Supplement for the custom MATLAB code for MSD calculation

Users should prepare the following data in advance

1) X and Y positions from trajectory for each condensate
2) tau (time lag).

Matrix (n_time x n_track) for X and Y positions.

X	$Track\ \#1$	$Track\ \#2$	…	$Track\ \#n\_track$
$t(1)$	$X(1,1)$	$X(1,2)$	…	$X(1,n\_track)$
$t(2)$	…	…	…	$X(2,n\_track)$
	…	…	…	…
$t(n\_time)$	$X(n\_time,1)$	$X(n\_time,2)$	…	$X(n\_time,n\_track)$

Example (n_time = 7, n_track = 24)

Variables - X

X

7x24 double

	1	2	3	4	5	6	7	8	9	10	11	12
1	244.2375	94.5750	66.3000	51.0250	91.4875	128.5375	130.1625	215.9625	16.5750	128.3750	136.1750	27.1375
2	243.2625	93.6000	66.6250	51.8375	92.6250	128.5375	129.6750	215.6375	15.9250	128.8625	136.0125	27.7875
3	243.2625	92.7875	66.7875	51.1875	93.4375	129.3500	130.4875	215.1500	15.1125	128.7000	134.8750	28.4375
4	242.9375	91.8125	67.2750	51.0250	93.2750	129.5125	130.4875	215.8000	13.9750	128.7000	135.6875	28.2750
5	244.2375	91.4875	66.4625	51.1875	93.1125	129.0250	129.8375	215.8000	14.9500	129.0250	135.8500	27.4625
6	244.2375	91.4875	66.4625	51.0250	93.7625	128.5375	130.0000	216.2875	14.7875	130.6500	136.5000	26.9750
7	244.7250	91.9750	67.4375	49.8875	93.7625	128.7000	129.8375	216.7750	16.5750	130.8125	137.4750	27.1375

Tau (length = (n_time -1)) Zero tau is not included.

tau (1)
…
…
$tau(n_{time}, -1)$

Variables - tau

tau

6x1 double

	1
1	1
2	2
3	3
4	4
5	5
6	6

```matlab
%% Empty MSD matrix generation
msd_X = zeros((n_time - 1), n_track) ; % Empty MSD matrix
msd_Y = zeros((n_time - 1), n_track) ; % Empty MSD matrix
%% MSD calculation
square_disp = zeros (length(tau) , length(tau));

idx_x = n_time - 1 ;
for k = 1 : n_track
    idx_x = n_time - 1 ; % Reset idx_x
    for i = 1 : n_time - 1
        for j = 1 : idx_x
        square_disp(j, i) = (X(j, k) - X((j + i), k))^2 ;
        end
        msd_X (i, k) = sum(square_disp(: , i)) / idx_x; % MSD_X
        idx_x = idx_x - 1 ;
    end
end

idx_y = n_time - 1 ;
for k = 1 : n_track
    idx_y = n_time - 1 ; % Reset idx_y
    for i = 1 : n_time - 1
        for j = 1 : idx_y
        square_disp(j, i) = (Y(j, k) - Y((j + i), k))^2 ;
        end
        msd_Y (i, k) = sum(square_disp(: , i)) / idx_y; % MSD_Y
        idx_y = idx_y - 1 ;
    end
end

msd_sum = msd_X + msd_Y ; % FINAL MSD FOR EACH TRACK
```

msd_X	*Track #1*	…	*Track #n_track*
	msd_X (1, 1)		
	...	...	
	...	...	...
	msd_X (1, (n_time -1))		

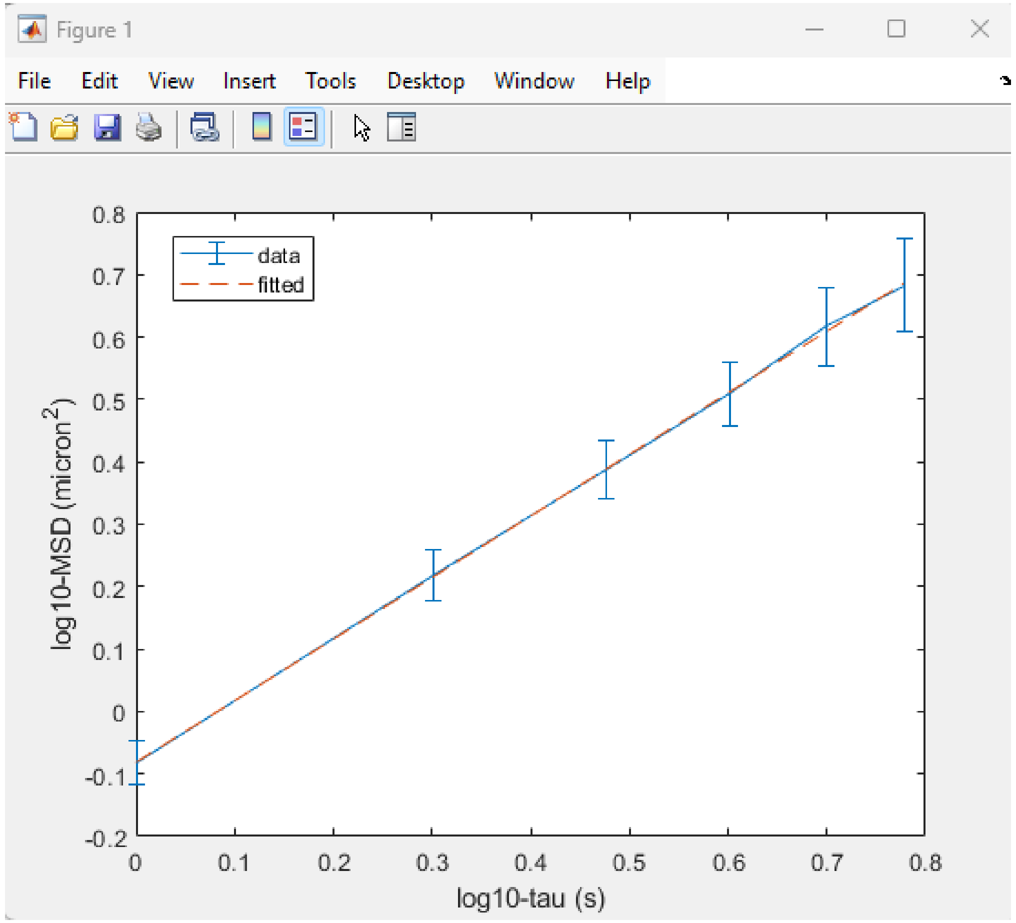
Variables - msd_sum

msd_sum

6x24 double

	1	2	3	4	5	6	7
1	0.8274	1.1839	0.6778	1.0254	1.0474	0.5589	0.5457
2	1.4946	2.8519	1.7164	1.8062	2.2181	1.9435	0.6866
3	3.1357	4.7597	2.4030	3.1621	2.3105	3.3338	0.2575
4	2.8431	6.6984	3.2920	4.8852	2.9223	5.0612	0.3961
5	2.9971	8.0011	5.3209	5.7302	7.4202	8.1595	0.3697
6	0.8978	6.7864	7.2353	6.4695	11.9356	10.5889	0.2113

```matlab
%% Statistics
msd_mean = ones(length(tau), 1);
msd_std = ones(length(tau), 1); % standard deviation
msd_se = ones(length(tau), 1); % standard error

for i = 1 : length(tau)
    msd_mean(i) = mean(msd_sum(i, :));
end

for i = 1 : length(tau)
    msd_std(i) = std(msd_sum(i, :));
end

for i = 1 : length(tau)
    msd_se(i) = msd_std(i) / sqrt(n_track) ;
end
%% log_tau vs. log_MSD linear fitting to obtain the diffusion exponent.
% x = log10_tau, y = log10_MSD.
% Fitting to "y = a*x + b", a: diffusion exponent.
x = log10(tau) ;
y = log10(msd_mean) ;
[p, S] = polyfit(x, y, 1) ;

log_msd_se = ones(length(tau), 1) ; % Standard errors for log10-MSD
for i = 1 : length(tau)
    log_msd_se(i) = (1/log(10))*(msd_se(i) / msd_mean(i)); % Error propagation.
end

f = polyval(p, x)' ;
figure(1)
plot(x, y, 'o')
errorbar(x, y, log_msd_se)
hold on
plot(x, f, '--')
xlabel ('log10-tau (s)')
ylabel ('log10-MSD (micron^2)')
legend('data', 'fitted')
%% Print results
fprintf('Diffusion exponent is %f\n', p(1))
fprintf('R^2 for linear fitting = %f\n', S.rsquared)
```



Command Window

```
Diffusion exponent is 0.986582
R^2 for linear fitting = 0.999721
>>
```

Supplement for the custom MATLAB code for MSD linear fitting to obtain diffusivity

Matrix (n_tau x n_track) for MSDs for each track.

	Track #1	*Track #2*	…	*Track #n_track*
$tau(1)$	$MSD(1,1)$	$MSD(1,2)$	…	$MSD(1,n\_track)$
$tau(2)$	…	…	…	$MSD(2,n\_track)$
	…	…	…	…
$tau(n\_tau)$	$MSD(n\_tau,1)$	$MSD(n\_tau,2)$	…	$MSD(n\_tau,n\_track)$

Example (n_tau = 6, n_track = 24)

Variables - MSD

MSD

6x24 double

	1	2	3	4	5	6	7	8	9
1	0.8274	1.1839	0.6778	1.0254	1.0474	0.5589	0.5457	0.3301	1.7560
2	1.4946	2.8519	1.7164	1.8062	2.2181	1.9435	0.6866	0.6654	3.5120
3	3.1357	4.7597	2.4030	3.1621	2.3105	3.3338	0.2575	0.7724	4.9248
4	2.8431	6.6984	3.2920	4.8852	2.9223	5.0612	0.3961	1.0827	3.5913
5	2.9971	8.0011	5.3209	5.7302	7.4202	8.1595	0.3697	0.7658	6.2979
6	0.8978	6.7864	7.2353	6.4695	11.9356	10.5889	0.2113	1.0827	7.6314

```
%% Variables
filename0 = 'MSD_example_BJ' ; % !!MAKE SURE IT IS THE RIGHT FILE!!
MSD = readmatrix(filename0,'Sheet', 1) ;
tau = readmatrix(filename0,'Sheet', 2) ;
n_tau = length(tau) ;
n_track = length(MSD(1, :)) ; % Number of tracks
%% Fitting MSD = b*Tau (zero-intercept)
D = ones(n_track, 1) ; % Diffusivity
x = tau ;
y = MSD ;

for i = 1 : n_track
    D(i) = linearfit(x, y(:, i)) / 4 ;
end
D_avg = mean(D) ;
D_se = std(D) / sqrt(n_track) ; % Standard error
%% Final results summary
fprintf('Mean Diffusivity : %f um2/s\n', D_avg)
```

A function for linear fitting with a zero y-intercept.

linearfit.m

```
% Function for linear fitting with a zero y-intercept.
function slope = linearfit(x, y)
    % Ensure x and y are column vectors
    if isrow(x), x = x' ; end
    if isrow(y), y = y' ; end

    % Compute the slope using least squares (Y = slope*X)
    slope = (x' * y) / (x' * x); % The best-fit slope minimizing the sum of squared errors.
end
```

Command Window

```
Mean Diffusivity : 0.203013 um2/s
>>
```

How to obtain the best-fit slope that minimizes the sum of squared errors.

$$Y = \beta \cdot X$$

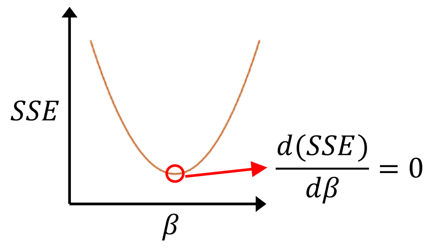
$$\begin{bmatrix} y_1 \\ y_2 \\ \cdots \\ y_n \end{bmatrix} = \begin{bmatrix} \beta x_1 \\ \beta x_2 \\ \cdots \\ \beta x_n \end{bmatrix}$$

X and Y are column vectors.
X: tau
Y: MSD
β: slope

$$sum\ of\ squared\ errors\ (SSE) = \sum_{i=1}^{n} (y_i - \beta x_i)^2$$

$$= ({y_1}^2 + {y_2}^2 + \cdots + {y_n}^2) - 2\beta(x_1 y_1 + x_2 y_2 + \cdots + x_n y_n) + \beta^2({x_1}^2 + {x_2}^2 + \cdots + {x_n}^2)$$

$$= Y^T Y - 2\beta X^T Y + \beta^2 X^T X$$



$$\frac{d(SSE)}{d\beta} = -2X^T Y + 2\beta X^T X = 0$$

$$\therefore \beta_{best} = \frac{X^T Y}{X^T X}$$

linearfit.m

```
% Function for linear fitting with a zero y-intercept.
function slope = linearfit(x, y)
    % Ensure x and y are column vectors
    if isrow(x), x = x' ; end
    if isrow(y), y = y' ; end

    % Compute the slope using least squares (Y = slope*X)
    slope = (x' * y) / (x' * x); % The best-fit slope minimizing the sum of squared errors.
end
```