Supplementary Material

MATLAB script to find experimental enhanced dispersion coefficient, K

James Teague, Lingyun Ding and Francesca Bernardi

This document contains one MATLAB script and five supporting functions that can be used to find the experimental enhanced dispersion coefficient of a passive tracer undergoing Taylor dispersion in a microchannel. The first function (runExperimentAnalysis.m) calls the other five.

1) Main Script: runExperimentAnalysis.m

```
% Add path to folder containing all code files
Addpath(”Path to directory containing code files”)
[FullPath, q] = selectImageStack

% Select images to process
[FullPath, q] = selectImageStack();
% Specify experimental parameters
[Experiment_number, Aspect_ratio, U, ChannelNum, Wait_time] = getExperimentMetadata();

% Rotate and crop region of interest (ROI) for
img = imread(char(FullPath{round(q/2)}));
[rot, roi] = defineROIs(img, FullPath);
close all

% Extract intensity data from each image selected
[meanInvBlue, sd] = analyzeBlueChannel(FullPath, q, rot, roi);


a = 0.00005;  % Channel half height (in meters)
Pe = U * a / (5.7e-10); % Peclet number, kappa = 5.7e-10 m^2/sec

% Produce fit and plot
Time = 1:q;
baseline = mean(meanInvBlue(30:80)); % Experimental baseline value
[fitresult, gof] = GaussianFit_Jove(Time, meanInvBlue, U, baseline);
% Best fit value for the experimental enhanced dispersion coefficient
K = double(fitresult.K); % K (m^2/s)
```

i) Helper function 1: selectImageStack

This function is used to select images for processing.

```
function [FullPath, q] = selectImageStack()
% Select the folder containg the images to be processed
    selpath = uigetdir('Select folder containing images');
    if isequal(selpath, 0)
        error("No folder selected.");
    end
% Select images within folder
    cd(selpath);
    [FileName, PathName] = uigetfile({'*.JPG','JPG Files'}, 'Select image files',
'MultiSelect', 'on');
    if isequal(FileName, 0)
        error("No files selected.");
    end
% Save path for each image individually and number of images
    FileName = cellstr(FileName);
    FullPath = strcat(PathName, FileName);
    q = numel(FileName); % number of images
end
```

ii) Helper function 2: getExperimentMetaData

This function prompts the user to specify experimental parameters.

```
function [Experiment_number, Aspect_ratio, U, ChannelNum, Wait_time] =
getExperimentMetadata()
    Experiment_number = input("Enter experiment number: ", 's');
    Aspect_ratio = input("Enter aspect ratio in decimal (i.e., use 0.25 for 1/4): ",
's'); % Channel aspect ratio defined as half height divided by half width
    U = input("Enter flow rate in m/s: ", 's');
    ChannelNum = input("Enter microchannel number: ", 's'); % Specifying the
microchannel used
    Wait_time = input("Enter wait time in seconds: ", 's'); % Diffusion wait time for
the initial condition
end
```

iii) Helper function 3: defineROIs

This function rotates the experimental images (if needed) and selects the region of interest to be analyzed.

```
function [rot, roi] = defineROIs(img,FullPath)
    ImageCheck = 0;
        figure;
        % Rotate image if needed to make microchannel horizontal
        imshow(img);
        title('Draw rotation ROI');
        area = drawrectangle('Rotatable', true);
```

```
        pause;
        BW = createMask(area);
        rot = regionprops(BW, 'Orientation'); % angle of rotation (rad)
   while ImageCheck == 0
       % Crop region of interest
        imgRot = imrotate(img, -rot.Orientation, 'bilinear', 'crop');
        figure;
        imshow(imgRot);
        title('Draw cropping ROI');
        roi = drawrectangle('Rotatable', true);
        BW = createMask(roi);
        roi = regionprops(BW, 'BoundingBox'); % Region of interest
        pause;
        close all
        % Check that cropped roi is correct
        img = imread(char(FullPath(50)));
        img = imrotate(img, -rot.Orientation, 'bilinear', 'crop');
        Icropped = imcrop(img, roi.BoundingBox);
        B = single(Icropped(:, :, 3));  % Show blue channel image for verification
        imshow(B, [0 255])
        ImageCheck = input('Enter 0 to redo image, else enter 1: ');
    end
end
```

iv) Helper function 5: analyzeBlueChannel

This function computes the average inverted blue channel intensity value for the region of interest in each image.

```
function [meanInvBlue, sd] = analyzeBlueChannel(FullPath, q, rot, roi)
    Blue = zeros(1, q);
    invBlue = zeros(1, q);
    sd = zeros(1, q);
% Run through each image selected (1 through q)
    for i = 1:q
        img = imread(char(FullPath(i)));
        % Rotate image
        img = imrotate(img, -rot.Orientation, 'bilinear', 'crop');
        % Crop image
        Icropped = imcrop(img, roi.BoundingBox);
        % Isolate blue channel from full RGB
        B = single(Icropped(:, :, 3));
        %imshow(B, [0 255]) % Visualize images during processing
        disp (i) % Counter of processed images appearing in command window
        % Invert blue channel
        invBlue = 255 - B;
        % Compute mean and standard deviation of inverted blue channel
        % image
        meanInvBlue(i) = mean(invBlue,'all');
        sd(i) = std(meanInvBlue(:), 1);
    end
end
```

v) Helper function 5: GaussianFit_Jove

This function produces a custom Gaussian-like fit for the experimental data and plots it.

```
function [fitresult, gof] = GaussianFit_Jove(Time, meanInvBlue, U, baseline)
% Fit: 'GaussianCurveFitter'.
[xData, yData] = prepareCurveData(Time, meanInvBlue ); % Transform data for curve
fitting
U2 = num2str(U);
% Set up fittype and options
% Fittype based on equation (1) on manuscript
ft = fittype( ['h3/(x.^.5)*exp((-((X1)-x).^2)/((4*K/((' U2 ').^2))*x))+h4'],
'independent', 'x', 'dependent', 'y' );
opts = fitoptions( 'Method', 'NonlinearLeastSquares' );
opts.Display = 'Off';
% Set wide bounds for parameters [K tmax Cf bl]
opts.Lower = [9E-12 10 0 baseline-5];
opts.StartPoint = [1.6E-07 150 200 baseline];
opts.Upper = [1E-05 300 Inf baseline+5];
% Set intial point for parameter fit
% 150 is the approximate time at which the intensity peaks


% Fit model to data
[fitresult, gof] = fit( xData, yData, ft, opts );

% Plot fit with data.
figure( 'Name', 'GaussianCurveFitter' );
h = plot(fitresult, xData, yData);  % Plot fit and data
set(h(1), 'LineStyle', '--', 'Color', 'k', 'LineWidth', 1, 'Marker', 'none');  % Fit
curve
set(h(2), 'LineStyle', '-', 'Color', 'k', 'LineWidth', 1, 'Marker', 'none'); %
Experiment data points
legend( h, 'Experimental data', 'MATLAB fit', 'Location', 'West', 'Interpreter',
'none' );
% Label axes
xlabel( 'Time (s)', 'Interpreter', 'none' );
ylabel( 'Intensity', 'Interpreter', 'none' );
grid on
```