Resolving Severely Overlapping Ion Mobility Peaks Using Enhanced Fourier Self-Deconvolution

Shujuan Liu,[a,b] Jian Jia,[a] Xiaoguang Gao,[*a] and Xiuli He,[*a]

November 14, 2024

[a] *State Key Laboratory of Transducer Technology, Aerospace Information Research Institute, Chinese Academy of Sciences, Beijing, 100190, P. R. China. E-mail: hxl@mail.ie.ac.cn. E-mail: xggao@mail.ie.ac.cn*

[b] *School of Electronic, Electrical and Communication Engineering, University of Chinese Academy of Sciences, Beijing, 100049, P. R. China.*

Contents

1 Derivation process of Equ.12

Tom O'Haver[1] used a zero-centered Gaussian function (maximum at $t = 0$) as a deconvolving function, expressed as:

$$c(t) = \begin{cases} e^{-\frac{(t-t_{max})^2}{2\sigma^2}}, & t > \frac{t_{max}}{2} \\ e^{-\frac{t^2}{2\sigma^2}}, & t \leq \frac{t_{max}}{2} \end{cases} \tag{1}$$

where t_{max} represents the maximum value of t.

To address the asymmetric broadening observed in the IMS spectrum, this study introduces a modified deconvolving function $g(t)$ that combines Gaussian and Breit-Wigner functions for application in Fourier self-deconvolution. This newly defined function, represented in Equation 2, captures spectral asymmetry by centring its left and right components at t_{max} and 0, respectively. This configuration improves the model's capacity to handle asymmetrical broadening, thereby reducing errors commonly associated with tailing effects and enhancing overall deconvolution accuracy.

$$g(t) = \begin{cases} e^{-\frac{(t-t_{max})^2}{2\sigma^2}}, & t > \frac{t_{max}}{2} \\ \frac{w_{bw}^2}{w_{bw}^2+t^2}, & t \leq \frac{t_{max}}{2} \end{cases} \tag{2}$$

According to the derived relationships, the half-width of the left (Gaussian) and right (Breit-Wigner) half-peaks, denoted as w_{gaus} and w_{bw}, respectively, is determined by the horizontal distance between the crest and the left (right) trough of the wavelet coefficient curve, represented by Δt_l and Δt_r.

$$w_{gaus} = \sqrt{\frac{2\ln 2}{3}}\Delta t_l \tag{3}$$

$$w_{bw} = \Delta t_r \tag{4}$$

For Gaussian functions, the half-width w_{gaus} relates to the standard deviation σ as follows:

$$w_{gaus} = \sigma \cdot \sqrt{2\ln 2} \tag{5}$$

By substituting Equation 3 into Equation 5, we derive:

$$\begin{aligned} \sigma &= \frac{w_{gaus}}{\sqrt{2\ln 2}} \\ &= \frac{\Delta t_l}{\sqrt{3}} \end{aligned} \tag{6}$$

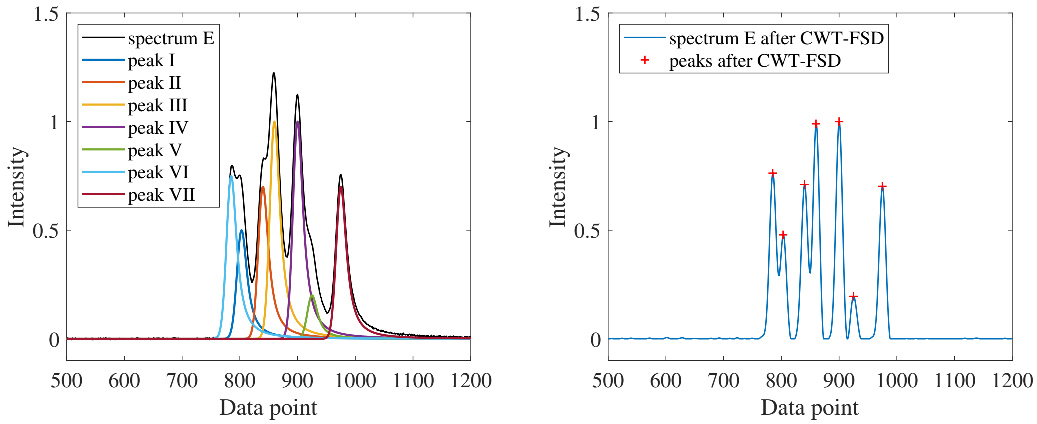
Finally, substituting Equations 6 and 4 into Equation 2 yields:

$$g(t) = \begin{cases} e^{-\frac{3(t-t_{max})^2}{2\Delta t_l^2}}, & t > \frac{t_{max}}{2} \\ \frac{\Delta t_r^2}{\Delta t_r^2+t^2}, & t \leq \frac{t_{max}}{2} \end{cases} \tag{7}$$

2 Tables and Figures

Table S1 Parameters of spectra G and H discussed in this paper

Parameter	Spectrum G							Spectrum H						
	I	II	III	IV	V	VI	VII	I	II	III	IV	V	VI	VII
μ	664	675	715	724	744	755	804	664	675	714	726	744	755	804
w_l	5.25	5.03	5.79	5.02	4.97	5.57	4.67	4.90	5.18	5.51	5.12	5.19	4.85	5.01
w_r	5.89	6.58	5.48	6.09	5.86	5.40	6.37	6.71	6.39	6.32	6.76	6.25	6.85	6.92
h	0.28	0.77	1.00	0.87	0.59	0.28	0.16	1.00	0.86	0.27	0.78	0.58	0.28	0.16



(a) (b)

Figure S1 Spectrum E and the deconvolution results using CWT-FSD at a cutoff frequency of 150. (a) Spectrum E before deconvolution. (b) Use the combination of the Gaussian function and the Breit-Wigner function as the linear function for CWT-FSD.

Table S2 Deconvolution results for spectra G and H using different deconvolution functions at a cutoff frequency of 210

True value - Spectrum G	Estimated value / Absolute error		
	Function 1 ($w = 12.27$)	Function 2 ($w = 4.82 + 7.45$)	Function 3 ($w = 4.82 + 7.45$)
664	663 / -1	665 / 1	664 / 0
675	675 / 0	676 / 1	676 / 1
715	713 / -2	715 / 0	715 / 0
724	723 / -1	725 / 1	724 / 0
744	743 / -1	744 / 0	745 / 1
755	754 / -1	756 / 1	755 / 0
804	804 / 0	805 / 1	805 / 1
Misidentification	641, 652, 691, 701, 732, 764, 792	691	/

True value - Spectrum H	Estimated value / Absolute error		
	Function 1 ($w = 13.09$)	Function 2 ($w = 4.78 + 8.31$)	Function 3 ($w = 4.78 + 8.31$)
664	663 / -1	665 / 1	664 / 0
675	674 / -1	676 / 1	675 / 0
714	712 / -2	714 / 0	714 / 0
726	726 / 0	727 / 1	726 / 0
744	743 / -1	744 / 0	744 / 0
755	754 / -1	757 / 2	755 / 0
804	803 / -1	804 / 0	804 / 0
Misidentification	628, 639, 651, 686, 700, 765, 790	693	/

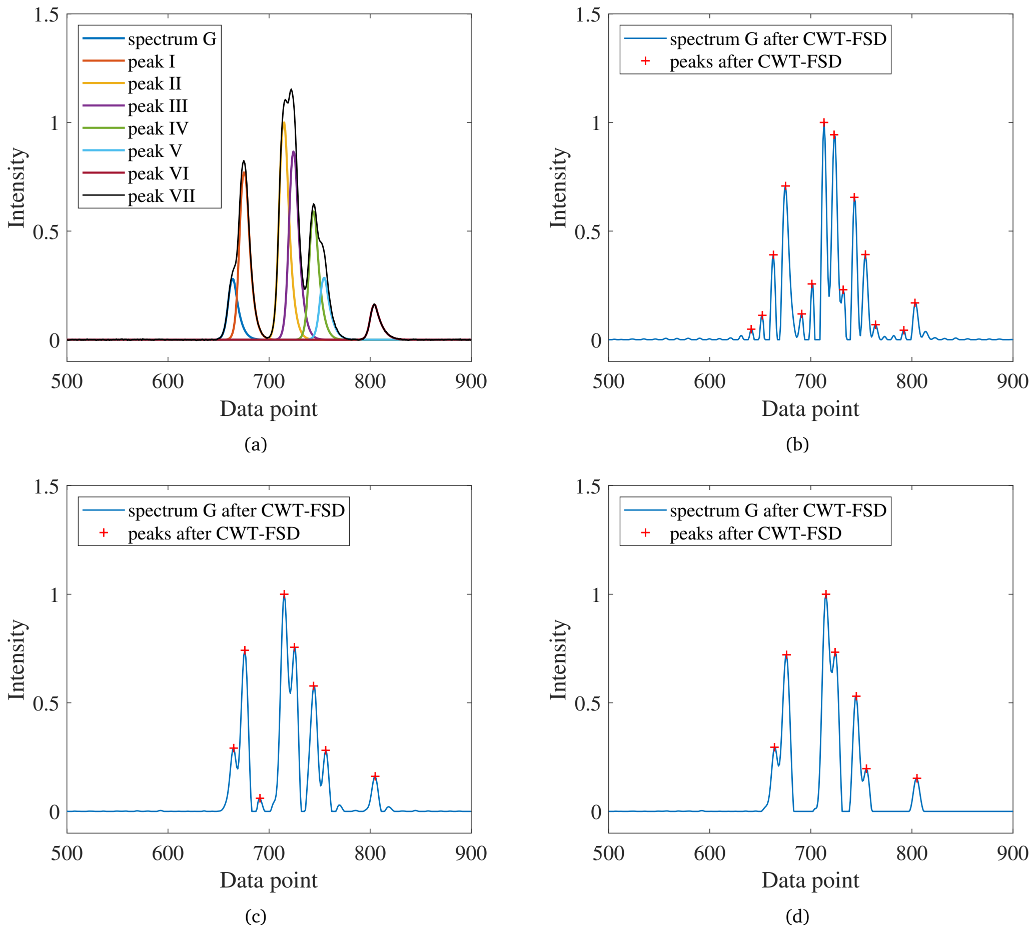


Figure S2 Spectrum G and the deconvolution results using different deconvolution functions at a cutoff frequency of 210. (a) Spectrum G before deconvolution. (b) Use the symmetric Gaussian function as the deconvolving function for CWT-FSD. (c) Use the asymmetric Gaussian function as the deconvolving function for CWT-FSD. (d) Use the combination of the Gaussian function and the Breit-Wigner function as the deconvolving function for CWT-FSD.

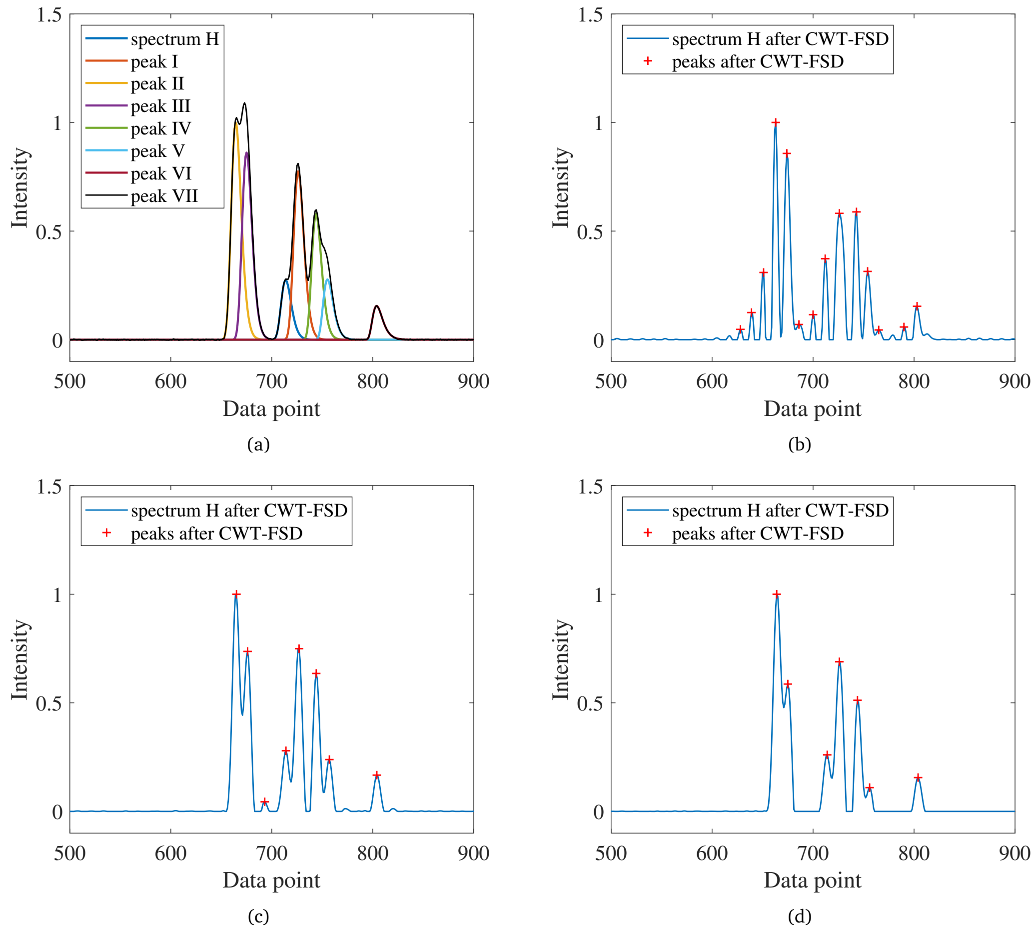


Figure S3 Spectrum H and the deconvolution results using different deconvolution functions at a cutoff frequency of 210. (a) Spectrum H before deconvolution. (b) Use the symmetric Gaussian function as the deconvolving function for CWT-FSD. (c) Use the asymmetric Gaussian function as the deconvolving function for CWT-FSD. (d) Use the combination of the Gaussian function and the Breit-Wigner function as the deconvolving function for CWT-FSD.

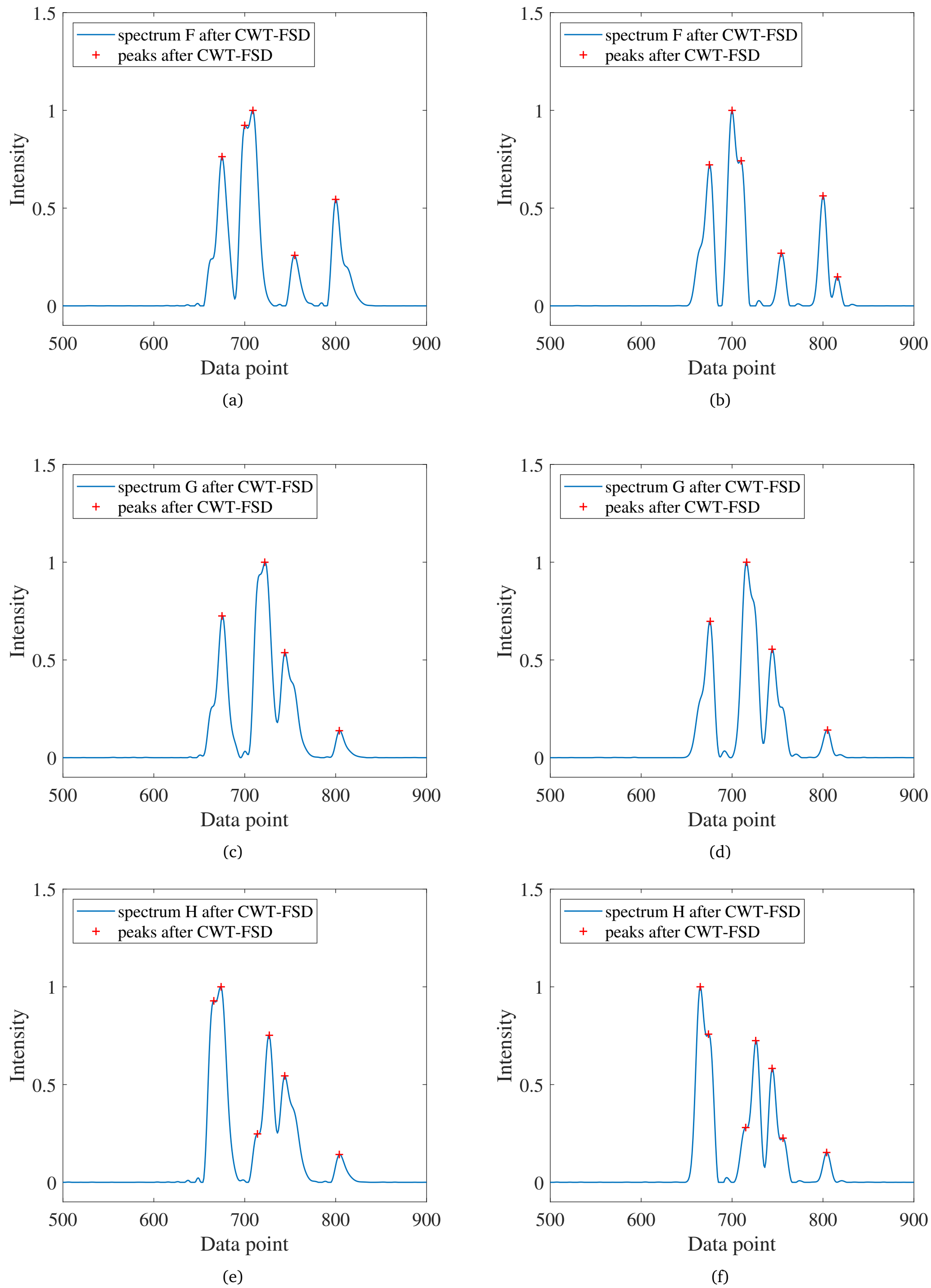


Figure S4 Deconvolution results of spectra F, G, and H at lower cut-off frequencies. (a) Spectrum F processed with a Gaussian function as the convolution function at a cut-off frequency of 130. (b) Spectrum F processed with an asymmetric Gaussian function at a cut-off frequency of 160. (c) Spectrum G processed with a Gaussian function at a cut-off frequency of 140. (d) Spectrum G processed with an asymmetric Gaussian function at a cut-off frequency of 160. (e) Spectrum H processed with a Gaussian function at a cut-off frequency of 130. (f) Spectrum H processed with an asymmetric Gaussian function at a cut-off frequency of 160.

Table S3 Results of different methods on spectrum F, G, and H

Spectrum F	True value	Estimated value / Absolute error		
		WFD	DWT-fitting-FSD	CWT-FSD
I	664	/	660 / -4	664 / 0
II	675	676 / 1	676 / 1	675 / 0
III	699	702 / 3	699 / 0	699 / 0
VI	710	708 / -2	710 / 0	710 / 0
V	754	753 / -1	755 / 1	754 / 0
VI	800	800 / 0	800 / 0	800 / 0
VII	814	812 / -2	817 / 3	815 / 1
Misidentification		673, 755	733, 780	/
Run time (s)		0.223	0.170	0.018
FWHM	5.12+6.62	39.71	18.92	4.88+8.53

Spectrum G	True value	Estimated value / Absolute error		
		WFD	DWT-fitting-FSD	CWT-FSD
I	664	/	660 / -4	664 / 0
II	675	676 / 1	676 / 1	676 / 1
III	715	718 / 3	/	715 / 0
VI	724	724 / 0	723 / -1	724 / 0
V	744	744 / 0	744 / 0	745 / 1
VI	755	750 / -5	/	755 / 0
VII	804	804 / 0	804 / 0	805 / 1
Misidentification		/	695	/
Run time (s)		0.213	0.170	0.014
FWHM	5.18+5.95	33.87	18.94	4.82+7.45

Spectrum H	True value	Estimated value / Absolute error		
		WFD	DWT-fitting-FSD	CWT-FSD
I	664	665 / 1	665 / 1	664 / 0
II	675	672 / -3	674 / -1	675 / 0
III	714	713 / -1	709 / -5	714 / 0
VI	726	725 / -1	726 / 0	726 / 0
V	744	744 / 0	745 / 1	744 / 0
VI	755	750 / -5	/	755 / 0
VII	804	804 / 0	804 / 0	804 / 0
Misidentification		/	644	/
Run time (s)		0.216	0.289	0.017
FWHM	5.11+6.60	45.95	20.66	4.78+8.31

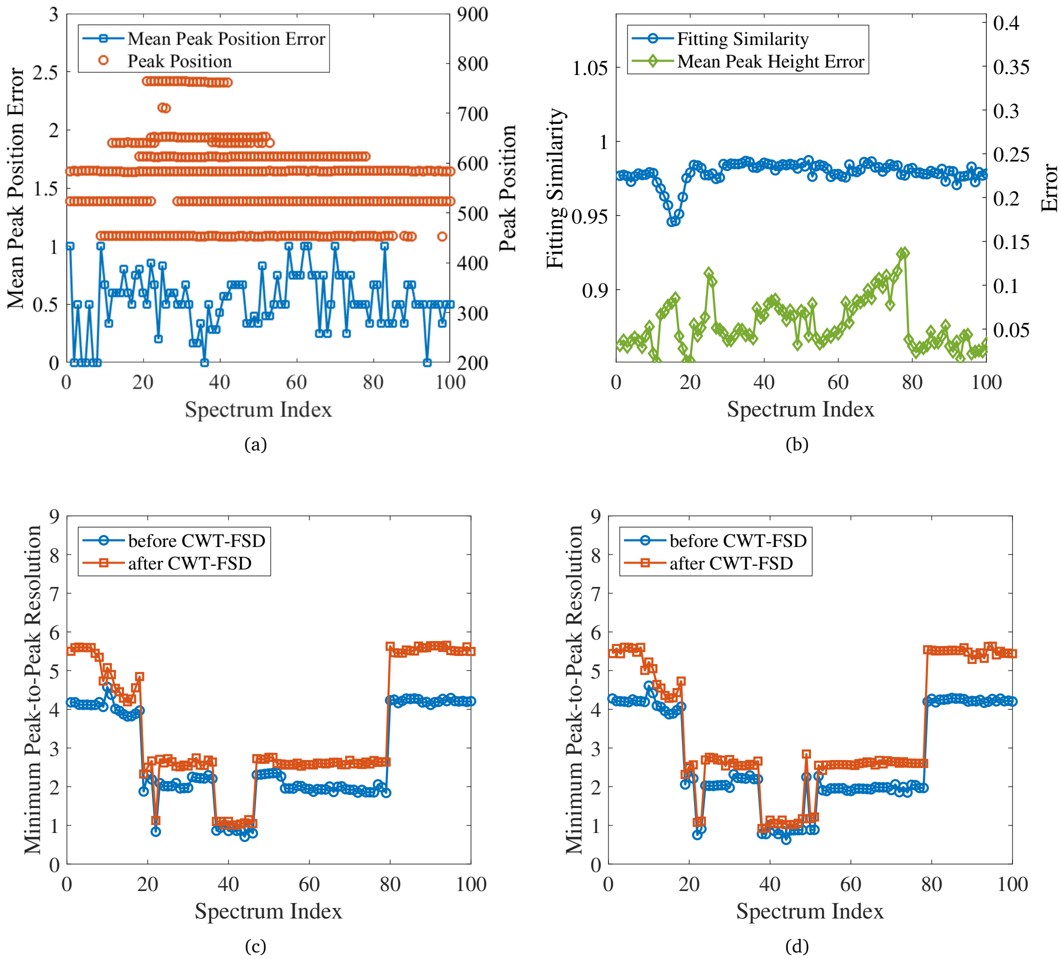


Figure S5 Results of human exhaled gas signals using CWT-FSD. (a) Peak position and average absolute error detected after deconvolution of spectral sequence B. (b) Peak height mean relative error and spectral similarity detected after deconvolution of spectral sequence B. (c) Change of minimum resolution before and after deconvolution of spectral sequence A. (d) Change of minimum resolution before and after deconvolution of spectral sequence B.

3 The code used in this article

The following code is used for processing spectral data using Continuous Wavelet Transform (CWT) and Fourier Self-Deconvolution (FSD) methods. Each function's input and output are specified below:

```

% Main Program
% This program performs Continuous Wavelet Transform (CWT) and Fourier Self-Deconvolution (FSD) on spectral data to detect peaks.
%
% Input: Spectral data from an external file.
% Output: Processed data with identified peak positions and plots.
%
% Function Descriptions:
% 1. Cwt_my - Performs Continuous Wavelet Transform.
%    - Input: s (array) - spectral data, N (int) - length of data.
%    - Output: L_l (float) - the weighted average horizontal distances between each peak and its corresponding left trough,
%              L_r (float) - the weighted average horizontal distances between each peak and its corresponding right trough.
%
% 2. FSD - Performs Fourier Self-Deconvolution.
%    - Input: s (array) - spectral data, N (int) - length of data,
%              L_l (float) - results of Cwt_my, L_r (float) - results of Cwt_my.
%    - Output: Ln_ (array) - deconvoluted data.
%
% 3. Peak_find - Detects peaks based on thresholds.
%    - Input: s (array) - spectral data, Ln_ (array) - wavelet coefficient curve, locs (array), locs1 (array), Ln_max (float).
%    - Output: locs_ (array) - detected peak locations.
%
% 4. findFuzzyMode - Calculates the fuzzy mode of an array within a tolerance.
%    - Input: values (array), tolerance (float).
%    - Output: fuzzyMode (float) - mode value within tolerance.
%
% 5. gaussian - Generates a Gaussian-shaped function.
%    - Input: x (array) - x-axis values, pos (float) - position of peak, wid (float) - width of peak.
%    - Output: g (array) - Gaussian function values.
%
% 6. bw - Generates a Breit-Wigner-shaped function.
%    - Input: x (array) - x-axis values, pos (float) - position of peak, wid (float) - width of peak.
%    - Output: g (array) - Breit-Wigner function values.
%
% 7. FourierFilter - Applies Fourier filtering to a signal based on specified frequency parameters.
%    - Input: xvector (array) - x-axis values, yvector (array) - y-axis (signal) values,
%             centerfrequency (float) - center frequency, filterwidth (float) - width of filter,
%             filtershape (int) - shape of the filter, filtermode (int) - mode of filter.
%    - Output: ry (array) - filtered signal.
%
% 8. shapefunction - determines the sharpness of the cut-off.
%    - Input: x (array) - x-axis values, pos (float) - peak position, wid (float) - width,
%             n (int) - shape type (0 for Lorentzian, 1 or more for Gaussian).
%    - Output: g (array) - function values.



clearvars
tic
s = importdata('C:/Users/19464/Desktop/FFT/WT/data.txt'); % Read data file
N = length(s);
t = 1:N;

[L_l, L_r] = Cwt_my(s, N); % Continuous Wavelet Transform, returns L_l and L_r
Ln_ = FSD(s, N, L_l, L_r); % Fourier Self-Deconvolution
Ln_ = Ln_';
Ln_max = max(Ln_);

for m = 1:length(Ln_)
    if Ln_(m) < 0
        Ln_(m) = 0; % Set negative values to zero
    end
end

[pks, locs] = findpeaks(Ln_, 'minpeakheight', 0.05 * Ln_max); % Find peaks after deconvolution
locs_ = locs';
toc

% Plot deconvoluted curve and its peaks
hold on;
plot(t, Ln_, locs_, Ln_(locs_), "r+", 'linewidth', 1)
disp('Peaks after CWT-FSD processing:');
disp(locs_);
disp(Ln_(locs_)');

legend({'Spectrum E after CWT-FSD', 'Peaks after CWT-FSD'}, 'Location', 'NorthWest');
ylim([-0.1 1.5])
xlabel('Data point');
ylabel({'Intensity'});
```

```

% Peak detection on original curve
L_max = max(s);
[pks2, locs2] = findpeaks(s, 'minpeakheight', 0.05 * L_max);
disp('Peaks detected by MATLAB built-in function:');
disp(locs2');
set(gcf, 'color', 'white');

% Function: Cwt_my
function [L_l, L_r] = Cwt_my(s, N)
    % Filter the raw function
    fs = N;
    fc = 220; % Cut-off frequency for CWT (It can be adjusted according to the signal)
    [b, a] = butter(4, fc / (fs / 2)); % 4th order low-pass filter
    filtered_y = filtfilt(b, a, s); % Zero-phase filtering

    % Continuous Wavelet Transform
    p = 1:1;
    scales = 2 .^ p; % scales as powers of 2
    wname = 'gaus2'; % Using Gaussian wavelet
    coefs = cwt(filtered_y, scales, wname);

    % Find peaks and troughs in wavelet coefficients
    L_max = max(coefs);
    [pks_0, locs_0] = findpeaks(coefs);
    [pks_1, locs_1] = findpeaks(L_max - coefs);
    peaks = Peak_find(filtered_y, coefs, locs_0, locs_1, L_max); % Peak locations
    Ln_ = max(coefs) - coefs;
    Ln_max = max(Ln_);
    [pks0, locs0] = findpeaks(Ln_);
    [pks1, locs1] = findpeaks(Ln_max - Ln_);
    troughs = Peak_find(filtered_y, Ln_, locs0, locs1, Ln_max); % Find troughs

    left_differences = zeros(1, length(peaks));
    right_differences = zeros(1, length(peaks));
    peak_differences = zeros(1, length(peaks));
    for i = 1:length(peaks)
        % Find the nearest trough on the left of the current peak
        left_troughs = troughs(troughs < peaks(i));
        if ~isempty(left_troughs)
            left_diff = peaks(i) - left_troughs(end);
        else
            left_diff = NaN; % If no left trough, set to NaN
        end
        left_differences(i) = left_diff;

        % Find the nearest trough on the right of the current peak
        right_troughs = troughs(troughs > peaks(i));
        if ~isempty(right_troughs)
            right_diff = right_troughs(1) - peaks(i);
        else
            right_diff = NaN; % If no right trough, set to NaN
        end
        right_differences(i) = right_diff;
        if i < length(peaks)
            peak_differences(i) = peaks(i + 1) - peaks(i);
        end
    end

    disp('Differences between each peak and the nearest left trough:');
    disp(left_differences);

    disp('Differences between each peak and the nearest right trough:');
    disp(right_differences);

    disp('Differences between peaks:');
    disp(peak_differences);
    max_peak_diff = max(peak_differences);
    if isnan(left_differences(1))
        left_differences(1) = findFuzzyMode(left_differences, 2);
    end

    L_l = left_differences(1) * max_peak_diff;
    peak_diff = max_peak_diff;
    for j = 2:length(peaks)
        if abs(left_differences(j) - left_differences(1)) < 5
            L_l = L_l + left_differences(j) * peak_differences(j - 1);
            peak_diff = peak_diff + peak_differences(j - 1);
        end
    end
    L_l = L_l / peak_diff;
    disp('Weighted average distance between each peak and its nearest left trough:');
    disp(L_l);
    if isnan(right_differences(length(peaks)))
        right_differences(length(peaks)) = findFuzzyMode(right_differences, 2);
```

```matlab
    end
    peak_diff = max_peak_diff;
    L_r = right_differences(length(peaks)) * max_peak_diff;
    for j = 1:length(peaks) - 1
        if abs(right_differences(j) - right_differences(length(peaks))) < 5
            L_r = L_r + right_differences(j) * peak_differences(j);
            peak_diff = peak_diff + peak_differences(j);
        end
    end
    L_r = L_r / peak_diff;
    disp('Weighted average distance between each peak and its nearest right trough:');
    disp(L_r);
end

% Function to find peaks based on thresholds
function locs_ = Peak_find(s, Ln_, locs, locs1, Ln_max)
    q = Ln_max * 0.1; % Height difference threshold between peaks and troughs
    m = Ln_max * 0.1; % Peak height threshold
    p = 1;
    locs_ = [];
    locs_(1) = 0;
    if locs(1) > locs1(1)  % First trough, then peak
        for i = 2:length(locs) - 1
            if Ln_(locs(i)) - Ln_(locs1(i)) > q || Ln_(locs(i)) - Ln_(locs1(i + 1)) > q
                if Ln_(locs(i)) > m && locs(i) - locs_(max(1, p - 1)) > 4
                    locs_(p) = locs(i);
                    p = p + 1;
                end
            end
        end
    elseif locs(1) < locs1(1) % First peak, then trough
        for i = 2:length(locs) - 1
            if Ln_(locs(i)) - Ln_(locs1(i)) > q || Ln_(locs(i)) - Ln_(locs1(i - 1)) > q
                if Ln_(locs(i)) > m && locs(i) - locs_(max(1, p - 1)) > 4
                    locs_(p) = locs(i);
                    p = p + 1;
                end
            end
        end
    end
end

% Function to find the fuzzy mode of an array
function fuzzyMode = findFuzzyMode(values, tolerance)
    % Remove NaN values
    values = values(~isnan(values));

    % Calculate fuzzy mode
    unique_vals = unique(values);
    counts = zeros(size(unique_vals));
    for i = 1:length(unique_vals)
        counts(i) = sum(abs(values - unique_vals(i)) <= tolerance);
    end
    % Find the value with maximum frequency
    [~, maxIdx] = max(counts);
    fuzzyMode = unique_vals(maxIdx);

end

function xe = FSD(s, N, L_l, L_r)
    % Set parameters for Fourier Self-Deconvolution
    FrequencyCutoff = 150; % Frequency cutoff (simulation: 210, real: 165, Gaussian-BW simulation: 150)  (It can be adjusted according to
    ↪  the signal)
    CutOffRate = 1;
    DA = 1;
    index = 1:N;

    % Calculate Gaussian and Breit-Wigner width parameters
    % Theoretical value
    dw_1 = 0.6798 * (L_l) * 2;
    dw_2 = 1 * L_r * 2;
    % True value (can be used in the deconvolution of the true spectrum)
    % dw_1 = 0.65*(L_l)*2;
    % dw_2 =1.09*L_r*2;

    dw = (dw_1 + dw_2) / 2;
    F = s';

    % Combine Gaussian and Breit-Wigner function
    df = bw(index, min(index), dw_2) + gaussian(index, max(index), dw_1);

    % Asymmetric Gaussian function
    % df=gaussian(index,min(index),dw_2)+gaussian(index,max(index),dw_1);
    % Symmetric Gaussian function
    % df=gaussian(index,min(index),dw)+gaussian(index,max(index),dw);
```

```
    % Perform Fourier Transform and Inverse Fourier Transform for FSD
    ca5_FFT = fft(df);
    X_ifft = ifft(fft(F) ./ (ca5_FFT + DA * 0.01 * max(ca5_FFT))) * sum(df);
    syDA = FourierFilter(F, X_ifft, 0, FrequencyCutoff, CutOffRate, 1);

    % Normalize the result
    xe = syDA / max(syDA);
end

function g = gaussian(x, pos, wid)
    % Generate a Gaussian-shaped function
    % g = exp(-((x - pos) / (0.60056120439323 * wid)) ^ 2);
    g = exp(-((x - pos) / (0.60056120439323 * wid)) .^ 2);
end

function g = bw(x, pos, wid)
    % Generate a Breit-Wigner-shaped function
    w_bw = wid * 0.5;
    h = 1;
    g = (w_bw .^ 2 * h) ./ (w_bw .^ 2 + (x - pos) .^ 2);
end

function ry = FourierFilter(xvector, yvector, centerfrequency, filterwidth, filtershape, filtermode)
    % Computes a Fourier filter for a signal
    % Centerfrequency and filterwidth specify the frequency range of the pass band.
    % filtershape determines the sharpness of the cutoff, and filtermode specifies the filter type.

    % Adjust x and y vectors to row format
    xvector = reshape(xvector, 1, length(xvector));
    yvector = reshape(yvector, 1, length(yvector));
    fy = fft(yvector);
    lft1 = 1:(length(fy) / 2);
    lft2 = (length(fy) / 2 + 1):length(fy);

    % Compute filter shape based on filtermode
    if filtermode == 1 % 'Band-pass'
        ffilter1 = shapefunction(lft1, centerfrequency + 1, filterwidth, filtershape);
        ffilter2 = shapefunction(lft2, length(fy) - centerfrequency + 1, filterwidth, filtershape);
        ffilter = [ffilter1, ffilter2];
    elseif filtermode == 2 % 'High-pass'
        centerfrequency = length(xvector) / 2;
        ffilter1 = shapefunction(lft1, centerfrequency + 1, filterwidth, filtershape);
        ffilter2 = shapefunction(lft2, length(fy) - centerfrequency + 1, filterwidth, filtershape);
        ffilter = [ffilter1, ffilter2];
    elseif filtermode == 3 % 'Low-pass'
        centerfrequency = 0;
        ffilter1 = shapefunction(lft1, centerfrequency + 1, filterwidth, filtershape);
        ffilter2 = shapefunction(lft2, length(fy) - centerfrequency + 1, filterwidth, filtershape);
        ffilter = [ffilter1, ffilter2];
    elseif filtermode == 4 % 'Band-reject (notch)'
        ffilter1 = shapefunction(lft1, centerfrequency + 1, filterwidth, filtershape);
        ffilter2 = shapefunction(lft2, length(fy) - centerfrequency + 1, filterwidth, filtershape);
        ffilter = 1 - [ffilter1, ffilter2];
    elseif filtermode == 5 % 'Comb pass'
        n = 2;
        ffilter1 = shapefunction(lft1, centerfrequency + 1, filterwidth, filtershape);
        ffilter2 = shapefunction(lft2, length(fy) - centerfrequency + 1, filterwidth, filtershape);
        while n < 50
            ffilter1 = ffilter1 + shapefunction(lft1, n * (centerfrequency + 1), filterwidth, filtershape);
            ffilter2 = ffilter2 + shapefunction(lft2, length(fy) - n * (centerfrequency + 1), filterwidth, filtershape);
            n = n + 1;
        end
        ffilter = [ffilter1, ffilter2];
    elseif filtermode == 6 % 'Comb notch'
        n = 2;
        ffilter1 = shapefunction(lft1, centerfrequency + 1, filterwidth, filtershape);
        ffilter2 = shapefunction(lft2, length(fy) - centerfrequency + 1, filterwidth, filtershape);
        while n < 50
            ffilter1 = ffilter1 + shapefunction(lft1, n * (centerfrequency + 1), filterwidth, filtershape);
            ffilter2 = ffilter2 + shapefunction(lft2, length(fy) - n * (centerfrequency + 1), filterwidth, filtershape);
            n = n + 1;
        end
        ffilter = 1 - [ffilter1, ffilter2];
    end

    if length(fy) > length(ffilter)
        ffilter = [ffilter ffilter(1)];
    end
    ffy = fy .* ffilter; % Apply the filter in the frequency domain
    ry = real(ifft(ffy));
end

function g = shapefunction(x, pos, wid, n)
    % Generates a shaped peak function (Gaussian or Lorentzian)
```

```
    % Shape is Lorentzian (1/x^2) when n=0, Gaussian (exp(-x^2)) when n=1,
    % and becomes more rectangular as n increases.
    if n == 0
        g = ones(size(x)) ./ (1 + ((x - pos) / (0.5 * wid)) .^ 2);
    else
        g = exp(-((x - pos) / (0.6 * wid)) .^ (2 * round(n)));
    end
end

```

Notes and references

[1] T. O'Haver, A pragmatic introduction to signal processing, University of Maryland at College Park (1997).