SOP

Single DMA scan inversion procedure

José Morán[a]

[a] *Department of Mechanical Engineering, University of Minnesota, Minneapolis, MN, 55455, USA.*

Hogan's Lab

July 29, 2022

Table of contents

1 About this document

- ***Who needs this document?*** *People using a neutralizer+DMA+CPC experimental configuration (see Fig. 3.1) and willing to conduct the inversion process to obtain the particle size distribution.*
- ***What do I need to use this procedure?*** *You may just use the methods discussed here for which you just need some basic knowledge of aerosols. You may read Hind's book [1] if neccessary. You may also need a Python compiler (and ideally Jupyter Notebook) to directly use the codes provided here.*
- ***Important assumptions*** *I assume you will use the Labview codes developed in this group to set the voltages on the DMA and read the signal from the CPC. I assume you are using a cylindric DMA (it could be nano or long DMA, the codes given here will work for both of them). It is assumed that particles are properly bipolarly charged prior to entering to the DMA which is achieved using a neutralizer. It is assumed that the sheath flow inlet and outlet in the DMA are same.*
- ***What can improve the quality of my results?*** *There some experimental aspects that will improve the quality of the inversion process. First, the sheath flow to aerosol flow in the DMA should be as high as possible. For example, our SMPS has a 4/1 ratio for aerosol to sheath flows but ratios as high as 10 are recommended. The less noisy is your measurement, the bettwe will be the inversion. Indeed, inversion means deconvolution which is highly sensible to noise. You can average different measurements or play with sampling times to reduce noise in your data.*

2 Introduction

3 Experimental setup

Figure 3.1 shows the experimental setup that you will likely be using for single DMA scanning. The sampled aerosol is injected to the DMA by prior passing through a neutralizer (typically Po-210 or Kr-85). The latter is essential for particles to have a ("known") bipolar charge distribution. Subsequently, the selected particles are injected into the CPC where particle counting takes place.

On the other hand, LabView is used in a computer connected to a DAQ via USB port. An output of the DAQ is connected to the high voltage (HV) power supply unit which provides the DMA with the desired voltage (typically between 5 V up to 10 kV). At the same time, the DAQ is connected to the CPC output (via BNC connector) to obtain the CPC reading.

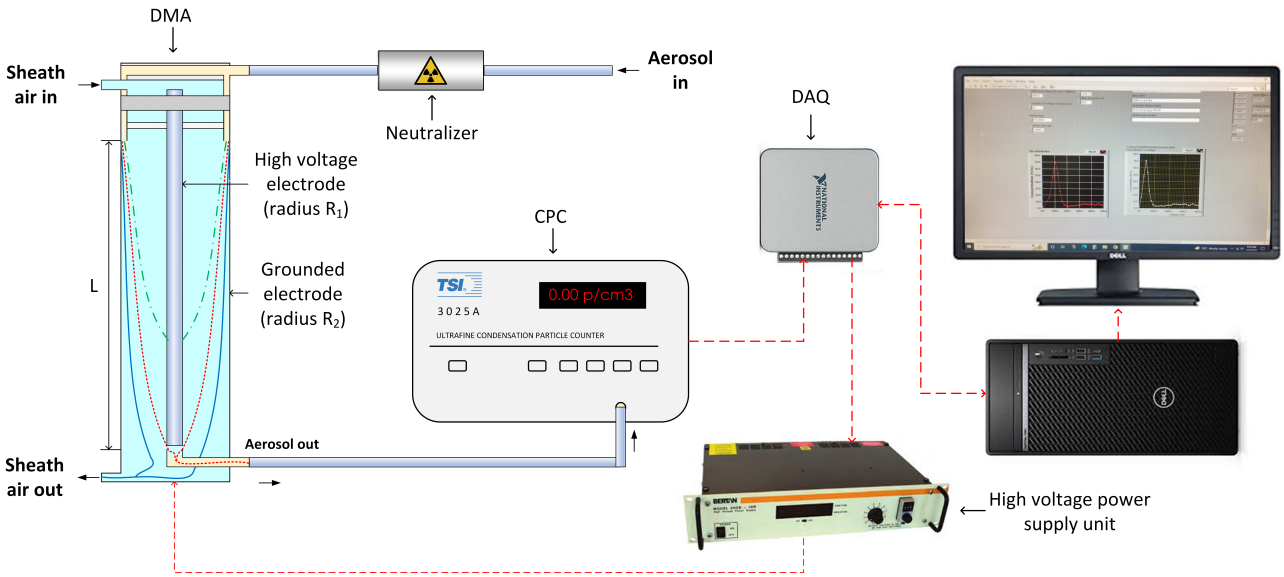

Figure 3.1: Experimental setup for single DMA scanning measurements.

4 Differential Mobility Analyzer

A long DMA is shown schematically on the left hand side of Fig. 3.1. As observed, the sheath flow is injected from an horizontal inlet and the flow is made uniform by using a screen mesh. This sheath flow continue down and is evacuated at the bottom of the DMA by a horizontal outlet. Particles are injected at the right inlet shown at the top and then directed downwards to be mixed with the sheath flow. These devices are developed to carry small particles (roughly < 500 nm) that are carried by the flows and do not significantly disturbe the sheath flow inside the DMA. Also, the outer electrode of the DMA is grounded and the central one is kept at high voltage (it may be positive or negative polarity) and particles will move radially towards the central electrode depending on their electrical mobility (Z_p),

$$Z_p = \frac{ze}{f}, \tag{4.1}$$

where z is the number of elemental electrical charges, $e = 1.602 \cdot 10^{-19}$ C is the value of one individual electrical charge, and $f = 3\pi\eta D_p / C_c(D_p)$ is the particle's friction coefficient (kg·s^{-1}) in the Stokes regime (Reynolds number lower than < 10). Also, η is the gas viscosity (Pa·s), D_p is the particle's mobility diameter (m), and $C_c(D_p) = 1 + \mathrm{Kn}\,(2.34 + 1.05\exp[-0.39/\mathrm{Kn}])$ is the Cunningham correction factor. This factor corrects the friction coefficient for the slip of gas molecules at the interface which depends on both the mean free path of the gas λ and D_p as expressed by the dimensionless Knudsen number $\mathrm{Kn} = \lambda / D_p$.

For those particles having opposite polarity than the central electrode 3 outcomes are possible: (1) as shown by the green dash-dotted line, the particles may just be collected by the central electrode if their electrical mobility is beyond a cricical value, (2) as shown by the blue continuous line, particles with electrical mobility lower than a critical value will not be collected at the aerosol outlet, and (3) as shown by the red dashed line, particles which electrical mobility is just between a specific range determined by the DMA voltage will be collected at the aerosol outlet.

Implementation in Python

In order to centralize all the properties of a DMA and the different functions to calculate its properties, we define the *class DMA* as shown below. This class is initialized with a "dict" Python data type containing the main information of the DMA including the *aerosol flow*, the *sheath flow* (both in lpm), the diffusion *effective length* to calculate the penetration efficiency (see section **??**), the DMA current *voltage* (V), and the *model* (it can be long or nano). Based on the model, the DMA length (L), internal electrode radius (R_1), and external electrode radius (R_2) are defined by the function *Load_parameters_model()*. Based on the DMA parameters we determine the selected electrical mobility is given by the following equation [2],

$$Z_p^* = \frac{q_s + q_c}{4\pi\Lambda V} \tag{4.2}$$

where q_s and q_c are the sheath flow in and out (m^3/s), respectively (considered the same). Also, the $\Lambda = L/\log(R_1/R_2)$ (denoted as A in the Python code below).

```
class DMA():
    def __init__(self, DMA_props):
        self.Reset_DMA(DMA_props)
        return
    def Reset_DMA(self, DMA_props):
        self.v_range = np.array([0.998e+0, 1.0001e+04])
        self.flow_aerosol = DMA_props["flow_aerosol"]   # L/min
        self.flow_sheath = DMA_props["flow_sheath"]     # L/min
        self.L_eff = DMA_props["L_eff"]   # m
        self.model = DMA_props["model"]
        self.V = DMA_props["Voltage"]     # V
        self.Load_parameters_model()
        self.A = self.L/np.log(self.R2/self.R1)
        self.Zp = (2*self.flow_sheath)*(1.66667e-5)/(4*np.pi*self.A*self.V
    )
        self.Charge_limit = DMA_props["Charge_limit"]
        return
        # Ambient properties
        self.Pressure = DMA_props["Pressure"]          # Pa
        self.Temperature = DMA_props["Temperature"]    # K
    def Load_parameters_model(self):
        if(self.model == "nano"):
            self.R1 = 0.937e-02   # m
            self.R2 = 1.905e-02   # m
            self.L = 4.987e-02    # m
        elif(self.model == "long"):
            self.R1 = 9.37e-03    # m
            self.R2 = 19.61e-03   # m
            self.L = 44.37e-02    # m
        return
```

4.1 DMA transfer function

DMA transfer function are determined as [3],

$$\Omega(\tilde{z},\beta,\sigma) = \frac{\sigma}{\sqrt{2}\beta}\left[\epsilon\left(\frac{\tilde{z}-(1+\beta)}{\sqrt{2}\sigma}\right) + \epsilon\left(\frac{\tilde{z}-(1-\beta)}{\sqrt{2}\sigma}\right) - 2\epsilon\left(\frac{\tilde{z}-1}{\sqrt{2}\sigma}\right)\right] \tag{4.3}$$

where $\tilde{z} = \frac{z}{z^s}$ is the dimensionless mobility, z is the particle mobility z^s is the centroid mobility selected by the DMA, $\epsilon = x\text{erf}(x) + \left(\exp(-x^2)/\sqrt{\pi}\right)$, erf is the error function, and $\beta = \frac{q_m}{q_s}$. The

parameter σ accounts for diffusional broading of the transfer function. Assuming plug flow, σ can be computed using the following equations,

$$\gamma = \left(\frac{R_1}{R_2}\right)^2$$

$$I = \frac{1}{2}(1+\gamma)$$

$$\kappa = \frac{LR_2}{R_2^2 - R_1^2}$$

$$G = \frac{4(1+\beta)^2}{(1-\gamma)}\left[I + \{2(1+\beta)\kappa\}^{-2}\right]$$

$$\sigma = \sqrt{\frac{2G\pi L d_{ab}}{q_{sh}}}$$

Fig. 4.2 shows an example of transfer function for a long DMA ($L = 443.69$ mm, $R_1 = 9.37$ mm, $R_2 = 19.61$ mm, $q_m = 1$ lpm, and $q_s = 5$ lpm) for a voltage $V = 100$ V (blue continuous line), and $V = 10$ kV (red dashed line). As observed in this figure, the probability to collect particles with a mobility $Z_p \to Z_p^*$ tends to 1 while particles within the $Z_p \to Z_p^* \in [0.8, 1.2]$ range can be collected with a non-null probability.

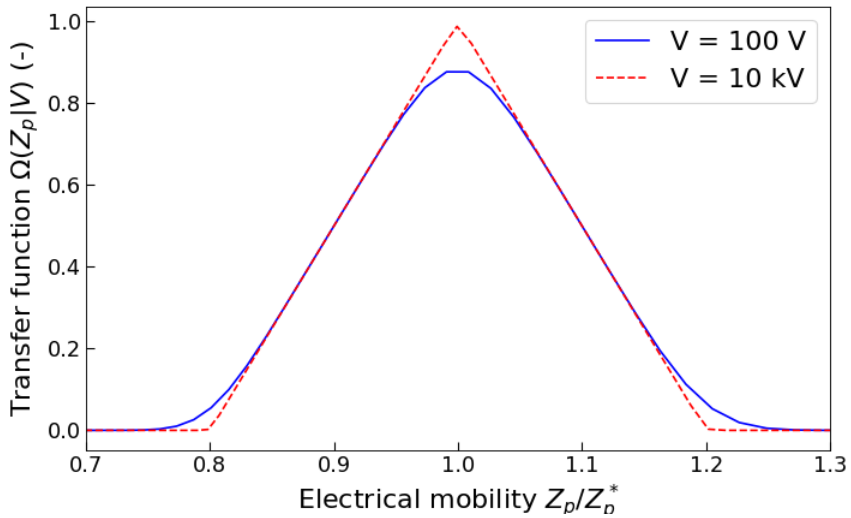

Figure 4.2: Transfer function of a DMA with $L = 443.69$ mm, $R_1 = 9.37$ mm, $R_2 = 19.61$ mm, $q_m = 1$ lpm, and $q_s = 5$ lpm.

Python implementation

The determination of the transfer function is implemented as a function belonging to the class DMA defined in section 4 as follows,

```python
class DMA():
    .
    .
    def Transfer_function_Tf(self,Dp,voltage,charge):
        Diff = self.Diffusion_coefficient(Dp)
        beta = self.flow_aerosol/self.flow_sheath
        gamma = np.power(self.R1/self.R2,2)
        I = 0.5 * (1 + gamma)
        kappa = self.L * self.R2/(np.power(self.R2,2)-np.power(self.R1,2))
        G = 4 *np.power(1+beta,2)/(1-gamma) * (I+np.power(2*(1+beta)*kappa,-2))
        sigma = np.sqrt(2*G*np.pi*self.L*Diff/(self.flow_sheath*1.66667e-5))
        Z = self.Electric_mobility(Dp,charge)
        Zp = self.Calculate_Zp(voltage)#/charge
        z_til = np.abs(Z/Zp)
        tf = sigma/(np.sqrt(2)*beta)*(self.erf_func((z_til-(1+beta))/(np.sqrt(2)*sigma))+\
                                      self.erf_func((z_til-(1-beta))/(np.sqrt(2)*sigma))-\
                                      2*self.erf_func((z_til-1)/(np.sqrt(2)*sigma)))
        return np.max([tf,1e-60])
```

4.2 Penetration efficiency

Penetration efficiency through the DMA is computed based on the parameterized results from Ref. [4],

$$T = 0.82\exp(-11.5u) + 0.1\exp(-70.0u) + 0.03\exp(-180.0u) + 0.02\exp(-340.0u) \tag{4.4}$$

where $u = Dl_{\text{eff}}/q_m$, l_{eff} is the parameterized effective diffusion length, and q_m is the aerosol flow rate through the DMA, and $D = k_B T/f$ is the particle diffusion coefficient, $k_B = 1.380 \cdot 10^{-23}$ $\text{m}^2\text{kg}\,\text{s}^{-2}\,\text{K}^{-1}$ is the Boltzmann constant, and T is the gas temperature (K). Fig. 4.3 shows the penetration efficiency as determined based on Eq. (4.4) assuming $l_{\text{eff}} = 13$ m, $q_m = 1$ lpm, and $q_s = 5$ lpm. As observed in this figure, the penetration efficiency tends to 0 for particles with $D_p \ll 10$ nm while for larger diameters it increases steadily to reach a plateau close to 0.9 for roughly $D_p > 50$.

Implementation in Python

This is implemented as a function belonging to the class DMA described in section 4,

```python
class DMA():
    .
    .
    def Penetration_efficiency_Tl(self,Dp):
        Diff = self.Diffusion_coefficient(Dp)
        q_sa = self.flow_aerosol * (1.66667e-5)
        u = Diff * self.L_eff/q_sa
        t = np.zeros(4)
        t[0] = 0.82 * np.exp(-11.5 * u)
        t[1] = 0.10 * np.exp(-70.0 * u)
        t[2] = -0.03 * np.exp(-180.0 * u)
        t[3] = -0.02 * np.exp(-340.0 * u)
        return np.sum(t)
```

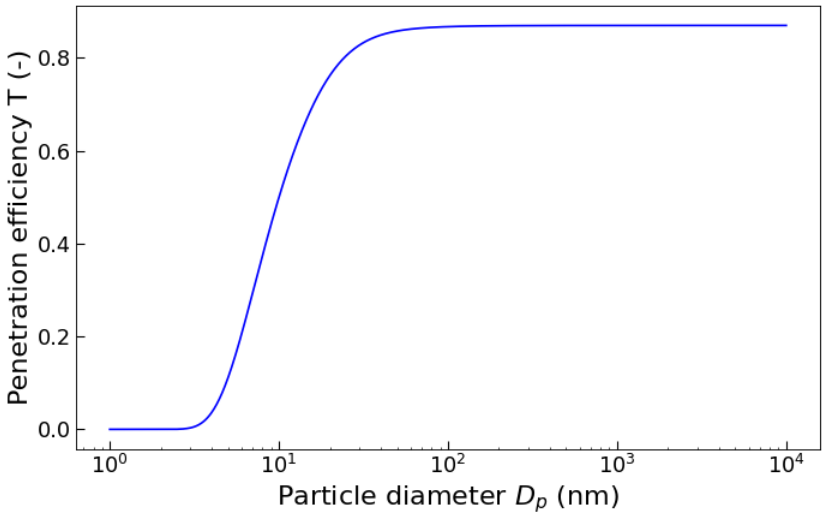

Figure 4.3: DMA characteristic penetration efficiency considering $l_{\text{eff}} = 13$ m, $q_m = 1$ lpm, and $q_s = 5$ lpm.

4.3 Bipolar charge distribution

When particles traverse a DMA it is important to know their electrical charge state. For this reason, a neutralizer is typically used to bring particles into a known charge state by exposing them to bipolar ions generated by a radiation source such as Po-210 or Kr-85 [5].

Charging efficiency (charge equilibrium) obtained in the bipolar charger is computed based on the parameterized measurements by Wiedensohler et al. [6].

$$f(z) = 10^{\left\{\sum_{i=1}^{6} a_i(z)\left[\log\left(\frac{D_p}{nm}\right)\right]^{i-1}\right\}} \tag{4.5}$$

where $z = -2, -1, 1, 2$ is the number and polarity of particle charge and a_i are empirical coefficients based on Fuchs theory [6]. This equation is valid for $1 < D_p < 1000$ nm for $z = [-1, 1]$ and $20 < D_p < 1000$ nm for $z = [-2, 2]$. For charges $z < -2$ or $z > 2$ this can be approximated based on the Boltzmann distribution [7],

$$f(z, D_p) = \left(\frac{Ke^2}{\pi D_p k_B T}\right)^{1/2} \exp\left[\frac{-K_E z^2 e^2}{D_p k_B T}\right] \tag{4.6}$$

where $K = 9.0 \cdot 10^9$ Nm^2/C^2, k_B is the Boltzmann constant, e is the elementary charge, and D_p is the mobility diameter of the particle. For a fixed mobility diameter can be interpreted as a normal distribution with variance $\sigma_z^2 = (D_p k_B T)/(2Ke^2)$ where the variable is the number of elemental charges z_n,

$$f(z_n)|_{D_p} = \left(\frac{1}{2\pi\sigma_z}\right)^{1/2} \exp\left[\frac{-z_n^2}{2\sigma_z^2}\right] \tag{4.7}$$

Note that another option to be implemented in the future, is to determine the particle charging efficiency includes the model proposed by Gopalakrishnan et al. [8].

Fig. 4.4 shows the charging efficiency predicted by Eq. (4.6) for three different states of charges (1: black continous line, 3: blue dashed line, and 4: red dash-dotted line).

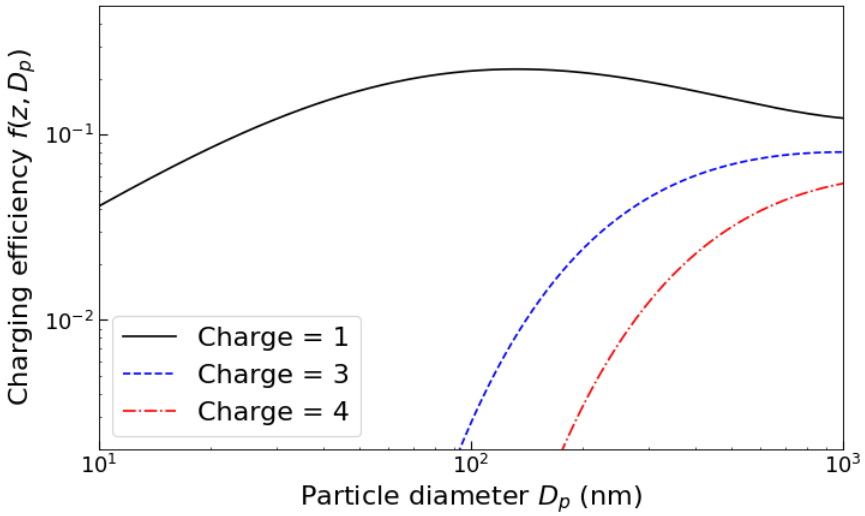

Figure 4.4: Particles bipolar charge after passing through a neutralizer.

Implementation in Python

This is implemented as a function belonging to the class DMA described in section 4,

```
class DMA():
    .
    .
    def Charging_efficiency_Tc(self,Dp,charge):
        # Charges [-2, +2]: Wiedensohler et al. (1988) parameters
        if((charge >= -2) and (charge <= 2)):
            a = {
            "-2": [-26.3328,35.9044,-21.4608,
                   7.0867,-1.3088,0.1051],
            "-1": [-2.3197,0.6175,0.6201,
                   -0.1105,-0.1260,0.0297],
            "0": [-0.0003,-0.1014,0.3073,
                  -0.3372,0.1023,-0.0105],
            "1": [-2.3484,0.6044,0.4800,
                  0.0013,-0.1544,0.0320],
            "2": [-44.4756,79.3772,-62.8900,
                  26.4492,-5.7480,0.5059] }
            ch_exp = 0
            ai = a[str(charge)]
            for i in range(len(ai)):
                ch_exp += ai[i] * np.power(np.log10(Dp*1e+09),i)
            ch_frac = np.power(10,ch_exp)
        else:
            K_e = 9e+09 # N*m^2/C^2
            temp10 = np.pi * Dp *Aerosol_tools.k_B * self.Temperature/np.
   power(Aerosol_tools.q_e,2)/K_e
            temp11 = np.power(charge,2)
            temp12 = temp10 / np.pi
            ch_frac = (1/np.sqrt(temp10)) * np.exp(-temp11/temp12)
        return ch_frac
```

5 Data inversion procedure

5.1 Fredholm equation

Under the experimental setup discussed in section 3, the voltage of the DMA is commonly varied from $V_1 = 5$ V to $V_\theta = 10$ kV in scanning mode then the particle size distribution dN/dD_p, where N is the particle number concentration ($\#/\text{cm}^3$), is obtained by inverting the following expression known as the Fredholm equation,

$$R(V) = \sum_{z_n} \int_{D_p} \Omega(Z_p|V) T(D_p) f(z_n, D_p) \frac{\partial N}{\partial D_p} dD_p \tag{5.8}$$

Here, $\Omega(Z_p|V)$ is the DMA transfer function (see section 4.1), $T(D_p)$ is the penetration efficiency (see section 4.2), and $f(z_n, D_p)$ is the bipolar charge fraction (see section 4.3). At a fixed DMA voltage, particles of any size and any state of charges can enter the DMA and therefore we should sum over all posibles states of charges ($z_n \in [1, \infty]$) and integrate over all sizes ($D_p > 0$). However, in practice after neutralization, most of the particles will have a state of charge between $[-3, +3]$. Also, the integral in this equation, can be discretized according to the finite number of voltages analyzed $i \in [1,\, \theta]$ as,

$$R(V_i) = \sum_{z_n} \sum_{D_{p,i}} \Omega(Z_p|V_i) T(D_{p,i}) f(z_n, D_{p,i}) \frac{\partial N}{\partial D_p} \Delta D_{p,i} \tag{5.9}$$

This problem can be writen in a matrix form as,

$$\mathbf{b} = \mathbf{A}\mathbf{x} + \varepsilon \tag{5.10}$$

where ε is the vector of experimental errors and the other matrix and vectors are given as follows,

$$\mathbf{b} = \begin{pmatrix} R(V_1) \\ \vdots \\ R(V_\theta) \end{pmatrix}$$

$$\mathbf{A} = \begin{pmatrix} \sum_{z_n} \Omega(Z_p|V_1) T(D_{p,1}) f(z_n, D_{p,1}) & \dots & \sum_{z_n} \Omega(Z_p|V_1) T(D_{p,\theta}) f(z_n, D_{p,\theta}) \\ \vdots & \ddots & \vdots \\ \sum_{z_n} \Omega(Z_p|V_\theta) T(D_{p,1}) f(z_n, D_{p,1}) & \dots & \sum_{z_n} \Omega(Z_p|V_\theta) T(D_{p,\theta}) f(z_n, D_{p,\theta}) \end{pmatrix}$$

$$\mathbf{x} = \begin{pmatrix} \frac{\partial N}{\partial D_{p,1}} \Delta D_{p,1} \\ \vdots \\ \frac{\partial N}{\partial D_{p,\theta}} \Delta D_{p,\theta} \end{pmatrix}$$

The dimensions of $\mathbf{A}$ are $\theta \times \theta$, $\mathbf{b}$ is $\theta \times 1$, and $\mathbf{x}$ is $\theta \times 1$.

Implementation in Python

This is implemented as a function belonging to the class DMA described in section 4,

```python
class DMA():
    .
    .
    def Convolution_matrix(self, Dp, Voltage):
        n = len(Dp)
        k_vec = np.array(range(self.Charge_limit)) + 1
        A_mat = np.zeros((n,n))
        for i in range(n):
            for j in range(n):
                Tl = self.Penetration_efficiency_Tl(Dp[j])
                for charge in k_vec:
                    Tc = self.Charging_efficiency_Tc(Dp[j],charge)
                    Tf = self.Transfer_function_Tf(Dp[j],Voltage[i],charge
    )
                    A_mat[i,j] += Tf * Tc * Tl
        self.A_mat = A_mat
        return A_mat
```

5.2 Tikhonov regularization

Equation (5.10) is a Fredholm integral which is ill-posed and therefore requieres a regularization algorithm to be accurately solved. Under this method, the equation (5.10) can be solved as (see equation 1.15 from [9]),

$$\widetilde{\mathbf{x}} = (\mathbf{A}^t\mathbf{A} + \alpha\mathbf{I})^{-1}\mathbf{A}^t\mathbf{b} \tag{5.11}$$

where the exponent t means the transpose matrix, $\mathbf{I}$ is the identity matrix (with dimensions $\theta \times \theta$) and α is the regularization parameter ($\alpha \in [0, \infty]$).

Implementation in Python

This is implemented as a function belonging to the class DMA described in section 4,

```python
class DMA():
    .
    .
    def Deconvolution(self, R, A, alpha):
        H = np.dot( A.transpose(), A) + alpha*np.identity(A.shape[1])
        rhs = np.dot( A.transpose(), R)
        return np.linalg.solve(H, rhs)
```

When α is very small noise filtering is not enough and $\widetilde{\mathbf{x}}$ will be oscillatory. On the other hand, when α is large, then noise is considerably reduced. However, most components of the solution are also filtered out and $\widetilde{\mathbf{x}}$ becomes overly smooth [9].

6 Examples of inversion

6.1 Inverting an idealized smooth curve

In order to probe the correct inversion procedure proposed in this work, we will use the example given by Markus Petters's GitHub [10] (see Notebook S02).

Implementation in Python

The following script is an example of inversion,

```
from linker import *
%matplotlib notebook

# Import Petters' data
cols = ["Dp","N","R","Z","Zs","V2"]
test_case = pd.read_csv('Test_inversion/convolution_case.txt', sep = "\t",
    names = cols)
test_case.sort_values(by="V2", inplace=True, ignore_index=True)

Dp0 = test_case.Dp.values * 1e-09
Voltage0 = test_case.V2.values
R0 = test_case.R.values
N0 = test_case.N.values

dNdLogD0 = np.zeros_like(N0)
dNdLogD0[0] = N0[0]/np.log10(Dp0[1]/Dp0[0])
for i in range(1,len(N0)):
    dNdLogD0[i] = N0[i]/np.log10(Dp0[i]/Dp0[i-1])

# Setup DMA
DMA_props = {
    "model":          "long",
    "flow_aerosol":   1,
    "flow_sheath":    5,
    "Voltage":        100,
    "L_eff":          13,
    "Pressure":       101000,
    "Temperature":    300,
    "Charge_limit":   6}
dma0 = DMA_tools.DMA(DMA_props)

dNdLogD,dNdLogD_r,R_conv = DMA_tools.Deconvolute_singleDMA(dma0,test_case)

A0 = dma0.Convolution_matrix(Dp0, Voltage0)
R_conv1 = dma0.Convolution(A0, N0)
N_alpha = dma0.Deconvolution(R0, A0, 1e-03)
R_conv2 = dma0.Convolution(A0, N_alpha)

# Plot results
fig, ax = plt.subplots()
plt.plot(Dp0 * 1e+09, R0,"o r", label="Read DMA")
plt.plot(Dp0 * 1e+09, R_conv1,".b", label="Convoluted (original)")
plt.plot(Dp0 * 1e+09, R_conv2,"-c", label="Convoluted (inverted)")
plt.ylabel("Number concentration, R (cm$^{-3}$)", fontsize=20)
plt.xlabel("Particle diameter (nm)", fontsize=20)
plt.legend(fontsize=20); plt.grid(); plt.show()

fig, ax = plt.subplots()
plt.plot(Dp0 * 1e+09, dNdLogD0,"o r", label="Original")
plt.plot(Dp0 * 1e+09, dNdLogD,"-c", label="Inverted")
plt.ylabel("dN/dLog(Dp) (cm$^{-3}$)", fontsize=20)
plt.xlabel("Particle diameter (nm)", fontsize=20)
plt.legend(fontsize=20); plt.grid(); plt.show()
```

In the first line of this script we import a file called *linker*. This a file used to call all the Python libraries necessary for this script to work including Numpy, Pandas, Matplotlib, tqdm, etc. Also, a Python file called *DMA_tools* is called, which contains the class DMA described in section 4. The second line is only needed if you are working in Jupyter Notebeook. In addition, lines 5-7 will open the data to be analyzed as a Pandas dataframe. Lines 9-12 will obtain the important parameters from this dataframe including the vector of diameters (*Dp0*), the vector of voltages (*Voltage0*), the vector of DMA number concentration reading (*R0*), and the originally inverted number concentration (*N0*). Subsequently (lines 14-17), the inverted number concentration

is expressed as $dN/dLog(D_p)$ for comparison with our own inversion approach.

Lines 20-28 will setup the DMA parameters corresponding to a long DMA with an aerosol flow of 1 lpm, a sheath flow of 5 lpm, a reference voltage of 100 V (to initialize data only), an effective length $L_{eff} = 13$, atmospheric pressure and temperature, and a charge limit of 6. Line 29 initializes the DMA class with these parameters. Subsequently, in line 31 the inversion is carried out. Note that, the output of this inversion are three vectors corresponding to the inverted $dN/dLogDp$ ($dNdLogD$) and the same vector after applying a noise reduction filter ($dNdLogD\_r$), and the convolution of the inverted data (R_conv). At this point the data has been already inverted and lines 33-36 are just added for teaching purposes in this tutorial. Finally, lines 39-52 are used to plot the results. Such results are shown in Fig. 6.5.

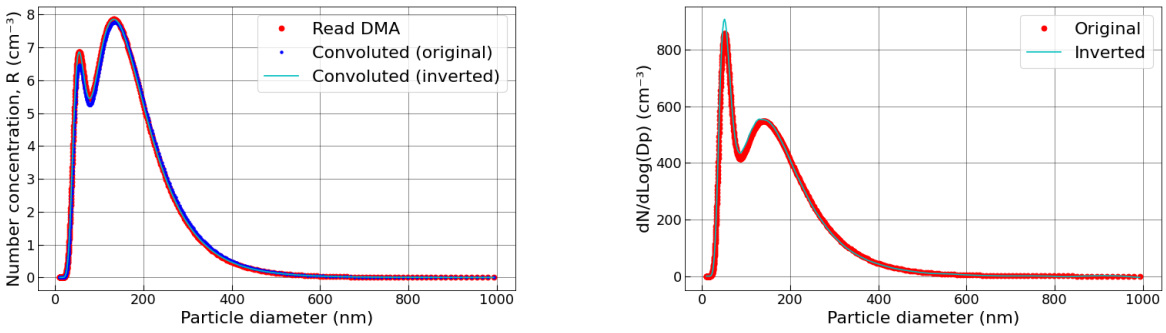

Figure 6.5: Example of single DMA inversion. Left figure shows the DMA read in number concentration, the right figure shows the inverted data in $dN/dLog(D_p)$.

6.2 Inverting noisy data

In this section we are adding a Gaussian noise to the data where the observed values are multiplied by a Gaussian random variable with average 1 and standard deviation σ_{noise} given as a percentage of the peak value in the data. The corresponding Python script is given below,

```
1  cols = ["Dp","N","R","Z","Zs","V2"]
2  test_case = pd.read_csv('Test_inversion/convolution_case.txt', sep = "\t",
       names = cols)
3  test_case.sort_values(by="V2", inplace=True, ignore_index=True)
4
5  percent_noise = 20
6  noise = np.random.normal(1,percent_noise/100,len(R0))
7
8  Dp0 = test_case.Dp.values * 1e-09
9  Voltage0 = test_case.V2.values
10 R0 = test_case.R.values
11 R0 = R0* noise
12 test_case.R = R0
13
14 dNdLogD,dNdLogD_r,R_conv = DMA_tools.Deconvolute_singleDMA(dma0,test_case)
```

The results are shown in FIg. 6.6 at 1% and 10% noise levels. The blue curve on the right column correspond to the variable $dNdLogD\_r$ corresponding to a filtered (for noise reduction) version of $dN/dLog(D_p)$.

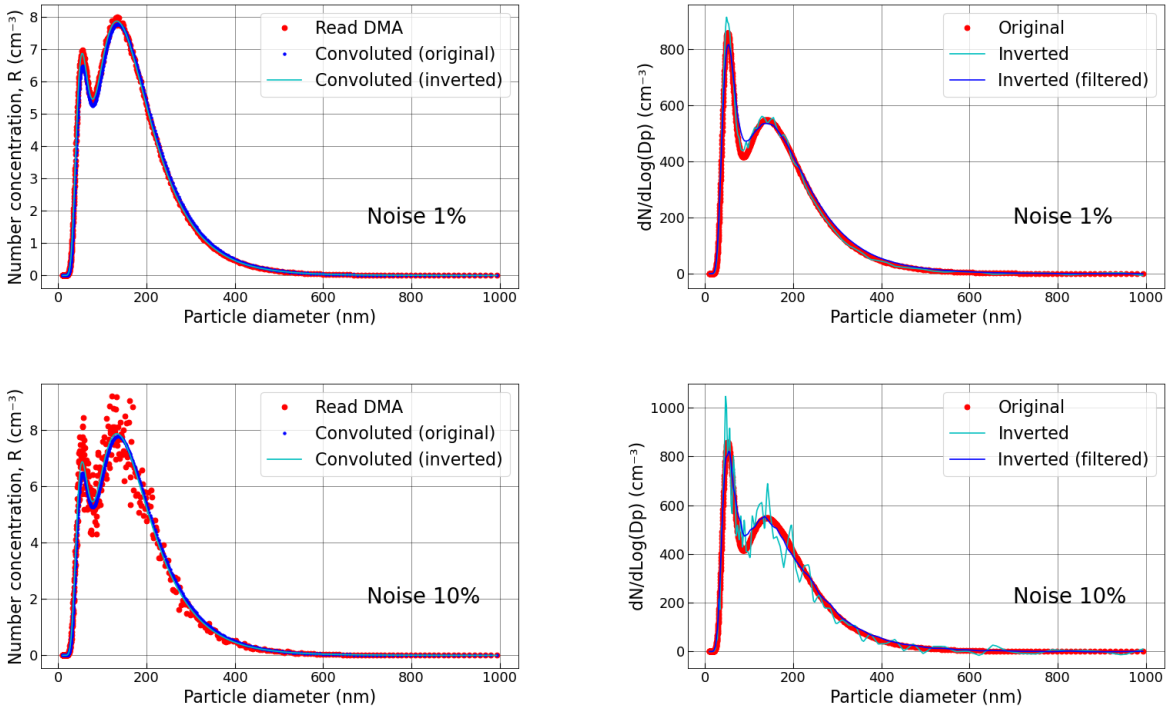

Figure 6.6: Example of single DMA inversion of noisy data. The introduced noise level is indicated on each figure.

References

[1] William C Hinds and Yifang Zhu. *Aerosol technology: properties, behavior, and measurement of airborne particles*. John Wiley & Sons, 2022.

[2] EO Knutson and KT Whitby. Aerosol classification by electric mobility: apparatus, theory, and applications. *Journal of Aerosol Science*, 6(6):443–451, 1975.

[3] Mark R Stolzenburg and Peter H McMurry. Equations governing single and tandem dma configurations and a new lognormal approximation to the transfer function. *Aerosol Science and Technology*, 42(6):421–432, 2008.

[4] A Reineking and J Porstendörfer. Measurements of particle loss functions in a differential mobility analyzer (tsi, model 3071) for different flow rates. *Aerosol Science and Technology*, 5(4):483–486, 1986.

[5] Anne Maißer, Jikku M Thomas, Carlos Larriba-Andaluz, Siqin He, and Christopher J Hogan Jr. The mass–mobility distributions of ions produced by a po-210 source in air. *Journal of Aerosol Science*, 90:36–50, 2015.

[6] A Wiedensohler. An approximation of the bipolar charge distribution for particles in the submicron size range. *Journal of aerosol science*, 19(3):387–389, 1988.

[7] M Matti Maricq. Thermal equilibration of soot charge distributions by coagulation. *Journal of aerosol science*, 39(2):141–149, 2008.

[8] Ranganathan Gopalakrishnan, Peter H McMurry, and Christopher J Hogan Jr. The bipolar diffusion charging of nanoparticles: A review and development of approaches for non-spherical particles. *Aerosol Science and Technology*, 49(12):1181–1194, 2015.

[9] Curtis R Vogel. *Computational methods for inverse problems*. SIAM, 2002.

[10] Petters Markus. DMA inversion tutorials, 2022.