AutoFFT v1.3.1 - User Manual

Table of Contents

License Information

Contact

Please, report any issues at https://github.com/CarlistRieekan/autofft/issues.

Alternatively, you can use e-mail carlist@ntis.zcu.cz.

Credits

Lead developer: Luboš Smolík, University of West Bohemia

Tester, new functionalities: Roman Pašek, Výzkumný a zkušební ústav Plzeň

Lead tester: Jan Rendl, University of West Bohemia

Introduction

Signal Processing Toolbox™ provides several functions to power spectrum estimation, including `pspectrum` and `stft`. Although these functions are excellent and well-documented, they might be cumbersome in some engineering applications. These applications include estimating magnitudes of vibration, noise and other discrete-time signals in engineering units or comparing theoretical results with measurements.

In such applications, you can use `autofft`, a Matlab function to estimate the discrete Fourier transform (DFT), which mimics options of the Brüel & Kjaer FFT analysers. Based on your input, autoFFT segments signal, applies window functions and spectral averaging. The resulting DFT can be returned in various engineering spectral units, including magnitude, RMS, peak-to-peak and power spectral density (PSD). `autofft` can also estimate spectral derivation or spectral integral of DFT and even perform the short-time Fourier transform (STFT).

When to use `autofft`

- You want to estimate magnitudes of components in your data in engineering units, e.g. m/s, V or Pa.
- You want to have better control over the setup of the frequency analyser than is possible with `pspectrum`.
- You want to use spectral derivation or spectral integration of DFT.

What's new in v1.3?

- **Bug fix v1.3.1:** When `'jwWeigthing'` is set to derivation or integration, times `t` are computed properly.
- A brand new user manual is a part of this release.
- Syntax change: `[s, f, t, setup] = autofft(___)` returns the times `t` at which the STFT is evaluated.
- Syntax change: The setup of the FFT analyser is now specified using `setup` structure; parameters used in the `setup` structure have the same names as parameters used in `pspectrum` and `stft` functions. Output variable `setup` can also be used as input. The structure `fftset` used in the previous versions can still be used to specify the analyser setup. However, `fftset` is no longer documented.
- New functionality: Overlap length can be now set in samples using the `'OverlapLength'` parameter.
- Functionality change: The default shape factor of the Kaiser window is now $\beta = 1.6$ instead of $\beta = 0.5$.
- Bug fix: `'OverlapPercentage'` 100% or higher is now automatically decreased to a realistic value.
- Bug fix: All windows are now treated as column vectors. Row vectors can cause an error during the application of a window to data.

Compatibility and requirements

- Created with R2021b
- Compatible with R2013a to R2021b
- Requires Signal Processing Toolbox

Syntax

```
s = autofft(xs, fs)
s = autofft(xs, ts)
```

```
s = autofft(___, setup)

[s, f] = autofft(___)
[s, f, setup] = autofft(___)
[s, f, t, setup] = autofft(___)
```

Description

`s = autofft(xs, fs)` returns the DFT or STFT `s` of `xs` using sampling frequency `fs` (Hz). `xs` can be either a vector or an array consisting of column vectors.

`s = autofft(xs, ts)` returns the DFT or STFT `s` of `xs` using a vector of time stamps `ts` (s). Also returns the frequencies `f` at which the DFT or STFT `s` is evaluated.

`[___] = autofft(___, setup)` performs the DFT or STFT name-value pair arguments specified in structure `setup`.

`[s, f] = autofft(___)` returns the frequencies `f` at which the DFT or STFT `s` is evaluated.

`[s, f, setup] = autofft(___)` returns the setup of the FFT analyser.

`[s, f, t, setup] = autofft(___)` returns the times `t` at which the STFT is evaluated.

Examples

Perform DFT of a single signal

Generate a testing voltage signal with a duration of 1 second with harmonic components at 80 Hz and 150 Hz.

```
dt = 1e-3;                                          % Sampling period (s)
T  = 1;                                             % Signal duration (s)
t  = dt:dt:T;                                       % Time vector
S  = sin(2*pi*50*t) + 0.5 * sin(2*pi*150*t);    % Voltage (V)
```

Corrupt the signal with zero-mean white noise with a variance of 4.

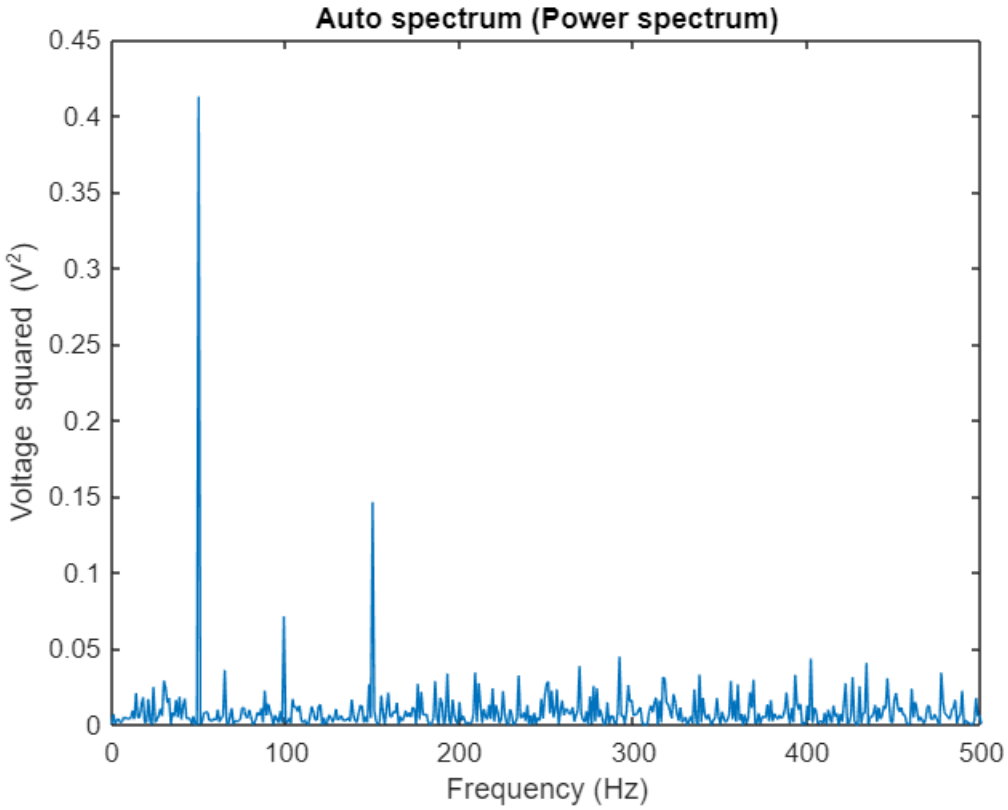
```
X = S + 2*randn(size(t));
```

Estimate the auto spectrum of the noisy signal.

```
[s, f] = autofft(X, t);
```

Plot the result. The magnitudes are roughly squares of the effective values (RMS). The discrepancies are present due to the added noise. The effect of the noise can be reduced by spectral averaging, which is in section Spectral Averaging.

```
figure;
plot(f, s);
xlabel("Frequency (Hz)")
ylabel("Voltage squared (V^2)")
title("Auto spectrum (Power spectrum)")
```



Modify the analyser setup

Using vector `X` from the previous example, one can check a setup used by the FFT analyser using the following code:

```
[s, f, setup] = autofft(X, t);
```

Now, the setup can be displayed.

```
disp(setup)
```

```
       SamplingFrequency: 1000
            DataDuration: 1
              DataLength: 1000
               FFTLength: 1000
          TimeResolution: 1
    FrequencyResolution: 1
      HighPassFrequency: NaN
       LowPassFrequency: 500
                 Window: "uniform"
    WindowNoiseBandwidth: 1
          OverlapLength: 500
      OverlapPercentage: 50
              Averaging: "linear"
       NumberOfAverages: 1
           SpectralUnit: "power"
            jwWeigthing: "none"
```

Change the `'SpectralUnit'` parameter to `'rms'` and use the modified setup in the FFT analyser.

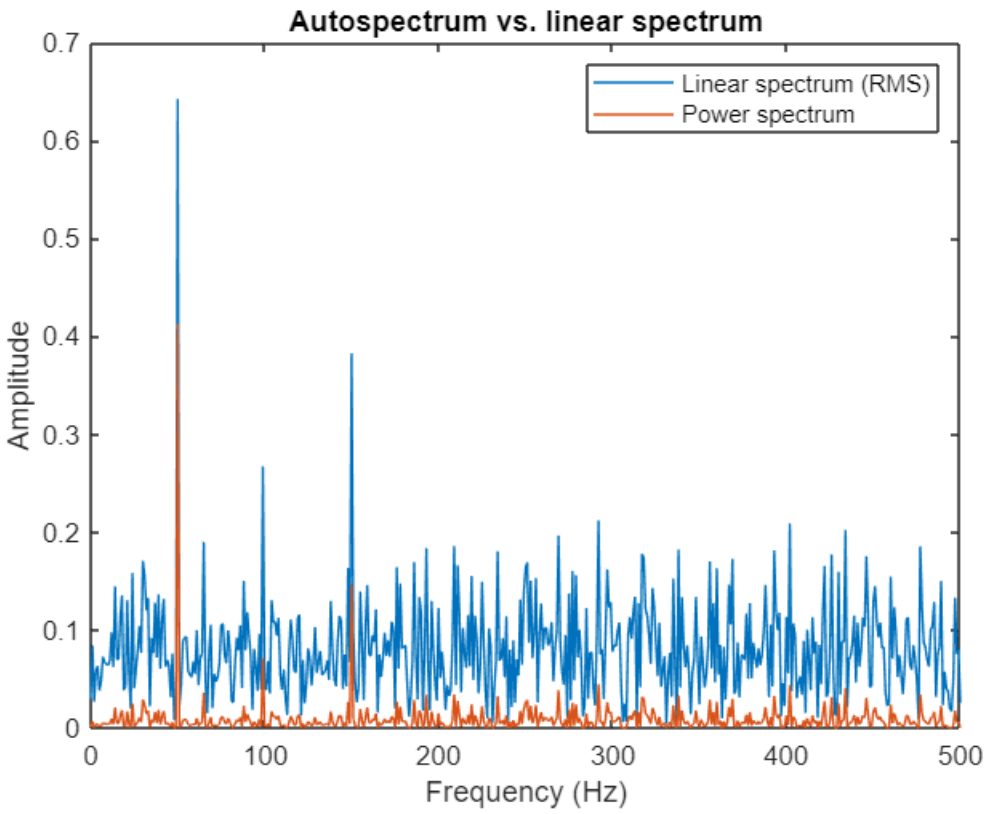
```
setup.SpectralUnit = "rms";
```

```
[sLin, f, setup] = autofft(X, t, setup);
disp(setup)
```

```
       SamplingFrequency: 1000
            DataDuration: 1
              DataLength: 1000
               FFTLength: 1000
          TimeResolution: 1
     FrequencyResolution: 1
       HighPassFrequency: NaN
        LowPassFrequency: 500
                  Window: "uniform"
    WindowNoiseBandwidth: 1
           OverlapLength: 500
       OverlapPercentage: 50
               Averaging: "linear"
        NumberOfAverages: 1
            SpectralUnit: "RMS"
             jwWeigthing: "none"
```

Plot the results.

```
figure;
plot(f, [sLin, s]);
xlabel("Frequency (Hz)")
ylabel("Amplitude")
title("Autospectrum vs. linear spectrum")
legend("Linear spectrum (RMS)", "Power spectrum")
```



The structure containing the analyser setup can also be constructed manually, field by field. In such a case, field names and their values are case insensitive, and unspecified parameters are generated internally.

```
userSetup = struct("spectralunit", "rms");
```

Use `userSetup` in the FFT analyser and compare the resulting spectrum with the previously computed spectrum stored in variable `sLin`.

```
sLin2 = autofft(X, t, userSetup);
maxError = max(abs(sLin - sLin2));
disp("Maximum difference between sLin and sLin2 is " + maxError)
```

```
Maximum difference between sLin and sLin2 is 0
```

Perform DFT of multiple signals

Generate three testing voltage signals with a duration of 2 seconds with various harmonic components.

```
dt = 1e-3;                                              % Sampling period (s)
fs = 1/dt;                                              % Sampling frequency (Hz)
T  = 2;                                                 % Signal duration (s)
t  = dt:dt:T;                                           % Time vector
S1 = sin(2*pi*50*t) + 0.5 * sin(2*pi*150*t);            % Signal 1 (V)
S2 = 0.7 * sin(2*pi*80*t) + 0.5 * sin(2*pi*100*t);      % Signal 2 (V)
S3 = 0.8 * sin(2*pi*90*t) + 0.9 * sin(2*pi*120*t);      % Signal 3 (V)
```

Arrange the testing signals into an array and corrupt the signals with zero-mean white noise with a variance of 1. Note that the testing signals have to be converted to **column vectors**.

```
X = [S1(:), S2(:), S3(:)];
X = X + randn(size(X));
```

Construct the structure containing the analyser setup and divide the testing signals into 1 s long segments using `'TimeResolution'` parameter. The segments are overlapping by 75 % and the Hann window is applied. The magnitude of the resulting spectra is equal to a zero-to-peak value.

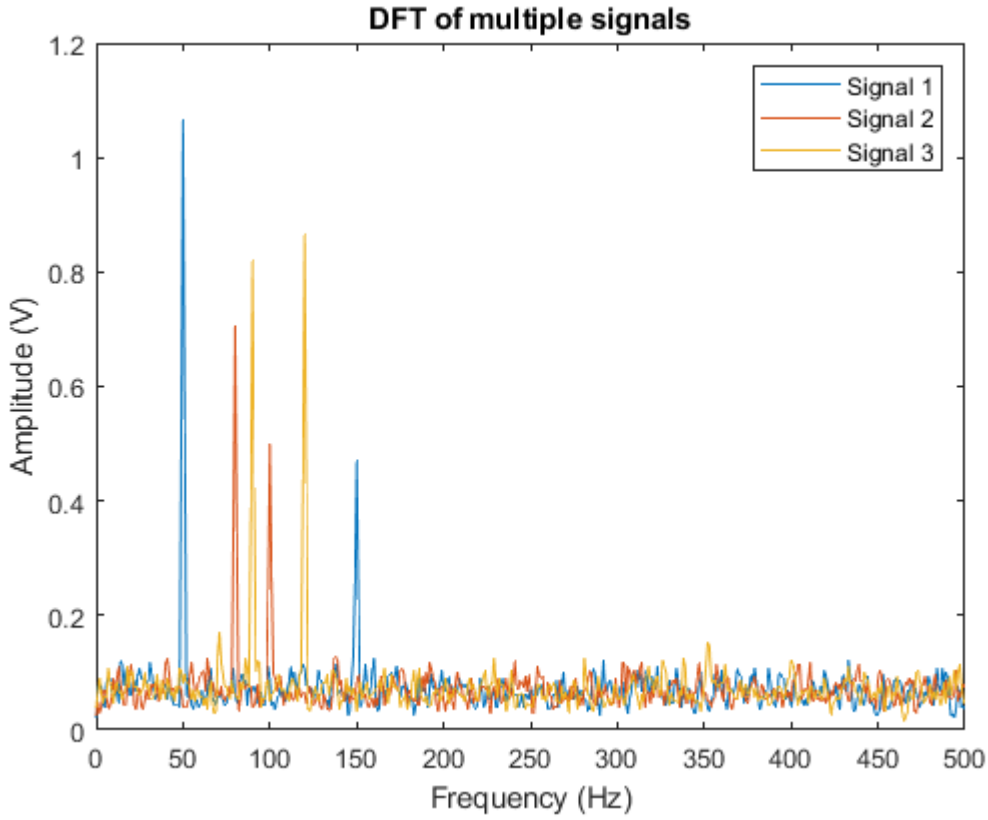
```
setup = struct("TimeResolution", 1, "Window", "hann", ...
               "OverlapPercentage", 75, "SpectralUnit", "peak");
```

Perform DFT of the testing signals and plot results.

```
[s, f] = autofft(X, t, setup);
disp(setup)
```

```
       TimeResolution: 1
               Window: "hann"
    OverlapPercentage: 75
         SpectralUnit: "peak"
```

```
plot(f, s);
xlabel("Frequency (Hz)")
ylabel("Amplitude (V)")
title("DFT of multiple signals")
legend("Signal 1", "Signal 2", "Signal 3")
```



Perform STFT of a single signal

Generate two seconds of a voltage controlled oscillator output, controlled by a sinusoid sampled at 10 kHz.

```
fs = 10e3;                                % Sampling frequency (Hz)
t  = 0:1/fs:2;                            % Time stamps (s)
x  = vco(sin(2*pi*t),[0.1 0.4]*fs,fs);  % Oscillator output (V)
```

Compute and plot the STFT of the signal. Use a Kaiser window of length 256 and shape parameter $\beta = 5$. Specify the length of overlap as 220 samples and the frequency resolution as 40 Hz. The spectral unit of the STFT is set to zero-to-peak. Note that in contrast to the `stft` function, additional parameter `'Averaging'` has to be set to `'none'`. This parameter switches between the DFT and STFT.

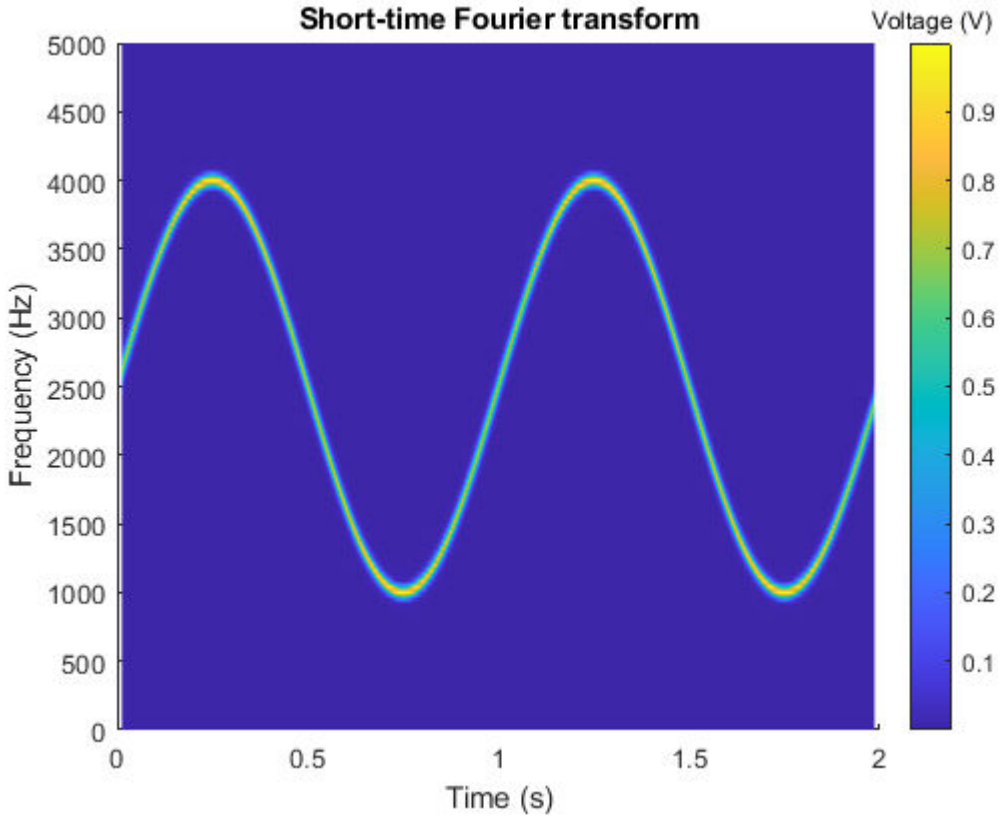
```
setup = struct("FrequencyResolution", 40, "Window", kaiser(256,5), ...
               "OverlapLength", 220, "SpectralUnit", "peak", "Averaging", "none");
[s, f, t, setup] = autofft(x, fs, setup);
```

```
Warning: The window function has been interpolated to 250 samples.
```

Note that if the window length does not correspond with the FFT length, autofft uses an interpolated window function. Now, plot the STFT with default colormap and view.

```
figure;
surface(t, f, s);
shading interp;
xlabel("Time (s)")
ylabel("Frequency (Hz)")
```

```
title("Short-time Fourier transform")
c = colorbar;
title(c, "Voltage (V)")
```



Perform STFT of multiple signals

Generate a quadratic chirps of length 2 s sampled at 1 kHz:

```
t  = 0:0.001:2;               % 2 secs @ 1kHz sample rate
yq = chirp(t,100,1,200,'q');  % Start @ 100Hz, cross 200Hz at t=1sec
```

Corrupt the chirp with zero-mean white noise and arrange both original and corrupted chirps into a single array. Note that the testing signals have to be converted to **column vectors**.

```
yqCor = yq + randn(size(yq));
ych = [yq(:), yqCor(:)];
```

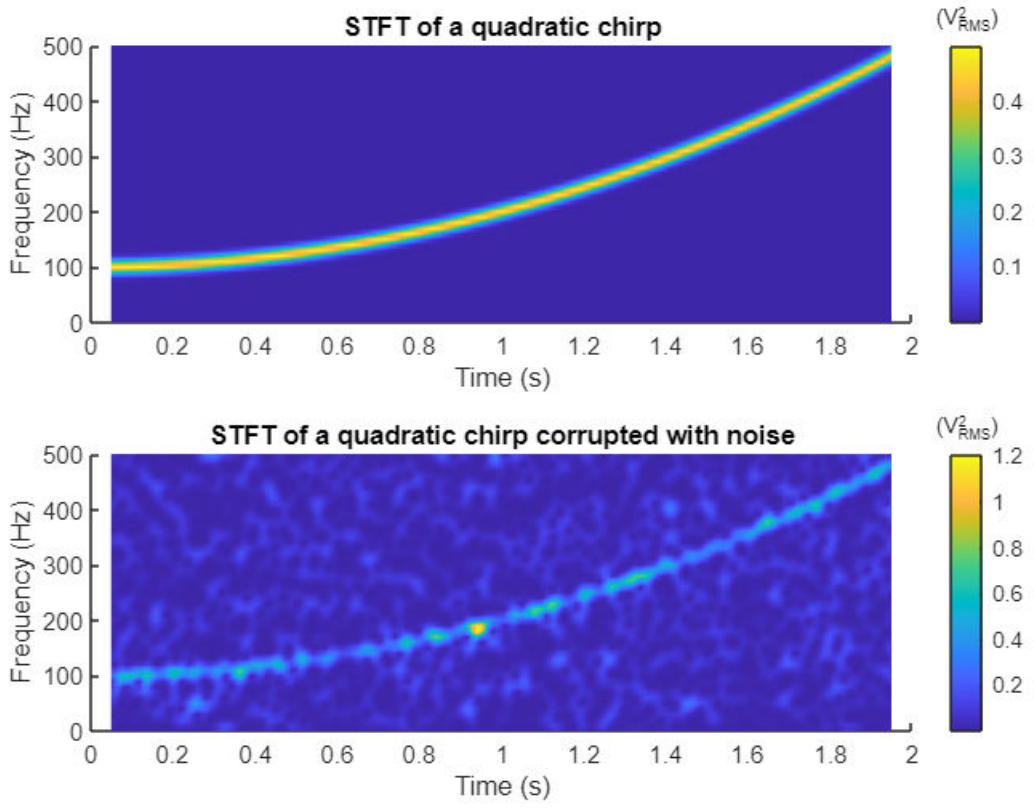
Compute and plot the STFT of both chirps. Set the frequency resolution to 10 Hz and use a Blackmann-Harris window with overlap 90 %.

```
setup = struct("FrequencyResolution", 10, "Window", "b", ...
               "OverlapPercentage", 90, "Averaging", "none");
[s, f, t, setup] = autofft(ych, t, setup);

tiledlayout(2, 1);
nexttile;
surface(t, f, squeeze(s(:,:,1)));
```

```
ylim([0 f(end)]);
shading interp;
xlabel("Time (s)")
ylabel("Frequency (Hz)")
title("STFT of a quadratic chirp")
c = colorbar;
title(c, "(V_{RMS}^2)")

nexttile;
surface(t, f, squeeze(s(:,:,2)));
ylim([0 f(end)]);
shading interp;
xlabel("Time (s)")
ylabel("Frequency (Hz)")
title("STFT of a quadratic chirp corrupted with noise")
c = colorbar;
title(c, "(V_{RMS}^2)")
```



Detailed Description of the Analyser Setup

Frequency resolution

The frequency resolution Δf is the difference in frequency between two consecutive spectral lines (bins) of the DFT. Two signal components are indistinguishable if their frequencies f_1 and f_2 are close relatively to the frequency resolution. By rule of thumb, two independent components should be separated at least by one spectral line if the uniform window is used, i.e. $|f_1 - f_2| \geq 2 \cdot \Delta f$.

The frequency resolution is equal to

$$\Delta f = \frac{f_s}{N_s} = \frac{1}{T}$$

where f_s is sampling frequency in Hz, N_s is a total number of samples in the analysed signal and T is a length of the analysed signal in s. **By default**, the analysed **signal is treated as one segment** and the frequency resolution of the DFT depends on the total number of samples N_s. The freqeuncy resolution can be adjusted in order to perform the STFT or employ the spectral averaging using one of the following parameters:

`'FFTLength'` — Number of DFT points

`total number of samples (default) | positive integer`

If `'FFTLength'` n_s is provided, then the analysed signal is divided into segments with each containing n_s samples.

Data type: `double | single`

`'TimeResolution'` — Length (in seconds) of the DFT

`duration of the signal (default) | positive scalar`

If `'TimeResolution'` T_s is provided, then the analysed signal is divided into segments with each containing $T_s \cdot f_s$ samples.

Data type: `double | single`

`'FrequencyResolution'` — Desired frequency resolution of the DFT

`fs / size(xs, 2) (default) | positive scalar`

If `'FrequencyResolution'` Δf is provided, then the analysed signal is divided into segments with each containing $f_s/\Delta f$ samples.

Data type: `double | single`

If the user-specified number of samples per segment is higher than actual length of the analysed signal , the value of n_s is internally set to $n_s = N_s$. Overlapping of adjoining segments is explained hereafter.

Note: If `'FFTLength'` is set, then both `'TimeResolution'` and `'FrequencyResolution'` are ignored by the analyser. If `'TimeResolution'` is set, then `'FrequencyResolution'` is ignored by the analyser. This behaviour is different from the `stft` function, where `'FFTLength'` and `'TimeResolution'` can be adjusted independently.

High-pass filtering

`'HighPassFrequency'` — Cut-off frequency of the high-pass filter

`NaN (default) | real number`

- If `fc` is `NaN` or `'HighPassFrequency'` is not provided, no high-pass filter is applied.
- If `fc = 0`, only the direct (DC) component is removed with the use of the `detrend` function.
- If `fc > 0`, the `detrend` function is applied and the high-pass filter with the cut-off frequency `fc` is applied subsequently.
- If `fc < 0`, then only the high-pass filter with the cut-off frequency `abs(fc)`.

Data type: `double | single`

The high-pass filter is often chosen if the analysed signal is contaminated with low-frequency data or a constant offset (DC), which is not in the interest frequency range. `autofft` has built-in digital high-pass filters, which correspond to high-pass filters used in **Brüel & Kjaer IDAe modules.** These high-pass filters are the 1st order elliptic filters with the slope -20 dB/dec. Attenuation at the cut-off frequency is -0.1 dB. A typical magnitude response is shown in fig. 1.

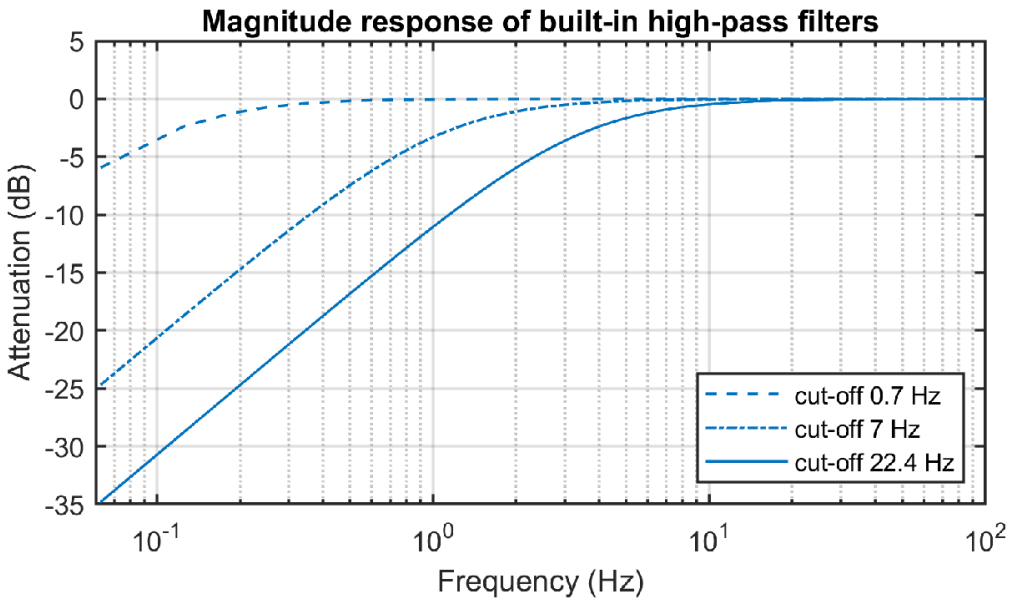


Figure 1: Magnitude response of several 1st order elliptic filters implemented in `autofft`.

Low-pass filtering

`'LowPassFrequency'` — Maximum frequency of interest

`half of the sampling frequency (default) | positive scalar up to half of the sampling frequency`

In the current version of `autofft`, `'LowPassFrequency'` is used to specify the maximum frequency of interest. Data at higher frequencies are not used for the spectral averaging and evaluation of spectral units and are not returned in output variables. No low-pass filter is applied in the time domain. Therefore, you can use this parameter to limit the output frequency range and slightly speed up computations. The **recommended** maximum value of `'LowPassFrequency'` is $f_s/2.56$, where f_s is the sampling frequency.

Data type: `double | single`

Applying a window function

`'Window'` — Applied window function / time weighting

`'uniform' (default) | 'blackmann' | 'flat-top' | 'hann' | 'hamming' | 'kaiser' | 'kaiserA.B' | vector`

When you apply a window function, each segment of the analysed signal is multiplied by this window function. This process is sometimes called *time weighting*. Time weighting aims to reduce unwanted effects as *spectral leakage* and *passband ripple* or improve the signal-to-noise ratio. The following window functions can be selected:

- `'u'` or `'uniform'` applies the uniform window, also known as the *rectangular*, *flat* or *box-car window*. The application of the uniform window is not proper time weighting because this window leaves the analysed signal as is. This window avoids spectral leakage into adjacent spectral lines, and therefore it is desirable to analyse broad-band signals such as pseudo-random noise, periodic random signals or bursts. It can also be helpful for the analysis of transients shorter than the length of the DFT.
- `'b'` or `'blackmann'` applies the *Blackman window.* This window function is smooth, and both absolute value and the first derivation are zero at the boundaries. The Blackman window is similar to the Kaiser-Bessel window used in the Brüel & Kjaer analysers. Because of its good selectivity is used when a separation of closely spaced frequency components is required. Typical applications include analysis of rotating machinery, analysis of spectra with sidebands and analysis of spectra containing two or more independent harmonic families.
- `'f'` or `'flat-top'` applies the *flat-top window*. This window has a very low picket-fence error (0.01 dB), which guarantees a high magnitude accuracy at the cost of the zero leakage factor. The flat-top window is commonly used for calibration, i.e. for the analysis of harmonic signals.
- `'h'` or `'hann'` applies the *Hann window*. The Hann window is a smooth weighting function that is equal to zero at the beginning and end. This window reduces spectral leakage whilst maintaining good selectivity. Therefore, it is often employed as a general-purpose window with typical applications, including analysis of rotating machinery, system analysis using random noise as excitation and analysis of spectra with a significant number of non-order related components.
- `'m'` or `'hamming'` applies the *Hamming window*. This window is similar to the Hann window, but equals to 0.08 at the beginning and end, which improves the spectral leakage. The Hamming window is a solid

general-purpose window in applications where magnitudes of the analysed components are of the same or similar orders.

- `'k'` or `'kaiser'` applies the *Kaiser window* with the shape factor $\beta = 1.6$. The properties of this window are somewhere between the uniform and Hann windows. Therefore, it is a good starting window to analyse signals with both harmonic and broad-band components.
- `'kA.B'` or `'kaiserA.B'` applies the *Kaiser window* with the shape factor $\beta = A.B$. The Kaiser window with $\beta = 6$ roughly approximates the Hann window, $\beta = 8.5$ corresponds with the Blackman window.
- Alternatively, the window function can be specified as a vector. In this case, the length of the vector should be the same as `'FFTLength'`. Otherwise, `autofft` interpolates the vector to match `'FFTLength'` employing the modified Akima algorithm. This behaviour is different from the `stft` function, where `'FFTLength'` and `'Window'` length can be adjusted independently.

Data type: `char | string | double | single`

Each window function has specific characteristics summarised in fig. 2 and tab. 1. These characteristics include:

- **Noise bandwidth** characterizes how the window would transmit noise. In general, windows with high values of the nouse bandwidth distort analysed broad-band signals.
- **Leakage** is related to a ratio between the window sidelobe attenuation and its main lobe width. A small spectral leakage reduces the frequency resolution (two close components can be indistinguishable) but allows detecting weak components in the vicinity of dominant ones. A high spectral leakage resolves closely spaced components but masks nearby weak ones.
- **The highest side lobe** and **side lobe fall-off rate characterise** the magnitude response of the window. Windows with high attenuation (e.g. flat-top) tend to mask weak components.
- **Main lobe width** determines the uncertainty with which the magnitude of the analysed component is estimated. This uncertainty is caused due to a passband ripple, which occurs when the frequency of the analysed signal does not coincide with the frequency of calculated spectral lines. Generally speaking, a wide main lobe eliminates the passband ripple but decreases the leakage.

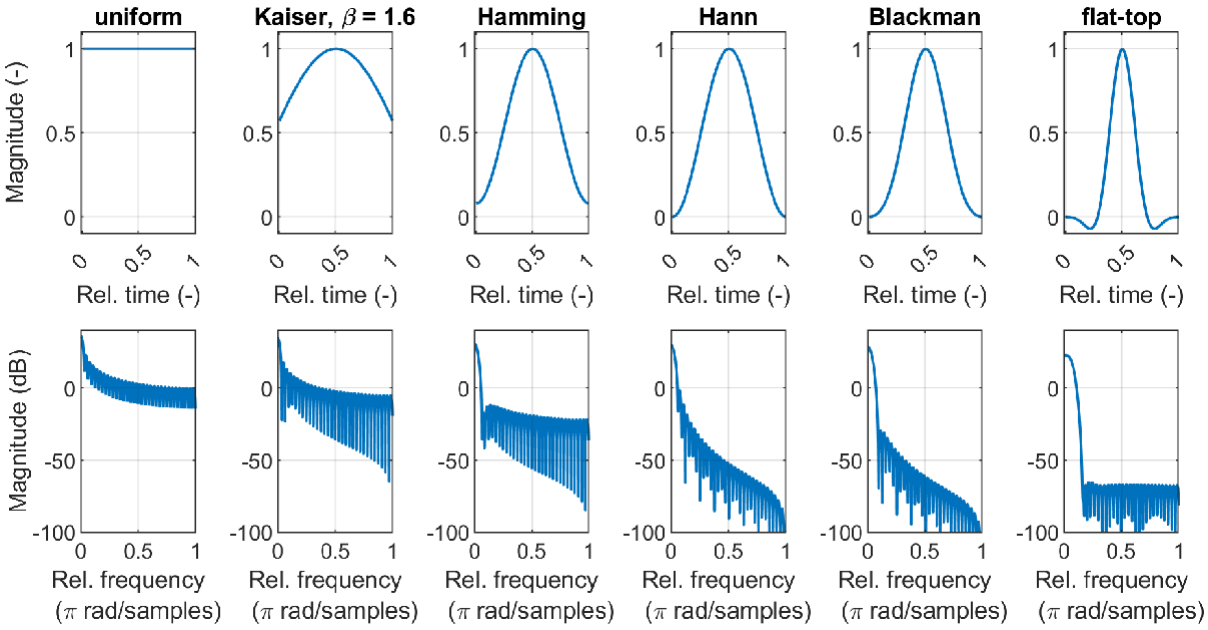


Figure 2: Shape and magnitude response of window functions implemented in `autofft`.

Parameter value	*Window*	*Noise bandwidth (Hz)*	*Leakage factor (%)*	*Rel. side-lobe attenuation (dB)*	*Main lobe width (π rad/sample)*
'u'	uniform	1.00	9.14	−13.3	0.0273
'k'	Kaiser, $\beta = 1.6$	1.02	3.92	−16.7	0.0017
'm'	Hamming	1.36	0.03	−42.5	0.0391
'h'	Hann	1.50	0.05	−31.5	0.0430
'b'	Blackman	1.73	0.00	−58.1	0.0508
'f'	flat-top	3.77	0.00	−88.0	0.1172

Table 1: Some characteristics of window functions implemented in `autofft`.

Overlapping

The analysed signal is divided into equally long segments whose length determines the frequency resolution. Two adjoining segments may or may not overlap. The amount of overlap determines the total number of segments. Assuming that o is the overlap length in samples, the total number of segments is

$$N_{\text{seg}} = \left\lfloor \frac{N_s - o}{n_s - o} \right\rfloor,$$

where N_s is the a total number of samples, n_s is the number of samples in one segment and $\lfloor\ \rfloor$ is the floor function. The relation allows for using negative overlaps. There are samples between two successive segments in such a case, which are omitted from the time-frequency analysis. Too high overlaps are lowered automatically per the relation

$$\text{If } o \geq n_s \Rightarrow o = n_s - 1.$$

The optimal overlap depends on the used window function, which adds weights to individual samples. The overlap can be selected so that loss of measured data is avoided and the overall weighting is flat (except parts at the beginning and end of the signal). The following rules, shown in fig. 3, apply for individual windows:

- **Uniform window:** 0 % overlap guarantees a truly flat overall weighting of the data. Any other overlap results in a step-wise overall weighting at the beginning and end of the signal.
- **Kaiser window with** $\beta = 1.6$: A reasonably flat overall weighting of the data can be achieved only with 99 % and higher overlaps. More common values, including 66.67 %, 75 % and 80 %, result in a non-uniform weighting. This ripple increases the uncertainty of the analysed magnitudes.
- **Hamming window:** a flat overall weighting of the data is achieved with overlaps 66.67 %, 75 %, 80 % and higher. Note that due to the non-zero values at the boundaries of the Hamming window, a ripple in the form of outlying samples can appear, see fig. 3. This ripple appears, e.g. if the overlap is 66.67 %, but the segment length in samples is indivisible by 3.
- **Hann window:** a flat overall weighting of the data is achieved with overlaps 66.67 %, 75 %, 80 % and higher.
- **Blackman window:** a flat overall weighting of the data is achieved with overlaps 66.67 %, 75 %, 80 % and higher.
- **Flat-top window:** a reasonably flat overall data weighting is achieved with overlaps of 80 % and higher.

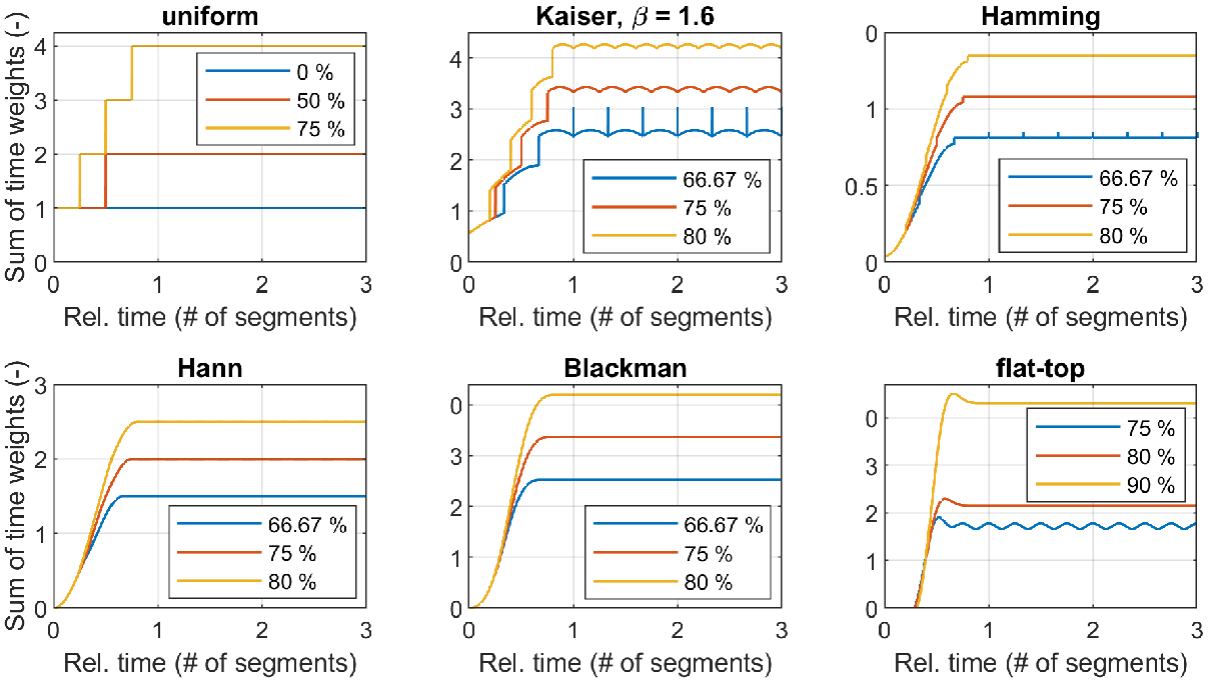


Figure 3: Overall weighting of the data for window functions implemented in `autofft` with various amounts of the overlap.

`'OverlapLength'` — Number of overlapped samples of two successive segments

`50 % of 'FFTLength' (default) | integer lower than 'FFTLength'`

`'OverlapLength'` indicates the overlap length in samples o. If both `'OverlapLength'` and `'OverlapPercentage'` are provided, then `'OverlapPercentage'` is omitted.

Data type: `double | single`

`'OverlapPercentage'` — Overlap percentage of two successive segments

`50 (default) | <0, 100>`

`'OverlapPercentage'` indicates the overlap length relative to the length of the window o_{rel}. If this parameter is provided, the overlap length in samples is computed as

$$o = \left[\frac{o_{\text{rel}} \cdot n_s}{100}\right],$$

where n_s is the number of samples in one segment and $[\ ]$ is the round function.

Data type: `double | single`

Spectral averaging

`'Averaging'` — Algorithm for spectral averaging

`'linear' (default) | 'energy' | 'max' | 'median' | 'min' | 'var' | 'none'`

This parameter enables averaging of a series of the DFT and basically serves as a switch between the output in a form of the averaged DFT or STFT. If the avearging is enabled, the output is averaged in the frequency domain. The averaging takes place after the computation of spectral units. This method eliminates phase shifts of the frequency components in the individual segments and results in the average of the magnitudes.

Following averaging modes are available:

- `'linear'` applies linear averaging which places equal emphasis on all samples. Assuming that $s_{j,i}$ is a real value located the*j*-th spectral line in the i-th segment, the linear averaging reads $\bar{s}_j = \frac{1}{N_{\text{seg}}} \sum_{i=1}^{N_{\text{seg}}} s_{j,i}$.
- `'energy'` or `'rms'` calculates the averages of the squared values of all values, i.e. $\bar{s}_j = \sqrt{\frac{1}{N_{\text{seg}}} \sum_{i=1}^{N_{\text{seg}}} (s_{j,i})^2}$. Due to the nature of the used eqution, this averaging mode puts more weight to the higner magnitudes and it is commonly used in acoustic applications.
- `'max', 'maximum'` or `'peak'` holds the *maximum value* at the*j*-th spectral line, i.e. $\bar{s}_j = \max\left(s_{j,1}, \ldots, s_{j,N_{\text{seg}}}\right)$. This averaging mode can be used to estimate the worst case scenario in periodic signals or to evaluate the upper envelope of broad-band signals.
- `'med'` or `'median'` holds the *median* at the*j*-th spectral line, i.e. $\bar{s}_j = \text{median}\left(s_{j,1}, \ldots, s_{j,N_{\text{seg}}}\right)$. This averaging mode is useful to estimate mean values of stationary signals corrupted by transients.

- `'min'` or `'minimum'` holds the *minimum value* at the*j*-th spectral line, i.e. $\bar{s}_j = \min\left(s_{j,1}, \ldots, s_{j,N_{\text{seg}}}\right)$. This averaging mode is usually used to evaluate the lower envelope of broad-band signals.
- `'var'` or `'variance'` computes the *variance* at the*j*-th spectral line, i.e $\bar{s}_j = \frac{1}{N_{\text{seg}} - 1} \sum_{i=1}^{N_{\text{seg}}} |s_{j,i} - \mu|^2$ where $\mu = \frac{1}{N_{\text{seg}}} \sum_{i=1}^{N_{\text{seg}}} s_{j,i}$. The variance is essential to evaluate the dispersion of broad-band signals and for the classification of the components with the constant frequency and time-varying magnitude.
- `'none'` disables the spectral averaging and returns the STFT of the input.

Data type: `char | string`

Spectral differentiation and integration - $j\omega$ weighting

`'jwWeigthing'` — Frequency-domain post-weighting

`'1' (default) | '1/jw2' | '1/jw' | 'jw' | 'jw2'`

A frequency-domain post-weighting can be used to estimate derivation or integral of a measured signal. The post-weighting is beneficial if e.g. velocity of vibration has to be estimated from the measured acceleration. The post-weighting is performed before the computation of spectral units and spectral averaging.

The single integration of the DFT at the i-th spectral line is computed using the formula

$$\bar{X}_i = \frac{X_i}{\mathrm{j} \cdot 2 \cdot \pi \cdot f_i},$$

where X_i is a complex number representing the DFT at the i-th spectral line, f_i is the frequency of the i-th spectral line in Hz and j is the imaginary unit.

Similarly, the single differentiation of the DFT at the i-th spectral line is computed using the formula

$$\bar{X}_i = \mathrm{j} \cdot 2 \cdot \pi \cdot f_i \cdot X_i.$$

Note: The frequency-domain integration should be used as a last resort to determine the integral of the measured signal, because it tends to overestimate magnitudes at low frequencies.

Following frequency-domain post-weighting modes are available:

- `'1/jw2'` or `'double integration'` performs a double integration.
- `'1/jw'` or `'single integration'` performs a single integration.
- `'1'` or `'none'` leaves the resulting DFT as is.
- `'jw'` or `'single differentiation'` performs a single differentiation.
- `'jw2'` or `'double differentiation'` performs a double differentiation.

Data type: `char | string`

Spectral units

`'SpectralUnit'` — Engineering unit of the output spectra

`'pow' (default) | 'rms' | 'pk' | 'pp' | 'psd' | 'rsd'`

Various types of the analysed signals require different spectral units. `autofft` uses the complex values of the DFT for the spectral differentiation and integration and transforms the complex spectrum into the linear spectrum or autospectrum before the spectral averaging is performed. The available spectral units include:

- `'pow'` or `'power'` estimates *power spectrum* (also called *autospectrum*). This spectral unit is useful for analysing general stationary deterministic signals. Per definition, the power spectrum is the square of the root mean square value.
- `'rms'` estimates the *root mean square magnitude* of the signal. This spectral unit is commonly used for analysing the velocity of vibration.
- `'pk'` or `'0-pk'` estimates the *peak magnitude* of the signal. This spectral unit is commonly used for analysing the acceleration of vibration.
- `'pp'` or `'pk-pk'` estimates the *peak-to-peak magnitude* of the signal. This spectral unit is commonly used for analysing the displacement of vibration.
- `'psd'` or `'asd'` estimates *power spectral density* of the analysed signal. The power spectral density is a correct unit for displaying broad-band random signals. If applied on the broad-band signal, its magnitude is independent on the frequency resolution of the DFT.
- `'rsd'` or `'rmssd'` is equal to the *square root of the power spectral density*. Similarly to the PSD, it is used for analysing of broad-band signals.

Data type: `char | string`

References

1. RANDALL, R. B. *Frequency Analysis*. 3rd edition. Naerum (Denmark): Brüel & Kjaer, 1987.
2. *PULSE LabShop* [software]. v21.0.0. Naerum (Denmark): Brüel & Kjaer, 2016.
3. *LabView* [software]. v2019. Austin (USA): National Instruments, 2019.
4. *Simcenter Testlab* [software]. v2019.1. Munich (Germany): Siemens, 2019.

Appendix A: Road Map

Features which I plan to implement in future releases of `autofft` (in no specific order):

- support for tables and timetables,
- new spectral units (Fourier spectrum, energy spectral density)
- new analysis modes (full-spectrum, frequency zoom, envelope analysis, Bode plot),
- advanced multi-buffering and triggering,
- acoustic weighting, and
- GUI (?)

Appendix B: Historical Release Notes

v1.3.1

- Bug fix: When `'jwWeigthing'` is set to derivation or integration, times `t` are computed properly.

v1.3.0

- A brand new user manual is a part of this release.
- Syntax change: `[s, f, t, setup] = autofft(___)` returns the times `t` at which the STFT is evaluated.
- Syntax change: The setup of the FFT analyser is now specified using `setup` structure; parameters used in the `setup` structure have the same names as parameters used in `pspectrum` and `stft` functions. Output variable `setup` can also be used as input. The structure `fftset` used in the previous versions can still be used to specify the analyser setup. However, `fftset` is no longer documented.
- New functionality: Overlap length can be now set in samples using the `'OverlapLength'` parameter.
- Functionality change: The default shape factor of the Kaiser window is now $\beta = 1.6$ instead of $\beta = 0.5$.
- Bug fix: `'OverlapPercentage'` 100% or higher is now automatically decreased to a realistic value.
- Bug fix: All windows are now treated as column vectors. Row vectors could cause an error during the application of a window to data.

v1.2.5a (hotfix)

- Bug fix: Outputs which appeared twice in `setup` now appear only once.

v1.2.5

- New functionality: An optional highpass filtering has been implemented.

v1.2.4b (hotfix)

- Bug fix: Overlap length (in samples) returned in setup is now correct.

v1.2.4a

- Bug fix: Error occuring during the estimation of auto spectrum has been fixed.

v1.2.4

- Documentation has been improved.
- Syntax change: Results of the STFT of multiple signals are now returned as 3D array rather than cell array of 2D arrays.
- New functionality: New types of averaging: *median filter* and *variance of spectral unit*.
- Performance has been improved significantly.
- Accuracy of PSD and RMSSD estimates has been slightly improved.
- Dealing with a content at the Nyquist frequency has been improved.

v1.2.3

- Syntax change: The analyser setup can be returned as an output variable.
- New functionality: User can define frequency resolution of the analyser using `'df'` parameter.
- New functionality: The window function can be directly specified as a vector.
- Warning messages are now displayed if user specified paramters are changed internally.

v1.2.2a (hotfix)

- Bug fix: Error occuring during peak hold averaging has been fixed.

v1.2.2

- New functionality: The Kaiser-Bessel window parameter (beta) can now be specified.
- Autospectrum is now properly square of RMS rather than square of 0-Pk.
- Relations for evaluation of PSD and RMSPSD now consider the noise power bandwidth of the used window function.

v1.2.1

- Syntax change: Options for `'unit'` and `'peak'` have been merged (into `'unit'`).
- New functionality: Maximum frequency can now be specified using `'lowpass'` parameter.
- New functionality: New types of averaging have been added: *no averaging*, *energy averaging* and *minimum value hold*.
- Relations for evaluation of PSD and RMSPSD have been improved.
- Dealing with a content at the Nyquist frequency has been improved.

v1.2.0

- Syntax change: Input parameters are now specified in a structured variable. Due to this change, v1.2.0 is not compatible with the previous releases.

v1.1.2

- Syntax change: Input parameter `xs` can now be an array.

v1.1.1

- Performance has been improved.
- Handling of input vectors with the even number of samples has been improved.

v1.1.0

- Relations for evaluation of spectral units now consider the influence of non-uniform windows.
- Parameters `'nwin'` and `'overlap'` can now be skipped by user.