

```

import pandas
import math
from scipy.fft import fft, fftfreq
from scipy.signal import butter, filtfilt, savgol_filter
import numpy as np
import matplotlib.pyplot as plt

# Assumptions:
# The noise level is consistent throughout the data set, and relevant noise occurs
# above 450 Hz. If you need to
# identify a noise frequency, use and graph the FFT function.

# Python, as well as the Pandas, SciPy.fft, and NumPy libraries must be installed
# separately in order for the code to
# be functional. If graphs are required, the matplotlib library must also be
# installed.

##### FUNCTION BLOCK
#####

def pad_lists_to_even(data, padding_value=None):
    # Only look at items where the value is a list
    list_values = [v for v in data.values() if isinstance(v, list)]
    if not list_values:
        return data
    max_length = max(len(lst) for lst in list_values)
    for key, value in data.items():
        if isinstance(value, list):
            data[key] += [padding_value] * (max_length - len(value))
    return data

def view_fft_graph(frequencies2, fft_values2):
    # Requires that matplotlib is downloaded to the terminal
    plt.plot(np.abs(frequencies2), np.abs(fft_values2), color='red')
    plt.show()

def graph_data(x, y, x2, y2):
    # Requires that matplotlib is downloaded to the terminal
    plt.plot(x, y, color='blue')
    plt.plot(x2, y2, color='red')
    plt.show()

##### SETUP AND CONTROLS
#####

# SET FILE AND CONTROLS, i.e. FILENAME, FILE DESTINATION, NUMBER OF CYCLES, STARTING
# POINT OF CODE, AND BINS
# Initial information
data_file_path = r"C:\Users\abbyk\Downloads\Neg Pot 7200_C01.txt"
data_file_background_path = r"C:\Users\abbyk\Downloads\1_C01.txt"

```

```

csv_file_path_for_final_data = r"C:\Users\abbyk\Downloads\negpot24hroctober.csv"
number_of_cycles = 7201
first_line_with_data = 1
sampling_rate = 100 # in samples/second
extract_a_cycle = "yes" # "yes" or "no"
cycle_to_extract = 1007
lowpass_filter_frequency = 450
data_format = "yx" # Type "yx" or "xy" for data that is current first or data that
is potential first
data_format_bkgd = "yx"
charging_current_present = "yes" # If your instrument records the charging current,
put "yes" here. Otherwise, put "no"
cycle_start = 'high' # If you start scanning from the higher value, put "high" here.
Otherwise, put "low"
view_fft_plot = "no" # If you want to see the plot, put "yes" here. Otherwise, put
"no"
low_e = -1
high_e = -0.5

```

```

##### MAIN BLOCK
#####

```

```

# Set up lists and dictionaries
bins_dictionary = {}
bin_results_dictionary = {}
for binnum in range(1, (number_of_cycles + 1)): # Create bins
    bins_dictionary.update({f"{binnum} x-values": [], f"{binnum} y-values": []})
y_value_list = []
x_value_list = []
y_value_list_bkgd = []
y_value_list_bkgd_full_length = []
y_value_list_raw = []
filtered_y_values = []
peak_to_peak_separation_list = []
e_half_cycle_list = []

```

```

# Load data from file
if data_format.lower() == "yx":
    with open(data_file_path, "r") as file:
        lines=file.readlines()
        # Assume that the first line of the file is the header
        for line in lines[first_line_with_data:]:
            line_components = line.strip().split()
            y_value_list_raw.append(float(line_components[0]))
            x_value_list.append(float(line_components[1]))
elif data_format.lower() == "xy":
    with open(data_file_path, "r") as file:
        lines=file.readlines()
        # Assume that the first line of the file is the header
        for line in lines[first_line_with_data:]:

```

```

        line_components = line.strip().split()
        y_value_list_raw.append(float(line_components[1]))
        x_value_list.append(float(line_components[0]))
else:
    print('Please enter a valid option under "data_format"')

# Load background data
if data_format_bkgd.lower() == "yx":
    with open(data_file_background_path, "r") as file:
        lines=file.readlines()

        # Assume that the first line of the file is the header and there is time
data in the second column
        for line in lines[first_line_with_data:]:
            line_components = line.strip().split()
            y_value_list_bkgd.append(float(line_components[0]))
elif data_format_bkgd.lower() == "xy":
    with open(data_file_background_path, "r") as file:
        lines=file.readlines()

        # Assume that the first line of the file is the header and there is time
data in the second column
        for line in lines[first_line_with_data:]:
            line_components = line.strip().split()
            y_value_list_bkgd.append(float(line_components[1]))
else:
    print('Please enter a valid option under "data_format_bkgd"')

# Eliminate first 5 points of charging current for both sets of data by setting them
to 0
if charging_current_present.lower() == "yes":
    for i in range(5):
        y_value_list_raw[i] = 0.0
        y_value_list_bkgd[i] = 0.0

# CREATE BINS AND ADD TO DICTIONARY
# Find minima
x_array = np.array(x_value_list)
y_array = np.array(y_value_list_raw)
inverted_x = -x_array
if number_of_cycles ==1:
    pass
elif cycle_start.lower().strip() == "low":
    x_values_for_bin_division = x_value_list
    x_smooth = savgol_filter(x_array, window_length=51, polyorder=3)
    dx = np.diff(x_smooth)
    turning_points = np.where(np.diff(np.sign(dx)) != 0)[0] + 1
    expected_cycle_length = len(x_array) / number_of_cycles
    minima = []
    if low_e == 0:

```

```

        low_e = 0.1
    for idx in turning_points:
        if x_array[idx] <= (low_e * 1.5):
            minima.append(idx)

    # After detecting and filtering the extrema (minima or maxima)
    if len(minima) == number_of_cycles + 1:
        pass
    elif len(minima) == number_of_cycles:
        # Missing one endpoint, try to determine which
        if minima[0] > expected_cycle_length * 0.5:
            minima.insert(0, 0)
        elif (len(x_array) - minima[-1]) > expected_cycle_length * 0.5:
            minima.append(len(x_array) - 1)
        else:
            raise ValueError("Only one endpoint can be added, but unsure which is
missing.")
    elif len(minima) == number_of_cycles - 1:
        # Missing two endpoints, try to determine which
        if minima[0] >= expected_cycle_length * 0.5:
            minima.insert(0, 0)
            if (len(x_array) - minima[-1]) >= expected_cycle_length * 0.5:
                minima.append(len(x_array) - 1)
            else: raise ValueError("Only one endpoint was able to be added.")
        else: raise ValueError("Unable to determine which two values are missing.")
    else:
        raise ValueError(
            f"Expected {number_of_cycles + 1} minima, but found {len(minima)}.\n"
            f"This may be due to duplicate detections, missing endpoints, or a
filtering error."
        )
    elif cycle_start.lower().strip() == "high":
        x_values_for_bin_division = x_value_list
        x_smooth = savgol_filter(x_array, window_length=51, polyorder=3)
        dx = np.diff(x_smooth)
        turning_points = np.where(np.diff(np.sign(dx)) != 0)[0] + 1
        expected_cycle_length = len(x_array) / number_of_cycles
        maxima = []
        if high_e == 0:
            high_e = 0.01
        for idx in turning_points:
            if x_array[idx] >= (high_e * 1.5):
                maxima.append(idx)

    # After detecting and filtering the extrema (minima or maxima)
    if len(maxima) == number_of_cycles + 1:
        pass
    elif len(maxima) == number_of_cycles:
        # Missing one endpoint, try to determine which
        if maxima[0] > expected_cycle_length * 0.5:

```

```

        maxima.insert(0, 0)
    elif (len(x_array) - maxima[-1]) > expected_cycle_length * 0.5:
        maxima.append(len(x_array) - 1)
    else:
        raise ValueError("Only one endpoint can be added, but unsure which is
missing.")
    elif len(maxima) == number_of_cycles - 1:
        # Missing two endpoints, try to determine which
        if maxima[0] >= expected_cycle_length * 0.5:
            maxima.insert(0, 0)
            if (len(x_array) - maxima[-1]) >= expected_cycle_length * 0.5:
                maxima.append(len(x_array) - 1)
            else: raise ValueError("Only one endpoint was able to be added.")
        else: raise ValueError("Unable to determine which two values are missing.")
    else:
        raise ValueError(
            f"Expected {number_of_cycles + 1} maxima, but found {len(maxima)}.\n"
            f"This may be due to duplicate detections, missing endpoints, or a
filtering error."
        )

    # For logistical simplicity
    minima = maxima
else:
    raise ValueError("Please enter a valid option under cycle_start! Use 'low' or
'high'.")

if number_of_cycles == 1:
    minima = [0, len(x_value_list)-1]
    x_values_for_bin_division = x_value_list

# Add data to bins
for value in range(number_of_cycles):
    bin_number = value + 1
    cycle_start_index = minima[value]
    cycle_end_index = minima[value + 1]
    x_value_list_bin = x_values_for_bin_division[cycle_start_index:cycle_end_index]
    y_value_list_bin = y_value_list_raw[cycle_start_index:cycle_end_index]
    bins_dictionary[f'{bin_number} x-values'] = x_value_list_bin
    bins_dictionary[f'{bin_number} y-values'] = y_value_list_bin

# Subtract the background from each bin
for binnum in range(1, number_of_cycles+1):
    unsubtracted_y = bins_dictionary[f'{binnum} y-values']
    unpadded_x_values = bins_dictionary[f'{binnum} x-values']
    array1 = np.array(unsubtracted_y)
    array2 = np.array(y_value_list_bkgd)
    if len(unsubtracted_y) < len(y_value_list_bkgd):
        pad_amount = len(y_value_list_bkgd) - len(unsubtracted_y)
        padded1 = np.pad(unsubtracted_y, (0, pad_amount), 'edge')

```

```

        x_values = np.pad(unpadded_x_values, (0, pad_amount), 'edge')
        subtracted_array = np.subtract(padded1, array2)
        x_values_padded = x_values.tolist()
        bins_dictionary[f'{binnum} x-values'] = x_values_padded
    elif len(y_value_list_bkgd) < len(unsubtracted_y):
        pad_amount = len(unsubtracted_y) - len(y_value_list_bkgd)
        padded2 = np.pad(y_value_list_bkgd, (0, pad_amount), 'edge')
        subtracted_array = np.subtract(array1, padded2)
        bins_dictionary[f'{binnum} x-values'] = unpadded_x_values
    else:
        subtracted_array = np.subtract(array1, array2)
        bins_dictionary[f'{binnum} x-values'] = unpadded_x_values
    subtracted_list = subtracted_array.tolist()
    y_value_list.extend(subtracted_list)
    bins_dictionary[f'{binnum} y-values'] = subtracted_list

# IDENTIFY THE NOISE FREQUENCY FOR CYCLE ONE, TO BE APPLIED TO THE DATASET (to be
input by user, comment in the graph line)
# Isolate bin 1 to find noise frequency for data set
bin_1_y_values = bins_dictionary["1 y-values"]
bin_1_x_values = bins_dictionary["1 x-values"]

# Identify noise frequency using Fast Fourier Transform via SciPy
# Graphically compare the background frequencies to the data frequencies
N = len(y_value_list_bkgd) # Number of samples
fft_values1 = fft(np.array(y_value_list_bkgd))
frequencies1 = fftfreq(N, 1.0 / float(sampling_rate))

N = len(bin_1_y_values) # Number of samples
fft_values2 = fft(np.array(bin_1_y_values))
frequencies2 = fftfreq(N, 1.0 / float(sampling_rate))
if view_fft_plot.lower() == "yes":
    view_fft_graph(frequencies2, fft_values2)

# FILTER NOISE
# Create filter using a butterworth low-pass filter via SciPy
nyquist = 0.5 * sampling_rate
low = lowpass_filter_frequency / nyquist
b, a = butter(1, low, btype='low', fs=sampling_rate)

# Filter noise by cycle
for binnum in range(1, number_of_cycles+1):
    unfiltered_y = bins_dictionary[f'{binnum} y-values']
    filtered_list = filtfilt(b, a, unfiltered_y).tolist()
    filtered_y_values.extend(filtered_list)
    bins_dictionary[f'{binnum} y-values'] = filtered_list
#graph_data(bin_1_x_values, bin_1_y_values, x_value_list, filtered_y_values)

# FIND THE IPA AND IPC FOR EACH BIN AND LOCATE THE CORRESPONDING EPA AND EPC
# Find the values and add them to a list

```

```

for i in range(0, number_of_cycles):
    cycle_bin = i+1
    y_bin_values = bins_dictionary[f'{cycle_bin} y-values']
    x_bin_values = bins_dictionary[f'{cycle_bin} x-values']
    ipa = np.min(np.array(y_bin_values))
    ipc = np.max(np.array(y_bin_values))
    ipa_placement = y_bin_values.index(ipa)
    ipc_placement = y_bin_values.index(ipc)
    epa = x_bin_values[ipa_placement]
    epc = x_bin_values[ipc_placement]
    peak_to_peak_separation = abs(epa-epc)
    e_half_cycle = np.mean([epc, epa])
    peak_to_peak_separation_list.append(peak_to_peak_separation)
    e_half_cycle_list.append(e_half_cycle)

# Add E1/2 and Peak-to-peak separation to results dictionary
bin_results_dictionary.update({'peak-to-peak separation':
peak_to_peak_separation_list})
bin_results_dictionary.update({'E1/2': e_half_cycle_list})

# DETERMINE ENSEMBLE E1/2
ensemble_half = np.mean(e_half_cycle_list)
bin_results_dictionary.update({'ensemble E1/2': ensemble_half})

# OPTION TO EXTRACT A SINGLE CYCLE
if extract_a_cycle.lower() == "no":
    # Convert scalars to single-value lists
    for key, value in bin_results_dictionary.items():
        if not isinstance(value, list):
            bin_results_dictionary[key] = [value]
    # WRITE THE DATA TO A CSV FILE
    bin_results_dictionary_padded = pad_lists_to_even(bin_results_dictionary)
    df_results = pandas.DataFrame(bin_results_dictionary_padded)
    df_results.to_csv(csv_file_path_for_final_data, index=False)
elif extract_a_cycle.lower() == "yes":
    # EXTRACT A SINGLE CYCLE
    bin_results_dictionary.update({
        f'Cycle {cycle_to_extract} x-values': bins_dictionary[f'{cycle_to_extract}
x-values']
    })
    bin_results_dictionary.update({
        f'Cycle {cycle_to_extract} y-values': bins_dictionary[f'{cycle_to_extract}
y-values']
    })
    # Convert scalars to single-value lists
    for key, value in bin_results_dictionary.items():
        if not isinstance(value, list):
            bin_results_dictionary[key] = [value]
    # WRITE THE DATA TO A CSV FILE
    bin_results_dictionary_padded = pad_lists_to_even(bin_results_dictionary)

```

```
df_results = pandas.DataFrame(bin_results_dictionary_padded)
df_results.to_csv(csv_file_path_for_final_data, index=False)
else:
    raise ValueError("Please enter a valid option under extract_a_cycle! Use 'yes'
or 'no'.")
```