dsptoolbox

Release 0.0.5

Nicolas Franco-Gomez

Jan 11, 2023

CONTENTS

CHAPTER

ONE

README

This is a toolbox in form of a python package that contains algorithms to be used in dsp (digital signal processing) projects.

This project is under active development and it will take some time until it reaches a certain level of maturity. Beware that backwards compatibility is not an actual concern and important changes to the API might come in the future. If you find some implementations interesting or useful, please feel free to use it for your projects and expand or change functionalities.

1.1 Getting Started

Check out the examples for some basic examples of the dsptoolbox package and refer to the documentation for the complete description of classes and methods.

1.2 Installation

Use pip to install dsptoolbox

```
$ pip install dsptoolbox
```

(Requires Python 3.10 or higher)

CHAPTER

TWO

CLASSES (DSPTOOLBOX.CLASSES)

The main classes are listed here. For some filterbanks, special classes are used but the api works almost equally as for the main FilterBank class.

2.1 Signal

Signal class

class `dsptoolbox.classes.signal_class.`**`Signal`**(*path: Optional[str] = None*, *time_data=None*, *sampling_rate_hz: int = 48000*, *signal_type: str = 'general'*, *signal_id: str = ''*)

Bases: `object`

Class for general signals (time series). Most of the methods and supported computations are focused on audio signals, but some features might be generalizable to all kinds of time series.

Attributes

number_of_channels
sampling_rate_hz
signal_id
signal_type
time_data
time_data_imaginary
time_vector_s

Methods

add_channel([path, new_time_data, ...])	Adds new channels to this signal object.
copy()	Returns a copy of the object.
get_channels(channels)	Returns a signal object with the selected channels.
get_coherence()	Returns the coherence matrix.
get_csm([force_computation])	Get Cross spectral matrix for all channels with the shape (frequencies, channels, channels)
get_spectrogram([channel_number, ...])	Returns a matrix containing the STFT of a specific channel.
get_spectrum([force_computation])	Returns spectrum.
plot_coherence([returns])	Plots coherence measurements if there are any.
plot_csm([range_hz, logx, with_phase, returns])	Plots the cross spectral matrix of the multichannel signal.
plot_group_delay([range_hz, returns])	Plots group delay of each channel.
plot_magnitude([range_hz, normalize, ...])	Plots magnitude spectrum.
plot_phase([range_hz, unwrap, returns])	Plots phase of the frequency response, only available if the method for the spectrum parameters is not welch.
plot_spectrogram([channel_number, logfreqs, ...])	Plots STFT matrix of the given channel.
plot_time([returns])	Plots time signals.
remove_channel([channel_number])	Removes a channel.
save_signal([path, mode])	Saves the Signal object as wav, flac or pickle.
set_coherence(coherence)	Sets the coherence measurements of the transfer function.
set_csm_parameters([window_length_samples, ...])	Sets all necessary parameters for the computation of the CSM.
set_spectrogram_parameters([channel_number, ...])	Sets all necessary parameters for the computation of the spectrogram.
set_spectrum_parameters([method, smoothe, ...])	Sets all necessary parameters for the computation of the spectrum.
set_streaming_position([position_samples])	Sets the start position for streaming.
set_window(window)	Sets the window used for the IR.
show_info()	Prints all the signal information to the console.
stream_samples(blocksize_samples[, signal_mode])	Returns block of samples to be reproduced in an audio stream.
swap_channels(new_order)	Rearranges the channels in the new given order.

__init__(*path: Optional[str] = None*, *time_data=None*, *sampling_rate_hz: int = 48000*, *signal_type: str = 'general'*, *signal_id: str = ''*)

Signal class that saves time data, channel and sampling rate information as well as spectrum, cross-spectral matrix and more.

Parameters

path
[str, optional] A path to audio files. Reading is done with soundfile. Wave and Flac audio files are accepted. Default: *None*.

time_data
[array-like, *np.ndarray*, optional] Time data of the signal. It is saved as a matrix with the form (time samples, channel number). Default: *None*.

sampling_rate_hz
[int, optional] Sampling rate of the signal in Hz. Default: 48000.

signal_type
[str, optional] A generic signal type. Some functionalities are only unlocked for impulse responses with *'ir'*, *'h1'*, *'h2'*, *'h3'* or *'rir'*. Default: *'general'*.

signal_id
[str, optional] An even more generic signal id that can be set by the user. Default: *''*.

Methods

Time data:	add_channel, remove_channel, swap_channels, get_channels.
Spectrum:	set_spectrum_parameters, get_spectrum.
Cross spectral matrix:	set_csm_parameters, get_csm.
Spectrogram:	set_spectrogram_parameters, get_spectrogram.
Plots:	plot_magnitude, plot_time, plot_spectrogram, plot_phase, plot_csm.
General:	save_signal, get_stream_samples.
Only for `signal_type in ('rir', 'ir', 'h1', 'h2', 'h3')`:	set_window, set_coherence, plot_group_delay, plot_coherence.

`add_channel`(*path: Optional[str] = None*, *new_time_data: Optional[ndarray] = None*, *sampling_rate_hz: Optional[int] = None*, *padding_trimming: bool = True*)

Adds new channels to this signal object.

Parameters

path
[str, optional] Path to the file containing new channel information.

new_time_data
[*np.ndarray*, optional] np.array with new channel data.

sampling_rate_hz
[int, optional] Sampling rate for the new data

padding_trimming
[bool, optional] Activates padding or trimming at the end of signal in case the new data does not match previous data. Default: *True*.

`copy`()

Returns a copy of the object.

Returns

new_sig
[*Signal*] Copy of Signal.

`get_channels`(*channels*)

Returns a signal object with the selected channels. Note: first channel index is 0!

Parameters

channels
[array-like or int] Channels to be returned as a new Signal object.

Returns

new_sig
[*Signal*] New signal object with selected channels.

`get_coherence()`

Returns the coherence matrix.

Returns

f
[*np.ndarray*] Frequency vector.

coherence
[*np.ndarray*] Coherence matrix.

`get_csm`(*force_computation=False*)

Get Cross spectral matrix for all channels with the shape (frequencies, channels, channels)

Returns

f_csm
[*np.ndarray*] Frequency vector.

csm
[*np.ndarray*] Cross spectral matrix with shape (frequency, channels, channels).

`get_spectrogram`(*channel_number: int = 0*, *force_computation: bool = False*)

Returns a matrix containing the STFT of a specific channel.

Parameters

channel_number
[int, optional] Channel number for which to compute the STFT. Default: 0.

force_computation
[bool, optional] Forces new computation of the STFT. Default: False.

Returns

t_s
[*np.ndarray*] Time vector.

f_hz
[*np.ndarray*] Frequency vector.

spectrogram
[*np.ndarray*] Spectrogram.

`get_spectrum`(*force_computation=False*)

Returns spectrum.

Parameters

force_computation
[bool, optional] Forces spectrum computation.

Returns

spectrum_freqs
[*np.ndarray*] Frequency vector.

spectrum
[*np.ndarray*] Spectrum matrix for each channel.

`property number_of_channels: int`

`plot_coherence`(*returns: bool = False*)

Plots coherence measurements if there are any.

Parameters

returns
[bool, optional] When *True*, the plot's figure and axis are returned. Default: *False*.

Returns

fig, ax
Only returned if *returns=True*.

`plot_csm`(*range_hz=[20, 20000.0]*, *logx: bool = True*, *with_phase: bool = True*, *returns: bool = False*)

Plots the cross spectral matrix of the multichannel signal.

Parameters

range_hz
[array-like with length 2, optional] Range of Hz to be showed. Default: [20, 20e3].

logx
[bool, optional] Logarithmic x axis. Default: *True*.

with_phase
[bool, optional] When *True*, the unwrapped phase is also plotted. Default: *True*.

returns
[bool, optional] Gives back figure and axis objects. Default: *False*.

Returns

fig, ax if *returns* = *True*.

`plot_group_delay`(*range_hz=[20, 20000]*, *returns: bool = False*)

Plots group delay of each channel. Only works if *signal_type in ('ir', 'h1', 'h2', 'h3', 'rir', chirp, noise, dirac)*.

Parameters

range_hz
[array-like with length 2, optional] Range of frequencies for which to show group delay. Default: [20, 20e3].

returns
[bool, optional] When *True*, the plot's figure and axis are returned. Default: *False*.

Returns

fig, ax
Only returned if *returns=True*.

`plot_magnitude`(*range_hz=[20, 20000.0]*, *normalize: str = '1k'*, *range_db=None*, *smoothe: int = 0*, *show_info_box: bool = False*, *returns: bool = False*)

Plots magnitude spectrum. Change parameters of spectrum with set_spectrum_parameters. NOTE: Smoothing is only applied on the plot data.

Parameters

range_hz
[array-like with length 2, optional] Range for which to plot the magnitude response. Default: [20, 20000].

normalize
[str, optional] Mode for normalization, supported are *'1k'* for normalization with value at frequency 1 kHz or *'max'* for normalization with maximal value. Use *None* for no normalization. Default: *'1k'*.

range_db
[array-like with length 2, optional] Range in dB for which to plot the magnitude response. Default: *None*.

smoothe
[int, optional] Smoothing across the (1/smoothe) octave band. Default: 0 (no smoothing).

show_info_box
[bool, optional] Plots a info box regarding spectrum parameters and plot parameters. If it is str, it overwrites the standard message. Default: *False*.

returns
[bool, optional] When *True* figure and axis are returned. Default: *False*.

Returns

fig, ax
Only returned if *returns=True*.

`plot_phase`(*range_hz=[20, 20000.0]*, *unwrap: bool = False*, *returns: bool = False*)

Plots phase of the frequency response, only available if the method for the spectrum parameters is not welch.

Parameters

range_hz
[array-like with length 2, optional] Range of frequencies for which to show group delay. Default: [20, 20e3].

unwrap
[bool, optional] When *True*, the unwrapped phase is plotted. Default: *False*.

returns
[bool, optional] When *True*, the plot's figure and axis are returned. Default: *False*.

Returns

fig, ax
Only returned if *returns=True*.

`plot_spectrogram`(*channel_number: int = 0*, *logfreqs: bool = True*, *returns: bool = False*)

Plots STFT matrix of the given channel.

Parameters

channel_number
[int, optional] Selected channel to plot spectrogram. Default: 0 (first).

logfreqs
[bool, optional] When *True*, frequency axis is plotted logarithmically. Default: *True*.

returns
[bool, optional] When *True*, the plot's figure and axis are returned. Default: *False*.

Returns

fig, ax
Only returned if *returns=True*.

`plot_time`(*returns: bool = False*)

Plots time signals.

Parameters

returns
[bool, optional] When *True*, the plot's figure and axis are returned. Default: *False*.

Returns

fig, ax
Only returned if *returns=True*.

`remove_channel`(*channel_number: int = -1*)

Removes a channel.

Parameters

channel_number
[int, optional] Channel number to be removed. Default: -1 (last).

`property sampling_rate_hz: int`

`save_signal`(*path: str = 'signal'*, *mode: str = 'wav'*)

Saves the Signal object as wav, flac or pickle.

Parameters

path
[str, optional] Path for the signal to be saved. Use only folders/name (without format). Default: *'signal'* (local folder, object named signal).

mode
[str, optional] Mode of saving. Available modes are *'wav'*, *'flac'*, *'pickle'*. Default: *'wav'*.

`set_coherence`(*coherence: ndarray*)

Sets the coherence measurements of the transfer function. It only works for *signal_type = ('ir', 'h1', 'h2', 'h3', 'rir')*.

Parameters

coherence
[*np.ndarray*] Coherence matrix.

`set_csm_parameters`(*window_length_samples: int = 1024*, *window_type='hann'*, *overlap_percent=75*, *detrend=True*, *average='mean'*, *scaling='power'*)

Sets all necessary parameters for the computation of the CSM.

Parameters

window_length_samples
[int, optional] Window size. Default: 1024.

overlap_percent
[float, optional] Overlap in percent. Default: 75.

detrend
[bool, optional] Detrending (subtracting mean). Default: True.

average
[str, optional] Averaging method. Choose from *'mean'* or *'median'*. Default: *'mean'*.

scaling
[str, optional] Type of scaling. *'power'* or *'spectrum'*. Default: *'power'*.

`set_spectrogram_parameters`(*channel_number: int = 0*, *window_length_samples: int = 1024*, *window_type: str = 'hann'*, *overlap_percent=75*, *detrend: bool = True*, *padding: bool = True*, *scaling: bool = False*)

Sets all necessary parameters for the computation of the spectrogram.

Parameters

window_length_samples
[int, optional] Window size. Default: 1024.

overlap_percent
[float, optional] Overlap in percent. Default: 75.

detrend
[bool, optional] Detrending (subtracting mean). Default: True.

padding
[bool, optional] Padding signal in the beginning and end to center it. Default: True.

scaling
[bool, optional] Scaling or not after np.fft. Default: False.

`set_spectrum_parameters`(*method='welch'*, *smoothe: int = 0*, *window_length_samples: int = 1024*, *window_type='hann'*, *overlap_percent=50*, *detrend=True*, *average='mean'*, *scaling='power'*)

Sets all necessary parameters for the computation of the spectrum.

Parameters

method
[str, optional] *'welch'* or *'standard'*. Default: *'welch'*.

smoothe
[int, optional] Smoothing across (1/smoothe) octave bands using a hamming window. Smoothes magnitude AND phase. For accesing the smoothing algorithm, refer to *dsptoolbox._general_helpers._fractional_octave_smoothing()*. If smoothing is applied here, *Signal.get_spectrum()* returns the smoothed spectrum as well. Default: 0 (no smoothing).

window_length_samples
[int, optional] Window size. Default: 1024.

window_type
[str,optional] Choose type of window from the options in scipy.windows. Default: *'hann'*.

overlap_percent
[float, optional] Overlap in percent. Default: 50.

detrend
[bool, optional] Detrending (subtracting mean). Default: True.

average
[str, optional] Averaging method. Choose from *'mean'* or *'median'*. Default: *'mean'*.

scaling
[str, optional] Type of scaling. *'power'* or *'spectrum'*. Default: *'power'*.

`set_streaming_position`(*position_samples: int = 0*)

Sets the start position for streaming.

Parameters

position_samples
[int, optional] Position (in samples) for starting the stream. Default: 0.

`set_window`(*window*)

Sets the window used for the IR. It only works for *signal_type in ('ir', 'h1', 'h2', 'h3', 'rir')*.

Parameters

window
[*np.ndarray*] Window used for the IR.

`show_info()`

Prints all the signal information to the console.

property signal_id: str

property signal_type: str

`stream_samples`(*blocksize_samples: int*, *signal_mode: bool = True*)

Returns block of samples to be reproduced in an audio stream. Use *set_streaming_position* to define start of streaming.

Parameters

blocksize_samples
[int] Blocksize to be returned (in samples).

signal_mode
[bool, optional] When *True* a *Signal* object is returned containing the desired samples. *False* returns a numpy array. Default: *True*.

Returns

sig
[*np.ndarray* or *Signal*] Numpy array with samples used for reproduction with shape (time_samples, channels) or *Signal* object.

stop_flag
[bool] When *True*, all samples of the available signal have been reproduced and the stream should be stopped.

`swap_channels`(*new_order*)

Rearranges the channels in the new given order.

Parameters

new_order
[array-like] New rearrangement of channels.

property time_data: ndarray

property time_data_imaginary: numpy.ndarray | None

property time_vector_s: ndarray

2.2 MultiBandSignal

class dsptoolbox.classes.multibandsignal.**MultiBandSignal**(*bands: list = []*, *same_sampling_rate: bool = True*, *info: dict = {}*)

Bases: `object`

The *MultiBandSignal* class contains multiple Signal objects which are to be interpreted as frequency bands of the same signal. Since every signal has also multiple channels, the object resembles somewhat a 3D-Matrix representation of a signal.

The *MultiBandSignal* can be multirate system if the attribute *same_sampling_rate* is set to *False*. A dictionary called *info* can also carry all kinds of metadata that might characterize the signals.

Attributes

bands
number_of_bands
same_sampling_rate
sampling_rate_hz

Methods

add_band(sig[, index])	Adds a new band to the *MultiBandSignal*.
collapse()	Collapses MultiBandSignal by summing all of its bands and returning one Signal (possibly multichannel).
copy()	Returns a copy of the object.
get_all_bands([channel])	Broadcasts and returns the *MultiBandSignal* as a *Signal* object with all bands as channels in the output.
remove_band([index, return_band])	Removes a band from the *MultiBandSignal*.
save_signal([path])	Saves the *MultiBandSignal* object as a pickle.
show_info([show_band_info])	Show information about the *MultiBandSignal*.
swap_bands(new_order)	Rearranges the bands in the new given order.

__init__(*bands: list = []*, *same_sampling_rate: bool = True*, *info: dict = {}*)

MultiBandSignal contains a composite band list where each index is a Signal object with the same number of channels. For multirate systems, the parameter *same_sampling_rate* has to be set to *False*.

Parameters

bands
[list, optional] List or tuple containing different Signal objects. All of them should be associated to the same Signal. This means that the channel numbers have to match. Set to *None* for initializing the object. Default: *None*.

same_sampling_rate
[bool, optional] When *True*, every Signal should have the same sampling rate. Set to *False* for a multirate system. Default: *True*.

info
[dict, optional] A dictionary with generic information about the *MultiBandSignal* can be passed. Default: *None*.

add_band(*sig: Signal*, *index: int = -1*)

Adds a new band to the *MultiBandSignal*.

Parameters

sig
[*Signal*] Signal to be added.

index
[int, optional] Index at which to insert the new Signal. Default: -1.

property bands: list

collapse() → *Signal*

Collapses MultiBandSignal by summing all of its bands and returning one Signal (possibly multichannel).

Returns

new_sig
[*Signal*] Collapsed Signal.

copy()

Returns a copy of the object.

Returns

new_sig
[*MultiBandSignal*] Copy of Signal.

get_all_bands(*channel: int = 0*) → *dsptoolbox.classes.signal_class.Signal* | list

Broadcasts and returns the *MultiBandSignal* as a *Signal* object with all bands as channels in the output. This is done only for a single channel of the original signal.

Parameters

channel
[int, optional] Channel to choose from the band signals.

Returns

sig
[*Signal* or list of *np.ndarray* and dict] Multichannel signal with all the bands. If the *MultiBandSignal* does not have the same sampling rate for all signals, a list with the time data vectors and a dictionary containing their sampling rates with the key *'sampling_rates'* are returned.

property number_of_bands: int

remove_band(*index: int = -1*, *return_band: bool = False*)

Removes a band from the *MultiBandSignal*.

Parameters

index
[int, optional] This is the index from the bands list at which the band will be erased. When -1, last band is erased. Default: -1.

return_band
[bool, optional] When *True*, the erased band is returned. Default: *False*.

property same_sampling_rate: bool

property sampling_rate_hz: int

save_signal(*path: str = 'multibandsignal'*)

Saves the *MultiBandSignal* object as a pickle.

Parameters

path
[str, optional] Path for the signal to be saved. Use only folder/folder/name (without format). Default: *'multibandsignal'* (local folder, object named multibandsignal).

show_info(*show_band_info: bool = False*)

Show information about the *MultiBandSignal*.

Parameters

show_band_info
[bool, optional] When *True*, a longer message is printed with all available information regarding each *Signal* in the *MultiBandSignal*. Default: *True*.

swap_bands(*new_order*)

Rearranges the bands in the new given order.

Parameters

new_order
[array-like] New rearrangement of bands.

2.3 Filter

Contains Filter classes

class dsptoolbox.classes.filter_class.**Filter**(*filter_type: str = 'biquad', filter_configuration: Optional[dict] = None, sampling_rate_hz: int = 48000*)

Bases: `object`

Class for creating and storing linear digital filters with all their metadata.

Attributes

sampling_rate_hz
warning_if_complex

Methods

`copy`()	Returns a copy of the object.
`filter_signal`(signal[, channels, ...])	Takes in a *Signal* object and filters selected channels.
`get_coefficients`([mode])	Returns the filter coefficients.
`get_filter_metadata`()	Returns filter metadata.
`get_ir`([length_samples])	Gets an impulse response of the filter with given length.
`initialize_zi`([number_of_channels])	Initializes zi for steady-state filtering.
`plot_group_delay`([length_samples, range_hz, ...])	Plots group delay of the filter.
`plot_magnitude`([length_samples, range_hz, ...])	Plots magnitude spectrum.
`plot_phase`([length_samples, range_hz, ...])	Plots phase spectrum.
`plot_zp`([show_info_box, returns])	Plots zeros and poles with the unit circle.
`save_filter`([path])	Saves the Filter object as a pickle.
`set_filter_parameters`(filter_type, ...)	
`show_info`()	Prints all the filter parameters to the console.

`__init__`(*filter_type: str = 'biquad', filter_configuration: Optional[dict] = None, sampling_rate_hz: int = 48000*)

The Filter class contains all parameters and metadata needed for using a digital filter.

Parameters

filter_type
[str, optional] String defining the filter type. Options are *'iir'*, *'fir'*, *'biquad'* or *'other'*. Default: creates a dummy biquad bell filter with no gain.

filter_configuration
[dict, optional] Dictionary containing configuration for the filter. Default: some dummy parameters.

sampling_rate_hz
[int, optional] Sampling rate in Hz for the digital filter. Default: 48000.

Notes

For *iir*:
Keys: order, freqs, type_of_pass, filter_design_method, filter_id (optional).

- order (int): Filter order
- freqs (float, array-like): array with len 2 when 'bandpass' or 'bandstop'.
- type_of_pass (str): 'bandpass', 'lowpass', 'highpass', 'bandstop'.
- filter_design_method (str): 'butter', 'bessel', 'ellip', 'cheby1', 'cheby2'.

For *fir*:
Keys: order, freqs, type_of_pass, filter_design_method (optional), width (optional, necessary for 'kaiser'), filter_id (optional).

- order (int): Filter order.
- freqs (float, array-like): array with len 2 when 'bandpass' or 'bandstop'.

- type_of_pass (str): ‘bandpass’, ‘lowpass’, ‘highpass’, ‘bandstop’.
- filter_design_method (str): Window to be used. Default: ‘hamming’. Supported types are: ‘boxcar’, ‘triang’, ‘blackman’, ‘hamming’, ‘hann’, ‘bartlett’, ‘flattop’, ‘parzen’, ‘bohman’, ‘blackmanharris’, ‘nuttall’, ‘barthann’, ‘cosine’, ‘exponential’, ‘tukey’, ‘taylor’.
- width (float): estimated width of transition region in Hz for kaiser window. Default: *None*.

For *biquad*:
Keys: eq_type, freqs, gain, q, filter_id (optional).

- eq_type (int or str): 0 = Peaking, 1 = Lowpass, 2 = Highpass, 3 = Bandpass skirt, 4 = Bandpass peak, 5 = Notch, 6 = Allpass, 7 = Lowshelf, 8 = Highshelf.
- freqs: float or array-like with length 2 (depending on eq_type).
- gain (float): in dB.
- q (float): Q-factor.

For *other* or *general*:
ba or sos or zpk, filter_id (optional).

Methods

General	set_filter_parameters, get_filter_metadata, get_ir.
Plots or prints	show_filter_parameters, plot_magnitude, plot_group_delay, plot_phase, plot_zp.
Filtering	filter_signal.

`copy()`

Returns a copy of the object.

Returns

new_sig
[*Filter*] Copy of filter.

`filter_signal`(*signal:* Signal, *channels=None*, *activate_zi:* bool = *False*, *zero_phase:* bool = *False*) → *Signal*

Takes in a *Signal* object and filters selected channels. Exports a new *Signal* object.

Parameters

signal
[*Signal*] Signal to be filtered.

channels
[int or array-like, optional] Channel or array of channels to be filtered. When *None*, all channels are filtered. Default: *None*.

activate_zi
[int, optional] Gives the zi to update the filter values. Default: *False*.

zero_phase
[bool, optional] Uses zero-phase filtering on signal. Be aware that the filter is applied twice in this case. Default: *False*.

Returns

new_signal
[*Signal*] New Signal object.

`get_coefficients`(*mode: str = 'sos'*)

Returns the filter coefficients.

Parameters

mode
[str, optional] Type of filter coefficients to be returned. Choose from *'sos'*, *'ba'* or *'zpk'*. Default: *'sos'*.

Returns

coefficients
[array-like] Array with filter parameters.

`get_filter_metadata`()

Returns filter metadata.

Returns

info
[dict] Dictionary containing all filter metadata.

`get_ir`(*length_samples: int = 512*)

Gets an impulse response of the filter with given length.

Parameters

length_samples
[int, optional] Length for the impulse response in samples. Default: 512.

Returns

ir_filt
[*Signal*] Impulse response of the filter.

`initialize_zi`(*number_of_channels: int = 1*)

Initializes zi for steady-state filtering. The number of parallel zi's can be defined externally.

Parameters

number_of_channels
[int, optional] Number of channels is needed for the number of filter's zi's. Default: 1.

`plot_group_delay`(*length_samples: int = 512*, *range_hz=[20, 20000.0]*, *show_info_box: bool = False*, *returns: bool = True*)

Plots group delay of the filter. Different methods are used for FIR or IIR filters.

Parameters

length_samples
[int, optional] Length of ir for magnitude plot. Default: 512.

range_hz
[array-like with length 2, optional] Range for which to plot the magnitude response. Default: [20, 20000].

show_info_box
[bool, optional] Shows an information box on the plot. Default: *False*.

returns
[bool, optional] When *True* figure and axis are returned. Default: *False*.

Returns

fig, ax
Returned only when *returns=True*.

`plot_magnitude`(*length_samples: int = 512*, *range_hz=[20, 20000.0]*, *normalize: Optional[str] = None*, *show_info_box: bool = True*, *returns: bool = False*)

Plots magnitude spectrum. Change parameters of spectrum with set_spectrum_parameters.

Parameters

length_samples
[int, optional] Length of ir for magnitude plot. Default: 512.

range_hz
[array-like with length 2, optional] Range for which to plot the magnitude response. Default: [20, 20000].

normalize
[str, optional] Mode for normalization, supported are *'1k'* for normalization with value at frequency 1 kHz or *'max'* for normalization with maximal value. Use *None* for no normalization. Default: *None*.

show_info_box
[bool, optional] Shows an information box on the plot. Default: *True*.

returns
[bool, optional] When *True* figure and axis are returned. Default: *False*.

Returns

fig, ax
Returned only when *returns=True*.

`plot_phase`(*length_samples: int = 512*, *range_hz=[20, 20000.0]*, *unwrap: bool = False*, *show_info_box: bool = False*, *returns: bool = False*)

Plots phase spectrum.

Parameters

length_samples
[int, optional] Length of ir for magnitude plot. Default: 512.

range_hz
[array-like with length 2, optional] Range for which to plot the magnitude response. Default: [20, 20000].

unwrap
[bool, optional] Unwraps the phase to show. Default: *False*.

show_info_box
[bool, optional] Shows an information box on the plot. Default: *False*.

returns
[bool, optional] When *True* figure and axis are returned. Default: *False*.

Returns

fig, ax
Returned only when *returns=True*.

`plot_zp`(*show_info_box: bool = False*, *returns: bool = False*)

Plots zeros and poles with the unit circle.

Parameters

returns
[bool, optional] When *True* figure and axis are returned. Default: *False*.

show_info_box
[bool, optional] Shows an information box on the plot. Default: *False*.

Returns

fig, ax
Returned only when *returns=True*.

`property sampling_rate_hz`

`save_filter`(*path: str = 'filter'*)

Saves the Filter object as a pickle.

Parameters

path
[str, optional] Path for the filter to be saved. Use only folder1/folder2/name (without format). Default: *'filter'* (local folder, object named filter).

`set_filter_parameters`(*filter_type: str*, *filter_configuration: dict*)

`show_info()`

Prints all the filter parameters to the console.

`property warning_if_complex`

2.4 Filterbank

`class dsptoolbox.classes.filterbank.FilterBank`(*filters: list = []*, *same_sampling_rate: bool = True*, *info: dict = {}*)

Bases: `object`

Standard template for filter banks containing filters, filters' initial values, metadata and some useful plotting methods.

Attributes

filters
same_sampling_rate
sampling_rate_hz

Methods

add_filter(filt[, index])	Adds a new filter at the end of the filters dictionary.
copy()	Returns a copy of the object.
filter_signal(signal[, mode, activate_zi, ...])	Applies the filter bank to a signal and returns a multi-band signal or a *Signal* object.
get_ir([mode, length_samples, test_zi, ...])	Returns impulse response from the filter bank.
initialize_zi([number_of_channels])	Initiates the zi of the filters for the given number of channels.
plot_group_delay([mode, range_hz, ...])	Plots the phase response of each filter.
plot_magnitude([mode, range_hz, ...])	Plots the magnitude response of each filter.
plot_phase([mode, range_hz, unwrap, ...])	Plots the phase response of each filter.
remove_filter([index, return_filter])	Removes a filter from the filter bank.
save_filterbank([path])	Saves the FilterBank object as a pickle.
show_info([show_filter_info])	Show information about the filter bank.
swap_filters(new_order)	Rearranges the filters in the new given order.

__init__(*filters: list = [], same_sampling_rate: bool = True, info: dict = {}*)

FilterBank object saves multiple filters and some metadata. It also allows for easy filtering with multiple filters. Since the digital filters that are supported are linear systems, the order in which they are saved and applied to a signal is not relevant.

Parameters

filters
[list, optional] List containing filters.

same_sampling_rate
[bool, optional] When *True*, every Filter should have the same sampling rate. Set to *False* for a multirate system. Default: *True*.

info
[dict, optional] Dictionary containing general information about the filter bank. Some parameters of the filter bank are automatically read from the filters dictionary.

Methods

General	add_filter, remove_filter, save_filterbank.
Prints and plots	plot_magnitude, plot_phase, plot_group_delay, show_info.

add_filter(*filt: Filter, index: int = -1*)

Adds a new filter at the end of the filters dictionary.

Parameters

filt
[*Filter*] Filter to be added to the FilterBank.

index
[int, optional] Index at which to insert the new Filter. Default: -1.

copy()

Returns a copy of the object.

Returns

new_sig
[*FilterBank*] Copy of filter bank.

filter_signal(*signal: Signal*, *mode: str = 'parallel'*, *activate_zi: bool = False*, *zero_phase: bool = False*) → *dsptoolbox.classes.signal_class.Signal* | *dsptoolbox.classes.multibandsignal.MultiBandSignal*

Applies the filter bank to a signal and returns a multiband signal or a *Signal* object. *'parallel'*: returns a *MultiBandSignal* object where each band is the output of each filter. *'sequential'*: applies each filter to the given Signal in a sequential manner and returns output with same dimension. *'summed'*: applies every filter as parallel and then summs the outputs returning same dimensional output as input.

Parameters

signal
[*Signal*] Signal to be filtered.

mode
[str, optional] Way to apply filter bank to the signal. Supported modes are: *'parallel'*, *'sequential'*, *'summed'*. Default: *'parallel'*.

activate_zi
[bool, optional] Takes in the filter initial values and updates them for streaming purposes. Default: *False*.

zero_phase
[bool, optional] Activates zero_phase filtering for the filter bank. It cannot be used at the same time with *zi=True*. Default: *False*.

Returns

new_sig
[*'sequential'* or *'summed'* -> *Signal*.]

'parallel' -> *MultiBandSignal*.

New signal after filtering.

property filters: list

get_ir(*mode: str = 'parallel'*, *length_samples: int = 2048*, *test_zi: bool = False*, *zero_phase: bool = False*) → *dsptoolbox.classes.signal_class.Signal* | *dsptoolbox.classes.multibandsignal.MultiBandSignal*

Returns impulse response from the filter bank.

Parameters

mode
[str, optional] Filtering mode. Choose from *'parallel'*, *'sequential'* or *'summed'*. Default: *'parallel'*.

length_samples
[int, optional] Length of the impulse response to be generated. If some filter is longer than the given length, then the length is adapted. Default: 2048.

test_zi
[bool, optional] When *True*, filtering is done while updating filters' initial values. Default: *False*.

zero_phase
[bool, optional] When *True*, zero phase filtering is activated. Default: *False*.

Returns

ir
[*MultiBandSignal* or *Signal*] Impulse response of the filter bank.

`initialize_zi`(*number_of_channels: int = 1*)

Initiates the zi of the filters for the given number of channels.

Parameters

number_of_channels
[int, optional] Number of channels is needed for the number of filters' zi's. Default: 1.

`plot_group_delay`(*mode: str = 'parallel'*, *range_hz=[20, 20000.0]*, *length_samples: int = 2048*, *test_zi: bool = False*, *returns: bool = False*)

Plots the phase response of each filter.

Parameters

mode
[str, optional] Type of plot. *'parallel'* plots every filter's frequency response, *'sequential'* plots the frequency response after having filtered one impulse with every filter in the FilterBank. *'summed'* sums up every filter output. Default: *'parallel'*.

range_hz
[array-like, optional] Range of Hz to plot. Default: [20, 20e3].

length_samples
[int, optional] Length (in samples) of the IR to be generated for the plot. Default: 2048.

test_zi
[bool, optional] Uses the zi's of each filter to test the FilterBank's output. Default: *False*.

returns
[bool, optional] When *True*, the figure and axis are returned. Default: *False*.

Returns

fig, ax
Returned only when *returns=True*.

`plot_magnitude`(*mode: str = 'parallel'*, *range_hz=[20, 20000.0]*, *length_samples: int = 2048*, *test_zi: bool = False*, *returns: bool = False*)

Plots the magnitude response of each filter.

Parameters

mode
[str, optional] Type of plot. *'parallel'* plots every filter's frequency response, *'sequential'* plots the frequency response after having filtered one impulse with every filter in the FilterBank. *'summed'* sums up every frequency response. Default: *'parallel'*.

range_hz
[array-like, optional] Range of Hz to plot. Default: [20, 20e3].

length_samples
[int, optional] Length (in samples) of the IR to be generated for the plot. Default: 2048.

test_zi
[bool, optional] Uses the zi's of each filter to test the FilterBank's output. Default: *False*.

returns
[bool, optional] When *True*, the figure and axis are returned. Default: *False*.

Returns

fig, ax
Returned only when *returns=True*.

`plot_phase`(*mode: str = 'parallel', range_hz=[20, 20000.0], unwrap: bool = False, length_samples: int = 2048, test_zi: bool = False, returns: bool = False*)

Plots the phase response of each filter.

Parameters

mode
[str, optional] Type of plot. *'parallel'* plots every filter's frequency response, *'sequential'* plots the frequency response after having filtered one impulse with every filter in the FilterBank. *'summed'* sums up every filter output. Default: *'parallel'*.

range_hz
[array-like, optional] Range of Hz to plot. Default: [20, 20e3].

unwrap
[bool, optional] When *True*, unwrapped phase is plotted. Default: *False*.

length_samples
[int, optional] Length (in samples) of the IR to be generated for the plot. Default: 2048.

test_zi
[bool, optional] Uses the zi's of each filter to test the FilterBank's output. Default: *False*.

returns
[bool, optional] When *True*, the figure and axis are returned. Default: *False*.

Returns

fig, ax
Returned only when *returns=True*.

`remove_filter`(*index: int = -1, return_filter: bool = False*)

Removes a filter from the filter bank.

Parameters

index
[int, optional] This is the index from the filters list at which the filter will be erased. When -1, last filter is erased. Default: -1.

return_filter
[bool, optional] When *True*, the erased filter is returned. Default: *False*.

`property same_sampling_rate: bool`

`property sampling_rate_hz: int`

`save_filterbank`(*path: str = 'filterbank'*)

Saves the FilterBank object as a pickle.

Parameters

path
[str, optional] Path for the filterbank to be saved. Use only folder1/folder2/name (without format). Default: *'filterbank'* (local folder, object named filterbank).

`show_info`(*show_filter_info: bool = True*)

Show information about the filter bank.

Parameters

show_filters_info
[bool, optional] When *True*, a longer message is printed with all available information regarding each filter in the filter bank. Default: *True*.

`swap_filters`(*new_order*)

Rearranges the filters in the new given order.

Parameters

new_order
[array-like] New rearrangement of filters.

CHAPTER THREE

MODULES IN DSPTOOLBOX

The modules and functions of dsptoolbox are listed down below.

3.1 Distances (dsptoolbox.distances)

3.1.1 Distances

This module contains distance measurements between signals. Even though the results seem plausible, these implementations need validation from external tools.

Frequency domain:

- *log_spectral()*
- *itakura_saito()*

Time domain:

- *snr()*
- *si_sdr()*

Mixed:

- *fw_snr_seg()*

`dsptoolbox.distances.`**`fw_snr_seg`**(*x:* Signal, *xhat:* Signal, *freq_range_hz=[20, 10000.0]*, *snr_range_db=[-10, 35]*, *gamma:* float *= 0.2*)

Frequency-weighted segmental SNR (fwSNRseg) computation between two signals.

This distance measure divides the signal into auditory frequency bands (using the auditory gammatone filters) and splits the signal in time frames to further compute SNR. This distance was shown to correlate relatively well with results from listening tests and other subjective measurements. See references for more information.

NOTE: the time window is fixed to be a 75 ms Hamming window with 50% overlap instead as gaussian window (as in the paper) since no length and beta parameter were specified in the publication.

Parameters

x
[*Signal*] Original clean signal. If this signal only contains one channel and *xhat* more, it is assumed that this channel is the original of all the others.

xhat
[*Signal*] Enhanced/modified signal.

freq_range_hz
[array-like with length of 2, optional] Frequency range in which to analyze the signals. Default: [20, 10e3].

snr_range_db
[array-like with length of 2, optional] SNR range to be regarded. If any frame throws a value outside this range, it is set to the boundary. Default: [-10, 35].

gamma
[float, optional] Gamma parameter to be used for the frame weightning. See paper for more information about it. Its recommended range is (according to reference) constrained to [0.1, 2]. Default: 0.2.

Returns

snr_per_channel
[*np.ndarray*] Frequency-weighted, time-segmented SNR per channel.

References

- Y. Hu and P. C. Loizou, "Evaluation of Objective Quality Measures for Speech Enhancement," in IEEE Transactions on Audio, Speech, and Language Processing, vol. 16, no. 1, pp. 229-238, Jan. 2008, doi: 10.1109/TASL.2007.911054.
- https://ieeexplore.ieee.org/document/4389058

`dsptoolbox.distances.`**`itakura_saito`**`(`*insig1:* Signal, *insig2:* Signal, *method:* *str* = *'welch'*, *f_range_hz=[20, 20000]*, ***kwargs*`)`

Computes itakura-saito measure between two signals. Beware that this measure is not symmetric (x, y) != (y, x).

Parameters

insig1
[Signal] Signal 1.

insig2
[Signal] Signal 2.

f_range_hz
[array, optional] Range of frequencies in which to compute the distance. When *None*, it is computed in all frequencies. Default: [20, 20000].

Returns

distances
[*np.ndarray*] Itakura-saito measure for the given signals.

References

- https://en.wikipedia.org/wiki/Itakura–Saito_distance

`dsptoolbox.distances.`**`log_spectral`**`(`*insig1:* Signal, *insig2:* Signal, *method:* *str* = *'welch'*, *f_range_hz=[20, 20000]*, ***kwargs*`)`

Computes log spectral distance between two signals.

Parameters

insig1
[Signal] Signal 1.

insig2
[Signal] Signal 2.

f_range_hz
[array, optional] Range of frequencies in which to compute the distance. When *None*, it is computed in all frequencies. Default: [20, 20000].

Returns

distances
[*np.ndarray*] Log spectral distance per channel for the given signals.

References

- https://en.wikipedia.org/wiki/Log-spectral_distance

`dsptoolbox.distances.si_sdr`(*target_signal:* Signal, *modified_signal:* Signal)

Computes scale-invariant signal to distortion ratio from an original and a modified signal. If target signal only has one channel, it is assumed to be the target for all the channels in the modified signal. See reference for details.

Parameters

target_signal
[*Signal*] Original signal.

modified_signal
[*Signal*] Signal after modification/enhancement.

Returns

sdr
[*np.ndarray*] SI-SDR per channel.

References

- https://arxiv.org/abs/1811.02508

`dsptoolbox.distances.snr`(*signal:* Signal, *noise:* Signal)

Classical Signal-to-noise ratio. If noise only has one channel, it is assumed to be the noise for all channels of signal.

Parameters

signal
[*Signal*] Signal.

noise
[*Signal*] Noise.

Returns

snr_per_channel
[*np.ndarray*] SNR value per channel

References

- https://en.wikipedia.org/wiki/Signal-to-noise_ratio

3.2 Filterbanks (dsptoolbox.filterbanks)

3.2.1 Filter Banks

This is a collection of useful filter banks. They use primarily the *FilterBank* class or some derivation from it.

Perfect magnitude reconstruction:

- *linkwitz_riley_crossovers()*

Perfect reconstruction:

- *reconstructing_fractional_octave_bands()*

Psychoacoustics:

- *auditory_filters_gammatone()*

`dsptoolbox.filterbanks.`**`auditory_filters_gammatone`**(*freq_range_hz=[20, 20000]*, *resolution: float = 1*, *sampling_rate_hz: int = 48000*) → *FilterBank*

Generate an auditory filter bank for analysis purposes. This code was taken and adapted from the pyfar package. In this implementation, the reference frequency is fixed to 1000 Hz and there is no support for reconstructing the original signal.

For a more general implementation of this filter bank (reconstruction capabilities) please refer to the pyfar package or the octave/matlab auditory modelling toolbox. See references.

Parameters

freq_range
[array-like] The upper and lower frequency in Hz between which the filter bank is constructed. Values must be larger than 0 and not exceed half the sampling rate.

resolution
[number] The frequency resolution of the filter bands in equivalent rectangular bandwidth (ERB) units. The bands of the filter bank are distributed linearly on the ERB scale. The default value of 1 results in one filter band per ERB. A value of `0.5` would result in two filter bands per ERB.

sampling_rate_hz
[int, optional] The sampling rate of the filter bank in Hz. Default: 48000.

Returns

gammatone_fb
[FilterBank] Gammatone filter bank.

`dsptoolbox.filterbanks.`**`linkwitz_riley_crossovers`**(*freqs*, *order*, *sampling_rate_hz: int = 48000*)

Returns a linkwitz-riley crossovers filter bank.

Parameters

freqs
[array-like] Frequencies at which to set the crossovers.

order
[array-like] Order of the crossovers. The higher, the steeper.

sampling_rate_hz
[int, optional] Sampling rate for the filterbank. Default: 48000.

Returns

LRFilterBank
Filter bank in form of LRFilterBank class which contains the same methods as the FilterBank class but is generated with different internal methods.

`dsptoolbox.filterbanks.`**`reconstructing_fractional_octave_bands`**`(`*num_fractions: int = 1, frequency_range=[63, 16000], overlap: float = 1, slope: int = 0, n_samples: int = 4096, sampling_rate_hz: int = 48000*`)` → *FilterBank*

Create and/or apply an amplitude preserving fractional octave filter bank. This implementation is taken directly from the pyfar package. See references for more information about it.

Parameters

num_fractions
[int, optional] Octave fraction, e.g., `3` for third-octave bands. The default is `1`.

frequency_range
[tuple, optional] Frequency range for fractional octave in Hz. The default is `(63, 16000)`

overlap
[float] Band overlap of the filter slopes between 0 and 1. Smaller values yield wider pass-bands and steeper filter slopes. The default is `1`.

slope
[int, optional] Number > 0 that defines the width and steepness of the filter slopes. Larger values yield wider pass-bands and steeper filter slopes. The default is `0`.

n_samples
[int, optional] Length of the filter in samples. Longer filters yield more exact filters. The default is `2**12`.

sampling_rate
[int] Sampling frequency in Hz. The default is `None`. Only required if `signal=None`.

Returns

filt_bank
[*FilterBank*] Filter Bank object with FIR filters.

References

- https://pubmed.ncbi.nlm.nih.gov/20136211/
- https://github.com/pyfar/pyfar

3.3 Generators (dsptoolbox.generators)

3.3.1 Generators

This module contains some utility signal generators. Choose from:

- *chirp()* (sweep)
- *noise()* (white, pink, red, blue, violet, grey)
- *dirac()* (impulse)
- *sinus()* (harmonic)

`dsptoolbox.generators.`**`chirp`**(*type_of_chirp: str = 'log', range_hz=None, length_seconds: float = 1, sampling_rate_hz: int = 48000, peak_level_dbfs: float = -10, number_of_channels: int = 1, fade: str = 'log', padding_end_seconds: Optional[float] = None*) → *Signal*

Creates a sweep signal.

Parameters

type_of_chirp
[str, optional] Choose from *'lin'*, *'log'*. Default: *'log'*.

range_hz
[array-like with length 2] Define range of chirp in Hz. When *None*, all frequencies between 1 and nyquist are taken. Default: *None*.

length_seconds
[float, optional] Length of the generated signal in seconds. Default: 1.

sampling_rate_hz
[int, optional] Sampling rate in Hz. Default: 48000.

peak_level_dbfs
[float, optional] Peak level of the signal in dBFS. Default: -10.

number_of_channels
[int, optional] Number of channels (with the same chirp) to be created. Default: 1.

fade
[str, optional] Type of fade done on the generated signal. By default, 10% of signal length (without the padding in the end) is faded at the beginning and end. Options are *'exp'*, *'lin'*, *'log'*. Pass *None* for no fading. Default: *'log'*.

padding_end_seconds
[float, optional] Padding at the end of signal. Use *None* to avoid any padding. Default: *None*.

Returns

chirp_sig
[*Signal*] Chirp Signal object.

References

https://de.wikipedia.org/wiki/Chirp

`dsptoolbox.generators.dirac`(*length_samples: int = 512*, *number_of_channels: int = 1*, *sampling_rate_hz: int = 48000*) → *Signal*

Generates a dirac impulse Signal with the specified length and sampling rate.

Parameters

length_samples
[int, optional] Length in samples. Default: 512.

number_of_channels
[int, optional] Number of channels to be generated with the same impulse. Default: 1.

sampling_rate_hz
[int, optional] Sampling rate to be used. Default: 480000.

Returns

imp
[*Signal*] Signal with dirac impulse.

`dsptoolbox.generators.noise`(*type_of_noise: str = 'white'*, *length_seconds: float = 1*, *sampling_rate_hz: int = 48000*, *peak_level_dbfs: float = -10*, *number_of_channels: int = 1*, *fade: str = 'log'*, *padding_end_seconds: Optional[float] = None*) → *Signal*

Creates a noise signal.

Parameters

type_of_noise
[str, optional] Choose from *'white'*, *'pink'*, *'red'*, *'blue'*, *'violet'* or *'grey'*. Default: *'white'*.

length_seconds
[float, optional] Length of the generated signal in seconds. Default: 1.

sampling_rate_hz
[int, optional] Sampling rate in Hz. Default: 48000.

peak_level_dbfs
[float, optional] Peak level of the signal in dBFS. Default: -10.

number_of_channels
[int, optional] Number of channels (with different noise signals) to be created. Default: 1.

fade
[str, optional] Type of fade done on the generated signal. By default, 10% of signal length (without the padding in the end) is faded at the beginning and end. Options are *'exp'*, *'lin'*, *'log'*. Pass *None* for no fading. Default: *'log'*.

padding_end_seconds
[float, optional] Padding at the end of signal. Use *None* to avoid any padding. Default: *None*.

Returns

noise_sig
[*Signal*] Noise Signal object.

References

https://en.wikipedia.org/wiki/Colors_of_noise

`dsptoolbox.generators.`**`sinus`**(*frequency_hz: float = 1000*, *length_seconds: float = 1*, *sampling_rate_hz: int = 48000*, *peak_level_dbfs: float = -10*, *number_of_channels: int = 1*, *uncorrelated: bool = False*, *fade: str = 'log'*, *padding_end_seconds: Optional[float] = None*) → *Signal*

Creates a multi-channel sinus tone.

Parameters

frequency_hz
[float, optional] Frequency (in Hz) for the sinus tone. Default: 1000.

length_seconds
[float, optional] Length of the generated signal in seconds. Default: 1.

sampling_rate_hz
[int, optional] Sampling rate in Hz. Default: 48000.

peak_level_dbfs
[float, optional] Peak level of the signal in dBFS. Default: -10.

number_of_channels
[int, optional] Number of channels (with the same chirp) to be created. Default: 1.

uncorrelated
[bool, optional] When *True*, each tone gets a random phase shift so that the signals are not perfectly correlated. Default: *False*.

fade
[str, optional] Type of fade done on the generated signal. By default, 10% of signal length (without the padding in the end) is faded at the beginning and end. Options are *'exp'*, *'lin'*, *'log'*. Pass *None* for no fading. Default: *'log'*.

padding_end_seconds
[float, optional] Padding at the end of signal. Use *None* to avoid any padding. Default: *None*.

Returns

sinus_sig
[*Signal*] Sinus tone signal.

3.4 Audio IO (dsptoolbox.audio_io)

3.4.1 Audio IO

This module handles audio playback and recording. It is based on sounddevice (see link down below). While some functions offer full functionality, some are just wrappers around sounddevice's functions:

Setting audio device:

- *set_device()*
- *print_device_info()*

Playing audio:

- *play()*

- *play_through_stream()*
- *play_and_record()*
- *output_stream()*

Recording:

- *record()*

Others:

- *CallbackStop()* (used for stopping callbacks)
- *standard_callback()* (an example of an audio callback)
- *sleep()* (sleep while audio playback is finished)

3.4.2 References

- https://pypi.org/project/sounddevice/

`dsptoolbox.audio_io.`**`CallbackStop`**`()`

Wrapper around sounddevice's CallbackStop. Used for stopping audio streamings.

`dsptoolbox.audio_io.`**`output_stream`**`(`*signal: Signal, blocksize=2048, device=None, latency=None, extra_settings=None, callback=None, finished_callback=None, clip_off=None, dither_off=None, never_drop_input=None, prime_output_buffers_using_stream_callback=None*`)`

Creates and return a sounddevice's OutputStream object. See sounddevice's documentation for more information.

Parameters

signal
[*Signal*] Signal for which the output stream will be created.

blocksize
[int, optional] Block size to be used during the stream. Default: 2048.

device
[str, optional] Device to be used. Pass *None* to use default device. Default: *None*.

callback
[callable] Function that defines the audio callback:

```
callback(outdata: np.ndarray, frames: int,
         time: CData, status: CallbackFlags) -> None
```

finished_callback
[callable]

clip_off
dither_off
never_drop_input
prime_output_buffers_using_stream_callback

Returns

stream
[*sd.OutputStream*] Stream object.

References

- https://python-sounddevice.readthedocs.io/en/0.4.5/

`dsptoolbox.audio_io.play`(*signal: Signal*, *duration_seconds: Optional[float] = None*, *normalized_dbfs: float = -6*, *device: Optional[str] = None*, *play_channels=None*)

Play some available device. Note that the channel numbers start here with 1.

Parameters

signal
[Signal] Signal to be reproduced. Its channel number must match the the length of the play_channels vector.

duration_seconds
[float, optional] If *None*, the whole signal is played, otherwise it is trimmed to the given length. Default: *None*.

normalized_dbfs: float, optional
Normalizes the signal (dBFS peak level) before playing it. Set to *None* to ignore normalization. Default: -6.

device
[str, optional] I/O device to be used. If *None*, the default device is used. Default: *None*.

play_channels
[int or array-like, optional] Output channels that will play the signal. The number of channels should match the number of channels in signal. When *None*, the channels are automatically set. Default: *None*.

`dsptoolbox.audio_io.play_and_record`(*signal: Signal*, *duration_seconds: Optional[float] = None*, *normalized_dbfs: float = -6*, *device: Optional[str] = None*, *play_channels=None*, *rec_channels=[1]*) → *Signal*

Play and record using some available device. Note that the channel numbers start here with 1.

Parameters

signal
[Signal] Signal object to be played. The number of channels has to match the total length and order of play_channels. The sampling rate of signal will define the sampling rate of the recorded signals.

duration_seconds
[float, optional] If *None*, the whole signal is played, otherwise it is trimmed to the given length. Default: *None*.

normalized_dbfs: float, optional
Normalizes the signal (dBFS peak level) before playing it. Set to *None* to ignore normalization. Default: -6.

device
[str, optional] I/O device to be used. If *None*, the default device is used. Default: *None*.

play_channels
[int or array-like, optional] Output channels that will play the signal. The number of channels should match the number of channels in signal. When *None*, the channels are automatically set. Default: *None*.

rec_channels
[int or array-like, optional] Channel numbers that will be recorded. Default: [1].

Returns

rec_sig
[*Signal*] Recorded signal.

`dsptoolbox.audio_io.`**`play_through_stream`**`(`*signal: ~dsptoolbox.classes.signal_class.Signal*, *blocksize: int = 2048*, *audio_callback=<function standard_callback>*, *device: ~typing.Optional[str] = None*`)`

Plays a signal using a stream and a callback function. See *sounddevice.OutputStream* for extensive information about functionalities.

NOTE: The *Signal*'s channel shape must match the device's output channels. The start of the signal can be set using *Signal*'s method *set_streaming_position*.

Parameters

signal
[*Signal*] Signal to be reproduced.

blocksize
[int, optional] Block size used for the stream. Powers of two are recommended. Default: 2048.

audio_callback
[callable, optional] This is the core of the stream! Callback modifies the signal as it is passed to the audio device. The *standard_callback* passes it through with a stage of preprocessing (normalizing and fade). *audio_callback* is a function that takes in a signal object and passes a valid sounddevice callback. There can be preprocessing but it shouldn't change the total length of the signal! Their signatures are:

```
audio_callback(signal: Signal) -> callback

callback(outdata: np.ndarray, frames: int,
         time: CData, status: CallbackFlags) -> None
```

See *sounddevice*'s examples of callbacks for more general information. Default: *standard_callback*.

device
[str, optional] Device to be used in the audio stream. Pass *None* to use the default device. Default: *None*.

References

- https://python-sounddevice.readthedocs.io/en/0.4.5/

`dsptoolbox.audio_io.`**`print_device_info`**`(`*device_number: Optional[int] = None*`)`

Prints available audio devices or information about a certain device when the device number is given.

Parameters

device_number
[int, optional] Prints information about the specific device and returns it as a dictionary. Use *None* to only print information about all devices without returning anything. Default: *None*.

Returns

d
[dict] Only when *device_number is not None*.

`dsptoolbox.audio_io.record`(*duration_seconds: float = 5*, *sampling_rate_hz: int = 48000*, *device: Optional[str] = None*, *rec_channels=[1]*) → *Signal*

Record using some available device. Note that the channel numbers start here with 1.

Parameters

duration_seconds
[float, optional] Duration of recording in seconds. Default: 5.

sampling_rate_hz
[int, optional] Sampling rate used for recording. Default: 48000.

device
[str, optional] I/O device to be used. If *None*, the default device is used. Default: *None*.

rec_channels
[int or array-like, optional] Number that will be recorded. Default: [1].

Returns

rec_sig
[*Signal*] Recorded signal.

`dsptoolbox.audio_io.set_device`(*device_number: Optional[int] = None*)

Takes in a device number to set it as the default. If *None* is passed, the available devices are first shown and then the user is asked for input to set the device.

Parameters

device_number
[int, optional] Sets the device as default. Use *None* to be prompted with the options. Default: *None*.

`dsptoolbox.audio_io.sleep`(*seconds: float*)

Wrapper around sounddevice's sleep. Use for waiting while a stream happens.

Parameters

seconds
[float] Seconds to wait.

`dsptoolbox.audio_io.standard_callback`(*signal:* Signal)

This standard callback function takes in a signal and does preprocessing (normalization and fade). After that, it returns a callback valid for the *sounddevice.OutputStream*.

Parameters

signal
[*Signal*] Signal to be played through the audio stream.

Returns

call
[callable] Function to be used as callback for the output stream. The signature must be valid for sounddevice's callback:

```
call(outdata: np.ndarray, frames: int, time, status) -> None
```

3.5 Plots (dsptoolbox.plots)

3.5.1 Plots

This module contains plotting templates that use matplotlib.

- *general_matrix_plot()*
- *general_plot()*
- *general_subplots_line()*
- *show()*

`dsptoolbox.plots.general_matrix_plot`(*matrix*, *range_x=None*, *range_y=None*, *range_z=None*, *xlabel: Optional[str] = None*, *ylabel: Optional[str] = None*, *zlabel: Optional[str] = None*, *xlog: bool = False*, *ylog: bool = False*, *colorbar: bool = True*, *cmap: str = 'magma'*, *returns: bool = False*)

Generic plot template for a matrix's heatmap.

Parameters

matrix
[*np.ndarray*] Matrix with data to plot.

range_x
[array-like, optional] Range to show for x axis. Default: None.

range_y
[array-like, optional] Range to show for y axis. Default: None.

xlabel
[str, optional] Label for x axis. Default: None.

ylabel
[str, optional] Label for y axis. Default: None.

zlabel
[str, optional] Label for z axis. Default: None.

xlog
[bool, optional] Show x axis as logarithmic. Default: *False*.

ylog
[bool, optional] Show y axis as logarithmic. Default: *False*.

colorbar
[bool, optional] When *True*, a colorbar for zaxis is shown. Default: *True*.

cmap
[str, optional] Type of colormap to use from matplotlib. See https://matplotlib.org/stable/tutorials/colors/colormaps.html. Default: *'magma'*.

returns
[bool, optional] When *True*, the figure and axis are returned. Default: *False*.

Returns

fig, ax
Returned only when *returns=True*.

`dsptoolbox.plots.`**`general_plot`**(*x*, *matrix*, *range_x=None*, *range_y=None*, *log: bool = True*, *labels=None*, *xlabel: str = 'Frequency / Hz'*, *ylabel: Optional[str] = None*, *info_box: Optional[str] = None*, *tight_layout: bool = True*, *returns: bool = False*)

Generic plot template.

Parameters

x
[array-like] Vector for x axis.

matrix
[*np.ndarray*] Matrix with data to plot.

range_x
[array-like, optional] Range to show for x axis. Default: None.

range_y
[array-like, optional] Range to show for y axis. Default: None.

log
[bool, optional] Show x axis as logarithmic. Default: *True*.

xlabel
[str, optional] Label for x axis. Default: None.

ylabel
[str, optional] Label for y axis. Default: None.

info_box
[str, optional] String containing extra information to be shown in a info box on the plot. Default: None.

tight_layout: bool, optional
When *True*, tight layout is activated. Default: *True*.

returns
[bool, optional] When *True*, the figure and axis are returned. Default: *False*.

Returns

fig, ax
Returned only when *returns=True*.

`dsptoolbox.plots.`**`general_subplots_line`**(*x*, *matrix*, *column: bool = True*, *sharex: bool = True*, *sharey: bool = False*, *log: bool = False*, *xlabels=None*, *ylabels=None*, *range_x=None*, *range_y=None*, *returns: bool = False*)

Generic plot template with subplots in one column or row.

Parameters

x
[array-like] Vector for x axis.

matrix
[*np.ndarray*] Matrix with data to plot.

column
[bool, optional] When *True*, the subplots are organized in one column. Default: *True*.

sharex
[bool, optional] When *True*, all subplots share the same values for the x axis. Default: *True*.

sharey
[bool, optional] When *True*, all subplots share the same values for the y axis. Default: *False*.

log
[bool, optional] Show x axis as logarithmic. Default: *False*.

xlabels
[array_like, optional] Labels for x axis. Default: None.

ylabels
[array_like, optional] Labels for y axis. Default: None.

range_x
[array-like, optional] Range to show for x axis. Default: None.

range_y
[array-like, optional] Range to show for y axis. Default: None.

returns
[bool, optional] When *True*, the figure and axis are returned. Default: *False*.

Returns

fig, ax
Returned only when *returns=True*.

`dsptoolbox.plots.show()`

Wrapper around matplotlib's show.

3.6 Room acoustics (dsptoolbox.room_acoustics)

3.6.1 Room Acoustics

Here are contained some functions that are related to room acoustics.

- *reverb_time()*
- *find_modes()*
- *convolve_rir_on_signal()*
- *find_ir_start()*

`dsptoolbox.room_acoustics.convolve_rir_on_signal`(*signal: Signal*, *rir: Signal*, *keep_peak_level: bool* = *True*, *keep_length: bool* = *True*) → *Signal*

Applies an RIR to a given signal. The RIR should also be a signal object with a single channel containing the RIR time data. Signal type should also be set to IR or RIR. By default, all channels are convolved with the RIR.

Parameters

signal
[Signal] Signal to which the RIR is applied. All channels are affected.

rir
[Signal] Single-channel Signal object containing the RIR.

keep_peak_level
[bool, optional] When *True*, output signal is normalized to the peak level of the original signal. Default: *True*.

keep_length
[bool, optional] When *True*, the original length is kept after convolution, otherwise the output signal is longer than the input one. Default: *True*.

Returns

new_sig
[*Signal*] Convolved signal with RIR.

`dsptoolbox.room_acoustics.`**`find_ir_start`**(*signal: Signal*, *threshold_dbfs: float = -20*)

This function finds the start of an IR defined as the first sample where a certain threshold is surpassed.

Parameters

signal
[*Signal*] IR signal.

threshold_dbfs
[float, optional] Threshold that should be passed (in dBFS). Default: -20.

Returns

start_index
[int or *np.ndarray*] Index of IR start for each channel. Returns an integer when signal only has one channel

References

- ISO 3382-1:2009-10, Acoustics - Measurement of the reverberation time of rooms with reference to other acoustical parameters. pp. 22.

`dsptoolbox.room_acoustics.`**`find_modes`**(*signal: Signal*, *f_range_hz=[50, 200]*, *proximity_effect: bool = False*, *dist_hz: float = 5*)

This metod is NOT validated. It might not be sufficient to find all modes in the given range.

Computes the room modes of a set of RIR using different criteria: Complex mode indication function, sum of magnitude responses and group delay peaks of the first RIR.

Parameters

signal
[*Signal*] Signal containing the RIR'S from which to find the modes.

f_range_hz
[array-like, optional] Vector setting range for mode search. Default: [50, 200].

proximity_effect
[bool, optional] When *True*, only group delay criteria is used for finding modes up until 200 Hz. This is done since a gradient transducer will not easily see peaks in its magnitude response in low frequencies due to near-field effects. Default: *False*.

dist_hz
[float, optional] Minimum distance (in Hz) between modes. Default: 5.

Returns

f_modes
[*np.ndarray*] Vector containing frequencies where modes have been localized.

References

- http://papers.vibetech.com/Paper17-CMIF.pdf

`dsptoolbox.room_acoustics.`**`reverb_time`**(*signal: dsptoolbox.classes.signal_class.Signal | dsptoolbox.classes.multibandsignal.MultiBandSignal*, *mode: str = 'T20'*, *ir_start: Optional[int] = None*, *return_ir_start: bool = False*)

Computes reverberation time. T20, T30, T60 and EDT.

Parameters

signal
[*Signal* or *MultiBandSignal*] Signal for which to compute reverberation times. It must be type *'ir'* or *'rir'*.

mode
[str, optional] Reverberation time mode. Options are *'T20'*, *'T30'*, *'T60'* or *'EDT'*. Default: *'T20'*.

ir_start
[int, optional] When not *None*, it is used as the index of the start of the impulse response. Otherwise it is automatically computed. Default: *None*.

return_ir_start
[bool, optional] When *True*, not only reverberation times are returned but also the index of the start of the impulse response (for the first channel only). It should not be set to *True* when *ir_start is not None*. Default: *False*.

Returns

reverberation_times
[*np.ndarray*] Reverberation times for each channel. Shape is (band, channel) if MultiBandSignal object is passed.

References

- DIN 3382
- ISO 3382-1:2009-10, Acoustics - Measurement of the reverberation time of

rooms with reference to other acoustical parameters. pp. 22.

3.7 Special (dsptoolbox.special)

3.7.1 Special

This module contains functions that might be regarded as not very common use.

- *cepstrum()*
- *min_phase_from_mag()* (generate a signal with minimum phase from a magnitude spectrum)
- *lin_phase_from_mag()* (generate a signal with linear phase from a magnitude spectrum)

`dsptoolbox.special.`**`cepstrum`**(*signal:* Signal, *mode='power'*)

Returns the cepstrum of a given signal in the Quefrency domain.

Parameters

signal
[Signal] Signal to compute the cepstrum from.

mode
[str, optional] Type of cepstrum. Supported modes are *'power'*, *'real'* and *'complex'*. Default: *'power'*.

Returns

ceps
[*np.ndarray*] Cepstrum.

References

https://de.wikipedia.org/wiki/Cepstrum

`dsptoolbox.special.`**`lin_phase_from_mag`**(*spectrum: ndarray*, *sampling_rate_hz:* int, *group_delay_ms='minimal'*, *check_causality:* bool *= True*, *signal_type:* str *= 'ir'*)

Returns a linear phase signal from a magnitude spectrum. It is possible to return the smallest causal group delay by checking the minimal phase version of the signal and choosing a constant group delay that is never lower than minimum group delay (for each channel). A value for the group delay can be also passed directly and applied to all channels. If check causility is activated, it is assessed that the given group delay is not less than each minimal group delay. If deactivated, the generated phase could lead to a non-causal system!

Parameters

spectrum
[*np.ndarray*] Spectrum with only positive frequencies and 0.

sampling_rate_hz
[int] Signal's sampling rate in Hz.

group_delay_ms
[str or float, optional] Constant group delay that the phase should have for all channels (in ms). Pass *'minimal'* to create a signal with the minimum linear phase possible (that is different for each channel). Default: *'minimal'*.

check_causality
[bool, optional] When *True*, it is assessed for each channel that the given group delay is not lower than the minimal group delay. Default: *True*.

signal_type
[str, optional] Type of signal to be returned. Default: *'ir'*.

Returns

sig_lin_phase
[*Signal*] Signal with same magnitude spectrum but linear phase.

`dsptoolbox.special.`**`min_phase_from_mag`**(*spectrum: ndarray*, *sampling_rate_hz:* int, *signal_type:* str *= 'ir'*)

Returns a minimal phase signal from a magnitude spectrum using the hilbert transform.

Parameters

spectrum
[*np.ndarray*] Spectrum with only positive frequencies and 0.

sampling_rate_hz
[int] Signal's sampling rate in Hz.

signal_type
[str, optional] Type of signal to be returned. Default: *'ir'*.

Returns

sig_min_phase
[*Signal*] Signal with same magnitude spectrum but minimal phase.

References

- https://en.wikipedia.org/wiki/Minimum_phase

3.8 Standard functions (dsptoolbox.*)

3.8.1 Standard functions

This module contains a general collection of DSP functions that do not fall under a same category.

`dsptoolbox.standard_functions.erb_frequencies`(*freq_range_hz=[20, 20000]*, *resolution: float = 1*, *reference_frequency_hz=1000*)

Get frequencies that are linearly spaced on the ERB frequency scale. This implementation was taken and adapted from the pyfar package. See references.

Parameters

freq_range
[array-like] The upper and lower frequency limits in Hz between which the frequency vector is computed.

resolution
[float, optional] The frequency resolution in ERB units. 1 returns frequencies that are spaced by 1 ERB unit, a value of 0.5 would return frequencies that are spaced by 0.5 ERB units. Default: 1.

reference_frequency
[float, optional] The reference frequency in Hz relative to which the frequency vector is constructed. Default: 1000.

Returns

frequencies
[*np.ndarray*] The frequencies in Hz that are linearly distributed on the ERB scale with a spacing given by *resolution* ERB units.

References

- B. C. J. Moore, An introduction to the psychology of hearing, (Leiden, Boston, Brill, 2013), 6th ed.
- V. Hohmann, "Frequency analysis and synthesis using a gammatone filterbank," Acta Acust. united Ac. 88, 433-442 (2002).
- P. L. Søndergaard, and P. Majdak, "The auditory modeling toolbox," in The technology of binaural listening, edited by J. Blauert (Heidelberg et al., Springer, 2013) pp. 33-56.
- The pyfar package: https://github.com/pyfar/pyfar

`dsptoolbox.standard_functions.`**`excess_group_delay`**`(`*signal:* Signal`)`

Computes excess group delay.

Parameters

signal
[Signal] Signal object for which to compute minimal group delay.

Returns

f
[*np.ndarray*] Frequency vector.

ex_gd
[*np.ndarray*] Excess group delays in seconds with shape (excess_gd, channel).

References

- https://www.roomeqwizard.com/help/help_en-GB/html/minimumphase.html

`dsptoolbox.standard_functions.`**`fade`**`(`*sig:* Signal, *type_fade: str = 'lin', length_fade_seconds: Optional[float] = None, at_start: bool = True, at_end: bool = True*`)` → *Signal*

Applies fading to signal.

Parameters

sig
[*Signal*] Signal to apply fade to.

type_fade
[str, optional] Type of fading to be applied. Choose from *'exp'* (exponential), *'lin'* (linear) or *'log'* (logarithmic). Default: *'lin'*.

length_fade_seconds
[float, optional] Fade length in seconds. If *None*, 2.5% of the signal's length is used for the fade. Default: *None*.

at_start
[bool, optional] When *True*, the start of signal of faded. Default: *True*.

at_end
[bool, optional] When *True*, the ending of signal of faded. Default: *True*.

Returns

new_sig
[*Signal*] New Signal

`dsptoolbox.standard_functions.`**`fractional_octave_frequencies`**(*num_fractions=1*, *frequency_range=(20, 20000.0)*, *return_cutoff=False*)

Return the octave center frequencies according to the IEC 61260:1:2014 standard.

For numbers of fractions other than `1` and `3`, only the exact center frequencies are returned, since nominal frequencies are not specified by corresponding standards.

Parameters

num_fractions
[int, optional] The number of bands an octave is divided into. Eg., `1` refers to octave bands and `3` to third octave bands. The default is `1`.

frequency_range
[array, tuple] The lower and upper frequency limits, the default is `frequency_range=(20, 20e3)`.

Returns

nominal
[array, float] The nominal center frequencies in Hz specified in the standard. Nominal frequencies are only returned for octave bands and third octave bands.

exact
[array, float] The exact center frequencies in Hz, resulting in a uniform distribution of frequency bands over the frequency range.

cutoff_freq
[tuple, array, float] The lower and upper critical frequencies in Hz of the bandpass filters for each band as a tuple corresponding to `(f_lower, f_upper)`.

References

- This implementation is taken from the pyfar package. See https://github.com/pyfar/pyfar

`dsptoolbox.standard_functions.`**`group_delay`**(*signal:* Signal, *method='matlab'*)

Computation of group delay.

Parameters

signal
[Signal] Signal for which to compute group delay.

method
[str, optional] *'direct'* uses gradient with unwrapped phase. *'matlab'* uses this implementation: https://www.dsprelated.com/freebooks/filters/Phase_Group_Delay.html. Default: *'matlab'*.

Returns

freqs
[*np.ndarray*] Frequency vector in Hz.

group_delays
[*np.ndarray*] Matrix containing group delays in seconds with shape (gd, channel).

`dsptoolbox.standard_functions.`**`ir_to_filter`**(*signal:* Signal, *channel: int = 0*, *phase_mode: str = 'direct'*) → *Signal*

This function takes in a signal with type ir or rir and turns the selected channel into an FIR filter. With *phase_mode* it is possible to use minimum phase or minimum linear phase.

Parameters

signal
[*Signal*] Signal to be converted into a filter.

channel
[int, optional] Channel of the signal to be used. Default: 0.

phase_mode
[str, optional] Phase of the FIR filter. Choose from *'direct'* (no phase changing), *'min'* (minimum phase) or *'lin'* (linear phase). Default: *'direct'*.

Returns

filt
[*Filter*] FIR filter object.

`dsptoolbox.standard_functions.`**`latency`**`(`*in1:* Signal, *in2:* *Optional*[Signal] *= None*`)`

Computes latency between two signals using the correlation method. If there is no second signal, the latency between the first and the other channels of the is computed.

Parameters

in1
[Signal] First signal.

in2
[Signal, optional] Second signal. Default: *None*.

Returns

latency_per_channel_samples
[*np.ndarray*] Array with latency between two signals in samples per channel.

`dsptoolbox.standard_functions.`**`load_pkl_object`**`(`*path:* *str*`)`

WARNING: This is not secure. Only unpickle data you know! Loads a optimization object with all its attributes and methods.

Parameters

path
[str] Path to the pickle object.

Returns

obj
[object] Unpacked pickle object.

`dsptoolbox.standard_functions.`**`merge_filterbanks`**`(`*fb1:* FilterBank, *fb2:* FilterBank`)` → *FilterBank*

Merges two filterbanks.

Parameters

fb1
[*FilterBank*] First filterbank.

fb2
[*FilterBank*] Second filterbank.

Returns

new_fb
[*FilterBank*] New filterbank with merged filters

`dsptoolbox.standard_functions.`**`merge_signals`**(*in1:* dsptoolbox.classes.signal_class.Signal | dsptoolbox.classes.multibandsignal.MultiBandSignal, *in2:* dsptoolbox.classes.signal_class.Signal | dsptoolbox.classes.multibandsignal.MultiBandSignal, *padding_trimming:* bool *= True*, *at_end:* bool *= True*) → *dsptoolbox.classes.signal_class.Signal* | *dsptoolbox.classes.multibandsignal.MultiBandSignal*

Merges two signals by appending the channels of the second one to the first. If the length of in2 is not the same, trimming or padding is applied at the end when *padding_trimming=True*, otherwise an error is raised.

Parameters

in1
[*Signal* or *MultiBandSignal*] First signal.

in2
[*Signal* or *MultiBandSignal*] Second signal.

padding_trimming
[bool, optional] If the signals do not have the same length, the second one is padded or trimmed. When *True*, padding/trimming is done. Default: *True*.

at_end
[bool, optional] When *True* and *padding_trimming=True*, padding or trimming is done at the end of signal. Otherwise it is done in the beginning. Default: *True*.

Returns

new_sig
[*Signal*] New merged signal.

`dsptoolbox.standard_functions.`**`minimum_group_delay`**(*signal:* Signal)

Computes minimum group delay of given signal.

Parameters

signal
[Signal] Signal object for which to compute minimal group delay.

Returns

f
[*np.ndarray*] Frequency vector.

min_gd
[*np.ndarray*] Minimal group delays in seconds as matrix with shape (gd, channel).

References

- https://www.roomeqwizard.com/help/help_en-GB/html/minimumphase.html

`dsptoolbox.standard_functions.`**`minimum_phase`**`(`*signal:* Signal`)`

Gives back a matrix containing the minimal phase for every channel.

Parameters

signal
[Signal] Signal for which to compute the minimal phase.

Returns

f
[*np.ndarray*] Frequency vector.

min_phases
[*np.ndarray*] Minimal phases as matrix with shape (phase, channel).

`dsptoolbox.standard_functions.`**`normalize`**`(`*sig:* dsptoolbox.classes.signal_class.Signal | dsptoolbox.classes.multibandsignal.MultiBandSignal, *peak_dbfs:* float *= -6*, *each_channel:* bool *= False*`)` → *dsptoolbox.classes.signal_class.Signal* | *dsptoolbox.classes.multibandsignal.MultiBandSignal*

Normalizes a signal to a given peak value. It either normalizes each channel or the signal as a whole.

Parameters

sig
[*Signal* or *MultiBandSignal*] Signal to be normalized.

peak_dbfs
[float, optional] dBFS to which to normalize peak. Default: -6.

each_channel
[bool, optional] When *True*, each channel on its own is normalized. When *False*, the peak value for all channels is regarded. Default: *False*.

Returns

new_sig
[*Signal* or *MultiBandSignal*]

`dsptoolbox.standard_functions.`**`pad_trim`**`(`*signal:* dsptoolbox.classes.signal_class.Signal | dsptoolbox.classes.multibandsignal.MultiBandSignal, *desired_length_samples:* int, *in_the_end:* bool *= True*`)` → *dsptoolbox.classes.signal_class.Signal* | *dsptoolbox.classes.multibandsignal.MultiBandSignal*

Returns a copy of the signal with padded or trimmed time data. Only valid for same sampling rate MultiBandSignal.

Parameters

signal
[*Signal* or *MultiBandSignal*] Signal to be padded or trimmed.

desired_length_samples
[int] Length of resulting signal.

in_the_end
[bool, optional] Defines if padding or trimming should be done in the beginning or in the end of the signal. Default: *True*.

Returns

new_signal
[*Signal* or *MultiBandSignal*] New padded signal.

`dsptoolbox.standard_functions.`**`resample`**`(`*sig:* Signal, *desired_sampling_rate_hz:* *int*`)` → *Signal*

Resamples signal to the desired sampling rate using *scipy.signal.resample_poly* with an efficient polyphase representation.

Parameters

sig
[*Signal*] Signal to be resampled.

desired_sampling_rate_hz
[int] Sampling rate to convert the signal to.

Returns

new_sig
[*Signal*] Resampled signal.

`dsptoolbox.standard_functions.`**`true_peak_level`**`(`*sig:* dsptoolbox.classes.signal_class.Signal | dsptoolbox.classes.multibandsignal.MultiBandSignal`)`

Computes true-peak level of a signal using the standardized method by the Rec. ITU-R BS.1770-4. See references.

Parameters

sig
[*Signal* or *MultiBandSignal*] Signal for which to compute the true-peak level.

Returns

true_peak_levels
[*np.ndarray*] True-peak levels (in dBTP) as an array with shape (channels) or (band, channels) in case that the input signal is *MultiBandSignal*.

peak_levels
[*np.ndarray*] Peak levels (in dBFS) as an array with shape (channels) or (band, channels) in case that the input signal is *MultiBandSignal*.

References

- https://www.itu.int/rec/R-REC-BS.1770

3.9 Transfer functions (dsptoolbox.transfer_functions)

3.9.1 Transfer functions

In this module there are functions used to obtain or modify transfer functions (TF).

Acquire TF from signals:

- *spectral_deconvolve()* (direct deconvolution)
- *compute_transfer_function()* (using welch's method for estimating auto- and cross correlation spectra from measurements)

Modify TF:

- *window_ir()* (Windows a TF in time domain)

`dsptoolbox.transfer_functions.`**`compute_transfer_function`**(*output:* Signal, *input:* Signal, *mode='h2'*, *window_length_samples:* *int = 1024*, ***kwargs*) → *Signal*

Gets transfer function H1, H2 or H3. H1: for noise in the output signal. *Gxy/Gxx*. H2: for noise in the input signal. *Gyy/Gyx*. H3: for noise in both signals. *G_xy / np.abs(G_xy) * (G_yy/G_xx)**0.5*. If the input signal only has one channel, it is assumed to be the input for all of the channels of the output.

Parameters

output
[*Signal*] Signal with output channels.

input
[*Signal*] Signal with input channels.

mode
[str, optional] Type of transfer function. *'h1'*, *'h2'* and *'h3'* are available. Default: *'h2'*.

window_length_samples
[int, optional] Window length for the IR. Spectrum has the length window_length_samples//2 + 1. Default: 1024.

****kwargs**
[dict, optional] Extra parameters for the computation of the cross spectral densities using welch's method.

Returns

tf
[*Signal*] Transfer functions. Coherences are also computed and saved in the Signal object.

`dsptoolbox.transfer_functions.`**`spectral_deconvolve`**(*num:* Signal, *denum:* Signal, *mode='regularized'*, *start_stop_hz=None*, *threshold_db=-30*, *padding:* *bool = False*, *keep_original_length:* *bool = False*) → *Signal*

Deconvolution by spectral division of two signals. If the denominator signal only has one channel, the deconvolution is done using it for all channels of the numerator.

Parameters

num
[*Signal*] Signal to deconvolve from.

denum
[*Signal*] Signal to deconvolve.

mode
[str, optional] *'window'* uses a spectral window in the numerator. *'regularized'* uses a regularized inversion. *'standard'* uses direct deconvolution. Default: *'regularized'*.

start_stop_hz
[array, None, optional] *'automatic'* uses a threshold dBFS to create a spectral window for the numerator or regularized inversion. Array of 2 or 4 frequency points can be also manually given. *None* uses no spectral window.

threshold
[int, optional] Threshold in dBFS for the automatic creation of the window. Default: -30.

padding
[bool, optional] Pads the time data with 2 length. Done for separating distortion in negative time bins when deconvolving sweep measurements. Default: *False*.

keep_original_length
[bool, optional] Only regarded when padding is *True*. It trims the newly deconvolved data to its original length. Default: *False*.

Returns

new_sig
[*Signal*] Deconvolved signal.

`dsptoolbox.transfer_functions.`**`window_ir`**(*signal: Signal*, *constant_percentage=0.75*, *exp2_trim: int = 13*, *window_type='hann'*, *at_start: bool = True*) → *Signal*

Windows an IR with trimming and selection of constant valued length.

Parameters

signal: `Signal`
Signal to window

constant_percentage: float, optional
Percentage (between 0 and 1) of the window that should be constant value. Default: 0.75

exp2_trim: int, optional
Exponent of two defining the length to which the IR should be trimmed. For avoiding trimming set to *None*. Default: 13.

window_type: str, optional
Window function to be used. Available selection from scipy.signal.windows: *barthann*, *bartlett*, *blackman*, *boxcar*, *cosine*, *hamming*, *hann*, *flattop*, *nuttall* and others without extra parameters. Default: *hann*.

at_start: bool, optional
Windows the start with a rising window as well as the end. Default: *True*.

Returns

new_sig
[*Signal*] Windowed signal. The used window is also saved under *new_sig.window*.

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX