SUPPORTING INFORMATION

Polymer Bead Size Revealed via Neural Network Analysis of Single-Entity Electrochemical Data

Gabriel Gemadzie, Baosen Zhang, Aliaksei Boika*
Department of Chemistry, The University of Akron, Akron, OH 44325, USA

* Corresponding Author, E-mail: aboika@uakron.edu

Table of Contents **Page**

Preparing polystyrene beads solution

The polystyrene beads samples of catalog numbers PC04N and PC05001N were purchased from Bangs Laboratories, Inc. The beads came as suspensions accompanied with their sample information data sheets. The information provided by the data sheet (mean diameter, density and solid percent) is used to calculate the concentration (M) of the suspension. For PC04N the concentration was calculated to be 0.316nM. 0.032mL (32µL) of the purchased suspension is diluted to 10mL with deionized water in a 10mL volumetric flask to obtain 1pM concentration. To achieve the desired concentration of 10fM in the electrochemical cell, 0.1mL (100µL) of the diluted bead suspension is injected into an electrochemical cell containing 9.9mL (9900µL) redox mediator (2mM $FcCH_2OH$, 1mM KNO_3). Similar procedure is used in the dilution of the other sample (PC05001N).

i-t curves recorded:

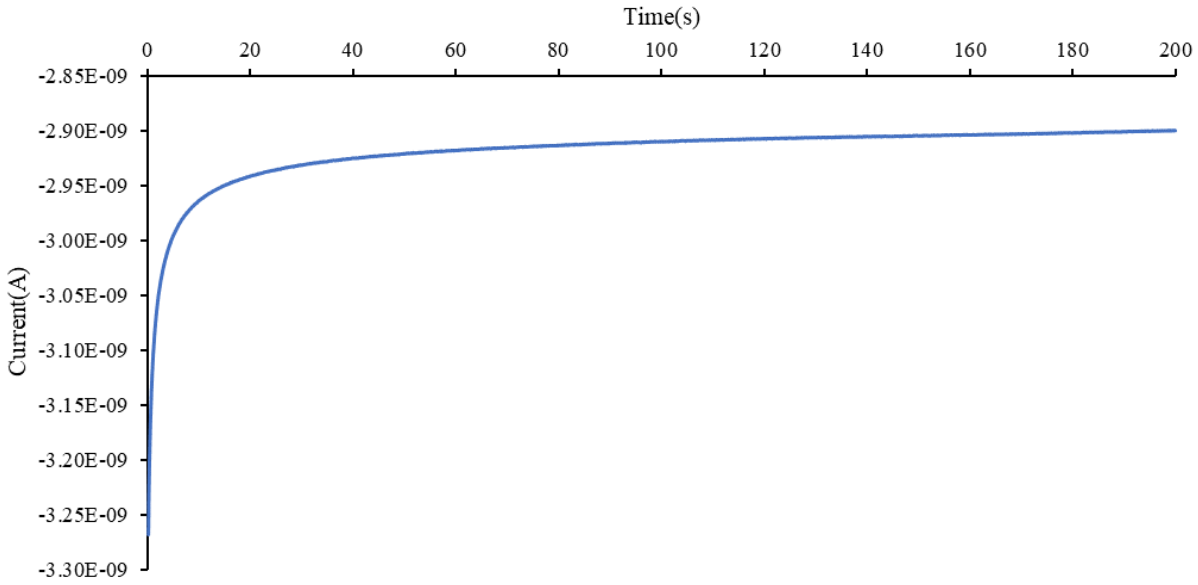


Figure S1 (case 1). Spheres free chronoamperogram recorded for 2mM $FcCH_2OH$, 1mM KNO_3 for 200s at +0.5V vs Ag/AgCl

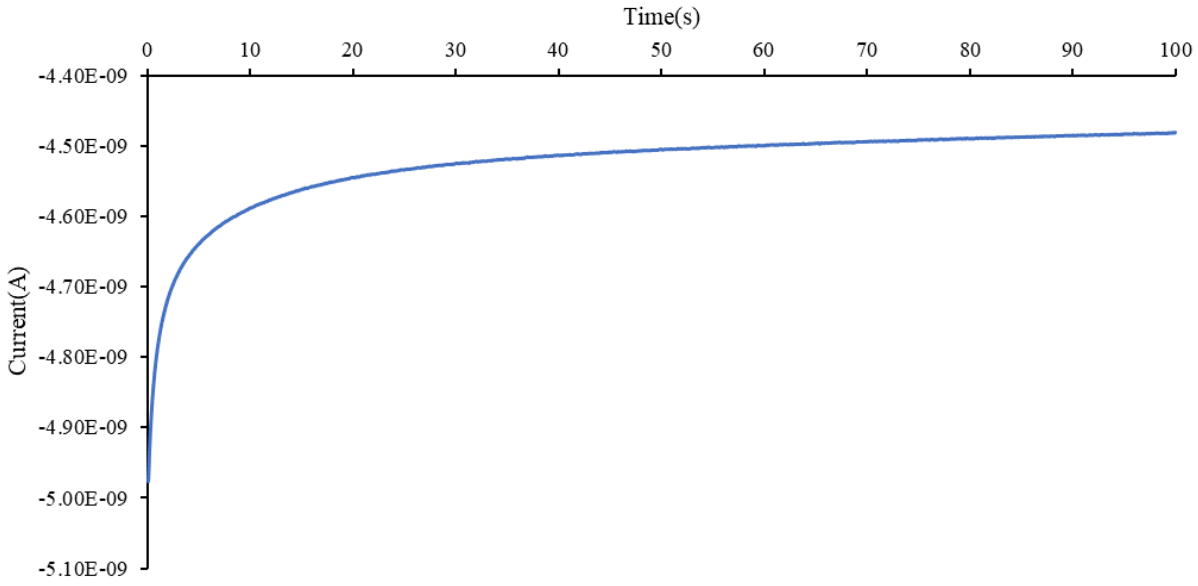


Figure S2 (case 2). Spheres free chronoamperogram recorded for 3mM $FcCH_2OH$, 1mM KNO_3 for 100s at +0.5V vs Ag/AgCl

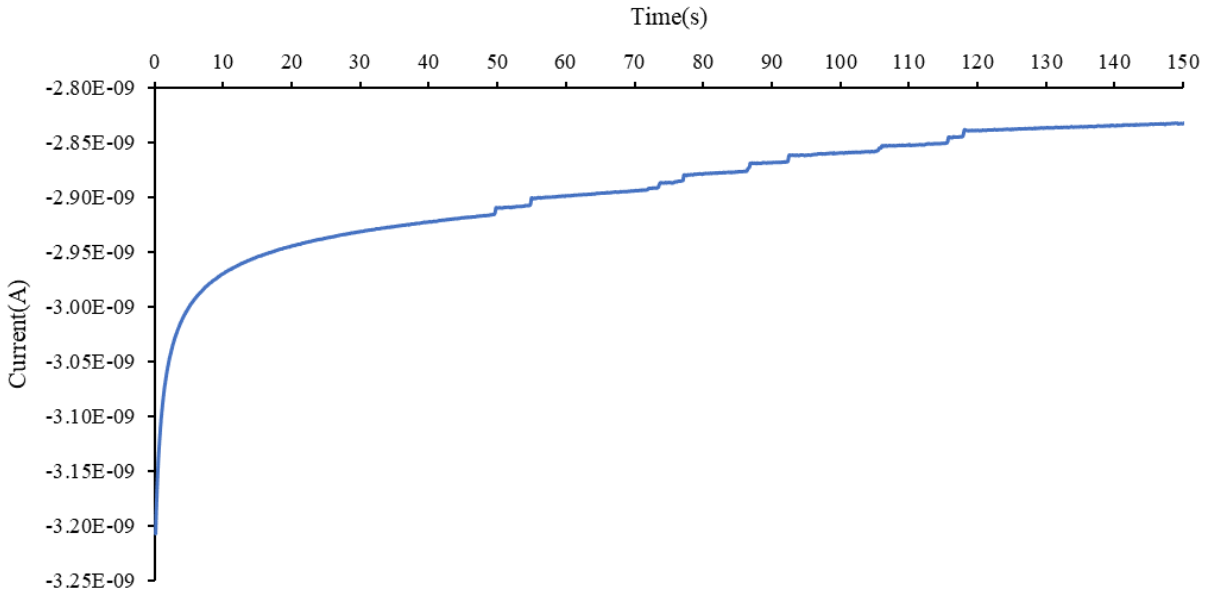


Figure S3 (case 1). Chronoamperogram recorded with 10 fM PC04N spheres in 2mM $FcCH_2OH$, 1mM KNO_3 for 150s at +0.5V vs Ag/AgCl

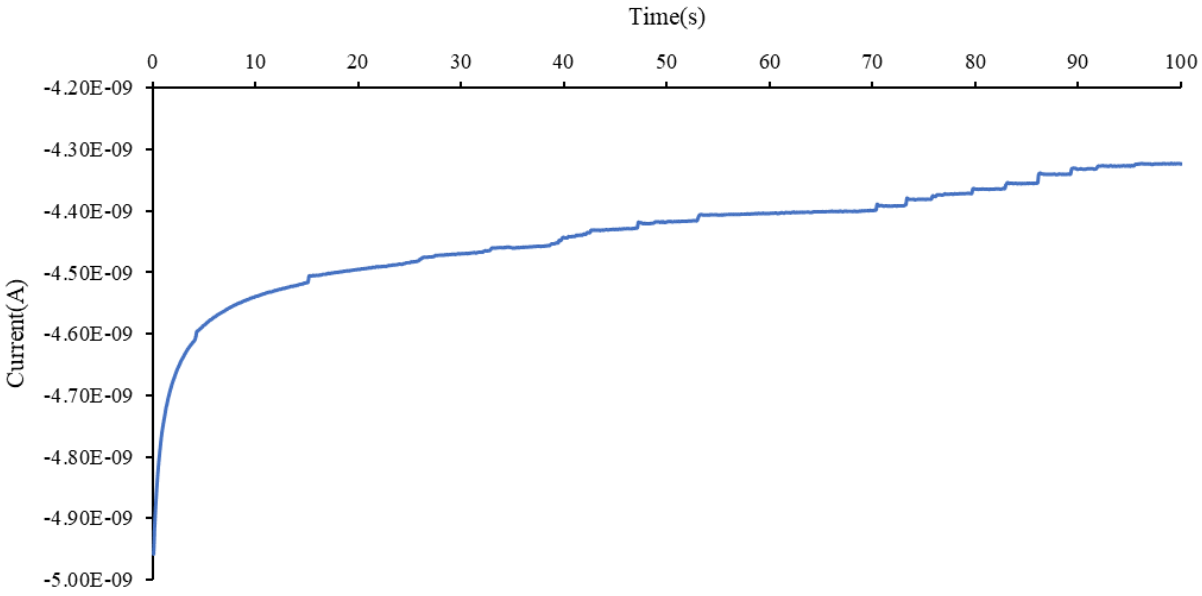


Figure S4 (case 2). Chronoamperogram recorded with 10 fM PC04N spheres in 3mM $FcCH_2OH$, 1mM KNO_3 for 100s at +0.5V vs Ag/AgCl

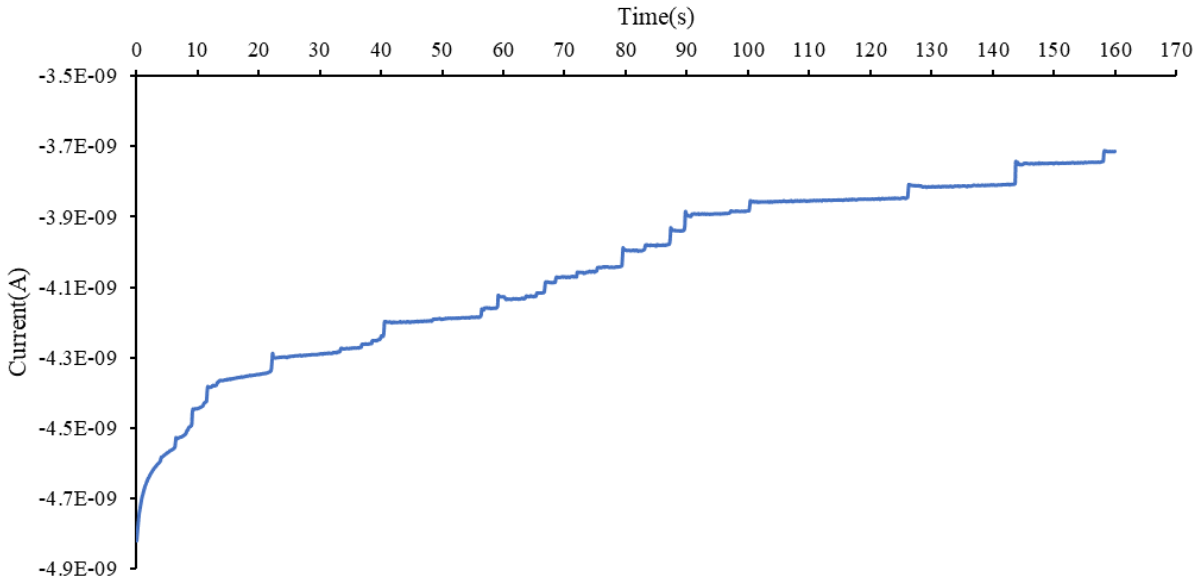


Figure S5 (case 3). Chronoamperogram recorded for a mixture of 10 fM each of PC04N and PC05001N spheres in 3mM $FcCH_2OH$, 1mM KNO_3 for 160s at +0.5V vs Ag/AgCl

Table S1. Collision times(s) with the corresponding current step heights for cases 1 and 2.

Case 1			Case 2	
Time(s)	**Step Height((E-09)/A)**		**Time(s)**	**Step Height(E-09)/A**
49.4664	**0.00629**		**15.0993**	**0.01031**
54.600	**0.007147**		**32.688**	**0.00516**
73.3312	**0.00420**		**42.460**	**0.00376**
77.016	**0.00458**		**47.116**	**0.00926**
86.2624	**0.00724**		**52.952**	**0.01065**
92.2176	**0.00591**		**70.216**	**0.00927**
105.312	**0.00241**		**73.168**	**0.01032**
105.877	**0.00257**		**75.6448**	**0.00549**
115.365	**0.00550**		**79.6192**	**0.00754**
117.856	**0.00635**		**82.8064**	**0.00892**
			85.9936	**0.01613**
			89.1808	**0.00893**
			91.6192	**0.00547**
			95.32	**0.00343**

Table S2. Collision times(s) with the corresponding current step heights for case 3.

Case 3	
Time(s)	**Step Height((E-09)/A)**
6.3312	**0.0289**
9.0864	**0.04586**
11.4736	**0.03968**
22.0944	**0.04885**
33.2419	**0.01072**
36.7531	**0.01066**
38.5307	**0.00763**
40.408	**0.04123**
56.2904	**0.02288**
58.9568	**0.03435**
65.3764	**0.00992**
66.6392	**0.02901**
68.4784	**0.01451**
72.0864	**0.01376**
75.2588	**0.01141**
79.342	**0.04773**
83.080	**0.01526**
87.214	**0.04767**
89.6200	**0.05727**
100.15	**0.02667**
126.067	**0.03584**
143.713	**0.06256**
158.087	**0.02976**

Reading current step height using Python

We provide a Python code that takes the experimental data in a "csv" format to generate the current step heights of all the collision events. The Python code is run using Google Colab, an open access platform for writing and running python codes. The basic guide on how to use Google Colab can be found in our recently published paper in the Journal of Chemical Education and our recorded videos.[1] To execute this code, we have provided the source code and a sample i-t curve data with the file name "i-t_example.txt" in a zip folder of "readBlockPython.zip". The "i-t_example.txt" is generated from a CHI electrochemical instrument. The data is collected with a 10 μm Pt microelectrode(working electrode) in 2 mM FcMeOH, 1mM KNO_3 supporting electrolyte, 10 fM 0.99μm beads, AgAgCl reference electrode vs +0.5V, Pt auxiliary electrode, 20mVs^{-1} scan rate at room temperature for 150s.

Step 1: First unzip the zip folder and open the source code in Google Colab. Once it is open, load the sample "i-t_exampel.txt" into "Files" in Colab by simply dragging the file into the interface. Note for the purpose of this illustration the file name is the same as seen in the red rectangle (Figure S6). Then, we visualize the i-t curve using the Python package of "matplotlib.pyplot" (Figure S7).

Step 2: Some parameters must be provided in this step (Figure S8) to run the code successfully, using the i-t curve in the zip file as an example. First, to determine when to start the analysis to get rid of unstable data at the beginning, simply read the graph (Figure 7) and determine a "start_analysis_time". In our example, we set "start_analysis_time = 40", which means we start reading the curve from 40 seconds. Secondly, the sample interval for the data is obtained from the CHI instrument, this is the 'Sample Interval(sec)' set in settings before running the chronoamperogram. In our example, we collected the data by setting the sample interval at 0.1 s. Thus, the value 0.1(Figure S8).

Step 3: The default value of "threshold_slope_ratio" is set at 15 (Figure S8). If the test settings for the i-t curve are nearly the same as ours, this value does not need to change. The threshold slope(T_S) is calculated using the equation $T\_S = threshold\_slope\_ratio \times \frac{\Delta current(A)}{\Delta time(s)}$ from the beginning of the curve with no observed collision event. Click "run" after this cell.

Step 4: After successfully running the code, the block of code (Figure S9) displays all the collision time and current step heights. Save this generated data to Excel for further analysis.

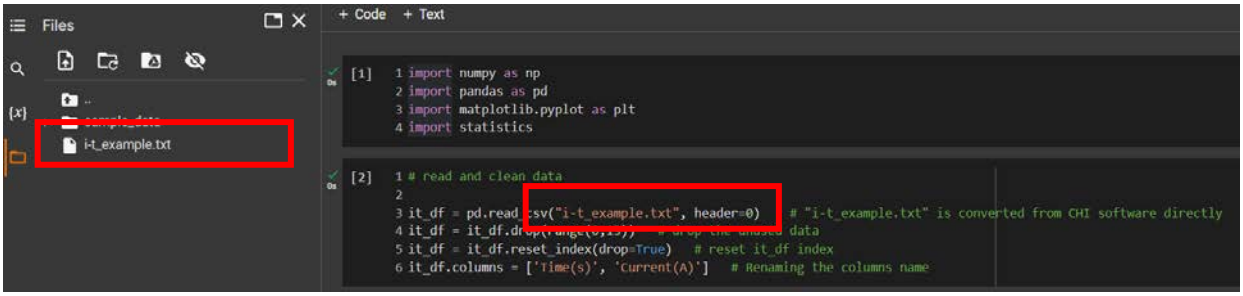


Figure S6. Screenshot of Files in Google Colab.

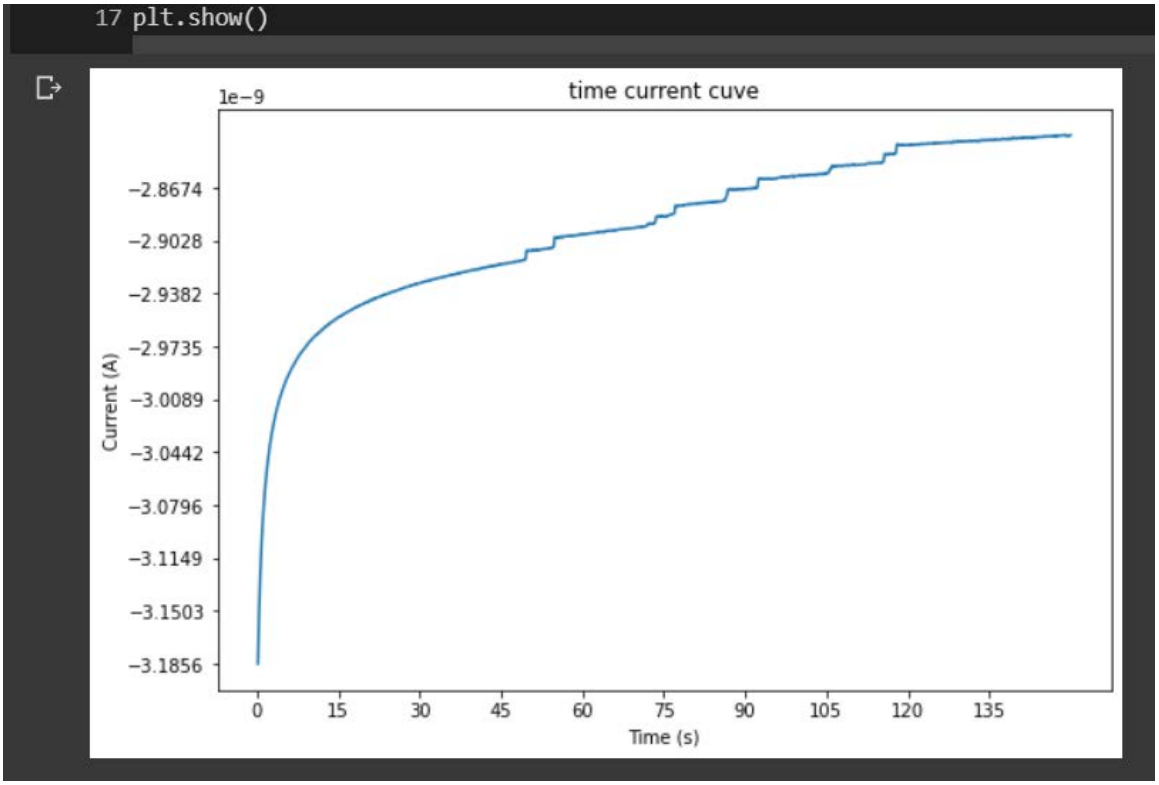


Figure S7. Screenshot of time-current curve.

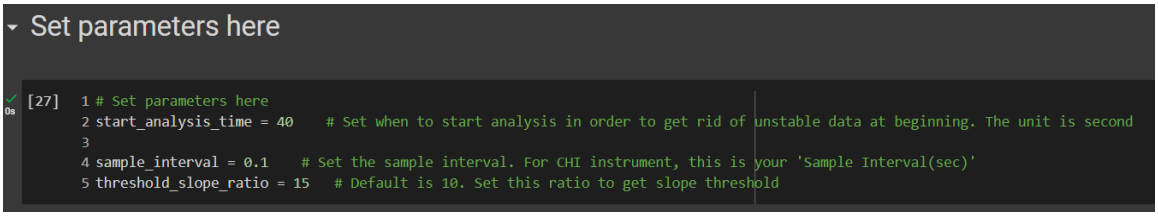
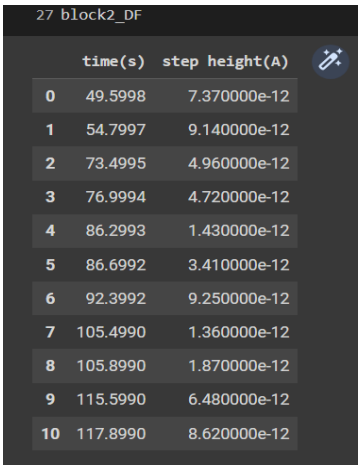


Figure S8. Screenshot of parameters setting.



	time(s)	step height(A)
0	49.5998	7.370000e-12
1	54.7997	9.140000e-12
2	73.4995	4.960000e-12
3	76.9994	4.720000e-12
4	86.2993	1.430000e-12
5	86.6992	3.410000e-12
6	92.3992	9.250000e-12
7	105.4990	1.360000e-12
8	105.8990	1.870000e-12
9	115.5990	6.480000e-12
10	117.8990	8.620000e-12

Figure S9. Screenshot of current step height.

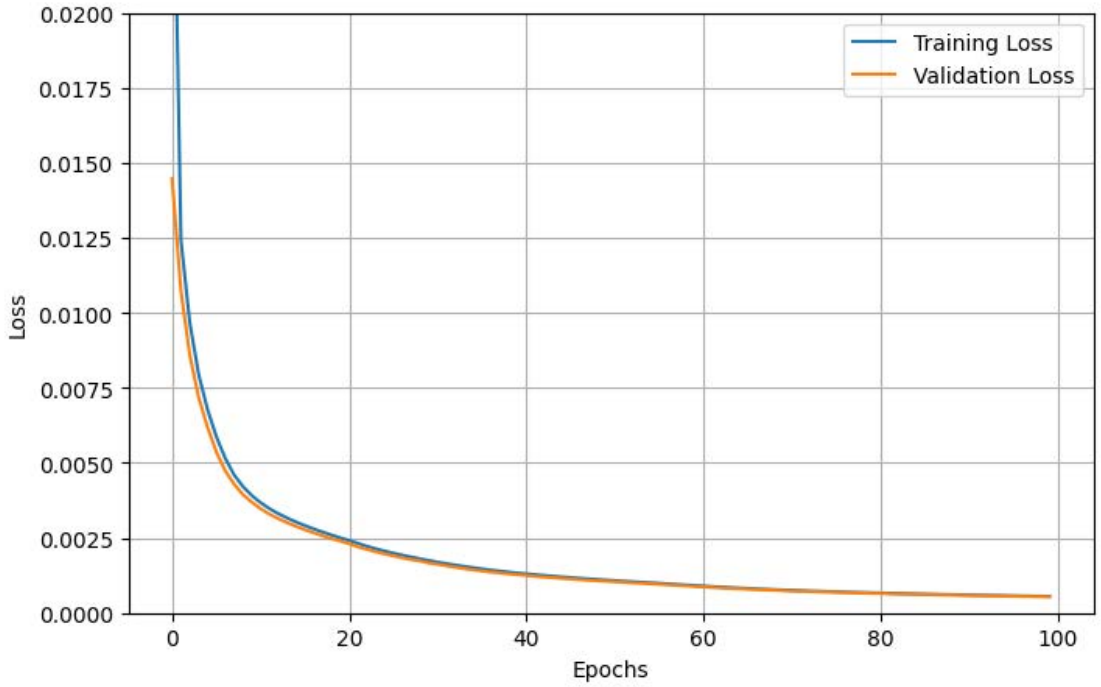


Figure S10. Comparison of training and validation losses for a NN model consisting of 30 neurons in a hidden layer. The training was done for 100 epochs and the learning rate was 0.003 (case 0 in a main paper).

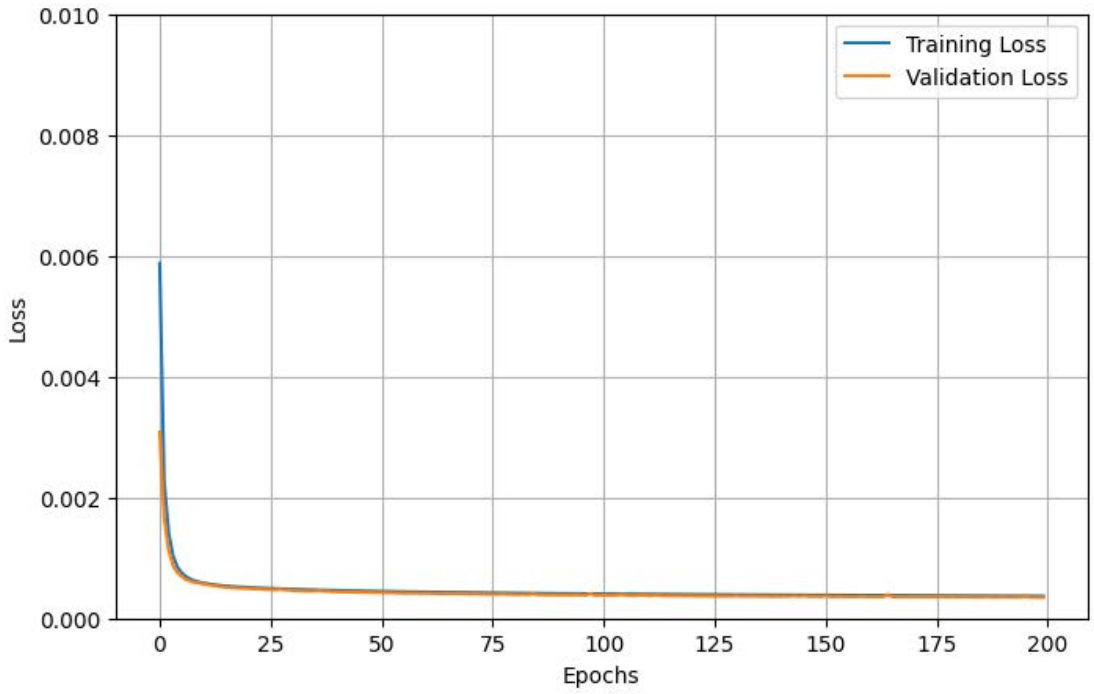


Figure S11. Comparison of training and validation losses for a NN model II consisting of 5 hidden layers with 60 neurons in each layer (see Table 1, case 3 in a main paper).

Experimental data for comparison of accuracy of prediction of NN models I and II

Polystyrene beads (PC04N from Bangs Laboratories, mean diameter 0.99 micron) at concentration 10 fM were detected in five independent blocking SEE experiments using a 5-micron radius Pt disk UME in 3 mM $FcCH_2OH$ and 1 mM KNO_3. Experimental collision *i-t* curves were analyzed, and the results are presented in Table S3 below. These results were further used to investigate mean prediction error by NN models I and II.

Table S3. Results of five independent blocking collision experiments together with model predictions and mean prediction error.

Experiment (total number of collisions)	Maximum current step, A	Minimum current step, A	Average current step, A	Prediction by model I, μm (% error)	Prediction by model II, μm (% error)
I and II (16)	1.351E-11	1.86E-12	6.15E-12	0.372 (-25%)	0.413 (-17%)
III and IV (15)	1.649E-11	1.70E-12	5.72E-12	0.401 (-19%)	0.430 (-13%)
V (17)	9.14E-12	2.04E-12	5.76E-12	0.315 (-36%)	0.366 (-26%)
Mean prediction error, %				-26.6%	-18.7%

Table S4. Effect of skewness parameter 'a' on predictions and their error. Predictions by NN model I are in the top cell, predictions by NN model II are in the bottom cell for each set of conditions.

Experiment (No of collisions)	a = 0 (% error)	a =4 (% error)	a=8 (% error)	a = -4 (% error)
I and II (16)	0.372 (-25%)	0.406 (-18%)	0.406 (-18%)	0.376 (-24%)
	0.413 (-17%)	0.473 (-4.4%)	0.438 (-12%)	0.379 (-23%)
III and IV (15)	0.401 (-19%)	0.425 (-14%)	0.430 (-13%)	0.391 (-21%)
	0.430 (-13%)	0.499 (0.81%)	0.465 (-6.1%)	0.401 (-19%)
V (17)	0.315 (-36%)	0.366 (-26%)	0.356 (-28%)	0.338 (-32%)
	0.366 (-26%)	0.416 (-16%)	0.380 (-23%)	0.328 (-34%)
mean prediction error	-26.6%	-19.3%	-19.7%	-25.6%
	-18.7%	-6.5%	-13.7%	-25.3%

References

(1) Zhang, B.; Frkonja-Kuczin, A.; Duan, Z.-H.; Boika, A. Workshop on Computer Vision for Bioanalytical Chemists: Classification and Detection of Amoebae Using Optical Microscopy Image Analysis with Machine Learning. *J. Chem. Educ.* **2023**, acs.jchemed.2c00631. https://doi.org/10.1021/acs.jchemed.2c00631.

Code for current step height determination

```
1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 import statistics
```

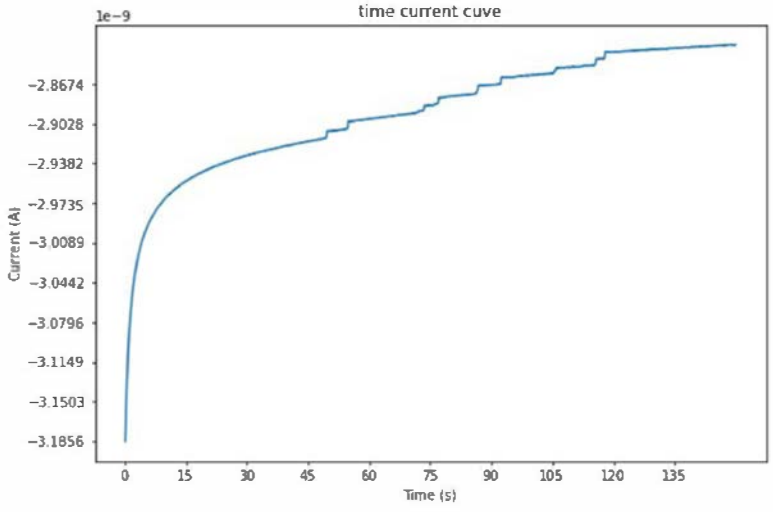
Change your file name here

```
1 # read and clean data
2
3 it_df = pd.read_csv("i-t_example1.txt", header=0)    # "i-t_example1.txt" is converted from CHI software directly
4 it_df = it_df.drop(range(0,13))   # drop the unused data
5 it_df = it_df.reset_index(drop=True)   # reset it_df index
6 it_df.columns = ['Time(s)', 'Current(A)']   # Renaming the columns name
```

```
1 # display all rows
2
3 # pd.set_option('display.max_rows', None)
4 # it_df
```

```
1 # Convert data from string to float type
2 it_df = it_df.astype('float')
```

```
1 plt.figure(figsize=(9,6))
2 plt.plot(it_df['Time(s)'], it_df['Current(A)'])
3
4 # Set the interval of xticks.
5 listOf_Xticks = np.arange(0, max(it_df['Time(s)']), max(it_df['Time(s)'])/10)
6 plt.xticks(listOf_Xticks)
7 # Set the interval of yticks.
8 listOf_Yticks = np.arange(min(it_df['Current(A)']), max(it_df['Current(A)']), abs(max(it_df['Current(A)'])-min(it_df['Current(A)']))/10)
9 plt.yticks(listOf_Yticks)
10
11 # Set the x and y label
12 plt.xlabel("Time (s)")
13 plt.ylabel("Current (A)")
14 # Giving title to the plot
15 plt.title('time current cuve')
16
17 plt.show()
```

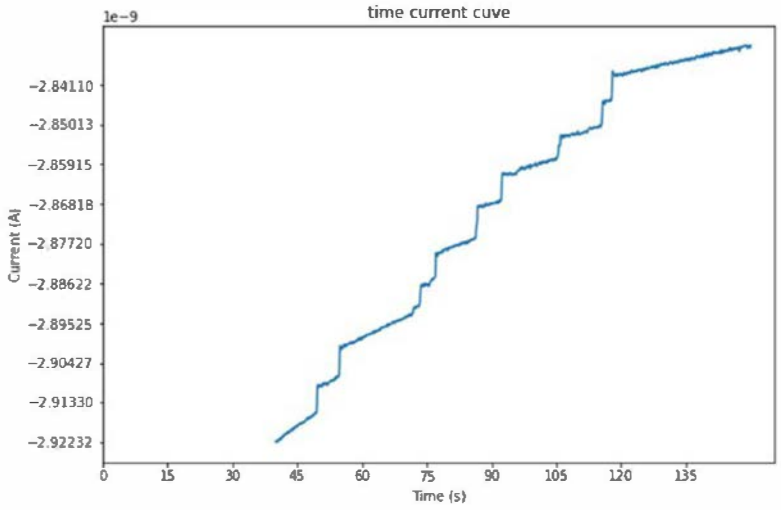


Set parameters here

```
# Set parameters here
start_analysis_time = 40    # Set when to start analysis in order to get rid of unstable data at beginning. The unit is second

sample_interval = 0.1    # Set the sample interval. For CHI instrument, this is your 'Sample Interval(sec)'
threshold_slope_ratio = 15   # Default is 10. Set this ratio to get slope threshold
```

```
# drop the beginning data

start_analysis_index_list = it_df.index[it_df['Time(s)'] == float(start_analysis_time)].to_list()   # find index based on time
start_analysis_index = start_analysis_index_list[0]
it_df = it_df.drop(range(0, start_analysis_index))   # drop the beginning data
it_df = it_df.reset_index(drop=True)   # reset it_df index
```

```
plt.figure(figsize=(9,6))
plt.plot(it_df['Time(s)'], it_df['Current(A)'])

# Set the interval of xticks.
listOf_Xticks = np.arange(0, max(it_df['Time(s)']), max(it_df['Time(s)'])/10)
plt.xticks(listOf_Xticks)
# Set the interval of yticks.
listOf_Yticks = np.arange(min(it_df['Current(A)']), max(it_df['Current(A)']), abs(max(it_df['Current(A)'])-min(it_df['Current(A)']))/10)
plt.yticks(listOf_Yticks)

# Set the x and y label
plt.xlabel("Time (s)")
plt.ylabel("Current (A)")
# Giving title to the plot
plt.title('time current cuve')

plt.show()
```




```
slope_list = []
for i in range(11):
    current_next = float(it_df.iloc[i+1, 1])
    current_now = float(it_df.iloc[i, 1])
    time_next=float(it_df.iloc[i+1,0])
    time_now=float(it_df.iloc[i-1,0])

    dif_current = current_next - current_now
    dif_time=time_next - time_now
    slope = dif_current/dif_time
    slope = abs(slope)
    slope_list.append(slope)

# We took absolute median value of first 11 points as slope
slope =  abs(statistics.median(slope_list))
threshold_slope = slope * threshold_slope_ratio
threshold_slope   # slope that is larger than threshold_slope will considered step height
```

```
5.99999999998956e-12
```

```
# read block step height

index_pre = -1
blockDF = pd.DataFrame(columns = ['time(s)', 'step height(A)'])

for index, row in it_df.iterrows():
    if index == len(it_df['Time(s)'])-1:
        break
    else:
        current_next = float(it_df.iloc[index+1, 1])
        current_now = float(it_df.iloc[index, 1])
        time_next=float(it_df.iloc[index+1,0])
        time_now=float(it_df.iloc[index-1,0])

        dif_current = current_next - current_now
        dif_time=time_next - time_now
        slope = dif_current/dif_time
        one_step_height = dif_current

        if slope > threshold_slope:
            if index - index_pre == 1:
                step_height = step_height + one_step_height
            else:
                step_height= dif_current
            oneStepHeight_df = pd.DataFrame([[it_df.iloc[index,0], step_height]], columns=['time(s)', 'step height(A)'])
            blockDF = pd.concat([blockDF, oneStepHeight_df], ignore_index = True)
        else:
            step_height = 0
        index_pre = index_pre + 1

# blockDF
```

```
# combine closed step height

blockDF_cumu = blockDF.copy()
for index, row in blockDF.iterrows():
    if index == 0:
        continue
    else:
        time1 = float(blockDF.iloc[index-1,0])
        time2 = float(blockDF.iloc[index,0])
        time_dif = round(time2-time1, 2)
        if time_dif == float(sample_interval):
            step_height = float(blockDF_cumu.iloc[index-1,1]) + float(blockDF.iloc[index,1])
            blockDF_cumu['step height(A)'].iloc[index] = step_height
block2_DF = blockDF_cumu.copy()

for index, row in blockDF_cumu.iterrows():
    if index == 0:
        continue
    else:
        time1 = float(blockDF_cumu.iloc[index-1,0])
        time2 = float(blockDF_cumu.iloc[index,0])
        time_dif = round(time2-time1, 2)
        if time_dif == float(sample_interval):
            block2_DF = block2_DF.drop([index-1])

block2_DF = block2_DF.reset_index(drop=True)
block2_DF
```

	time(s)	step height(A)
0	49.5998	7.370000e-12
1	54.7997	9.140000e-12
2	73.4995	4.960000e-12
3	76.9994	4.720000e-12
4	86.2993	1.430000e-12
5	86.6992	3.410000e-12
6	92.3992	9.250000e-12
7	105.4990	1.360000e-12
8	105.8990	1.870000e-12
9	115.5990	6.480000e-12
10	117.8990	8.620000e-12

```
block2_DF.to_excel('block events.xlsx')
```

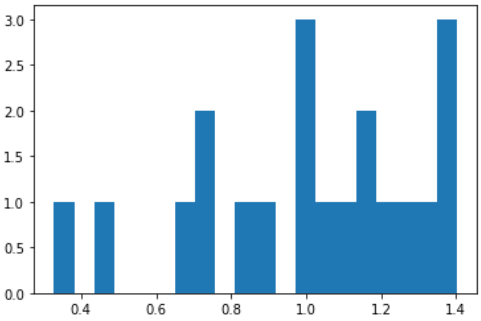
Code for dataset generation - 1 size particles (cases 1-2)

In [1]:
```
#import libraries and define constants
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn as skl
import random
import scipy.stats as st
from csv import writer
%matplotlib inline
N = 20 #number of collisions
Cb = 2
el_rad = 5E-6
D = 0.7E-9
F = 96485
I_ss = 4*F*D*Cb*el_rad
print(I_ss) #steady-state limiting current
```

```
2.70158e-09
```

In [2]:
```
#Random distribution of collisions across electrode surface; loc=1 is the disk edge
Rand = np.random.normal(loc=1.0, scale=0.333, size=N)
plt.hist(Rand, bins = 20)
```

Out[2]:
```
(array([1., 0., 1., 0., 0., 0., 1., 2., 0., 1., 1., 0., 3., 1., 1., 2., 1.,
        1., 1., 3.]),
 array([0.32799055, 0.38166866, 0.43534677, 0.48902488, 0.54270299,
        0.5963811 , 0.6500592 , 0.70373731, 0.75741542, 0.81109353,
        0.86477164, 0.91844975, 0.97212786, 1.02580597, 1.07948407,
        1.13316218, 1.18684029, 1.2405184 , 1.29419651, 1.34787462,
        1.40155273]),
 <BarContainer object of 20 artists>)
```

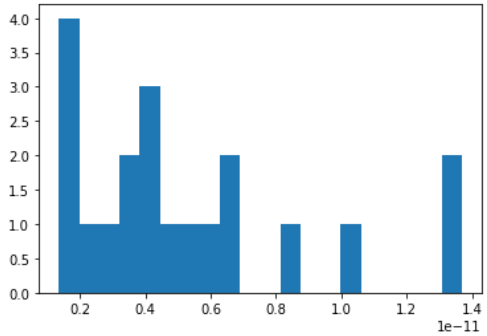


In [3]:
```
for i in range(N):
    if Rand[i] < 0:
        Rand[i] = 0
```

In [4]:
```
#calculating current step magnitude
r_p=0.5E-6
In=[]
for i in range(N):
    if Rand[i] > 1:
        Fg = -0.7373*Rand[i]**3 + 4.0209*Rand[i]**2 - 7.3858*Rand[i] + 4.6088
        current = 1.0806E-09*Fg*(r_p/el_rad)**2
        In.append(current)
    else:
        Fg = 8.9921E-01*Rand[i]**3 - 8.7335E-01*Rand[i]**2 + 4.0039E-01*Rand[i] + 8.0542E-02
        current = I_ss*Fg*(r_p/el_rad)**2
        In.append(current)
```

In [5]:
```
#plotting the data
plt.hist(In, bins=20)
```

Out[5]:
```
(array([4., 1., 1., 2., 3., 1., 1., 1., 2., 0., 0., 1., 0., 0., 1., 0., 0.,
        0., 0., 2.]),
 array([1.35914332e-12, 1.97530923e-12, 2.59147513e-12, 3.20764104e-12,
        3.82380694e-12, 4.43997285e-12, 5.05613875e-12, 5.67230466e-12,
        6.28847056e-12, 6.90463647e-12, 7.52080237e-12, 8.13696828e-12,
        8.75313418e-12, 9.36930009e-12, 9.98546599e-12, 1.06016319e-11,
        1.12177978e-11, 1.18339637e-11, 1.24501296e-11, 1.30662955e-11,
        1.36824614e-11]),
 <BarContainer object of 20 artists>)
```



In [6]:
```
#average current step
av = sum(In)/N
print(av)
```

```
5.303512557025549e-12
```

In [7]:
```
import statistics
statistics.median(In)
```

Out[7]:
```
4.307952121034172e-12
```

In [8]:
```
print(min(In))
print(max(In))
```

```
1.3591433201834274e-12
1.3682461425255355e-11
```

In [9]:
```
# Adding el_rad, Cb and D to a sample datafile
In.append(el_rad)
In.append(Cb)
In.append(D)
df = pd.DataFrame(In)

# saving the dataframe
#df.to_csv('data_example.csv')
```

In [10]:
```
def save_data(data, filename, label):
    data_list = data
    data_list.insert(0, label)
    # writing to csv file
    with open(filename, 'a', newline='') as f_object:
        # Pass the CSV  file object to the writer() function
        writer_object = writer(f_object)
        # Result - a writer object
        # Pass the data in the list as an argument into the writerow() function
        writer_object.writerow(data_list)
        # Close the file object
        f_object.close()
    return
```

In [11]:
```
#routine for synthetic data generation
runs = 250
D_list = [2e-9, 1e-9,5e-10]
Cb_list = [10, 5, 1]
r_p_list = [0.025e-6, 0.05e-6, 0.1e-6, 0.15e-6, 0.2e-6, 0.25e-6, 0.3e-6, 0.35e-6, 0.4e-6, 0.45e-6, 0.5e-6]
el_rad_list = [5e-6, 2.5e-6, 1e-6]
In = []
In_pass = []
Av = []
Med = []
Max = []
Min = []
#Bin_Max = []
data_bank=[]

for D in D_list:
    for Cb in Cb_list:
        for el_rad in el_rad_list:
            for r_p in r_p_list:
                I_ss=4*F*D*Cb*el_rad

                for l in range(runs):
                    Rand = np.random.normal(loc=1.0, scale=0.333, size=N)
                    for i in range(N):
                        if Rand[i] < 0:
                            Rand[i] = 0
                    for i in range(N):
                        if Rand[i] > 1:
```

```
            Fg = -0.73725*Rand[i]**3 + 4.0209*Rand[i]**2 - 7.3858*Rand[i] + 4.6088
            current = I_ss*Fg*(r_p/el_rad)**2
            In.append(current)
        else:
            Fg = 8.9921E-01*Rand[i]**3 - 8.7335E-01*Rand[i]**2 + 4.0039E-01*Rand[i] + 8.0542E-02
            current = I_ss*Fg*(r_p/el_rad)**2
            In.append(current)
    data_bank.append(D/1e-9)
    data_bank.append(Cb)
    data_bank.append(el_rad/1e-6)
    average = sum(In)/N
    maximum = max(In)
    minimum = min(In)
    data_bank.append(maximum/I_ss*1000.)
    data_bank.append(minimum/I_ss*1000.)
    data_bank.append(average/I_ss*1000.)

    for j in range(len(In)):
        In_pass.append(In[j]/I_ss*1000.0)
    #save data
    save_data(data_bank,'train_data_regr20.csv',r_p/1e-6)
    In.clear()
    In_pass.clear()
    data_bank.clear()
```

In [12]:

```
#routine for experimental data recording
data_exp = []
D = 0.7e-9
Cb = 2
el_rad = 5e-6
F = 96485
r_p = 0
I_ss = 4*F*D*Cb*el_rad
maximum = 17.51e-12 #here and below are example values
minimum = 1.6e-12
average = 5.16e-12
data_exp.append(D/1e-9)
data_exp.append(Cb)
data_exp.append(el_rad/1e-6)
data_exp.append(maximum/I_ss*1000.)
data_exp.append(minimum/I_ss*1000.)
data_exp.append(average/I_ss*1000.)
save_data(data_exp,'unknown_data_regr.csv',r_p/1e-6)
```

In [ ]:

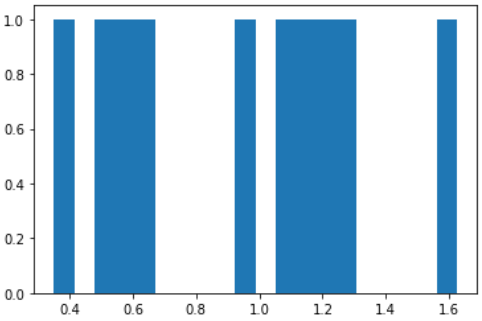
Code for dataset generation - 2 sizes of particles (cases 3)

In [1]:
```
#import libraries and define constants
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn as skl
import random
import scipy.stats as st
from csv import writer
%matplotlib inline
N = 10 #number of collisions of each particle population; total number of collisions is 2N
Cb = 3
el_rad = 5E-6
D = 0.7E-9
F = 96485
I_ss = 4*F*D*Cb*el_rad
print(I_ss) #steady-state limiting current
```

```
4.052370000000001e-09
```

In [2]:
```
#Random distribution of collisions across electrode surface; loc=1 is the disk edge
Rand = np.random.normal(loc=1.0, scale=0.333, size=N)
plt.hist(Rand, bins = 20)
```

Out[2]:
```
(array([1., 0., 1., 1., 1., 0., 0., 0., 0., 1., 0., 1., 1., 1., 1., 0., 0.,
        0., 0., 1.]),
 array([0.35104915, 0.41468143, 0.47831371, 0.54194599, 0.60557827,
        0.66921055, 0.73284283, 0.79647511, 0.86010739, 0.92373967,
        0.98737195, 1.05100423, 1.11463651, 1.17826879, 1.24190107,
        1.30553335, 1.36916563, 1.4327979 , 1.49643018, 1.56006246,
        1.62369474]),
 <BarContainer object of 20 artists>)
```

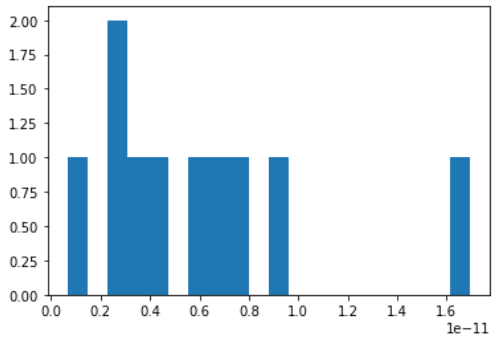


In [3]:
```
for i in range(N):
    if Rand[i] < 0:
        Rand[i] = 0
```

In [4]:
```
#calculating current step magnitude
r_p=0.5E-6
In=[]
for i in range(N):
    if Rand[i] > 1:
        Fg = -0.7373*Rand[i]**3 + 4.0209*Rand[i]**2 - 7.3858*Rand[i] + 4.6088
        current = 1.0806E-09*Fg*(r_p/el_rad)**2
        In.append(current)
    else:
        Fg = 8.9921E-01*Rand[i]**3 - 8.7335E-01*Rand[i]**2 + 4.0039E-01*Rand[i] + 8.0542E-02
        current = I_ss*Fg*(r_p/el_rad)**2
        In.append(current)
```

In [5]:
```
plt.hist(In, bins=20)
```

Out[5]:
```
(array([1., 0., 2., 1., 1., 0., 1., 1., 1., 0., 1., 0., 0., 0., 0., 0., 0.,
        0., 0., 1.]),
 array([6.59231700e-13, 1.47415506e-12, 2.28907843e-12, 3.10400179e-12,
        3.91892515e-12, 4.73384852e-12, 5.54877188e-12, 6.36369524e-12,
        7.17861861e-12, 7.99354197e-12, 8.80846534e-12, 9.62338870e-12,
        1.04383121e-11, 1.12532354e-11, 1.20681588e-11, 1.28830822e-11,
        1.36980055e-11, 1.45129289e-11, 1.53278522e-11, 1.61427756e-11,
        1.69576990e-11]),
 <BarContainer object of 20 artists>)
```



In [6]:
```
#average current step
av = sum(In)/N
print(av)
```

```
6.0446877080113614e-12
```

In [7]:
```
import statistics
statistics.median(In)
```

Out[7]:
```
5.357316916225565e-12
```

In [8]:
```
print(min(In))
print(max(In))
```

```
6.592316997088986e-13
1.6957698970832715e-11
```

In [9]:
```
# Adding el_rad, Cb and D
In.append(el_rad)
In.append(Cb)
In.append(D)
df = pd.DataFrame(In)

# saving the dataframe
#df.to_csv('data_example2.csv')
```

In [10]:
```
def save_data(data, filename, label_av):
    data_list = data

    data_list.insert(0, label_av)
    # writing to csv file
    with open(filename, 'a', newline='') as f_object:
        # Pass the CSV  file object to the writer() function
        writer_object = writer(f_object)
        # Result - a writer object
        # Pass the data in the list as an argument into the writerow() function
        writer_object.writerow(data_list)
        # Close the file object
        f_object.close()
    return
```

In [11]:
```
#routine for synthetic data generation
runs = 20
D_list = [2e-9, 1e-9,5e-10]
Cb_list = [10, 5, 1]
r_p1_list = [0.025e-6, 0.05e-6, 0.1e-6, 0.15e-6, 0.2e-6, 0.25e-6, 0.3e-6, 0.35e-6, 0.4e-6, 0.45e-6, 0.5e-6]
r_p2_list = [0.03e-6, 0.055e-6, 0.105e-6, 0.155e-6, 0.205e-6, 0.255e-6, 0.305e-6, 0.355e-6, 0.405e-6, 0.455e-6, 0.505e-6]
el_rad_list = [5e-6, 2.5e-6, 1e-6]
In = []
In_pass = []
Av = []
Med = []
Max = []
Min = []
Bin_Max = []
data_bank=[]

for D in D_list:
    for Cb in Cb_list:
        for el_rad in el_rad_list:
            for r_p1 in r_p1_list:
                for r_p2 in r_p2_list:
                    #
                    I_ss=4*F*D*Cb*el_rad

                    for l in range(runs):
                        Rand1 = np.random.normal(loc=1.0, scale=0.333, size=N)
                        for i in range(N):
```

```
            if Rand1[i] < 0:
                Rand1[i] = 0
        for i in range(N):
            if Rand1[i] > 1:
                Fg1 = -0.73725*Rand1[i]**3 + 4.0209*Rand1[i]**2 - 7.3858*Rand1[i] + 4.6088
                current1 = I_ss*Fg1*(r_p1/el_rad)**2

                In.append(current1)

            else:
                Fg1 = 8.9921E-01*Rand1[i]**3 - 8.7335E-01*Rand1[i]**2 + 4.0039E-01*Rand1[i] + 8.0542E-02
                current1 = I_ss*Fg1*(r_p1/el_rad)**2

                In.append(current1)

        Rand2 = np.random.normal(loc=1.0, scale=0.333, size=N)
        for j in range(N):
            if Rand2[j] < 0:
                Rand2[j] = 0
        for j in range(N):
            if Rand2[j] > 1:
                Fg2 = -0.73725*Rand2[j]**3 + 4.0209*Rand2[j]**2 - 7.3858*Rand2[j] + 4.6088
                current2 = I_ss*Fg2*(r_p2/el_rad)**2

                In.append(current2)

            else:
                Fg2 = 8.9921E-01*Rand2[j]**3 - 8.7335E-01*Rand2[j]**2 + 4.0039E-01*Rand2[j] + 8.0542E-02
                current2 = I_ss*Fg2*(r_p2/el_rad)**2

                In.append(current2)

        data_bank.append(D/1e-9)
        data_bank.append(Cb)
        data_bank.append(el_rad/1e-6)
        average = sum(In)/2/N
        maximum = max(In)
        minimum = min(In)
        data_bank.append(maximum/I_ss*1000.)
        data_bank.append(minimum/I_ss*1000.)
        data_bank.append(average/I_ss*1000.)

        for j in range(len(In)):
            In_pass.append(In[j]/I_ss*1000.0)

        save_data(data_bank,'test_data_regr20_1m.csv',round((r_p1+r_p2)/2e-6,4))
        In.clear()
        In_pass.clear()
        data_bank.clear()
```

In [12]:

```
#routine for experimental data recording
data_exp = []
D = 0.7e-9
Cb = 3
el_rad = 5e-6
F = 96485
r_p1 = 0
r_p2 = 0
I_ss = 4*F*D*Cb*el_rad
maximum = 51.54e-12
minimum = 7.63e-12
average = 27.87e-12
data_exp.append(D/1e-9)
data_exp.append(Cb)
data_exp.append(el_rad/1e-6)
data_exp.append(maximum/I_ss*1000.)
data_exp.append(minimum/I_ss*1000.)
data_exp.append(average/I_ss*1000.)
save_data(data_exp,'unknown_data_regr_Gabr_1m.csv',round((r_p1+r_p2)/2e-6,4))
```

In [ ]:

Code for dataset generation - 1 size of particles (skewed distribution)

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn as skl
import random
import scipy.stats as st
from csv import writer
%matplotlib inline
N = 20 #number of collisions
Cb = 2
el_rad = 5E-6
D = 0.7E-9
F = 96485
I_ss = 4*F*D*Cb*el_rad
# Parameters for skewed normal distribution
loc = 1.0  # Desired peak position (edge of the disk)
scale = 0.333  # Standard deviation
a = 8  # Skewness parameter for large skewness, 0 for normal distribution
```

In [ ]:
```
# Generate skewed normal distribution
Rand = st.skewnorm.rvs(a, loc=0, scale=scale, size=N)

# Calculate the mode of the distribution
delta = a / np.sqrt(1 + a**2)
mode_shift = delta * scale * np.sqrt(2 / np.pi)

# Adjust the distribution to ensure peak at loc=1.0
Rand = Rand - mode_shift + loc
```

In [ ]:
```
plt.hist(Rand, bins=20)
```

In [ ]:
```
# Ensure all values are non-negative
Rand[Rand < 0] = 0
```

In [ ]:
```
r_p=0.5E-6
In=[]
for i in range(N):
    if Rand[i] > 1:
        Fg = -0.7373*Rand[i]**3 + 4.0209*Rand[i]**2 - 7.3858*Rand[i] + 4.6088
        current = 1.0806E-09*Fg*(r_p/el_rad)**2
        In.append(current)
    else:
        Fg = 8.9921E-01*Rand[i]**3 - 8.7335E-01*Rand[i]**2 + 4.0039E-01*Rand[i] + 8.0542E-02
        current = I_ss*Fg*(r_p/el_rad)**2
        In.append(current)
print(type(In))
len(In)
```

In [ ]:
```
plt.hist(In, bins=20)
```

In [ ]:
```
av = sum(In)/N
print(av)
```

In [ ]:
```
import statistics
statistics.median(In)
```

In [ ]:
```
print(min(In))
print(max(In))
```

In [ ]:
```
n, b, patches = plt.hist(In, bins = 20)

bin_max = np.where(n == n.max())

print('maxbin', b[bin_max][0])
```

In [ ]:
```
# Adding el_rad, Cb and D
In.append(el_rad)
In.append(Cb)
In.append(D)
df = pd.DataFrame(In)

# saving the dataframe
df.to_csv('data_example.csv')
```

In [ ]:
```
def conf_int(results):
    interval = st.t.interval(alpha=0.95, df=len(results)-1, loc=np.mean(results), scale=st.sem(results))
    return interval
```

In [ ]:
```
def save_data(data, filename, label):
    data_list = data
```

```
    data_list.insert(0, label)
    # writing to csv file
    with open(filename, 'a', newline='') as f_object:
        # Pass the CSV  file object to the writer() function
        writer_object = writer(f_object)
        # Result - a writer object
        # Pass the data in the list as an argument into the writerow() function
        writer_object.writerow(data_list)
        # Close the file object
        f_object.close()
    return
```

In [ ]:
```
runs = 50
D_list = [2e-9, 1e-9,5e-10]
Cb_list = [10, 5, 1]
r_p_list = [0.025e-6, 0.05e-6, 0.1e-6, 0.15e-6, 0.2e-6, 0.25e-6, 0.3e-6, 0.35e-6, 0.4e-6, 0.45e-6, 0.5e-6]
el_rad_list = [5e-6, 2.5e-6, 1e-6]
In = []
In_pass = []
Av = []
Med = []
Max = []
Min = []
Bin_Max = []
data_bank=[]

for D in D_list:
    for Cb in Cb_list:
        for el_rad in el_rad_list:
            for r_p in r_p_list:
                I_ss=4*F*D*Cb*el_rad

                for l in range(runs):
                    Rand = np.random.normal(loc=1.0, scale=0.333, size=N)
                    for i in range(N):
                        if Rand[i] < 0:
                            Rand[i] = 0
                    for i in range(N):
                        if Rand[i] > 1:
                            Fg = -0.73725*Rand[i]**3 + 4.0209*Rand[i]**2 - 7.3858*Rand[i] + 4.6088
                            current = I_ss*Fg*(r_p/el_rad)**2
                            In.append(current)
                        else:
                            Fg = 8.9921E-01*Rand[i]**3 - 8.7335E-01*Rand[i]**2 + 4.0039E-01*Rand[i] + 8.0542E-02
                            current = I_ss*Fg*(r_p/el_rad)**2
                            In.append(current)
                    data_bank.append(D/1e-9)
                    data_bank.append(Cb)
                    data_bank.append(el_rad/1e-6)
                    average = sum(In)/N
                    Av.append(average)
                    median = statistics.median(In)
                    Med.append(median)
                    maximum = max(In)
                    Max.append(maximum)
                    minimum = min(In)
                    Min.append(minimum)
                    n, b, patches = plt.hist(In, bins = 20)
                    bin_max = np.where(n == n.max())
                    maxbin = b[bin_max][0]
                    Bin_Max.append(maxbin)
                    data_bank.append(maximum/I_ss*1000.)
                    data_bank.append(minimum/I_ss*1000.)
                    data_bank.append(average/I_ss*1000.)
                    #print(len(In))
                    for j in range(len(In)):
                        In_pass.append(In[j]/I_ss*1000.0)
                    #In_pass = [In / I_ss for item in In]
                    save_data(data_bank,'test_data_regr20_skew+8.csv',r_p/1e-6)
                    In.clear()
                    In_pass.clear()
                    data_bank.clear()

print("Data generation completed.")
```

In [ ]:
```
Av_int = conf_int(Av)
Av_val = sum(Av)/runs
print("Average = ", Av_val, Av_int)
Med_int = conf_int(Med)
Med_val = sum(Med)/runs
print("Median = ", Med_val, Med_int)
Max_int = conf_int(Max)
Max_val = sum(Max)/runs
print("Maximum = ", Max_val, Max_int)
```

```
Min_int = conf_int(Min)
Min_val = sum(Min)/runs
print("Minimum = ", Min_val, Min_int)
Bin_Max_int = conf_int(Bin_Max)
Bin_Max_val = sum(Bin_Max)/runs
print("Bin_Max = ", Bin_Max_val, Bin_Max_int)
```

In [ ]:
```
data_exp = []
D = 0.7e-9
Cb = 3
el_rad = 5e-6
F = 96485
r_p = 0
I_ss = 4*F*D*Cb*el_rad
maximum = 16.49e-12
minimum = 1.7e-12
average = 5.72e-12
data_exp.append(D/1e-9)
data_exp.append(Cb)
data_exp.append(el_rad/1e-6)
data_exp.append(maximum/I_ss*1000.)
data_exp.append(minimum/I_ss*1000.)
data_exp.append(average/I_ss*1000.)
save_data(data_exp,'unknown_data_regr_Gabr_repeat7.csv',r_p/1e-6)
```

In [ ]:
```

```

Code for NN architecture optimization

In [ ]:
```
#install required packages
#!pip3 install --upgrade pip setuptools wheel
#!pip3 install scikit-learn scipy keras pandas tensorflow numpy scikeras
```

In [ ]:
```
#import required libraries
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import scipy
import keras
import pandas as pd
import tensorflow as tf
import numpy as np
import scikeras
from scikeras.wrappers import KerasClassifier, KerasRegressor
```

In [ ]:
```
#Load data
train_data = pd.read_csv("~/ML blocking collisions/blocking_dataset/train_da
test_data = pd.read_csv("~/ML blocking collisions/blocking_dataset/test_data
# Shuffle data
train_data = train_data.sample(frac=1).reset_index(drop=True)
test_data = test_data.sample(frac=1).reset_index(drop=True)
#Select features and labels
train_features = train_data.iloc[:,1:]
train_labels = train_data.iloc[:,0]
test_features = test_data.iloc[:,1:]
test_labels = test_data.iloc[:,0]
train_features_full = np.array(train_features)
train_labels_full = np.array(train_labels)
test_features_full = np.array(test_features)
test_labels_full = np.array(test_labels)
```

In [ ]:
```
#split the data
X_train, X_valid, y_train, y_valid = train_test_split(train_features_full, t
X_test = test_features_full
y_test = test_labels_full
```

In [ ]:
```
#scale the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_valid = scaler.transform(X_valid)
X_test = scaler.transform(X_test)
```

In [ ]:
```
#build model
def build_model(n_hidden = 1, n_neurons = 30, learning_rate = 3e-3, input_sh
    model = keras.models.Sequential()
    model.add(keras.layers.InputLayer(input_shape = input_shape))
    for layer in range(n_hidden):
        model.add(keras.layers.Dense(n_neurons, activation = "relu"))
    model.add(keras.layers.Dense(1))
    optimizer = keras.optimizers.SGD(learning_rate=learning_rate)
    model.compile(loss = "mse", optimizer=optimizer)
    return model
```

In [ ]:
```
#create a KerasRegressor
#keras_reg = scikeras.wrappers.KerasRegressor(build_model)
keras_reg = scikeras.wrappers.KerasRegressor(
    build_model,
    n_neurons=91,
    n_hidden=3,
    optimizer__lr=3e-3,  # just specify parameters here, you won't have to e
    optimizer=keras.optimizers.SGD,  # note that I am giving SciKeras a clas
   loss="mse",
)
```

In [ ]:
```
keras_reg.fit(X_train, y_train,epochs = 100, validation_data = (X_valid,y_va
mse_test = keras_reg.score(X_test, y_test)
X_new = X_test
y_pred = keras_reg.predict(X_new)
```

In [ ]:
```
print(y_pred)
print(y_test)
```

In [ ]:
```
#explore
from scipy.stats import reciprocal
from sklearn.model_selection import RandomizedSearchCV

param_distribs = {
    "n_hidden": [0, 1, 2, 3, 4, 5],
    #"n_neurons": np.arange(1, 100),
    "n_neurons" : np.arange(1, 100, 10),
    #"optimizer__lr": reciprocal(3e-4, 3e-2),
    "optimizer__lr" : np.arange(3e-4, 3e-2, 0.3*(1e-2 - 1e-4)),
}

rnd_search_cv = RandomizedSearchCV(keras_reg, param_distribs, n_iter=10, cv=
rnd_search_cv.fit(
    X_train,
    y_train,
    epochs=25,
    validation_data=(X_valid, y_valid),
    callbacks=[keras.callbacks.EarlyStopping(patience=10)],
)
```

In [ ]:
```
rnd_search_cv.best_params_
```

In [ ]:
```
#{'optimizer__lr': 0.01218, 'n_neurons': 91, 'n_hidden': 3}
#{'optimizer__lr': 0.01515, 'n_neurons': 61, 'n_hidden': 5}
```

Code for NN model for 1 size particles (cases 1-2)

In [ ]:
```
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow import keras
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import joblib
```

In [ ]:
```
#Load data
train_data = pd.read_csv("blocking_dataset/train_data_regr20.csv", header = None)
test_data = pd.read_csv("blocking_dataset/test_data_regr20.csv", header = None)
#unknown_data = pd.read_csv("blocking_dataset/unknown_data_regr_Gabr2.csv", header = None)
# Shuffle data
train_data = train_data.sample(frac=1).reset_index(drop=True)
test_data = test_data.sample(frac=1).reset_index(drop=True)
```

In [ ]:
```
#train_data.head()
test_data.head()
```

In [ ]:
```
train_features = train_data.iloc[:,1:]
train_labels = train_data.iloc[:,0]
test_features = test_data.iloc[:,1:]
test_labels = test_data.iloc[:,0]
#unknown_features = unknown_data.iloc[:,1:]
```

In [ ]:
```
print(test_labels.head())
print(type(test_labels))
```

In [ ]:
```
train_features_full = np.array(train_features)
train_labels_full = np.array(train_labels)
test_features_full = np.array(test_features)
test_labels_full = np.array(test_labels)
#unknown_features_full = np.array(unknown_features)
```

In [ ]:
```
#scaling data
scaler = StandardScaler()
train_features_full=scaler.fit_transform(train_features_full)
test_features_full = scaler.transform(test_features_full)
#unknown_features_full = scaler.transform(unknown_features_full)
```

In [ ]:
```
print(test_features_full[0])
```

In [ ]:
```
#Defining validation data
train_features_valid, train_features = train_features_full[:7425], train_features_full[7425:]
train_labels_valid, train_labels = train_labels_full[:7425], train_labels_full[7425:]
```

In [ ]:
```
# Save the scaler
joblib.dump(scaler, "scaler_blocking_regr20.save")
```

In [ ]:
```
#Defyning NN model
model = keras.models.Sequential()
model.add(keras.layers.InputLayer(shape=(train_features_full.shape[1],)))
model.add(keras.layers.Dense(60, activation="relu"))
model.add(keras.layers.Dense(60, activation="relu"))
model.add(keras.layers.Dense(60, activation="relu"))
model.add(keras.layers.Dense(60, activation="relu"))
model.add(keras.layers.Dense(60, activation="relu"))
model.add(keras.layers.Dense(1))
model.summary()
```

In [ ]:
```
#Compile the model
model.compile(loss="mean_squared_error", optimizer=keras.optimizers.SGD(learning_rate = 0.005))
```

In [ ]:
```
#Run training
history = model.fit(train_features_full, train_labels_full, epochs=200, validation_data=(train_features_valid, train_labels
```

In [ ]:
```
model.evaluate(test_features_full, test_labels_full)
```

In [ ]:
```
model.save("model_blocking_regr20_NN2.keras")
```

In [ ]:
```
# Plotting training and validation loss
history_df = pd.DataFrame(history.history)
history_df.plot(figsize=(8, 5))
plt.grid(True)
plt.gca().set_ylim(0, 0.02)
#plt.title('Training and Validation Loss')  # Set the title of the graph
plt.xlabel('Epochs')  # Set the x-axis label
plt.ylabel('Loss')  # Set the y-axis label
```

```
plt.legend(['Training Loss', 'Validation Loss'])  # Explicitly define legend labels
plt.show()
```

In [ ]:
```
unknown_data = pd.read_csv("blocking_dataset/unknown_data_regr_Gabr3.csv", header = None)
#unknown_data = pd.read_csv("blocking_dataset/unknown_data_regr_Gabr_repeat5.csv", header = None)
unknown_features = unknown_data.iloc[:,1:]
unknown_features_full = np.array(unknown_features)
unknown_features_full = scaler.transform(unknown_features_full)
```

In [ ]:
```
# Making predictions
X_new = unknown_features_full
y_pred = model.predict(X_new)
print(y_pred)
```

Code for NN model for 2 sizes of particles (case 3)

In [ ]:
```
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow import keras
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import joblib
```

In [ ]:
```
#Data assembly; need to do only once
#file1 = 'train_data_regr20_1m_1.csv'
#file2 = 'train_data_regr20_1m_2.csv'
#file3 = 'train_data_regr20_1m_3.csv'
#file4 = 'train_data_regr20_1m_4.csv'
#file5 = 'train_data_regr20_1m_5.csv'
#df1 = pd.read_csv(file1, header=None)
#df2 = pd.read_csv(file2, header=None)
#df3 = pd.read_csv(file3, header=None)
#df4 = pd.read_csv(file4, header=None)
#df5 = pd.read_csv(file5, header=None)

#combined_df = pd.concat([df1, df2, df3, df4, df5], axis=0, ignore_index=True)
#combined_df = combined_df.reset_index(drop=True)

#combined_df.to_csv('train_data_regr20_1m_all.csv', index=False, header=None)
```

In [ ]:
```
#Load data
train_data = pd.read_csv("blocking_dataset/train_data_regr20_1m_all.csv", header = None)
test_data = pd.read_csv("blocking_dataset/test_data_regr20_1m.csv", header = None)
#unknown_data = pd.read_csv("blocking_dataset/unknown_data_regr_Gabr_1m.csv", header = None)
# Shuffle data
train_data = train_data.sample(frac=1).reset_index(drop=True)
test_data = test_data.sample(frac=1).reset_index(drop=True)
```

In [ ]:
```
#train_data.head()
test_data.head()
```

In [ ]:
```
train_features = train_data.iloc[:,1:]
train_labels = train_data.iloc[:,0]
test_features = test_data.iloc[:,1:]
test_labels = test_data.iloc[:,0]
#unknown_features = unknown_data.iloc[:,1:]
```

In [ ]:
```
print(test_labels.head())
print(type(test_labels))
```

In [ ]:
```
train_features_full = np.array(train_features)
train_labels_full = np.array(train_labels)
test_features_full = np.array(test_features)
test_labels_full = np.array(test_labels)
#unknown_features_full = np.array(unknown_features)
```

In [ ]:
```
print(test_labels_full[0])
```

In [ ]:
```
# Load the scaler
#scaler = joblib.load("scaler_blocking_regr20.save")
#scaling data
scaler = StandardScaler()
train_features_full=scaler.fit_transform(train_features_full)
test_features_full = scaler.transform(test_features_full)
#unknown_features_full = scaler.transform(unknown_features_full)
```

In [ ]:
```
# Save the scaler
joblib.dump(scaler, "scaler_blocking_regr20_1m.save")
```

In [ ]:
```
print(test_features_full[0])
```

In [ ]:
```
#Defining validation data
train_features_valid, train_features = train_features_full[:7425], train_features_full[7425:]
train_labels_valid, train_labels = train_labels_full[:7425], train_labels_full[7425:]
```

In [ ]:
```
#Defyning NN model
model = keras.models.Sequential()
model.add(keras.layers.InputLayer(shape=(train_features_full.shape[1],)))
model.add(keras.layers.Dense(60, activation="relu"))
model.add(keras.layers.Dense(60, activation="relu"))
model.add(keras.layers.Dense(60, activation="relu"))
model.add(keras.layers.Dense(60, activation="relu"))
model.add(keras.layers.Dense(60, activation="relu"))
model.add(keras.layers.Dense(1))
model.summary()
```

In [ ]:
```
#Compile the model
model.compile(loss="mean_squared_error", optimizer=keras.optimizers.SGD(learning_rate = 0.005))
```

In [ ]:
```
#Run training
history = model.fit(train_features_full, train_labels_full, epochs=200, validation_data=(train_features_valid, train_labels
```

In [ ]:
```
#model = keras.models.load_model('my_keras_model_blocking_regr20_1m.h5')
model.evaluate(test_features_full, test_labels_full)
```

In [ ]:
```
model.save("model_blocking_regr20_NN2_1m.keras")
```

In [ ]:
```
# Plotting training and validation loss
history_df = pd.DataFrame(history.history)
history_df.plot(figsize=(8, 5))
plt.grid(True)
plt.gca().set_ylim(0, 0.01)
#plt.title('Training and Validation Loss')  # Set the title of the graph
plt.xlabel('Epochs')  # Set the x-axis label
plt.ylabel('Loss')  # Set the y-axis label
plt.legend(['Training Loss', 'Validation Loss'])  # Explicitly define legend labels
plt.show()
```

In [ ]:
```
unknown_data = pd.read_csv("blocking_dataset/unknown_data_regr_Gabr_1m.csv", header = None)
#unknown_data = pd.read_csv("blocking_dataset/unknown_data_regr_Gabr_repeat1.csv", header = None)
unknown_features = unknown_data.iloc[:,1:]
unknown_features_full = np.array(unknown_features)
unknown_features_full = scaler.transform(unknown_features_full)
```

In [ ]:
```
X_new = unknown_features_full
y_pred = model.predict(X_new)
print(y_pred)
```

Code for inference (making predictions)

In [ ]:
```
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow import keras
import joblib
from sklearn.preprocessing import StandardScaler
```

In [ ]:
```
# Load the saved model
model = keras.models.load_model("model_blocking_regr20_NN2_skew+4.keras")
# Load the scaler
scaler = joblib.load("scaler_blocking_regr20.save")
```

In [ ]:
```
#unknown_data = pd.read_csv("blocking_dataset/unknown_data_regr_Gabr2.csv", header = None)
unknown_data = pd.read_csv("blocking_dataset/unknown_data_regr_Gabr_repeat6.csv", header = None)
unknown_features = unknown_data.iloc[:,1:]
unknown_features_full = np.array(unknown_features)
unknown_features_full = scaler.transform(unknown_features_full)
```

In [ ]:
```
# Making predictions
X_new = unknown_features_full
y_pred = model.predict(X_new)
print(y_pred)
```