ENGG*4810: Control of Atmospheric Particulates

Introduction to MATLAB Programming

Amir A. Aliabadi

November 13, 2017

1 Introduction

MATLAB is a programming language developed by MathWorks. It is based on performing operations on matrices. This is the world's most natural way to express computational mathematics. Figure 1 shows the MATLAB environment consisting of various windows: current directory, workspace, command history, script editor, and command window.

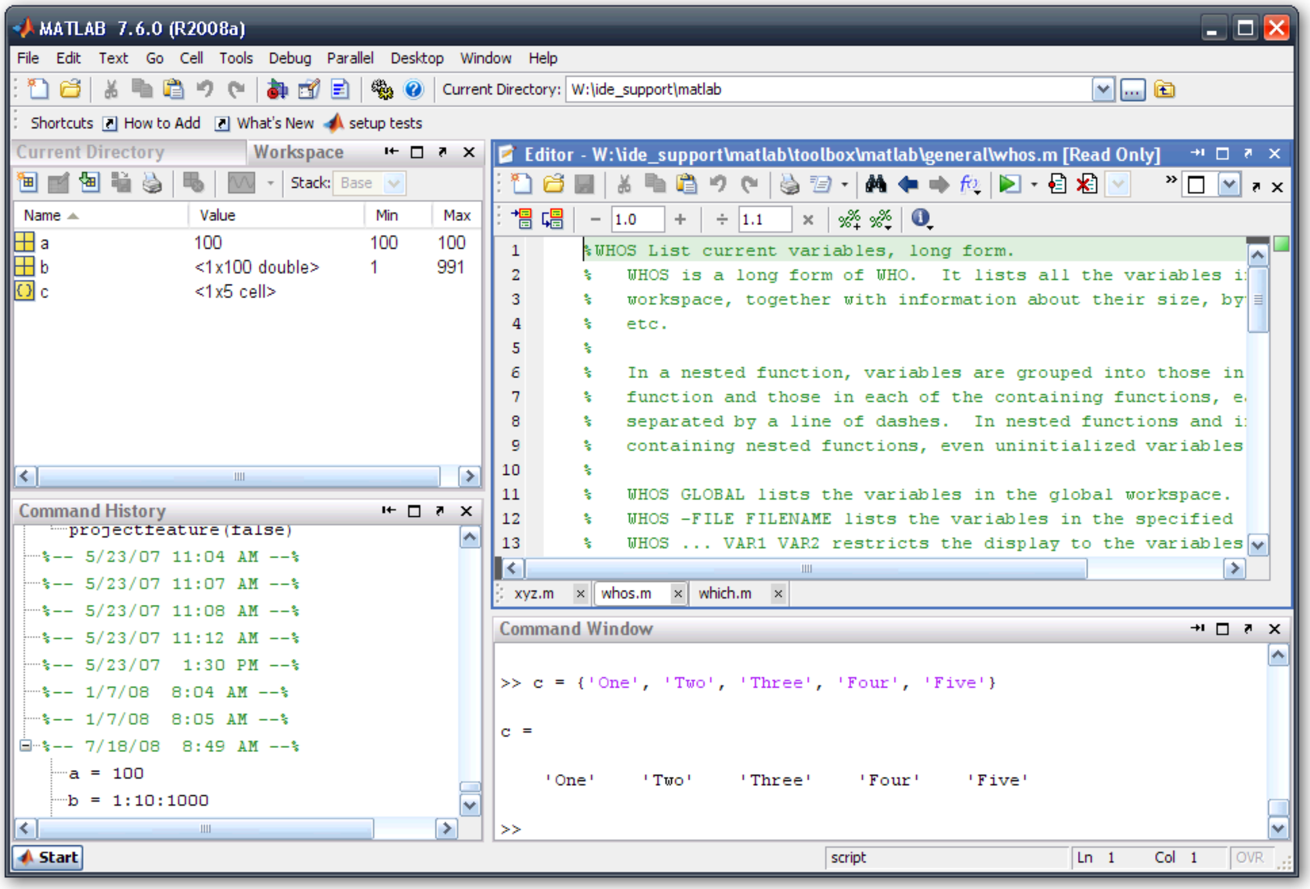
The current directory shows the local files and where MATLAB looks for other files by default if not given another path. The workspace shows the variables that are created through an analysis. These could be single integers, arrays, matrices, characters, strings, data structures, and more. The command window is where you execute single commands, very much like Windows command prompt or Mac's or Linux' terminal. The command history shows you the commands that have been executed before. The editor is where you can script, code, program, or simply run a succession of commands.

IMPORTANT NOTE: Save your work only on your designated H: drive on the server or a memory stick. DO NOT save your work on the local machine in the lab. The local storage on the machine is erased every time the computer is rebooted.

2 Using the Command Window

Type the following commands in the command window in order to calculate volume V_p [m^3] and mass m_p [kg] of a particle sphere with diameter $D_p = 1$ μm and unit density $\rho_p = 1000$ kg m^{-3}:

```
>>Dp=1e-6
Dp=
1.0000e-06
>>rhop=1000
rhop=
```


MATLAB 7.6.0 (R2008a)
File Edit Text Go Cell Tools Debug Parallel Desktop Window Help
Current Directory: W:\ide_support\matlab
Shortcuts How to Add What's New setup tests
Current Directory Workspace
Stack: Base
Command History
Editor - W:\ide_support\matlab\toolbox\matlab\general\whos.m [Read Only]
xyz.m whos.m which.m
Command Window
Start
script Ln 1 Col 1 OVR

Name	Value	Min	Max
a	100	100	100
b	<1x100 double>	1	991
c	<1x5 cell>		

```
projectfeature(false)
%-- 5/23/07 11:04 AM --%
%-- 5/23/07 11:07 AM --%
%-- 5/23/07 11:08 AM --%
%-- 5/23/07 11:12 AM --%
%-- 5/23/07  1:30 PM --%
%-- 1/7/08  8:04 AM --%
%-- 1/7/08  8:05 AM --%
%-- 7/18/08  8:49 AM --%
  a = 100
  b = 1:10:1000
```

```
%WHOS List current variables, long form.
%   WHOS is a long form of WHO.  It lists all the variables i
%   workspace, together with information about their size, by
%   etc.
%
%   In a nested function, variables are grouped into those in
%   function and those in each of the containing functions, e
%   separated by a line of dashes.  In nested functions and i
%   containing nested functions, even uninitialized variables
%
%   WHOS GLOBAL lists the variables in the global workspace.
%   WHOS -FILE FILENAME lists the variables in the specified
%   WHOS ... VAR1 VAR2 restricts the display to the variables
```

```
>> c = {'One', 'Two', 'Three', 'Four', 'Five'}

c =

    'One'    'Two'    'Three'    'Four'    'Five'

>>
```

Figure 1: The MATLAB 7.6.0 environment consisting of current directory, workspace, command history, script editor, and command window.

```
1000
>>Vp=(pi/6)*Dp^3
Vp=
5.2360e-19
>>mp=rhop*Vp
mp=
5.2360e-16
```

Notice that MATLAB echoes the variables in the command prompt. You can turn off the echo by placing a semicolon (;) after each command. Also notice that all variables

```
Dp, mp, rhop, Vp
```

now appear in the workspace with specific values. You can clear the command line and then delete these variables from memory (and therefore workspace) using the following commands in the command window:

```
>>clc
>>clear
```

We now perform the same calculation for $D_p = 1, 2, 3, 4, 5$ μm simultaneously using vector operation:

```
>>Dp=1e-6*[1 2 3 4 5];
>>rhop=1000;
>>Vp=(pi/6)*Dp.^3
Vp=
1.0e-16 *
0.0052    0.0419    0.1414    0.3351    0.6545
>>mp=rhop*Vp
mp=
1.0e-13 *
0.0052    0.0419    0.1414    0.3351    0.6545
```

where now `Dp, Vp, mp` are row vectors. The operation `Dp.∧3` tells MATLAB to take all elements of the vector and then raise each element to the power of three and store the result in a new vector. If you type in `Dp∧3` the compiler complains because it attempts to perform matrix algebra on a row vector and multiply it by itself twice. We know that from dimensional consideration this is not possible for a row vector. Note that multiplying a scalar `rhop` with a vector `Vp` is perfectly allowed in matrix algebra, so it is not required to use the `.*` for this operation. However, if one desires to multiply two matrices, element by element, this operator can be used.

3 Scripting with MATLAB

Next we use the MATLAB editor environment to write a script for the following particle kinematics problem: Consider a solid particle that is thrown from surface with initial velocity v_0 [m s^{-1}] with angle α [rad] relative to the horizon (x) to follow a two-dimensional projectile only under the force of gravity ($g = 9.81$ m s^{-2}) acting in the $-z$ direction. The particle is not surrounded by air, i.e. in vacuum, so there is no drag force. We wish to calculate and plot the particle's projectile for $t = 0$ to 20 s for the following initial conditions: a) $v_0 = 100$ m s^{-1}, $\alpha = \frac{\pi}{6}$, b) $v_0 = 150$ m s^{-1}, $\alpha = \frac{\pi}{4}$, c) $v_0 = 200$ m s^{-1}, $\alpha = \frac{\pi}{3}$.

The following equations govern the motion of the particle in the two directions x and y:

$$v_{0x} = v_0 \cos(\alpha) \tag{1}$$

$$v_{0z} = v_0 \sin(\alpha) \tag{2}$$

$$x = x_0 + v_{0x} t \tag{3}$$

$$z = z_0 + v_{0z} t - \frac{1}{2} g t^2 \tag{4}$$

Type the following script in the MATLAB editor. Comments are placed after %. Alternative to defining vectors by specific elements in [], one can use the syntax `start:step:finish` or `zeros(1,n+1)` to define a vector. In the former the elements will increment by the step size. In the latter the elements are all initialized to zero. The `for` loop advances the index of the vector, and if the increment of a loop is one then it can be omitted from the syntax. We also use an `if` statement to make sure the trajectory does not become negative in the z direction. We simply stop the particle as soon as z becomes negative.

```
%MATLABProgramming
%Solid particle projectile in vacuum

%Clear command window and memory
clc
clear

%Constants of simulation
g=9.81;         %Gravitational acceleration [m s^-2]
n=50;           %The number of time steps
alpha1=pi/6;    %Throw angle [rad]
v01=100;        %Initial velocity [m s^-1]

%Initial velocities
v0x1=v01*cos(alpha1);
v0z1=v01*sin(alpha1);

%Vectors
t=0:20/n:20;         %Time vector from 0 to 20 [s]
x1=zeros(1,n+1);     %Initialize x as a zero vector [m]
z1=zeros(1,n+1);     %Initialize z as a zero vector [m]

%Loop through the time steps and update x and z coordinates
for i=2:n+1
   x1(i)=v0x1*t(i);
   z1(i)=v0z1*t(i)-0.5*g*(t(i))^2;

   %Must check if the particle reaches the ground
   if (z1(i) < 0)
       %Set altitude to zero
       z1(i)=0;
       %Do not advance the x position
       x1(i)=x1(i-1);
   end
end
```

After writing the script save the file in your local directory as `MATLABProgramming.m`. Then execute the script by clicking on the run (play) button. The series of the commands will be executed, as if they were written in the command window. If you code runs you see that all the variables are generated in the work space. Check them out by double clicking on each variable to see its value. If your script does not run you should debug it first.

4 Plotting with MATLAB

We next plot the trajectory using MATLAB's plot command. Type the script below into the editor after the previous script:

```
figure
plot(x1,z1,'ko');
xlabel('x [m]');
ylabel('z [m]');
```

Command `figure` tells MATLAB to start a new window for this plot. The specifier `'ko'` tells MATLAB that we want to show this plot by black empty circles. The `xlabel` and `ylabel` command simple enable us to label each axis. If everything goes well you should get figure 2. You can save this figure using the figure menu.

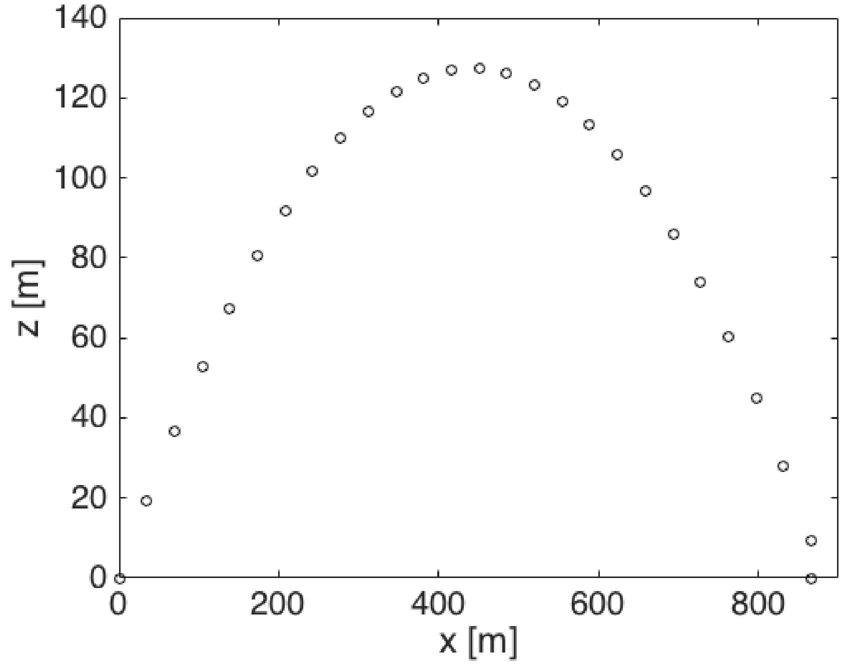


Figure 2: Trajectory of a single particle for initial conditions given in a)

We now continue with other initial conditions b) and c). Simply copy and paste the script into itself and create the vectors for the other initial conditions. To plot multiple curves on the same figure you can modify the plot command according to the following script. You can also add legends, increased marker size, use different colors, and use different font sizes for the legend and axes labels:

```
figure
plot(x1,z1,'ko','MarkerSize',10);
hold on
plot(x2,z2,'bd','MarkerSize',10);
plot(x3,z3,'rs','MarkerSize',10);
xlabel('x [m]','FontSize',20);
ylabel('z [m]','FontSize',20);
h_legend=legend('a)','b)', 'c)');
set(h_legend,'FontSize',20);
```

If everything goes well you should get figure 3.

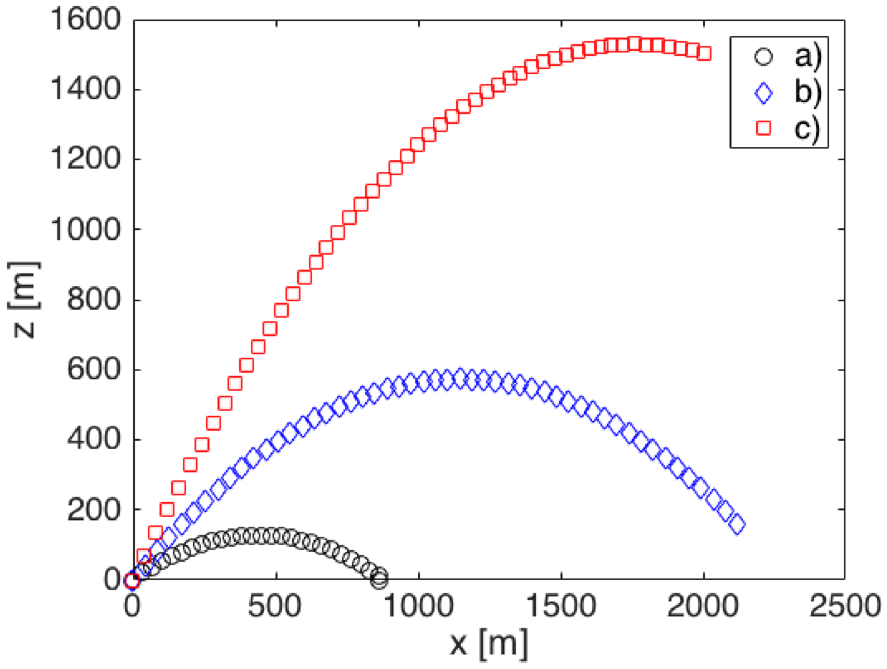


Figure 3: Multiple trajectories of a single particle for various initial conditions given in a), b), and c) plotted using markers.

You can plot vectors using lines instead of markers, or even both. Append your script with the following code and run it again. You should get the figure 4.

```
figure
plot(x1,z1,'g-','LineWidth',6);
hold on
plot(x2,z2,'c--','LineWidth',4);
plot(x3,z3,'m^:','MarkerSize',10,'LineWidth',2);
xlabel('x [m]','FontSize',20);
ylabel('z [m]','FontSize',20);
h_legend=legend('a)','b)', 'c)');
set(h_legend,'FontSize',20);
```

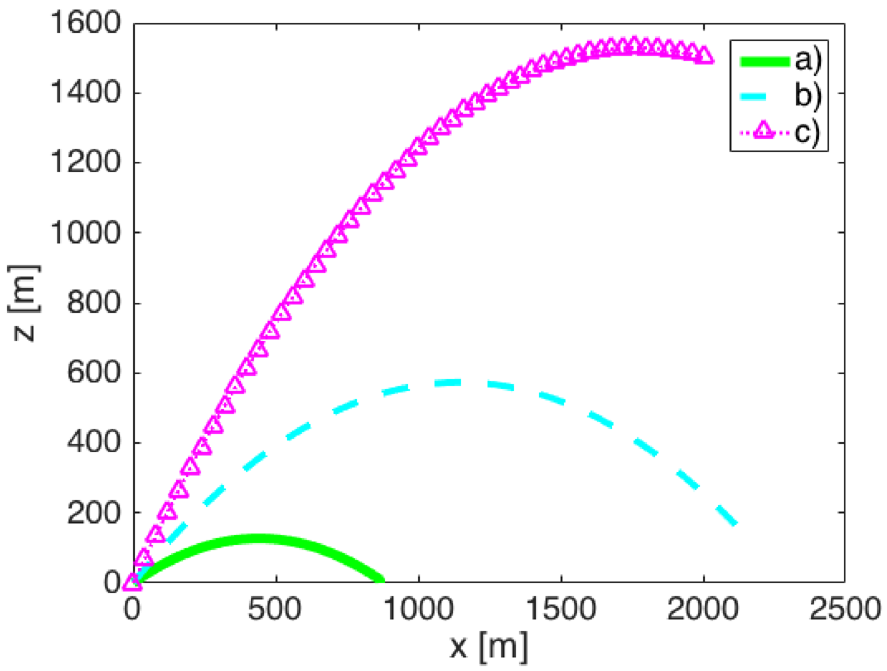


Figure 4: Multiple trajectories of a single particle for various initial conditions given in a), b), and c) plotted using lines or a combination of lines and markers.

5 Writing and Reading Text Files with MATLAB

We now attempt to write the simulation results into a text file. Append the following script in your editor:

```
%Create a file name and assign an ID to open the file for writing
fileName = 'MATLABProgramming.txt';
fileid=fopen(fileName,'w');

%Write the header for the file that consists of time and the coordinates
fprintf(fileid, 'time\t x1\t z1\t x2\t z2\t x3\t z3\n');

for i = 1:n+1
    %Write in the file the data for each time step
    fprintf(fileid, '%f\t %f\t %f\t %f\t %f\t %f\t %f\n',...
        t(i), x1(i), z1(i), x2(i), z2(i), x3(i), z3(i));
end

%Close the file
fclose(fileid);
```

This will create a new file `MATLABProgramming.txt` and adds the results into it. The `fopen` and `fclose` commands open a file `id` and the specifier `'w'` tells MATLAB that you are opening this file to write in it. The command `fprintf` is used to print specific strings or variables into the file. Specifiers \`t` and \`n` represent the tab and new line. Specifier `%f` tells MATLAB that you will write a floating point variable in the text file. If you open the text file you should see this:

```
time [m]  x1 [m]  z1 [m]  x2 [m]  z2 [m]  x3 [m]  z3 [m]
0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
0.400000  34.641016  19.215200  42.426407  41.641607  40.000000  68.497232
0.800000  69.282032  36.860800  84.852814  81.713614  80.000000  135.424865
1.200000  103.923048  52.936800  127.279221  120.216021  120.000000  200.782897
1.600000  138.564065  67.443200  169.705627  157.148827  160.000000  264.571329
2.000000  173.205081  80.380000  212.132034  192.512034  200.000000  326.790162
2.400000  207.846097  91.747200  254.558441  226.305641  240.000000  387.439394
2.800000  242.487113  101.544800  296.984848  258.529648  280.000000  446.519026
3.200000  277.128129  109.772800  339.411255  289.184055  320.000000  504.029058
3.600000  311.769145  116.431200  381.837662  318.268862  360.000000  559.969491
4.000000  346.410162  121.520000  424.264069  345.784069  400.000000  614.340323
4.400000  381.051178  125.039200  466.690476  371.729676  440.000000  667.141555
4.800000  415.692194  126.988800  509.116882  396.105682  480.000000  718.373188
5.200000  450.333210  127.368800  551.543289  418.912089  520.000000  768.035220
5.600000  484.974226  126.179200  593.969696  440.148896  560.000000  816.127652
6.000000  519.615242  123.420000  636.396103  459.816103  600.000000  862.650485
6.400000  554.256258  119.091200  678.822510  477.913710  640.000000  907.603717
6.800000  588.897275  113.192800  721.248917  494.441717  680.000000  950.987349
7.200000  623.538291  105.724800  763.675324  509.400124  720.000000  992.801381
7.600000  658.179307  96.687200  806.101731  522.788931  760.000000  1033.045814
8.000000  692.820323  86.080000  848.528137  534.608137  800.000000  1071.720646
8.400000  727.461339  73.903200  890.954544  544.857744  840.000000  1108.825878
8.800000  762.102355  60.156800  933.380951  553.537751  880.000000  1144.361511
9.200000  796.743371  44.840800  975.807358  560.648158  920.000000  1178.327543
9.600000  831.384388  27.955200  1018.233765  566.188965  960.000000  1210.723975
10.000000  866.025404  9.500000  1060.660172  570.160172  1000.000000  1241.550808
10.400000  866.025404  0.000000  1103.086579  572.561779  1040.000000  1270.808040
10.800000  866.025404  0.000000  1145.512986  573.393786  1080.000000  1298.495672
11.200000  866.025404  0.000000  1187.939392  572.656192  1120.000000  1324.613704
11.600000  866.025404  0.000000  1230.365799  570.348999  1160.000000  1349.162137
12.000000  866.025404  0.000000  1272.792206  566.472206  1200.000000  1372.140969
12.400000  866.025404  0.000000  1315.218613  561.025813  1240.000000  1393.550201
12.800000  866.025404  0.000000  1357.645020  554.009820  1280.000000  1413.389834
13.200000  866.025404  0.000000  1400.071427  545.424227  1320.000000  1431.659866
13.600000  866.025404  0.000000  1442.497834  535.269034  1360.000000  1448.360298
14.000000  866.025404  0.000000  1484.924240  523.544240  1400.000000  1463.491131
14.400000  866.025404  0.000000  1527.350647  510.249847  1440.000000  1477.052363
14.800000  866.025404  0.000000  1569.777054  495.385854  1480.000000  1489.043995
15.200000  866.025404  0.000000  1612.203461  478.952261  1520.000000  1499.466028
15.600000  866.025404  0.000000  1654.629868  460.949068  1560.000000  1508.318460
16.000000  866.025404  0.000000  1697.056275  441.376275  1600.000000  1515.601292
```

```
16.400000  866.025404  0.000000  1739.482682  420.233882  1640.000000  1521.314524
16.800000  866.025404  0.000000  1781.909089  397.521889  1680.000000  1525.458157
17.200000  866.025404  0.000000  1824.335495  373.240295  1720.000000  1528.032189
17.600000  866.025404  0.000000  1866.761902  347.389102  1760.000000  1529.036621
18.000000  866.025404  0.000000  1909.188309  319.968309  1800.000000  1528.471454
18.400000  866.025404  0.000000  1951.614716  290.977916  1840.000000  1526.336686
18.800000  866.025404  0.000000  1994.041123  260.417923  1880.000000  1522.632318
19.200000  866.025404  0.000000  2036.467530  228.288330  1920.000000  1517.358351
19.600000  866.025404  0.000000  2078.893937  194.589137  1960.000000  1510.514783
20.000000  866.025404  0.000000  2121.320344  159.320344  2000.000000  1502.101615
```

We now attempt to read the text file, partially, that we just created. Append the following script in your editor. The `textread` command is suited to read entire column of a text file into a vector. On the left hand side of the equal sign, the names for new vectors to be created appear in square brackets `[]`. As argument, the command takes the name of the text file, then the format specifiers are listed. Note that if an specifier is listed with an asterisk `%*f`, then that column is skipped. In this example we only read the coordinates for the second projectile, i.e. `x2` and `z2`. The specifier `'headerlines',1` tells MATLAB that the first line of the text file is just the header and must be skipped:

```
%Read elements of a text file, selectively, into three vectors
[tread, x2read, z2read] = textread('MATLABProgramming.txt',...
 '%f %*f %*f %f %f %*f %*f', 'headerlines',1);
```

Make sure that these new variables are created in the workspace. Double click on these variables to see their contents. Congratulations! you have just finished your first computer lab in programming with MATLAB.

ENGG*4810: Control of Atmospheric Particulates

Simulation of Particle Projectiles under
Gravity, Buoyancy, Drag, and Brownian Forces

Amir A. Aliabadi

November 13, 2017

1 Introduction

In this lab we wish to simulate the projectile of a particle under gravity, buoyancy, drag, and Brownian forces in stagnant air in a two-dimensional domain. Here we briefly provide the relevant equations from the lecture notes, but ask that you consult the lecture notes in detail if required.

The particle Reynolds number is an important non-dimensional parameter to calculate in order to compute the drag coefficient.

$$\mathrm{Re} = \frac{\text{Inertial forces}}{\text{Viscous forces}} = \frac{\rho u_\infty D_p}{\mu} = \frac{u_\infty D_p}{\nu} \tag{1}$$

If Re is known the drag force and coefficient on the particle can be calculated using

$$F_{drag} = \frac{C_D A_p \rho u_\infty^2}{2C_c} = \frac{\pi C_D \rho D_p^2 u_\infty^2}{8C_c} \text{ (Spherical particle)} \tag{2}$$

$$C_D = \begin{cases} \frac{24}{\mathrm{Re}} \rightarrow \text{Stokes law} & \mathrm{Re} < 0.1 \\ \frac{24}{Re}\left(1 + \frac{3}{16}\mathrm{Re} + \frac{9}{160}\mathrm{Re}^2 \ln(2\mathrm{Re})\right) & 0.1 < \mathrm{Re} < 2 \\ \frac{24}{Re}\left(1 + 0.15\mathrm{Re}^{0.687}\right) & 2 < \mathrm{Re} < 500 \\ 0.44 & 500 < \mathrm{Re} < 2 \times 10^5 \end{cases}$$

In order to properly calculate the drag force on very small particles, we need to compute the air mean free path, Knudsen number, and therefore slip correction factor:

$$\lambda = \frac{\mu}{0.499p(8M/\pi RT)^{1/2}} \tag{3}$$

$$\mathrm{Kn} = \frac{2\lambda}{D_p} \tag{4}$$

$$C_c = 1 + \mathrm{Kn}\left[1.257 + 0.40\exp\left(-\frac{1.10}{\mathrm{Kn}}\right)\right] \tag{5}$$

We can simulate the instantaneous Brownian acceleration of a particle, in any given coordinate x, y, or z using the methodology of [Li and Ahmadi, 1992] and [Ounis et al., 1991], where the acceleration is given by:

$$S_0 = \frac{216\mu kT}{\pi^2 D_p^5 \rho_p^2 C_c} \tag{6}$$

$$\alpha_{x,y,z}(t) = \delta\sqrt{\frac{\pi S_0}{dt}} \tag{7}$$

where k is the Boltzmann constant, δ is a random variable sampled from a unit normal or Gaussian distribution, and dt is the time step of the particle tracking simulation. Note that the simulation of the Brownian diffusion of particles is very sensitive to the choice of dt. In fact this time step must be selected appropriately to simulate Brownian diffusion correctly. Nevertheless, for the purpose of this lab, a value of dt will be used to show relative significance of the Brownian acceleration in comparison to drag, gravity, and buoyancy.

Having these terms and assuming that the vertical coordinate is positive upward, i.e. $\uparrow +z$, and that the particle is ejected in the positive direction in the horizon, i.e. $\rightarrow +x$, then the equation of motion for the particle is given as

$$m_p\frac{dv_z}{dt} = \underbrace{-\frac{\pi C_D \rho D_p^2}{8C_c}|v_z|(v_z)}_{\text{Drag force}} \quad \underbrace{-m_p g}_{\text{Gravity force}} \quad \underbrace{+\left(\frac{\pi}{6}\right)D_p^3\rho g}_{\text{Buoyancy Force}} \quad + \quad \underbrace{m_p\alpha_z(t)}_{\text{Brownian force}} \tag{8}$$

$$m_p\frac{dv_x}{dt} = \underbrace{-\frac{\pi C_D \rho D_p^2}{8C_c}|v_x|(v_x)}_{\text{Drag force}} + \quad \underbrace{m_p\alpha_x(t)}_{\text{Brownian force}} \tag{9}$$

2 Simulation Matrix

We assume that the initial position of the particle is $x = z = 0$ and that the particle is ejected upward at an angle $\alpha = \pi/4$ from x axis with an initial velocity of $\mathbf{v}(t = 0) = 0.01$ m s^{-1}. The particle density is $\rho_p = 1000$ kg m^{-3}. The gravitational acceleration is $g = 9.81$ m s^{-2} downward. The air pressure is atmospheric, i.e. $p = 101,000$ Pa. The molecular weigh of air is $M = 28.966$ g mol^{-1}. The universal gas constant is $R = 8.314$ J K^{-1} mol^{-1}. The Boltzmann constant is $k = 1.38 \times 10^{-23}$ J K^{-1}. The spatial domain is infinite and we wish to simulate particle motion for 100,000 time advances with time step $dt = 0.1\tau$, where τ is particle's characteristic time. We wish to study the effects of particle size and physical properties of air, i.e. temperature, pressure, and viscosity, on the trajectory of the particle. We develop the following simulation matrix. The particle size range represents typical airborne particle emissions. The air properties are similar to ambient conditions and conditions in hot exhaust gas from combustion processes.

Table 1: Simulation matrix

Simulation	1	2	3	4	5	6	7	8
D_p [m]	1e–7	1e–6	1e–5	1e–4	1e–7	1e–6	1e–5	1e–4
T [K]	300	300	300	300	600	600	600	600
μ [kg m^{-1} s^{-1}] $\times 10^{-5}$	1.846	1.846	1.846	1.846	3.017	3.017	3.017	3.017
ρ [kg m^{-3}]	1.777	1.777	1.777	1.777	0.5883	0.5883	0.5883	0.5883

Since the first terms on the right hand side of the equation of particle motion varies with particle velocity. We cannot integrate the equation to arrive at an analytical expression for particle trajectory. Instead, we have to use the finite difference method to approximate the particle trajectory. In this method a time step is chosen that is a fraction (usually 10%) of the particle characteristic time and the particle motion is advanced using the velocity and position information in the previous time step. The particle characteristic time is given by

$$\tau = \frac{\rho_p D_p^2 C_c}{18\mu} \tag{10}$$

3 MATLAB Script

Begin your script by clearing command window and workspace and then defining all constants required for the simulation:

```
%ParticleProjectile
%Particle projectile under forces of gravity, buoyancy, drag, and Brownian
%motion

%Clear command window and memory
clc
clear

%Constants of simulation
g=9.81;         %Gravitational acceleration [m s^-2]
alpha=pi/4;     %Throw angle [rad]
```

```
v0=0.01;         %Initial velocity [m s^-1]
p=101000;        %Air pressure [Pa]
R=8.314;         %Universal gas constant [J K^-1 mol^-1]
M=28.966e-3;     %Air molecular weight [kg mol^-1]
rhop=1000;       %Particle density [kg m^-3]
k=1.38e-23;      %Boltzmann constant [J K^-1]
Dp=1e-7;         %Particle diameter [m]
T=300;           %Air temperature [K]
mu=1.846e-5;     %Air dynamic viscosity [kg m^-1 s^-1]
rho=1.777;       %Air density [kg m^-3]
n=100000;        %Number of time advances
```

Then calculate initial velocity in the x and z direction, followed by mean free path `lambda`, Knudsen number `Kn`, slip correction factor `Cc`, and particle mass `mp`. The calculation of the Knudsen number, slip correction, and particle mass are left for you to add in the script. In general, you should code in the script below whenever you see (...). Be careful not to miss them:

```
%Initial velocities
v0x=v0*cos(alpha);
v0z=v0*sin(alpha);

%Calculate mean free path, Knudsen number, slip correction, and mass
lambda=mu/(0.499*p*sqrt(8*M/(pi*R*T)));
Kn=...
Cc=...
mp=...
```

Next we need to calculate the simulation timestep `dt` as a fraction (10%) of the particle characteristic time `tau`. Again, these two are left for you to script:

```
%Calculate particle characteristic time
tau=...
%Set time step as 10% of the smallest particle characteristic time
dt=...
```

We subsequently define the position vectors and initialize them as zero. The initial velocities also need to be defined.

```
%Define position vectors and initialize to zero
x=zeros(1,n);     %x Position vector [m]
z=zeros(1,n);     %z Position vector [m]

%Initialize the velocity
vxold=v0x;
vzold=...
```

Next we loop through the time steps, finding the instantaneous acceleration followed by updating particle velocities and position. At each iteration we reset the accelerations `ax` and `az` to zero. Then we add corresponding accelerations for each direction due to drag, gravity, buoyancy, and Brownian motion, respectively. The Reynolds number of the particle `Re` is calculate at each iteration given the particle velocity in the previous time step. Subsequently, the coefficient of drag `CD` is calculated. The Brownian acceleration is added by using the `randn(1)` function, which generates a random variable drawn from a Gaussian distribution with unit standard deviation and the `rand` function, which generates a random variable from 0 to 1 drawn from a uniform distribution.

```
%Loop through the time steps and update particle position and velocity
for (i=2:n)
    %Reset acceleration terms
    ax=0.0;
    az=0.0;
    %Calculate particle's Reynolds number (need total velocity)
    Re=(rho*Dp*sqrt(vxold^2+vzold^2))/mu;
    %Calculate the coefficient of drag based on this Reynolds number
    if (Re < 0.1)
        CD=24/Re;
    elseif (Re < 2)
        CD=(24/Re)*(1+3*Re/16+9*Re^2*log(2*Re)/160);
    elseif (Re < 500)
        CD=(24/Re)*(1+0.15*Re^0.687);
    elseif (Re < 2e5)
        CD=0.44;
    end
    %Append accelerations by drag accelerations
    ax=ax-(1/mp)*pi*CD*rho*Dp^2*vxold*abs(vxold)/(8*Cc);
    az=az-(1/mp)*pi*CD*rho*Dp^2*vzold*abs(vzold)/(8*Cc);
    %Append accelerations by gravity force
    az=...
    %Append accelerations by buoyancy force
    az=...
    %Append accelerations by Brownian force
    S0=(216*mu*k*T)/((pi^2*Dp^5*rhop^2)*Cc);
    ax=ax+randn(1)*((pi*S0/dt)^0.5);
    az=az+randn(1)*((pi*S0/dt)^0.5);
    %Find new velocities
    vx=vxold+ax*dt;
    vz=...
    %Find new position
    x(i)=x(i-1)+vx*dt;
    z(i)=...
    %Update old velocities for next iteration
    vxold=vx;
    vzold=...
```

```
end
```

Finally you should write script to plot the coordinates `z [m]` versus `x [m]` to show the particle trajectory.

4 Running the Script

Next you should run the script with the initial conditions provided in the simulation matrix. For each run, simply change the particle diameter `Dp` and air physical properties `T, mu, rho`. Each time you run the script, you can save the resulting plot in your local directory given a preferred format, i.e. `*.png, *.jpg, *.pdf` etc. To do this simply click `file` then `save` on the figure menu. Use a descriptive file name for each figure. If your script runs successfully you should get figures similar to the following figures. Your figures will not be exactly the same because you have a random number generator in the code, which samples from a Guassian distribution.

The figures reveal interesting and different projectiles as a function of particle size and air physical properties. Try to discuss and answer the following questions with your peers or instructor:

- For particle sizes of `Dp=1e-7` and `1e-6` m, the particle projectile appears as very *jittery* and the particle does not seem to either go up or down consistently. By observing this, which forces do you think are dominant in determining the particle motion?
- For particle size of `Dp=1e-5` m the particle projectile still appears as *jittery*, but to a lesser extent, while the particle seems to go down consistently. By observing this, which forces do you think are dominant in determining the particle motion?
- For particle size of `Dp=1e-4` m the particle projectile shows very little *jittery* movement, while the particle seems to go down significantly. By observing this, which forces do you think are dominant in determining the particle motion?
- Comparing the two figures, particles `Dp=1e-5` and `Dp=1e-4` m in size seem to drop a larger distance when `T=300` K, `mu=1.846e-5` kg m^{-1} s^{-1}, and `rho=1.777` kg m^{-3} compared to when `T=600` K, `mu=3.017e-5` kg m^{-1} s^{-1}, and `rho=0.5883` kg m^{-3}. Try to reason why this has happened by discussing the terms involved in the calculation of the drag force?

References

[Li and Ahmadi, 1992] Li, A. and Ahmadi, G. (1992). Dispersion and deposition of spherical particles from point sources in turbulent channel flow. *Aerosol Sci. Technol.*, 16:209–226.

[Ounis et al., 1991] Ounis, H., Ahmadi, G., and McLaughlin, J. B. (1991). Brownian diffusion of submicrometer particles in the viscous sublayer. *J. Colloid Interf. Sci.*, 143(1):266–277.

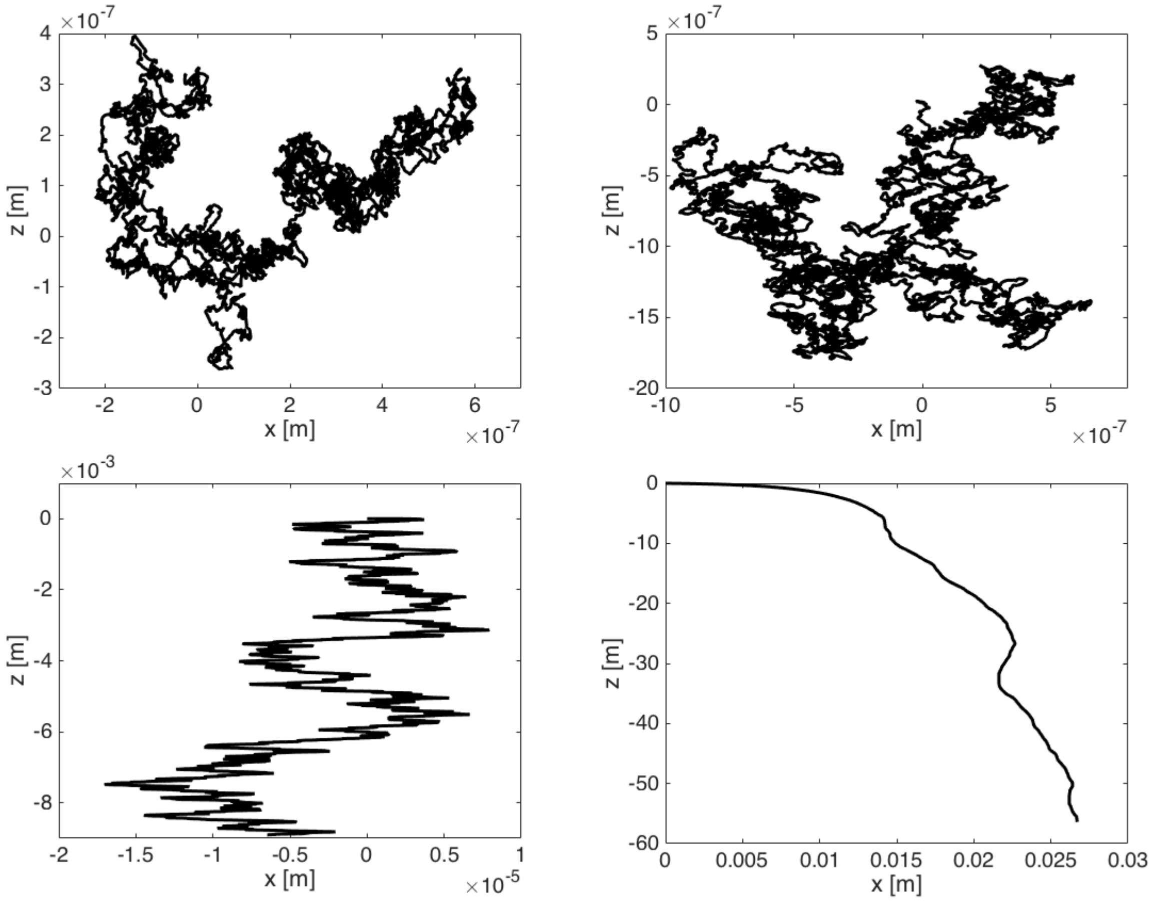


Figure 1: Trajectory of a single particle with sizes `Dp=1e-7` m (top left), `Dp=1e-6` m (top right), `Dp=1e-5` m (bottom left), `Dp=1e-4` m (bottom right) for initial conditions `T=300` K, `mu=1.846e-5` kg m^{-1} s^{-1}, and `rho=1.777` kg m^{-3}.

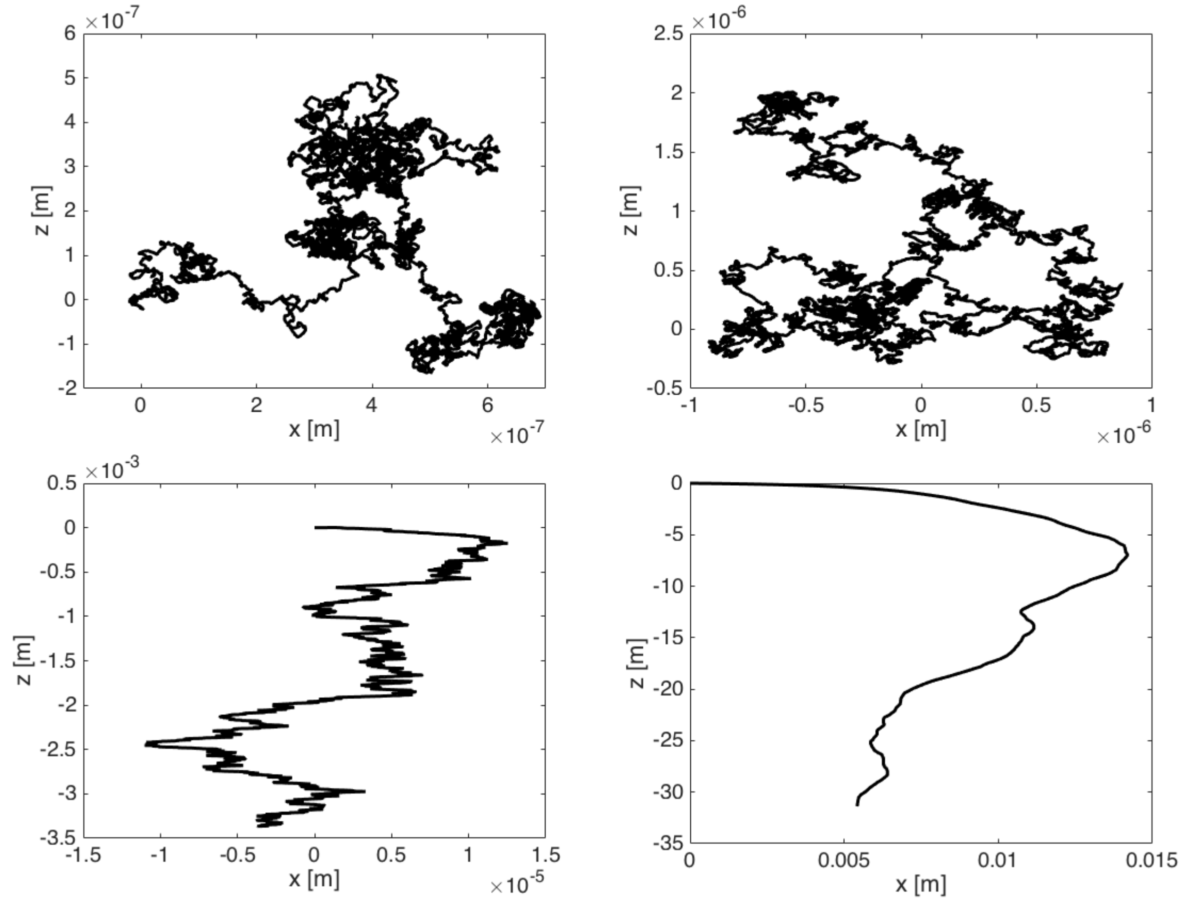


Figure 2: Trajectory of a single particle with sizes `Dp=1e-7` m (top left), `Dp=1e-6` m (top right), `Dp=1e-5` m (bottom left), `Dp=1e-4` m (bottom right) for initial conditions `T=600` K, `mu=3.017e-5` kg m^{-1} s^{-1}, and `rho=0.5883` kg m^{-3}.

ENGG*4810: Control of Atmospheric Particulates

Simulation of Particle Size Distributions

Amir A. Aliabadi

November 13, 2017

1 Introduction

In this lab we wish to simulate single mode and bimodal particle distribution functions. A crude representation of particle sizes is by histograms N, where the number concentration for given particle size bins each centred at D_p is plotted as a function of particle size D_p. A better representation is the particle size distribution n, which is obtained by dividing the particle concentration N by the width of the particle size bin ΔD_p:

$$n_i = \frac{N_i}{\Delta D_p} \text{ or } N_i = n_i \Delta D_p \tag{1}$$

In the limit of $\Delta D_p \to 0$ we present the particle number distribution by $n_N(D_p)$. The fundamental property of the particle number distribution is that by integrating the distribution function over all particle sizes we obtain the total particle number concentration across all sizes

$$N_t = \int_0^{\infty} n_N(D_p) dD_p \tag{2}$$

Also the cumulative particle number distribution $N(D_p)$ gives the particle number concentration upto particle size D_p and is given by

$$N(D_p) = \int_0^{D_p} n_N(D_p^*) dD_p^* \tag{3}$$

The following relationships allow us to convert particle number distributions to particle surface area $n_S(D_p)$ and volume $n_V(D_p)$ distributions

$$n_S(D_p) = \pi D_p^2 n_N(D_p) \tag{4}$$

$$n_V(D_p) = \frac{\pi}{6} D_p^3 n_N(D_p) \tag{5}$$

In a similar fashion the total and cumulative particle surface area (S, $S(D_p)$) and volume (V, $V(D_p)$) concentration can be found.

Alternative to particle size D_p one can express distributions as a function of the logarithm of the particle size LogD_p. This has the advantage that a larger range of particle sizes can be analyzed conveniently.

2 MATLAB Script

First we generate a random ensemble of 10,000 particles by drawing random variables from a Gaussian distribution. The ensemble can be defined given the total sample size `L1` a mean `mu1` and a standard deviation `std1`.

```
%ParticleSizeDistribution
%Particle size distributions

%Clear command window and memory
clc
clear

%Constants of simulation are simply particle ensembles within 1 cm^3 volume
%Each ensemble contains a total count, mean, and standard deviation
%We also define minimum and maximum particle size and bin width for analysis
L1=10000;       %Number of particles for ensemble 1
mu1=20;         %Mean particle diameter [um] of ensemble 1
std1=5;         %Standard deviation of particle diameter [um] of ensemble 1
dDp1=1;         %Width of each bin of particle diameter [um] for ensemble 1
Dpmin1=0;       %Minimum particle diameter [um] for ensemble 1
Dpmax1=70;      %Maximum particle diameter [um] for ensemble 1

%Generate random variable from a Gaussian distribution
rand1=std1.*randn(L1,1)+mu1;
```

Subsequently we analyze the particle sizes using a histogram that groups particles in a number of bins, given the minimum and maximum diameters `Dpmin1, Dpmax1` and the bin width `dDp1`. The function to perform this task with is `histogram`. Simultaneously by calculating the histogram, we also plot it. Since the ensemble of particles was defined over a 1 cm^3 volume. The histogram shows particle number concentration N [cm^{-3}] for each bin.

```
%Define the bins for analysis of the first distribution
bins1=[Dpmin1:dDp1:Dpmax1];
```

```
%Next we define the particle diameter vector. Each element contains bin centre
Dp1=[Dpmin1+dDp1/2:dDp1:Dpmax1-dDp1/2];

%The histogram of the particle ensemble is the number concentration
figure
h1=histogram(rand1,bins1);
xlabel('Dp [um]','FontSize',20);
ylabel('N [cm^{-3}]','FontSize',20);
```

Next we calculate particle number distribution by simply dividing the histogram elements by bin width `dDp1`. However, before doing this we need to copy the histogram values into a new vector `nN1` because the histogram data type `h1` contains a lot more information than we need. We access histogram values using the statement `h1.Values`. We can then plot the particle number distribution for ensemble 1 in the following figure. Notice that since the bin size width was 1 μm, the histogram and the particle number distributions are the same.

```
%Store data from the histogram in a new variable dedicated for number concentration
N1=h1.Values;

%Calculate particle number distribution by dividing number concentration by bin width
nN1=N1/dDp1;

%Now plot the number distribution
figure
plot(Dp1,nN1,'k-','LineWidth',3);
xlabel('Dp [um]','FontSize',20);
ylabel('n_N(Dp) [cm^{-3} um^{-1}]','FontSize',20);
```

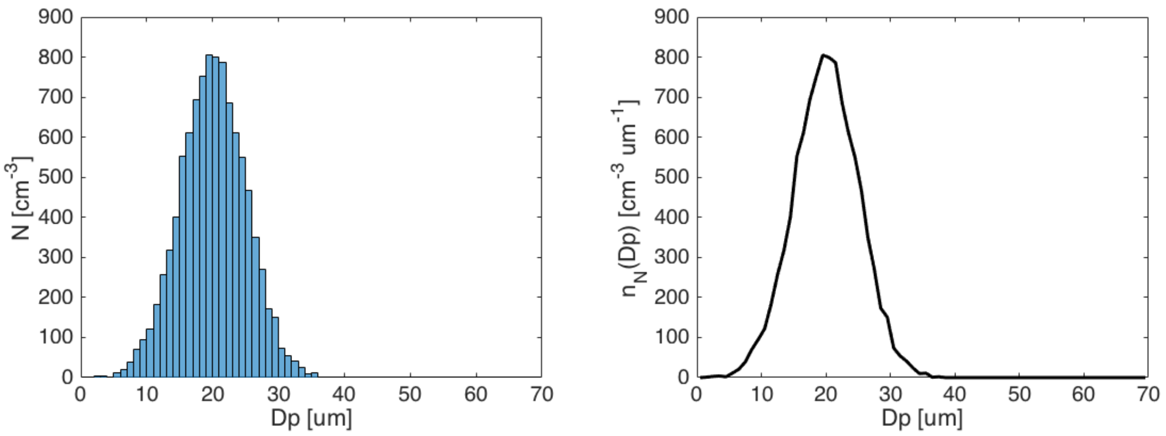


Figure 1: Histogram of particle number concentration (left) and particle number distribution (right) for particle ensemble 1.

We now generate a similar distribution but with a different bin size width. Use the following script to specify particle ensemble 2:

```
L2=10000;
mu2=20;
std2=5;
dDp2=2;
Dpmin2=0;
Dpmax2=70;
```

Subsequently write the necessary script to calculate and plot histogram and particle number distribution for particle ensemble 2. The following figure can be obtained.

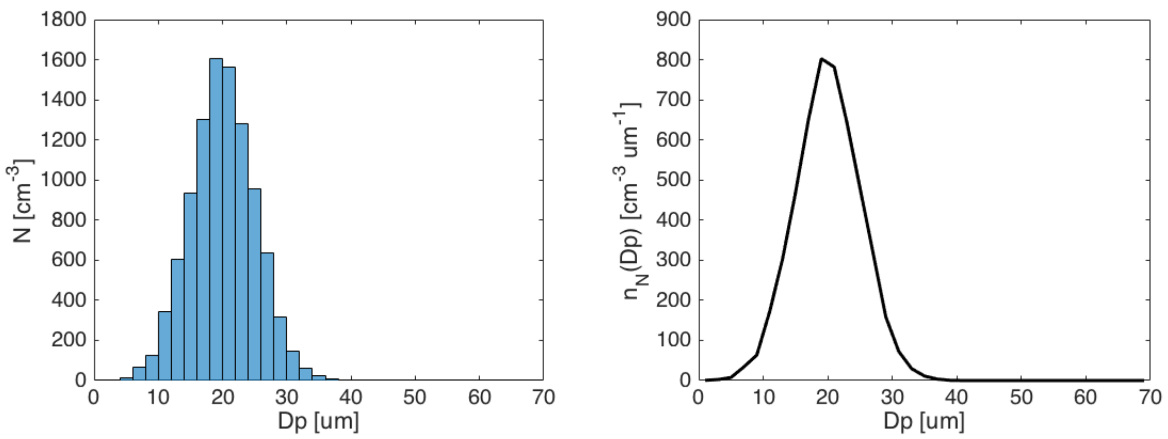


Figure 2: Histogram of particle number concentration (left) and particle number distribution (right) for particle ensemble 2.

Now by observing these figures discuss and answer the following questions:

- The vertical axis on the histogram plot for particle ensembles 1 and 2 shows that the histogram for particle ensemble 2 is twice larger than that for particle ensemble 1. Why is this?

- Unlike the histograms, the particle size distributions for particle ensembles 1 and 2 are similar, despite the difference in method to calculate them. Comment on why this is the case? By observing this behaviour, are you not convinced that particle size distributions are more useful than particle number concentrations?

Now we will compute the particle surface area `nS2` and volume `nV2` distributions for particle ensemble 2. This is, again, possible without needing to use `for` loops in MATLAB because we can perform short form vector operations element by element. Complete the following script to do this and then plot the number, area, and volume distributions on the same plot. The result should be the following figure.

```
%We now compute particle surface and volume distributions for ensemble 2
nS2=pi*(Dp2.^2).*nN2;
nV2=...
```

Now we will compute the cumulative particle number, surface area, and volume concentrations `N`,

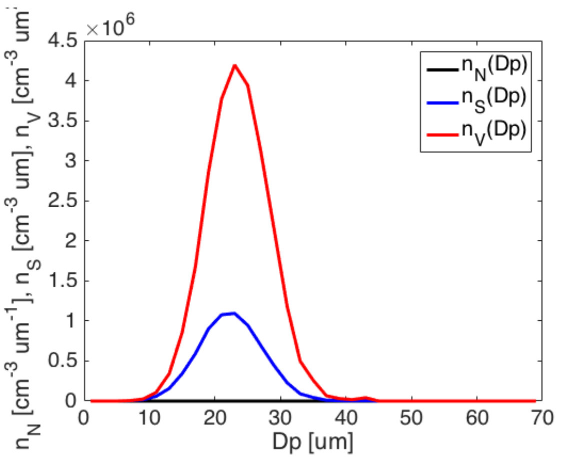


Figure 3: Particle number, surface area, and volume distributions for particle ensemble 2.

`S`, and `V`, respectively. Even though MATLAB may have a built-in function to do this, we do it by simple `for` loops, in order not to forget programming skills. To save memory, every time you want to fill in a vector element by element in a for loop, you must always define the vector first. We do this by the familiar `zeros` function and the dimension of the vector can be calculated by the `length` function, which gives the size of the vector supplied to it as an argument. Complete the following script to achieve this. Upon plotting you should get the following figure.

```
%We now compute the cumulative particle number, surface, and volume
%distributions for ensemble 2
%First create and initialize the cumulative distributions
N2=zeros(1,length(nN2));
S2=...
V2=...

%Then compute the cumulative distributions
N2(1)=nN2(1)*dDp2;
S2(1)=...
V2(1)=...
for i=2:length(nN2)
    N2(i)=N2(i-1)+nN2(i)*dDp2;
    S2(i)=...
    V2(i)=...
end
```

By comparing the last two figures, try to answer the following questions:

- Comment on why the volume and surface distribution and cumulative concentrations are much larger than the number distribution or concentration.
- Do the three distributions exhibit the same mode? Do the three cumulative concentrations exhibit the expansion point, i.e. the slope gradient, at the same particle diameter?

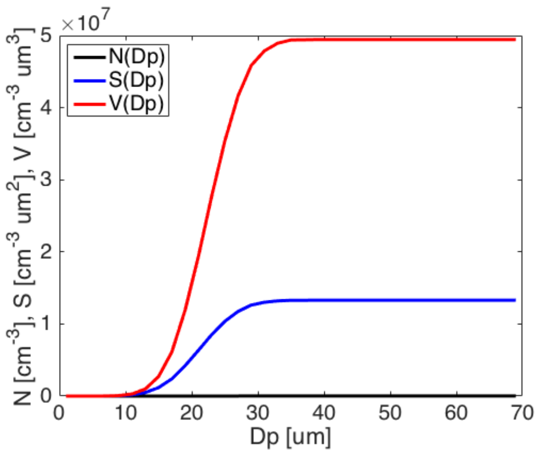


Figure 4: Cumulative particle number, surface area, and volume concentrations for particle ensemble 2.

Now we create a bimodal ensemble of particles, ensemble 3, with the logarithm of the particle size. The range for particle size is from `Dpmin3=0.001 um` or `LogDpmin3=-3` to `Dpmin3=1000 um` or `LogDpmin3=3` with the bin size width of `dLogDp3=0.2` in the logarithmic scale (Note: it is always good practice to use uniform bin size width, or else the analysis will be complicated). To create a bimodal ensemble of particles we define two sets of attributes (`a` and `b`) for two single-mode Gaussian distributions to draw random samples from. So we use different number of particles `L3a=40000, L3b=10000` and mean `mu3a=-1.5, mu3b=0`, but the same standard deviation `stda=stdb=0.4`. Use the following script to represent the particle ensemble.

```
L3a=40000;
mu3a=-1.5;
std3a=0.4;
L3b=10000;
mu3b=0;
std3b=0.4;
dLogDp3=0.2;
LogDpmin3=-3;
LogDpmax3=3;
```

Next, complete the following script to generate random vectors necessary to generate particle ensemble 3 and create overlapping histograms to represent the particle number concentrations

```
%Now we create a bimodal distribution in the lognormal representation
rand3a=std3a.*randn(L3a,1)+mu3a;
rand3b=...
bins3=[LogDpmin3:dLogDp3:LogDpmax3];
LogDp3=[LogDpmin3+dLogDp3/2:dLogDp3:LogDpmax3-dLogDp3/2];

figure
h3a=histogram(rand3a,bins3);
```

```
hold on
h3b=...
xlabel('Log Dp [um um^{-1}]','FontSize',20);
ylabel('N [cm^{-3}]','FontSize',20);
```

Then create the particle number distribution for particle ensemble 3 by adding the random vectors and dividing by bin size width in the logarithm scale using the script below. Then plot it. You should obtain the following figures.

```
N3=h3a.Values+h3b.Values;
nN3=...
```

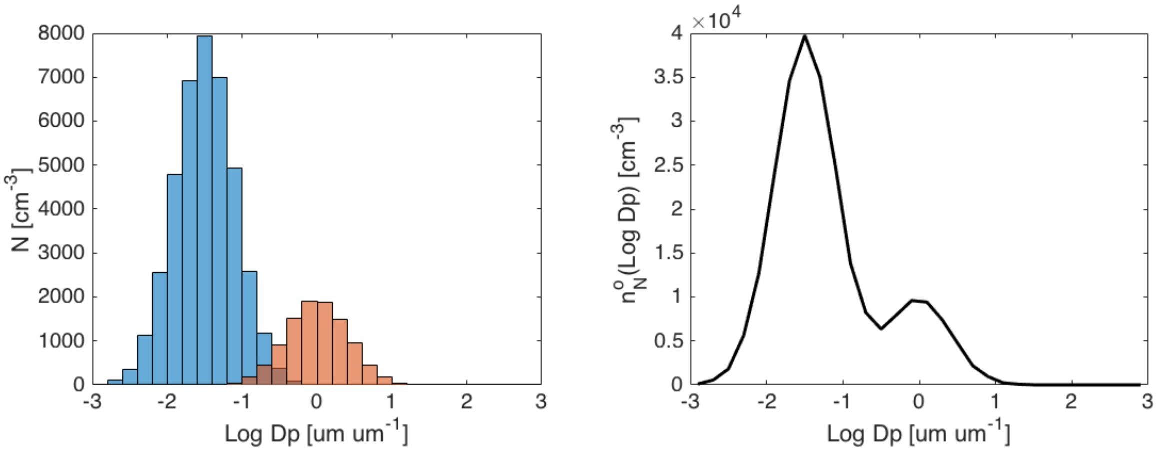


Figure 5: Histogram of particle number concentration (left) and particle number distribution (right) for particle ensemble 3.

Finally compute the particle surface area `nS3` and volume `nV3` distributions for particle ensemble 3. This is, again, possible without needing to use `for` loops in MATLAB because we can perform short form vector operations element by element. Complete the following script to do this and then plot the number, area, and volume distributions on the same plot.

```
%We now compute particle surface and volume distributions for ensemble 3
nS3=pi*((10.^LogDp3).^2).*nN3;
nV3=...
```

Now we will compute the cumulative particle number, surface area, and volume concentrations `N`, `S`, and `V`, respectively. Again, we do it by simple `for` loops. The scripting for this calculation and plotting is left for you. You should get the following figures.

By observing these figures try to answer the following questions:

- The larger peak in the number distribution is the peak to the left, while the larger peak in the surface and volume distributions is the peak to the right. Why is this?
- Try to explain the behaviour of the cumulative concentrations. Discuss the expansion points in these plots.

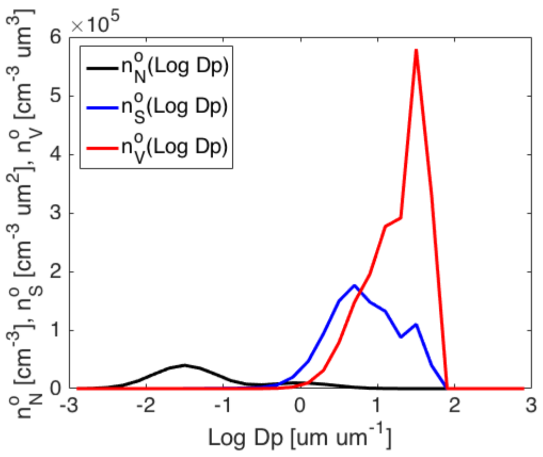


Figure 6: Particle number, surface area, and volume distributions for particle ensemble 3.

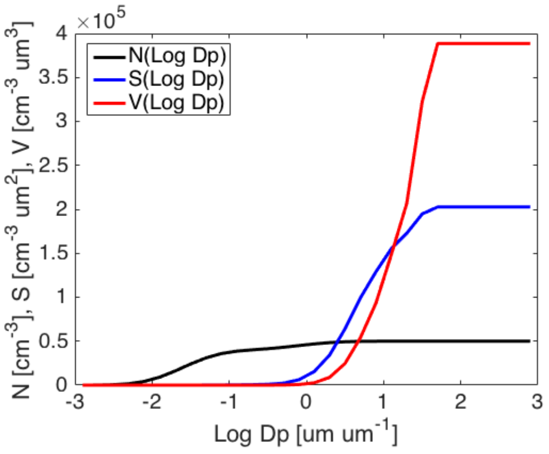


Figure 7: Cumulative particle number, surface area, and volume concentrations for particle ensemble 3.

ENGG*4810: Control of Atmospheric Particulates

Particle Heat and Mass Transfer

Amir A. Aliabadi

November 29, 2017

1 Introduction

In this lab we are going to study evaporation of water droplets in ambient air. The fluid outside the water droplet is a binary system consisting of air molecules and water molecules. So this binary system consists of species A for water vapor and B for air. For simplicity in the following formulations, we will drop index A or B, and insert p for A in places where properties should be inserted associated with the water and not air to avoid confusion. From lecture notes we learnt that the evaporation of a droplet is governed by the following equation

$$R_p^2 = R_{p0}^2 + \frac{2DM_p}{\rho_P}(c_\infty - c_0)t \tag{1}$$

Most terms in this equation can be determined easily except for molar concentrations c that need to be specified as a function of binary mixture temperature and relative humidity. The specific humidity in air is defined as the ratio of mass of water vapor contained in a mass of water vapor-air binary mixture and is given by

$$w = \frac{m_v}{m_a + m_v} \simeq \frac{m_v}{m_a} \tag{2}$$

where m_v is mass of water vapor and m_a is mass of air in the binary mixture. The approximation is valid since mass of water vapor in air is usually very small. Specific humidity can be expressed using partial pressures associated with water vapor and air

$$w = \frac{m_v}{m_a} = \frac{\frac{P_vV}{R_vT}}{\frac{P_aV}{R_aT}} = \frac{R_a}{R_v}\frac{P_v}{P_a} = 0.622\frac{P_v}{P_a} = \frac{0.622P_v}{P - P_v} \tag{3}$$

where V is common volume which the binary mixture occupies, P is pressure ($P = P_v + P_a$ is atmospheric pressure), T is temperature, and R is gas constant. This is essentially derived using the ideal gas equation of state. The relative humidity, on the other hand, is the ratio of mass of water vapor to the maximum possible mass of water vapor in air (i.e. before adding any more water vapor resulting in condensation) and is given by

$$\phi = \frac{m_v}{m_g} = \frac{\frac{P_v V}{R_v T}}{\frac{P_g V}{R_g T}} = \frac{P_v}{P_g} \tag{4}$$

where P_g is saturation pressure for water vapor at a given temperature. With the above developments we have

$$w = \frac{0.622\phi P_g}{P - \phi P_g} \tag{5}$$

From this discussion it is apparent that we can calculate molar concentrations necessary for the evaporation equation as

$$c_\infty = w_\infty \frac{\rho}{M_p} = \frac{0.622\phi P_{g\infty}}{P - \phi P_{g\infty}} \frac{\rho}{M_p} \tag{6}$$

$$c_0 = w_0 \frac{\rho}{M_p} = \frac{0.622 P_{g0}}{P - P_{g0}} \frac{\rho}{M_p} \tag{7}$$

where subscript ∞ indicates the far-field conditions, and subscript 0 indicates conditions near the droplet surface. $\phi = 1$ near the droplet, i.e. the binary mixture is saturated near the surface of the droplet. ρ [kg m^{-3}] is density of mixture that is practically the density of dry air, and M_p [kg mol^{-1}] is the molecular weight of water.

In general, water vapor pressure is s function of temperature, therefore both c_∞ and c_0 depend on temperature so that we cannot directly solve the evaporation equation as it is. Instead, we need to solve for droplet surface temperature first using iterative methods, and then solve the evaporation equation. An expression is provided in the lecture notes that gives the equilibrium surface temperature for an evaporating droplet. However, at the time of developing this lab, I could not find a numerical solution to it. Instead, I used another expression from Clausius and Clapeyron to solve for equilibrium droplet surface temperature of an evaporating droplet

$$\Delta_T = \frac{h}{c_p T_\infty} \frac{D}{\alpha} \left(w_\infty - w_{g\infty} \exp\left[\frac{h}{R_a T_\infty} \frac{\Delta_T}{1 + \Delta_T} \right] \right) \tag{8}$$

$$\Delta_T = \frac{T_0 - T_\infty}{T_\infty} \tag{9}$$

where T_0 [K] is equilibrium droplet surface temperature, h [J kg^{-1}] is enthalpy of evaporation that is also a function of temperature, T_∞ [K] is far-field temperature, c_p [J kg^{-1} K^{-1}] is specific heat capacity of water, D [m^2 s^{-1}] is diffusion coefficient of air, α [m^2 s^{-1}] is thermal diffusivity of air, w_∞ is mass fraction of water vapor in air at far-field, $w_{g\infty}$ is mass fraction of water vapor in air under saturated conditions, and R_a [J kg^{-1} K^{-1}] is the gas constant for air.

Once this equation is solved iteratively for T_0, we can then solve the evaporation equation numerically to simulate the rate of droplet evaporation.

2 Simulation Matrix

We wish to perform simulation of droplet evaporation in a binary mixture of air and water vapor at rest. We want to vary initial droplet radius as well as the relative humidity at far-field in the binary system to see the effects on droplet cooling and evaporation time. Table below shows the simulation matrix.

Table 1: Simulation matrix

Simulation	1	2	3	4	5	6	7	8
R_{p0} [m]	1e–3	1e–4	1e–5	1e–6	1e–3	1e–4	1e–5	1e–6
T [K]	300	300	300	300	300	300	300	300
ϕ	0.4	0.4	0.4	0.4	0.1	0.1	0.1	0.1

3 MATLAB Script

As usual, we begin be defining the simulation constants and the simulation time step. We wish to simulate droplet evaporation over `nt=2000` time steps. Complete the following script

```
%ParticleHeatMassTransfer
%Particle Heat and Mass Transfer

%Clear command window and memory
clc
clear

%Constants of simulation
pat=101.35;     %Atmospheric pressure [kPa]
p=101000;       %Reference air pressure [Pa]
Mp=18e-3;       %Water molecular weight [kg mol^-1]
M=28.966e-3;    %Air molecular weight [kg mol^-1]
cp=1007;        %Specific heat of particle [J kg^-1 K^-1]
R=8.314;        %Universal gas constant [J K^-1 mol^-1]
Ra=287.06;      %Air Gas constant [J kg^-1 K^-1]
rhop=1000;      %Water particle density [kg m^-3]
k=0.025;        %Air conductivity [W m^-1 K^-1]
Rp0=1e-3;       %Particle initial radius [m]
Tinf=300;       %Far-field air temperature [K]
Tref=300;       %Reference temperature [K]
mu=1.846e-5;    %Air dynamic viscosity [kg m^-1 s^-1]
rho=1.777;      %Air density [kg m^-3]
phi=0.4;        %Air relative humidity [0-1]
D=2.5e-5;       %Diffusivity of air [m^2 s^-1]
nt=2000;        %Number of time steps
```

```
%Calculate mean free path, Knudsen number, slip correction, and mass
lambda=...
Kn=...
Cc=...

%Calculate particle characteristic time
tau=...
%Set time step as 10% of the smallest particle characteristic time
dt=0.1*tau;
```

Next, we define vectors for analysis. First we make the time vector `t` and the particle radius vector `Rp`. We then make vectors from property tables of temperature, saturation pressure, enthalpy, and thermal diffusivity. These vectors are needed for the iterative solution of the equilibrium droplet surface temperature for an evaporating droplet. Insert the following script.

```
%Define time and particle radius vectors
t=0:dt:(nt-1)*dt;
Rp=zeros(1,nt);

%Water vapor pressures, enthalpy of vaporization and air thermal diffusivity
T=[5+273.15 10+273.15 15+273.15 20+273.15 25+273.15 30+273.15 35+273.15];
%[K]
Psat=[0.8721 1.2276 1.7051 2.339 3.169 4.246 5.628];
%[kPa]
h=1e3*[2489.6 2477.7 2465.9 2454.1 2442.3 2430.5 2418.6];
%[J kg^-1]
alpha=1e-5*[1.88 1.944 2.009 2.074 2.141 2.208 2.277];
%[m^2 s^-1]
```

It is easier to fit polynomials to these properties so we can efficiently interpolate when needed. The MATLAB function to use is `polyfit`. Insert the following script.

```
%Fit polynomials (1st or 2nd order) through the data
aPsat=polyfit(T,Psat,2);
ah=polyfit(T,h,1);
aalpha=polyfit(T,alpha,2);
```

Next we calculate properties at far-field given the far-field temperature and relative humidity. We make use of the polynomial coefficients for these calculations. Insert the following script.

```
%Calculate far-field vapor pressure [kPa]
Pginf=aPsat(1)*Tinf^2+aPsat(2)*Tinf+aPsat(3);

%Calculate far-field latent heat of vaporization [J kg^-1]
hinf=ah(1)*Tinf+ah(2);
```

```
%Calculate far-field specific humidity
winf=0.622*phi*Pginf/(101.35-phi*Pginf);

%Calculate far-field vapor molar concentration [mol m^-3]
cinf=winf*rho/Mp;

%Calculate far-field mass fraction under saturated conditions
wginf=winf/phi;

%Calculate far-field air thermal diffusivity [m^2 s^-1]
alphainf=...
```

Now we solve for the equilibrium surface temperature for the evaporating droplet iteratively using MATLAB's `fsolve` function, which attempts to find the zero of a function given an initial solution. The equation is the Clausius Clapeyron equation, and the initial solution is `Tinf`.

```
%Solve for the equilibrium particle surface temperature
T0=fsolve(@(x)-(x-Tinf)/Tinf+(hinf/Tinf)*(1/cp)*(D/alphainf)...
*(winf-wginf*exp((hinf/Ra/Tinf)*((x-Tinf)/x))),Tinf);
```

Once we have the surface temperature `T0` we can calculate other surface properties needed for the evaporation equation. Insert the following script for this calculation.

```
%Solve for surface saturation pressure and molar concentration
%Note at close to the particle we have phi=1
Pg0=aPsat(1)*T0^2+aPsat(2)*T0+aPsat(3);
c0=0.622*Pg0/(101.35-Pg0)*rho/Mp;
```

Finally we can loop through the time steps and calculate the droplet radius using the evaporation equation. Complete the following script. To compute the square root of an expression, you can use MATLAB's `sqrt()` function.

```
%Now loop through time steps to calculate evaporation time
for i=1:nt
    if ((Rp0^2+(2*D*Mp/rhop)*(cinf-c0)*t(i))>0)
        Rp(i)=...
    end
end
```

The last step is to plot droplet radius `Rp` versus time `t`. Perform the simulation for the cases identified in the simulation matrix. You should get the following figures.

Note that in our solution we used the continuum theory throughout the droplet evaporation process, even for the time when droplet becomes very small. There are errors associated with this assumption, but this analysis provides a good approximation, at least showing the relative differences in time scales involved. Discuss the following questions.

- Explain the relationship between initial droplet radius and evaporation time. Is this a linear relationship? for instance if the droplet size is reduced by a factor of 10, does the evaporation time also reduce by a factor of 10?

- Explain the relationship between relative humidity the amount of cooling of the drop. Does a lower relative humidity result in a lower equilibrium surface temperature of an evaporating droplet?

- Explain the relationship between evaporation time and relative humidity. Does lowering relative humidity reduce evaporation time?

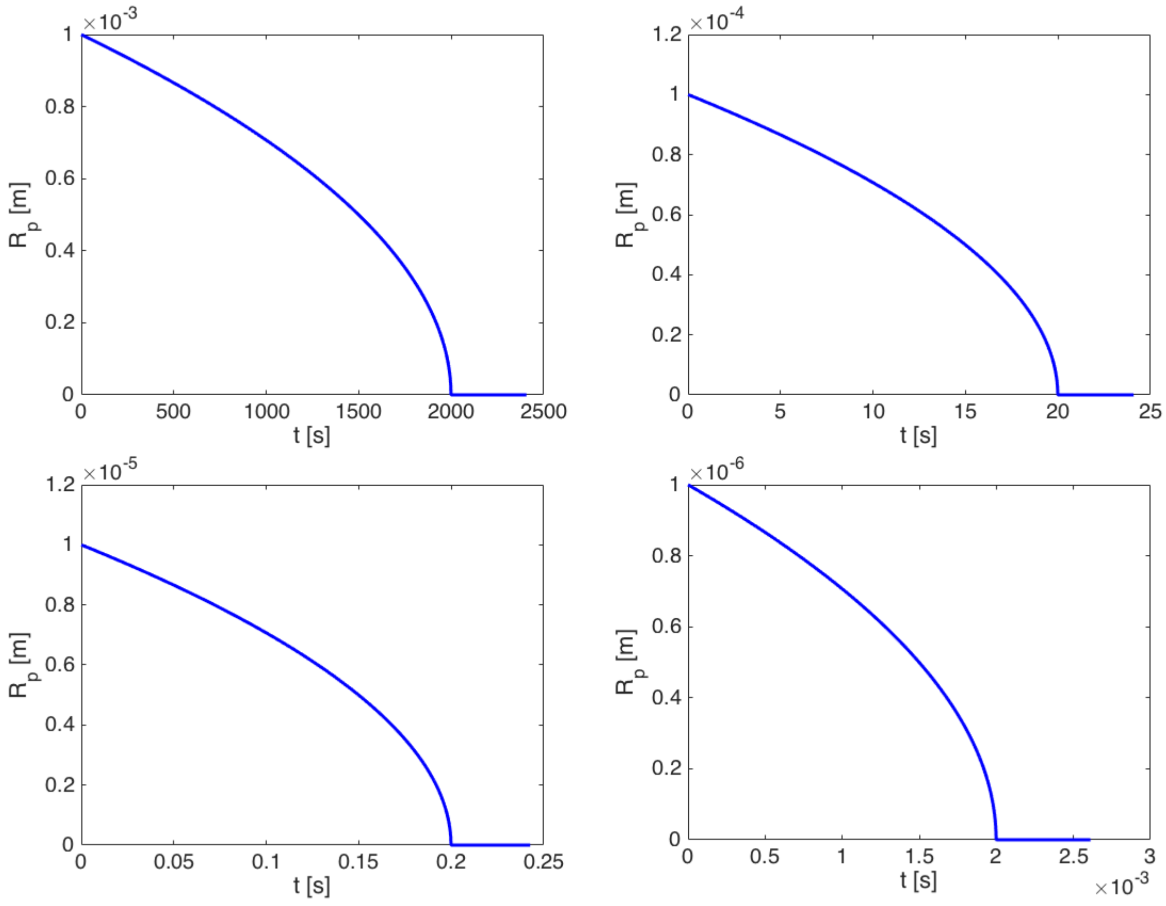


Figure 1: Time evolution of droplet radius due to evaporation for relative humidity `phi=0.4`. Particle radii are `Rp0=1e-3` m (top left), `Rp0=1e-4` m (top right), `Rp0=1e-5` m (bottom left), and `Rp0=1e-6` m (bottom right).

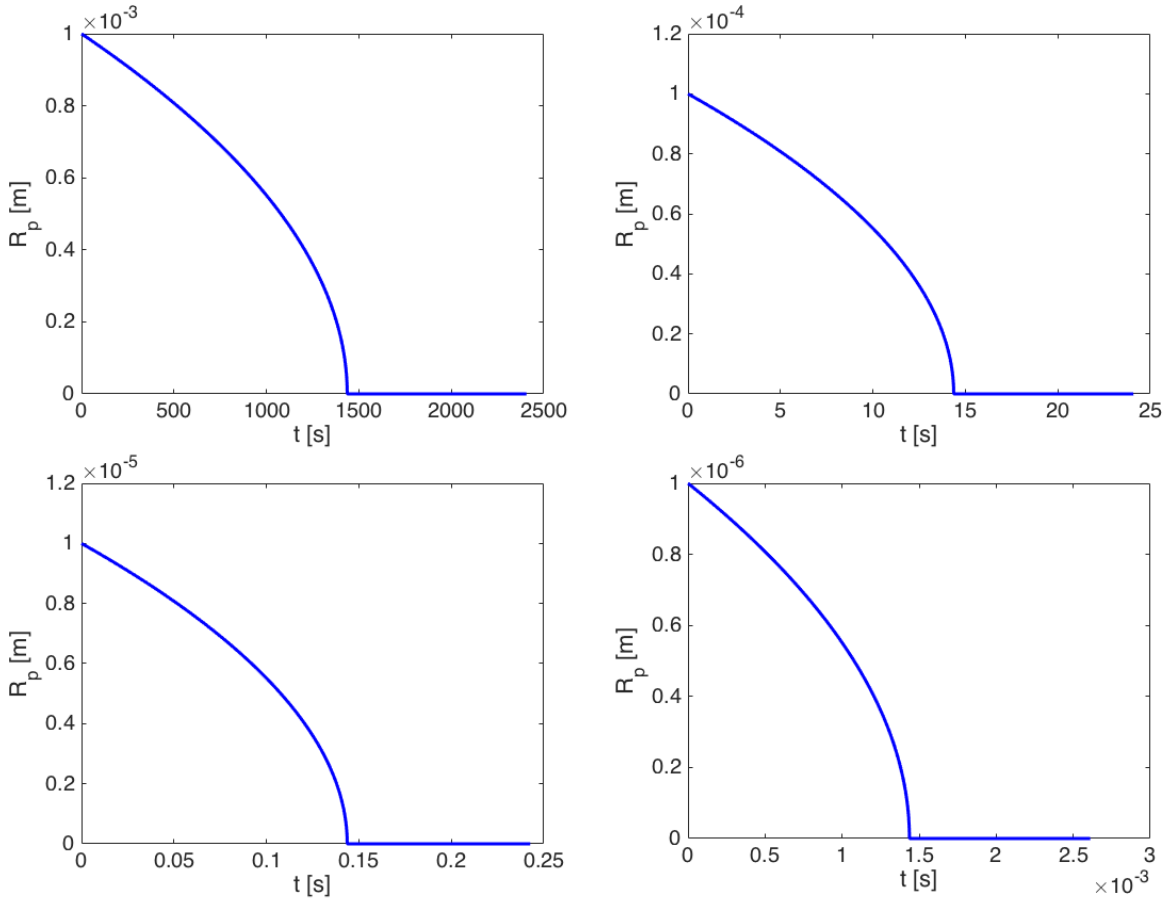


Figure 2: Time evolution of droplet radius due to evaporation for relative humidity `phi=0.1`. Particle radii are `Rp0=1e-3` m (top left), `Rp0=1e-4` m (top right), `Rp0=1e-5` m (bottom left), and `Rp0=1e-6` m (bottom right).

ENGG*4810: Control of Atmospheric Particulates

Brownian Diffusion of Particles

Amir A. Aliabadi

November 13, 2017

1 Introduction

In a previous lab we introduced the methodology of [Li and Ahmadi, 1992] and [Ounis et al., 1991] in the simulation of Brownian diffusion of particles. In this lab we apply the simulation to an ensemble of sub-micrometer particles and further compare the Brownian diffusion to theoretical predictions.

In 1905 Albert Einstein derived an expression that describes the Brownian diffusion of particles with diameter D_p [m] in a fluid with dynamic viscosity of μ [kg m^{-1} s^{-1}] and temperature T [K]. If an ensemble of particles are all released at a single point of a resting fluid where Brownian force is the only force applied to the particles, then the mean square displacement of the particles in the x, y, and z directions are given by

$$< x^2 >=< y^2 >=< z^2 >= 2 \underbrace{\frac{kTC_c}{3\pi\mu D_p}}_{D} t \tag{1}$$

where k [J K^{-1}] is the Boltzmann constant, C_c is the slip correction factor, and t is time elapsed since particle release. D [m^2 s^{-1}] is the familiar diffusion coefficient of the fluid. This equation has been shown, experimentally, to be very accurate for sub-micrometer particles.

It is of interest to check if the simulation of the Brownian diffusion introduced in a previous lab is in agreement with theoretical prediction.

2 MATLAB Script

We begin the MATLAB script by defining constants of simulation and determining a convenient time step taken as a fraction of the particle relaxation time. We consider an ensemble of 100 particles (`np`) and 100 time steps (`nt`).

```
%ParticleBrownianDiffusion
%Particle Browning diffusion

%Clear command window and memory
clc
clear

%Constants of simulation
p=101000;        %Air pressure [Pa]
R=8.314;         %Universal gas constant [J K^-1 mol^-1]
M=28.966e-3;     %Air molecular weight [kg mol^-1]
rhop=1.777;      %Particle density [kg m^-3]
k=1.38e-23;      %Boltzmann constant [J K^-1]
Dp=1e-7;         %Particle diameter [m]
T=300;           %Air temperature [K]
mu=1.846e-5;     %Air dynamic viscosity [kg m^-1 s^-1]
rho=1.777;       %Air density [kg m^-3]
np=100;          %Number of particles in ensemble
nt=100;          %Number of time advances

%Calculate mean free path, Knudsen number, slip correction, and mass
lambda=...
Kn=...
Cc=...
mp=...

%Calculate particle characteristic time
tau=...
%Set time step as 10% of the smallest particle characteristic time
dt=0.1*tau;

%Define time vector
t=0:dt:(nt-1)*dt;
```

We then define the position matrices, and not vectors, because now for each time step we want to keep track of individual particle positions. For particle velocities and accelerations, we only define vectors since we only need to ‘remember’ those for the last time step.

```
%Define position matrices and initialize to zero
%The two dimensional matrix contains position as a function of time
x=zeros(nt,np);     %x Position vector [m]
y=...
z=...

%Define acceleration vectors for the most recent time step
ax=zeros(1,np);
```

```
ay=...
az=...

vx=zeros(1,np);
vy=...
vz=...

vxold=zeros(1,np);
vyold=...
vzold=...
```

Then we iterate through time steps, only applying a Brownian acceleration in the three coordinates.

```
%Loop through the time steps for the entire particle ensemble
%Update particle position and velocity
for i=2:nt
    for j=1:np
        %Append accelerations by Brownian force
        S0=(216*mu*k*T)/((pi^2*Dp^5*rhop^2)*Cc);
        ax(j)=randn(1)*((pi*S0/dt)^0.5);
        ay(j)=...
        az(j)=...
        %Find new velocities
        vx(j)=vxold(j)+ax(j)*dt;
        vy(j)=...
        vz(j)=...
        %Find new position
        x(i,j)=x(i-1,j)+vx(j)*dt;
        y(i,j)=...
        z(i,j)=...
        %Update old velocities for next iteration
        vxold(j)=vx(j);
        vyold(j)=...
        vzold(j)=...
    end
end
```

After the computation is done, we wish to plot, in three dimensions, the distribution of the ensemble of particles in space in three times of `t(1)`, `t(nt/2)`, and `t(nt)`. For this we can use the `plot3` function. For ease of comparison we fix the axes limits using the `axis` specifier. Complete the following script to achieve this. If everything goes well, you should get the following figures.

```
figure
plot3(x(1,:),y(1,:),z(1,:),'bo','MarkerSize',5);
xlabel('x [m]','FontSize',20);
```

```
ylabel('y [m]','FontSize',20);
zlabel('z [m]','FontSize',20);
axis(3e-8*[-1 1 -1 1 -1 1]);

figure
...

figure
...
```

By looking at the range on x, y, and z axes we realize the the diffusion has just started because the spatial spread of the particles is a fraction of the particle diameter.

Subsequently we wish to calculate and plot the mean square distance of the ensemble of the particles in the x direction from the origin for particle sizes `Dp=1e-9, 1e-8, 1e-7,` and `1e-6` m. We wish to do this calculation using both simulation and the theoretical formula. Append the following script to perform this task. In order to be able to show data on a wider range we use a logarithm vertical axis by MATLAB function `semilogy`. You should obtain the following plots.

```
%Define vectors for mean square distance of particle ensemble from origin
x2=zeros(nt,1);
x2theory=zeros(nt,1);

%We now calculate mean square distance of particle ensemble from origin
for i=1:nt
    x2(i)=mean(x(i,:).^2);
    x2theory(i)=2*k*T*Cc*t(i)/(3*pi*mu*Dp);
end

%Plot the simulated and theoretical mean square distance of partilces from origin
figure
semilogy(t,x2,'k-','LineWidth',3);
hold on
semilogy(t,x2theory,'k:','LineWidth',3);
xlabel('t [s]','FontSize',20);
ylabel('<x^2> [m^2]','FontSize',20);
h_legend=legend('Simulation','Theory');
set(h_legend,'FontSize',20);
```

Notice that the two curves only agree in one point. At early times the simulation under-predicts the theoretical prediction, while at large times the simulation over-predicts the theoretical prediction. It appears that with a constant time step, chosen as a fraction of the particle relaxation time, the simulation is doomed to show different values from the theory. This shows one of the limitations of this simulation, which cannot accurately predict Brownian diffusion. In the presence of turbulent diffusion (usually the case), however, the Brownian diffusion is very insignificant compared to turbulent mechanisms which randomly disperse particles. Therefore, it is not necessary to simulate

Brownian motion. Nevertheless, this technique demonstrates the rough dispersion behaviour as a function of particle size and provides a qualitative result.

References

[Li and Ahmadi, 1992] Li, A. and Ahmadi, G. (1992). Dispersion and deposition of spherical particles from point sources in turbulent channel flow. *Aerosol Sci. Technol.*, 16:209–226.

[Ounis et al., 1991] Ounis, H., Ahmadi, G., and McLaughlin, J. B. (1991). Brownian diffusion of submicrometer particles in the viscous sublayer. *J. Colloid Interf. Sci.*, 143(1):266–277.

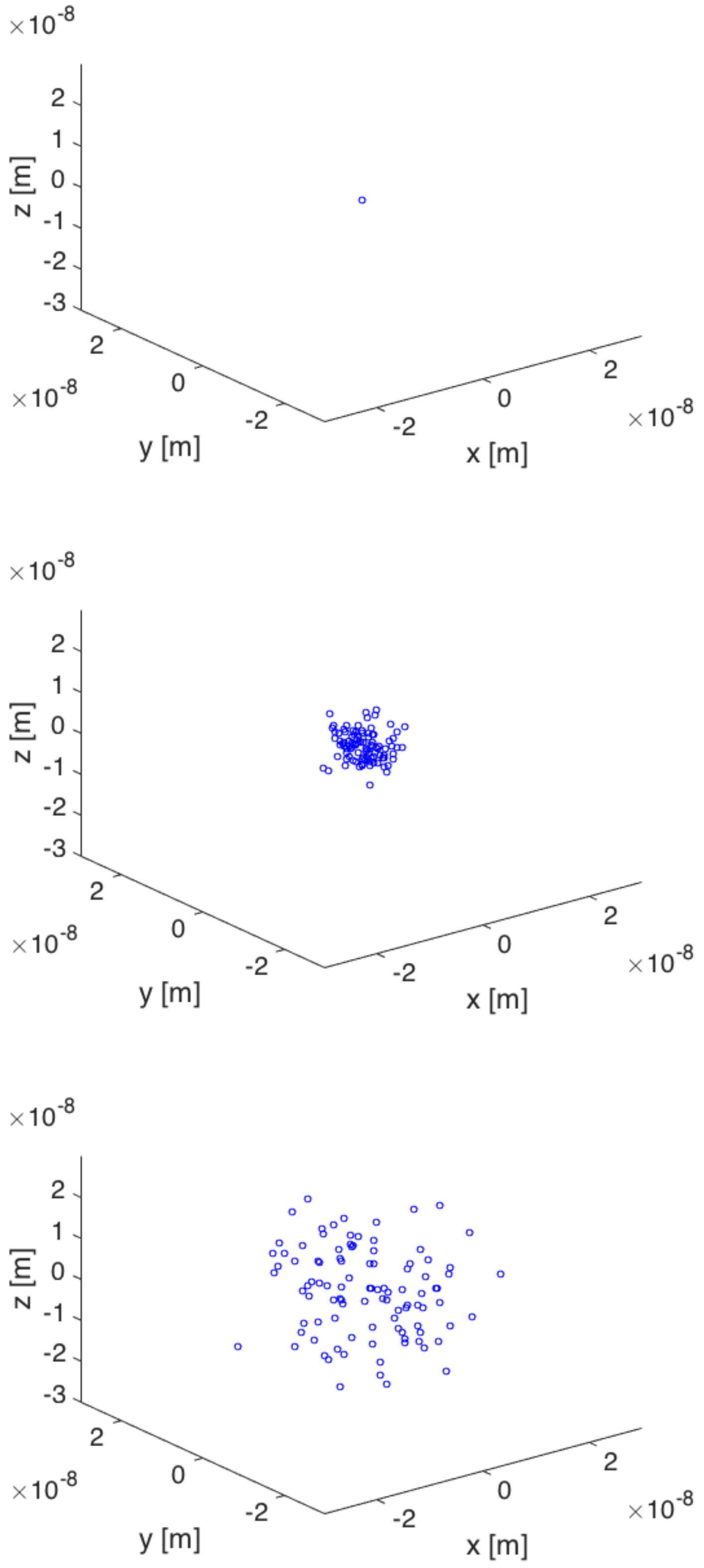


Figure 1: Distribution of an ensemble of 100 particles with diameter `Dp=1e-7` m at `t(1)`, `t(nt/2)`, and `t(nt)`.

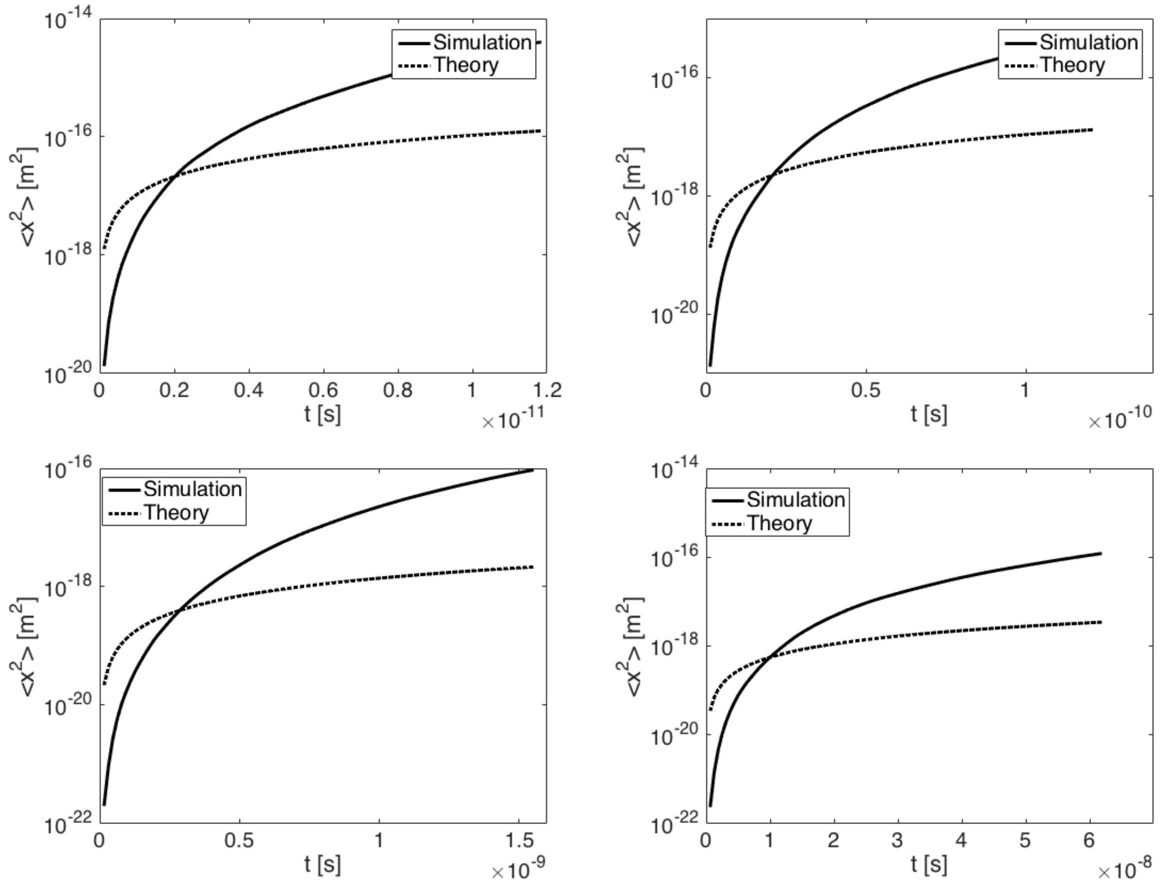


Figure 2: Time evolution of the mean square distance for ensemble of particles from the origin. Particle diameters are `Dp=1e-9` m (top left), `Dp=1e-8` m (top right), `Dp=1e-7` m (bottom left), and `Dp=1e-6` m (bottom right).

ENGG*4810: Control of Atmospheric Particulates

Particle Dispersion in a Turbulent Jet

Amir A. Aliabadi

October 7, 2019

1 Introduction

In this lab we are going to simulate turbulent dispersion of particles in a jet using the Random Walk Model (RWM) and the concept of Eddy Interaction (EI). Jets belong to a class of flows known as *free-shear* flows [Hussein et al., 1994]. Such flows exhibit sharp velocity gradients but are not constrained by walls or other types of boundaries, the reason they are called free-shear flows. Jets occur in numerous natural or human made processes such as sneezing, pollen release, fuel injection engines, aircraft propulsion, and more. Most jets are *self-similar* in structure, very much analogous to similarity in geometry. For example, two triangles are similar if they have the same angles. Jets too, have similar properties, no matter how far they are probed from the point of injection.

Figure below shows schematic of a self-similar jet. Note that a virtual origin, x_0, for the jet is defined for the point where the self-similar behaviour for the jet begins. It is assumed that the injecting nozzle diameter is D.

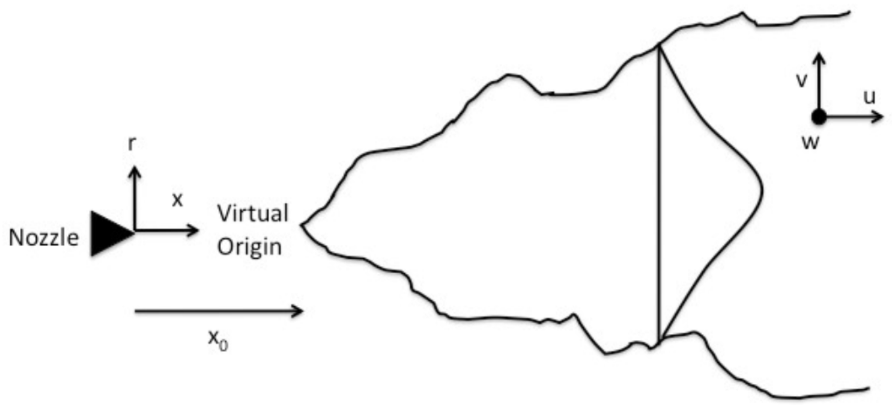


Figure 1: Schematic of a self-similar jet.

Here two coordinate systems can be used to specify jet flow dynamics. 1) In the cylindrical coordinate system, x is the direction of the axial propagation of the jet, and r and θ, specify the radial and angle positions in the transverse direction. 2) In the Cartesian coordinate system, x is still the the direction of the axial propagation of the jet, but y and z directions are fixed. z is usually chosen in such a way that gravitational acceleration points in the $-z$ direction. y is the horizontal direction. In both coordinate systems, fluid velocities $u-v-w$ can represent the axial and transverse velocities corresponding to $x-r-\theta$ or $x-y-z$ coordinate systems, respectively. The velocities in the two coordinate systems can be related such that

$$u_{car} = u_{cyl} \tag{1}$$
$$v_{car} = v_{cyl}\cos(\theta) - w_{cyl}\sin(\theta) \tag{2}$$
$$w_{car} = v_{cyl}\sin(\theta) + w_{cyl}\cos(\theta) \tag{3}$$

For a self-similar steady jet, the following linear correlation between the mean centre line velocity in the axial direction $\overline{U_c}$, initial mean velocity in the axial direction at the origin $\overline{U_0}$, nozzle diameter D, and distance from the nozzle x has been established:

$$\frac{\overline{U_0}}{\overline{U_c}} = \frac{1}{B_u}\left(\frac{x}{D} - \frac{x_0}{D}\right) \tag{4}$$

where B_u is an empirical constant found to be 5.8 by experiments. The dimensionless origin is found to be $\frac{x_0}{D} = 4$. Figure below shows contour plots of velocity magnitude, turbulent kinetic energy, and dissipation rate for a self-similar jet. For this jet, it is assumed that the nozzle diameter is $D = 0.01$ m and that a total gas volume of $V = 0.0005$ m^3 is injected over time $t = 1$ s, which allows calculation of injection velocity. The axial domain x starts from $x = 6D = 0.06$ m or six nozzle diameters.

A similarity variable has been found, experimentally, to describe mean flow turbulence properties of a self-similar jet

$$\eta = \frac{r}{x - x_0} \tag{5}$$

where r is the radial distance from the centre line of the jet. Numerous functions have been fitted to describe round jets [Hussein et al., 1994], where model fitting to experimental data is achieved by method of least squares to all measured profiles with a similarity variable and fitted an even function given below to calculate mean flow and turbulent properties,

$$p(\eta) = \left[C_0 + C_2\eta^2 + C_4\eta^4 + ...\right]\exp\left(-A\eta^2\right). \tag{6}$$

The multiplication of the polynomial and exponential function provides an excellent fit over the range in which data were taken ($\eta < 0.2$), and care must be given not to apply these fits beyond this range. Table below shows the fitting parameters.

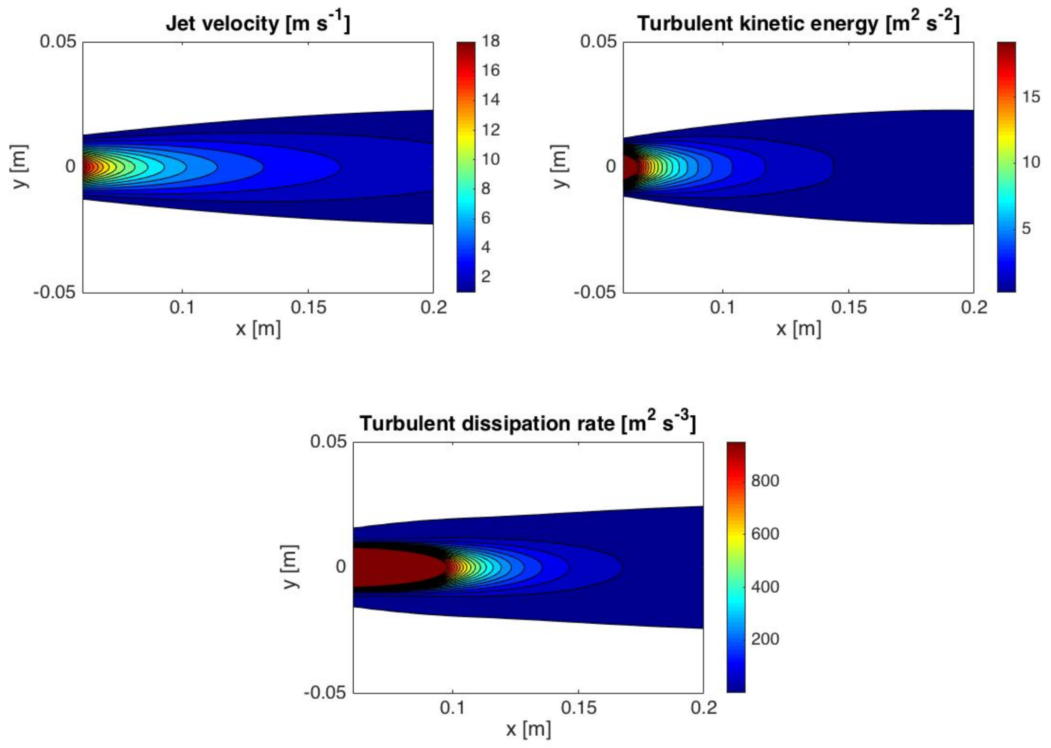


Figure 2: Contours of velocity magnitude, turbulent kinetic energy, and dissipation rate for a self-similar jet.

2 Simulation Matrix

We wish to simulate turbulent dispersion of $n_p = 200$ particles of two different diameters, D_p [m], in the above jet for two injection volumes, V_{inj}. Table below shows the simulation matrix.

3 MATLAB Script

As usual, we begin be defining the simulation constants and the simulation time step. Complete the following script

```
%TurbulentJetDispersion
%Particle dispersion in a jet under forces of
%gravity, buoyancy, drag, and turbulence

%Clear command window and memory
clc
clear
```

Table 1: Constants to determine turbulent properties of a self-similar jet [Hussein et al., 1994]

$p(\eta)$	C_0	C_2	C_4	C_6	A
$\overline{U}/\overline{U_c}$	1.0	–1.925	0.0	0.0	63
$\overline{u'^2}/\overline{U_c}^2$	7.778e–2	2.79e1	–2.02e3	4.3e5	257
$\overline{v'^2}/\overline{U_c}^2$	5.457e–2	0.355	–4.298e1	0.0	89
$\overline{w'^2}/\overline{U_c}^2$	5.78e–2	–1.71	2.73e–1	0.0	42
$\overline{u'v'}/\overline{U_c}^2$	4.375e–1	–3.931e1	1.55e2	1.342e4	90
$\epsilon/\left[\overline{U_c}^3/(x-x_0)\right]$	0.3549	11.99	–1635	43470	201

Table 2: Simulation matrix

Simulation	1	2	3	4
D_p [m]	1e–6	1e–5	1e–6	1e–5
V_{inj} [m^3]	0.5e–3	0.5e–3	1.0e–3	1.0e–3

```
%Constants of simulation
g=9.81;         %Gravitational acceleration [m s^-2]
p=101000;       %Air pressure [Pa]
R=8.314;        %Universal gas constant [J K^-1 mol^-1]
M=28.966e-3;    %Air molecular weight [kg mol^-1]
rhop=1000;      %Particle density [kg m^-3]
Dp=1e-6;        %Particle diameter [m]
T=300;          %Air temperature [K]
mu=1.846e-5;    %Air dynamic viscosity [kg m^-1 s^-1]
rho=1.777;      %Air density [kg m^-3]
np=200;         %Number of particles in ensemble
nt=40000;       %Number of time advances
tinj=1.0;       %Injection time [s]
Vinj=0.5e-3;    %Total amount of air injected [m^3]
Dnoz=0.01;      %Nozzle diameter [m]
Bu=5.8;
Cmu=0.09;

%Calculate mean free path, Knudsen number, slip correction, and mass
lambda=...
Kn=...
Cc=...
mp=...

%Calculate particle characteristic time
tau=...
%Set time step as 10% of the particle characteristic time
dt=0.1*tau;
```

Next we define position matrices for particles to have the x, y, and z coordinates of each individual particle at any time step. We position all particles, initially at `x(1,:)=6*Dnoz`, but we randomize particle placement in the y and z coordinates, although not far from the jet centre line. Complete the following script

```
%Define position matrices and initialize to zero
%The two dimensional matrix contains position as a function of time
x=zeros(nt,np);     %x Position vector [m]
y=...
z=...

%Initialize particle axial position at 6 times nozzle diameter
x(1,:)=6*Dnoz;

%Randomize the lateral position of particles
y(1,:)=0.1*Dnoz*(0.5-rand(1,np));
z(1,:)=0.1*Dnoz*(0.5-rand(1,np));
```

Next we calculate jet velocity at nozzle exit and the jet centre line velocity at a distance of six nozzle diameters. Insert the following code

```
%Calculate jet centerline velocity at the onset of self-similar behavior
U0=Vinj/(pi*(Dnoz/2)^2*tinj);

%Calculate jet center velocity at the onset of self-similar region
Uc=(U0*Bu)/((6*Dnoz/Dnoz)-4.0);
```

Now we are ready to begin iterating, first through number of particles and then through time steps. First we initialize the particle velocities. Like particle location, we randomize particle velocities. We also initialize eddy interaction, life, and crossing times with large enough values to be able run the algorithm properly at start. Complete the following script

```
%Loop through the time steps for the entire particle ensemble
%Update particle position and velocity
for j=1:np

    %Initialize particle velocity at the jet center with jet velocity

    %Calculate jet center velocity at the onset of self-similar region
    Uc=(U0*Bu)/((6*Dnoz/Dnoz)-4.0);
    %Initialize velocity to this center line velocity plus fluctuations
    vx=Uc+0.0001*Uc*(0.5-rand());
    vy=0.0001*Uc*(0.5-rand());
    vz=0.0001*Uc*(0.5-rand());
    vxold=vx;
    vyold=...
    vzold=...

    %Start with a "large" eddy interaction, life, and crossing times [s]
    %to make sure adequate if statements are executed
    ti=1;
    te=1;
    tc=1;
```

Next we iterate through time steps for each particle. The loop begins we setting the time step to a fraction of the particle relaxation time and resetting the particle accelerations. Complete the following script

```
    for i=2:nt
        %Set time step as 10% of the particle characteristic time
        dt=...

        %Reset acceleration terms
        ax=...
        ay=...
        az=...
```

Next we append acceleration in the `z` direction with gravity and buoyancy forces. Complete the following script

```
        %Append accelerations by gravity force
        az=...
        %Append accelerations by buoyancy force
        az=...
```

Next we calculate the turbulent eddy velocity fluctuations. Note that this is done only if the eddy interaction time exceeds the value of eddy life or crossing time for the most recent eddy. If this is true, then a new eddy is sampled and used and the eddy interaction time is reset. Otherwise the eddy interaction time compounds until it exceeds the value of eddy life or crossing times. Insert the following script

```
        %Calculate the turbulent fluctuations if necessary
        if ((ti >= te) || (ti >= tc))
            %Set time step as 10% of the particle characteristic time
            dt=0.1*tau;

            %Reset the eddy interaction time for the particle
            ti=0;

            %Calculate jet centreline mean velocity for the particle
            %as a function of xold, yold, and zold
            Uc=(((x(i-1,j)/Dnoz)-4.0)/(U0*Bu))^-1;
```

```
%Calculate particle's radial position from the jet center line
rold=(y(i-1,j)^2+z(i-1,j)^2)^0.5;

%Calculate fluctuating velocities for the particle
%in cylindrical coordinate
C0=7.778e-2;
C2=2.79e1;
C4=-2.02e3;
C6=4.3e5;
A=257;

up2=Uc^2*(C0+C2*(rold/x(i-1,j))^2+C4*(rold/x(i-1,j))^4+...
    C6*(rold/x(i-1,j))^6)*exp(-A*(rold/x(i-1,j))^2);

C0=5.457e-2;
C2=0.355;
C4=-4.298e1;
C6=0.0;
A=89;

vp2=Uc^2*(C0+C2*(rold/x(i-1,j))^2+C4*(rold/x(i-1,j))^4+...
    C6*(rold/x(i-1,j))^6)*exp(-A*(rold/x(i-1,j))^2);

C0=5.78e-2;
C2=-1.71;
C4=2.73e-1;
C6=0;
A=42;

wp2=Uc^2*(C0+C2*(rold/x(i-1,j))^2+...
    C4*(rold/x(i-1,j))^4+...
    C6*(rold/x(i-1,j))^6).*exp(-A*(rold/x(i-1,j))^2);

%Find angle in the polar cylindrical coordinate system
if (y(i-1,j)>=0)
    theta=asin(z(i-1,j)/rold);
else
    theta=pi-asin(z(i-1,j)/rold);
end

sqrtup2=(abs(up2))^0.5;
sqrtvp2=(abs(vp2))^0.5;
sqrtwp2=(abs(wp2))^0.5;

%Generate Gaussian distributed random fluctuating fluid velocities
%up and vp are correlated with a coefficient of 0.4
```

```
up=randn*sqrtup2;
vp=0.4*up+0.9163*randn*sqrtvp2;
wp=randn*sqrtwp2;

%Calculate turbulent kinetic energy for fluid
%at the location of particle
k=0.5*(sqrtup2^2+sqrtvp2^2+sqrtwp2^2);

%Calculate turbulent dissipation rate for fluid
%at the location of particle
C0=0.3549;
C2=11.99;
C4=-1635;
C6=43470;
A=201;

e=abs(Uc^3/(x(i-1,j))*(C0+C2*(rold/x(i-1,j))^2+...
    C4*(rold/x(i-1,j))^4+...
    C6*(rold/x(i-1,j))^6)*exp(-A*(rold/x(i-1,j))^2));

%Calculate eddy length scale and life time
le=2*(Cmu)^(3/4)*(k)^(3/2)/e;
te=2*(3/2)^0.5*(Cmu)^(3/4)*k/e;

%Calculate fluid instantaneous velocities at particle location
C0=1.0;
C2=-1.925;
C4=0.0;
C6=0.0;
A=63;

u=Uc*((C0+C2*(rold/x(i-1,j))^2+C4*(rold/x(i-1,j))^4+...
    C6*(rold/x(i-1,j))^6)*exp(-A*(rold/x(i-1,j))^2))+up;

%Transform fluctuating velocity components
%from cylindrical to cartesian coordinates
%using rotation matrix
v=vp*cos(theta)-wp*sin(theta);
w=vp*sin(theta)+wp*cos(theta);

%Calculate relative magnitude of fluid to particle velocity
urel=((u-vxold)^2+(v-vyold)^2+(w-vzold)^2)^0.5;

%Calculate eddy crossing time
if (1-le/(tau*urel))>0
    tc=-tau*log(1-le/(tau*urel));
    else
        tc=te;
    end

    %Lower dt to 0.05 of the minimum of eddy life and crossing times
    if (dt > 0.05*min(te, tc))
        dt=0.05*min(te, tc);
    end
else
    %Update eddy interaction time
    ti=ti+dt;
end
```

Note that we have accounted for a coordinate system transformation from cylindrical to cartesian system. There are also other subtleties in the above script that we leave you to investigate and analyze.

Now we include the effect of drag. Since there is no guarantee that the previous snippet of script runs at every iteration, we must calculate the necessary parameters again. Complete the following script

```
%Now include the effect of drag
%Calculate jet centreline mean velocity for the particle
%as a function of xold, yold, and zold
Uc=(((x(i-1,j)/Dnoz)-4.0)/(U0*Bu))^-1;

%Calculate fluid instantaneous velocities at particle location
C0=1.0;
C2=-1.925;
C4=0.0;
C6=0.0;
A=63;

%Calculate particle's radial position from the jet center line
rold=(y(i-1,j)^2+z(i-1,j)^2)^0.5;

u=Uc*((C0+C2*(rold/x(i-1,j))^2+C4*(rold/x(i-1,j))^4+...
    C6*(rold/x(i-1,j))^6)*exp(-A*(rold/x(i-1,j))^2))+up;

%Transform fluctuating velocity components
%from cylindrical to cartesian coordinates
%using rotation matrix
v=vp*cos(theta)-wp*sin(theta);
w=vp*sin(theta)+wp*cos(theta);
```

```
        %Calculate relative magnitude of fluid to particle velocity
        urel=((u-vxold)^2+(v-vyold)^2+(w-vzold)^2)^0.5;

        %Calculate particle's Reynolds number
        Re=...
        %Calculate the coefficient of drag based on this Reynolds number
        ...

        %Append accelerations by drag accelerations
        ax=ax-(1/mp)*pi*CD*rho*Dp^2*(vxold-u)*abs(vxold-u)/(8*Cc);
        ay=...
        az=...
```

Finally, update new velocities, positions, and old velocities. Complete the following script

```
        %Find new velocities
        vx=vxold+ax*dt;
        vy=...
        vz=...
        %Find new position
        x(i,j)=x(i-1,j)+vx*dt;
        y(i,j)=...
        z(i,j)=...
        %Update old velocities for next iteration
        vxold=vx;
        vyold=...
        vzold=...
    end
end
```

Now plot the particle positions for the entire ensemble of particles at different times in the `x-z` plane by the following script.

```
figure
plot(x(100,:),z(100,:),'ko','LineWidth',3);
hold on
plot(x(1000,:),z(1000,:),'bo','LineWidth',3);
plot(x(10000,:),z(10000,:),'ro','LineWidth',3);
plot(x(20000,:),z(20000,:),'go','LineWidth',3);
plot(x(40000,:),z(40000,:),'yo','LineWidth',3);
xlabel('x [m]','FontSize',20);
ylabel('z [m]','FontSize',20);
h_legend=legend('nt=100', 'nt=1000', 'nt=10000', 'nt=20000', 'nt=40000');
set(h_legend,'FontSize',20);
```

Make another plot for the dispersion in the `x-y` plane. In addition to plotting particle dispersion for the entire population at selected number of time steps, one can plot dispersion for a selected number of particles at all time steps in order to observe the motion of individual particles as they interact with the turbulent jet. Write a script to generate the entire particle trajectory for particles 1, 50, 100, 150, and 200, in both `x-z` and `x-y` planes. Now experiment with the script to complete the simulation matrix. You should obtain results similar to the following figures.

Comment on the following points and discuss the following questions.

- Try to explain the effect of particle size on axial penetration, i.e. for a given jet, which size particles penetrate the most in the x direction?
- Answer the above question for dispersion in y and z directions?
- Try to explain the effect of jet velocity on dispersion in the three directions.
- Comment if you can observe the effect of different eddies on dispersion of individual particles on different trajectories over the entire number of time steps. In which direction is this effect clearly seen, i.e. x, y, or z?
- In many particular instances shown, a particle is moving in the negative x direction. How is this possible?
- Regarding particle deflection in the y direction, comment if there is a preferred direction for deflection given particle's initial position in the y direction?
- Comment on computational cost and other possible limits to simulate particle dispersion using the RWM.

References

[Hussein et al., 1994] Hussein, H. J., Capp, S. P., and George, W. K. (1994). Velocity measurements in a high-Reynolds-number momentum-conserving, axisymmetric, tubulent jet. *J. Fluid Mech.*, 258:31–75.

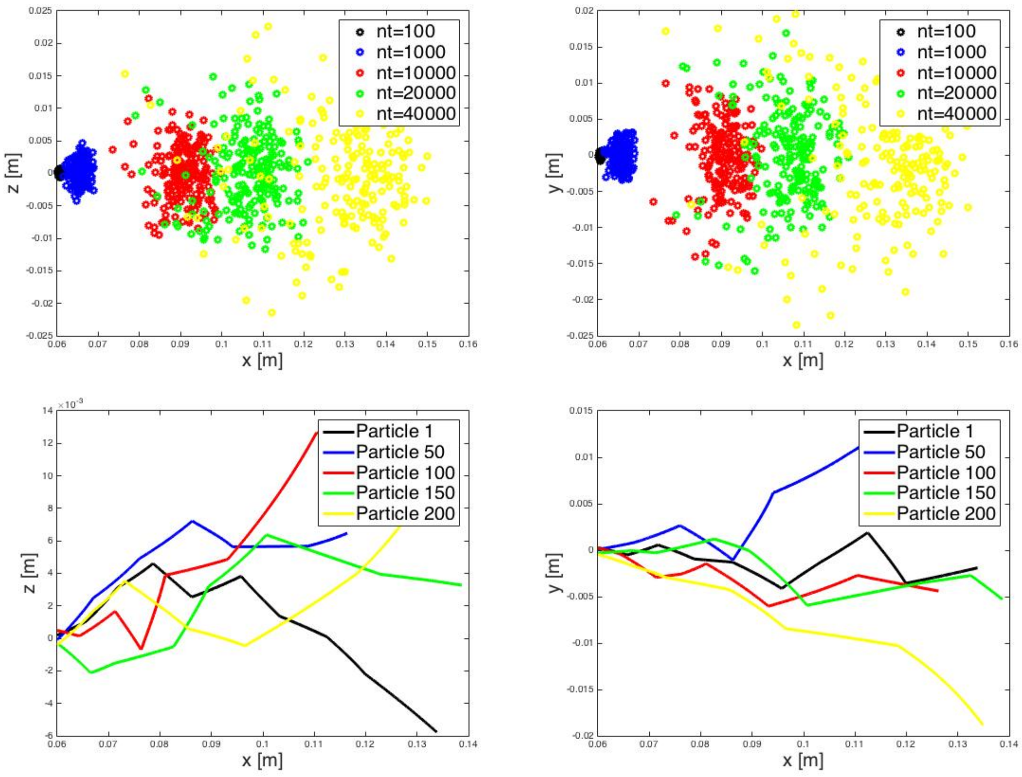


Figure 3: Particle dispersion for the entire particle ensemble at selected number of time steps (top) and particle dispersion for selected number of particles over the entire number of time steps (bottom) for `Dp=1e-6` m and `Vinj=0.5e-3`.

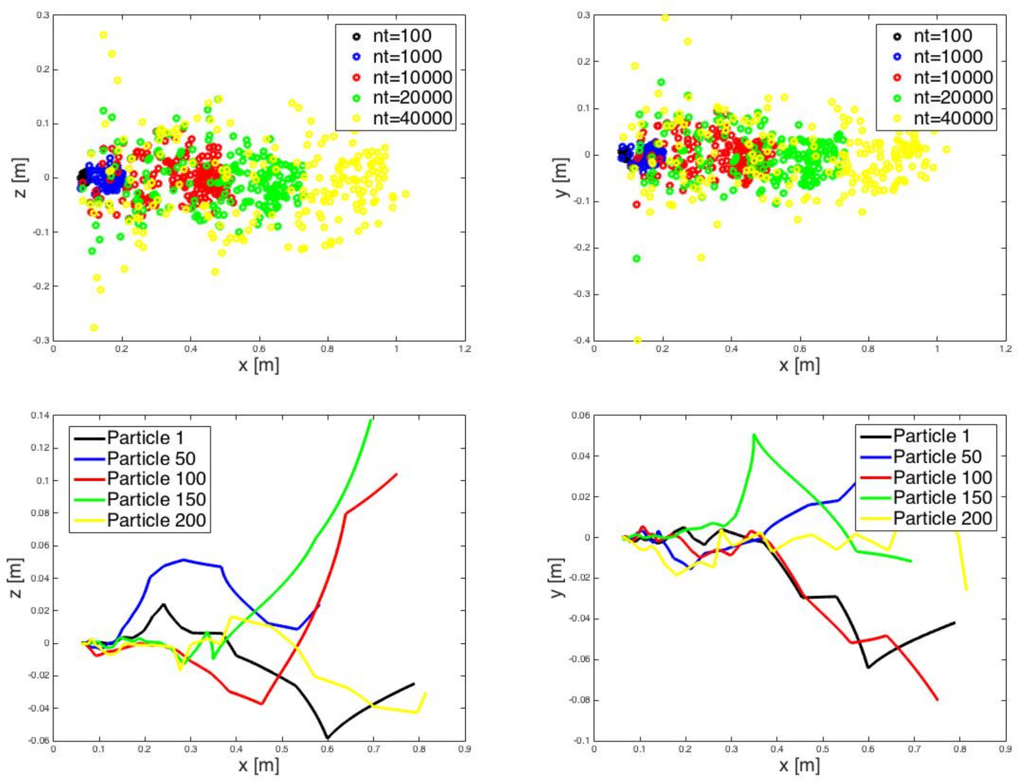


Figure 4: Particle dispersion for the entire particle ensemble at selected number of time steps (top) and particle dispersion for selected number of particles over the entire number of time steps (bottom) for `Dp=1e-5` m and `Vinj=0.5e-3`.

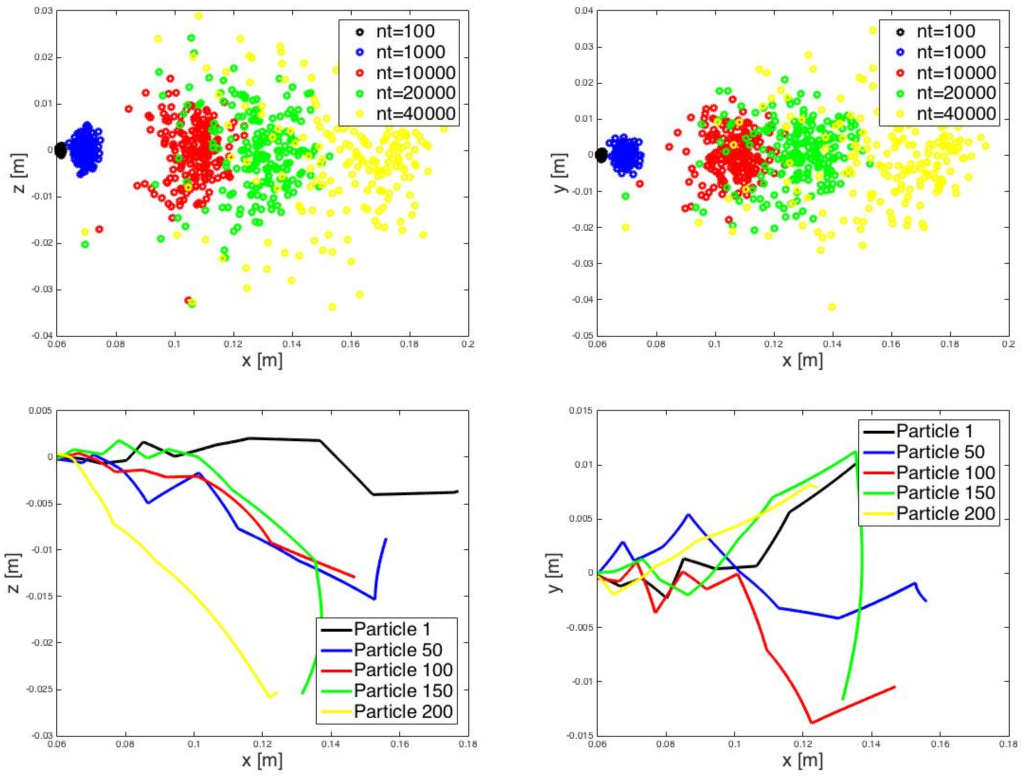


Figure 5: Particle dispersion for the entire particle ensemble at selected number of time steps (top) and particle dispersion for selected number of particles over the entire number of time steps (bottom) for `Dp=1e-6` m and `Vinj=1.0e-3`.

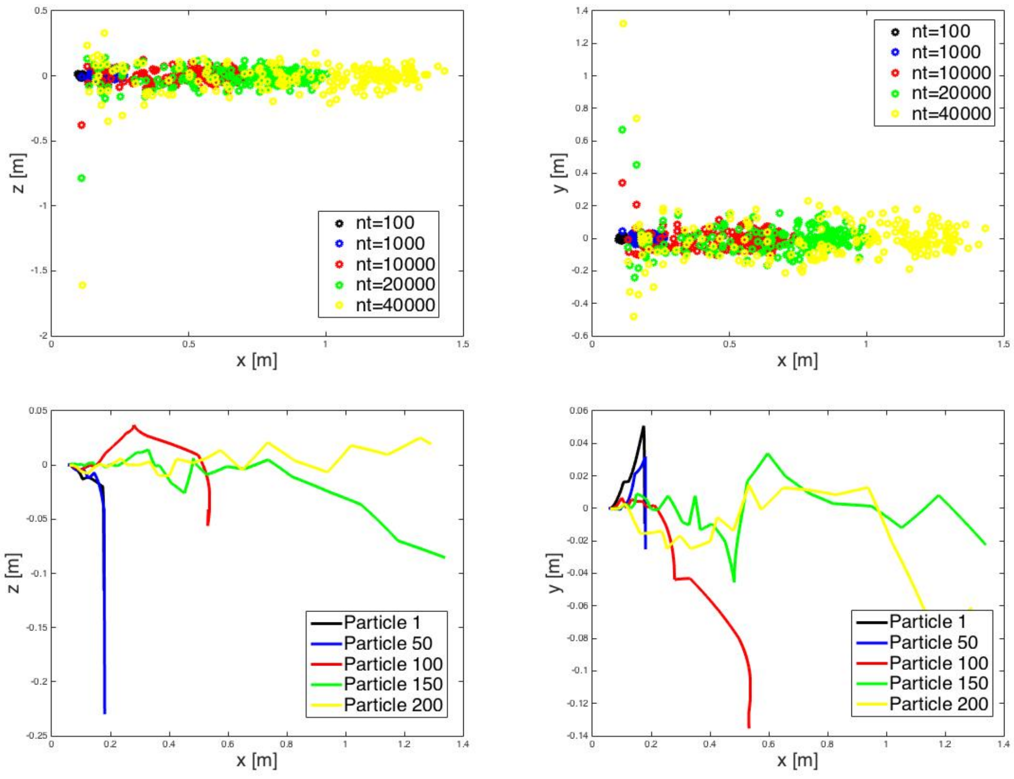


Figure 6: Particle dispersion for the entire particle ensemble at selected number of time steps (top) and particle dispersion for selected number of particles over the entire number of time steps (bottom) for `Dp=1e-5` m and `Vinj=1.0e-3`.

ENGG*4810: Control of Atmospheric Particulates

Particle Collection Efficiency of a Laminar Settling Chamber

Amir A. Aliabadi

November 13, 2017

1 Introduction

In this lab we are going to simulate the particle collection efficiency in a settling chamber. Laminar settling chambers are essentially two dimensional flow systems. For simplicity we will perform a 2D simulation. The chamber has a length of $L = 1$ m and a height of $H = 0.05$ m. The mean velocity in this chamber is $\overline{u} = 0.25$ m s^{-1}. Assuming the width of the chamber W is equal to its height H, the settling chamber Reynolds number can be calculated as

$$\text{Re}_c = \frac{4r_h\rho\overline{u}}{\mu} = \frac{4\left(\frac{H\times H}{2(H+H)}\right)\rho\overline{u}}{\mu} = \frac{H\rho\overline{u}}{\mu} \tag{1}$$

and must be checked to confirm laminar flow. From the lecture material we learn that the axial velocity profile in a laminar settling chamber can be given as

$$u = \frac{3}{2}\overline{u}\left[1 - \left(\frac{2z}{H}\right)^2\right] \tag{2}$$

Here x is the direction of the axial propagation of the flow and z is the vertical axis against which gravity acts. Also from the lectures we learn that the theoretical particle collection efficiency of a laminar settling chamber can be calculated given physical parameters such that

$$\eta(D_p) = \frac{v_t L}{\overline{u}H} \tag{3}$$

where v_t is terminal velocity given as

$$v_t = \frac{\rho_p g D_p^2}{18\mu} \tag{4}$$

This theoretical estimate is only valid if particle is small enough so that the particle Reynolds number Re < 0.1 but still large enough so that non continuum effects can be ignored, i.e. $C_c \simeq 1$.

2 Simulation Matrix

We wish to simulate particle collection efficiency for $n_p = 200$ particles of three different diameters, D_p [m], in the above settling chamber. Table below shows the simulation matrix.

Table 1: Simulation matrix

Simulation	1	2	3
D_p [m]	10e–6	15e–6	20e–6

3 MATLAB Script

As usual, we begin be defining the simulation constants and the simulation time step. We will need to choose a large enough number of time steps to ensure the seeded particles in the flow will exit the chamber or get deposited by the end of the simulation. Complete the following script

```
%LaminarSettlingChamber
%Particle collection efficiency of a laminar settling chamber

%Clear command window and memory
clc
clear

%Constants of simulation
g=9.81;         %Gravitational acceleration [m s^-2]
p=101000;       %Air pressure [Pa]
R=8.314;        %Universal gas constant [J K^-1 mol^-1]
M=28.966e-3;    %Air molecular weight [kg mol^-1]
rhop=1000;      %Particle density [kg m^-3]
Dp=10e-6;       %Particle diameter [m]
T=300;          %Air temperature [K]
mu=1.846e-5;    %Air dynamic viscosity [kg m^-1 s^-1]
rho=1.2;        %Air density [kg m^-3]
np=200;         %Number of particles in ensemble
nt=200000;      %Number of time advances
L=1;            %Settling chamber length [m] along x-axis
H=0.05;         %Settling chamber height [m] along z-axis
ubar=0.25;      %Mean velocity across the plates [m s^-1]

%Calculate mean free path, Knudsen number, slip correction, and mass
lambda=...
Kn=...
Cc=...
mp=...
```

```
%Calculate particle characteristic time
tau=...

%Set time step as 10% of the particle characteristic time
dt=...
```

Next, we write a script to calculate the Reynolds number in the settling chamber to ensure the flow is laminar. Subsequently we perform a simple calculation to find the theoretical particle collection efficiency. Complete the following script in such a way that the settling chamber Reynolds number and collection efficiency get printed on the command window of MATLAB.

```
%Calculate flow Reynolds number in settling chamber to ensure laminar flow
ReC=...

%First calculate and print the terminal velocity
vt=...

%With this velocity calculate the theoretical particle collection efficiency
etaTheory=...
```

To simulate particle collection efficiency, we need a counter variable to be able to increment it within the code every time a particle gets deposited on the bottom plate of the the settling chamber. Enter the following script to initialize a counter variable.

```
%Initialize the number of particles collected
npCollected=0;
```

Next we define position matrices for particles to have the `x` and `z` coordinates of each individual particle at any time step. Remember that this is a 2D simulation so there is no `y` position matrix. We position all particles, initially at `x(1,:)=0`, but we randomize particle placement in the `z` coordinates according to a uniform distribution. This places particles uniformly across the settling chamber. Complete the following script.

```
%Define position matrices and initialize to zero
%The two dimensional matrix contains position as a function of time
x=zeros(nt,np);     %x Position vector [m]
z=...

%Randomize the vertical (z) position of particles
z(1,:)=H*(0.5-rand(1,np));
```

Now we are ready to begin iterating, first through number of particles and then through time steps. First we initialize the particle velocities to zero, and then we calculate air velocity at the location of each particle. The only velocity component for the air is the axial velocity. Complete the following script.

```
%Loop through the time steps and entire particle ensemble
for j=1:np
    %Initialize the particle velocity components
    vxold=...
    vzold=...

    %Initialize the air velocity components at particle location
    u=(3/2)*ubar*(1-(2*z(1,j)/H)^2);
    w=...
```

Within the time advance loop, first we initialize accelerations to zero. Subsequently we append the vertical acceleration by gravity and buoyancy forces. Complete the following script.

```
    for i=2:nt
        %Reset acceleration terms
        ax=...
        az=...

        %Append accelerations by gravity force
        az=...

        %Append accelerations by buoyancy force
        az=...
```

Next we need to append the acceleration components by drag forces. First we need to calculate the relative magnitude of the air to particle velocity to be able to calculate the particle Reynolds number. Subsequently we calculate the particle Reynolds number and then the coefficient of drag. This helps us to append the acceleration terms. Complete the following script.

```
        %Calculate relative magnitude of air to particle velocity
        urel=((u-vxold)^2+(w-vzold)^2)^0.5;

        %Calculate particle's Reynolds number
        Re=...

        %Calculate the coefficient of drag based on this Reynolds number
        if (Re < 0.1)
            CD=...
        elseif (Re < 2)
            CD=...
        elseif (Re < 500)
            CD=...
        elseif (Re < 2e5)
            CD=...
        end
```

```
%Append accelerations by drag accelerations
ax=ax-(1/mp)*pi*CD*rho*Dp^2*(vxold-u)*abs(vxold-u)/(8*Cc);
az=...
```

Now we can update the particle velocities and positions given the total acceleration in each component. Complete the script below.

```
%Find new particle velocities
vx=...
vz=...

%Find new particle positions
x(i,j)=...
z(i,j)=...
```

Now we need to determine the fate of each particle in case it gets deposited at the bottom plate, i.e. collected, or in case it exists the settling chamber without being deposited. If the particle travels a distance greater than the settling chamber length, then we set its axial position equal to the length of the settling chamber, and then break the inner time loop and move on to the next particle. If the particle travels all the way to the bottom plate, then we set its vertical position equal to the position of the bottom plate, increment the counter variable, and then break the inner time loop and move on to the next particle. This coding strategy, significantly speeds up the code because we do not have to complete all the iterations. Insert the following script.

```
%If the particle has exited the settling chamber, it has not been
%collected. In this case break the inner loop and move on to the next
%particle
if (x(i,j)>L)
    x(i,j)=L;
    break;
end

%If the particle has settled at the bottom of the chamber
%Increment the npCollected and break the inner loop and move on to
%the next particle
if (z(i,j)<-H/2)
    z(i,j)=-H/2;
    npCollected=npCollected+1;
    break;
end
```

We then update the particle velocities and the air velocity at particle's location for the next time iteration. Complete the following script

```
%Update old velocities for next iteration
vxold=...
vzold=...
```

```
        %Update the air velocity component in x-axis at particle location
        u=...
    end
end
```

Finally, we can simply calculate the particle collection efficiency. Insert the following script.

```
%Now calculate the particle collection efficiency based on the simulation
etaSimulation=npCollected/np
```

As usual, we can plot representative results, such as location of all particles at selected time steps, or trajectories of individual particles at all time steps. Insert the following script.

```
figure
plot(x(1,:),z(1,:),'ko');
hold on
plot(x(1/1000*nt,:),z(1/1000*nt,:),'bo');
plot(x(1/100*nt,:),z(1/100*nt,:),'ro');
plot(x(1/10*nt,:),z(1/10*nt,:),'co');
plot(x(nt,:),z(nt,:),'yo');
axis([0 1 -0.025 0.025]);
xlabel('x [m]','FontSize',20);
ylabel('z [m]','FontSize',20);
h_legend=legend('Timestep=1', 'Timestep=1/1000*nt', 'Timestep=1/100*nt',...
    'Timestep=1/10*nt', 'Timestep=nt');
set(h_legend,'FontSize',20);

figure
plot(x(:,1),z(:,1),'ko');
hold on
plot(x(:,1/4*np),z(:,1/4*np),'bo');
plot(x(:,1/2*np),z(:,1/2*np),'ro');
plot(x(:,3/4*np),z(:,3/4*np),'co');
plot(x(:,np),z(:,np),'yo');
axis([0 1 -0.025 0.025]);
xlabel('x [m]','FontSize',20);
ylabel('z [m]','FontSize',20);
h_legend=legend('Particle=1', 'Particle=1/4*np', 'Particle=1/2*np',...
    'Particle=3/4*np', 'Particle=np');
set(h_legend,'FontSize',20);
```

Now run your code to complete the simulation matrix. You should get results similar to the following table and figures. Ensure that by the final simulation time step all particles either deposit on the bottom plate or leave the settling chamber. This can be easily verified by lack of particles in the plots at the final time step.

Table 2: Simulation results

Simulation	1	2	3
`Dp` [m]	10e–6	15e–6	20e–6
`ReC`	812.57	812.57	812.57
`etaTheoretical`	0.2362	0.5314	0.9447
`etaSimulation`	0.3100	0.5600	0.8750

Comment on the following points and discuss the following questions.

- Try to explain the effect of particle size on the collection efficiency of the settling chamber.
- We did not simulate the Brownian dispersion effect for the particles in this lab. Discuss if accounting for Brownian dispersion effects would drastically change our results.
- Discuss any potential sources for the mismatch between the theoretical and simulated particle efficiency.
- By doing this lab, would you think laminar settling chambers are suitable to remove particles with the following sizes in diameter: `Dp=1e-6, 5e-6, 10e-6, 15e-6, 20e-6, 50e-6` m?
- If laminar settling chambers are to be used to remove particles from large quantity of air, i.e. higher flow rates, what impact does this have on the size of the settling chamber?

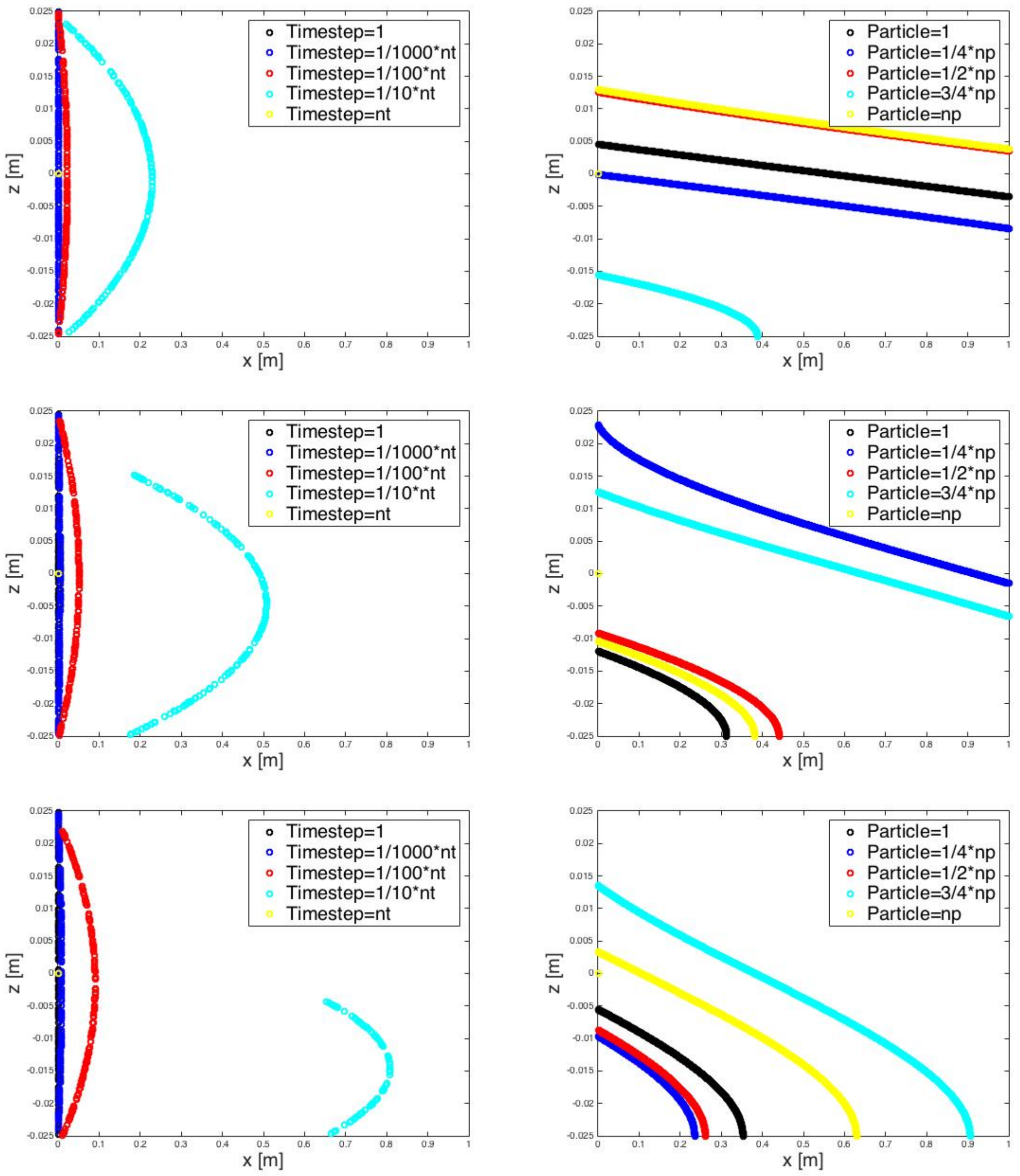


Figure 1: Particle dispersion and collection for the entire particle ensemble at selected number of time steps (left) and particle dispersion and collection for selected number of particles over the entire number of time steps (right) for `Dp=10e-6` m (top), `Dp=15e-6` m (middle), `Dp=20e-6` m (bottom).

ENGG*4810: Control of Atmospheric Particulates

Particle Collection Efficiency of a Turbulent Settling Chamber

Amir A. Aliabadi

October 30, 2018

1 Introduction

In this lab we are going to simulate the particle collection efficiency in a turbulent settling chamber. For simplicity we will perform a 2D simulation. The chamber has a length of $L = 1$ m and a height of $H = 0.05$ m. The mean velocity in this chamber is $\overline{u} = 2$ m s^{-1}. Assuming the width of the chamber W is equal to its height H, the settling chamber Reynolds number can be calculated as

$$\mathrm{Re}_c = \frac{4r_h\rho\overline{u}}{\mu} = \frac{4\left(\frac{H\times H}{2(H+H)}\right)\rho\overline{u}}{\mu} = \frac{H\rho\overline{u}}{\mu} \tag{1}$$

and must be checked to confirm turbulent flow. From the lectures we learn that the theoretical particle collection efficiency of a turbulent settling chamber can be calculated given physical parameters such that

$$\eta(D_p) = 1 - \exp\left(-\frac{v_t L}{\overline{u}H}\right) \tag{2}$$

where v_t is terminal velocity given as

$$v_t = \frac{\rho_p g D_p^2}{18\mu} \tag{3}$$

This theoretical estimate is only valid if particle is small enough so that the particle Reynolds number Re < 0.1 but still large enough so that non continuum effects can be ignored, i.e. $C_c \simeq 1$.

To simulate flow field in the settling chamber we have used the open source Computational Fluid Dynamics (CFD) software `OpenFOAM 3.0`. A standard $k-\epsilon$ turbulence model was used to give mean airflow velocity in the `x` and `z` directions, turbulent kinetic energy, and the turbulent kinetic energy dissipation rates. The computational domain was discretized in to $n_x \times n_z = 100 \times 20$ cells and shown below. Note that for this flow a $y^+ \simeq 1$ corresponds to mesh spacing near the walls at $\Delta y_{min} = 0.00012$ m resolution. Typical wall functions allow the grid spacing normal to the wall boundaries, customary defined in the y direction as opposed to our case which is in the z direction, be as large as $\Delta y = 250\Delta y_{min}$ and for our case we need a minimum of $\Delta y = 250\Delta y_{min} = 250 \times 0.00012\text{m} = 0.03\text{m}$. Our grid spacing results in $\Delta y = H/n_z = 0.05\text{m}/20 = 0.0025\text{m} << 0.03\text{m}$, as a result our grid resolution is adequate for the simulation very conservatively.

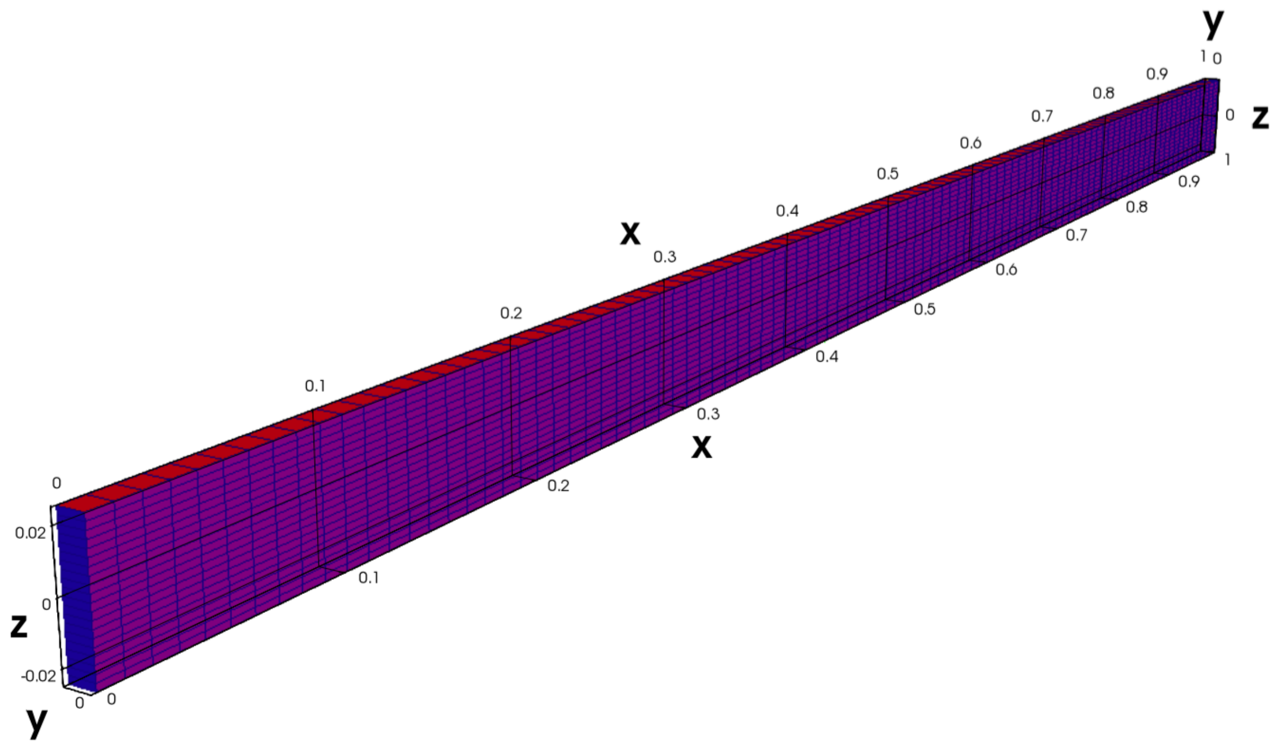
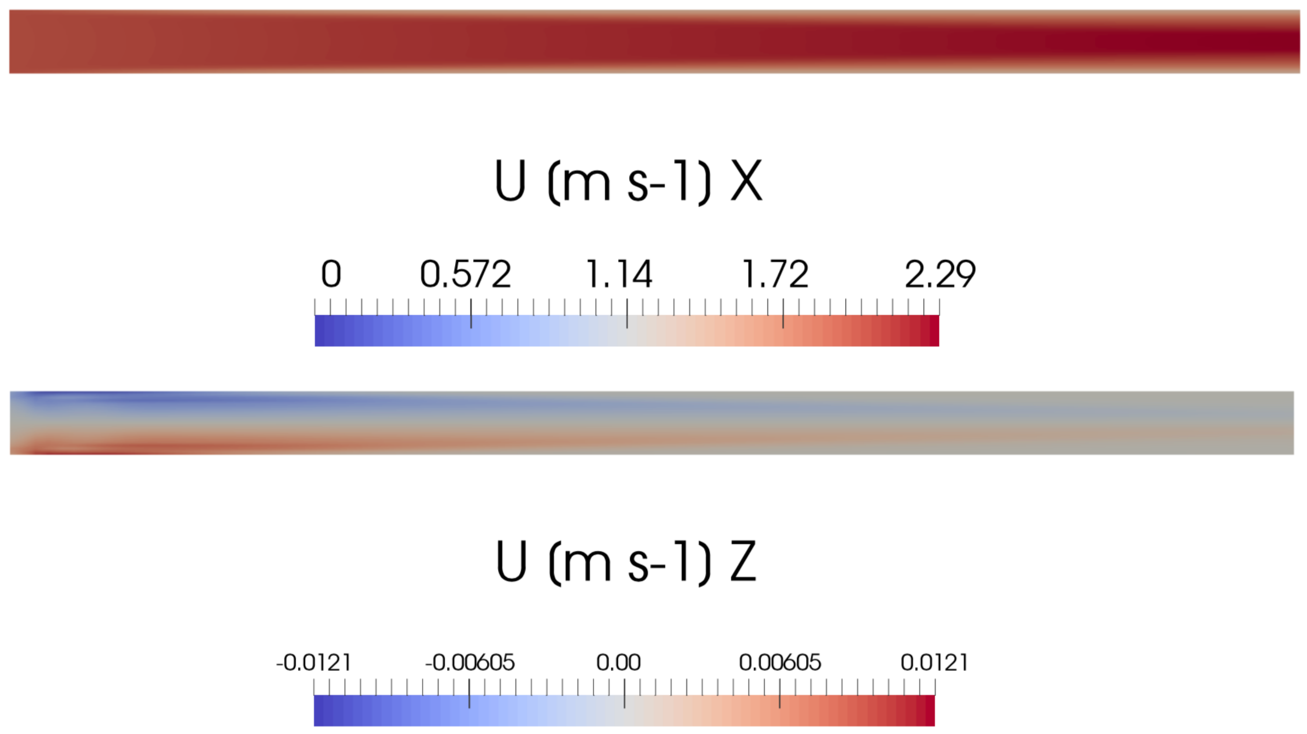


Figure 1: Computational domain for CFD simulations using `OpenFOAM 3.0`. The turbulent settling chamber is considered as a 2D geometry with flow in the `+x` direction and gravity acting in the `-z` direction.

The solutions obtained for velocity in the `x` and `z` directions is shown below. Contrary to the convenient assumption the velocity in the `x` direction is constant across the channel, we see that the simulation captures a growing boundary layer at the walls so that the airflow velocity is reduced. In addition, airflow velocity in the `z` direction is now entirely zero, but exhibits a small positive or negative value, which is nevertheless two orders of magnitude smaller than velocity in the `x` direction.

The simulated turbulent kinetic energy and kinetic energy dissipation rate is shown below. Observe that the turbulent kinetic energy and dissipation rate are highest near the walls, where turbulence is generated. In addition observe that the turbulent boundary layer grows downstream of the settling chamber.

Given these flow fields, we will simulate particle dispersion and collection using an Eddy Interaction model introduced earlier. The concentration of particles is low enough to have no major impact on airflow. With this assumption airflow and particle dispersion simulations can be performed separately. This is also known as *one way coupling*, otherwise, if the particle concentration is very high a *two way coupling* model is necessary where the airflow and particle dispersion simulations



are now separable.

2 Simulation Matrix

We wish to simulate particle collection efficiency for $n_p = 200$ particles of two different diameters, D_p [m], in the above settling chamber. Table below shows the simulation matrix.

Table 1: Simulation matrix

Simulation	1	2
D_p [m]	20e–6	40e–6

3 MATLAB Script

As usual, we begin be defining the simulation constants and the simulation time step. We will need to choose a large enough number of time steps to ensure the seeded particles in the flow will exit the chamber or get deposited by the end of the simulation. Complete the following script

```
%TurbulentSettlingChamber
%Particle collection efficiency of a turbulent settling chamber
%Using the Eddy Interaction model
```

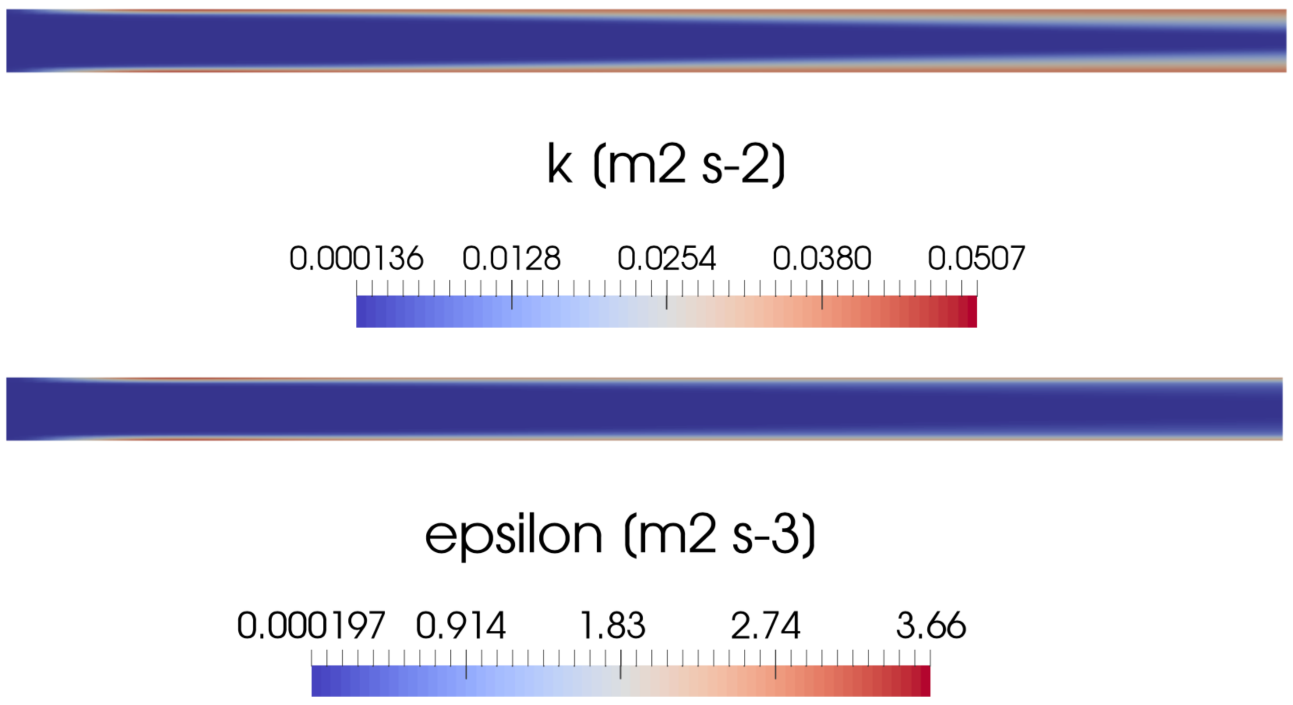


Figure 2: Solutions for CFD analysis in the turbulent settling chamber.

```
%Clear command window and memory
clc
clear

%Constants of simulation
g=9.81;          %Gravitational acceleration [m s^-2]
p=101000;        %Air pressure [Pa]
R=8.314;         %Universal gas constant [J K^-1 mol^-1]
M=28.966e-3;     %Air molecular weight [kg mol^-1]
rhop=1000;       %Particle density [kg m^-3]
Dp=40e-6;        %Particle diameter [m]
T=300;           %Air temperature [K]
mu=1.846e-5;     %Air dynamic viscosity [kg m^-1 s^-1]
rho=1.2;         %Air density [kg m^-3]
np=200;          %Number of particles in ensemble
nt=200000;       %Number of time advances
L=1;             %Settling chamber length [m] along x-axis
H=0.05;          %Settling chamber height [m] along z-axis
ubar=2;          %Mean velocity across the plates [m s^-1]
nx=100;          %Number of cells in fluid domain in the x direction
nz=20;           %Number of cells in fluid domain in the z direction
Cmu=0.09;
```

```
%Calculate mean free path, Knudsen number, slip correction, and mass
lambda=...
Kn=...
Cc=...
mp=...

%Calculate particle characteristic time
tau=...

%Set time step as 10% of the particle characteristic time
dt=...
```

Next, we write a script to calculate the Reynolds number in the settling chamber to ensure the flow is turbulent. Subsequently we perform a simple calculation to find the theoretical particle collection efficiency. Complete the following script in such a way that the settling chamber Reynolds number and collection efficiency get printed on the command window of MATLAB.

```
%Calculate flow Reynolds number in settling chamber to ensure laminar flow
ReC=...

%First calculate and print the terminal velocity
vt=...

%With this velocity calculate the theoretical particle collection efficiency
etaTheory=...
```

There is now some work involved to properly read the CFD simulation results from text files. First calculate the discretization lengths in the `x` and `z` directions. Then define the coordinates for the centre of each computational cell and store them in a matrices. Insert the following script.

```
%Calculate spatial discretization
dx=L/nx;
dz=H/nz;

%Define x and z position in fluid domain as matrices and initialize them
xDomain=zeros(nx,nz);
zDomain=zeros(nx,nz);

for i=1:nx
    for j=1:nz
        %Coordinate reference frame is at the beginning of the chamber
        xDomain(i,j)=dx/2+dx*(i-1);
        %Coordinate reference frame is half way up at the center of chamber
        zDomain(i,j)=dz/2+dz*(j-1)-0.025;
    end
```

```
end
```

To read in the text files with CFD results we need to create new matrices to hold values for the velocity components, turbulent kinetic energy, and the turbulent kinetic energy dissipation rate. We subsequently open the necessary files for reading. The information in the text files are ordered according to the computational cells from bottom left to top right. The indexing first progresses from left to right and then from bottom to top. This affects the order of iterative loops that we need to implement to read the information properly. Also note that we need to occasionally skip the brackets in the text files. We then close the files after reading them. Insert the following script.

```
%Define airflow velocities u and w (x and z components)
%in fluid domain and initialize them
%Define kinetic energy and dissipation rate in fluid domain and initialize them
%These are read from text files from left to right (+ve x)
%and then from bottom to top (+ve z)
uDomain=zeros(nx,nz);
wDomain=zeros(nx,nz);
kDomain=zeros(nx,nz);
eDomain=zeros(nx,nz);

%Read u, w, k, and e from files, if necessary skip left and right brackets
fileNameU='Lab08TurbulentSettlingChamberU.txt';
fidU=fopen(fileNameU);
fileNamek='Lab08TurbulentSettlingChamberk.txt';
fidk=fopen(fileNamek);
fileNamee='Lab08TurbulentSettlingChamberepsilon.txt';
fide=fopen(fileNamee);

%We need to loop through j first and then i; reason why?
for j=1:nz
   for i=1:nx
       %Read left bracket ( from fidU
       fscanf(fidU, '%1s', 1);
       %Read u and store it in uDomain
       uDomain(i,j)=fscanf(fidU, '%f', 1);
       %Read v and simply discard it
       fscanf(fidU, '%f', 1);
       %Read w and store it in wDomain
       wDomain(i,j)=fscanf(fidU, '%f', 1);
       %Read right bracket ) from fidU
       fscanf(fidU, '%1s', 1);

       %Read k from fidk
       kDomain(i,j)=fscanf(fidk, '%f', 1);
```

```
        %Read e from fide
        eDomain(i,j)=fscanf(fide, '%f', 1);
    end
end

%Close the files after reading the data into matrices
fclose(fidU);
fclose(fidk);
fclose(fide);
```

To simulate particle collection efficiency, we need a counter variable to be able to increment it within the code every time a particle gets deposited on the bottom plate of the the settling chamber. Enter the following script to initialize a counter variable.

```
%Initialize the number of particles collected
npCollected=0;
```

Next we define position matrices for particles to have the `x` and `z` coordinates of each individual particle at any time step. Remember that this is a 2D simulation so there is no `y` position matrix. We position all particles, initially at `x(1,:)=0`, but we randomize particle placement in the `z` coordinates according to a uniform distribution. This places particles uniformly across the settling chamber. Complete the following script.

```
%Define position matrices and initialize to zero
%The two dimensional matrix contains position as a function of time
x=zeros(nt,np);     %x Position vector [m]
z=...

%Randomize the vertical (z) position of particles
z(1,:)=H*(0.5-rand(1,np));
```

Now we are ready to begin iterating, first through number of particles and then through time steps. First we initialize the particle velocities to zero, and then we calculate air velocity at the location of each particle. The velocity components for the air is the axial and vertical directions. Note that we need to convert the particle position into indices so we can lookup up the desired CFD simulation value. Complete the following script.

```
%Loop through the time steps and entire particle ensemble
for particle=1:np
    %Initialize the particle velocity components
    vxold=...
    vzold=...

    %Initialize the air velocity components at particle location
    %We can convert particle positions to the suitable indices in order to
    %look up u, w, k, and e from the fluid domain
    iIndex=floor(x(1,particle)/dx+1);
    jIndex=floor((z(1,particle)+0.025)/dz+1);

    u=uDomain(iIndex,jIndex);
    w=...
```

We then initialize the eddy interaction, life, and crossing times necessary for the Eddy Interaction model. To start, we set these numbers to large values so we sample an eddy for the first time iteration. We then proceed to the time loop by initializing the particle accelerations and appending them with gravity and buoyancy accelerations.

```
    %Start with a "large" eddy interaction, life, and crossing times [s]
    %to make sure adequate if statements are executed
    ti=1;
    te=1;
    tc=1;

    %Now iterate through time steps
    for time=2:nt
        %Set time step as 10% of the particle characteristic time
        dt=0.1*tau;

        %Reset acceleration terms
        ax=0.0;
        az=0.0;

        %Append accelerations by gravity force
        az=az-g;
        %Append accelerations by buoyancy force
        az=az+(1/mp)*(pi/6)*Dp^3*rho*g;
```

It is necessary to check at every time step if a new eddy must be sampled when the eddy interaction time become greater than the eddy life time or the eddy crossing time. If this happens, a new eddy is sampled, and new fluctuating velocity components are added to the mean fluid velocity. These fluctuating components are computed having the information about turbulent kinetic energy and turbulent kinetic energy dissipation rate in the flow, which are available from CFD analysis. If a new eddy should not be sampled, we simply increment the eddy interaction time, until it becomes large enough for an eddy to sample. Complete the following script.

```
        %Calculate the turbulent fluctuations if necessary
        if ((ti >= te) || (ti >= tc))

            %Reset the eddy interaction time for the particle
            ti=0;

            %Look up the turbulent kinetic energy for fluid at particle location
            k=kDomain(iIndex,jIndex);
            %Look up the dissipation rate for fluid at particle location
            e=...

            %Sample turbulent fluctuating velocities
            %In 2D flow k=0.5*(up^2+wp^2) and for isotropic turbulence k=up^2
            up=randn*sqrt(k);
            wp=...

            %Append the fluid velocities with these fluctuations
            u=uDomain(iIndex,jIndex)+up;
            w=...

            %Calculate eddy length scale and life time
            le=2*(Cmu)^(3/4)*(k)^(3/2)/e;
            te=2*(3/2)^0.5*(Cmu)^(3/4)*k/e;

            %Calculate relative magnitude of fluid to particle velocity
            urel=((u-vxold)^2+(w-vzold)^2)^0.5;

            %Calculate eddy crossing time
            if (1-le/(tau*urel))>0
                tc=-tau*log(1-le/(tau*urel));
            else
                tc=te;
            end

            %Lower dt to 0.05 of the minimum of eddy life and crossing times
            if (dt > 0.05*min(te, tc))
                dt=0.05*min(te, tc);
            end
        else
            %Update eddy interaction time
            ti=ti+dt;
        end
```

We then account for the effect of drag on particle acceleration. Remember to append the velocity fluctuations to the mean velocity in the flow. This is important because a new eddy is not necessarily sampled at every time step. We subsequently calculate the coefficient of drag and them update the particle accelerations.

```
        %Now include the effect of drag
        u=uDomain(iIndex,jIndex)+up;
        w=wDomain(iIndex,jIndex)+wp;

        %Calculate relative magnitude of fluid to particle velocity
        urel=((u-vxold)^2+(w-vzold)^2)^0.5;
```

```
%Calculate particle's Reynolds number
Re=(rho*Dp*urel)/mu;

%Calculate the coefficient of drag based on this Reynolds number
if (Re < 0.1)
    CD=24/Re;
elseif (Re < 2)
    CD=(24/Re)*(1+3*Re/16+9*Re^2*log(2*Re)/160);
elseif (Re < 500)
    CD=(24/Re)*(1+0.15*Re^0.687);
elseif (Re < 2e5)
    CD=0.44;
end

%Append accelerations by drag accelerations
ax=ax-(1/mp)*pi*CD*rho*Dp^2*(vxold-u)*abs(vxold-u)/(8*Cc);
az=...
```

After finding the total particle acceleration, we are ready to update particle velocities and positions.

```
%Find new particle velocities
vx=vxold+ax*dt;
vz=...

%Find new particle positions
x(time,particle)=x(time-1,particle)+vx*dt;
z(time,particle)=...
```

Now we need to determine the fate of each particle in case it gets deposited at the bottom plate or top plate, i.e. collected, or in case it exists the settling chamber without being deposited. If the particle travels a distance greater than the settling chamber length, then we set its axial position equal to the length of the settling chamber, and then break the inner time loop and move on to the next particle. If the particle travels all the way to the bottom or top plate, then we set its vertical position equal to the position of the bottom or top plate, increment the counter variable, and then break the inner time loop and move on to the next particle. This coding strategy, significantly speeds up the code because we do not have to complete all the iterations. Insert the following script.

```
%If the particle has exited the settling chamber, it has not been
%collected. In this case break the inner loop and move on to the next
%particle
if (x(time,particle)>L)
    x(time,particle)=L;
    break;
end
```

```
        %If the particle has settled at the bottom or top of the chamber
        %Increment the npCollected and break the inner loop and move on to
        %the next particle
        if (z(time,particle)<-H/2)
            z(time,particle)=-H/2;
            npCollected=npCollected+1;
            break;
        end

        if (z(time,particle)>H/2)
            z(time,particle)=...
            npCollected=...
            break;
        end
```

We then update the particle velocities and the air velocity at particle's location for the next time iteration. Complete the following script

```
        %Update old velocities for next iteration
        vxold=...
        vzold=...

        %Update the fluid velocity components at particle location for next iteration
        iIndex=floor(x(time,particle)/dx+1);
        jIndex=floor((z(time,particle)+0.025)/dz+1);
        u=...
        w=...
    end
end
```

Finally, we can simply calculate the particle collection efficiency. Insert the following script.

```
%Now calculate the particle collection efficiency based on the simulation
etaSimulation=npCollected/np
```

As usual, we can plot representative results, such as location of all particles at selected time steps, or trajectories of individual particles at all time steps. Insert the following script.

```
figure
plot(x(1,:),z(1,:),'ko');
hold on
plot(x(1/1000*nt,:),z(1/1000*nt,:),'bo');
plot(x(1/500*nt,:),z(1/500*nt,:),'ro');
plot(x(1/100*nt,:),z(1/100*nt,:),'co');
plot(x(nt,:),z(nt,:),'yo');
axis([0 1 -0.025 0.025]);
```

```
xlabel('x [m]','FontSize',20);
ylabel('z [m]','FontSize',20);
h_legend=legend('Timestep=1', 'Timestep=1/1000*nt', 'Timestep=1/500*nt',...
    'Timestep=1/100*nt', 'Timestep=nt');
set(h_legend,'FontSize',20);

figure
plot(x(:,1),z(:,1),'ko');
hold on
plot(x(:,1/4*np),z(:,1/4*np),'bo');
plot(x(:,1/2*np),z(:,1/2*np),'ro');
plot(x(:,3/4*np),z(:,3/4*np),'co');
plot(x(:,np),z(:,np),'yo');
axis([0 1 -0.025 0.025]);
xlabel('x [m]','FontSize',20);
ylabel('z [m]','FontSize',20);
h_legend=legend('Particle=1', 'Particle=1/4*np', 'Particle=1/2*np',...
    'Particle=3/4*np', 'Particle=np');
set(h_legend,'FontSize',20);
```

Now run your code to complete the simulation matrix. You should get results similar to the following table and figures. Ensure that by the final simulation time step all particles either deposit on the bottom or top plates or leave the settling chamber. This can be easily verified by lack of particles in the plots at the final time step.

Table 2: Simulation results

Simulation	1	2
`Dp` [m]	20e–6	40e–6
`ReC`	6500	6500
`etaTheoretical`	0.111	0.377
`etaSimulation`	0.185	0.500

Comment on the following points and discuss the following questions.

- Try to explain the effect of particle size on the collection efficiency of the settling chamber.
- We did not simulate the Brownian dispersion effect for the particles in this lab. Discuss if accounting for Brownian dispersion effects would drastically change our results.
- Discuss any potential sources for the mismatch between the theoretical and simulated particle efficiency.
- Unlike the laminar settling chamber, it is possible to see particles depositing to the top plate, against the gravitational force that tends to pull particles down. Discuss how this is possible.
- It appears that most of the turbulent motion of particles occurs near the walls, as is evident from the particle dispersion plots. Discuss why is this the case, given the CFD plots.

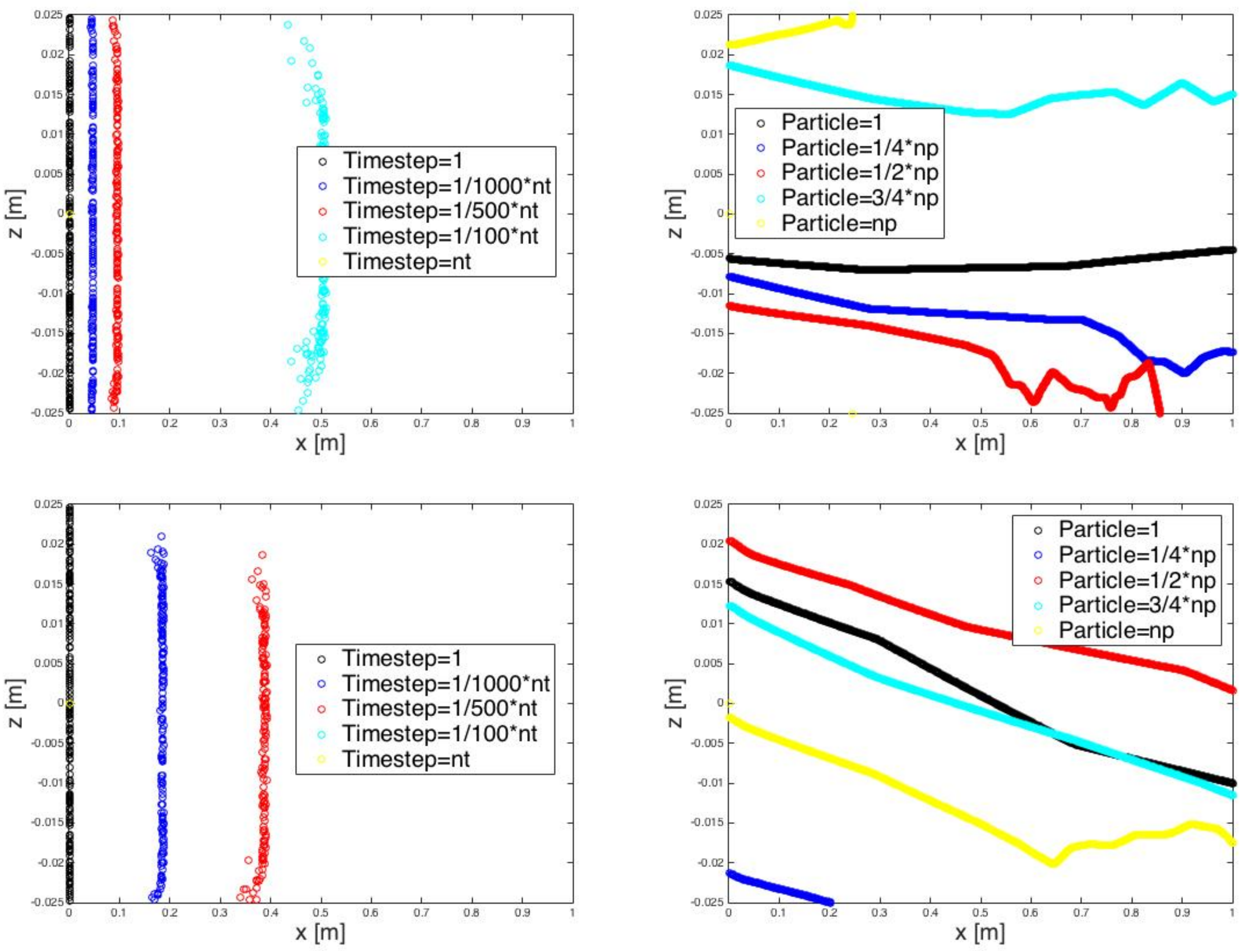


Figure 3: Particle dispersion and collection for the entire particle ensemble at selected number of time steps (left) and particle dispersion and collection for selected number of particles over the entire number of time steps (right) for `Dp=20e-6` m (top), `Dp=40e-6` m (bottom).

ENGG*4810: Control of Atmospheric Particulates

Homogeneous Water Drop Nucleation

Amir A. Aliabadi

November 13, 2017

1 Introduction

In this lab we are going to simulate homogeneous nucleation of water drops. In the lectures we learnt the Gibbs equation that tells us, for a system containing only a single species A, that the vapor pressure over a curved interface always exceeds that of the same substance over a flat surface

$$p_A = p_A^o \mathrm{exp}\left(\frac{2\sigma v_l}{kTR_p}\right) = p_A^o \mathrm{exp}\left(\frac{2\sigma M_A}{RT\rho_l R_p}\right) \tag{1}$$

This equation tells us the required vapor pressure p_A in order to be able to nucleate drops of radius R_p. We also learnt that the ratio of p_A/p_A^o is known as the saturation ratio (S), which is an important parameter in homogeneous nucleation processes. The nucleation rate J [m^{-3} s^{-1}] can be calculated in terms of measurable quantities such as

$$J = \frac{1}{\rho_l}\left(\frac{2\sigma M_A}{\pi}\right)^{1/2}\left(\frac{p_A^o}{T}\right)^2 \frac{N_{av}^{3/2}}{R^2} S^2 \mathrm{exp}\left(-\frac{16\pi M_A^2 \sigma^3 N_{av}}{3\rho_l^2 R^3 T^3 \mathrm{ln}^2 S}\right) \tag{2}$$

2 Simulation Matrix

We wish to simulate required saturation ratio as a function of drop size as well as nucleation rate as a function of saturation ratio for water at two temperatures. Table below shows the simulation matrix.

Table 1: Simulation matrix

Simulation	1	2
T [^{o}C]	25	100
p_A^o [Pa]	3.1690e3	101.3200e3
σ [J m^{-2}]	71.97e–3	58.85e–3

3 MATLAB Script

As usual, we begin be defining the simulation constants. Insert the following script

```
%HomogeneousNucleation
%Homogeneous water drop nucleation

%Clear command window and memory
clc
clear

%Constants of simulation
R=8.314;            %Universal gas constant [J K^-1 mol^-1]
MA=18.015e-3;       %Water molecular weight [kg mol^-1]
rhol=1000;          %Liquid density [kg m^-3]
k=1.38e-23;         %Boltzmann constant [J K^-1]
Nav=6.022e23;       %Avogadro's number [mol^-1]
T1=273.15+25;       %Drop-vapor mixture temperature [K]
T2=273.15+100;
sigma1=71.97e-3;    %Water surface tension (function of temperature) [J m^-2]
sigma2=58.85e-3;
pAo1=3.1690e3;      %Water vapor pressure [Pa]
pAo2=101.3200e3;
```

Next define a vector for drop sizes and then calculate the required saturation ratio for each case as a function of desired drop size for homogeneous nucleation of water. Complete the following script.

```
%Initialize water droplet radius vector
Rp=1e-9:1e-9:1e-6;

%Calculate saturation ratio for different temperatures as a function of Rp
S1=exp(2.*sigma1.*MA./(R.*T1.*rhol.*Rp));
S2=...
```

Next plot the results in the log-log plot.

```
figure
loglog(Rp,S1,'b-','LineWidth',3);
```

```
hold on
loglog(Rp,S2,'r-','LineWidth',3);
xlabel('Rp [m]','FontSize',20);
ylabel('Saturation Ratio S','FontSize',20);
h_legend=legend('T1=273.15+25 K', 'T2=273.15+100 K');
set(h_legend,'FontSize',20);
```

Next we redefine a range for saturation ratio to be able to simulate nucleation rates. Complete the following script and then plot the result in using `semilogy` command. You should get the following graphs.

```
%Now redefine S
S=1:0.1:10;

%Calculate nucleation rate
J1=(1./rhol).*(2.*sigma1.*MA./pi).^0.5.*(pAo1./T1).^2.*(Nav).^1.5./(R.^2).*S.^2.*...
    exp(-(16.*pi.*MA.^2.*sigma1.^3.*Nav)./(3.*rhol.^2.*R.^3.*T1.^3.*(log(S)).^2));

J2=...
```

Comment on the following points and discuss the following questions.

- Comment on the dependence of saturation ratio to temperature required to form drops of a certain size by homogeneous nucleation.
- A process engineer is designing a flow stream that contains pure water vapor. The pressure of the flow stream varies on somewhat an unpredictable fashion. She wants to avoid homogeneous nucleation if possible. Should she design the process to operate on low or high temperature?
- The curves in the plot for nucleation rate versus saturation ratio exhibit drastic scales in the vertical axis with critical points indicating a *burst* of nucleation. What does the drastic scale imply? for example what does `Log J` equal to 10^{-200} mean in comparison to 10^2? What is the approximate critical saturation ratio at each temperature?

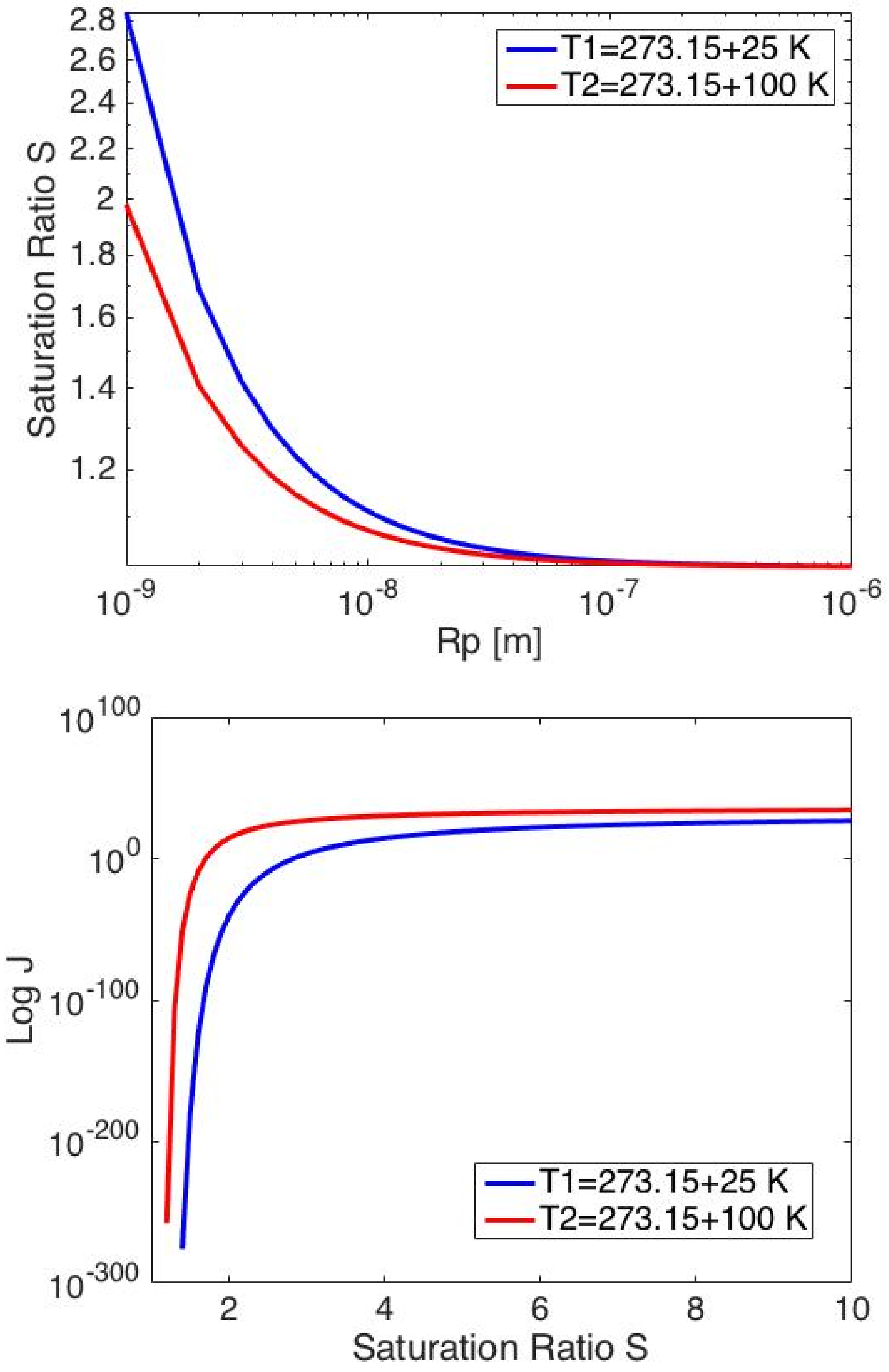


Figure 1: Required saturation ratio to form drops by nucleation of a desired diameter (top) and the functional dependence of nucleation rate to saturation ratio for different temperatures (bottom) for `T1=273.15+25` K and `T2=273.15+100` K.

ENGG*4810: Control of Atmospheric Particulates

Particle Collection Efficiency of a Turbulent Electrostatic Precipitator

Amir A. Aliabadi

November 1, 2019

1 Introduction

In this lab we are going to simulate the particle collection efficiency in a turbulent electrostatic precipitator. For simplicity we will assume particles are uniformly distributed over the cross section, particle charging is the same for all particles, and that the terminal velocity and electric field are constants over space and time. The chamber has a length of $L = 1$ m and a cross sectional radius of $r_c = 0.1$ m. The mean velocity in this chamber is $\overline{u} = 1$ m s^{-1} in the axial direction. The electric field is $E = 1 \times 10^6$ N C^{-1}. The chamber Reynolds number is given as

$$\mathrm{Re}_c = \frac{2 r_c \rho \overline{u}}{\mu} \tag{1}$$

and must be checked to confirm turbulent flow. Assuming particle Reynolds number is $\mathrm{Re} < 0.1$, the terminal or electrical migration velocity is given as

$$v_e = \frac{z_p e E C_c}{3 \pi \mu D_p} \tag{2}$$

where z_p is the number of charges on the particle, e [C] is the charge of a single electron, and E [N C^{-1}] is electric field. From the lectures we learn that the overall design equation for the turbulent flow electrostatic precipitator using these simplifying assumption gives the following particle collection efficiency.

$$\eta = 1 - \exp\left(-\frac{A v_e}{Q}\right) \tag{3}$$

2 Simulation Matrix

We wish to simulate particle electric migration velocity and collection efficiency using the above analytical formulae. We perform each simulation for a range of particle sizes in the range $D_p = 1 - 10 \times 10^{-6}$ m and number of charges on each particle in the range $z_p = 1 - 10$. Table below shows the simulation matrix.

Table 1: Simulation matrix

Simulation	1	2	3	4	5
E [N C^{-1}]	1e6	2e6	1e6	1e6	1e6
r_c [m]	0.1	0.1	0.2	0.1	0.1
L [m]	1	1	1	2	1
$\overline{u}$ [m s^{-1}]	1	1	1	1	2

3 MATLAB Script

As usual, we begin be defining the simulation constants. Complete the following script

```
%TurbulentElectrostaticPrecipitator
%Particle collection efficiency of an electrostatic precipitator
%under turbulent airflow condition

%Clear command window and memory
clc
clear

%Constants of simulation
g=9.81;         %Gravitational acceleration [m s^-2]
p=101000;       %Air pressure [Pa]
R=8.314;        %Universal gas constant [J K^-1 mol^-1]
M=28.966e-3;    %Air molecular weight [kg mol^-1]
rhop=1000;      %Particle density [kg m^-3]
T=300;          %Air temperature [K]
mu=1.846e-5;    %Air dynamic viscosity [kg m^-1 s^-1]
rho=1.2;        %Air density [kg m^-3]
e=1.602e-19;    %Electron charge [C]
E=1e6;          %Electric field [N C^-1]
rc=0.1;         %Collector radius [m]
L=1;            %Length of precipitator [m]
ubar=1;         %Precipitator air velocity [m s^-1]
```

Next, we write a script to calculate the Reynolds number in the electrostatic precipitator to ensure the flow is turbulent. Subsequently we perform a simple calculation to find the chamber electrode area and airflow rate. Complete the following script in such a way that the Reynolds number, chamber electrode area, and airflow rate get printed on the command window of MATLAB.

```
%Calculate the Reynolds number for the precipitator to ensure turbulent flow
ReC=...

%Calculate collector surface area and the airflow rate
A=...
Q=...
```

Next we create a matrix of all particle diameters and number of particle charges. This will be done using MATLAB's `linspace` and `meshgrid` commands. The first command creates a linearly spaced number of values given upper and lower bounds. The second command creates a matrix for this purpose. Insert the following script.

```
%Define particle size range for which we wish to find terminal velocity
dp=linspace(1e-6,1e-5);

%Define number of electric charges for which we wish to find terminal velocity
zp=linspace(1,10);

%Construct a matrix of particle diameters and number of charges
[Dp,Zp] = meshgrid(dp,zp);
```

We finally calculate electrical migration velocity and collection efficiency using matrix operations. Complete the following script.

```
%Calculate mean free path, Knudsen number, slip correction, and mass
lambda=mu/(0.499*p*sqrt(8*M/(pi*R*T)));
Kn=2.*lambda./Dp;
Cc=...

%Calculate the terminal or electrical migration velocity
ve=(Zp.*e.*E.*Cc)./(3.*pi.*mu.*Dp);

%Calculate the overall collection efficiency of the precipitator
eta=...
```

It is favourable to plot our results in 3D using MATLAB's `contour3` command. Insert the following script to plot your results. The argument 500 tells the script compiler to plot 500 contours equally spaced in values.

```
%Make a contour plot of electrical migration velocity as a function of Dp and Zp
figure
contour3(Dp,Zp,ve,500);
xlabel('Dp [m]','FontSize',20);
ylabel('Zp','FontSize',20);
```

```
zlabel('ve [m s^-1]','FontSize',20);

figure
contour3(Dp,Zp,eta,500);
xlabel('Dp [m]','FontSize',20);
ylabel('Zp','FontSize',20);
zlabel('eta','FontSize',20);
```

Now run your code to complete the simulation matrix. You should get results similar to the following table and figures.

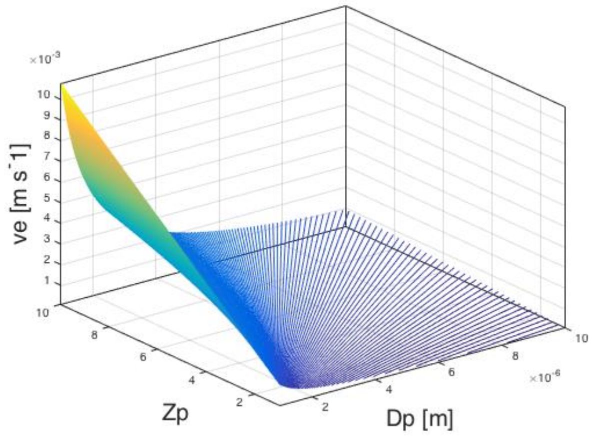
Table 2: Simulation results

Simulation	1	2	3	4	5
`ReC`	13000	13000	26000	13000	26000
`A` [m^2]	0.6283	0.6283	1.2566	1.2566	0.6283
`Q` [m^3 s^{-1}]	0.0314	0.0314	0.1257	0.0314	0.0628

Comment on the following points and discuss the following questions.

- Try to explain the effects of particle size and the number of charges on the electrical migration velocity and collection efficiency of the precipitator.

- Based on the number of simulation cases you performed, discuss which strategy is more cost effective to increase the particle collection efficiency. Assume that the largest cost is associated with the size of the precipitator and the material consumption. Assume that cost of electrical components are smaller.



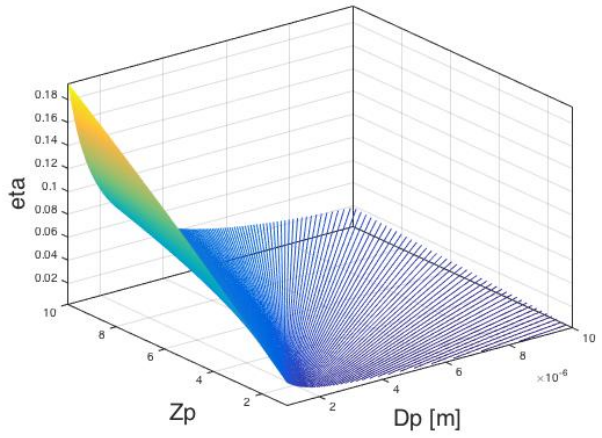


Figure 1: Electrical migration velocity and particle collection efficiency in a turbulent electrostatic precipitator for simulation case 1.

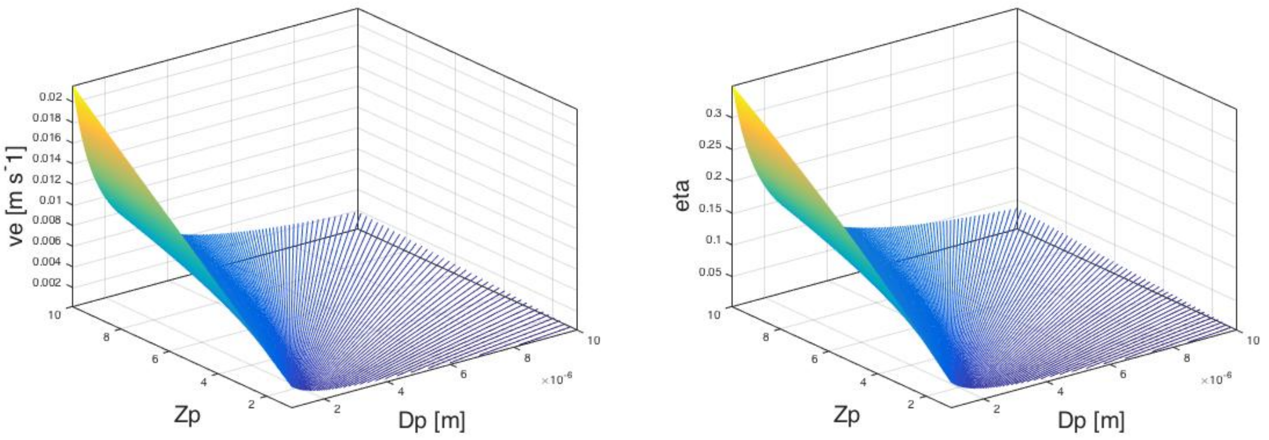


Figure 2: Electrical migration velocity and particle collection efficiency in a turbulent electrostatic precipitator for simulation case 2.

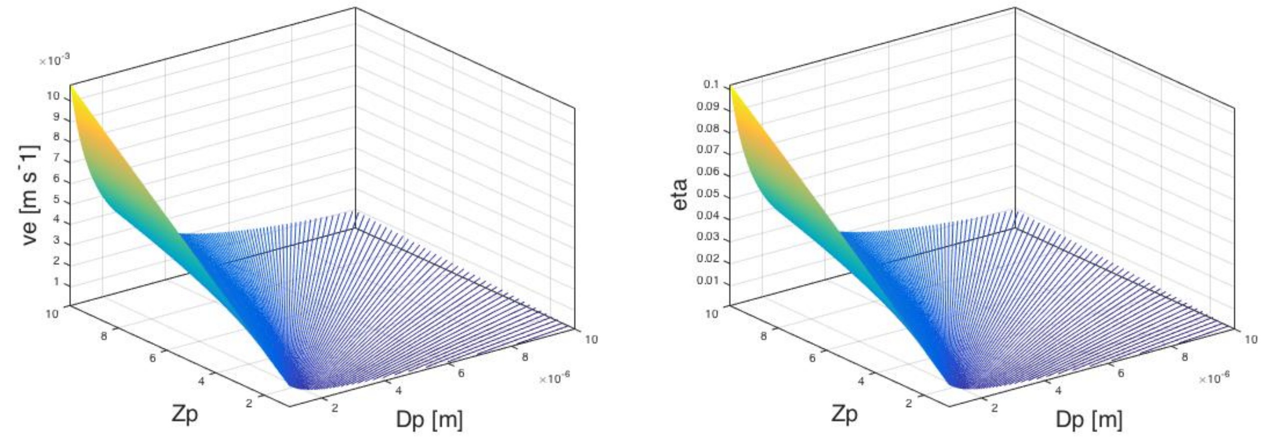


Figure 3: Electrical migration velocity and particle collection efficiency in a turbulent electrostatic precipitator for simulation case 3.

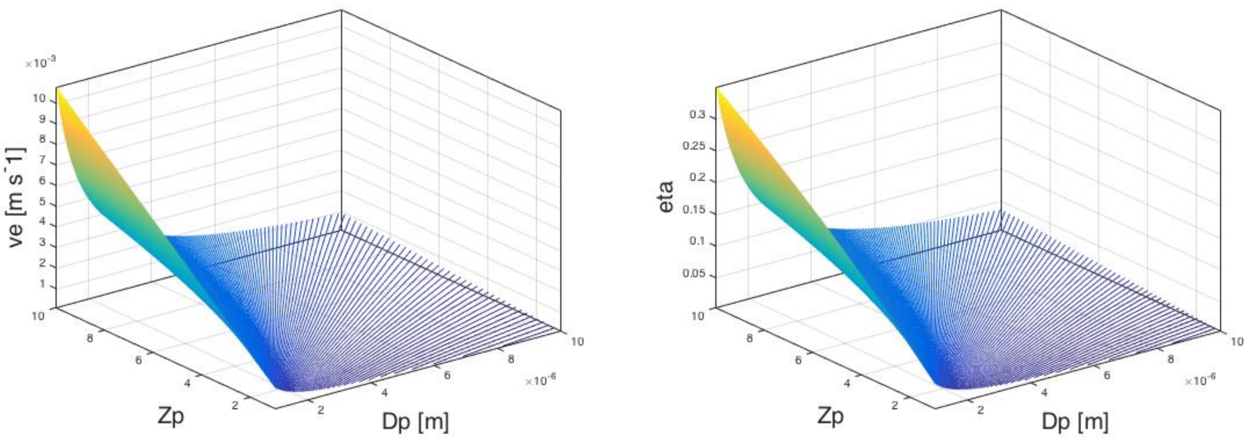


Figure 4: Electrical migration velocity and particle collection efficiency in a turbulent electrostatic precipitator for simulation case 4.

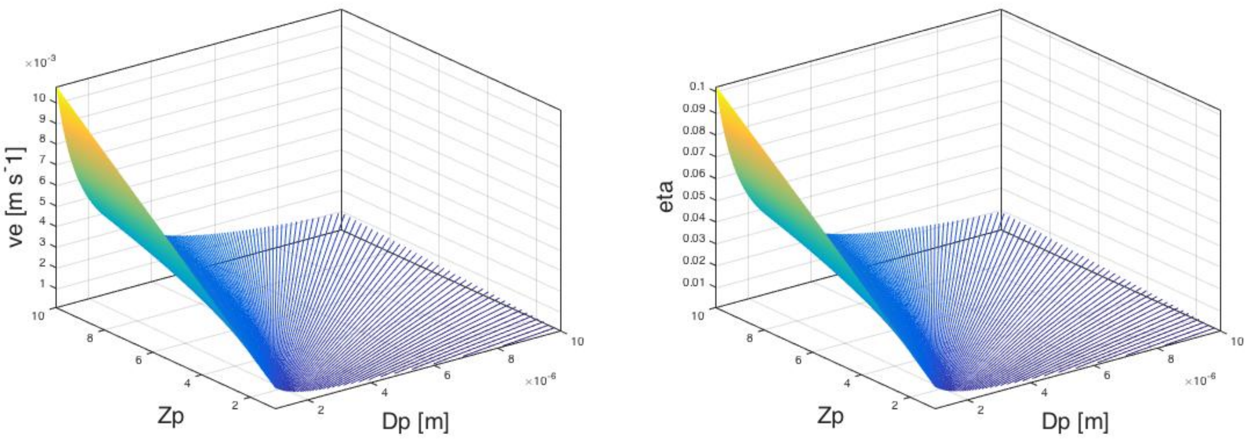


Figure 5: Electrical migration velocity and particle collection efficiency in a turbulent electrostatic precipitator for simulation case 5.