# Analisi Dati dell'Esperimento di Millikan

December 21, 2024

## 1 Analisi dati per l'esperimento di Millikan

Questo file è una rivisitazione corretta del file `analisi.ipynb`, che contiene tutta l'analisi dati sui dati presi in laboratorio.

Questo programma è in grado di calcolare la carica anche se i dati non sono omogenei tra loro, questo perché potrebbe capitare che una goccia acquista o perde carica durante la misura, e quindi c'è la necessità di avere un programma che sia stabile anche se manca una misura in mezzo ai dati.

NOTA: mancano le considerazioni su errori sistematici e la loro entità.

### 1.1 Import

#### 1.1.1 Import delle librerie

[ ]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from colorama import Fore, Style
from scipy import stats
from functools import reduce

import scienceplots

plt.style.use(["science", "grid", "ieee"])
```

#### 1.1.2 Import dei dati

Dati preliminari:

[ ]:
```
g = 9.806  # m/s^2
rho_o = 860  # kg/m^3
rho_a = 1.293  # kg/m^3
pressione = 101325 * 1.01  # Pa
b = 8.2e-3  # Pa m (costante correzione viscosa)
reticolo = 0.5 / 1000  # m

d_isolante = 7.1656 / 1000  # m
```

Dati delle misure:

[ ]:
```
# settings per i dati
n_gocce = 8
n_cols = 8
n_rows = 6

# load data from excel file
data = pd.read_excel("Dati_Grezzi.xlsx", sheet_name="AllData", header=None)

# drop zero rows and columns
data = data.dropna(axis=1, how="all").dropna(axis=0, how="all")

# initialize data containers
temperature = np.full(n_gocce, np.nan)
voltaggi = np.full((n_gocce, n_cols), np.nan)
tempi = np.full((n_gocce, n_cols, n_rows), np.nan)

# temporary variables for the loop
temp = np.full((n_rows, n_cols), np.nan)  # temporary container of tempi
g_idx = -1  # index of goccia
r_idx = 0  # index of row

# loop to populate `temperature`, `voltaggi` and `tempi`
for row in data.to_numpy():
    # tempi is a row with a number and then all nan values
    if np.isnan(row[1:]).all():
        g_idx += 1
        temperature[g_idx] = row[0]

        if np.isnan(temp).all():
            continue

        tempi[g_idx - 1] = temp.T

        temp = np.full((n_rows, n_cols), np.nan)
        r_idx = 0

    # voltaggi is a row starting with zero
    elif row[0] == 0:
        voltaggi[g_idx] = row

    # tempi is everything remaining
    else:
        temp[r_idx] = row
        r_idx += 1
else:
    tempi[g_idx] = temp.T
```

## 1.2 Analysis

Sezione di analisi dei dati raccolti, qui calcoliamo tutte le cose necessarie a trovare il valore di $q_{min}$.

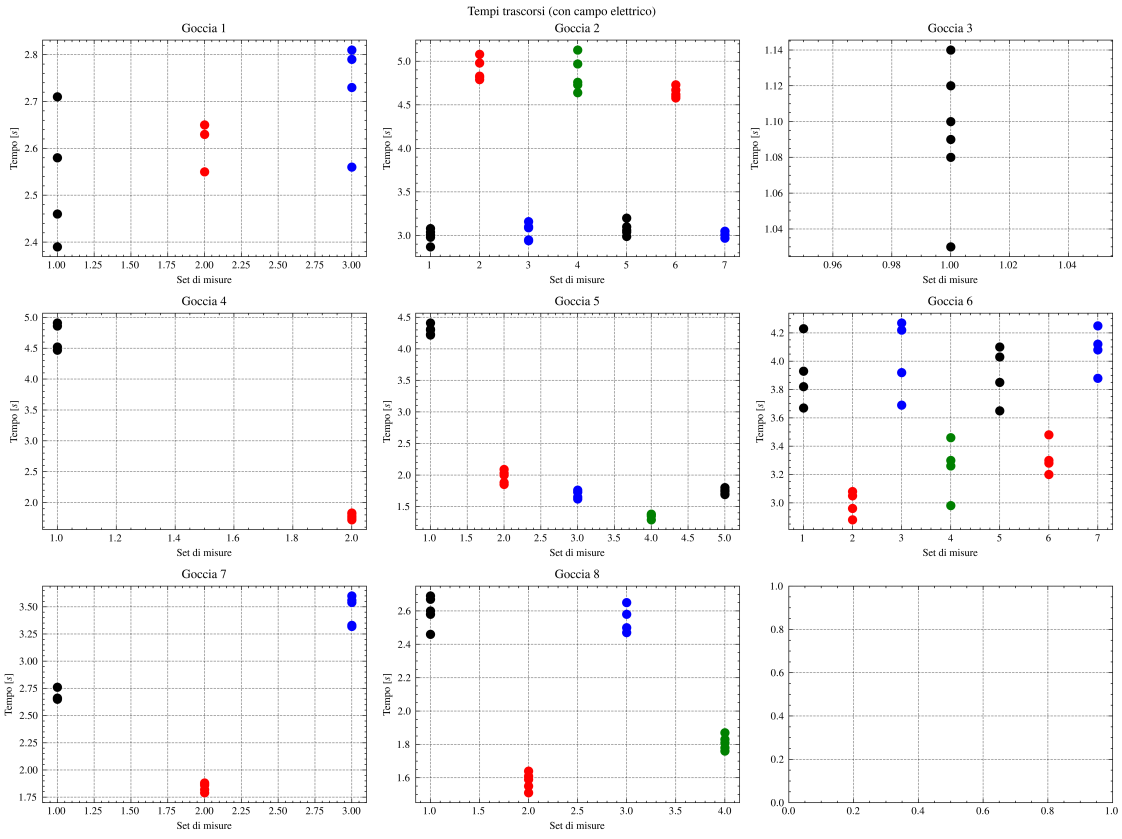
### 1.2.1 Show collected data

[ ]:
```
# Array way
print("Temperature [°C]:\n", temperature)
print("Voltaggi [V/m]:\n", voltaggi)
print("Tempi [s]:\n", tempi)

# plot way
plot_row = np.ceil(np.sqrt(n_gocce)).astype(int)
plot_col = np.ceil(n_gocce / plot_row).astype(int)
fig, axs = plt.subplots(plot_col, plot_row, figsize=(plot_row * 4, plot_col *␣
 ↪3))

plt.suptitle("Tempi trascorsi (con campo elettrico)")
for i in range(n_gocce):
    ax = axs[i // plot_row, i % plot_row]
    ax.set_title(f"Goccia {i + 1}")
    ax.set_xlabel("Set di misure")
    ax.set_ylabel(r"Tempo [$s$]")

    for j in range(1, n_cols):
        ax.scatter(j * np.ones(len(tempi[i, j, :n_rows])), tempi[i, j, :n_rows])

plt.tight_layout()
# plt.savefig("Images/Tempi trascorsi.svg") # todo una volta scaricata␣
 ↪l'immagone in svg rimuovere "goccia 9"
plt.show()
```

```
Temperature [°C]:
 [21. 21. 22. 22. 22. 23. 23. 23.]
Voltaggi [V/m]:
 [[   0.  282.  282. -282.   nan   nan   nan   nan]
 [   0.  322. -323.  323. -323.  323. -323.  323.]
 [   0.  354.   nan   nan   nan   nan   nan   nan]
 [   0. -354.  354.   nan   nan   nan   nan   nan]
 [   0. -354.  354. -354.  355. -355.   nan   nan]
 [   0. -375.  375. -375.  375. -375.  375. -375.]
 [   0. -373.  373. -373.   nan   nan   nan   nan]
 [   0. -373.  373. -373.  373.   nan   nan   nan]]
Tempi [s]:
 [[[60.25 67.9  49.92 56.53 50.1    nan]
  [ 2.71  2.39  2.58  2.58  2.46   nan]
  [ 2.65  2.55  2.65  2.63   nan   nan]
  [ 2.73  2.56  2.81  2.79   nan   nan]
```

```
 [  nan   nan   nan   nan   nan   nan]
 [  nan   nan   nan   nan   nan   nan]
 [  nan   nan   nan   nan   nan   nan]
 [  nan   nan   nan   nan   nan   nan]]

[[14.82 16.88 14.05 15.24 16.16   nan]
 [ 2.98  2.87  3.04  3.08  3.01   nan]
 [ 4.82  4.83  4.98  4.79  5.08   nan]
 [ 3.1   2.94  3.09  2.95  3.16   nan]
 [ 5.13  4.64  4.73  4.76  4.97   nan]
 [ 3.06  3.04  2.99  3.1   3.2    nan]
 [ 4.6   4.73  4.62  4.67  4.58   nan]
 [ 3.01  2.97  3.01  3.05  3.05   nan]]

[[25.13 21.35 22.15 20.19 19.56 19.78]
 [ 1.12  1.03  1.09  1.08  1.1   1.14]
 [  nan   nan   nan   nan   nan   nan]
 [  nan   nan   nan   nan   nan   nan]
 [  nan   nan   nan   nan   nan   nan]
 [  nan   nan   nan   nan   nan   nan]
 [  nan   nan   nan   nan   nan   nan]
 [  nan   nan   nan   nan   nan   nan]]

[[ 9.28 10.   10.57  9.7  10.95   nan]
 [ 4.86  4.91  4.52  4.47   nan   nan]
 [  nan  1.75  1.72  1.78  1.83  1.82]
 [  nan   nan   nan   nan   nan   nan]
 [  nan   nan   nan   nan   nan   nan]
 [  nan   nan   nan   nan   nan   nan]
 [  nan   nan   nan   nan   nan   nan]
 [  nan   nan   nan   nan   nan   nan]]

[[10.57 11.93 11.13 11.97 12.45   nan]
 [ 4.31  4.23  4.22  4.41  4.3    nan]
 [ 1.85  2.03  2.    2.09  1.88   nan]
 [ 1.62  1.76  1.65  1.73  1.73   nan]
 [ 1.37  1.38  1.29  1.37  1.35   nan]
 [ 1.7   1.69  1.76  1.74  1.8    nan]
 [  nan   nan   nan   nan   nan   nan]
 [  nan   nan   nan   nan   nan   nan]]

[[25.7  26.21 27.4  28.     nan   nan]
 [ 3.93  3.82  4.23  3.67   nan   nan]
 [ 2.96  3.05  3.08  2.88   nan   nan]
 [ 4.27  3.92  3.69  4.22   nan   nan]
 [ 3.26  2.98  3.3   3.46   nan   nan]
 [ 4.1   3.65  4.03  3.85   nan   nan]
 [ 3.28  3.2   3.3   3.48   nan   nan]
```

```
 [ 4.12  4.08  3.88  4.25   nan   nan]]

[[ 7.39  7.64  8.89  7.8   8.05   nan]
 [ 2.65  2.66  2.76  2.66   nan   nan]
 [ 1.86  1.79  1.82  1.87  1.88   nan]
 [ 3.54  3.56  3.33  3.32  3.6    nan]
 [  nan   nan   nan   nan   nan   nan]
 [  nan   nan   nan   nan   nan   nan]
 [  nan   nan   nan   nan   nan   nan]
 [  nan   nan   nan   nan   nan   nan]]

[[ 8.97  8.08  7.95  7.1   8.37   nan]
 [ 2.69  2.58  2.46  2.6   2.67   nan]
 [ 1.55  1.51  1.64  1.59  1.61   nan]
 [ 2.65  2.5   2.47  2.58   nan   nan]
 [ 1.87  1.83  1.76  1.81  1.78   nan]
 [  nan   nan   nan   nan   nan   nan]
 [  nan   nan   nan   nan   nan   nan]
 [  nan   nan   nan   nan   nan   nan]]]
```



### 1.2.2 Funzioni utili

`calc_visco(...)` calcola il coefficiente di viscosità $\eta$ in funzione della temperatura usando la formula:

$$\eta = 1.800 \cdot 10^{-5} + (T - 15) \cdot 4.765 \cdot 10^{-8}$$

dove $T$ è la temperatura in gradi Celsius.

[ ]:
```
def calc_visco(t: float) -> float:
    # T deve essere in celsius
    return (1.8 + 4.765e-3 * (t - 15)) * 1e-5  # N s m^-2
```

`calc_radius(...)`, come dice il nome, calcola il raggio considerando la correzione del moto Browninano, ovvero calcolando il coefficiente di viscosità efficace $\eta_{\text{eff}}$.

$$\eta_{\text{eff}} = \frac{\eta}{1 + \frac{b}{pr}}$$

Dove $b$ è la costante di correzione viscosa, $p$ la pressione atmosferica che consideriamo costante durante tutto l'esperimento e pari a $1.01atm$, e $r$ è il raggio della goccia.

Inserendo questa correzione nella formula di caduta libera: $\frac{4\pi}{3} r^3 (\rho_{\text{olio}} - \rho_{\text{aria}}) g - 6\pi\eta r v_r = 0$ otteniamo che il raggio è pari a:

$$r = \sqrt{\left(\frac{b}{2p}\right) + \frac{9\eta + v_r}{2g(\rho_{\text{olio}} - \rho_{\text{aria}})}} - \frac{b}{2p}$$

[ ]:
```
def calc_radius(eta: float, vel: np.ndarray) -> np.ndarray:
    return np.sqrt(
        (b / (2 * pressione)) ** 2 + (9 * eta * vel) / (2 * g * (rho_o - rho_a))
    ) - b / (
        2 * pressione
    )  # m
```

`calc_charge(...)`, come dice il nome, calcola la carica utilizzando la formula:

$$q = -\frac{4\pi}{3} r^3 (\rho_{\text{olio}} - \rho_{\text{aria}}) \frac{g}{E} (1 - \frac{v}{v_{\text{r}}})$$

dove $v_{\text{r}}$ è la velocità in caduta libera.

[ ]:
```
def calc_charge(
    r: float, v_r: float, e_field: np.ndarray, v_e: np.ndarray
) -> np.ndarray:
    # calcola la carica di tutti i set di una goccia
    e_field = e_field[:, np.newaxis]  # rende compatibile i calcoli tra matrici
    return (
        -4
        / 3
```

```
        * np.pi
        * r**3
        * (rho_o - rho_a)
        * g
        * (1 - np.sign(e_field) * v_e / v_r)
        / e_field
    )  # C
```

[ ]:
```
def calc_stats(arr: np.ndarray):
    # Calcola la media e stddev ignorando i nan
    # 1D array
    if arr.ndim == 1:
        return np.nanmean(arr), np.nanstd(arr, ddof=1)

    # 2D array
    return np.nanmean(arr, axis=1), np.nanstd(arr, axis=1, ddof=1)
```

### 1.2.3 Analisi vera e propria

Dai tempi calcoliamo (per ogni sezione del reticolo): - Le velocità di caduta libera: `vel_r`, di cui calcoleremo il valor medio delle sezioni per trovare il miglior valore: `mean_v_r` per ogni goccia. - Le velocità con campo elettrico: `vel_field`.

Dalle temperature calcoliamo: il coefficiente di viscosità $\eta$: `eta`.

Dalle velocità di caduta libera e dal coefficiente di viscosità calcoliamo il raggio `r` (applicando la correzione di $\eta_{\text{efficace}}$), dai raggi di ogni sezione facciamo una media per trovare il miglior valore del raggio della goccia.

Dai voltaggi e dallo spessore dell'isolante troviamo i valori del campo elettrico: `E`.

Questi valori ci permettono di calcolare le cariche $Q_i$: `charges[i] <- q`, usando la formula descritta precedentemente.

[ ]:
```
# initialize data container
charges = np.full((n_gocce, (n_cols - 1) * n_rows), np.nan)

# loop trough gocce
for i in range(0, n_gocce):
    vel_r = reticolo / tempi[i, 0]  # velocità senza campo
    mean_v_r, std_v_r = np.nanmean(vel_r), np.nanstd(vel_r, ddof=1)

    vel_field = reticolo / tempi[i, 1:]  # velocità con il campo

    # coefficiente di attrito
    eta = calc_visco(temperature[i])
    # Raggi
    r = calc_radius(eta, vel_r)
    # Raggio medio e StdDev (la std non serve effettivamente)
```

```
    mean_r, std_r = np.nanmean(r), np.nanstd(r, ddof=1)

    print(Style.BRIGHT + f"~~~~~ Goccia {i + 1} ~~~~~" + Style.RESET_ALL)
    print(f"Raggio: {mean_r:.4g} ±{std_r:.2g}m")

    # campo elettrico
    E = voltaggi[i, 1:n_cols] / d_isolante
    print(f"Campi elettrici [V/m] per set (2-{n_cols})\n\t", E)

    # Carica
    q = calc_charge(mean_r, mean_v_r, E, vel_field).flatten()
    print(f"Cariche [C] per set (2-{n_cols}): \n\t", q)

    # save data to container
    charges[i] = q
```

```
~~~~~ Goccia 1 ~~~~~
Raggio: 2.571e-07 ±1.9e-08m
Campi elettrici [V/m] per set (2-8)
         [ 39354.69465223  39354.69465223 -39354.69465223             nan
             nan             nan             nan]
Cariche [C] per set (2-8):
         [3.00473905e-19 3.42743899e-19 3.16381472e-19 3.16381472e-19
 3.32557646e-19            nan 3.07621920e-19 3.20282784e-19
 3.07621920e-19 3.10077069e-19            nan            nan
 3.28621279e-19 3.49432411e-19 3.19699110e-19 3.21881684e-19
            nan            nan            nan            nan
            nan            nan            nan            nan
            nan            nan            nan            nan
            nan            nan            nan            nan
            nan            nan            nan            nan
            nan            nan            nan            nan
            nan            nan]
~~~~~ Goccia 2 ~~~~~
Raggio: 5.25e-07 ±2e-08m
Campi elettrici [V/m] per set (2-8)
         [ 44936.92084403 -45076.47649883  45076.47649883 -45076.47649883
  45076.47649883 -45076.47649883  45076.47649883]
Cariche [C] per set (2-8):
         [4.71987791e-19 4.94430240e-19 4.60431003e-19 4.52976625e-19
 4.66151805e-19            nan 4.74100528e-19 4.73353331e-19
 4.62505476e-19 4.76360837e-19 4.55629474e-19            nan
 4.47930487e-19 4.78468448e-19 4.49746458e-19 4.76462777e-19
 4.37276024e-19            nan 4.52291999e-19 4.88100803e-19
 4.80967470e-19 4.78649637e-19 4.63208295e-19            nan
 4.55265585e-19 4.59005520e-19 4.68574251e-19 4.47930487e-19
 4.30395017e-19            nan 4.91360770e-19 4.80967470e-19
 4.89723730e-19 4.85692476e-19 4.93012107e-19            nan
```

```
 4.64708611e-19 4.72491953e-19 4.64708611e-19 4.57129422e-19
 4.57129422e-19            nan]
~~~~~ Goccia 3 ~~~~~
Raggio: 4.419e-07 ±2.2e-08m
Campi elettrici [V/m] per set (2-8)
         [49402.70179748            nan             nan             nan
            nan             nan             nan]
Cariche [C] per set (2-8):
         [1.10438813e-18 1.20626992e-18 1.13647933e-18 1.14757259e-18
 1.12558777e-18 1.08393233e-18             nan             nan
             nan             nan             nan             nan
             nan             nan             nan             nan
             nan             nan             nan             nan
             nan             nan             nan             nan
             nan             nan             nan             nan
             nan             nan             nan             nan
             nan             nan             nan             nan
             nan             nan             nan             nan
             nan             nan]
~~~~~ Goccia 4 ~~~~~
Raggio: 6.584e-07 ±2.3e-08m
Campi elettrici [V/m] per set (2-8)
         [-49402.70179748  49402.70179748             nan             nan
             nan             nan             nan]
Cariche [C] per set (2-8):
         [6.25897141e-19 6.21598933e-19 6.57646830e-19 6.62723271e-19
            nan             nan             nan 9.68374802e-19
 9.88819939e-19 9.48618827e-19 9.17131618e-19 9.23290654e-19
             nan             nan             nan             nan
             nan             nan             nan             nan
             nan             nan             nan             nan
             nan             nan             nan             nan
             nan             nan             nan             nan
             nan             nan             nan             nan
             nan             nan             nan             nan
             nan             nan             nan             nan
             nan             nan]
~~~~~ Goccia 5 ~~~~~
Raggio: 6.116e-07 ±2.1e-08m
Campi elettrici [V/m] per set (2-8)
         [-49402.70179748  49402.70179748 -49402.70179748  49542.25745227
 -49542.25745227             nan             nan]
Cariche [C] per set (2-8):
         [6.01744044e-19 6.10035818e-19 6.11094395e-19 5.91802375e-19
 6.02763643e-19            nan 8.58101264e-19 7.67532203e-19
 7.81494933e-19 7.40809274e-19 8.41802044e-19             nan
 1.32975031e-18 1.23696580e-18 1.30854243e-18 1.25558400e-18
 1.25558400e-18            nan 1.21254498e-18 1.20257831e-18
 1.29784120e-18 1.21254498e-18 1.23292130e-18             nan
```

```
 1.27126816e-18 1.27782681e-18 1.23348139e-18 1.24578743e-18
 1.20968973e-18            nan            nan            nan
            nan            nan            nan            nan
            nan            nan            nan            nan
            nan            nan]
~~~~~ Goccia 6 ~~~~~
Raggio: 3.899e-07 ±8.4e-09m
Campi elettrici [V/m] per set (2-8)
         [-52333.37054817  52333.37054817 -52333.37054817  52333.37054817
 -52333.37054817  52333.37054817 -52333.37054817]
Cariche [C] per set (2-8):
         [3.12271608e-19 3.20113599e-19 2.92957356e-19 3.31564809e-19
            nan            nan 3.21633876e-19 3.10964463e-19
 3.07546556e-19 3.31677613e-19            nan            nan
 2.90587177e-19 3.12966330e-19 3.29984190e-19 2.93556922e-19
            nan            nan 2.88360146e-19 3.19207201e-19
 2.84380742e-19 2.69383220e-19            nan            nan
 3.00979837e-19 3.33162749e-19 3.05514016e-19 3.17930433e-19
            nan            nan 2.86358311e-19 2.94515786e-19
 2.84380742e-19 2.67605497e-19            nan            nan
 2.99712656e-19 3.02259441e-19 3.15781027e-19 2.91766690e-19
            nan            nan]
~~~~~ Goccia 7 ~~~~~
Raggio: 7.48e-07 ±2.7e-08m
Campi elettrici [V/m] per set (2-8)
         [-52054.25923858  52054.25923858 -52054.25923858            nan
             nan            nan            nan]
Cariche [C] per set (2-8):
         [1.13113558e-18 1.12794906e-18 1.09735389e-18 1.12794906e-18
            nan            nan 9.24098516e-19 9.71323915e-19
 9.50639635e-19 9.17640650e-19 9.11251485e-19            nan
 9.18035090e-19 9.14470412e-19 9.58049404e-19 9.60081111e-19
 9.07459879e-19            nan            nan            nan
            nan            nan            nan            nan
            nan            nan            nan            nan
            nan            nan            nan            nan
            nan            nan            nan            nan
            nan            nan            nan            nan
            nan            nan]
~~~~~ Goccia 8 ~~~~~
Raggio: 7.416e-07 ±3.4e-08m
Campi elettrici [V/m] per set (2-8)
         [-52054.25923858  52054.25923858 -52054.25923858  52054.25923858
             nan            nan            nan]
Cariche [C] per set (2-8):
         [1.10328024e-18 1.13853435e-18 1.18058913e-18 1.13190263e-18
 1.10947402e-18            nan 1.15860652e-18 1.19662024e-18
 1.07985555e-18 1.12250545e-18 1.10512760e-18            nan
```

```
1.11576128e-18 1.16612228e-18 1.17692849e-18 1.13853435e-18
           nan            nan 9.13041983e-19 9.39040961e-19
9.87382811e-19 9.52471372e-19 9.73182878e-19            nan
           nan            nan            nan            nan
           nan            nan            nan            nan
           nan            nan            nan            nan
           nan            nan            nan            nan
           nan            nan]
```

### 1.2.4 Plot the results for each goccia

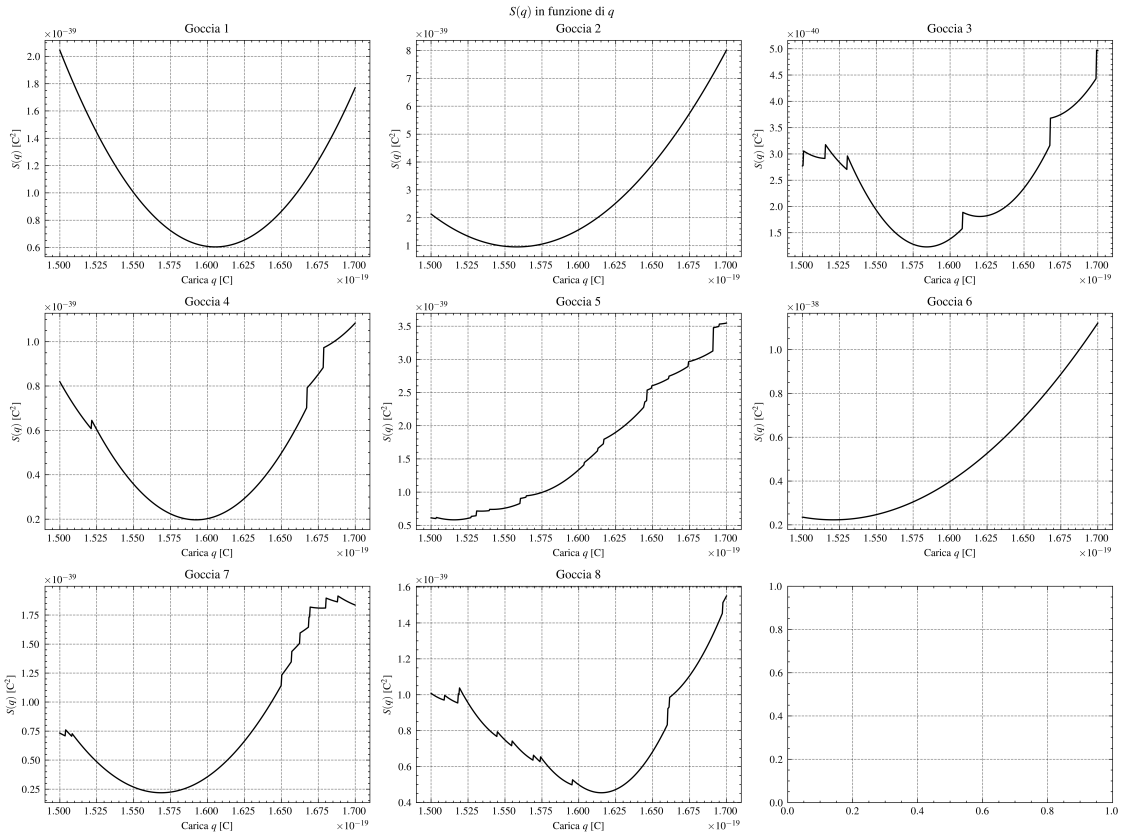
Usando la formula:

$$S(q) = \sum_{i=1}^{N} \left( \frac{Q_i}{k_i(q)} - q \right)^2$$

Dove $Q_i$ sono i valori delle $N$ cariche e $k_i(q)$ è l'intero più vicino al rapporto: $\frac{Q_i}{q}$.

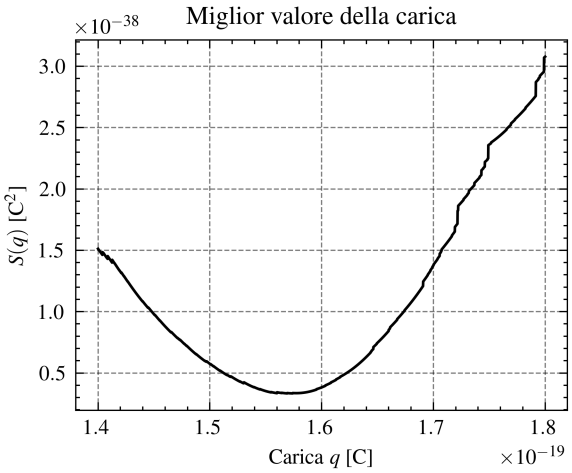
Nel codice $Q_i$ corrisponde a `charges[i]` e $q$ corrisponde allo spazio `x`.

[ ]:
```
plot_row = np.ceil(np.sqrt(n_gocce)).astype(int)
plot_col = np.ceil(n_gocce / plot_row).astype(int)
fig, axs = plt.subplots(plot_col, plot_row, figsize=(plot_row * 4, plot_col *␣
 ↪3))

for i in range(n_gocce):
    x = np.linspace(1.5e-19, 1.7e-19, 500)
    y = 0

    for charge in charges[i]:
        if not np.isnan(charge):  # ignore nans
            y += np.square(x - charge / np.round(charge / x))  # this is S(q)

    n, m = i // plot_row, i % plot_row
    axs[n, m].plot(x, y)

    axs[n, m].set_title(f"Goccia {i + 1}")
    axs[n, m].set_xlabel(r"Carica $q$ [C]")
    axs[n, m].set_ylabel(r"$S(q)$ [C$^2$]")

plt.suptitle(r"$S(q)$ in funzione di $q$")
plt.tight_layout()
# plt.savefig("Images/Carica delle gocce.svg") # todo una volta scaricata␣
 ↪l'immagone in svg rimuovere "goccia 9"
plt.show()
```




### 1.2.5 Plot finale

[ ]:
```
charges = reduce(
    np.union1d, (charges[:3], charges[4:5], charges[7:])
).flatten()  # add all charges together, excluding goccia 3, 5 e 6

charges = charges[~np.isnan(charges)]  # remove nans

fig, ax = plt.subplots()

x = np.linspace(1.4e-19, 1.8e-19, 3000)
y = 0
for charge in charges:
    y += np.square(x - charge / np.round(charge / x))  # this is S(q)

plt.plot(x, y)
plt.title("Miglior valore della carica")
plt.xlabel(r"Carica $q$ [C]")
plt.ylabel(r"$S(q)$ [C$^2$]")
# plt.savefig("Images/Carica.svg")
plt.show()
```




### 1.2.6 Compute $q_{min}$ and $\sigma_{q_{min}}$

Consideriamo il valore dell'introno della carica minima pari a $q_intor = 1.6 \cdot 10^{-19} C$.

Usando le formule:

$$q_{min} = \frac{1}{N} \sum_{i=1}^{N} \frac{Q_i}{k_i}$$

Dove $Q_i$ sono le cariche $i$-esime ($i$ da 1 a $N$ pari al numero totale di cariche) e $k_i$ l'intero più vicino al rapporto tra $Q_i$ e $q_intor$.

$$\sigma_{q_{min}} = \sqrt{\frac{S(q_{min})}{N(N-1)}}$$

[ ]:
```
N = len(charges)
q_intor = 1.6e-19  # utilizzo l'intorno di 1.6 per cercare q_e

# Valore di q_min calcolato come punto stazionario di S(q)
q_min = 1 / N * np.sum(charges / np.round(charges / q_intor))

# Incertezza di q_min
sigma_q_min = np.sqrt(
    np.sum(np.square(q_min - charges / np.round(charges / q_min)))  # S(q_min)
    / (N * (N - 1))  # denominatore
)

print(
    Style.BRIGHT
    + Fore.GREEN
```

```
    + "Valore di q_min: "
    + Fore.RED
    + f"{q_min*1e19:.3f} ±{sigma_q_min*1e19:.3f} *10^-19 C"
    + Style.RESET_ALL
)

print(f"Errore relativo: {sigma_q_min/q_min:%}")
```

```
Valore di q_min: 1.580 ±0.007 *10^-19 C
Errore relativo: 0.414730%
```

Vorrei far notare che sul libro c'è scritto che potremmo avere anche uno scostamento dal valore vero del 2-3%, quindi c'è qualcosa che non va con le incertezze (nel senso che dovrebbe venirci compatibile seppur di poco).

### 1.2.7 Calculate probabilities

Usando una gaussiana rispetto al valore universalmente accettato: $e = 1.6021 \cdot 10^{-19} C$.

[ ]:
```
z_value = abs(q_min - 1.6021e-19) / sigma_q_min
p_student = stats.norm.cdf(z_value)
p_value = 2 * (1 - p_student)

print(
    Style.BRIGHT
    + Fore.GREEN
    + "P-value di q_min compatibili con q_e vero: "
    + Fore.RED
    + f"{p_value:.2%}"
)
```

```
P-value di q_min compatibili con q_e vero: 0.06%
```