Homework 1

Masoud Akbarzadeh

2/12/2024

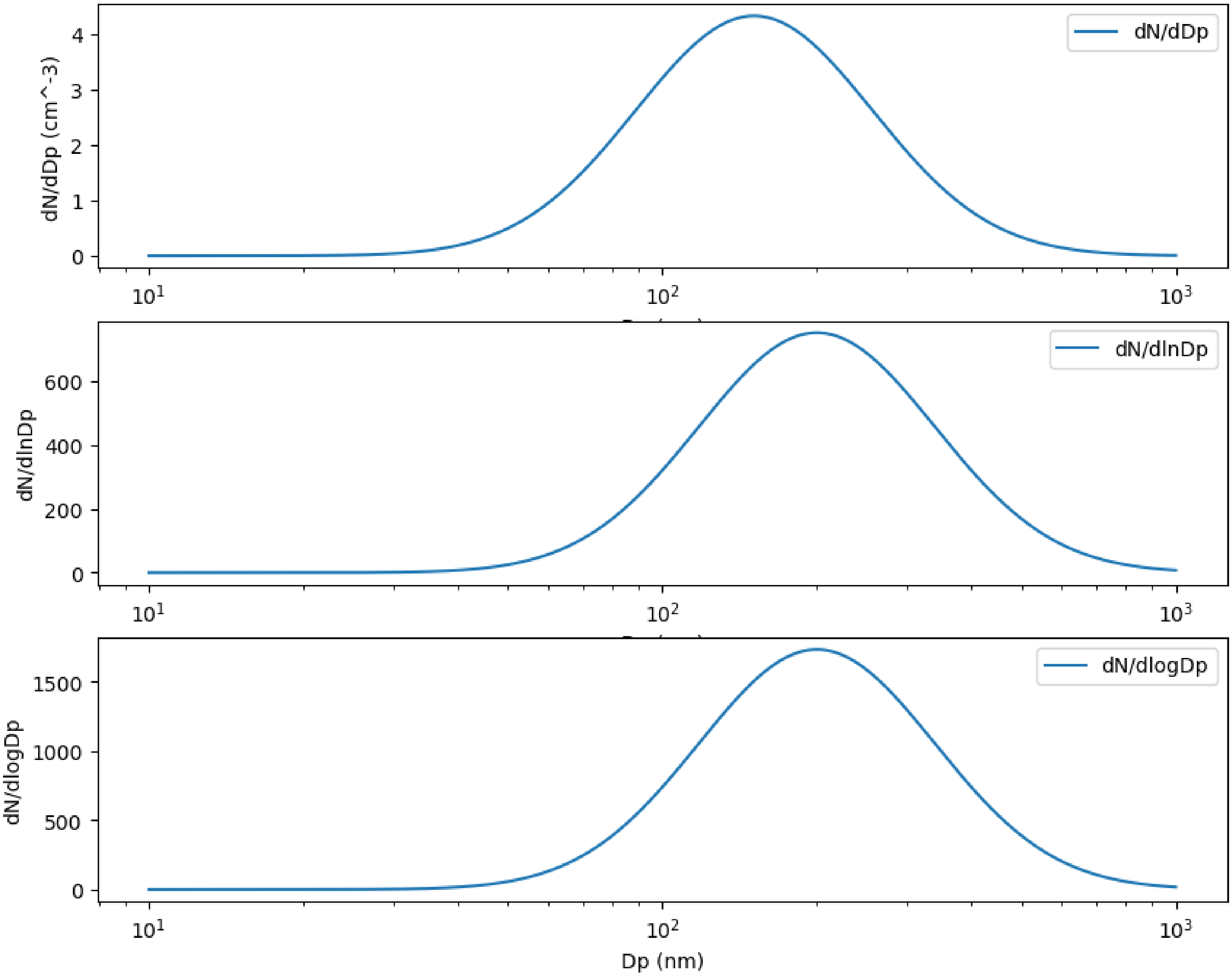
—

Aerosol Physics & Chemistry

—

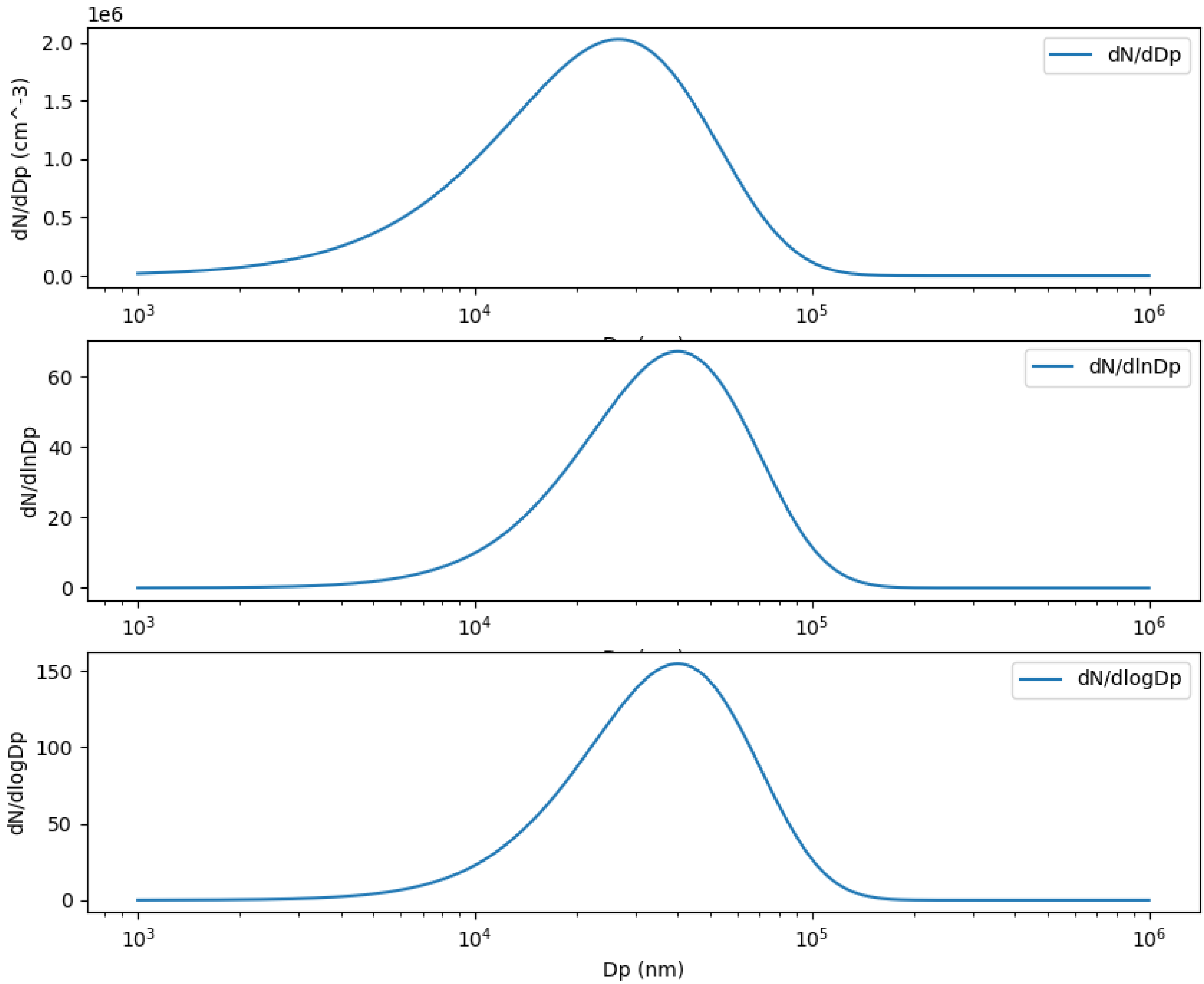
Problem 1

1. a.

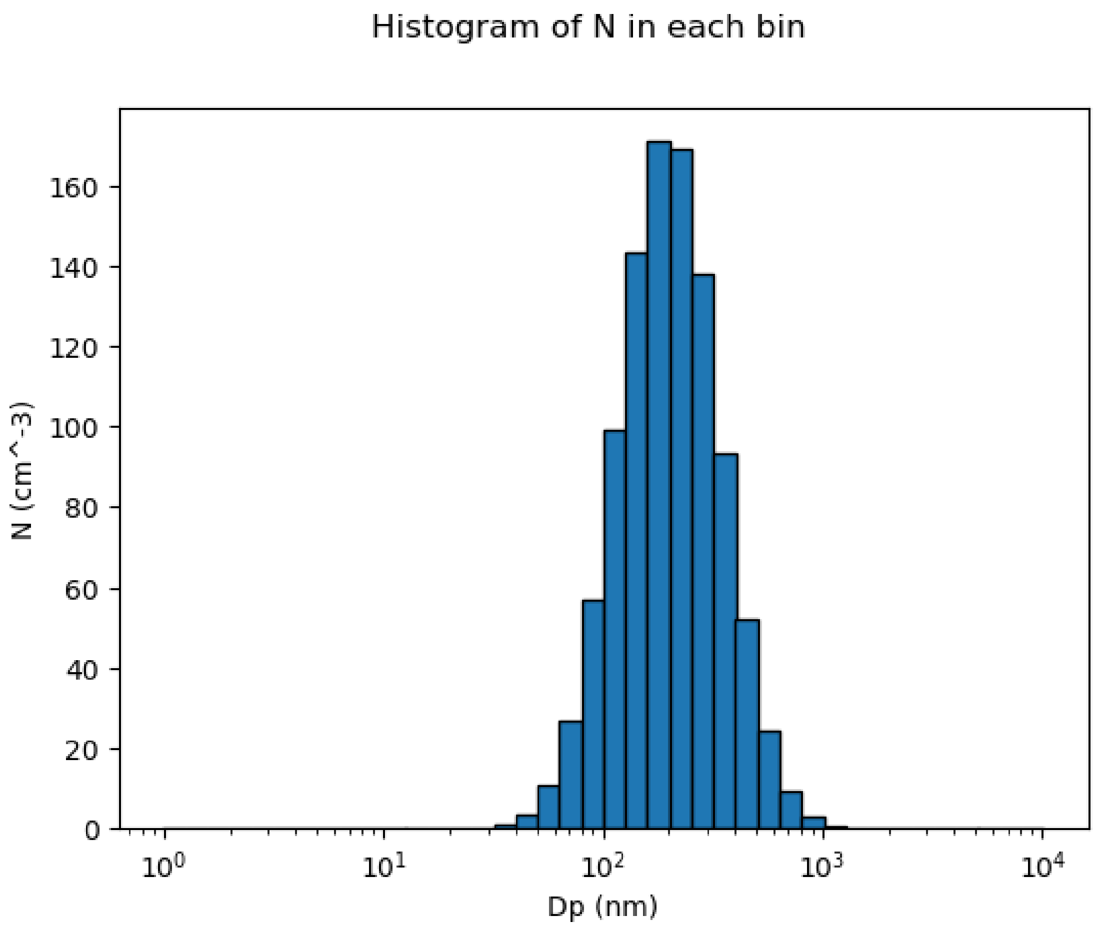


dN/dDp
dN/dDp (cm^-3)
0
1
2
3
4
dN/dlnDp
dN/dlnDp
0
200
400
600
dN/dlogDp
dN/dlogDp
0
500
1000
1500
10^1
10^2
10^3
Dp (nm)

1. b



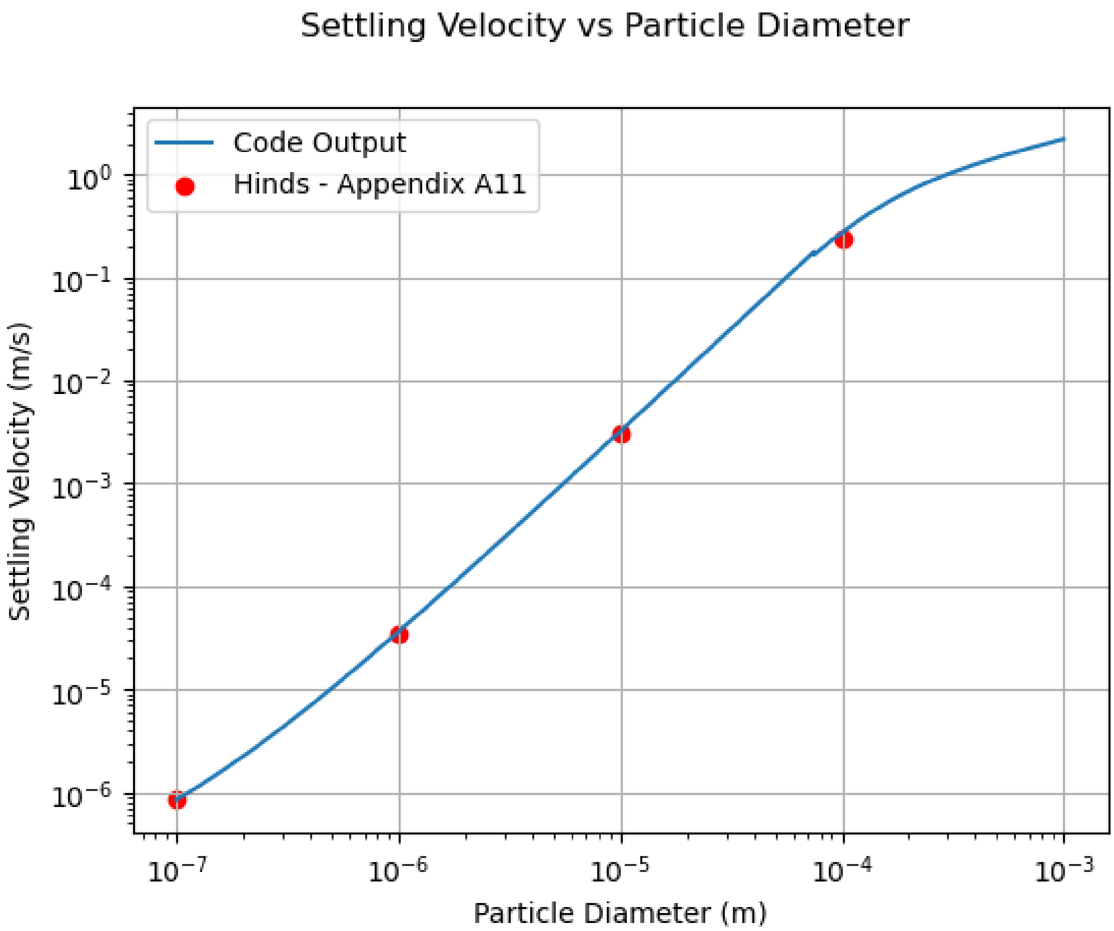
1. **C**

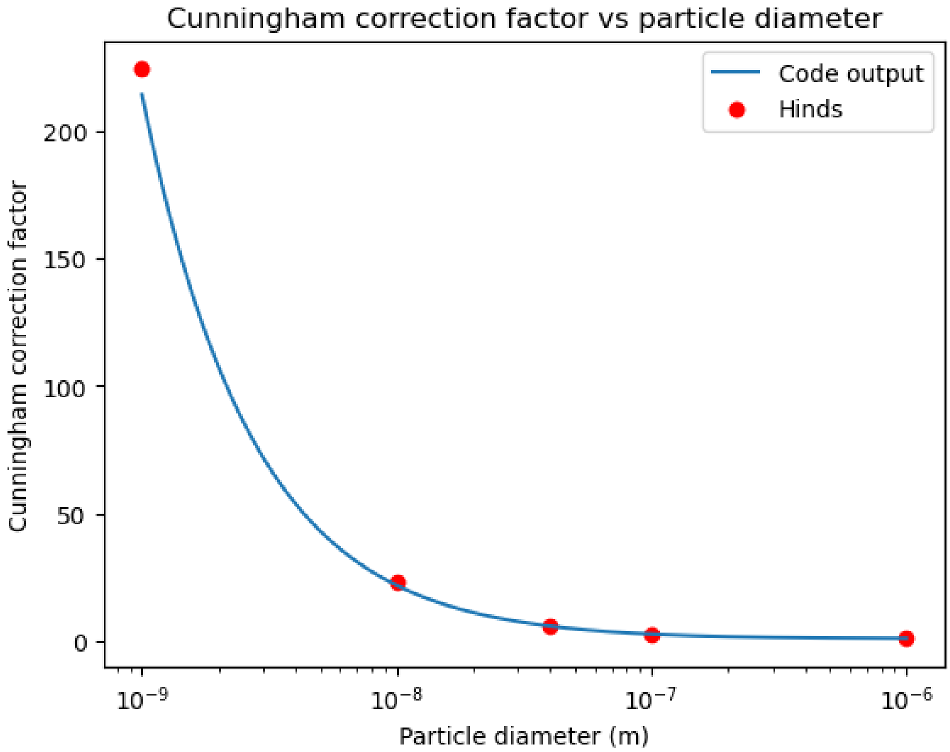


Histogram of N in each bin
N (cm^-3)
0
20
40
60
80
100
120
140
160
10^0
10^1
10^2
10^3
10^4
Dp (nm)

Problem 2

2. Assumptions:
   Air:
   M = 0.289 kg/mol
   Viscosity = 1.84e-5 Pa.s
   Density = 1.18 kg/m^3
   Mean free path = 6.67e-8 m




Cunningham correction factor vs particle diameter
Code output
Hinds
Cunningham correction factor
200
150
100
50
0
10^{-9}
10^{-8}
10^{-7}
10^{-6}
Particle diameter (m)

```
#%%
# Importing libraries
import matplotlib.pyplot as plt
import numpy as np
import matplotlib as mpl

mpl.rcParams["figure.dpi"] = 100
#%%
# all diameters are in nm
S_g = 1.7 # standard deviation of the lognormal distribution
N = 1000 # number of particles in the distribution (cm^-3)
D_pg = 200 # geometric mean diameter (nm)

Dp = np.geomspace(1e-8, 1e-6, 1000)*1e9 # convert to nm
dN_dDp = N/(np.sqrt(2*np.pi)*np.log(S_g)*Dp)*np.exp(-(np.log(Dp/D_pg
))**2/(2*(np.log(S_g))**2))

dN_dlogDp = dN_dDp*Dp*np.log(10)
dN_dlnDp = dN_dDp*Dp
#%%
# Creating subplots
fig, axs = plt.subplots(3, 1, figsize=(10, 8))

# First plot
axs[0].plot(Dp, dN_dDp)
axs[0].set_xscale('log')
axs[0].set_xlabel('Dp (nm)')
axs[0].set_ylabel('dN/dDp (cm^-3)')
axs[0].legend(['dN/dDp'])

# Second plot
axs[1].plot(Dp, dN_dlnDp)
axs[1].set_xscale('log')
axs[1].set_xlabel('Dp (nm)')
axs[1].set_ylabel('dN/dlnDp')
axs[1].legend(['dN/dlnDp'])

# Third plot
axs[2].plot(Dp, dN_dlogDp)
axs[2].set_xscale('log')
axs[2].set_xlabel('Dp (nm)')
axs[2].set_ylabel('dN/dlogDp')
axs[2].legend(['dN/dlogDp'])

plt.savefig('1-2.png', bbox_inches='tight')
plt.show()
#%%
bin_number = 40
bins_lower = np.geomspace(1e-9, 10.3e-6, bin_number + 1)  #
bins_upper = bins_lower[1:]
bins_lower = bins_lower[:-1]
bins_mid = np.sqrt(bins_lower * bins_upper)  # geometric mean
for i in range(bin_number):
```

```
    print(i, bins_lower[i], bins_upper[i], bins_mid[i])
#%%
# all diameters are in nm
S_g = 1.7 # standard deviation of the lognormal distribution
N = 1000 # number of particles in the distribution (cm^-3)
D_pg = 200e-9 # geometric mean diameter (nm)

Dp = bins_mid
dN_dDp = N/(np.sqrt(2*np.pi)*np.log(S_g)*Dp)*np.exp(-(np.log(Dp/D_pg
))**2/(2*(np.log(S_g))**2))
N = dN_dDp * (bins_upper - bins_lower)
#%%
# Creating histogram of N
plt.bar(bins_mid*1e9, N, width=(bins_upper - bins_lower)*1e9, align='
center', edgecolor='black')
plt.xscale('log')
# plt.yscale('log')
plt.xlabel('Dp (nm)')
plt.ylabel('N (cm^-3)')
plt.suptitle('Histogram of N in each bin')
#save the plot
plt.savefig('1-3.png', bbox_inches='tight')
plt.show()

#
#%%
# Creating a gamma distribution of droplet sizes
# Total number of drops in gamma distribution is N_d = 100 cm^-3
# The mean diameter is D_pg = 20 um = 20000 nm
# Dp_mean_2 = 20000 # nm
# gamma = 1
# Beta = 2

N_2 = 100 # cm^-3 (total number of droplets)
rp_mean_2 = 20e-6 # m (mean diameter of droplets)
Dp_2 = np.geomspace(1e-6, 1e-3, 1000) # convert to nm
B_2 =  3 / rp_mean_2 # r_mean = 3/B_2
A_2 = N_2 * B_2**3 / 2 # N_2 = 2 * A_2 / B_2**3
dN2_dr = A_2 * ((Dp_2 / 2 )**2) * np.exp( -B_2 * (Dp_2 / 2) ) #
dN2_dr = A_2 * r^2 * exp(-B_2 * r)
dN2_dDp = dN2_dr / 2 # dN2_dDp = dN2_dr / 2
dN2_dlnDp = dN2_dDp * Dp_2
dN2_dlogDp = dN2_dDp * Dp_2 * np.log(10)
#%%
# Creating subplots
fig, axs = plt.subplots(3, 1, figsize=(10, 8))

# First plot
axs[0].plot(Dp_2*1e9, dN2_dDp) # convert to nm
axs[0].set_xscale('log')
axs[0].set_xlabel('Dp (nm)')
axs[0].set_ylabel('dN/dDp (cm^-3)')
axs[0].legend(['dN/dDp'])
```

```

# Second plot
axs[1].plot(Dp_2*1e9, dN2_dlnDp) # convert to nm
axs[1].set_xscale('log')
axs[1].set_xlabel('Dp (nm)')
axs[1].set_ylabel('dN/dlnDp')
axs[1].legend(['dN/dlnDp'])

# Third plot
axs[2].plot(Dp_2*1e9, dN2_dlogDp) # convert to nm
axs[2].set_xscale('log')
axs[2].set_xlabel('Dp (nm)')
axs[2].set_ylabel('dN/dlogDp')
axs[2].legend(['dN/dlogDp'])
plt.savefig('1-2.png', bbox_inches='tight')
plt.show()
```

```python
#%%
#importing libraries
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
from pyfluids import Fluid, FluidsList, Input

mpl.rcParams["figure.dpi"] = 100
#%%
# Default air standard properties
pressure_std = 101325 # Pa
temperature_std = 273.15 + 25 # K #??????? need to make sure this is the right temperature
air_std = Fluid(FluidsList.Air).with_state(Input.pressure(pressure_std), Input.temperature(temperature_std-273.15))

#defining the function
#Cunningham correction factor
#Dp is the particle diameter in meters
#lamda is the mean free path of the gas in meters
#C is the Cunningham correction factor
def c_cunningham(Dp, lamda = 65E-9):
    kn = 2 * lamda / Dp
    return 1 + kn * (1.257 + 0.4 * np.exp(-1.1 / kn))

# mean free path of air calculator
#T is the temperature in Kelvin
#P is the pressure in Pascals
#lamda is the mean free path of the gas in meters
def mean_free_path(temperature, pressure):
    R = 8.314 # J/(mol K) gas constant
    M = 0.0289647 # kg/mol molar mass of air
    air = Fluid(FluidsList.Air).with_state(Input.pressure(pressure), Input.temperature(temperature - 273.15))
    viscosity = air.dynamic_viscosity # Pa s dynamic viscosity
    return 2 * viscosity / (pressure * np.sqrt(8 * M / (np.pi * R * temperature)))

#Reynolds number calculator
#Dp is the particle diameter in meters
#rho_f is the density of the fluid in kg/m^3
#g is the acceleration due to gravity in m/s^2
#C is the Cunningham correction factor
#Re is the Reynolds number
def reynolds_number(Dp, velocity, fluid_density = air_std.density, dynamic_viscosity = air_std.dynamic_viscosity):
    return (Dp * fluid_density * velocity) / dynamic_viscosity


#%%
# print values for air at 25C and 101325 Pa
print(air_std.dynamic_viscosity, air_std.density, mean_free_path(temperature_std, pressure_std))
```

```python
#%%
def settling_velocity(Dp_input, rho_p, temperature, pressure):
    # Check if Dp_input is an array or a single value
    if np.isscalar(Dp_input):
        Dp_array = np.array([Dp_input])  # Convert to array for uniform processing
    else:
        Dp_array = Dp_input  # Use the array as is

    velocities = []  # Empty list to store calculated velocities
    for Dp in Dp_array:  # Process each Dp individually
        g = 9.81  # m/s^2
        l_mfp = mean_free_path(temperature, pressure)
        c_cun = c_cunningham(Dp, l_mfp)
        air = Fluid(FluidsList.Air).with_state(Input.pressure(pressure), Input.temperature(temperature - 273.15))
        mu_f = air.dynamic_viscosity
        rho_f = air.density
        s_velocity = c_cun * (rho_p * g * Dp**2) / (18 * mu_f)  # Stokes settling velocity
        Re = reynolds_number(Dp, s_velocity, fluid_density=rho_f, dynamic_viscosity=mu_f)
        if Re < 1:
            velocities.append(s_velocity)
        else:
            # Adjusted iterative approach for Re > 1, similar to before
            m_p = np.pi * rho_p * Dp**3 / 6
            for i in range(100):
                # c_d = 24 / Re * (1 + 0.15 * Re**(0.687))  # Updated drag coefficient expression
                c_d = 24 / Re * (1 + 3/16 * 0.43 * Re)
                # s_velocity = np.sqrt((4 * m_p * g) / (3 * np.pi * c_d * rho_f * Dp**2))
                s_velocity = np.sqrt((m_p * g) / (1/8 * np.pi * c_d * rho_f * Dp**2))
                Re_new = reynolds_number(Dp, s_velocity, fluid_density=rho_f, dynamic_viscosity=mu_f)
                if abs(Re_new - Re) < 0.01:
                    break  # Exit the loop if the change in Reynolds number is small enough
                else:
                    Re = Re_new
            velocities.append(s_velocity)

    velocities_array = np.array(velocities)  # Convert list to array

    if np.isscalar(Dp_input):
        return velocities_array[0]  # Return a single value if input was scalar
    else:
        return velocities_array  # Return array if input was array
#%%
```

```
# Homwework 1-2
Dp = np.geomspace(100E-9, 1E-3, 10000)  # Array of particle diameters from 1 nm to 1 micron
velocities = settling_velocity(Dp, 1000, 273, 101325)  # Assuming room temperature is 25°C in Kelvin
plt.plot(Dp, velocities, label='Code Output')
plt.xscale('log')
plt.yscale('log')
plt.xlabel('Particle Diameter (m)')
plt.ylabel('Settling Velocity (m/s)')
plt.suptitle('Settling Velocity vs Particle Diameter')
C_actual = np.array([(0.1E-6,1E-6,10E-6, 100E-6), (8.82E-7,3.48E-5, 3.06E-3, 2.40E-1)])
plt.scatter(C_actual[0,:], C_actual[1,:], color='red', label='Hinds - Appendix A11')
plt.legend()
plt.grid()
plt.savefig('2-1.png', bbox_inches='tight')
plt.show()
#%%
#testing the function Cunningham correction factor
Dp = np.geomspace(1E-9,1E-6,1000) # 1 micron particle
lamda = mean_free_path(290, 101325)
c_cunningham(Dp, lamda)
plt.plot(Dp, c_cunningham(Dp, lamda), label='Code output')
plt.xscale('log')
plt.xlabel('Particle diameter (m)')
plt.ylabel('Cunningham correction factor')
plt.title('Cunningham correction factor vs particle diameter')
# add point to the plot in an array
C_actual = np.array([(1E-6,1E-7,4E-8, 1E-9, 1E-8), (1.2,3, 6, 224, 22.97)])
plt.scatter(C_actual[0,:], C_actual[1,:], color='red', label='Hinds')
plt.legend()
plt.savefig('2-2.png', bbox_inches='tight')
plt.show()
#%%

```