ภาคผนวก ข

โปรแกรมสำหรับการหาผลเฉลยและใช้จำลองการเคลื่อนที่ของอนุภาคยา

Exactnewok_test.m

```
clear all;
clc;
ImportReferenceData
% Set up the problem
L1 =0.065;  % [m]   length of 2-dimensional tube
L2 =0.025 ; % [m]   diameter of 2-dimensional tube
L=0.07;
L3=0.09;
initialspeed =10;          % [m/sec]   module of initial velocity vector% (2-10m/s)
maxangle = pi/6;           % [radians]  maximum angle of initial velocity vector (pi/4,pi/6, pi/8,pi/20)
maxdiameter = 0.00002;% m ; dimeter=1-20 micrometer, radius = 10 micronmeter
mindiameter = 0.000001;              %m
%maxdiameter = 0.000001;             %  dimeter=1-20 micrometer, radius = 10 micron
%mindiameter = 0.0000001;
maxtime = 4;               % [seconds]    maximum allowable time for integrating trajectory
mintime = 0;
deltatime = maxtime/5000;
TIME = mintime:deltatime:maxtime;   %  vector with all the times
NumDroplets = 300;
g = [0;-9.80];
```

```
%k =0.0011;                % [g/sec]   frictional coefficient (or damping in other contexts)
k=0.39;                    % high reynold number
                           % water droplet lubricattion friction coeff=1.1*10-3
%dpar=0.800;                        % [g/cm3] particle density powder=800 kg/m3; water droplet = %997kg/m3; aerogel=1-2 kg/m3 aerosol particle=1550kg/m3;
dpar= 997;
%dpar=1200;
%dpar=1550;
dair=1.148;                %[kg/m3]  air density o
 % [cm/sec]   the (constant in this version) velocity field of the airflow
x1final = zeros(NumDroplets,1);
tfinal  = zeros(NumDroplets,1);
hold on;
% Prepare to define the close form of the solution of Navier-stoke
for droplet=1:NumDroplets
        % Generate ONE trajectory
        % generate initial position
        x0 = [0;L3+L+5*L2/8]; %rand * L2];  % initial position starts at beginning of tube (x10 = 0)
        %  and somewhere within tube (x20 random)  +(1/2)*rand *(L2/2)
        % generate initial velocity vector
        angle = -maxangle + rand *2* maxangle;
         v0 = initialspeed * [cos(angle); sin(angle)];
        %  m=rand*maxmass;
        rad = (mindiameter+ (maxdiameter-mindiameter)*rand)/2;
        co=(3*dair*k)/(8*dpar*rad);          % original
```

```
% Initialize state variables
STATEX = zeros(2,length(TIME));  % positions of droplet
STATEV = zeros(2,length(TIME));  % speeds of droplet
index  = logical(zeros(1,length(TIME)));
%  index  = logical(length(TIME));
u = zeros(2,1);
STATEX(:,1) = x0;
STATEV(:,1) = v0;
index(1)=1;
% Integrate the trajectory
for j=2:length(TIME)
        x = STATEX(:,j-1);  % current position
        v = STATEV(:,j-1);  % current speed
        % find out the velocity of air flow in that position
        % Apply trivial Euler method... g
        dx = v;
        x = x + dx * deltatime;
        X = x(1);
        Y = x(2);
        T = TIME(j);
        u = calculateU(X,Y,T);
        dv =  g +(co)*sqrt((u-v)'*(u-v))*(u-v);
        v = v + dv * deltatime;
        STATEX(:,j) = x;
        STATEV(:,j) = v;
        if index(j-1) == 0
                index(j)=0;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        elseif x(1) <= L1/2
```

```
                    if   x(2)> (L3+L+L2/2)
                        index(j) = x(2)<=(L1/4)*sqrt(1-((x(1)-(L1/2))^2)/(L1/2)^2)+(L3+L+5*L2/4) ;   %upper line in zone 1
                    elseif x(2) < (L3+L+L2/2)
                        index(j)= x(2)>= -(L1/16)*sqrt(1-((x(1)-(L1/4))^2)/(L1/4)^2)+(L3+L); % lower line in zone 1
                    end
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                elseif x(1)<=L1
                    if x(2) > (L3+L+L2/2)
                        index(j) = x(2)<=(L1/4)*sqrt(1-((x(1)-(L1/2))^2)/(L1/2)^2)+(L3+L+5*L2/4) ;   %upper line in zone 2
                    elseif  x(2) < (L3+L+L2/2)
                        a = (L1/2+L1/12)/2;
                        h=L1/2+a;
                        index(j) = x(2)>= (L1/8)*sqrt(1-((x(1)-h)^2)/a^2)+(L3+L); % lower line in zone 2
                    end
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                elseif  x(1) <= (L1+L1/12)              % x in zone 3
                    if  x(2) > (L3+L+L2/2)
                        index(j) = x(2)<= -3*(L2/L1)*(x(1)-13*L1/12)+(L3+L+L2); % upper line in zone 3
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                    elseif x(2) <= (L3+L+L2/2)
                        a = (L1/2+L1/12)/2;
                        h=L1/2+a;
                        index(j) = x(2)>= (L1/8)*sqrt(1-((x(1)-h)^2)/a^2)+(L3+L); % lower line in zone 3
```

```
            end
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        elseif  x(1) <= (L1+L1/12+L/6+L2)   % x in the 4th part in zone 4
            if  x(2) > (L3)
                index(j) =  (((x(1)-(13*L1/12))^2)/((L/6+L2))^2 +((x(2)-L3)^2)/(L+L2)^2)< 1 &&  (((x(1)-(13*L1/12))^2)/(L/6)^2 +((x(2)-L3)^2)/L^2)> 1 ; % it run in
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            elseif  x(2) >= 2*L3/3
                index(j)= x(2)<=(4*L3/L)*(x(1)-(L1+L/6))+2*L3/3 &&   x(1)<=(L1+L/6+L2+L1/12); % right line in zone 5; % left line in zone 5
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            elseif  x(2) >= L3/3
                index(j)=x(1)>=(L1+L/6) &&   x(1)<=(L1+L/6+L2+L1/12) ; % right line in zone 6; % left line in zone 6
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            elseif  x(2) >= 0
                index(j)= x(2)>=(-4*L3/L)*(x(1)-(L1+L/6+L/12))&&   x(2)>=(4*L3/L)*(x(1)-(L1+L/6+L2)); % right line in zone 7 % left line in zone 7
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            elseif   x(2) < 0
                index(j)=0;    %bottom
            end
        end
    end
    % Determine characteristic values of the trajectory
```

```
117                     goodx1 = STATEX(1,index);
118                     goodx2 = STATEX(2,index);
119                      x1final(droplet) = max(0, goodx1(end));
120                     %x2final(droplet) = goodx2(end);
121                     %x1final(droplet) = min(goodx1(end),13*L1/12+L2+L/6);  %
SPLAT!!!
122                     x2final(droplet)= max(goodx2(end),0);  % SPLAT!!!
123                     goodt = TIME(index);
124                     tfinal(droplet) = goodt(end);
125                     plot(goodx1,goodx2,'r-');
126                     axis ([-0.01,L+L3+5*L2/4+L1/4+0.01,-
0.01,L+L3+5*L2/4+L1/4+0.01]);
127                     %  newx2final(droplet) = (L3+L+5*L2/8+L1/4) - goodx2(end);
128                     newx2final(droplet) = (L3+L+5*L2/4+L1/4) - x2final(droplet);
129         end
130         x = 0:0.001:L1;
131         % draw domain in zone 1
132         h = (L1/2);  k = L3+L+5*L2/4;  N = 256;
133         t = (0:N/2)*2*pi/N;
134         plot( (L1/2)*cos(t)+h, (L1/4)*sin(t)+k);                    % upper line
in zone 1
135         axis('square')
136         h = (L1/4);  k = L3+L;  N = 256;
137         t = (-N/2:0)*2*pi/N;
138         plot( (L1/4)*cos(t)+h, (L1/16)*sin(t)+k);                   % lower line
curve1 in zone 1
139         % draw domain in zone 2
140         v2 = (L1/2+L1/12)/2;
141         h = (L1/2+v2);  k = L3+L;  N = 256;
```

```
142         t = (0:N/2)*2*pi/N;
143         plot( v2*cos(t)+h, (L1/8)*sin(t)+k);     % lower line curve2 in zone 2
144         % draw domain in zone 3
145         x = L1:0.001:L1+L1/12;
146         x2 = (-3*L2/L1)*(x-(L1+L1/12))+L3+L+L2;          % outside line number 2
147         plot(x,x2);  hold on
148         % draw domain in zone 4
149         h = L1+L1/12;  k = L3;  N = 256;
150         t = (0:N/4)*2*pi/N;
151         plot( (L/6)*cos(t)+h, L*sin(t)+k);        % inside circle in zone 2
152         axis('square')
153         h = L1+L1/12;  k = L3;  r = L+L2;  N = 256;
154         t = (0:N/4)*2*pi/N;
155         plot( (L/6+L2)*cos(t)+h, (L+L2)*sin(t)+k);          % outside circle in zone 2
156         axis('square')
157         % draw domain in zone 5
158         x = L1+L/6:0.001:L1+L/6+L1/12;
159         x2 = (4*L3)/(L1)*(x-(L1+L/6))+2*L3/3;                    %  inside line
number 3
160         plot(x,x2);  hold on
161         x = L1+L/6+L1/12+L2;
162         x2 = 2*L3/3:0.001:L3;                % outside line number 4
163         plot(x,x2);  hold on
164         % draw domain in zone 6
165         x = L1+L/6;
166         x2 = L3/3:0.001:2*L3/3;  % inside line number 4
167         plot(x,x2);  hold on
168         x = L1+L/6+L1/12+L2;
169         x2 = L3/3:0.001:2*L3/3;  % outside line number 4
```

```
170         plot(x,x2);  hold on
171         % draw domain in zone 7
172         x = L1+L/6:0.001:L1+L/6+L1/12;
173         x2 = (-4*L3)/(L)*(x-(L1+L/6))+L3/3;    %  inside line number 3
174         plot(x,x2);  hold on
175         x = L1+L/6+L2:0.001:L1+L/6+L1/12+L2;
176         x2 = (4*L3)/(L)*(x-(L1+L/6+L1/12+L2))+L3/3;                    % outside line
number 4
177         plot(x,x2); hold on
178         hold off
179         figure
180         hist(x1final,100);
181         figure
182         hist(newx2final,100);
183         figure
184         hist(tfinal,100);
```

ImportReferenceData.m

```
1       format long
2       ImportTextFile('expressionU.txt');
3       global xREF yREF u1REF tREF
4       xREF = data(:,1);
5       yREF = data(:,2);
6       u1REF = data(:, 3:size(data,2));
7       tREF = zeros(1, size(u1REF, 2));
8       for i = 1:size(u1REF, 2)
9               tREF(i) = str2double(strrep(colheaders(i+2), 'Time=', ''));
10      end
11      clear textdata;
12      clear data;
13      clear colheaders;
14      clear i;
15      ImportTextFile('expressionV.txt');
16      global u2REF
17      u2REF = data(:, 3:size(data,2));
18      clear textdata;
19      clear data;
20      clear colheaders;
```

ImportTextFile.m

```
1       function ImportTextFile(fileToRead1)
2       %IMPORTFILE(FILETOREAD1)
3       %  Imports data from the specified file
4       %  FILETOREAD1:  file to read
5       %  Auto-generated by MATLAB
6       % Import the file
7       newData1 = importdata(fileToRead1);
8       % Create new variables in the base workspace from those fields.
9       vars = fieldnames(newData1);
10      for i = 1:length(vars)
11              assignin('base', vars{i}, newData1.(vars{i}));
12      end
```

CalculateU.m

```
function U = CalculateU(xInput, yInput, tInput)
global xREF yREF tREF u1REF u2REF
vertexIndex = 0;
meshIndex = 0;
tIndex = 0;
%#####  find tIndex (column) ####################
for i = 1:length(tREF)
        if (tREF(i) == tInput)
                tIndex = i;
        else
                % pickup 2 t column
                if i ~= 1
                        %tREF(i-1) < tInput < tREF(i)
                        if (tREF(i-1) < tInput) && (tInput <
tREF(i))
                                tIndex = [i-1, i];
                        end
                end
        end
end
%################################################
% find index (mesh or vertex) that match xInput, yInput
for i = 1:4:length(xREF)
        x_p1 = xREF(i);
        x_p2 = xREF(i+1);
        x_p3 = xREF(i+2);
        x_p4 = xREF(i+3);
        y_p1 = yREF(i);
```

```
28                  y_p2 = yREF(i+1);
29                   y_p3 = yREF(i+2);
30                  y_p4 = yREF(i+3);
31                  %##############################################
32                  % check xInput, yInput match exactly with vertex
33                  if (xInput == x_p1)
34                              if (yInput == y_p1)
35                                          vertexIndex = i;
36                                          break;
37                              end
38                  elseif (xInput == x_p2)
39                              if (yInput == y_p2)
40                                          vertexIndex = i+1;
41                                          break;
42                              end
43                  elseif (xInput == x_p3)
44                              if (yInput == y_p3)
45                                          vertexIndex = i+2;
46                                          break;
47                              end
48                  elseif (xInput == x_p4)
49                              if (yInput == y_p4)
50                                          vertexIndex = i+3;
51                                          break;
52                              end
53                  end
54                  %##############################################
55                  % compare with point2-point3 (diagonal)
56                  if (x_p2 < x_p3)
```

```
57                          if (x_p2 <= xInput) && (xInput <= x_p3)
58                                  if (y_p2 <= yInput) && (yInput <=
y_p3)
59                                          meshIndex = i;
60                                          break;
61                                  end
62                          end
63                  else
64                          if (x_p3 <= xInput) && (xInput <= x_p2)
65                                  if (y_p3 <= yInput) && (yInput <=
y_p2)
66                                          meshIndex = i;
67                                          break;
68                                  end
69                          end
70                  end
71                  %################################################
72                  % compare with point1-point4 (diagonal)
73                  if (x_p1 < x_p4)
74                          if (x_p1 <= xInput) && (xInput <= x_p4)
75                                  if (y_p1 <= yInput) && (yInput <=
y_p4)
76                                          meshIndex = i;
77                                          break;
78                                  end
79                          end
80                  else
81                          if (x_p4 <= xInput) && (xInput <= x_p1)
82                                  if (y_p4 <= yInput) && (yInput <=
```

```
y_p1)
83                                                  meshIndex = i;
84                                                  break;
85                                          end
86                                  end
87                          end
88                          %################################################
89                          % compare with point1-point2 (edge)
90                          if (x_p1 < x_p2)
91                                  if (x_p1 <= xInput) && (xInput <= x_p2)
92                                          if (y_p1 <= yInput) && (yInput <=
y_p2)
93                                                  meshIndex = i;
94                                                  break;
95                                          end
96                                  end
97                          else
98                                  if (x_p2 <= xInput) && (xInput <= x_p1)
99                                          if (y_p2 <= yInput) && (yInput <=
y_p1)
100                                                 meshIndex = i;
101                                                 break;
102                                         end
103                                 end
104                         end
105                         %################################################
106                         % compare with point2-point4 (edge)
107                         if (x_p2 < x_p4)
108                                 if (x_p2 <= xInput) && (xInput <= x_p4)
```

```
109                                     if (y_p2 <= yInput) && (yInput <=
y_p4)
110                                             meshIndex = i;
111                                             break;
112                                     end
113                             end
114                     else
115                             if (x_p4 <= xInput) && (xInput <= x_p2)
116                                     if (y_p4 <= yInput) && (yInput <=
y_p2)
117                                             meshIndex = i;
118                                             break;
119                                     end
120                             end
121                     end
122                     %################################################
123                     % compare with point4-point3 (edge)
124                     if (x_p4 < x_p3)
125                             if (x_p4 <= xInput) && (xInput <= x_p3)
126                                     if (y_p4 <= yInput) && (yInput <=
y_p3)
127                                             meshIndex = i;
128                                             break;
129                                     end
130                             end
131                     else
132                             if (x_p3 <= xInput) && (xInput <= x_p4)
133                                     if (y_p3 <= yInput) && (yInput <=
y_p4)
```

```
134                                                     meshIndex = i;
135                                                     break;
136                                             end
137                                     end
138                             end
139                             %################################################
140                             % compare with point3-point1 (edge)
141                             if (x_p3 < x_p1)
142                                     if (x_p3 <= xInput) && (xInput <= x_p1)
143                                             if (y_p3 <= yInput) && (yInput <=
y_p1)
144                                                     meshIndex = i;
145                                                     break;
146                                             end
147                                     end
148                             else
149                                     if (x_p1 <= xInput) && (xInput <= x_p3)
150                                             if (y_p1 <= yInput) && (yInput <=
y_p3)
151                                                     meshIndex = i;
152                                                     break;
153                                             end
154                                     end
155                             end
156             end
157             %#####################################################
158             if length(tIndex) == 1
159                             if (vertexIndex ~= 0)       % match with vertex
160                                     U = [u1REF(vertexIndex, tIndex)
```

```
u2REF(vertexIndex, tIndex)];
161                     else
162                             if (meshIndex ~= 0)       % match with mesh
163                                     tmpX = [xREF(meshIndex),
xREF(meshIndex + 1), xREF(meshIndex + 2), xREF(meshIndex + 3)];
164                                     tmpY = [yREF(meshIndex),
yREF(meshIndex + 1), yREF(meshIndex + 2), yREF(meshIndex + 3)];
165                                     %tmpU1 = [u1REF(meshIndex),
u1REF(meshIndex + 1), u1REF(meshIndex + 2), u1REF(meshIndex + 3)];
166                                     %tmpU2 = [u2REF(meshIndex),
u2REF(meshIndex + 1), u2REF(meshIndex + 2), u2REF(meshIndex + 3)];
167                                     tmpU1 = [u1REF(meshIndex, tIndex),
u1REF(meshIndex + 1, tIndex), u1REF(meshIndex + 2, tIndex), u1REF(meshIndex + 3,
tIndex)];
168                                     tmpU2 = [u2REF(meshIndex, tIndex),
u2REF(meshIndex + 1, tIndex), u2REF(meshIndex + 2, tIndex), u2REF(meshIndex + 3,
tIndex)];
169                                     U = PlainApprox(tmpX, tmpY, tmpU1,
tmpU2, xInput, yInput);
170                             else
171                                     % nothing match
172                                     U = [-1 ;-1];
173                             end
174                     end
175             %#########################################################
176             % has 2 t column then find U each column and interpolate
177             %#########################################################
178             else
179                     if (vertexIndex ~= 0)     % row match with vertex
```

```
180                                     % polate 2 U
181                                     U_first = [u1REF(vertexIndex, tIndex(1))
u2REF(vertexIndex, tIndex(1))];
182                                     U_second = [u1REF(vertexIndex, tIndex(2))
u2REF(vertexIndex, tIndex(2))];
183                                     % Polate with this solution
184                                     % Upolated = U_first + ( ((U_second -
U_first)*(tInput - t1)) /(t2-t1) )
185                                     uu = U_first(1) + (((U_second(1) - U_first(1))*(tInput
- tREF(tIndex(1)))) / (tREF(tIndex(2)) - tREF(tIndex(1))));
186                                     vv = U_first(2) + (((U_second(2) - U_first(2))*(tInput
- tREF(tIndex(1)))) / (tREF(tIndex(2)) - tREF(tIndex(1))));
187                                     U = [uu; vv];
188                             else
189                                     if (meshIndex ~= 0)       % row match with mesh
190                                             % find 2 U (with PlainApprox) and
polate 2 U
191                                             tmpX = [xREF(meshIndex),
xREF(meshIndex + 1), xREF(meshIndex + 2), xREF(meshIndex + 3)];
192                                             tmpY = [yREF(meshIndex),
yREF(meshIndex + 1), yREF(meshIndex + 2), yREF(meshIndex + 3)];
193                                             tmpU1_first = [u1REF(meshIndex,
tIndex(1)), u1REF(meshIndex + 1, tIndex(1)), u1REF(meshIndex + 2, tIndex(1)),
u1REF(meshIndex + 3, tIndex(1))];
194                                             tmpU2_first = [u2REF(meshIndex,
tIndex(1)), u2REF(meshIndex + 1, tIndex(1)), u2REF(meshIndex + 2, tIndex(1)),
u2REF(meshIndex + 3, tIndex(1))];
195                                             tmpU1_second = [u1REF(meshIndex,
tIndex(2)), u1REF(meshIndex + 1, tIndex(2)), u1REF(meshIndex + 2, tIndex(2)),
```

```
u1REF(meshIndex + 3, tIndex(2))];
196                                                 tmpU2_second = [u1REF(meshIndex,
tIndex(2)), u1REF(meshIndex + 1, tIndex(2)), u1REF(meshIndex + 2, tIndex(2)),
u1REF(meshIndex + 3, tIndex(2))];
197                                                 U_first = PlainApprox(tmpX, tmpY,
tmpU1_first, tmpU2_first, xInput, yInput);
198                                                 U_second = PlainApprox(tmpX, tmpY,
tmpU1_second, tmpU2_second, xInput, yInput);
199                                                 % Polate with this solution
200                                                 % Upolated = U_first + ( ((U_second -
U_first)*(tInput - t1)) /(t2-t1) )
201                                                 uu = U_first(1) + (((U_second(1) -
U_first(1))*(tInput - tREF(tIndex(1)))) / (tREF(tIndex(2)) - tREF(tIndex(1))));
202                                                 vv = U_first(2) + (((U_second(2) -
U_first(2))*(tInput - tREF(tIndex(1)))) / (tREF(tIndex(2)) - tREF(tIndex(1))));
203                                                 U = [uu; vv];
204                                     else
205                                                 % nothing match
206                                                 U = [-1; -1];
207                                     end
208                         end
209             end         % End If, Line 158
210             end         % End Function
```

PlainApprox.m

```
1       function U = PlainApprox(tmpX, tmpY, tmpU1, tmpU2, xInput, yInput)
2               %##### for U1 ################################
3               vectorA1 = [tmpX(2) - tmpX(1), tmpY(2) - tmpY(1), tmpU1(2) -
tmpU1(1)];
4               vectorB1 = [tmpX(3) - tmpX(1), tmpY(3) - tmpY(1), tmpU1(3) -
tmpU1(1)];
5               vectorN1 = cross(vectorA1, vectorB1);
6               %##### for U2 ################################
7               vectorA2 = [tmpX(2) - tmpX(1), tmpY(2) - tmpY(1), tmpU2(2) -
tmpU2(1)];
8               vectorB2 = [tmpX(3) - tmpX(1), tmpY(3) - tmpY(1), tmpU2(3) -
tmpU2(1)];
9               vectorN2 = cross(vectorA2, vectorB2);
10              %##### formula
11              %#######   u = (a1p1 + a2p2 + a3p3 - a1x - a2y) / a3
12              %#######   a1, a2, a3 --> element of normal vector
13              %#######   p1, p2, p3 --> vector [tmpX(?) tmpY(?) tmpU(?)]
14              %#######   x --> xInput
15              %#######   y --> yInput
16              u1 = (vectorN1(1)*tmpX(1) + vectorN1(2)*tmpY(1) +
vectorN1(3)*tmpU1(1) - vectorN1(1)*xInput - vectorN1(2)*yInput) / vectorN1(3);
17              u2 = (vectorN2(1)*tmpX(1) + vectorN2(2)*tmpY(1) +
vectorN2(3)*tmpU2(1) - vectorN2(1)*xInput - vectorN2(2)*yInput) / vectorN2(3);
18              %  U = [u1, u2];
19              U=[u1;u2];
20      end     % End Function
```