Acoustics Documentation

Release 0.0

Frederik Rietdijk

November 28, 2013

Contents

Contents:

CHAPTER 1

Reference

This section contains a description of all classes and functions.

1.1 Octave

acoustics.octave.Octave

Module for working with octaves.

The following is an example on how to use `acoustics.octave.Octave`.

```
"""
An example of how to use :class:`Acoustics.Octave.Octave`.
"""

from acoustics.octave import Octave
import numpy as np

def main():

    """We happen to have the following frequency vector."""
    f = np.logspace(2, 4, 100)

    """And we want to get some 1/3-octave results, so we set octave to 3."""
    o = Octave(interval=f, order=3)

    """We can now print the center frequencies of the 1/3 octave bands belonging to this frequency ve
    print(o.center())

    """
    Since by default ``unique=True`` in :class:`Auraliser.Octave.Octave` we get a
    value for every value in ``interval``. If we only want to obtain unique
```

```
    values, it is fastest to set ``unique=True``. This could be done during
    initialization but also now.
    """
    o.unique = True

    """Now we should only get the unique center frequencies."""
    print(o.center())

    """We can also calculate the bandwidth of each band."""
    print(o.bandwidth())

    """As well as the lower limits of the frequency limits..."""
    print(o.lower())

    """...and the upper frequency limits."""
    print(o.upper())

    """
    So far we used a frequency interval. Sometimes you have a lower frequency and an upper frequency
    Instead of requiring to generate a frequency vector you can just give these boundary values as we
    """
    o = Octave(fmin=100.0, fmax=20000, order=6)

    print(o.center())

if __name__ == '__main__':
    main()
```

class acoustics.octave.**Octave**(*order=1*, *interval=None*, *fmin=None*, *fmax=None*, *unique=False*)

Bases: object

Class to calculate octave center frequencies.

REF = 1000.0

Reference center frequency $f_{c,0}$.

bandwidth()

Bandwidth of bands.

$$B = f_u - f_l$$

center()

Return center frequencies f_c.

$$f_c = f_{ref} \cdot 2^{n/N} \cdot 10^{\frac{3}{10N}}$$

fmax

Maximum frequency of an interval.

fmin

Minimum frequency of an interval.

interval

Interval.

lower()
Lower frequency limits of bands.

$$f_l = f_c \cdot 2^{\frac{-1}{2N}}$$

n()
Return band `n` for a given frequency.

order = None
Order

unique = None
Whether or not to calculate the requested values for every value of `interval`.

upper()
Upper frequency limits of bands.

$$f_u = f_c \cdot 2^{\frac{+1}{2N}}$$

`acoustics.octave.`**band_of_frequency**(*f*, *order=1*, *ref=1000.0*)
Calculate the band `n` from a given frequency.

Parameters **f** – Frequency

1.2 Doppler

Doppler shift module.

`acoustics.doppler.`**velocity_from_doppler_shift**(*c*, *f1*, *f2*)
Calculate velocity based on measured frequency shifts due to Doppler shift. The assumption is made that the velocity is constant between the observation times.

$$v = c \cdot \left(\frac{f_2 - f_1}{f_2 + f_1} \right)$$

Parameters

- **c** – Speed of sound c.
- **f1** – Lower frequency f_1.
- **f2** – Upper frequency f_2.

1.3 Building

This module contains functions related to building acoustics.

`acoustics.building.`**mass_law**(*freq*, *vol_density*, *thickness*, *theta=0*, *c=343*, *rho0=1.225*)
Calculate transmission loss according to mass law.

Parameters

- **freq** (*float* or *NumPy array*) – Frequency of interest in Hz.

- **vol_density** (*float*) – Volumetric density of material in [kg/m^3].
- **thickness** (*float*) – Thickness of wall.
- **theta** (*float*) – Angle of incidence in degrees. Default value is *0* (normal incidence).
- **c** (*float*) – Speed of sound in [m/s].
- **rho0** (*float*) – Density of air in kg/m^3.

`acoustics.building.`**`rw`**(*tl*)
Calculate R_W from a NumPy array *tl* with third octave data between 100 Hz and 3.15 kHz.

Parameters **tl** – Transmission Loss

`acoustics.building.`**`rw_c`**(*tl*)
Calculate $R_W + C$ from a NumPy array *tl* with third octave data between 100 Hz and 3.15 kHz.

Parameters **tl** – Transmission Loss

`acoustics.building.`**`rw_ctr`**(*tl*)
Calculate $R_W + C_{tr}$ from a NumPy array *tl* with third octave data between 100 Hz and 3.15 kHz.

Parameters **tl** – Transmission Loss

`acoustics.building.`**`rw_curve`**(*tl*)
Calculate the curve of Rw from a NumPy array *tl* with third octave data between 100 Hz and 3.15 kHz.

Parameters **tl** – Transmission Loss

`acoustics.building.`**`stc`**(*tl*)
Calculate the Sound Transmission Class (STC) from a NumPy array *tl* with third octave data between 125 Hz and 4 kHz.

Parameters **tl** – Transmission Loss

`acoustics.building.`**`stc_curve`**(*tl*)
Calculate the Sound Transmission Class (STC) curve from a NumPy array *tl* with third octave data between 125 Hz and 4 kHz.

Parameters **tl** – Transmission Loss

1.4 Signal

This module constains a function to perform a convolution of signal with a Linear Time-Variant system.

`acoustics.signal.`**`butter_bandpass_filter`**(*data*, *lowcut*, *highcut*, *fs*, *order=3*)
Butterworth bandpass filter.

Parameters

- **data** – data
- **lowcut** – Lower cut-off frequency
- **highcut** – Upper cut-off frequency
- **fs** – Sample frequency
- **order** – Order

`acoustics.signal.`**`convolve`**(*signal*, *ltv*)
Perform convolution of signal with linear time-variant system `ltv`.

Parameters

- **signal** – Vector representing input signal u.
- **ltv** – 2D array where each column represents an impulse response

The convolution of two sequences is given by

$$\mathbf{y} = \mathbf{t} \star \mathbf{u}$$

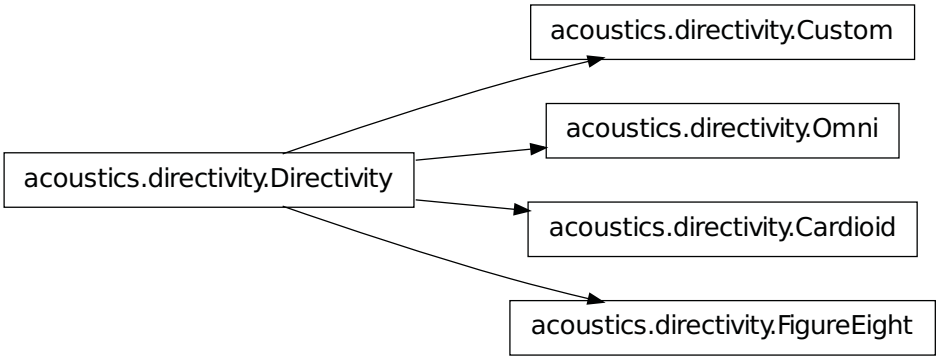
This can be written as a matrix-vector multiplication

$$\mathbf{y} = \mathbf{T} \cdot \mathbf{u}$$

where T is a Toeplitz matrix in which each column represents an impulse response. In the case of a linear time-invariant (LTI) system, each column represents a time-shifted copy of the first column. In the time-variant case (LTV), every column can contain a unique impulse response, both in values as in size.

This function assumes all impulse responses are of the same size. The input matrix `ltv` thus represents the non-shifted version of the Toeplitz matrix.

1.5 Directivity



Class and functions to work with directivity.

The following conventions are used within this module:

- The angle θ has a range $[0, \pi]$.
- The angle ϕ has a range $[-pi, pi]$.

class `acoustics.directivity.`**`Cardioid`**(*rotation=None*)
Bases: `acoustics.directivity.Directivity`

Cardioid directivity.

class `acoustics.directivity.`**`Custom`**(*theta=None*, *phi=None*, *r=None*)
Bases: `acoustics.directivity.Directivity`

A class to work with directivity.

phi = None
Longitude. 1-D array.

r = None
Magnitude or radius. 2-D array.

theta = None
Latitude. 1-D array.

class acoustics.directivity.**Directivity**(*rotation=None*)
Bases: object

Abstract directivity class.

This class defines several methods to be implemented by subclasses.

plot(*filename=None*, *include_rotation=True*)
Directivity plot. Plot to filename when given.

Parameters

- **filename** – Filename
- **include_rotation** – Apply the rotation to the directivity. By default the rotation is applied in this figure.

rotation = None
Rotation of the directivity pattern.

using_cartesian(*x*, *y*, *z*, *include_rotation=True*)
Return the directivity for given cartesian coordinates.

Parameters

- **x** – x
- **y** – y
- **z** – z

using_spherical(*r*, *theta*, *phi*, *include_rotation=True*)
Return the directivity for given spherical coordinates.

Parameters

- **r** – norm
- **theta** – angle θ
- **phi** – angle ϕ

class acoustics.directivity.**FigureEight**(*rotation=None*)
Bases: acoustics.directivity.Directivity

Figure of eight directivity.

class acoustics.directivity.**Omni**(*rotation=None*)
Bases: acoustics.directivity.Directivity

Class to work with omni-directional directivity.

acoustics.directivity.**cardioid**(*theta*, *a=1.0*, *k=1.0*)
A cardioid pattern.

Parameters

- **a** – a

- **k** – k

acoustics.directivity.**cartesian_to_spherical**(*x*, *y*, *z*)

Convert cartesian coordinates to spherical coordinates.

Parameters

- **x** – x
- **y** – y
- **z** – z

$$r = \sqrt{(x^2 + y^2 + z^2)}$$

$$\theta = \arccos \frac{z}{r}$$

$$\phi = \arccos \frac{y}{x}$$

acoustics.directivity.**figure_eight**(*theta*)

A figure-of-eight pattern.

Parameters **theta** – angle θ

acoustics.directivity.**spherical_to_cartesian**(*r*, *theta*, *phi*)

Convert spherical coordinates to cartesian coordinates.

Parameters

- **r** – norm
- **theta** – angle θ
- **phi** – angle ϕ

$$x = r \sin\theta \cos\phi$$

$$y = r \sin\theta \sin\phi$$

$$z = r \cos\theta$$

1.6 Reflection

acoustics.reflection.Boundary

The reflection module contains functions for calculating reflection factors.

class acoustics.reflection.**Boundary**(*frequency*, *flow_resistivity*, *density=1.296*, *soundspeed=343.0*, *porosity_decrease=120.0*, *specific_heat_ratio=1.4*, *angle=None*, *distance=None*, *impedance_model='db'*, *reflection_model='plane'*)

Bases: object

An object describing a boundary.

impedance()
Impedance according to chosen impedance model defined using impedance_model().

impedance_model = None
Impedance model.

Possibilities are 'db' and 'att' for respectively impedance_delany_and_bazley() and impedance_attenborough().

plot_impedance(*filename=None*)
Plot impedance.

reflection_factor()
Reflection factor according to chosen reflection factor model defined using reflection_model().

reflection_model = None
Reflection factor model.

Possibilities are 'plane' and 'spherical'` for respectively reflection_factor_plane_wave() and reflection_factor_spherical_wave().

acoustics.reflection.**DENSITY = 1.296**
Density of air ρ.

acoustics.reflection.**POROSITY_DECREASE = 120.0**
Rate of exponential decrease of porosity with depth α.

acoustics.reflection.**SOUNDSPEED = 343.0**
Speed of sound in air c.

acoustics.reflection.**SPECIFIC_HEAT_RATIO = 1.4**
Specific heat ratio of air γ.

acoustics.reflection.**impedance_attenborough**(*frequency*, *flow_resistivity*, *density=1.296*, *soundspeed=343.0*, *porosity_decrease=120.0*, *specific_heat_ratio=1.4*)
Impedance according to the two-parameter model by Attenborough.

Parameters

- **frequency** – Frequency f.
- **flow_resistivity** – Flow resistivity σ.
- **soundspeed** – Speed of sound in air c.
- **density** – Density of air ρ.
- **porosity_decrease** – Rate of exponential decrease of porosity with depth α.
- **specific_heat_ratio** – Ratio of specific heats for air.

The impedance Z is given by

$$Z = \frac{(1-j)\sqrt{\sigma/f}}{\sqrt{\pi\gamma_0\rho_0}} - \frac{jc\alpha}{8\pi\gamma_0 f}$$

acoustics.reflection.**impedance_delany_and_bazley**(*frequency*, *flow_resistivity*)

Impedance according to the empirical one-parameter model by Delany and Bazley.

Parameters

- **frequency** – Frequency f.
- **flow_resistivity** – Flow resistivity σ.

The impedance Z is given by

$$Z = 1 + 9.08\left(\frac{1000f}{\sigma}\right)^{-0.75} - 11.9j\left(\frac{1000f}{\sigma}\right)^{-0.73}$$

acoustics.reflection.**numerical_distance**(*impedance*, *angle*, *distance*, *wavenumber*)

Numerical distance.

Parameters

- **impedance** – Normalized impedance.
- **angle** – Angle of incidence.
- **distance** – Path length of the reflected ray.
- **wavenumber** – Wavenumber.

The numerical distance w is given by

$$w = \sqrt{-jkR\left(1 + \frac{1}{Z}\cos\theta - \sqrt{1 - \left(\frac{1}{Z}\right)^2}\sin\theta\right)}$$

acoustics.reflection.**reflection_factor_plane_wave**(*impedance*, *angle*)

Plane wave reflection factor.

Parameters

- **impedance** – Normalized impedance.
- **angle** – Angle of incidence.

The plane wave reflection factor R is given by

$$R = \frac{Z\cos\theta - 1}{Z\cos\theta + 1}$$

where Z is the normalized impedance and θ the angle of incidence.

acoustics.reflection.**reflection_factor_spherical_wave**(*impedance*, *angle*, *distance*, *wavenumber*)

Spherical wave reflection factor.

Parameters

- **impedance** – Normalized impedance.
- **angle** – Angle of incidence.
- **distance** – Path length of the reflected ray.
- **wavenumber** – Wavenumber.

The spherical wave relfection factor Q is given by

$$Q = R\left(1 - R\right)F$$

where R is the plane wave reflection factor as calculated in `plane_wave_reflection_factor()` and F is given by

$$F = 1 - j\sqrt{\pi}we^{-w^2}\operatorname{erfc}\left(jw\right)$$

where w is the numerical distance as calculated in `numerical_distance()`.

1.7 Room

acoustics.room.**mean_alpha**(*alphas*, *surfaces*)

Calculate mean of absorption coefficients.

Parameters

- **alphas** – Absorption coefficients
- **surfaces** – Surfaces

acoustics.room.**nrc**(*alphas*)

Calculate Noise Reduction Coefficient (NRC) from four absorption coefficient values (250, 500, 1000 and 2000 Hz).

Parameters **alphas** – Absorption coefficients

acoustics.room.**t60_arau**(*Sx*, *Sy*, *Sz*, *alpha*, *volume*, *c=343*)

Reverberation time according to Arau. [1]

`Sx`: Sum of side walls.

`Sy`: Sum of other side walls.

`Sz`: Sum of room and floor surfaces.

[1] For more details, plase see http://www.arauacustica.com/files/publicaciones/pdf_esp_7.pdf

`alpha`: Absorption coefficients for Sx, Sy and Sz, respectively.

`volume`: Volume of the room.

`c`: Speed of sound.

`acoustics.room.`**`t60_eyring`**(*surfaces*, *alpha*, *volume*, *c=343*)
Reverberation time according to Eyring.

Parameters

- **surfaces** – Surfaces
- **alpha** – Mean absorption coefficient
- **volume** – Volume
- **c** – Speed of sound

`acoustics.room.`**`t60_fitzroy`**(*surfaces*, *alpha*, *volume*, *c=343*)
Reverberation time according to Fitzroy.

Parameters

- **surfaces** – Surfaces
- **alpha** – Mean absorption coefficient
- **volume** – Volume
- **c** – Speed of sound

`acoustics.room.`**`t60_millington`**(*surfaces*, *alpha*, *volume*, *c=343*)
Reverberation time according to Millington.

Parameters

- **surfaces** – Surfaces
- **alpha** – Mean absorption coefficient
- **volume** – Volume
- **c** – Speed of sound

`acoustics.room.`**`t60_sabine`**(*surfaces*, *alpha*, *volume*, *c=343*)
Reverberation time according to Sabine:

$$T_{60} = \frac{4ln(10^6)}{c}\frac{V}{S\alpha}$$

Where:

•c: speed of sound.

•V: Volume of the room.

•S: Surface of the room.

•α: Absorption coefficient of the room.

Parameters:

surfaces [ndarray] NumPy array that contains different surfaces.

alpha [ndarray] Contains absorption coefficients of `surfaces`. It could be one value or some values in different bands (1D and 2D array, respectively).

volume [float] Volume of the room.

c [float] Speed of sound (343 m/s by default).

CHAPTER 2

Indices and tables

- *genindex*
- *modindex*
- *search*

Python Module Index

a