BIOEN 3301-004
Optimizing Particle Separation in a Spiral Channel Device
Hunter Frederiksen, Naveen Rathi, Alexander Shu, Shravan Parthasarathy
Sponsor: Dr. Bruce Gale, PhD

Abstract:

An automated method of particle separation would be helpful for biologists to accurately purify and isolate samples of desired particle size. We can use this separation on an eclectic array of particles such as chromosomes, red blood cells, white blood cells, nanoparticles, etc., to purify and sort them based on the inertial properties of the particles under certain fluid-flow conditions. Genomic research will benefit significantly from this device as they can isolate one specific whole chromosome to study specific genes within the chromosome[2]. The spiral channel device contains a rectangular channel that spirals three times, into outlets that segregate different sizes of particles. Particulate separation occurs through inertial effects consisting of two interacting forces. In a straight channel, there is a lift force that occurs when a fluid interacts with a particle, as well as a counter force from the wall. The combination of these forces push the particle up or down. Secondly, there is a set of forces that occur specifically as a result of the curved flow from the spiral device known as Dean Flow. These Dean forces push the particles against the walls of the channel, and the subsequent wall interaction force pushes back against the Dean Flow, eventually reaching an equilibrium position. This is based on the inertial properties of the particles themselves as well as the properties of the fluid the particles are suspended in. While this occurs, the particles are immobilized with respect to cross-sectional position in the channel (but are still moving through the spiral). Thus, when they approach the outlets, they are easily separated based on their inertial properties.

Using MATLAB, our group reverse-engineered a method to determine optimum dimensions of the spiral channel to give the best separation of particles, as well as the properties of the flow, given the dimensions of the particles. In our case, we simulate a separation of chromosomes by approximating them as spheres. Using a series of equations to calculate values for: the diameter of the channel, height of the channel, Reynolds number, and velocity of the fluid, among others, we will calculated the exact properties to best separate the chromosomes. We obtained a range of acceptable dimensions for a given sample of chromosomes, and optimized our calculation to provide the highest degree of separation.

Previously, the associated lab has been relying on manufacturing devices and then testing them to analyze the effectiveness in particle sorting. A program to optimize the dimensions of this device will reduce production cost as well as provide a more accurate and passive form of separation that doesn't involve fragmentation of the particles.

Introduction:

Particle separation devices are useful for researchers who require individual microscale or nanoscale particles. An example of this is a modified version of the spiral channel device from Lim C. et al.[4], for usage with chromosome separation in Dr. Bruce Gale's lab. This device is able to separate singular metaphase chromosomes from other cell materials based on a desired particle size. Applications for this usage scenarios can include genomic research such as isolating Chromosome 21 for studying trisomy 21, or Down Syndrome[2]. The spiral channel device is a

superior alternative to other methods for particle separation such as centrifugation, which may cause DNA fragmentation and subsequently, a lower yield of the desired product. On the other hand, the spiral channel device provides a maximum amount of separation with minimal particle fragmentation. The device is created using a mold containing the channel shape with the appropriate dimensions and curing Polydimethyl Siloxane (PDMS), an elastomer, onto it. Afterwards, the PDMS is plasma bonded to a slide, and the inlets and outlets are manually created. It functions by funneling the particles through an inlet, followed by a series of three spirals, and then collecting the desired samples through a set of outlets. The particles are organized in the channel based on a series of force equations called Dean Flow, which creates a tubular pinch effect[3] allowing for the separation of the particles.

In the spiral channel, particles are moving in two planes. One is in the lateral direction, in conjunction with the channel itself, and one is in a 2-D plane along the cross-section of the channel. The tubular pinch effect provides the characteristic that particles moving in a circular-shaped channel tend to migrate to certain equilibrium positions in the channel. The reason this happens is because of a balance of forces between the lift force and the wall interaction force. The curvature of the channel creates a pair of velocity profiles along the cross-sectional axis that are two countercurrents. Particles, as they flow through the channel, are caught within these countercurrents based on their inertial properties and are heaved towards the equilibrium positions. Because particles of different sizes will migrate to different positions, outlets can be conveniently positioned in order to collect them.

The ability of the device to separate the chromosomes can be quantified by the R_f value, which is the ratio of the lift force to the drag force. Ideally, we would want this value to be very small in magnitude; we found maximum separation of chromosomes occurred where there was an R_f value greater than .08[1]. This indicates that the drag force must be much larger than the lift force, which will cause the particles to migrate to the outer edges of the channel and localize around the corners of the channel. An upper limit to the R_f value was approximated at 0.15 based on information provided by the members of Dr. Gale's lab. If the ratio was lower than the value of 0.08, it would mean that the lift force was not sufficient enough to push the particle to the outer edges of the channel, and particle separation would not be achieved.

Methods:

The project consists of three main parts: the user-defined function, test cases to define the boundaries of the function, and the graphical user interface (GUI). The user-defined function, named channel_spiral, contains all the code for the optimization of the device channel dimensions and works as a standalone MATLAB function by using several Dean Flow equations as well as other force equations relevant to isolate the particles. Afterwards, the test cases are used to examine the limits and boundaries of the functionality for channel_spiral. Lastly, channel_spiral is incorporated into a GUI to create a user-friendly program that clearly outlines the results.

User-Defined Function Design:

The first step to creating the user-defined function was to establish what inputs the user would provide, and what outputs the user would obtain. Initially, we thought of outputting a

possible channel width and height to the user by simply rearranging the equations to solve for the desired dimensions, but all of the relevant Dean flow equations are dependent on the hydraulic mean diameter value (D_h), which is a singular value that takes into account that, 1) we are using a rectangular channel, and 2) the radius of curvature of the circular shape of the channel increases as you reach the outer spiral. To tackle this problem, we predefined an array of practical widths and heights and used these for our Dean Flow equations. We are able to use these values to solve for the optimum heights and widths. To do this, the user inputs the particle diameter as well as the density, viscosity, and flow rate of the medium the particle is suspended in. The function outputs all optimized channel widths, heights, and R_f values associated with each width-height pair.

Next, the second parameter we coded for was the lambda value. This determines whether or not the particle size is too large or too small for the array of predefined heights and widths described earlier. We wanted to find the values of lambda greater than or equal to 0.07, which was the value found to give the most suitable heights and widths[1]. The benefit of this is to cut

$$D_h = \frac{2hw}{h+w} \quad \lambda = \frac{a}{D_h} \geq 0.07$$

down on the run time of our program. The way this was implemented was by first examining where the lambda values were greater than 0.07, thus generating a logical matrix full of ones and zeros, with the ones describing lambda values greater than or equal to 0.07 and the zeros describing lambda values less than 0.07. Note that the logical matrix is the same size as the lambda matrix. After the logical matrix was obtained, it was dot-multiplied by the original lambda matrix to generate a more informative lambda matrix, where the wanted lambda values were obtained, and the unwanted ones set to zero.

The next equation we used to find the Dean Flow forces was Reynolds Number. This number describes the type of flow that is occurring. If the number is high, the flow is turbulent and if it is low, the flow is laminar. Because the channels are curved, the Reynolds number is different for (and usually much lower than other fluid dynamic models) the device[7]. The spiral device uses a Reynolds number in between a high and a low value because the flow can't be too turbulent or there would be no control over the separation, but not too laminar or there would be no separation at all. Once a medium Reynolds number was obtained, we knew that the previous equations were correct and we could move onto Dean Drag force.

$$Re_c = \frac{\rho U_{max} D_h}{\mu}$$

The Dean Drag Force uses three equations to summarize the drag that particles experience while fully enclosed in fluid, which in this case is cell medium, and moving in the counter-currents of the Dean Flow. It is derived via dimensional analysis, where the variables present are speed, in the form of dean velocity, which describes how fast the particles are moving in the countercurrents; within the equation, fluid viscosity, fluid density; and particle radius are considered.

$$D_e = Re_c\sqrt{\frac{D_h}{2R}} \quad U_{SF} = 4.0x102{D_e}^{1.39} \quad F_D = 3\pi\mu U_{SF} a$$

The shear gradient force describes the lateral behavior of the particle. As the particle travels through the channel, the flow will either push it up or down depending on the inertial properties of the particle and certain characteristics of the fluid such as density, velocity, etc. Because finding the exact location of each particle with respect to time would be extremely complex, we can approximate the evaluation of its degree of separation by finding equilibrium position using R_f values.

$$F_{SG} = \frac{C_{SG}\rho U_{Max}^2 a^4}{D_h^2}$$

The R_f ratio describes the equilibrium of the particle location by comparing the lift force (F_{SG}) to the drag force (F_D) values. If the R_f value is too small, the drag force will be much bigger than the lift force, resulting in the particle staying around the middle of the channel, resulting in separation not occurring. If the R_f value is too big, the lift force will be much bigger than the drag force, resulting in the particle being too close to the wall, which may cause damage to the particles and lead to an inaccurate representation.

We were able to calculate the R_f values we wanted by using the method we used to calculate the desired lambda values - that is, we multiplied our obtained R_f values by a logical matrix, resulting in another matrix that contained zeros and our desired R_f values. The reason why logical matrices were used throughout our code was because all of the matrices used had the same size, which made it extremely easy and convenient when the wanted elements needed to be extracted from the matrix. This effectively consisted of finding which R_f values in our final matrix were non-zero, and then using their indices to extract the associated widths and heights.

$$R_F = \frac{F_{SG}}{F_D}$$

Test Cases:

After creating the user-defined function, unit test cases were implemented to ensure that the user-defined function worked properly. This process involved picking reasonable values that the lab would actually use in their experiments, and manually calculating the optimal widths, heights, and the associated R_f values. The values we chose were then used as inputs for our user-defined function, and the function's outputs were compared to the outputs that were manually obtained. Five such test cases were written in a separate script, channel_spiralTest.m, and when ran, would result in a 100% pass rate. This showed that the created function was outputting proper values and would be ready to give to the lab.

Graphical User Interface Design

Finally, a graphical user interface was created to facilitate visual representation of the resulting dimensions after entering parameters inputted by the user. The GUI is divided into three sections consisting of an area for entering inputs, an area filled with tables of the results, and an area that graphs the outputs. The inputs allow the user to input the density, viscosity, and the flow rate of the medium inside the spiral channel. In addition, a drop-down menu allows the user to select from a list of seven predefined ranges of particle diameters. Users must select a range to avoid displaying an error window. Once the user clicks the calculate pushbutton, the GUI will essentially run through inputs through the user-defined function, which will output the results to the table region. This area consists of three tables for channel width, channel height, and R_f values for every particle diameter in the selected range. The table displays multiple outputs that satisfies the requirement of having the R_f value above 0.08 so that the user is able to choose what channel dimensions they want to use. Coding for the GUI is simplified by calling the user-defined function channel_spiral rather than coding all optimization operations directly into the GUI. Next, channel width is plotted against channel height to distinguish trends based on selected particle size ranges. This division effectively creates a user-friendly environment to test multiple parameter scenarios rather than merely having the user operate a user-defined function via syntax and returning outputs that would be more difficult to interpret.

Results:

Based on our array data and GUI graph results, for any given range of particle diameters, there are usually multiple channel widths that are compatible for a single channel height. In addition, for larger chromosome sizes, there are a greater number of channel dimensions that are compatible. Using sample inputs based off of the properties Dr. Gale uses in his lab of 10^{-12} g/μm^3 for density of medium, 10^{-6} g/μm*s for viscosity of medium, and $8.33x10^9$ μm^3/s for flow rate, we obtained anywhere from 19 sets of channel dimensions and R_f values for the smallest particle diameters to 296 sets for the largest particle diameters. The highest R_f values are yielded from channel dimensions that have small heights and large widths. The channel height to width ratio for dimensions with the highest R_f value is about 1:6. Decreasing the density of the medium and increases the amount of channel dimensions, while decreasing the viscosity of the medium and flow rate decreases the amount of channel dimensions that meet the minimum R_f value.

GUI Instructions:

The GUI is fairly straightforward for users since there is only one pushbutton and four possible inputs (Figure 1). After running the .m file for the GUI, users must select a particle diameter range from the dropdown menu and input the density, viscosity, and flow rate of the particle's medium in their respective correct units. Next, the user simply clicks the "Calculate" pushbutton and data will be displayed in the tables and graph.

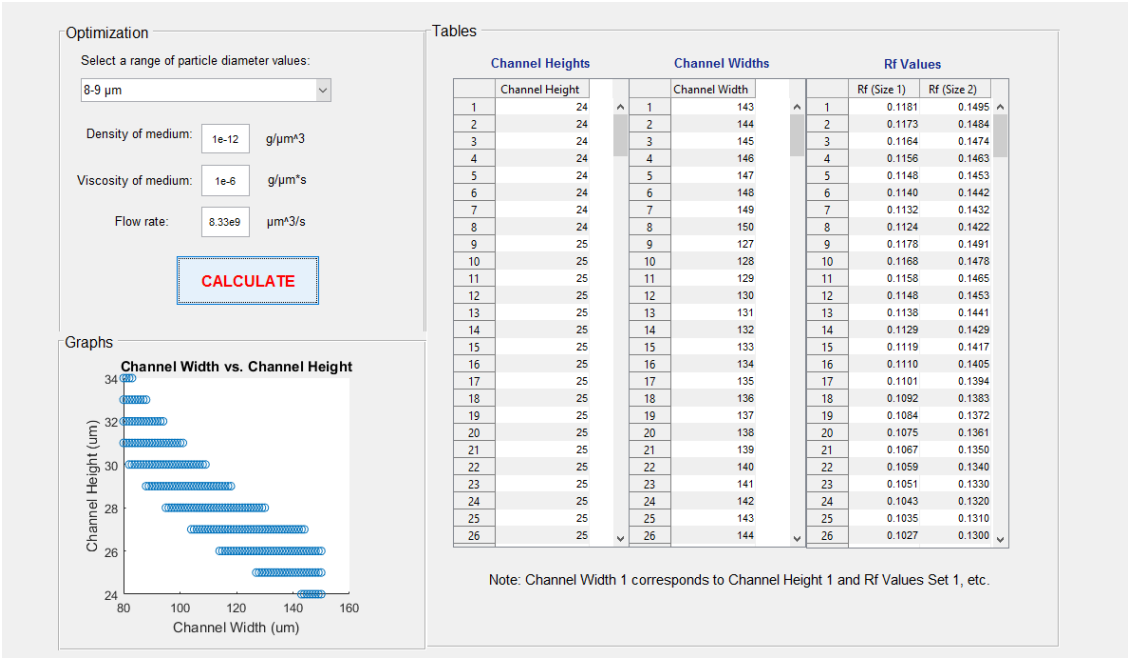

Figure 1: GUI Layout and Example Parameters

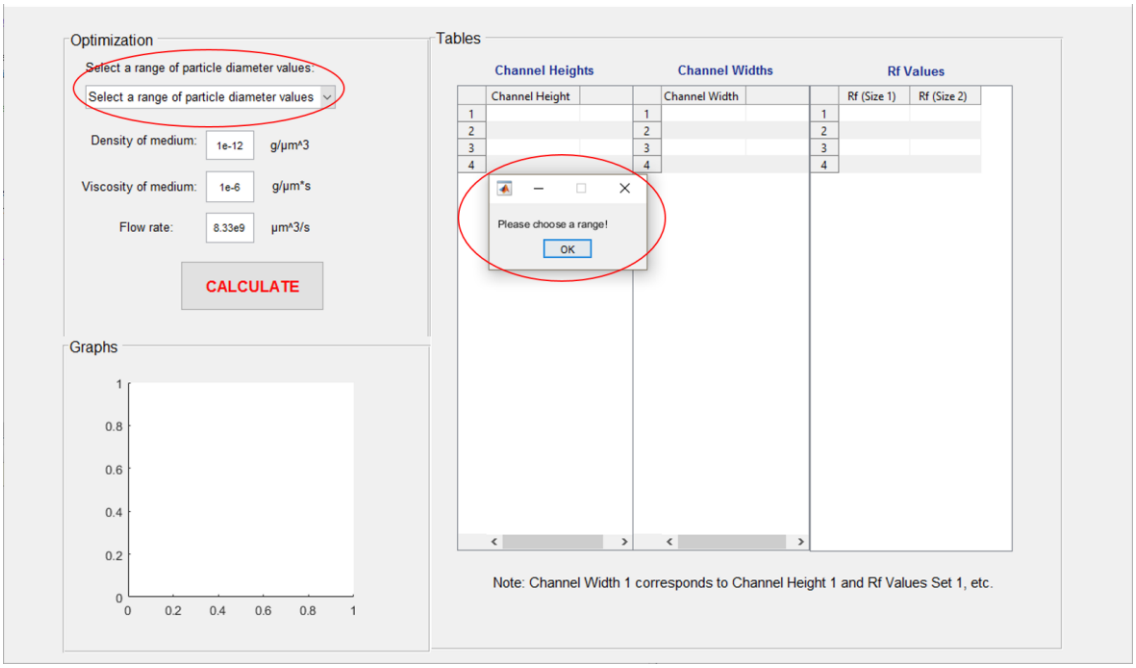

Figure 2: Be sure to choose a particle range to avoid receiving an error

Tables

Channel Heights

	Channel Height
1	21
2	21
3	21
4	21
5	21
6	21
7	21
8	21
9	21
10	21
11	21
12	21
13	21
14	21
15	21
16	21
17	21
18	22
19	22
20	22
21	22
22	22
23	22
24	22
25	22
26	22

Channel Widths

	Channel Width
1	134
2	135
3	136
4	137
5	138
6	139
7	140
8	141
9	142
10	143
11	144
12	145
13	146
14	147
15	148
16	149
17	150
18	117
19	118
20	119
21	120
22	121
23	122
24	123
25	124
26	125

Rf Values

	Rf (Size 1)	Rf (Size 2)
1	0.1099	0.1497
2	0.1091	0.1485
3	0.1083	0.1474
4	0.1075	0.1463
5	0.1067	0.1452
6	0.1059	0.1442
7	0.1051	0.1431
8	0.1044	0.1421
9	0.1036	0.1411
10	0.1029	0.1401
11	0.1022	0.1391
12	0.1015	0.1381
13	0.1008	0.1372
14	0.1001	0.1363
15	0.0994	0.1354
16	0.0988	0.1345
17	0.0981	0.1336
18	0.1092	0.1487
19	0.1082	0.1473
20	0.1072	0.1459
21	0.1062	0.1446
22	0.1053	0.1433
23	0.1043	0.1420
24	0.1034	0.1408
25	0.1025	0.1396
26	0.1017	0.1384

Note: Channel Width 1 corresponds to Channel Height 1 and Rf Values Set 1, etc.

Figure 3: The numbers in each type of table indicates that the results are part of the same group

Discussion:

Previously, the gold-standard method for manufacturing spiral channel devices was the forward-engineered method in which users decide their channel dimensions and find the particle sizes that are compatible with those dimensions. This project reverse-engineers this method to provide a list of optimized channel dimensions based on any range of particle sizes the user desires, in addition to allowing the user to customize properties of the medium that the particles travel in. This is significantly more practical than the forward-engineered problem because a researcher is more likely to have known sizes of particles, and wants to know the optimal dimensions in order to separate the particles. Thus, companies can mass manufacture the devices without using a guess and check method to find the optimized dimensions.

There were, however, some drawbacks to our reverse-engineering method. First, our code approximates the shape of chromosomes as spheres, whereas chromosomes have a cylindrical geometry. The implications of this approximation were explored by comparing simulation results to experimental data. This most likely changes the results we were getting for our Dean flow equations as the chromosomes don't behave the same as spherical particles. There could be additional movement of the chromosomes inside the channel causing the separation to be inaccurate compared to our approximated value.

Second, the radius value was approximated to be the same throughout all the equations even though there are multiple radii in the spiral device. Based on data provided by the lab, we

used the innermost radius of the device because maximum separation occurs during the first spiral, and the separation during the proceeding spirals is necessary for leftover unorganized particles[8]. It's important to note the difference between the largest radius of the device and smallest is five millimeters for most of the devices used in Dr. Gale's lab[3].

The GUI can be improved by containing custom ranges that the user can choose rather than only having particle diameter ranges which are one micrometer apart. We used a range of values that are predefined, however, as it is more applicable for a mass manufacturing context. Another useful feature would be to have an automatic unit converter for density, viscosity, and flow rate inputs so that the user can choose what units they want to input their parameters in.

When comparing the results between the simulation and experimental data, it was discovered that, unfortunately, our model did not accurately represent the maximal separation of particles by the device. The experimental data suggested that maximal separation occurs when the ratio of height-to-width is 1:3, specifically 50 micrometers x 150 micrometers. Although some of our data approached that ratio, we did not actually obtain the ratio. Conceptually, this ratio makes sense because too small of a ratio would not provide enough space for distinct countercurrents to occur, and effective particle separation would not be obtained. The reason this happens is because of our approximation of chromosomes as spheres. Because the orientation of the chromosome would likely change as it is flowing through the channel, there is a very high chance that the symmetrical dean flow would be impacted, and this would cause the flow profile in the cross-sectional plane to change. To remedy this situation, we can either multiply one of the user-defined inputs by a certain factor which will account for this geometry, or use the dimensions of the chromosomes themselves. If we come to the conclusion that a 1:3 ratio of height to width leads to the greatest degree of separation, we can reformat our code to solve for other variables such as the number of spirals or the radius of curvature, which will also produce varying degrees of separations and are easily adjustable during the manufacturing process of the device.

References:

[1] Amini, H., Lee, W., Carlo, D. Inertial microfluidic physics. Lab Chip, 14, 2738. **2014.** Retrieved from: http://pubs.rsc.org/en/Content/ArticleLanding/2014/LC/c4lc00128a#!divAbstract

[2] Dougherty, A., Ruf, S., Mulligan, C., Fisher, E., *An aneuploidy mouse strain carrying human chromsome 21 with down syndrome phenotypes.* Science 309(5), 2033-2037. **2007.** Retrieved from: http://www.ncbi.nlm.nih.gov/pubmed/16179473

[3] Gale, B., *Chromosome Separation Using Inertial Microfluidics.* Grant Proposal. **2015**

[4] Lim, C., Han, J., Hou, H., Bhagat, A., Vliet, K., Lee, W., *Microfluidics Sorter For Cell Detection and Isolation.* **2011**. Retrieved from: http://www.google.com/patents/US20130130226#backward-citations

[5] Son, J., Murphy, K., Samuel, R., Gale, B., Carrell, D., Hotaling, J., *Non-motile sperm cell separation using spiral channel. Anal. Methods, 7, 8041-8047, **2015.** Retrieved from: http://pubs.rsc.org/en/Content/ArticleLanding/AY/2015/C5AY02205C#!divAbstract*

[6] Al-Halhouli, A., Alshare, A., Mohsen, M., Matar, M., Dietzel, A., Buttgenbach, S. *Passive Micromixers with Interlocking Semi-Circle and Omega-Shaped Modules: Experiments and Simulations.* Micromachines, 6, 953-968, **2015.** doi 10.3390/mi6070953

[7] Marsa Taheri, BIOEN 3301 TA

[8] Bruce Gale, PhD, Project Sponsor

Appendix:

User-defined function:

```
function[width, height, Rftable] = channel_spiral(p, u, ap, flow) % density,
viscosity, particle diameter, flow rate
% Function that returns possible dimensions of a spiral channel used for
% particle separation.
% Function takes in four inputs: density, viscosity, particle radius, and
% flow rate of the solution. Function has three outputs: possible heights,
% possible widths, and satisfactory Rf values

% Varying heights and widths are useful for mass manufacturing
h = [10:1:90]; % possible heights of device (um)
w = [80:1:150]'; % possible widths of device (um)

% ap  = 2*[3:4];
% ap = ap/2; % particle radius in um
% flow = 0.5*(10^12)/60; % flow rate (um^3/s)
% p = 1/(10^12); % density (rho) of water (g/um^3)
% u = 0.01/(10^4); % viscosity of water (g/(um*s))
r_c = 0.5*10^4; % radius of curvature in um

area=w*h; % calculates areas from all possible combinations of heights and
widths

% calculates D_h (um), which is the hydraulic mean diameter, with the
equation D_h = 2hw/(h+w)
for ii=1:length(h)
    for jj=1:length(w)
        D_h(jj,ii)=2*area(jj,ii)./(h(ii)+w(jj));
    end
end

% calculates lambda values, which is particle radius divided by the D_h
values
lambda1 = ap(1)./D_h; % (unitless)
% dimensions will be considered only if their lambda value is greater than or
equal to 0.07
wanted_lambdas1 = lambda1.*(lambda1 >= 0.07);  % generates matrix of ones and
zeros

wanted_Dh1 = D_h.*(lambda1 >= 0.07); % all non-zero values are D_h values
worth considering

% Velocity
Uf1 = (flow)./(area); % (cm/s)  *60cm/min_  1ml/min:1600, 2ml/min:3200
3ml/min:4800
%Uf2 = Uf;
Um1 = Uf1.*(3/2); % (cm/s) maximum channel velocity

% Reynold's number (unitless) = (density * max channel velocity * hydraulic
mean diameter)/viscosity
Re1 = (p.*Um1.*wanted_Dh1./u);  %*(10^4/60); % Reynolds number.
```

```
% Dean number (unitless) = Reynold's number * sqrt(D_h/(2*radius of
curvature))
De1 = Re1.*sqrt(wanted_Dh1./(2*r_c));

% Average Dean velocity (unitless) = (4*10^-4) * (Dean velocity)^1.39
U_sf1 = 4.0E4.*De1.^1.39;
% secondary flow drag force. from Stokes drag force, calculated from
% dean

% Dean drag force (g/cm^2) = 3*pi*density*average Dean velocity*particle
radius
F_d1 = 3*pi.*u.*U_sf1.*ap(1);

% Lift coefficient (unitless) = 0.05*hydraulic mean diameter / particle
radius
C_sg1 = (0.05*wanted_Dh1./ap(1));

% Lift force (g*um/s^2) = lift coefficient*density*(max channel velocity)^2 *
(particle radius)^4/(hydraulic mean diameter)^2
F_sg1 = ((C_sg1.*p.*(Um1).^2.*(ap(1)).^4)./(wanted_Dh1).^2);

% ratio (cm^2/s^2) = Lift force/Dean drag force
R_f1 = F_sg1./F_d1;

% we're looking for the R_f values that are greater than 0.08
wanted_Rf1 = ((R_f1 >= 0.08) & (R_f1 <= 0.15)); % logical matrix
R_f1 = double(wanted_Rf1.*R_f1);
[a1,b1] = find((R_f1 ~= 0) & (~isnan(R_f1)));
c1 = find((R_f1 ~= 0) & (~isnan(R_f1)));

% compiling all the dimensions whose corresponding R_f value is greater than
0.08
width1 = w(a1); % width output in cm
height1 = h(b1)'; % height output in cm

% Rftable1 = R_f(a1); % table displaying values of ratio greater than or
equal to .08
% Rftable2 = R_f(b1);
for kk = 1:size(a1)
    Rftable_1(kk,1) = R_f1(c1(kk));
end

% calculates lambda values, which is particle radius divided by the D_h
values
lambda2 = ap(2)./D_h; % (unitless)
% dimensions will be considered only if their lambda value is greater than or
equal to 0.07
wanted_lambdas2 = lambda2.*(lambda2 >= 0.07);  % generates matrix of ones and
zeros

wanted_Dh2 = D_h.*(lambda2 >= 0.07); % all non-zero values are D_h values
worth considering
```

```
% Velocity
Uf2 = (flow)./(area); % (cm/s)  *60cm/min_  1ml/min:1600, 2ml/min:3200
3ml/min:4800
%Uf2 = Uf;
Um2 = Uf2.*(3/2); % (cm/s) maximum channel velocity

% Reynold's number (unitless) = (density * max channel velocity * hydraulic
mean diameter)/viscosity
Re2 = (p.*Um2.*wanted_Dh2./u);  %*(10^4/60); % Reynolds number.

% Dean number (unitless) = Reynold's number * sqrt(D_h/(2*radius of
curvature))
De2 = Re2.*sqrt(wanted_Dh2./(2*r_c));

% Average Dean velocity (unitless) = (4*10^-4) * (Dean velocity)^1.39
U_sf2 = 4.0E4.*De2.^1.39;
% secondary flow drag force. from Stokes drag force, calculated from
% dean

% Dean drag force (g/cm^2) = 3*pi*density*average Dean velocity*particle
radius
F_d2 = 3*pi.*u.*U_sf2.*ap(2);

% Lift coefficient (unitless) = 0.05*hydraulic mean diameter / particle
radius
C_sg2 = (0.05*wanted_Dh2./ap(2));

% Lift force (g*um/s^2) = lift coefficient*density*(max channel velocity)^2 *
(particle radius)^4/(hydraulic mean diameter)^2
F_sg2 = ((C_sg2.*p.*(Um2).^2.*(ap(2)).^4)./(wanted_Dh2).^2);

% ratio (cm^2/s^2) = Lift force/Dean drag force
R_f2 = F_sg2./F_d2;

% we're looking for the R_f values that are greater than 0.08
wanted_Rf2 = ((R_f2 >= 0.08) & (R_f2 <= 0.15)); % logical matrix
R_f2 = double(wanted_Rf2.*R_f2);
[a2,b2] = find((R_f2 ~= 0) & (~isnan(R_f2)));
c2 = find((R_f2 ~= 0) & (~isnan(R_f2)));

% compiling all the dimensions whose corresponding R_f value is greater than
0.08
width2 = w(a2); % width output in cm
height2 = h(b2)'; % height output in cm

% Rftable2 = R_f(a2); % table displaying values of ratio greater than or
equal to .08
% Rftable2 = R_f(b2);
for kk = 1:size(a2)
    Rftable_2(kk,1) = R_f2(c2(kk));
end

% generates matrix consisting of overlapping dimensions and their
% corresponding Rf values
overlap = []; % initializes overlap
```

```
for tt = 1:length(width1)
    for ss = 1:length(width2)
        % compares both a height and its paired width with another height and
its paired width
        if (width1(tt) == width2(ss)) && (height1(tt) == height2(ss))
            % adds the width, height, and the two Rf values generated by
            % the pair to the overlap matrix
            overlap = [overlap; width2(ss),height2(ss), Rftable_1(tt),
Rftable_2(ss)];
        end
    end
end

% assigns function outputs to proper columns of the overlap matrix
if isequal(overlap, [])
    width = overlap;
    height = overlap;
    Rftable = overlap;
else
    width = overlap(:,1);
    height = overlap(:,2);
    Rftable = round(overlap(:,3:4),4);
end
```

Test Cases:

```
% Test case suite for channel_spiral.m
% Tests the function with various values of parameters. If all test cases
% pass, "All tests passsed. Good job" will be displayed in the command
% window. Otherwise, "Not all tests have passed." will be displayed.
% NOTE: channel_spiral.m must be in the same folder as channel_spiralTest.m

pRange = [0.8:0.1:1.2]/10^12; % 5 values
uRange = [0.08 0.008]./(10^4); % 5 values
diameter = [3:10]; % 7 values
flowRange = [0.1:0.1:1].*(10^12)/60; % 10 values

% Case 1
[a1 b1 c1] =
channel_spiral(pRange(1),uRange(1),diameter(1:2)/2,flowRange(1));
ex_out1_1 = [123  124  125  126  127  128  129  130  86  87  88  89  90]';
ex_out1_2 = [10 10 10 10 10 10 10 10 11 11 11 11 11]';
ex_out1_3 = [0.0839,0.1492;0.0833,0.1481;0.0828,0.1472;0.0822,0.1462; ...
    0.0817,0.1452;0.0811,0.1443;0.0806,0.1433;0.0801,0.1424; ...
    0.0836,0.1486;0.0827,0.1470;0.0818,0.1454;0.0809,0.1438;0.0800,0.1423];
case1_result = isequal(isequal(a1,ex_out1_1), ...
    isequal(b1,ex_out1_2), isequal(c1,ex_out1_3));

% Case 2
[a2 b2 c2] =
channel_spiral3(pRange(1),uRange(1),diameter(2:3)/2,flowRange(1));
ex_out2_1 = [141 142 143 144 145 146 147 148 149 150 101 102 103 104 ...
    105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 ...
    121 122 80 81 82 83 84 85 86 87 88 89 90 91]';
```

```
ex_out2_2 = [11 11 11 11 11 11 11 11 11 11 12 12 12 12 12 12 12 12 12 ...
    12 12 12 12 12 12 12 12 12 12 12 12 12 13 13 13 13 13 13 13 13 13 ...
    13 13 13]';
ex_out2_3 = [0.0956,0.1494;0.0950,0.1485;0.0945,0.1476;0.0939,0.1468; ...
    0.0934,0.1459;0.0929,0.1451;0.0924,0.1443;0.0918,0.1435; ...
    0.0913,0.1427;0.0908,0.1419;0.0953,0.1489;0.0944,0.1475; ...
    0.0935,0.1462;0.0927,0.1449;0.0919,0.1436;0.0911,0.1423; ...
    0.0903,0.1411;0.0895,0.1399;0.0888,0.1387;0.0880,0.1376; ...
    0.0873,0.1364;0.0866,0.1353;0.0859,0.1342;0.0852,0.1332; ...
    0.0846,0.1321;0.0839,0.1311;0.0833,0.1301;0.0826,0.1291; ...
    0.0820,0.1281;0.0814,0.1272;0.0808,0.1262;0.0802,0.1253; ...
    0.0920,0.1437;0.0908,0.1418;0.0896,0.1401;0.0885,0.1383; ...
    0.0874,0.1366;0.0864,0.1350;0.0854,0.1334;0.0844,0.1318; ...
    0.0834,0.1303;0.0825,0.1289;0.0815,0.1274;0.0807,0.1260];
case2_result = isequal(isequal(a2,ex_out2_1), ...
    isequal(b2,ex_out2_2), isequal(c2,ex_out2_3));

% Case 3
[a3 b3 c3] =
channel_spiral(pRange(1),uRange(1),diameter(1:2)/2,flowRange(2));
ex_out3_1 = [137 138 139 140 141 142 143 144 145 98 99 100]';
ex_out3_2 = [11 11 11 11 11 11 11 11 11 12 12 12]';
ex_out3_3 = [0.0841,0.1495;0.0836,0.1486;0.0831,0.1477;0.0826,0.1468; ...
    0.0821,0.1459;0.0816,0.1450;0.0811,0.1442;0.0806,0.1434; ...
    0.0802,0.1426;0.0842,0.1496;0.0833,0.1482;0.0826,0.1468];
case3_result = isequal(isequal(a3,ex_out3_1), ...
    isequal(b3,ex_out3_2), isequal(c3,ex_out3_3));

% Case 4
[a4 b4 c4] =
channel_spiral(pRange(3),uRange(2),diameter(1:2)/2,flowRange(3));
ex_out4_1 = [86 87 88 89 90]';
ex_out4_2 = [10 10 10 10 10]';
ex_out4_3 = [0.084,0.1494;0.0831,0.1478;0.0823,0.1462;0.0814,0.1447; ...
    0.0806,0.1432];
case4_result = isequal(isequal(a4,ex_out4_1), ...
    isequal(b4,ex_out4_2), isequal(c4,ex_out4_3));

% Case 5
[a5 b5 c5] =
channel_spiral(pRange(3),uRange(2),diameter(1:2)/2,flowRange(4));
ex_out5_1 = [104 105 106 107 108 109 110]';
ex_out5_2 = [10 10 10 10 10 10 10]';
ex_out5_3 = [0.0843,0.1499;0.0836,0.1487;0.0829,0.1474;0.0822,0.1462; ...
    0.0816,0.1450;0.0809,0.1439;0.0803,0.1427];
case5_result = isequal(isequal(a5,ex_out5_1), ...
    isequal(b5,ex_out5_2), isequal(c5,ex_out5_3));

% result
if (case5_result == 1) & isequal(case1_result, case2_result, ...
        case3_result, case4_result, case5_result)
    disp('All tests passed. Good job!')
else
    disp('Not all tests have passsed.')
end
```

GUI:

```
function varargout = SpiralDeviceOptimizationfunction1(varargin)
% SPIRALDEVICEOPTIMIZATIONFUNCTION1 MATLAB code for
SpiralDeviceOptimizationfunction1.fig
%      SPIRALDEVICEOPTIMIZATIONFUNCTION1, by itself, creates a new
SPIRALDEVICEOPTIMIZATIONFUNCTION1 or raises the existing
%      singleton*.
%
%      H = SPIRALDEVICEOPTIMIZATIONFUNCTION1 returns the handle to a new
SPIRALDEVICEOPTIMIZATIONFUNCTION1 or the handle to
%      the existing singleton*.
%
%
SPIRALDEVICEOPTIMIZATIONFUNCTION1('CALLBACK',hObject,eventData,handles,...)
calls the local
%      function named CALLBACK in SPIRALDEVICEOPTIMIZATIONFUNCTION1.M with
the given input arguments.
%
%      SPIRALDEVICEOPTIMIZATIONFUNCTION1('Property','Value',...) creates a
new SPIRALDEVICEOPTIMIZATIONFUNCTION1 or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before SpiralDeviceOptimizationfunction1_OpeningFcn
gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to
SpiralDeviceOptimizationfunction1_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help
SpiralDeviceOptimizationfunction1

% Last Modified by GUIDE v2.5 19-Apr-2016 13:42:29

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn',
@SpiralDeviceOptimizationfunction1_OpeningFcn, ...
                   'gui_OutputFcn',
@SpiralDeviceOptimizationfunction1_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
```

```
end
% End initialization code - DO NOT EDIT


% --- Executes just before SpiralDeviceOptimizationfunction1 is made visible.
function SpiralDeviceOptimizationfunction1_OpeningFcn(hObject, eventdata,
handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to SpiralDeviceOptimizationfunction1 (see
VARARGIN)

% Choose default command line output for SpiralDeviceOptimizationfunction1
handles.output = hObject;

% UIWAIT makes SpiralDeviceOptimizationfunction1 wait for user response (see
UIRESUME)
% uiwait(handles.figure1);

% Pre-defining switch case options
handles.switchfunc = 0;
handles.threeto4 = [3:4];
handles.fourto5 = [4:5];
handles.fiveto6 = [5:6];
handles.sixto7 = [6:7];
handles.seventо8 = [7:8];
handles.eightto9 = [8:9];
handles.nineto10 = [9:10];

% Update handles structure
guidata(hObject, handles);


% --- Outputs from this function are returned to the command line.
function varargout = SpiralDeviceOptimizationfunction1_OutputFcn(hObject,
eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on selection change in particle_diam.
function particle_diam_Callback(hObject, eventdata, handles)
% hObject    handle to particle_diam (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hints: contents = cellstr(get(hObject,'String')) returns particle_diam
contents as cell array
%        contents{get(hObject,'Value')} returns selected item from
particle_diam

% Switch case menu for pop up menu that lets user select particle diameters
str = get(hObject, 'String');
val = get(hObject,'Value');
switch str{val};
    case 'Select a range of particle diameter values'
        handles.switchfunc = 0;
    case '3-4 µm'
        handles.switchfunc = handles.threeto4;
    case '4-5 µm'
        handles.switchfunc = handles.fourto5;
    case '5-6 µm'
        handles.switchfunc = handles.fiveto6;
    case '6-7 µm'
        handles.switchfunc = handles.sixto7;
    case '7-8 µm'
        handles.switchfunc = handles.sevento8;
    case '8-9 µm'
        handles.switchfunc = handles.eightto9;
    case '9-10 µm'
        handles.switchfunc = handles.nineto10;
end
% Save the handles structure.
guidata(hObject,handles)


% --- Executes during object creation, after setting all properties.
function particle_diam_CreateFcn(hObject, eventdata, handles)
% hObject    handle to particle_diam (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function density_Callback(hObject, eventdata, handles)
% hObject    handle to density (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of density as text
%        str2double(get(hObject,'String')) returns contents of density as a
double
handles.density = str2double(get(hObject,'String'));
```

```
guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function density_CreateFcn(hObject, eventdata, handles)
% hObject    handle to density (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function viscosity_Callback(hObject, eventdata, handles)
% hObject    handle to viscosity (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of viscosity as text
%        str2double(get(hObject,'String')) returns contents of viscosity as a
double
handles.viscosity = str2double(get(hObject,'String'));

guidata(hObject, handles);


% --- Executes during object creation, after setting all properties.
function viscosity_CreateFcn(hObject, eventdata, handles)
% hObject    handle to viscosity (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function flowrate_Callback(hObject, eventdata, handles)
% hObject    handle to flowrate (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of flowrate as text
%        str2double(get(hObject,'String')) returns contents of flowrate as a
double
handles.flowrate = str2double(get(hObject,'String'));
```

```
guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function flowrate_CreateFcn(hObject, eventdata, handles)
% hObject    handle to flowrate (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function channelwidth_Callback(hObject, eventdata, handles)
% hObject    handle to channelwidth (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of channelwidth as text
%        str2double(get(hObject,'String')) returns contents of channelwidth
as a double
handles.channelwidth = str2double(get(hObject,'String'));

guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function channelwidth_CreateFcn(hObject, eventdata, handles)
% hObject    handle to channelwidth (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function channelheight_Callback(hObject, eventdata, handles)
% hObject    handle to channelheight (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of channelheight as text
%        str2double(get(hObject,'String')) returns contents of channelheight
as a double
handles.channelheight = str2double(get(hObject,'String'));
```

```
guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function channelheight_CreateFcn(hObject, eventdata, handles)
% hObject    handle to channelheight (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes on button press in calculate.
function calculate_Callback(hObject, eventdata, handles)
% hObject    handle to calculate (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

if handles.switchfunc == 0
    msgbox('Please choose a range!')
else
    [handles.channelwidth, handles.channelheight, handles.R_f] =
channel_spiral3(handles.density, ...
    handles.viscosity, handles.switchfunc, handles.flowrate); % User-defined
function to calculate outputs

    % Displays outputs into tables
    set(handles.uitable1, 'Data', handles.channelwidth);
    set(handles.uitable2, 'Data', handles.channelheight);
    set(handles.uitable3, 'Data', handles.R_f);

    axes(findobj('Tag', 'axes1'));
    scatter(handles.channelwidth,handles.channelheight);
    xlabel('Channel Width (um)')
    ylabel('Channel Height (um)')
    title('Channel Width vs. Channel Height')
end

% Sample values that work:
% density = 1e-12
% viscosity = 1e-6
% flow rate = 8.33e9

guidata(hObject, handles);
```