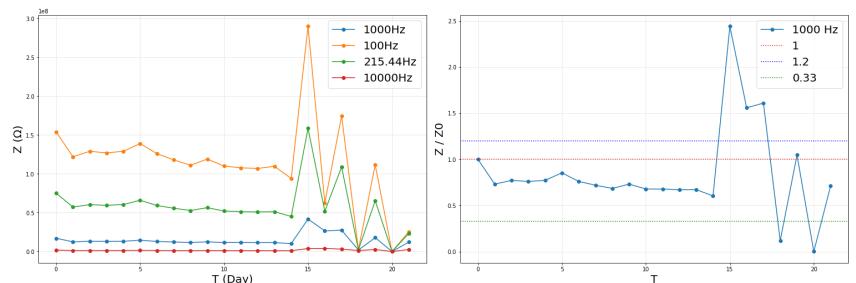
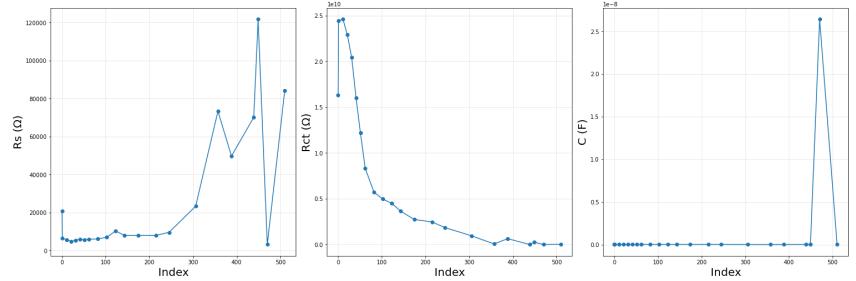
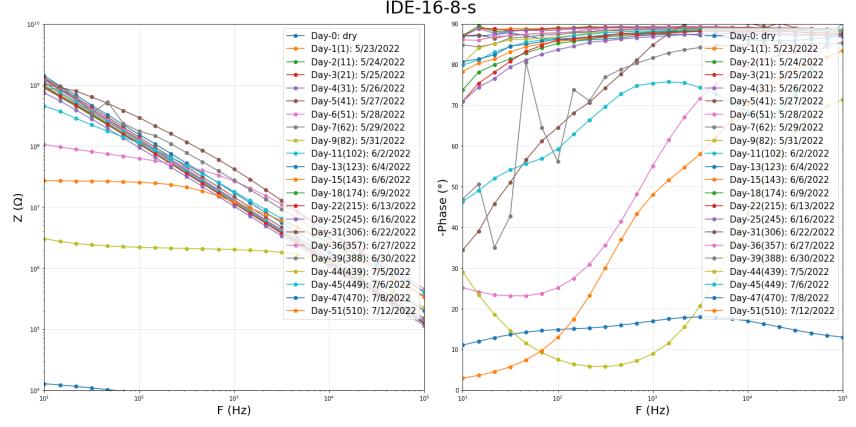
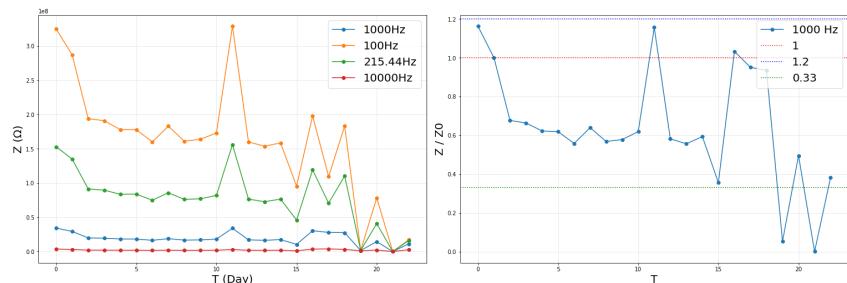
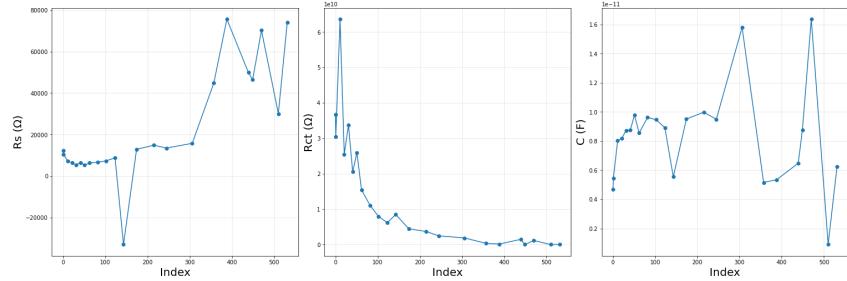
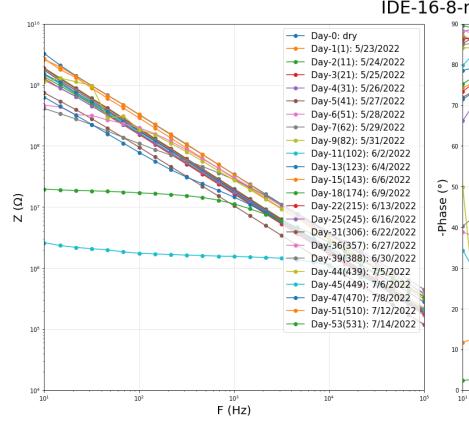
IDE-16-2-s



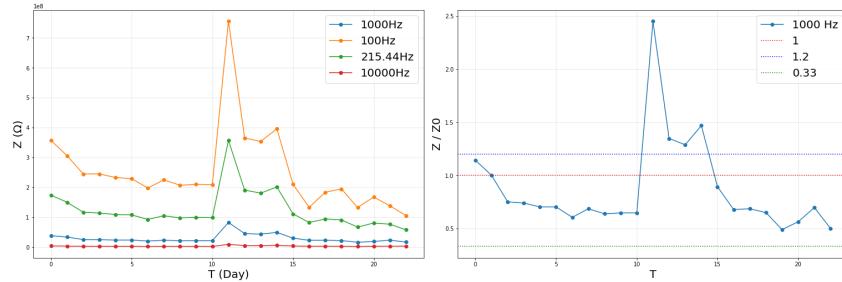
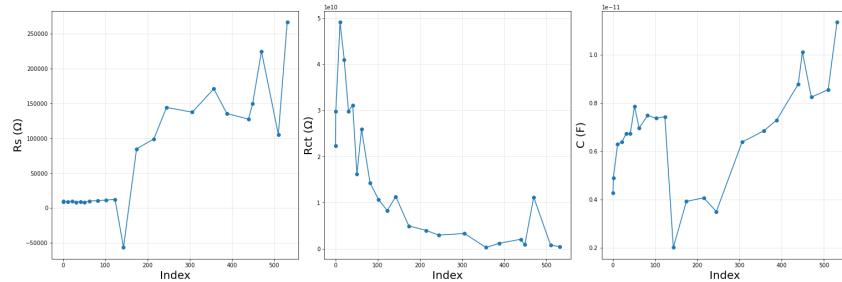
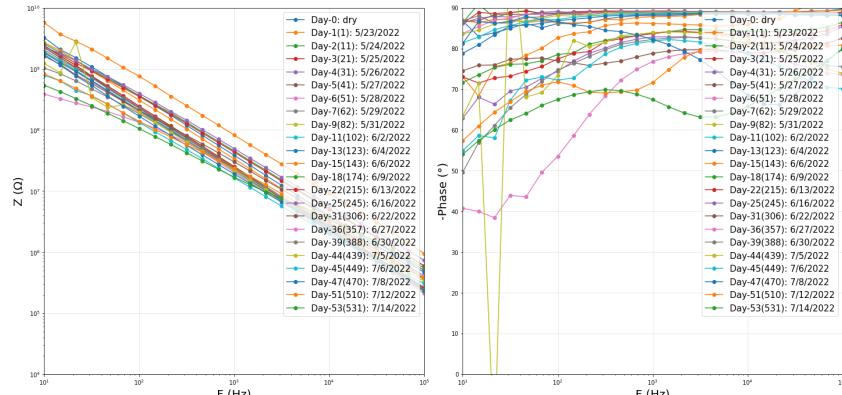
IDE-16-2-m



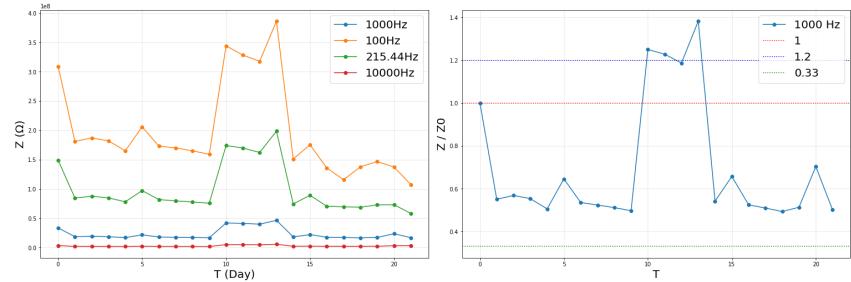
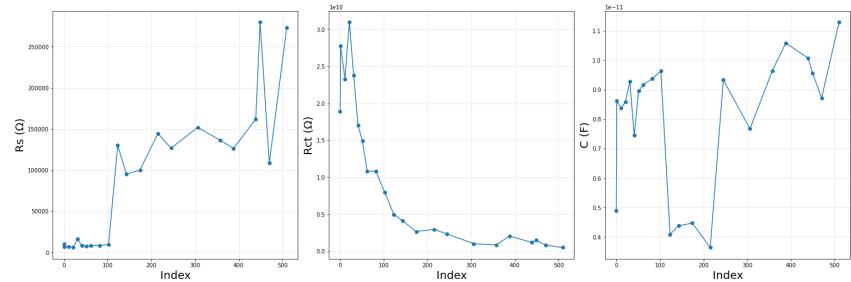
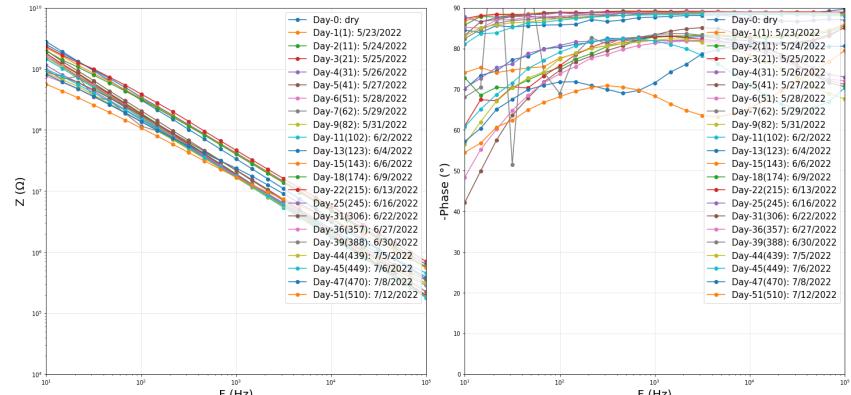




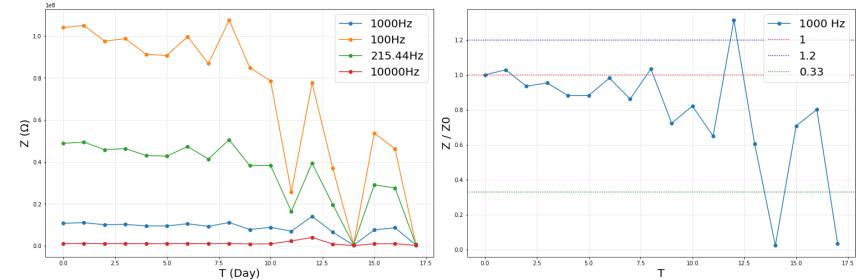
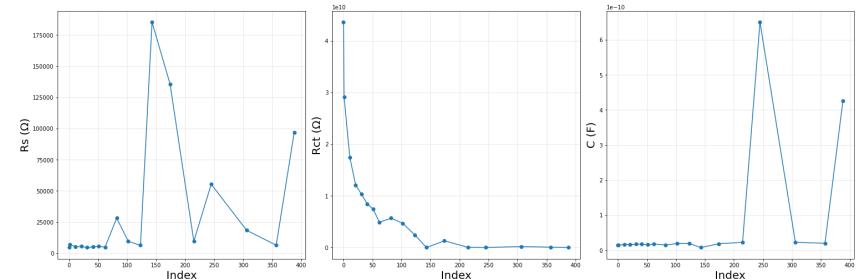
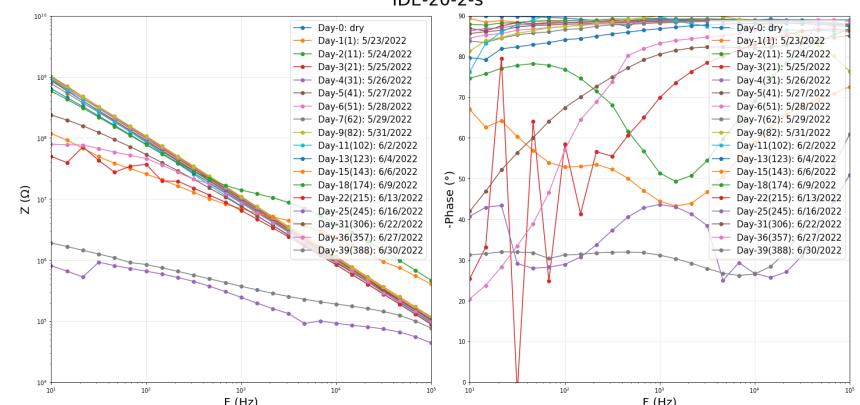
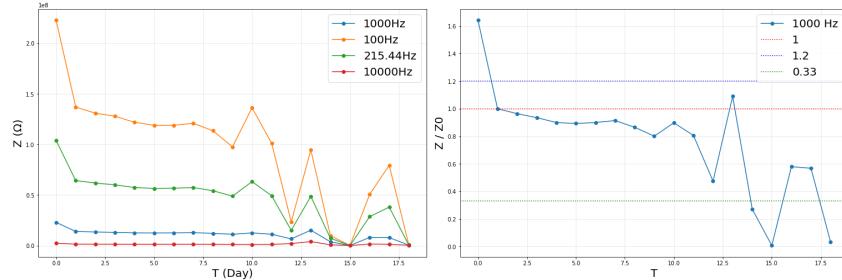
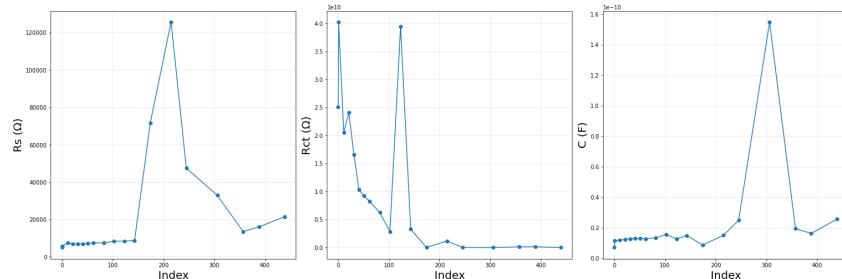
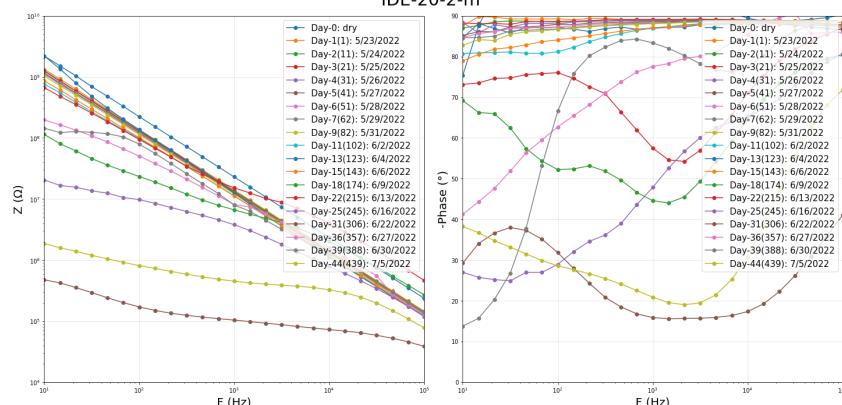
IDE-16-16-m



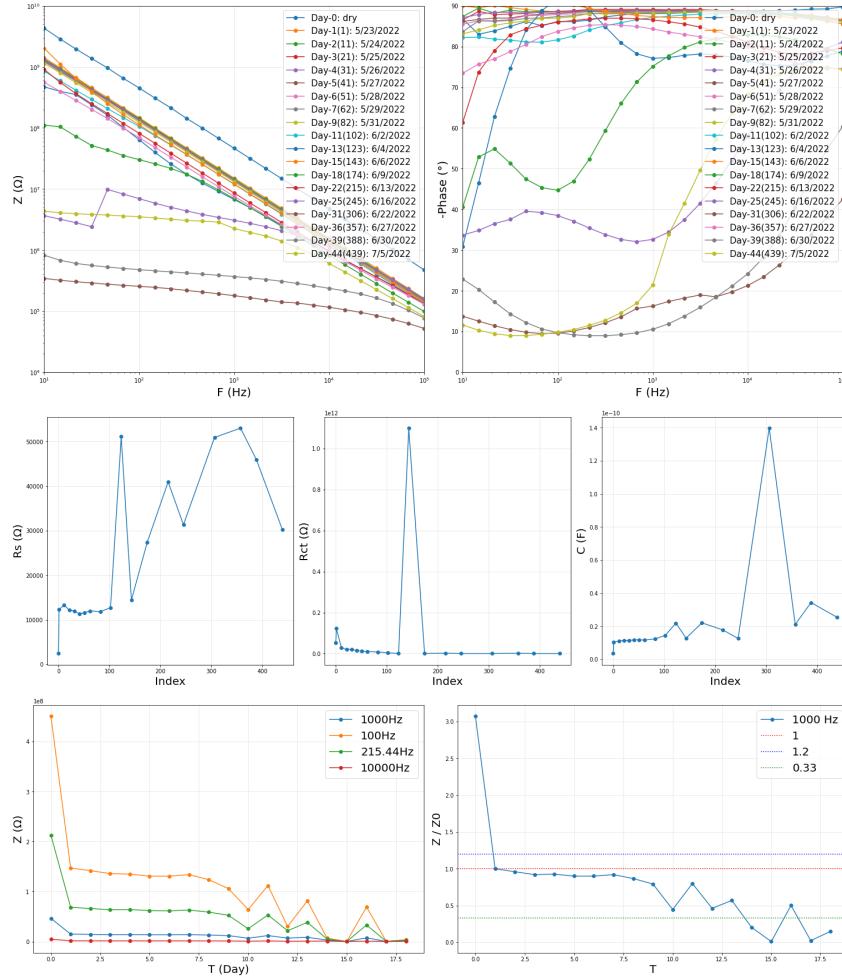
IDE-16-16-s



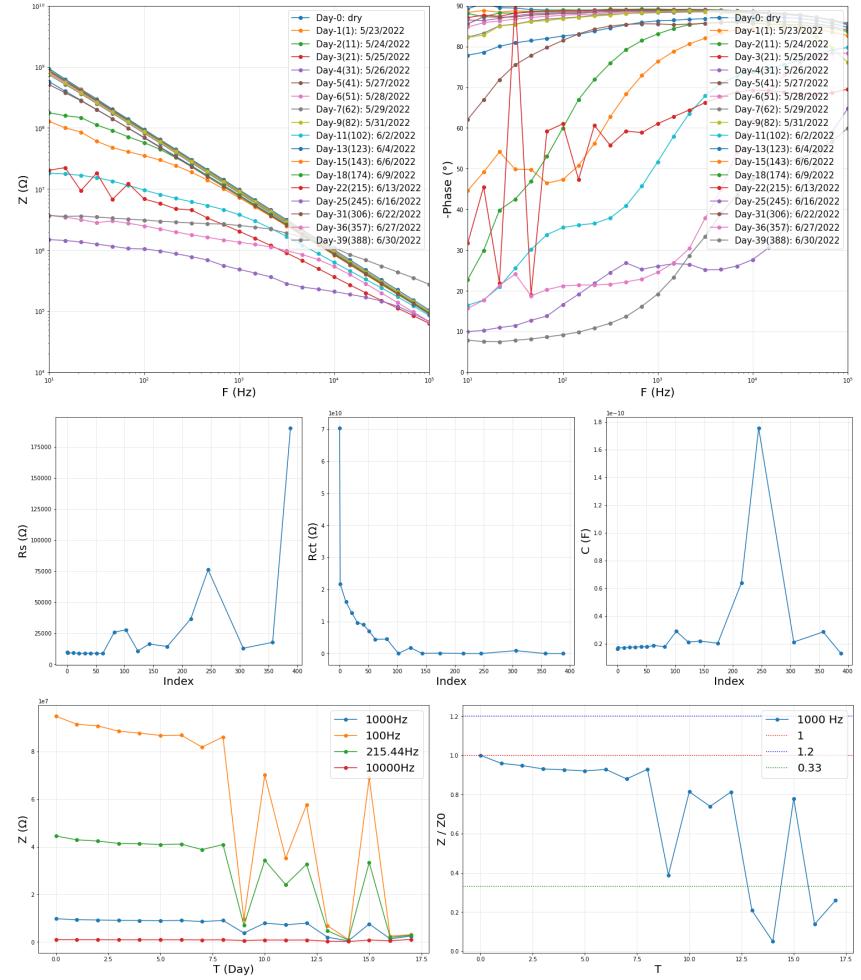
IDE-20-2-m



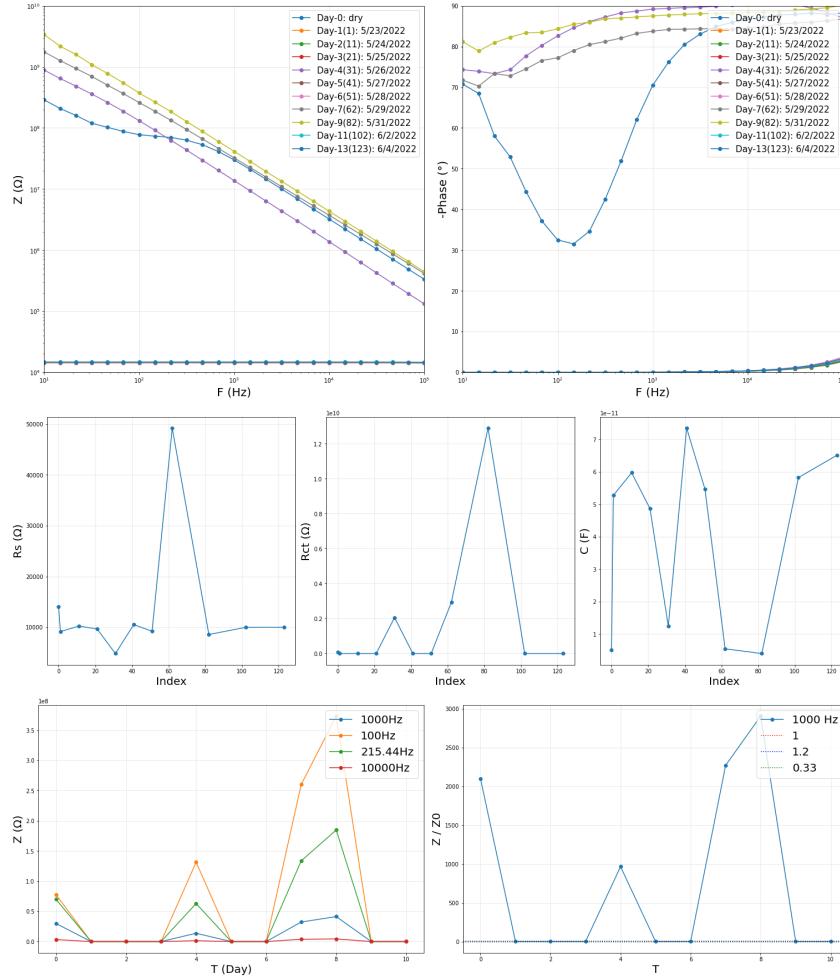
IDE-20-16-m



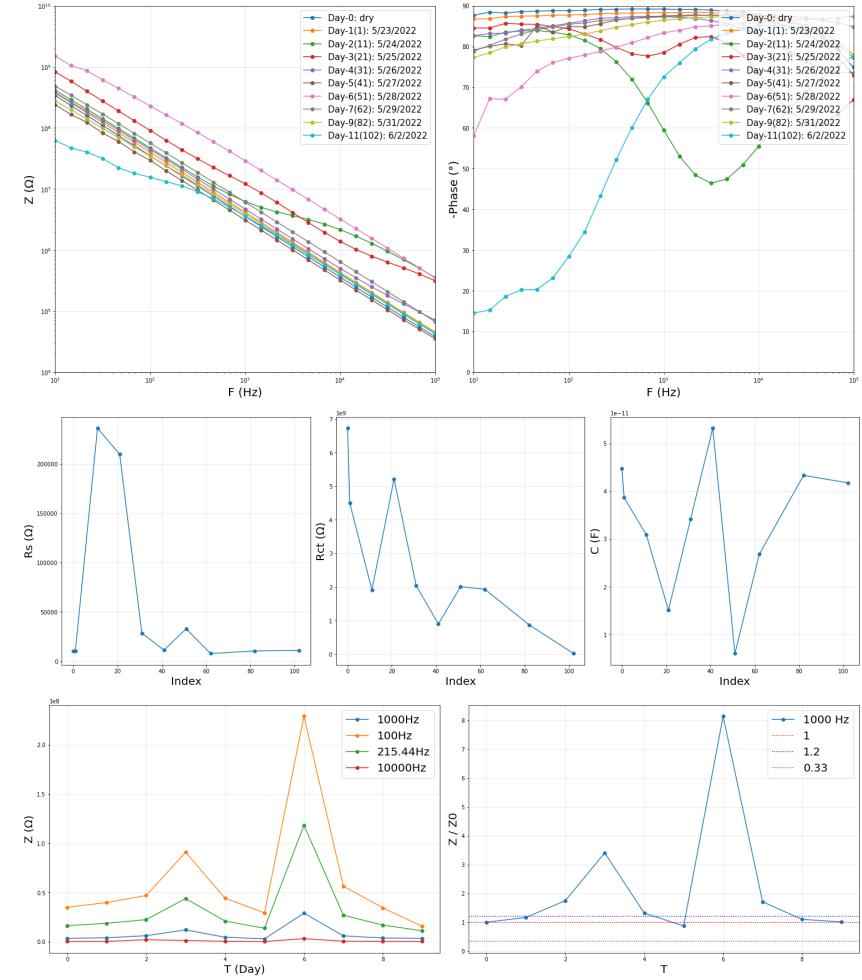
IDE-20-16-s

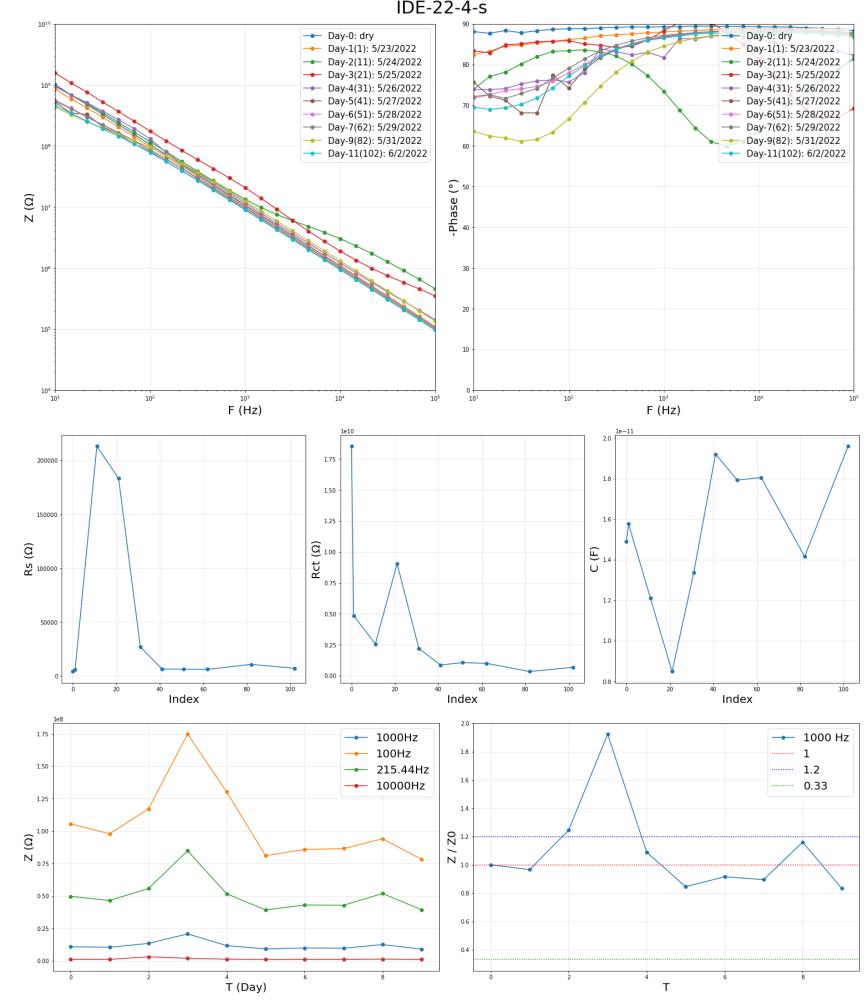
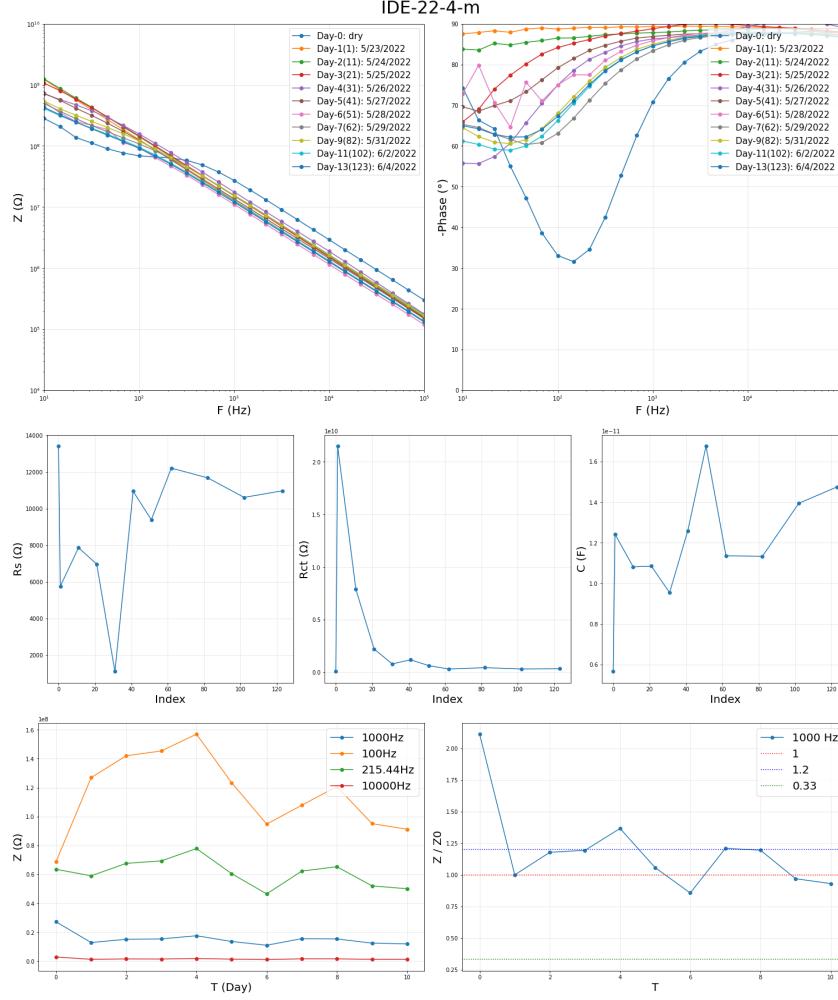


IDE-22-2-m

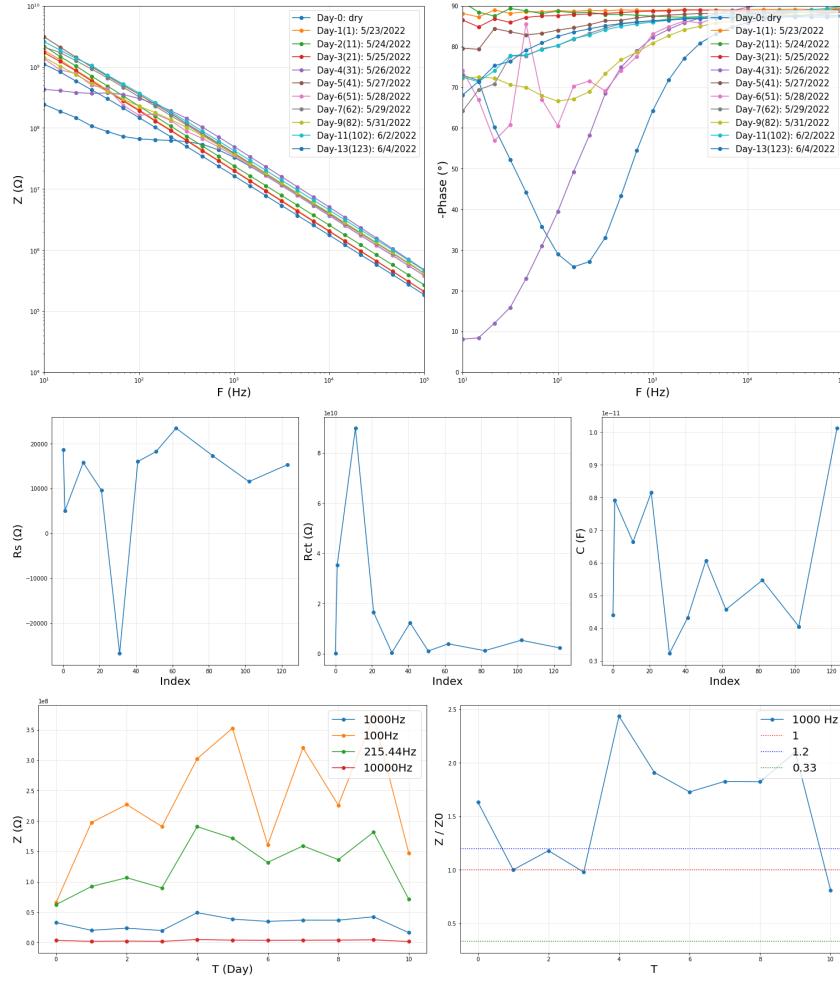


IDE-22-2-s

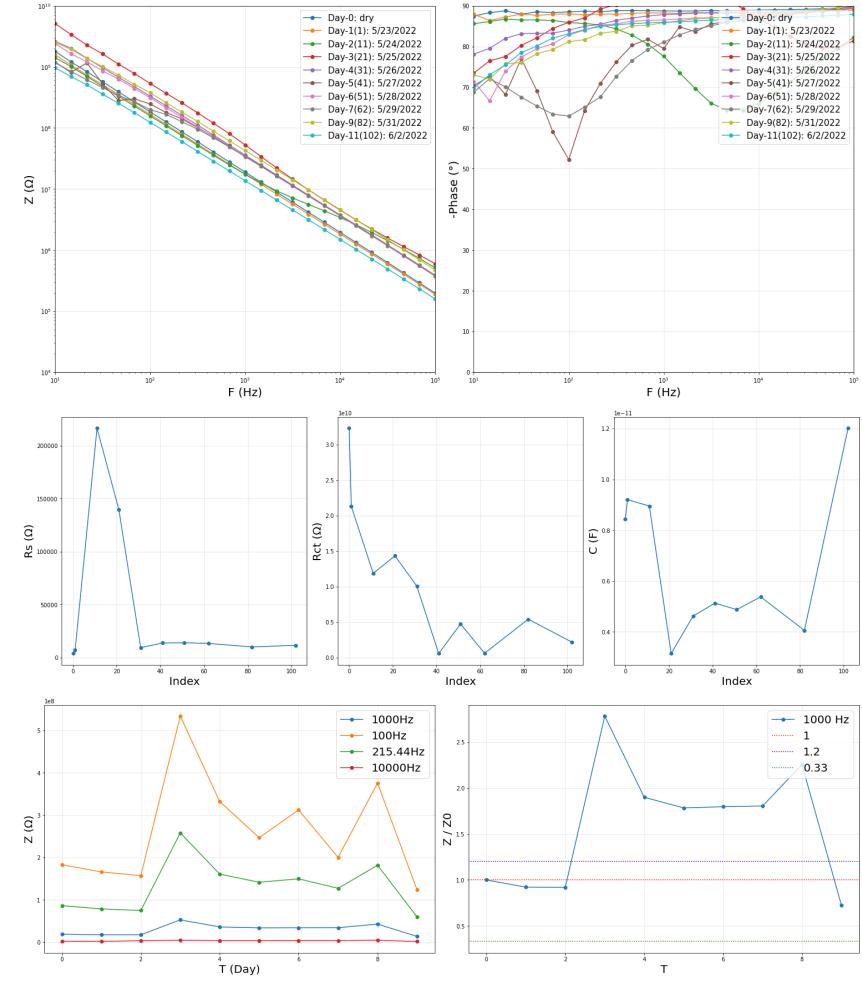


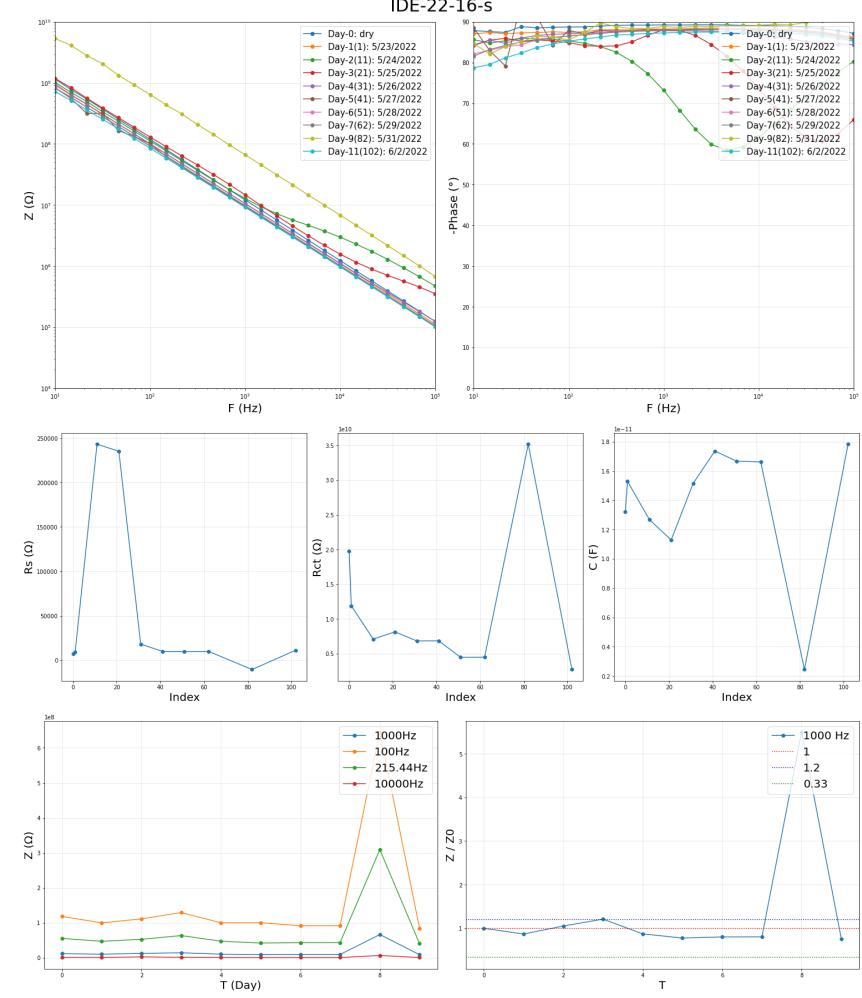
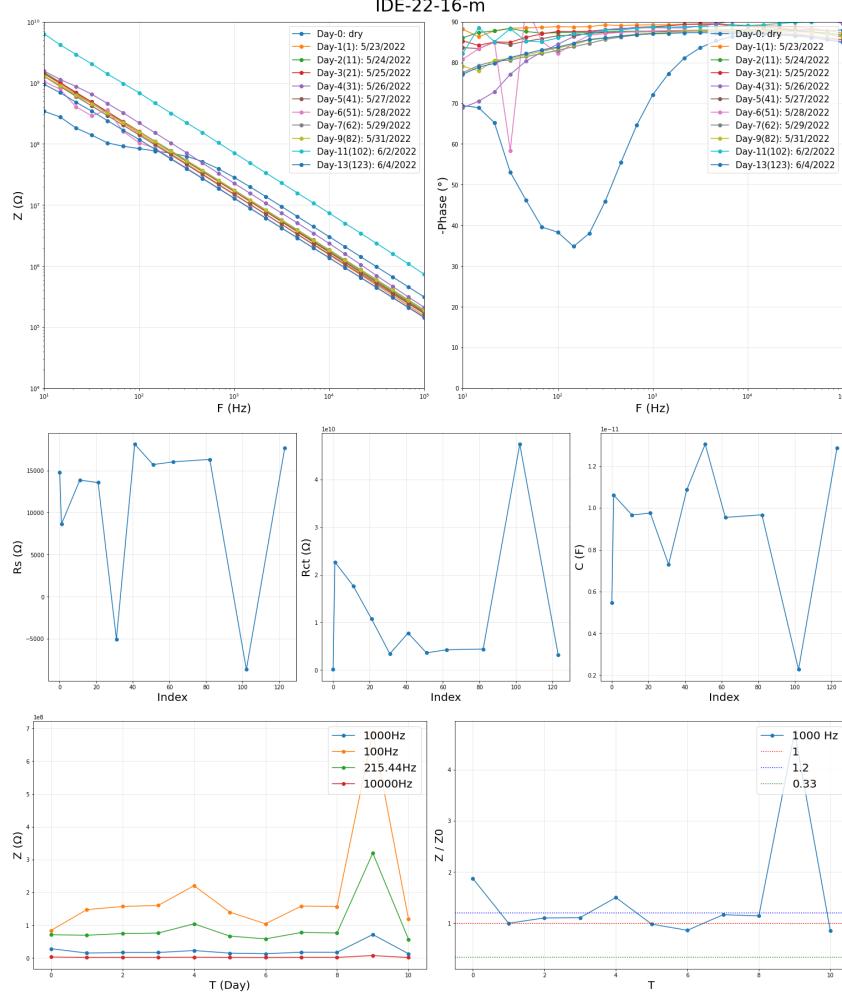


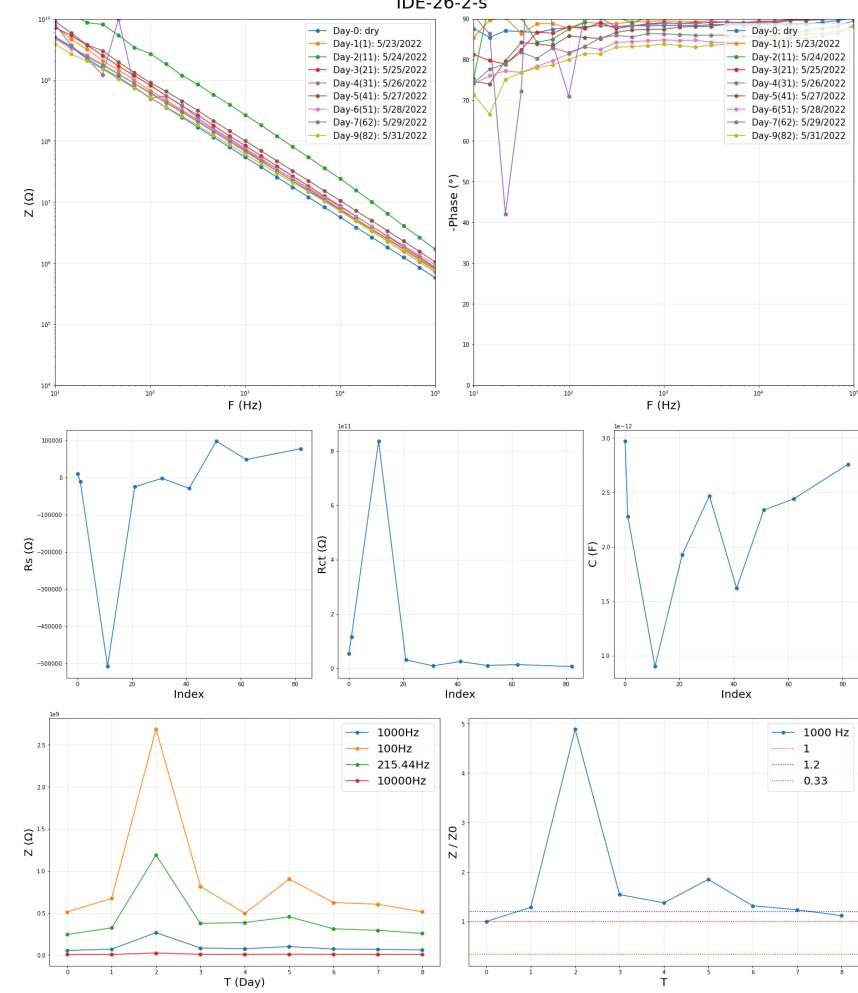
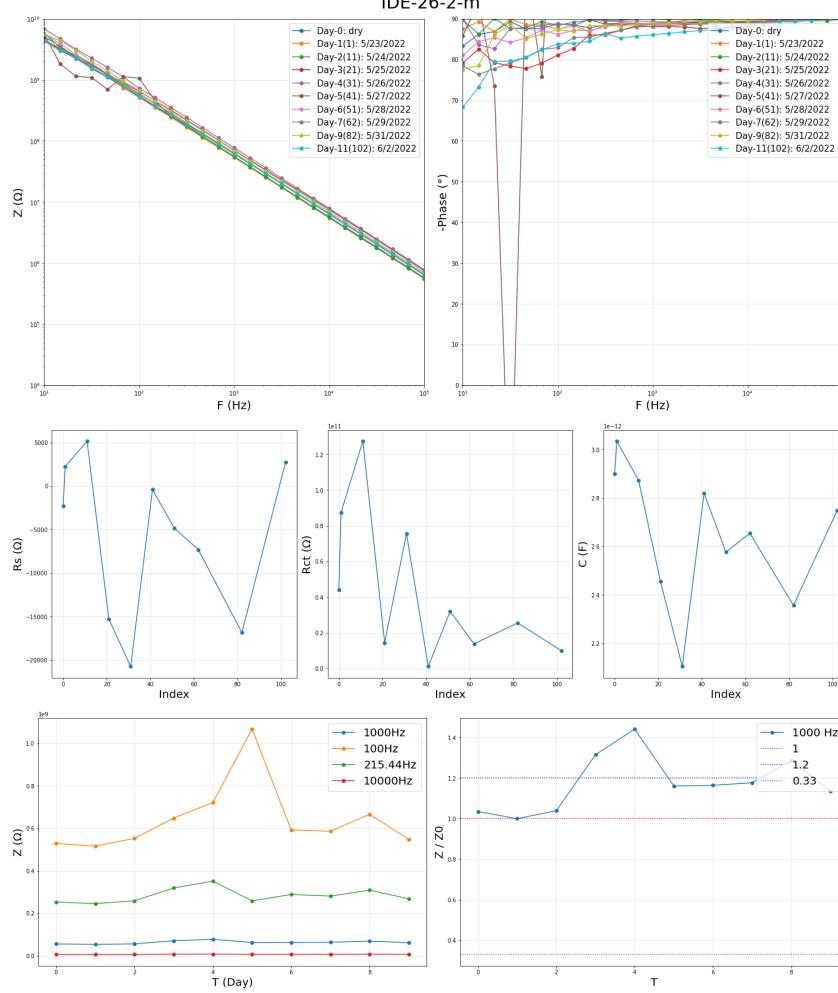
IDE-22-8-m

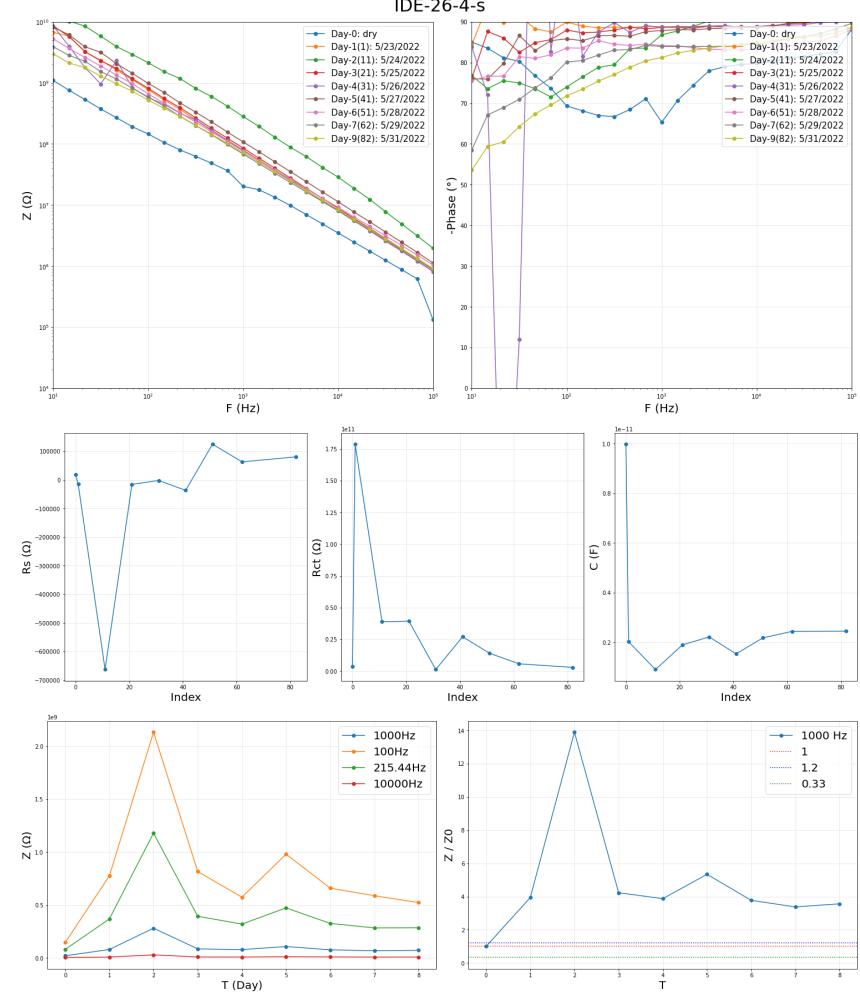
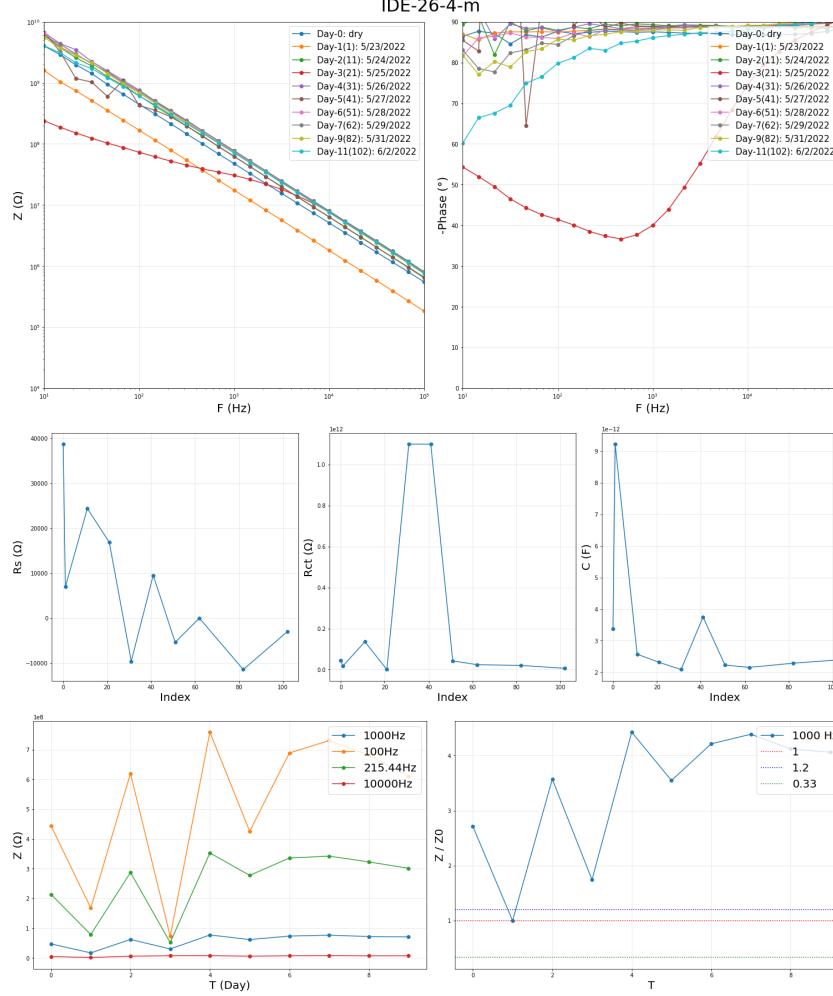


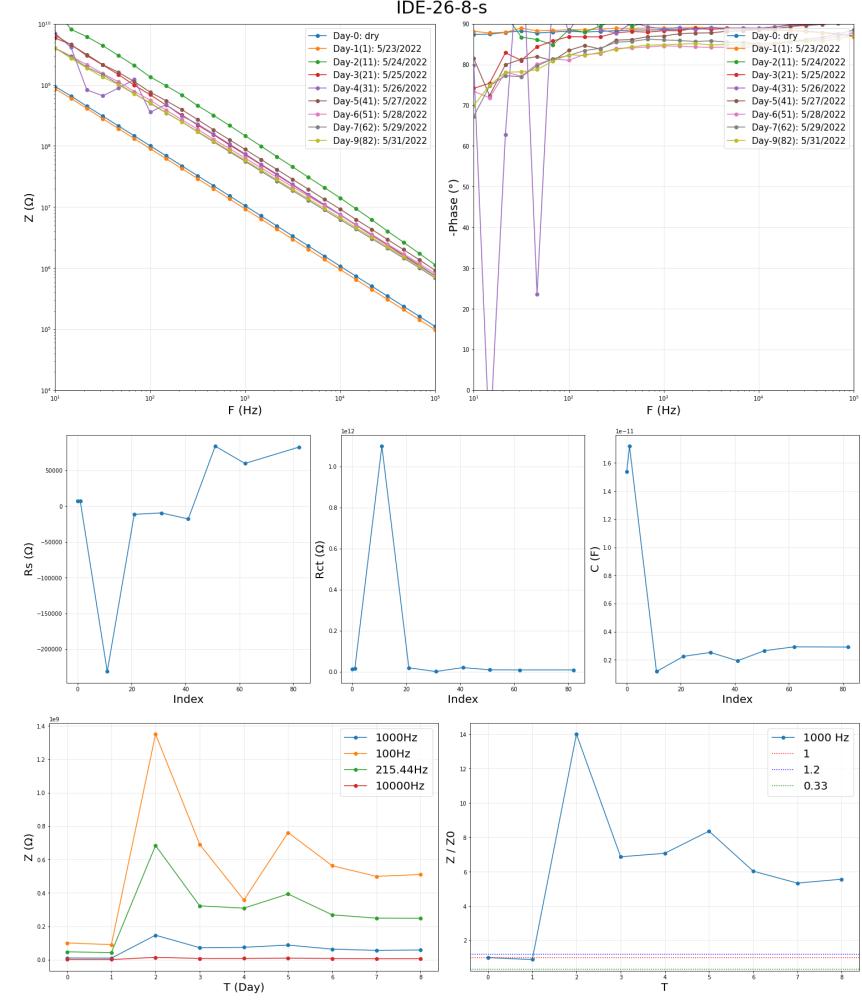
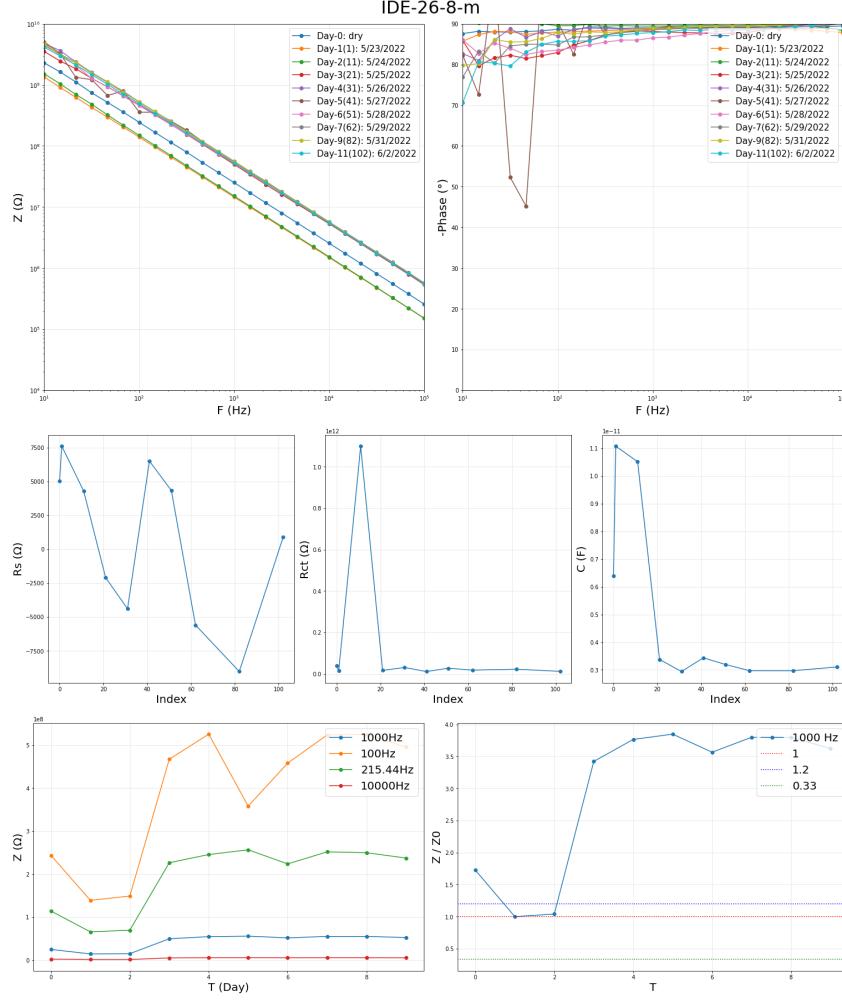
IDE-22-8-s

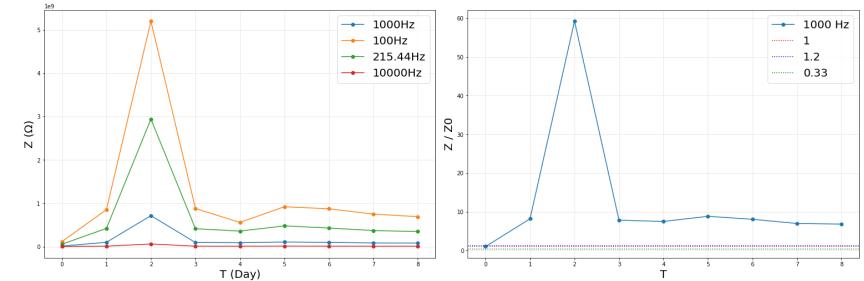
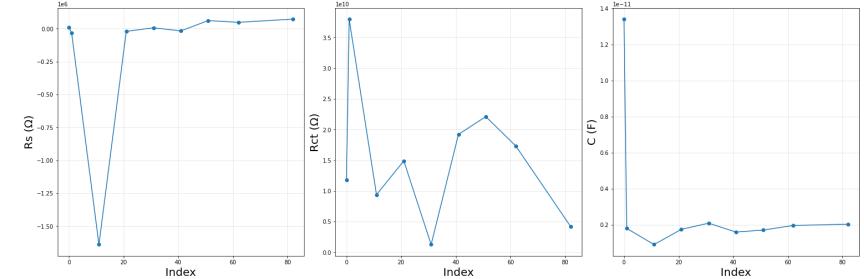
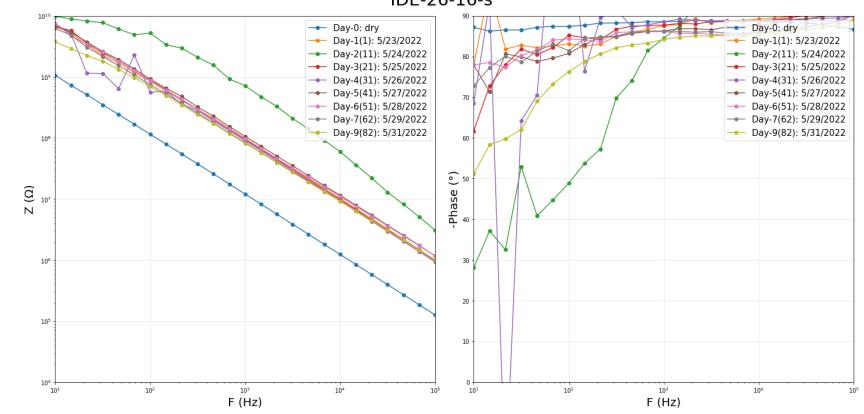
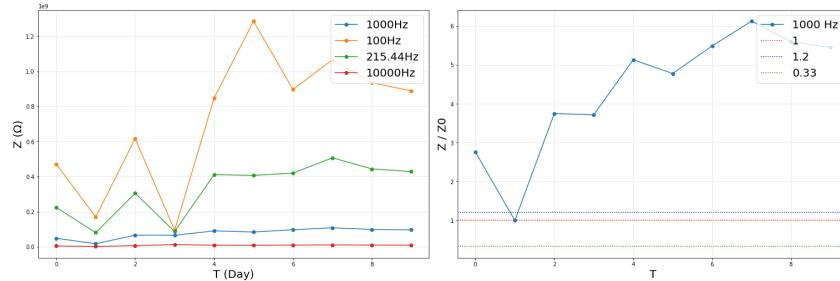
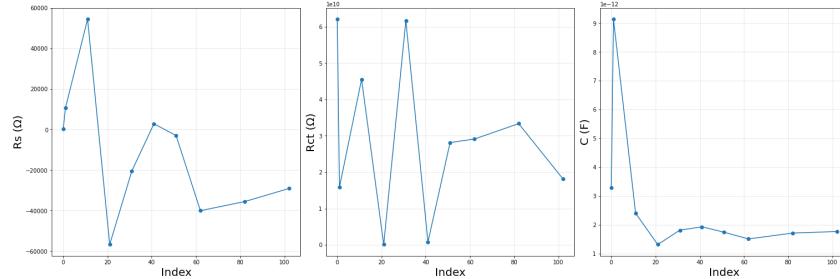
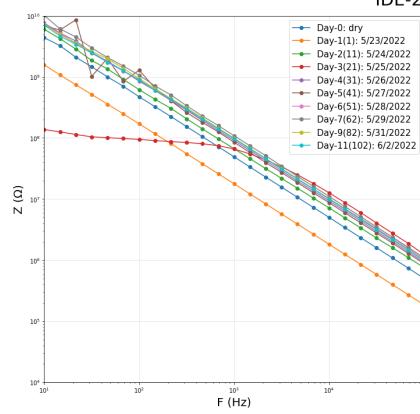


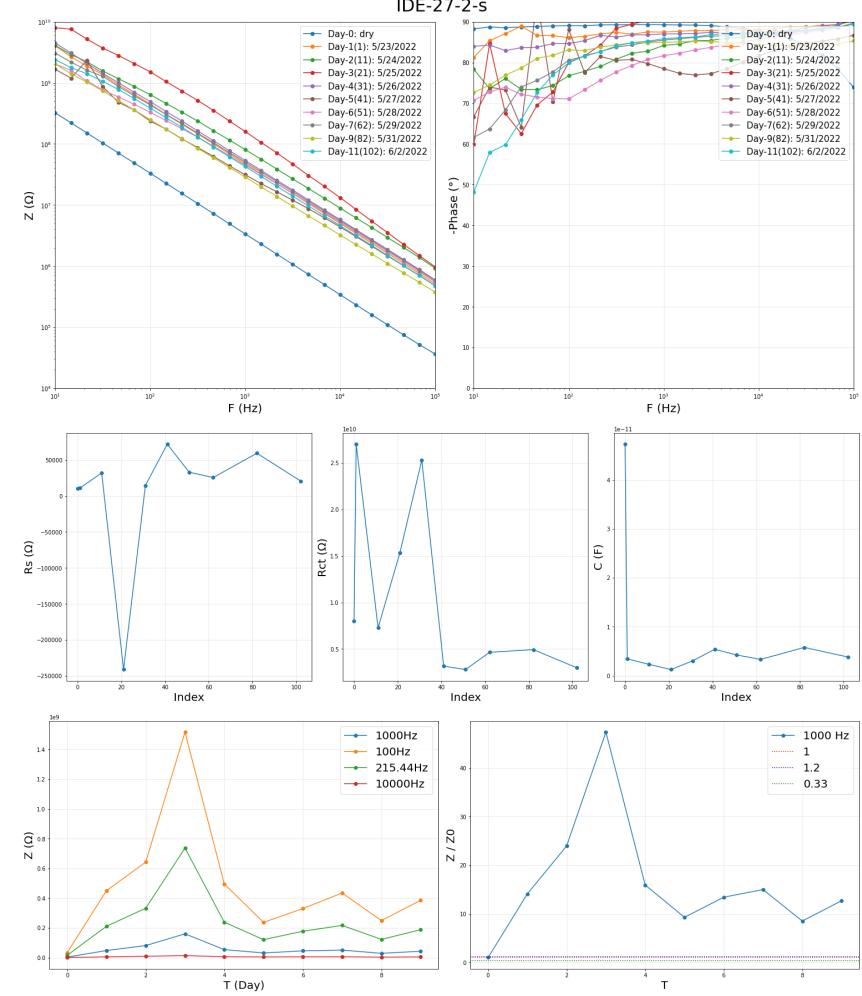
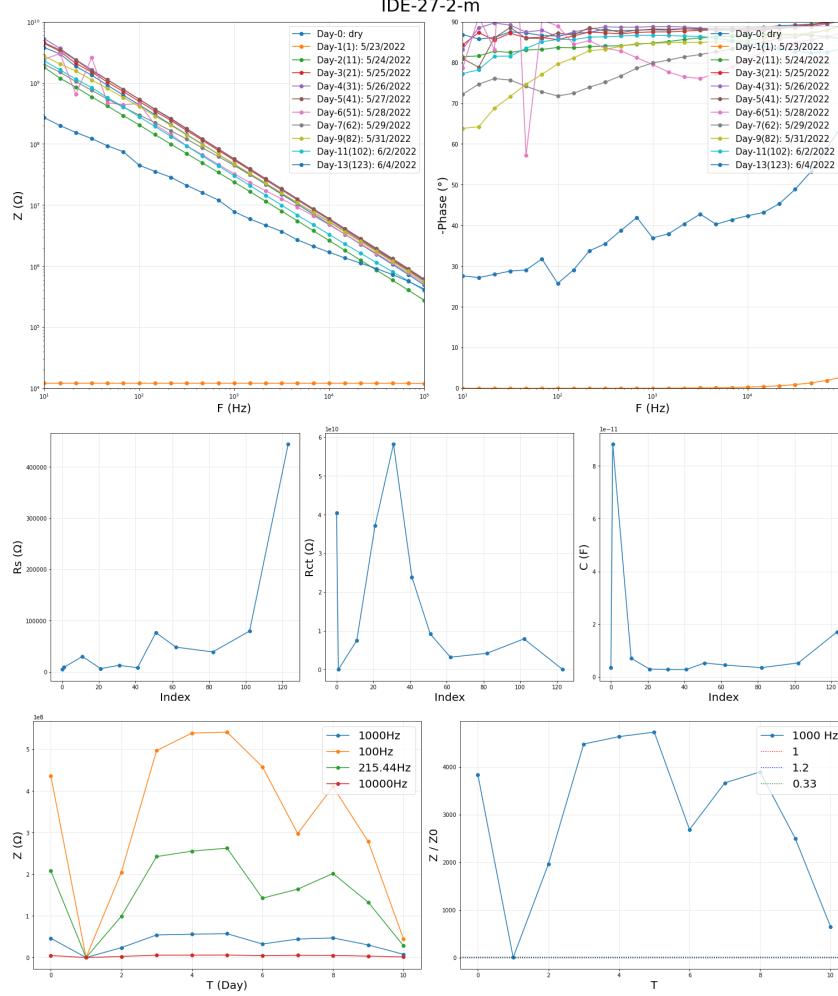




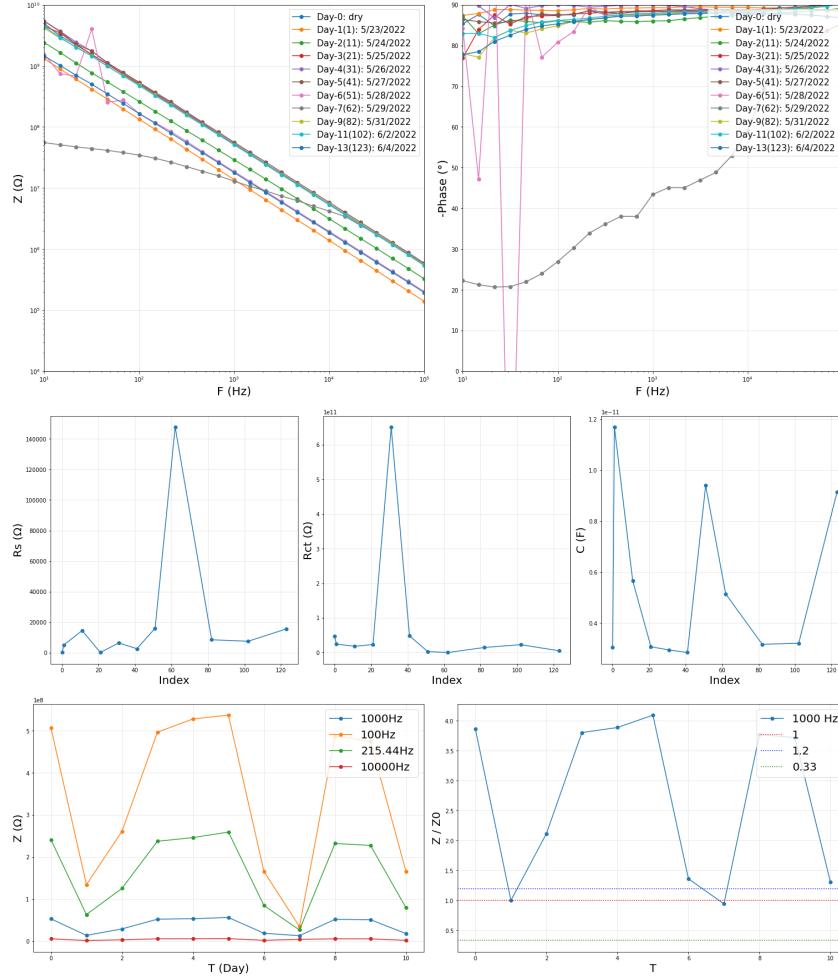




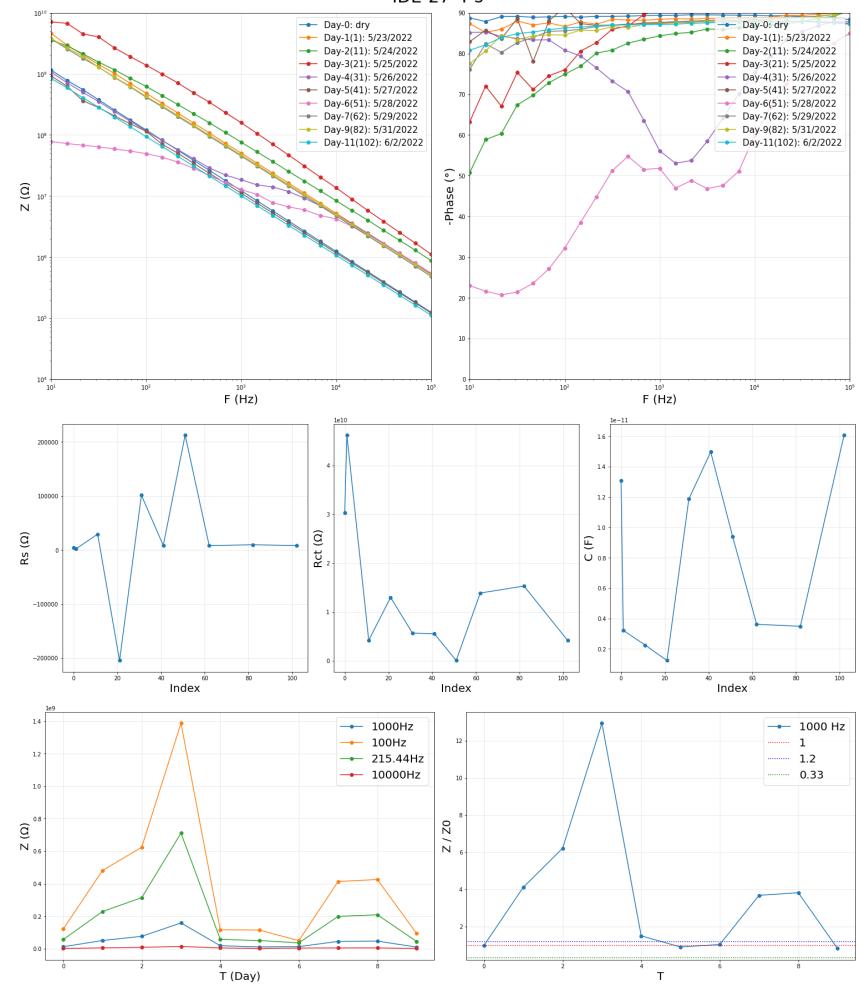




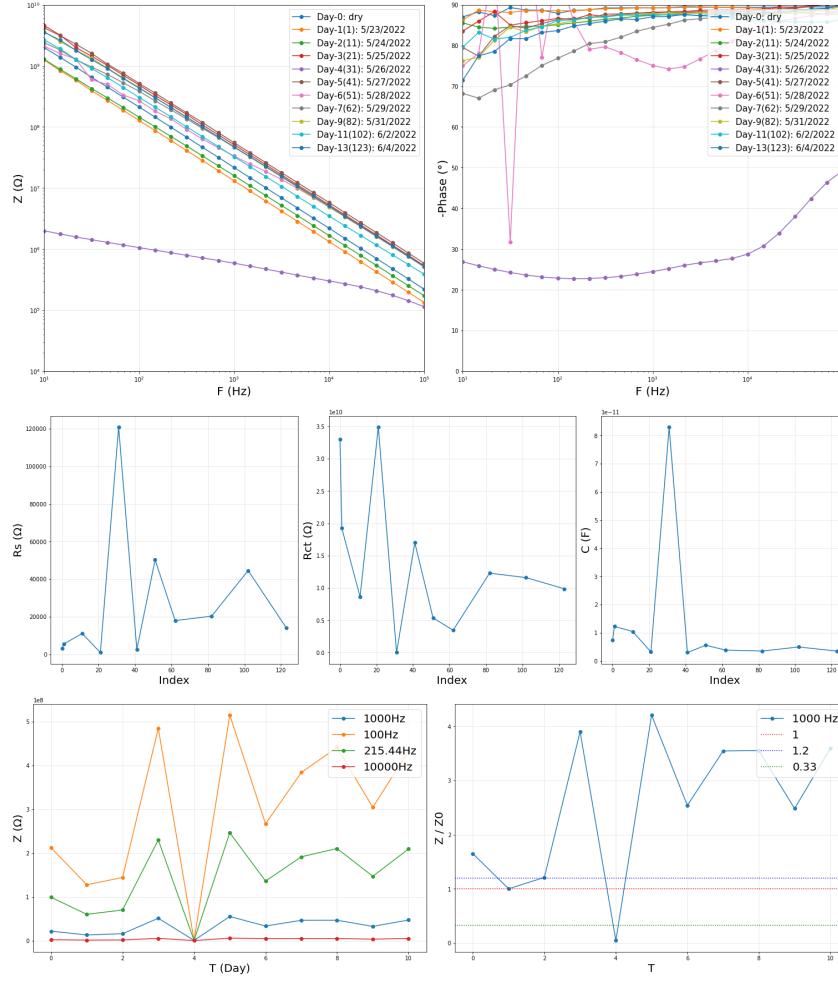
IDE-27-4-m



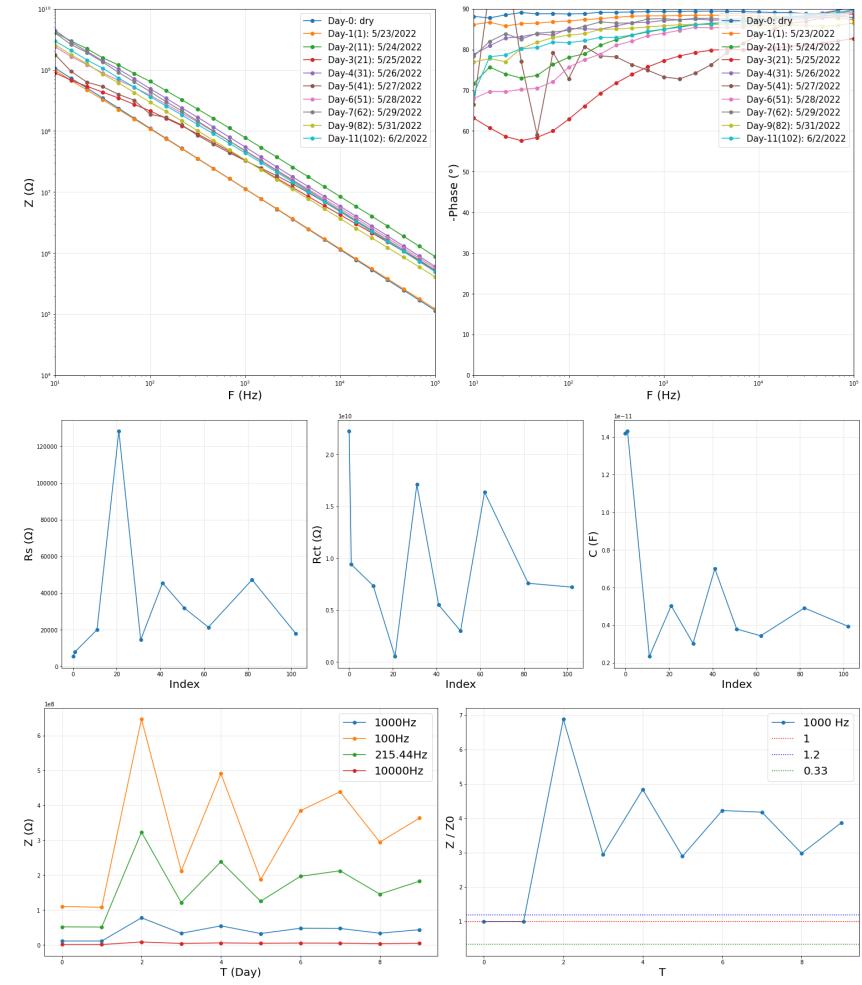
IDE-27-4-s

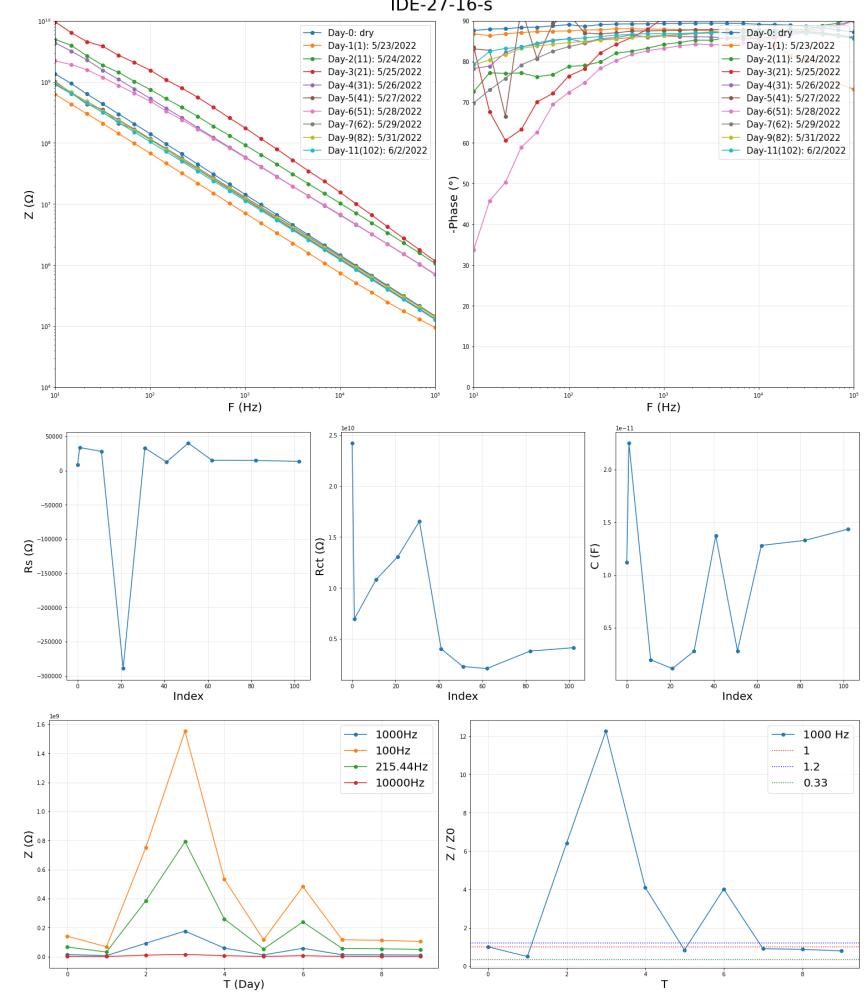
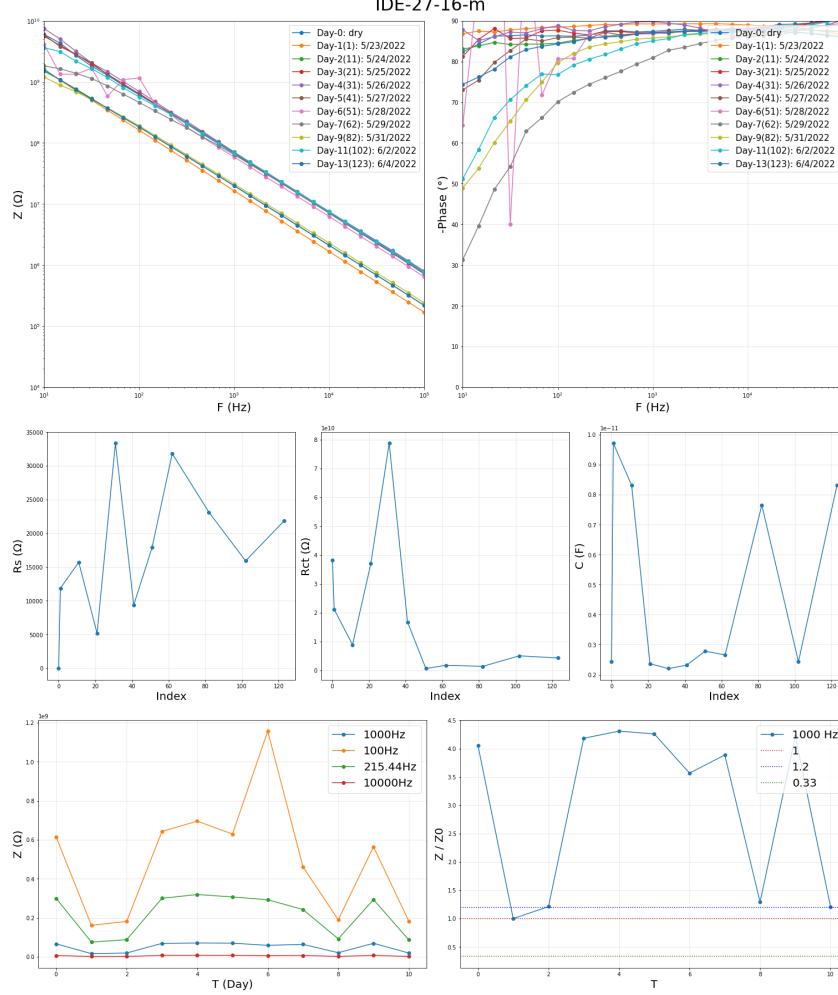


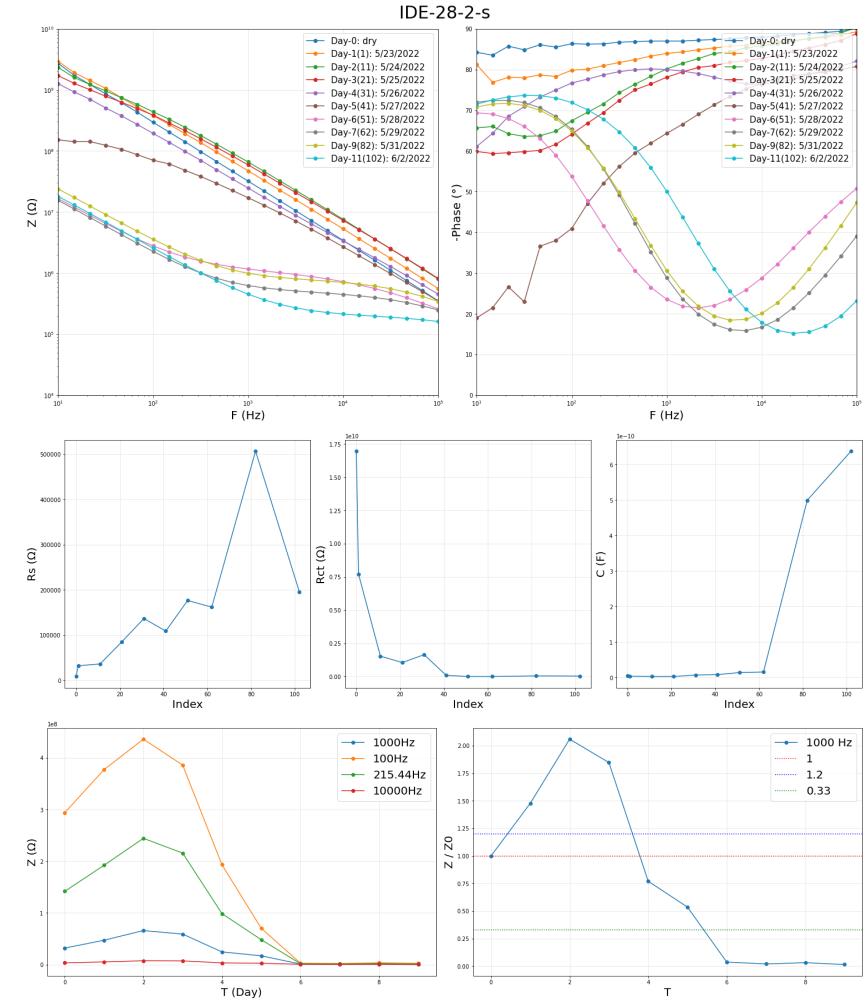
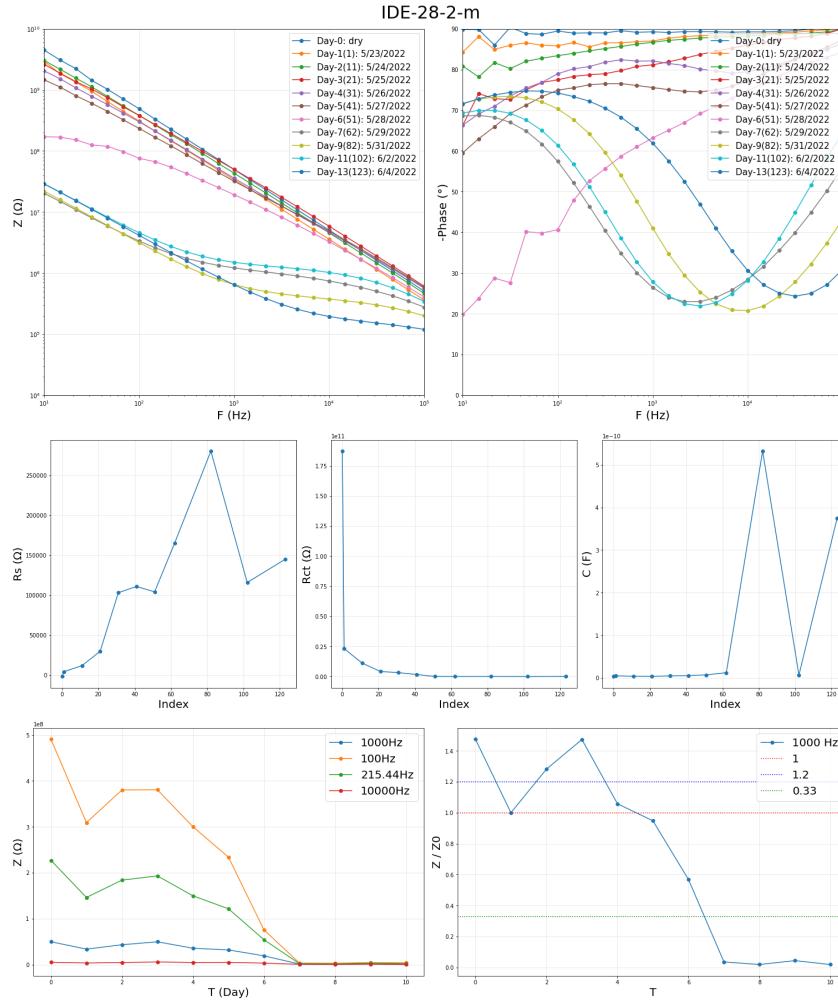
IDE-27-8-m

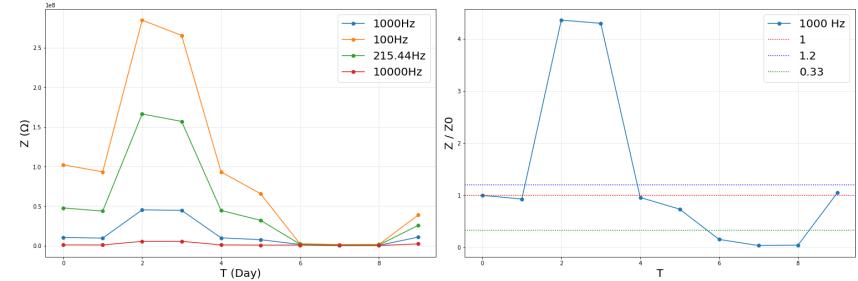
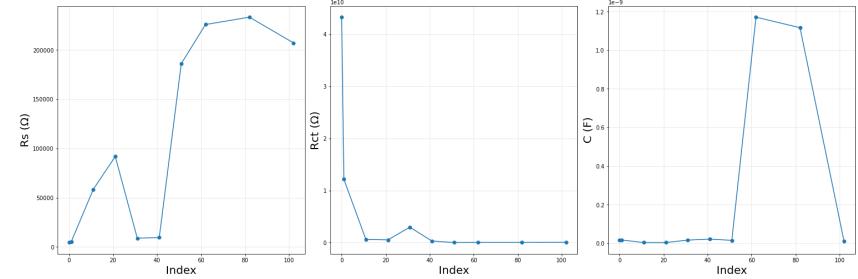
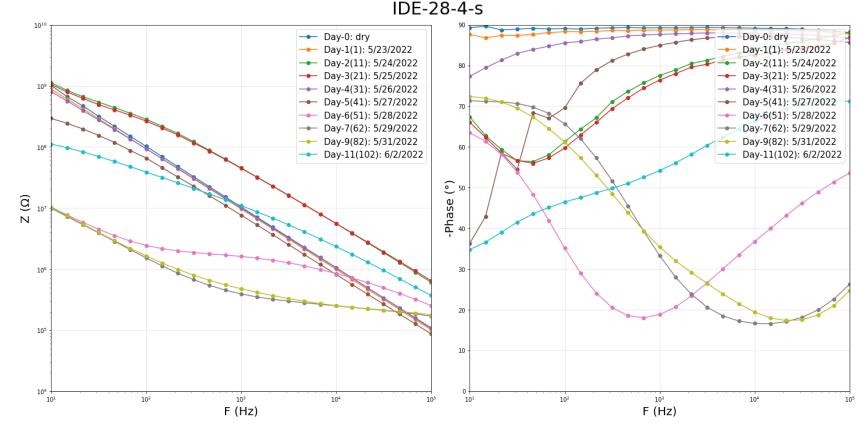
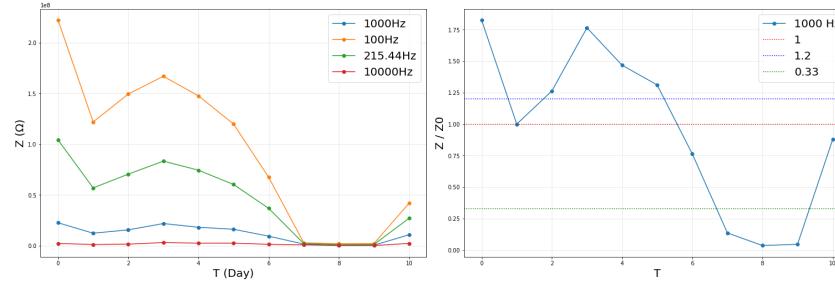
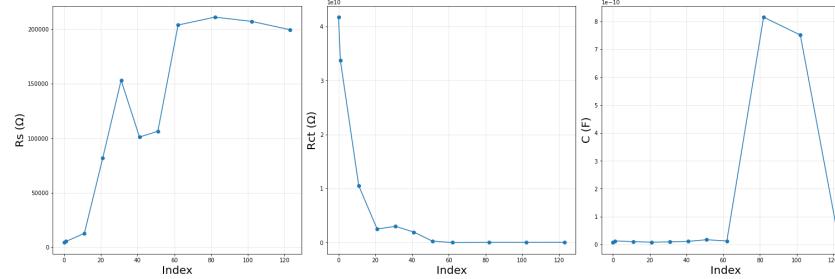
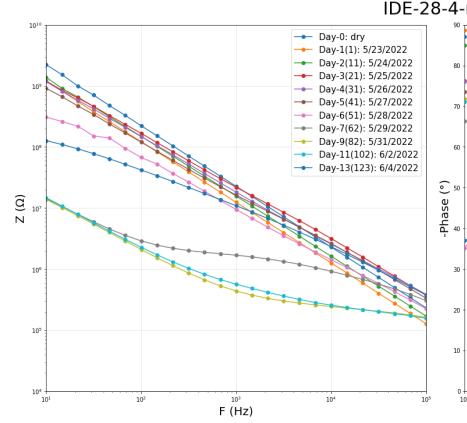


IDE-27-8-s

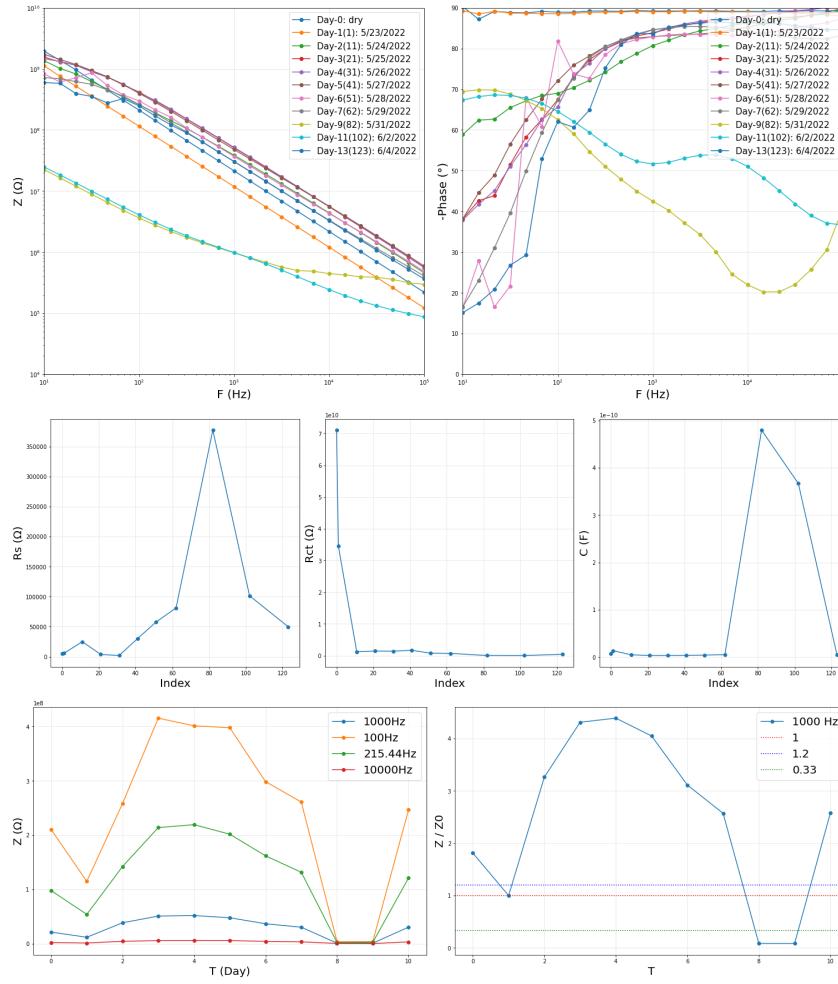




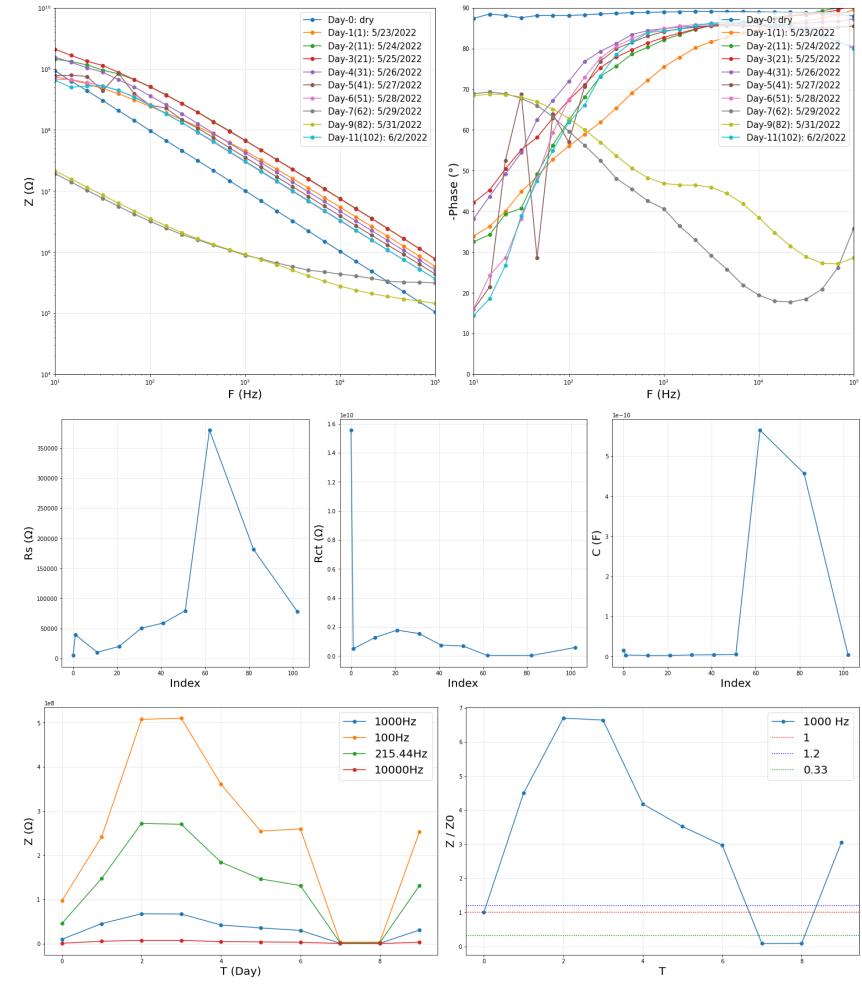




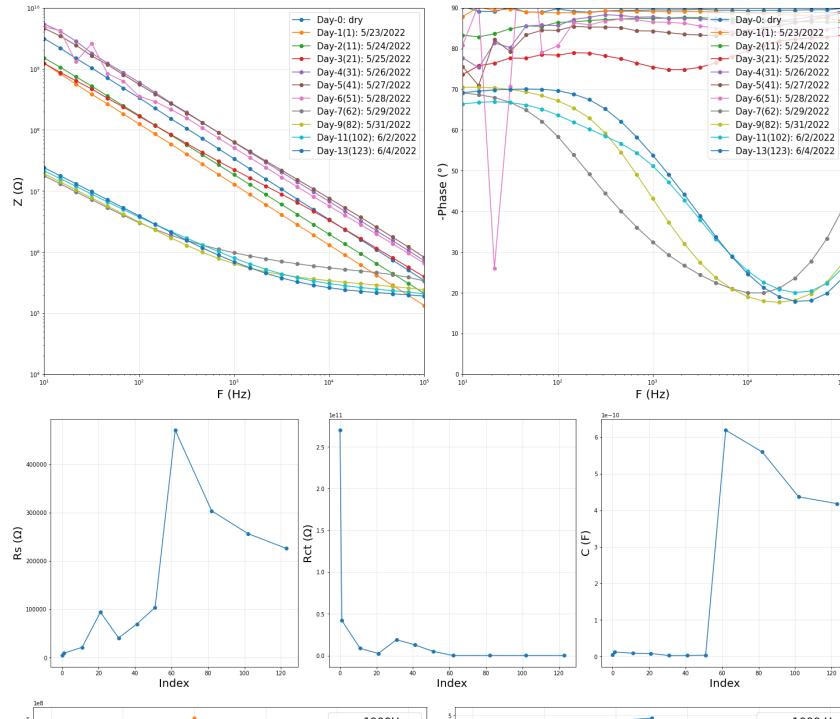
IDE-28-8-m



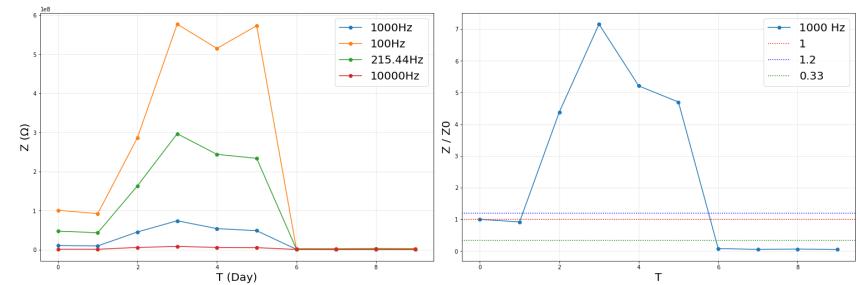
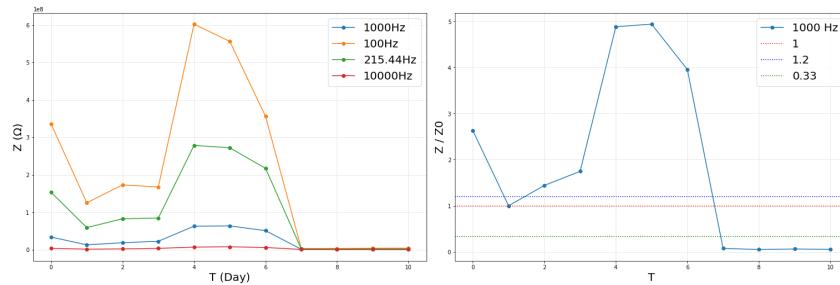
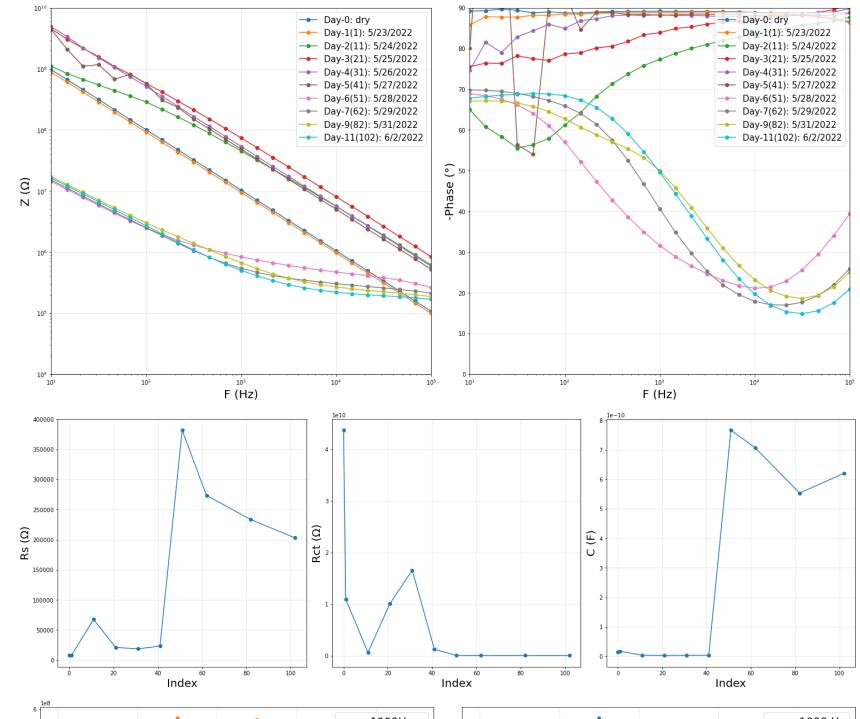
IDE-28-8-s



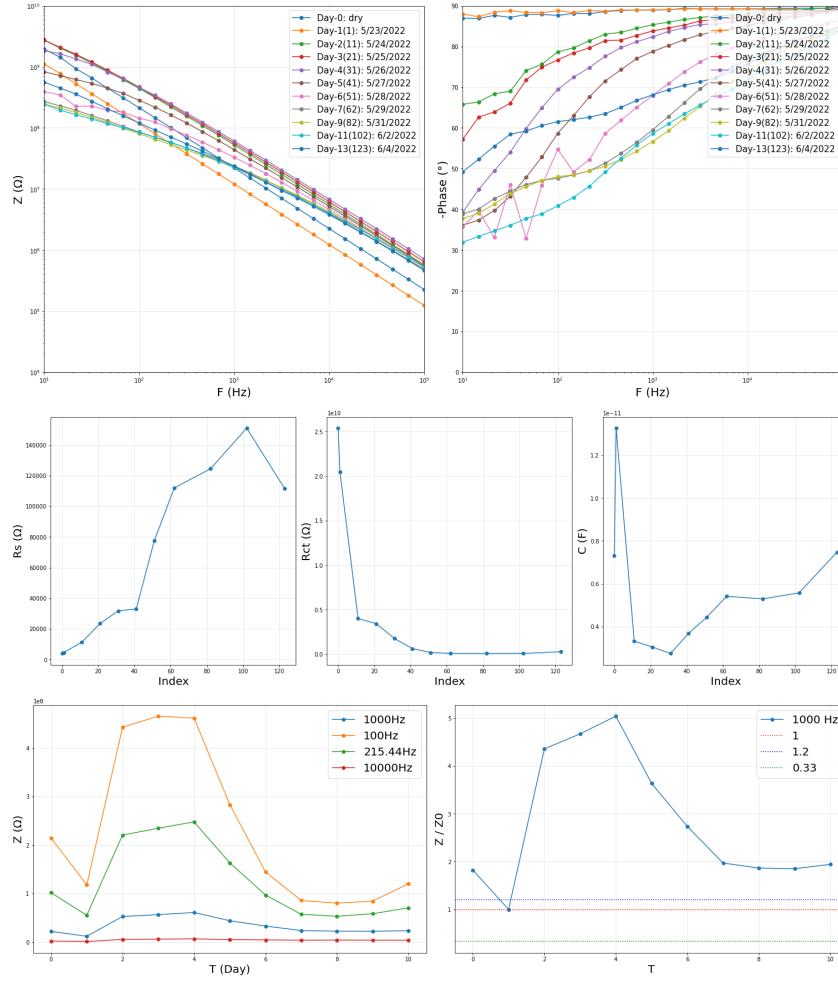
IDE-28-16-m



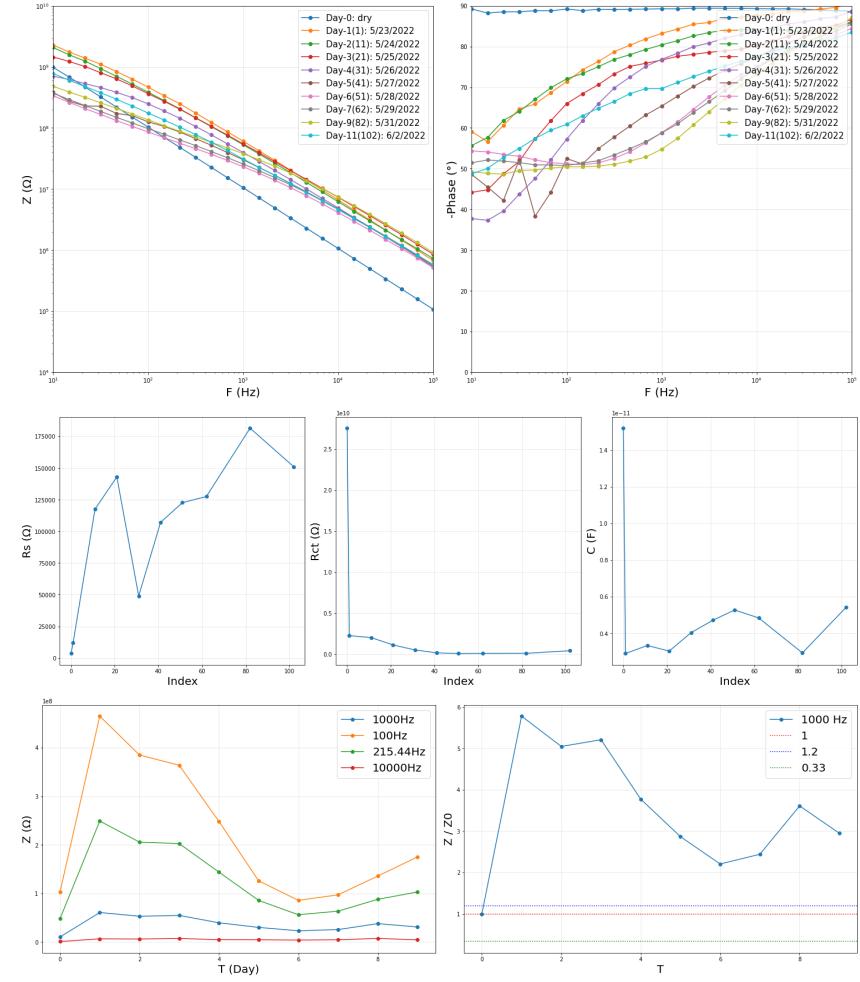
IDE-28-16-s



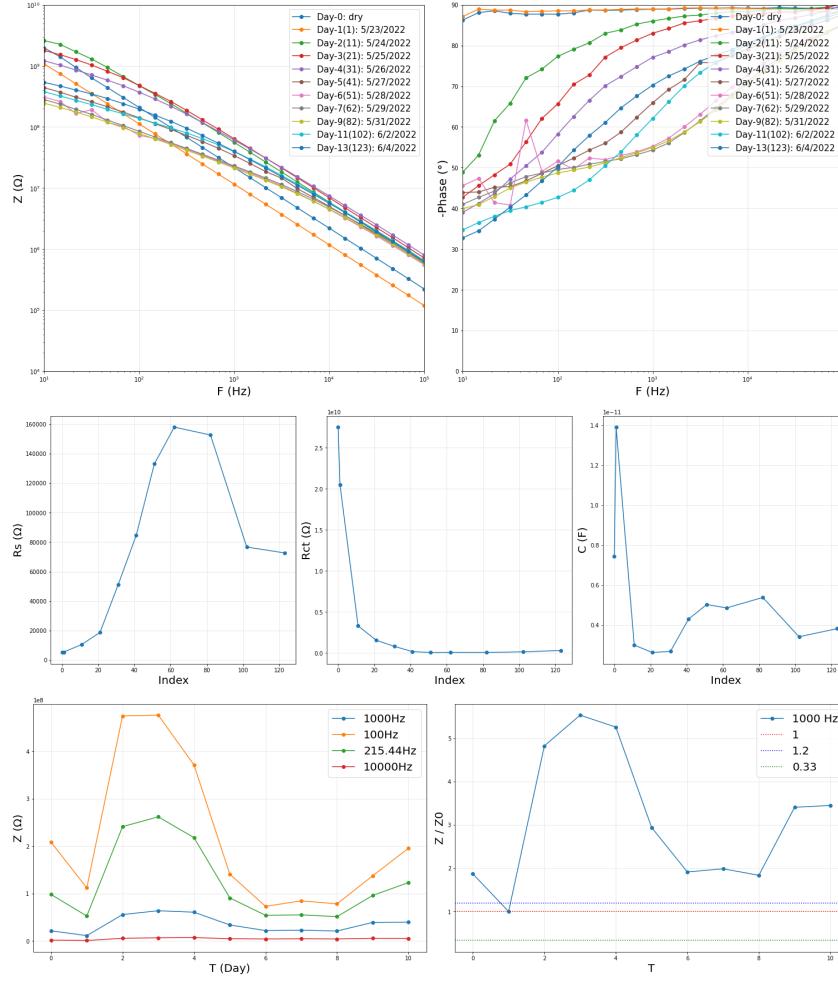
IDE-31-2-m



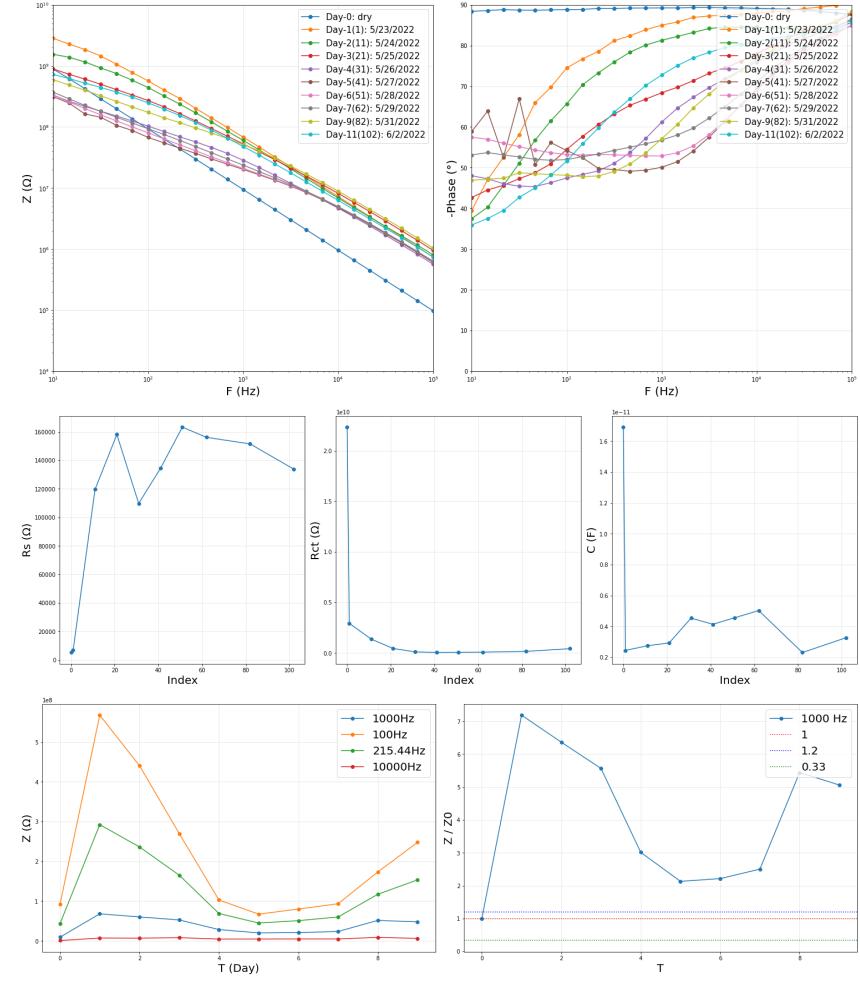
IDE-31-2-s



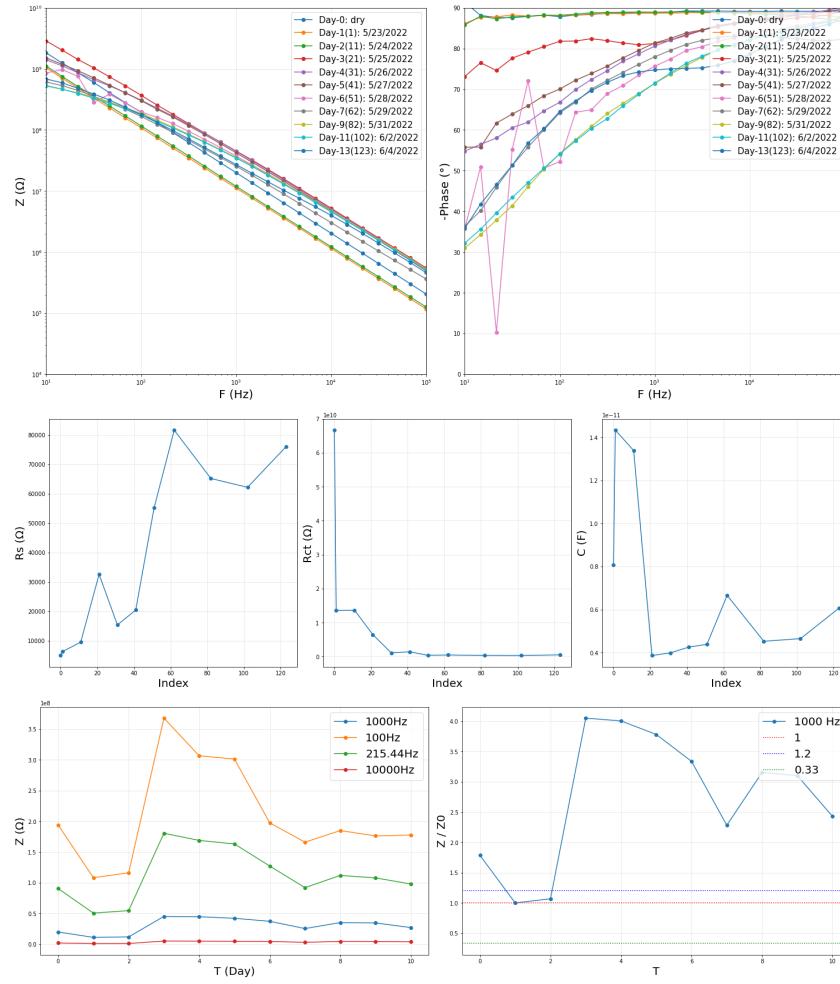
IDE-31-4-m



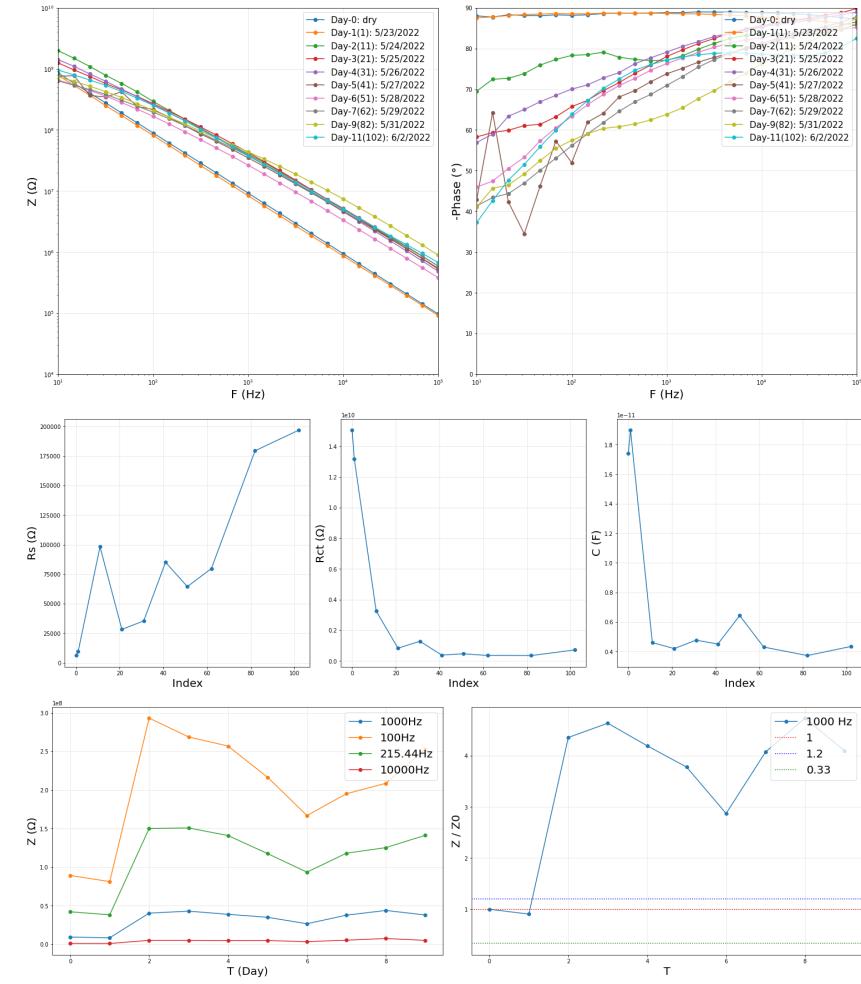
IDE-31-4-s



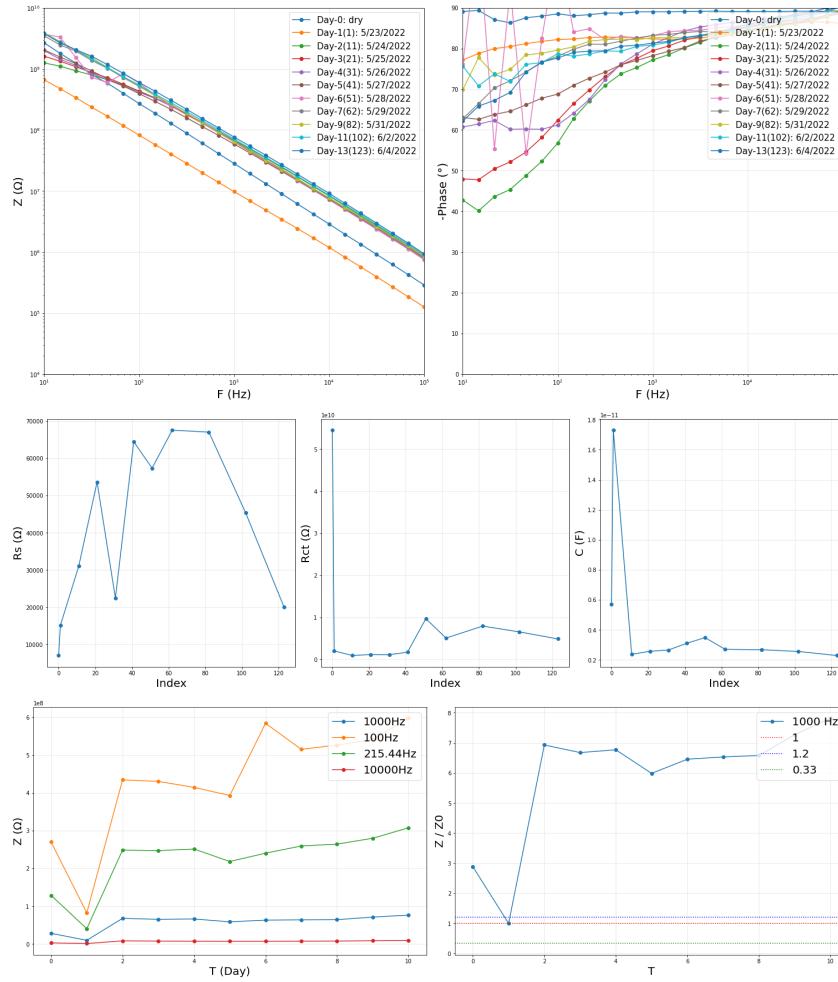
IDE-31-8-m



IDE-31-8-s



IDE-31-16-m



IDE-31-16-s

