Millikan_Droplet

May 15, 2019

1 Millikan Oil Drop

___Meansurement of the electron charge

University of California, Santa Barbara, 93117 Goleta
Yuning Zhang---May.8th 2019

In [244]:
```
import numpy as np
import pandas as pd
import matplotlib.pylab as plt
import os
```

In [245]:
```
rho=886 # kg/m^3
g=9.8 # m/s^2
b=8.20*10**(-3) # Pa*m
p=101204 #Pa
e=1.6*10**(-19)
V_std=1 #V

def get_eta(vid):
    # determine viscosity for each measurement
    vid=int(vid)
    eta= 1.8330*10**(-5) # N*s/m^2
    if 1<=vid<=5:
        return 1.8310*10**(-5)
    elif 6<=vid<=13:
        return 1.8340*10**(-5)
    else:
        return 1.8280*10**(-5)


def get_V(vid):
     # determine voltage for each measurement
    V=505 #V
    vid=int(vid)
    if 1<=vid<=5:
        return 504
    elif 6<=vid<=13:
```

```
            return 501
        else:
            return 505
```

```
In [246]: d_array=10**(-3)*np.array([7.55,7.59,7.60,7.60,7.60,7.61]) # unit: m
          d_mean=d_array.mean()
          d_std=d_array.std()
          print("d_mean: ",d_mean)
          print("d_std: ",d_std)
```

```
d_mean:  0.00759166666667
d_std:  1.95078331845e-05
```

```
In [247]: def reject_outliers(data, m=3):
              '''
              remove anomalous data points that outside 2 standard deviation in the array
              '''
          #     data=data[abs(data - np.mean(data)) < m * np.std(data)]
              return data[abs(data - np.mean(data)) < m * np.std(data)]
```

Load data from files

```
In [248]: data_path = "./data/"
          statistics=[]
          for file_name in os.listdir(data_path):
              name=file_name[:3]
              obj_drop=pd.read_csv(data_path+file_name).dropna()
              # seperate rising and falling velocities, remove anomalous velocities at switching
              v_y=obj_drop["v_{y}"].values
              n_points=len(v_y)
              v_r=reject_outliers(v_y[v_y>0])
              v_f=reject_outliers(v_y[v_y<0])
              # calculate mean and deviation
              (v_r_mean,v_r_std)=(v_r.mean(),v_r.std())
              (v_f_mean,v_f_std)=(np.abs(v_f.mean()),v_f.std())
              # calculate other properties
              vid=file_name[:2]
              eta=get_eta(vid)
              V=get_V(vid)
              a=np.sqrt((b/2/p)**2+9*eta*v_f_mean/2/rho/g)-b/(2*p) #droplet radius
              m=4*np.pi/3*a**3*rho # droplet mass
              q=m*g*d_mean*(v_f_mean+v_r_mean)/V/v_f_mean #droplet charge
              # long formula for error propagation
              q_std=q*np.sqrt((((b**2*g*rho+18*p**2*eta*v_f_mean+3*b*g*rho*p*np.sqrt(b**2/p**2+1
              # stack to dataframe
              statistics.append(np.array((name,n_points,v_r_mean,v_r_std,v_f_mean,v_f_std,a,m,q,
```

Calculation of the attached charge

In [249]:
```
labels=["name","n_points","v_r_mean","v_r_std","v_f_mean","v_f_std","a","m","q","q_std
overall=pd.DataFrame(statistics,columns=labels,dtype="float32")
```

In [250]:
```
overall#he droplet charge calculated
```

Out[250]:

	name	n_points	v_r_mean	v_r_std	v_f_mean	v_f_std	a	\
0	01A	960.0	0.000097	0.000011	0.000066	0.000014	7.541285e-07	
1	01B	1140.0	0.000054	0.000011	0.000079	0.000042	8.265568e-07	
2	01C	835.0	0.000137	0.000017	0.000067	0.000013	7.569128e-07	
3	01D	571.0	0.000044	0.000011	0.000049	0.000012	6.452343e-07	
4	02A	1249.0	0.000083	0.000024	0.000037	0.000013	5.572819e-07	
5	02B	704.0	0.000020	0.000011	0.000041	0.000019	5.825815e-07	
6	04A	350.0	0.000062	0.000011	0.000064	0.000015	7.403968e-07	
7	04B	154.0	0.000142	0.000083	0.000028	0.000037	4.748169e-07	
8	04C	109.0	0.000039	0.000011	0.000051	0.000021	6.554863e-07	
9	04D	48.0	0.000036	0.000012	0.000051	0.000009	6.539939e-07	
10	04E	432.0	0.000018	0.000010	0.000060	0.000027	7.163554e-07	
11	05A	112.0	0.000234	0.000080	0.000054	0.000035	6.732974e-07	
12	06A	124.0	0.000059	0.000015	0.000020	0.000008	3.993551e-07	
13	06B	356.0	0.000041	0.000038	0.000023	0.000013	4.255928e-07	
14	06D	396.0	0.000132	0.000099	0.000033	0.000014	5.232803e-07	
15	06E	206.0	0.000100	0.000063	0.000025	0.000014	4.472741e-07	
16	06F	121.0	0.000075	0.000026	0.000054	0.000006	6.764853e-07	
17	06H	385.0	0.000049	0.000031	0.000021	0.000013	4.050480e-07	
18	07A	336.0	0.000014	0.000008	0.000023	0.000024	4.302464e-07	
19	08A	872.0	0.000033	0.000038	0.000014	0.000010	3.236547e-07	
20	08B	124.0	0.000106	0.000125	0.000023	0.000036	4.323034e-07	
21	09A	409.0	0.000052	0.000022	0.000024	0.000014	4.384432e-07	
22	09B	364.0	0.000265	0.000189	0.000027	0.000018	4.649105e-07	
23	10A	243.0	0.000061	0.000059	0.000016	0.000011	3.526907e-07	
24	11A	354.0	0.000074	0.000132	0.000018	0.000012	3.737323e-07	
25	11B	708.0	0.000023	0.000038	0.000015	0.000013	3.367016e-07	
26	13A	150.0	0.000096	0.000025	0.000043	0.000013	6.023454e-07	
27	13B	285.0	0.000175	0.000073	0.000022	0.000013	4.190767e-07	
28	13C	154.0	0.000127	0.000021	0.000033	0.000013	5.208464e-07	
29	13D	478.0	0.000118	0.000015	0.000041	0.000013	5.847114e-07	
30	13E	1089.0	0.000078	0.000021	0.000039	0.000019	5.660419e-07	
31	13F	181.0	0.000172	0.000019	0.000041	0.000011	5.882166e-07	
32	13G	161.0	0.000064	0.000014	0.000042	0.000013	5.909288e-07	
33	13H	155.0	0.000087	0.000029	0.000018	0.000012	3.789482e-07	
34	13I	157.0	0.000042	0.000020	0.000024	0.000015	4.390456e-07	
35	13J	100.0	0.000076	0.000008	0.000038	0.000007	5.609695e-07	
36	13K	92.0	0.000098	0.000035	0.000017	0.000009	3.612755e-07	
37	13L	110.0	0.000043	0.000008	0.000026	0.000008	4.614205e-07	
38	14A	624.0	0.000041	0.000012	0.000096	0.000023	9.119827e-07	
39	14B	482.0	0.000040	0.000015	0.000096	0.000017	9.128012e-07	
40	14C	286.0	0.000051	0.000026	0.000020	0.000012	3.920644e-07	
41	16A	587.0	0.000100	0.000013	0.000051	0.000014	6.569278e-07	

42	17A	821.0	0.000111	0.000023	0.000079	0.000028	8.230424e-07
43	17B	308.0	0.000114	0.000012	0.000077	0.000016	8.158515e-07
44	17C	389.0	0.000036	0.000012	0.000078	0.000012	8.191622e-07
45	18A	603.0	0.000097	0.000017	0.000033	0.000012	5.205169e-07

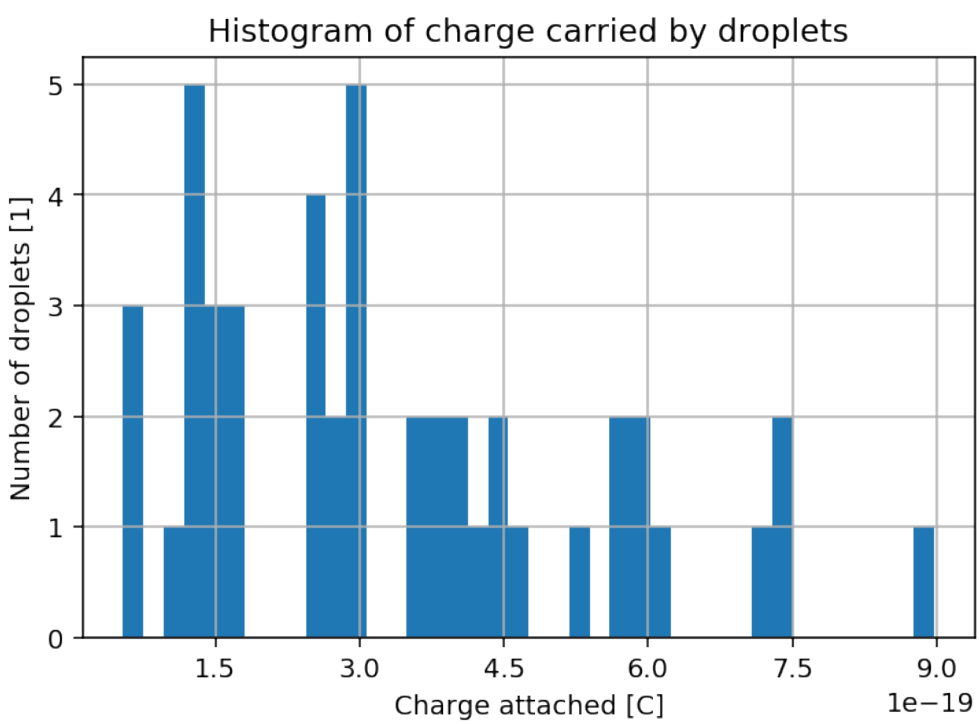
	m	q	q_std
0	1.591690e-15	5.771566e-19	1.261804e-19
1	2.095755e-15	5.222924e-19	3.264994e-19
2	1.609384e-15	7.241972e-19	1.445641e-19
3	9.969526e-16	2.782697e-19	8.106526e-20
4	6.423136e-16	3.039631e-19	1.163181e-19
5	7.338251e-16	1.607750e-19	9.995117e-20
6	1.506316e-15	4.361534e-19	1.158103e-19
7	3.972838e-16	3.573406e-19	4.076589e-19
8	1.045233e-15	2.717123e-19	1.341717e-19
9	1.038110e-15	2.628353e-19	6.389324e-20
10	1.364294e-15	2.599770e-19	1.618532e-19
11	1.132774e-15	8.973500e-19	5.176288e-19
12	2.363742e-16	1.369184e-19	5.247263e-20
13	2.860917e-16	1.200775e-19	9.921591e-20
14	5.317720e-16	3.920115e-19	2.710639e-19
15	3.320806e-16	2.476113e-19	1.724092e-19
16	1.148940e-15	4.075351e-19	9.300076e-20
17	2.466276e-16	1.238745e-19	8.883027e-20
18	2.955794e-16	6.979906e-20	9.213031e-20
19	1.258253e-16	6.339593e-20	6.636466e-20
20	2.998390e-16	2.464733e-19	3.887668e-19
21	3.127969e-16	1.477152e-19	8.911137e-20
22	3.729327e-16	6.043287e-19	4.903621e-19
23	1.628188e-16	1.160608e-19	1.134375e-19
24	1.937332e-16	1.484244e-19	2.289573e-19
25	1.416635e-16	5.406359e-20	7.306212e-20
26	8.110714e-16	3.868082e-19	1.296810e-19
27	2.731510e-16	3.630988e-19	2.040628e-19
28	5.243863e-16	3.784747e-19	1.335429e-19
29	7.419029e-16	4.276518e-19	1.221074e-19
30	6.730821e-16	3.010972e-19	1.458919e-19
31	7.553256e-16	5.767505e-19	1.348694e-19
32	7.658219e-16	2.890824e-19	9.779566e-20
33	2.019583e-16	1.716367e-19	1.039117e-19
34	3.140878e-16	1.278192e-19	8.687053e-20
35	6.551489e-16	2.921510e-19	5.208002e-20
36	1.749999e-16	1.777900e-19	9.502843e-20
37	3.645968e-16	1.418015e-19	4.567891e-20
38	2.815024e-15	5.939706e-19	1.914406e-19
39	2.822610e-15	5.904479e-19	1.499494e-19
40	2.236632e-16	1.193663e-19	7.944851e-20
41	1.052144e-15	4.582708e-19	1.200143e-19

```
42  2.069136e-15  7.365568e-19  2.759127e-19
43  2.015374e-15  7.355359e-19  1.582469e-19
44  2.040009e-15  4.409025e-19  9.907497e-20
45  5.233916e-16  3.026955e-19  1.057990e-19
```

the charge per oil droplet calculated from data is

In [290]:
```
plt.figure().dpi=140
plt.xlabel("Charge attached [C]")
plt.ylabel("Number of droplets [1]")
plt.xticks(np.arange(0,9.5E-19,1.5E-19))
plt.title("Histogram of charge carried by droplets")
(overall.q).hist(bins=40)
```

Out[290]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x11b5189c4a8>
```



In [252]:
```
def clustering(arr,x):
    '''
    Assume the charge is quantized, and the minimum quantum of charge is x.
    That means the charge can only be integral multiple of x, (x,2x,3x...)
    Thus we can divide the droplets into different clusters based on their attached ch

    group1=[droplets with charge between 0 and 1x]
    group2=[droplets with charge between 1x and 2x]
```

```
        return: list of cluters(numpy array)
        '''
        arr=list(arr/x)
        num=int(max(arr))
        clusters=[]
        for i in range(num+1):
            clusters.append(x*(np.array(list(filter(lambda x:i<x<=i+1 ,arr)))))
        return clusters
```

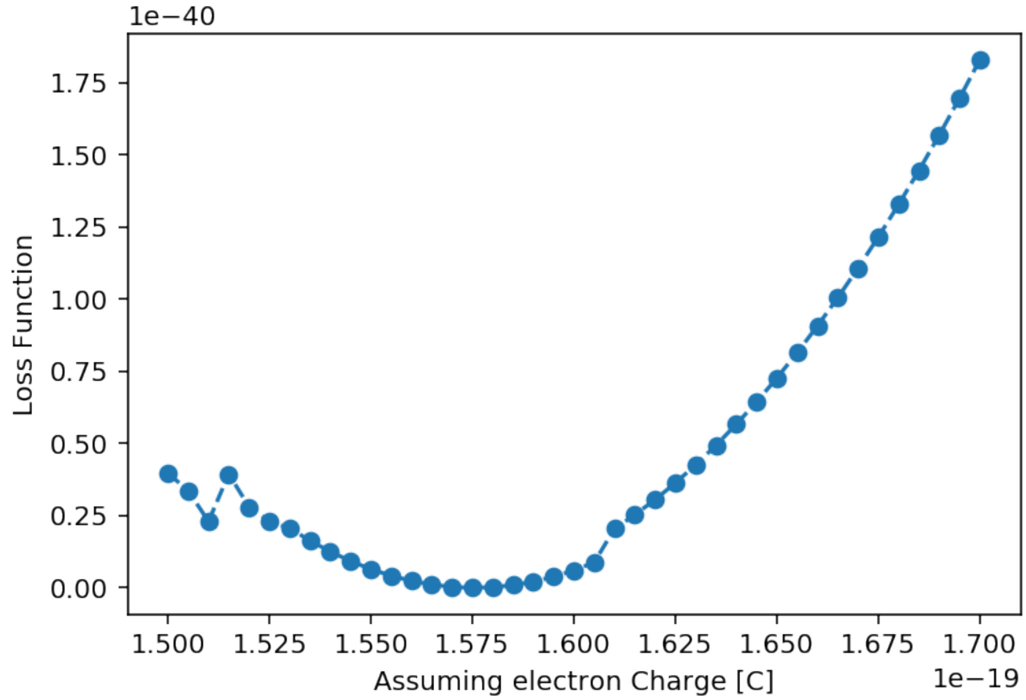
In [275]:
```
def obj_error(x):
        '''
        objective function used for optimization, compare the average difference between e
        group with charge e and penalize deviation from e
        a proper value of assumed basic charge e (x here) will minimize the obj_error func

        optimize this function and we can get value of e

        '''
        clusters=clustering(overall.q,x)
        test=np.nan_to_num(np.array(list(map(np.mean,clusters))))
        estimate_delta_q=[]
        n_len=len(test)
        for i in range(1,n_len):
            for j in range(i):
                estimate_delta_q.append(abs(test[i]-test[j])/(i-j))
        return  (np.array(estimate_delta_q).mean()-x)**2
#sum((np.array(estimate_delta_q)-e)**2)
#
```

In [289]:
```
steps=0.005E-19
min_bound=1.5E-19
max_bound=1.7E-19
ran=np.arange(min_bound,max_bound,steps)
est=list(map(obj_error,ran))
plt.figure().dpi=140
plt.xlabel("Assuming electron Charge [C]")
plt.ylabel("Loss Function")
plt.plot(ran,est,"--o")
```

Out[289]:
```
[<matplotlib.lines.Line2D at 0x11b507eb2b0>]
```



In [288]:
```
steps=0.001E-19
ran=np.arange(min_bound,max_bound,steps)
est=list(map(obj_error,ran))
# search for the minimized error and give
e_estimate,min_loss=min(list(zip(ran,est)),key=lambda x: x[1])
```

In [285]:
```
print("Estimate e value:",e_estimate,"C")
print("Error",(e-e_estimate)/e*100,"%")
```

```
Estimate e value: 1.575e-19 C
Error 1.5625 %
```