spaudiopy Documentation

Release v0.1.4-53-ge80c0e3

Chris Hold

Apr 26, 2022

CONTENTS:

1 Installation **1**
- 1.1 Requirements . . . 1
- 1.2 Installation . . . 1

2 API Documentation **3**
- 2.1 spaudiopy.IO . . . 3
- 2.2 spaudiopy.sig . . . 7
- 2.3 spaudiopy.sph . . . 10
- 2.4 spaudiopy.decoder . . . 28
- 2.5 spaudiopy.process . . . 51
- 2.6 spaudiopy.utils . . . 58
- 2.7 spaudiopy.grids . . . 60
- 2.8 spaudiopy.sdm . . . 68
- 2.9 spaudiopy.plots . . . 70
- 2.10 Multiprocessing . . . 75

3 Indices and tables **77**

Python Module Index **79**

Index **81**

CHAPTER

ONE

INSTALLATION

For the unpatient, you can just install the pip version *pip install spaudiopy*

1.1 Requirements

It's easiest to start with something like Anaconda as a Python distribution. You'll need Python >= 3.6 .

1. **Create a conda environment:**
    - *conda create –name spaudio python=3.6 anaconda joblib portaudio*
2. **Activate this new environment:**
    - *conda activate spaudio*

Have a look at the *setup.py* file, all dependencies are listed there. When using *pip* to install this package as shown below, all remaining dependencies not available from conda will be downloaded and installed automatically.

1.2 Installation

Download this package from GitHub and navigate to there. Then simply run the line:

```
pip install -e .
```

This will check all dependencies and install this package as editable.

CHAPTER

TWO

API DOCUMENTATION

spaudiopy

Submodules

IO	Input Output (IO) helpers.
sig	Signal class.
sph	Spherical Harmonics.
decoder	Loudspeaker decoders.
process	Collection of audio processing tools.
utils	A few helpers.
grids	Sampling grids.
sdm	Spatial Decomposition Method (SDM).
plots	Plotting helpers.

2.1 spaudiopy.IO

Input Output (IO) helpers.

```
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['axes.grid'] = True

import spaudiopy as spa
```

Functions

get_default_hrirs([grid_azi, grid_colat])	Creates the default HRIRs loaded by load_hrirs() by inverse SHT.
load_audio(filenames[, fs])	Load mono and multichannel audio from files.
load_hrirs(fs[, filename])	Convenience function to load 'HRIRs.mat'.
load_layout(filename[, listener_position, ...])	Load loudspeaker layout from json configuration file.
load_sdm(filename[, init_nan])	Convenience function to load 'SDM.mat'.
load_sofa_data(filename)	Load .sofa file into python dictionary that contains the data in numpy arrays.
load_sofa_hrirs(filename)	Load SOFA file containing HRIRs.
save_audio(signal, filename[, fs, subtype])	Save signal to audio file.
save_layout(filename, ls_layout[, name, ...])	Save loudspeaker layout to json configuration file.
sofa_to_sh(filename, N_sph[, SH_type])	Load and transform SOFA IRs to the Spherical Harmonic Domain.
write_ssr_brirs_loudspeaker(filename, ...[, ...])	Write binaural room impulse responses (BRIRs) and save as wav file.
write_ssr_brirs_sdm(filename, sdm_p, ...[, ...])	Write binaural room impulse responses (BRIRs) and save as wav file.

spaudiopy.IO.**load_audio**(*filenames*, *fs=None*)

Load mono and multichannel audio from files.

Parameters **filenames** (*string or list of strings*) – Audio files.

Returns **sig** (*sig.MonoSignal or sig.MultiSignal*) – Audio signal.

spaudiopy.IO.**save_audio**(*signal*, *filename*, *fs=None*, *subtype='FLOAT'*)

Save signal to audio file.

Parameters

- **signal** (*sig. MonoSignal, sig.MultiSignal or np.ndarray*) – Audio Signal, forwarded to sf.write(); (frames x channels).
- **filename** (*string*) – Audio file name.
- **fs** (*int*) – fs(t).
- **subtype** (*optional*)

spaudiopy.IO.**load_hrirs**(*fs*, *filename=None*)

Convenience function to load 'HRIRs.mat'. The file contains ['hrir_l', 'hrir_r', 'fs', 'azi', 'colat'].

Parameters

- **fs** (*int*) – fs(t).
- **filename** (*string, optional*) – HRTF.mat file or default set, or 'dummy' for debugging.

Returns

HRIRs (*sig.HRIRs instance*) –

left [(g, h) numpy.ndarray] h(t) for grid position g.

right [(g, h) numpy.ndarray] h(t) for grid position g.

grid [(g, 2) pandas.dataframe] [azi: azimuth, colat: colatitude] for hrirs.

fs [int] fs(t).

spaudiopy.IO.**get_default_hrirs**(*grid_azi=None*, *grid_colat=None*)

Creates the default HRIRs loaded by load_hrirs() by inverse SHT. By default it renders onto a gauss grid of order N=35, and additionally resamples fs to 48kHz.

Parameters

- **grid_azi** (*array_like, optional*)
- **grid_colat** (*array_like, optional*)

Notes

HRTFs in SH domain obtained from http://dx.doi.org/10.14279/depositonce-5718.5

spaudiopy.IO.**load_sofa_data**(*filename*)

Load .sofa file into python dictionary that contains the data in numpy arrays.

spaudiopy.IO.**load_sofa_hrirs**(*filename*)

Load SOFA file containing HRIRs.

Parameters **filename** (*string*) – SOFA filepath.

Returns

HRIRs (*sig.HRIRs instance*) –

left [(g, h) numpy.ndarray] h(t) for grid position g.

right [(g, h) numpy.ndarray] h(t) for grid position g.

grid [(g, 2) pandas.dataframe] [azi: azimuth, colat: colatitude] for hrirs.

fs [int] fs(t).

spaudiopy.IO.**sofa_to_sh**(*filename*, *N_sph*, *SH_type='real'*)

Load and transform SOFA IRs to the Spherical Harmonic Domain.

Parameters

- **filename** (*string*) – SOFA file name.
- **N_sph** (*int*) – Spherical Harmonic Transform order.
- **SH_type** (*'real' (default) or 'complex' spherical harmonics.*)

Returns

- **IRs_nm** (*(2, (N_sph+1)**2, S) numpy.ndarray*) – Left and right (stacked) SH coefficients.
- **fs** (*int*)

spaudiopy.IO.**load_sdm**(*filename*, *init_nan=True*)

Convenience function to load 'SDM.mat'. The file contains ['h_ref' or 'p', 'sdm_azi' or 'sdm_phi', 'sdm_colat' or 'sdm_theta', 'fs'].

Parameters

- **filename** (*string*) – SDM.mat file
- **init_nan** (*bool, optional*) – Initialize nan to [0, pi/2].

Returns

- **h** (*(n,) array_like*) – p(t).
- **sdm_azi** (*(n,) array_like*) – Azimuth angle.

- **sdm_colat** (*(n,) array_like*) – Colatitude angle.
- **fs** (*int*) – fs(t).

spaudiopy.IO.**write_ssr_brirs_loudspeaker**(*filename*, *ls_irs*, *hull*, *fs*, *subtype='FLOAT'*, *hrirs=None*, *jobs_count=1*)

Write binaural room impulse responses (BRIRs) and save as wav file.

The azimuth resolution is one degree. The channels are interleaved and directly compatible to the SoundScape Renderer (SSR) ssr-brs.

Parameters

- **filename** (*string*)
- **ls_irs** (*(L, S) np.ndarray*) – Impulse responses of L loudspeakers, e.g. by hull.loudspeaker_signals().
- **hull** (*decoder.LoudspeakerSetup*)
- **fs** (*int*)
- **subtype** (*forwarded to sf.write(), optional*)
- **hrirs** (*sig.HRIRs, optional*)
- **jobs_count** (*int, optional*) – [CPU Cores], Number of Processes, switches implementation for n > 1.

spaudiopy.IO.**write_ssr_brirs_sdm**(*filename*, *sdm_p*, *sdm_phi*, *sdm_theta*, *fs*, *subtype='FLOAT'*, *hrirs=None*)

Write binaural room impulse responses (BRIRs) and save as wav file.

The azimuth resolution is one degree. The channels are interleaved and directly compatible to the SoundScape Renderer (SSR) ssr-brs.

Parameters

- **filename** (*string*)
- **sdm_p** (*(n,) array_like*) – Pressure p(t).
- **sdm_phi** (*(n,) array_like*) – Azimuth phi(t).
- **sdm_theta** (*(n,) array_like*) – Colatitude theta(t).
- **fs** (*int*)
- **subtype** (*forwarded to sf.write(), optional*)
- **hrirs** (*sig.HRIRs, optional*)

spaudiopy.IO.**load_layout**(*filename*, *listener_position=None*, *N_kernel=50*)

Load loudspeaker layout from json configuration file.

spaudiopy.IO.**save_layout**(*filename*, *ls_layout*, *name='unknown'*, *description='unknown'*)

Save loudspeaker layout to json configuration file.

2.2 spaudiopy.sig

Signal class. Avoid code duplications (and errors) by defining a few custom classes here.

```
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['axes.grid'] = True

import spaudiopy as spa
```

Functions

trim_audio(A, start, stop)	Trim copy of MultiSignal audio to start and stop in seconds.

Classes

AmbiBSignal(signals[, fs])	Signal class for first order Ambisonics B-format signals.
HRIRs(left, right, grid, fs)	Signal class for head-related impulse responses.
MonoSignal(signal, fs)	Signal class for a MONO channel audio signal.
MultiSignal(signals[, fs])	Signal class for a MULTI channel audio signal.

class spaudiopy.sig.**MonoSignal**(*signal*, *fs*)

Bases: object

Signal class for a MONO channel audio signal.

__init__(*signal*, *fs*)

Constructor

Parameters

- **signal** (*array_like*)
- **fs** (*int*)

classmethod from_file(*filename*, *fs=None*)

Alternative constructor, load signal from filename.

copy()

Return an independent (deep) copy of the signal.

save(*filename*)

Save to file.

trim(*start*, *stop*)

Trim audio to start and stop in seconds.

apply(*func*, **args*, ***kwargs*)

Apply function 'func' to signal, arguments are forwarded.

conv(*h*, ***kwargs*)

Convolve signal, kwargs are forwarded to signal.convolve.

play(*gain=1*, *wait=True*)

Play sound signal. Adjust gain and wait until finished.

class spaudiopy.sig.**MultiSignal**(*signals*, *fs=None*)

Bases: *spaudiopy.sig.MonoSignal*

Signal class for a MULTI channel audio signal.

__init__(*signals*, *fs=None*)

Constructor

Parameters

- **signals** (*list of array_like*)
- **fs** (*int*)

classmethod from_file(*filename*, *fs=None*)

Alternative constructor, load signal from filename.

get_signals()

Return ndarray of signals, stacked along rows.

trim(*start*, *stop*)

Trim all channels to start and stop in seconds.

apply(*func*, **args*, ***kwargs*)

Apply function 'func' to all signals, arguments are forwarded.

conv(*irs*, ***kwargs*)

Convolve signal, kwargs are forwarded to signal.convolve.

play(*gain=1*, *wait=True*)

Play sound signal. Adjust gain and wait until finished.

copy()

Return an independent (deep) copy of the signal.

save(*filename*)

Save to file.

class spaudiopy.sig.**AmbiBSignal**(*signals*, *fs=None*)

Bases: *spaudiopy.sig.MultiSignal*

Signal class for first order Ambisonics B-format signals.

__init__(*signals*, *fs=None*)

Constructor

Parameters

- **signals** (*list of array_like*)
- **fs** (*int*)

classmethod from_file(*filename*, *fs=None*)

Alternative constructor, load signal from filename.

classmethod sh_to_b(*multisig*)

Alternative constructor, convert from sig.Multisignal. Assumes ACN channel order.

apply(*func*, **args*, ***kwargs*)

Apply function 'func' to all signals, arguments are forwarded.

conv(*irs*, ***kwargs*)

Convolve signal, kwargs are forwarded to signal.convolve.

copy()

Return an independent (deep) copy of the signal.

get_signals()

Return ndarray of signals, stacked along rows.

play(*gain=1*, *wait=True*)

Play sound signal. Adjust gain and wait until finished.

save(*filename*)

Save to file.

trim(*start*, *stop*)

Trim all channels to start and stop in seconds.

class spaudiopy.sig.**HRIRs**(*left*, *right*, *grid*, *fs*)

Bases: object

Signal class for head-related impulse responses.

__init__(*left*, *right*, *grid*, *fs*)

Constructor. HRIRs of size [numDirs, numTaps]

nearest_hrirs(*azi*, *colat*)

For a point on the sphere, select closest hrir defined on grid, based on the haversine distance.

Parameters

- **azi** (*float*) – Azimuth.
- **colat** (*float*) – Zenith / Colatitude.

Returns

- **h_l** (*(n,) array_like*) – h(t) closest to [phi, theta].
- **h_r** (*(n,) array_like*) – h(t) closest to [phi, theta].

nearest(*azi*, *colat*)

Index of nearest hrir grid point based on haversine distance.

Parameters

- **azi** (*float*) – Azimuth.
- **colat** (*float*) – Zenith / Colatitude.

Returns **idx** (*int*) – Index.

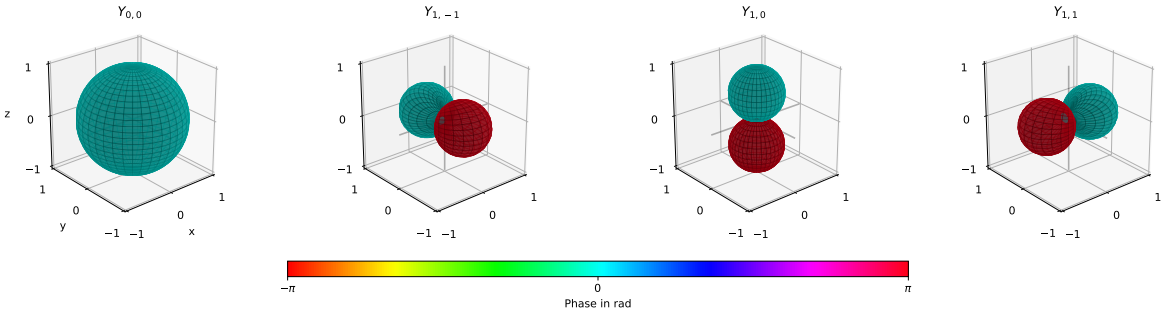
spaudiopy.sig.**trim_audio**(*A*, *start*, *stop*)

Trim copy of MultiSignal audio to start and stop in seconds.

2.3 spaudiopy.sph

Spherical Harmonics.

```
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['axes.grid'] = True

import spaudiopy as spa

spa.plots.sh_coeffs_subplot([np.sqrt(4*np.pi) * np.array([1, 0, 0, 0]),
                             np.sqrt(4/3*np.pi) * np.array([0, 1, 0, 0]),
                             np.sqrt(4/3*np.pi) * np.array([0, 0, 1, 0]),
                             np.sqrt(4/3*np.pi) * np.array([0, 0, 0, 1])],
                            titles=["$Y_{0, 0}$", "$Y_{1, -1}$",
                                    "$Y_{1, 0}$", "$Y_{1, 1}$"])
```



Functions

b_to_sh(B[, W_weight])	Convert B-format input to first order SH signals.
bandlimited_dirac(N, d[, w_n])	Order N spatially bandlimited Dirac pulse at central angle d.
binaural_coloration_compensation(N, f[, ...])	Spectral equalization gain G(kr)|N for diffuse field of order N.
butterworth_modal_weights(N_sph, k, n_c[, ...])	Modal weights for spatial butterworth filter / beamformer.
calculate_grid_weights(azi, zen[, order])	Approximate quadrature weights by pseudo-inverse.
cardioid_modal_weights(N_sph)	Modal weights for beamformer resulting in a cardioid.
check_cond_sht(N, azi, colat, SH_type[, lim])	Check if condition number for a least-squares SHT(N) is greater 'lim'.
design_spat_filterbank(N_sph, sec_azi, ...)	Design spatial filter bank analysis and reconstruction matrix.
hypercardioid_modal_weights(N_sph)	Modal weights for beamformer resulting in a hypercardioid.
inverse_sht(F_nm, azi, colat, SH_type[, N, Y_nm])	Perform the inverse spherical harmonics transform.
max_rE_weights(N)	Return max-rE modal weight coefficients for spherical harmonics order N.
maxre_modal_weights(N_sph[, UNITAMP])	Modal weights for beamformer resulting with max-rE weighting.
mode_strength(n, kr[, sphere_type])	Mode strength b_n(kr) for an incident plane wave on sphere.
n3d_to_sn3d(F_nm[, sh_axis])	Convert N3D (orthonormal) to SN3D (Schmidt semi-normalized) signals.
platonic_solid(shape)	Return coordinates of shape='tetrahedron' only, yet.
pressure_on_sphere(N, kr[, weights])	Calculate the diffuse field pressure frequency response of a spherical scatterer, up to SH order N.
project_on_sphere(x, y, z)	Little helper that projects x, y, z onto unit sphere.
r_E(p, g)	Calculate r_E vector and magnitude from loudspeaker gains.
repeat_per_order(c)	Repeat each coefficient in 'c' m times per spherical order n.
sh_matrix(N, azi, colat[, SH_type, weights])	Matrix of spherical harmonics up to order N for given angles.
sh_to_b(F_nm[, W_weight])	Convert first order SH input signals to B-format.
sht(f, N, azi, colat, SH_type[, weights, Y_nm])	Spherical harmonics transform of f for appropriate point sets.
sht_lstsq(f, N, azi, colat, SH_type[, Y_nm])	Spherical harmonics transform of f as least-squares solution.
sn3d_to_n3d(F_nm[, sh_axis])	Convert SN3D (Schmidt semi-normalized) to N3D (orthonormal) signals.
soundfield_to_b(sig[, W_weight])	Convert soundfield tetraeder mic input signals to B-format by SHT.
spat_filterbank_reconstruction_factor(w_nm, ...)	Reconstruction factor for restoring amplitude/energy preservation.
spherical_hn2(n, z[, derivative])	Spherical Hankel function of the second kind.
src_to_b(sig, src_azi, src_colat)	Get B format signal channels for source in direction azi/colat.
src_to_sh(sig, src_azi, src_zen, N_sph[, ...])	Source signal(s) plane wave encoded in spherical harmonics.
unity_gain(w_n)	Make modal weighting / tapering unit amplitude in steering direction.

spaudiopy.sph.**sh_matrix**(*N*, *azi*, *colat*, *SH_type='complex'*, *weights=None*)

Matrix of spherical harmonics up to order N for given angles.

Computes a matrix of spherical harmonics up to order N for the given angles/grid.

$$\mathbf{Y} = \begin{bmatrix} Y_0^0(\theta[0], \phi[0]) & Y_1^{-1}(\theta[0], \phi[0]) & Y_1^0(\theta[0], \phi[0]) & \dots & Y_N^N(\theta[0], \phi[0]) \\ Y_0^0(\theta[1], \phi[1]) & Y_1^{-1}(\theta[1], \phi[1]) & Y_1^0(\theta[1], \phi[1]) & \dots & Y_N^N(\theta[1], \phi[1]) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ Y_0^0(\theta[Q-1], \phi[Q-1]) & Y_1^{-1}(\theta[Q-1], \phi[Q-1]) & Y_1^0(\theta[Q-1], \phi[Q-1]) & \dots & Y_N^N(\theta[Q-1], \phi[Q-1]) \end{bmatrix}$$

where

$$Y_n^m(\theta, \phi) = \sqrt{\frac{2n+1}{4\pi} \frac{(n-m)!}{(n+m)!}} P_n^m(\cos\theta) e^{im\phi}$$

When using *SH_type='real'*, the real spherical harmonics $Y_{n,m}(\theta, \phi)$ are implemented as a relation to $Y_n^m(\theta, \phi)$.

Parameters

- **N** (*int*) – Maximum SH order.
- **azi** (*(Q,) array_like*) – Azimuth.
- **colat** (*(Q,) array_like*) – Colatitude.
- **SH_type** (*'complex' or 'real' spherical harmonics.*)
- **weights** (*(Q,) array_like, optional*) – Quadrature weights.

Returns **Ymn** (*(Q, (N+1)**2) numpy.ndarray*) – Matrix of spherical harmonics.

Notes

The convention used here is also known as N3D-ACN (for SH_type='real').

spaudiopy.sph.**sht**(*f*, *N*, *azi*, *colat*, *SH_type*, *weights=None*, *Y_nm=None*)

Spherical harmonics transform of f for appropriate point sets.

If f is a QxS matrix then the transform is applied to each column of f, and returns the coefficients at each column of F_nm respectively.

Parameters

- **f** (*(Q, S)*) – The spherical function(S) evaluated at Q directions 'azi/colat'.
- **N** (*int*) – Maximum SH order.
- **azi** (*(Q,) array_like*) – Azimuth.
- **colat** (*(Q,) array_like*) – Colatitude.
- **SH_type** (*'complex' or 'real' spherical harmonics.*)
- **weights** (*(Q,) array_like, optional*) – Quadrature weights.
- **Y_nm** (*(Q, (N+1)**2) numpy.ndarray, optional*) – Matrix of spherical harmonics.

Returns **F_nm** (*((N+1)**2, S) numpy.ndarray*) – Matrix of spherical harmonics coefficients of spherical function(S).

spaudiopy.sph.**sht_lstsq**(*f*, *N*, *azi*, *colat*, *SH_type*, *Y_nm=None*)

Spherical harmonics transform of f as least-squares solution.

If f is a QxS matrix then the transform is applied to each column of f, and returns the coefficients at each column of F_nm respectively.

Parameters

- **f** (*(Q, S)*) – The spherical function(S) evaluated at Q directions 'azi/colat'.
- **N** (*int*) – Maximum SH order.
- **azi** (*(Q,) array_like*) – Azimuth.
- **colat** (*(Q,) array_like*) – Colatitude.
- **SH_type** ('*complex' or 'real' spherical harmonics.*)
- **Y_nm** (*(Q, (N+1)**2) numpy.ndarray, optional*) – Matrix of spherical harmonics.

Returns **F_nm** (*((N+1)**2, S) numpy.ndarray*) – Matrix of spherical harmonics coefficients of spherical function(S).

spaudiopy.sph.**inverse_sht**(*F_nm*, *azi*, *colat*, *SH_type*, *N=None*, *Y_nm=None*)

Perform the inverse spherical harmonics transform.

Parameters

- **F_nm** (*((N+1)**2, S) numpy.ndarray*) – Matrix of spherical harmonics coefficients of spherical function(S).
- **azi** (*(Q,) array_like*) – Azimuth.
- **colat** (*(Q,) array_like*) – Colatitude.
- **SH_type** ('*complex' or 'real' spherical harmonics.*)
- **N** (*int, optional*) – Maximum SH order.
- **Y_nm** (*(Q, (N+1)**2) numpy.ndarray, optional*) – Matrix of spherical harmonics.

Returns **f** (*(Q, S)*) – The spherical function(S) evaluated at Q directions 'azi/colat'.

spaudiopy.sph.**check_cond_sht**(*N*, *azi*, *colat*, *SH_type*, *lim=None*)

Check if condition number for a least-squares SHT(N) is greater 'lim'.

spaudiopy.sph.**calculate_grid_weights**(*azi*, *zen*, *order=None*)

Approximate quadrature weights by pseudo-inverse.

Parameters

- **azi** (*(Q,) array_like*) – Azimuth.
- **zen** (*(Q,) array_like*) – Zenith / Colatitude.
- **order** (*int, optional*) – Supported order N, searched if not provided.

Returns **weights** (*(Q,) array_like*) – Grid / Quadrature weights.

References

Fornberg, B., & Martel, J. M. (2014). On spherical harmonics based numerical quadrature over the surface of a sphere. Advances in Computational Mathematics.

spaudiopy.sph.**n3d_to_sn3d**(*F_nm*, *sh_axis=0*)

Convert N3D (orthonormal) to SN3D (Schmidt semi-normalized) signals.

Parameters

- **F_nm** (*((N_sph+1)**2, S) numpy.ndarray*) – Matrix of spherical harmonics coefficients of spherical function(S).
- **sh_axis** (*int, optional*) – SH axis. The default is 0.

Returns **F_nm** (*((N_sph+1)**2, S) numpy.ndarray*) – Matrix of spherical harmonics coefficients of spherical function(S).

spaudiopy.sph.**sn3d_to_n3d**(*F_nm*, *sh_axis=0*)

Convert SN3D (Schmidt semi-normalized) to N3D (orthonormal) signals.

Parameters

- **F_nm** (*((N_sph+1)**2, S) numpy.ndarray*) – Matrix of spherical harmonics coefficients of spherical function(S).
- **sh_axis** (*int, optional*) – SH axis. The default is 0.

Returns **F_nm** (*((N_sph+1)**2, S) numpy.ndarray*) – Matrix of spherical harmonics coefficients of spherical function(S).

spaudiopy.sph.**platonic_solid**(*shape*)

Return coordinates of shape='tetrahedron' only, yet.

spaudiopy.sph.**sh_to_b**(*F_nm*, *W_weight=None*)

Convert first order SH input signals to B-format.

Parameters

- **F_nm** (*(4, S) numpy.ndarray*) – First order spherical harmonics function coefficients.
- **W-weight** (*float*) – Weight on W channel.

Returns **B_nm** (*(4, S) numpy.ndarray*) – B-format signal(S).

spaudiopy.sph.**b_to_sh**(*B*, *W_weight=None*)

Convert B-format input to first order SH signals.

Parameters

- **B_nm** (*(4, S) numpy.ndarray*) – B-format signal(S).
- **W-weight** (*float*) – Weight on W channel.

Returns **F_nm** (*(4, S) numpy.ndarray*) – First order spherical harmonics function coefficients.

spaudiopy.sph.**soundfield_to_b**(*sig*, *W_weight=None*)

Convert soundfield tetraeder mic input signals to B-format by SHT.

Parameters

- **sig** (*(4, S)*) – Tetraeder mic signals(S).
- **W-weight** (*float*) – Weight on W channel.

Returns **B_nm** (*(4, S) numpy.ndarray*) – B-format signal(S).

spaudiopy.sph.**src_to_b**(*sig*, *src_azi*, *src_colat*)

Get B format signal channels for source in direction azi/colat.

spaudiopy.sph.**src_to_sh**(*sig*, *src_azi*, *src_zen*, *N_sph*, *SH_type='real'*)

Source signal(s) plane wave encoded in spherical harmonics.

Parameters

- **sig** (*(num_src, S) numpy.ndarray*) – Source signal(s).
- **src_azi** (*array_like*)
- **src_zen** (*array_like*)
- **N_sph** (*int*)
- **SH_type** (*'real' (default) or 'complex', optional*)

Returns *((N_sph+1)**2, S) numpy.ndarray* – Source signal(s) in SHD.

Examples

```
src_sig = np.array([1, 0.5], ndmin=2).T * np.random.randn(2, 1000)
N_sph = 3
src_azi = [np.pi/2, -np.pi/3]
src_zen = [np.pi/4, np.pi/2]

sig_nm = spa.sph.src_to_sh(src_sig, src_azi, src_zen, N_sph)

spa.plots.sh_rms_map(sig_nm)
```

spaudiopy.sph.**bandlimited_dirac**(*N*, *d*, *w_n=None*)

Order N spatially bandlimited Dirac pulse at central angle d.

Parameters

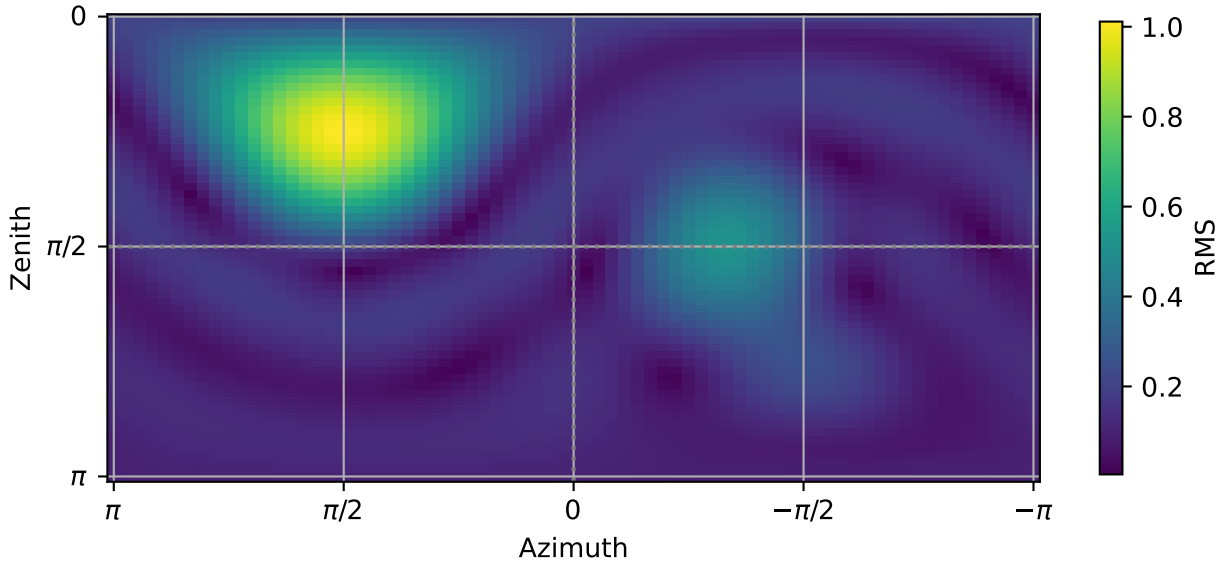
- **N** (*int*) – SH order.
- **d** (*(Q,) array_like*) – Central angle in rad.
- **w_n** (*(N,) array_like, optional. Default is None.*) – Tapering window w_n.

Returns **dirac** (*(Q,) array_like*) – Amplitude at central angle d.

Notes

Normalize with

$$\sum_{}^{N} \frac{2N+1}{4\pi} = \frac{(N+1)^2}{4\pi}.$$

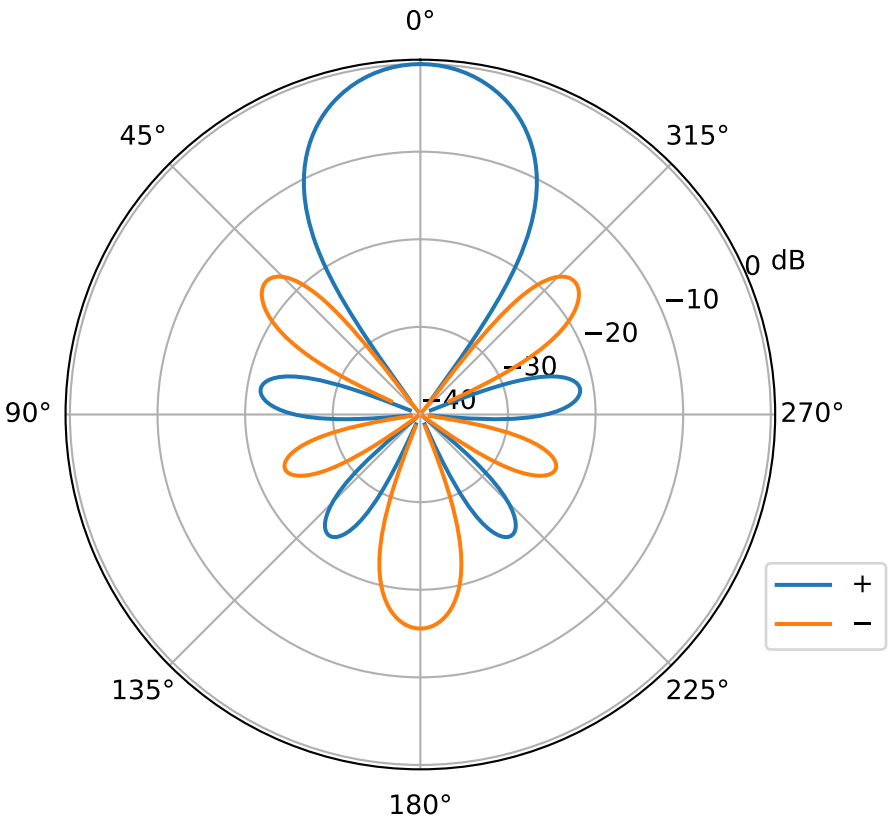

0
π/2
π
Zenith
π
π/2
0
−π/2
−π
Azimuth
1.0
0.8
0.6
0.4
0.2
RMS

References

Rafaely, B. (2015). Fundamentals of Spherical Array Processing. Springer., eq. (1.60).

Examples

```
dirac_azi = np.deg2rad(0)
dirac_colat = np.deg2rad(90)
N = 5

# cross section
azi = np.linspace(0, 2 * np.pi, 720, endpoint=True)

# Bandlimited Dirac pulse
dirac_bandlim = 4 * np.pi / (N + 1) ** 2 * \
                    spa.sph.bandlimited_dirac(N, azi - dirac_azi)

spa.plots.polar(azi, dirac_bandlim)
```



spaudiopy.sph.**max_rE_weights**(*N*)

Return max-rE modal weight coefficients for spherical harmonics order N.

See also:

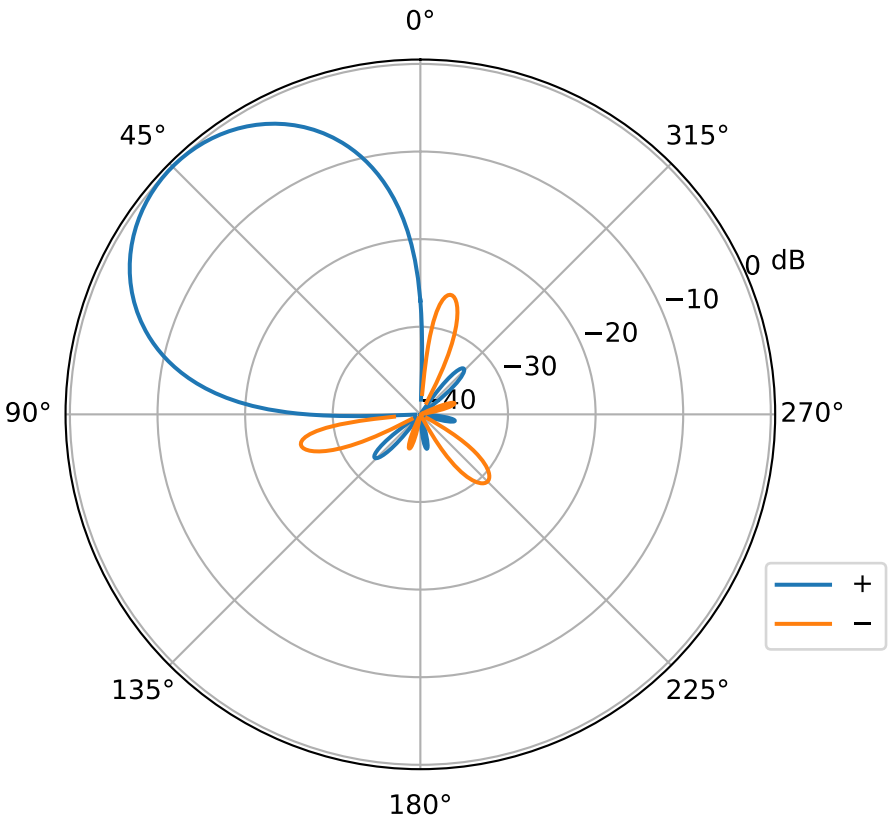
spaudiopy.sph.unity_gain() Unit amplitude compensation.

References

Zotter, F., & Frank, M. (2012). All-Round Ambisonic Panning and Decoding. Journal of Audio Engineering Society, eq. (10).

Examples

```
dirac_azi = np.deg2rad(45)
dirac_colat = np.deg2rad(45)
N = 5

# cross section
azi = np.linspace(0, 2 * np.pi, 720, endpoint=True)

# Bandlimited Dirac pulse, with max r_E tapering window
w_n = spa.sph.max_rE_weights(N)
w_n = spa.sph.unity_gain(w_n)
dirac_tapered = spa.sph.bandlimited_dirac(N, azi - dirac_azi, w_n=w_n)

spa.plots.polar(azi, dirac_tapered)
```



spaudiopy.sph.**r_E**(*p*, *g*)

Calculate r_E vector and magnitude from loudspeaker gains.

Parameters

- **p** (*(Q, 3) numpy.ndarray*) – Q loudspeaker position vectors.
- **g** (*(S, Q) numpy.ndarray*) – Q gain vectors per source S.

Returns

- **rE** (*(S, 3) numpy.ndarray*) – rE vector.
- **rE_mag** (*(S,) array_like*) – rE magnitude (radius).

References

Zotter, F., & Frank, M. (2012). All-Round Ambisonic Panning and Decoding. Journal of Audio Engineering Society, eq. (16).

spaudiopy.sph.**project_on_sphere**(*x*, *y*, *z*)

Little helper that projects x, y, z onto unit sphere.

spaudiopy.sph.**repeat_per_order**(*c*)

Repeat each coefficient in 'c' m times per spherical order n.

Parameters **c** (*(N,) array_like*) – Coefficients up to SH order N.

Returns **c_reshaped** (*((N+1)**2,) array like*) – Reshaped input coefficients.

spaudiopy.sph.**spherical_hn2**(*n*, *z*, *derivative=False*)

Spherical Hankel function of the second kind.

Parameters

- **n** (*int, array_like*) – Order of the spherical Hankel function (n >= 0).
- **z** (*complex or float, array_like*) – Argument of the spherical Hankel function.
- **derivative** (*bool, optional*) – If True, the value of the derivative (rather than the function itself) is returned.

Returns **hn2** (*array_like*)

References

http://mathworld.wolfram.com/SphericalHankelFunctionoftheSecondKind.html

spaudiopy.sph.**mode_strength**(*n*, *kr*, *sphere_type='rigid'*)

Mode strength b_n(kr) for an incident plane wave on sphere.

Parameters

- **n** (*int*) – Degree.
- **kr** (*array_like*) – kr vector, product of wavenumber k and radius r_0.
- **sphere_type** (*'rigid' or 'open'*)

Returns **b_n** (*array_like*) – Mode strength b_n(kr).

References

Rafaely, B. (2015). Fundamentals of Spherical Array Processing. Springer. eq. (4.4) and (4.5).

spaudiopy.sph.**pressure_on_sphere**(*N*, *kr*, *weights=None*)

Calculate the diffuse field pressure frequency response of a spherical scatterer, up to SH order N.

Parameters

- **N** (*int*) – SH order.
- **kr** (*array_like*) – kr vector, product of wavenumber k and radius r_0.
- **weights** (*(N+1,) array_like*) – SH order weights.

Returns **p** (*array_like*) – Pressure p(kr)|N.

References

Ben-Hur, Z., Brinkmann, F., Sheaffer, J., et.al. (2017). Spectral equalization in binaural signals represented by order-truncated spherical harmonics. The Journal of the Acoustical Society of America, eq. (11).

spaudiopy.sph.**binaural_coloration_compensation**(*N*, *f*, *r_0=0.0875*, *w_taper=None*)

Spectral equalization gain G(kr)|N for diffuse field of order N. This filter compensates the high frequency roll of that occurs for order truncated SH signals. It models the human head as a rigid sphere of radius r_0 (e.g. 0.0875m) and compensates the binaural signals.

Parameters

- **N** (*int*) – SH order.
- **f** (*array_like*) – Time-frequency in Hz.
- **r_0** (*radius*) – Rigid sphere radius (approx. human head).
- **w_taper** (*(N+1,) array_like*) – SH order weights for tapering. See e.g. 'process.half_sided_Hann'.

Returns **gain** (*array_like*) – Filter gain(kr).

See also:

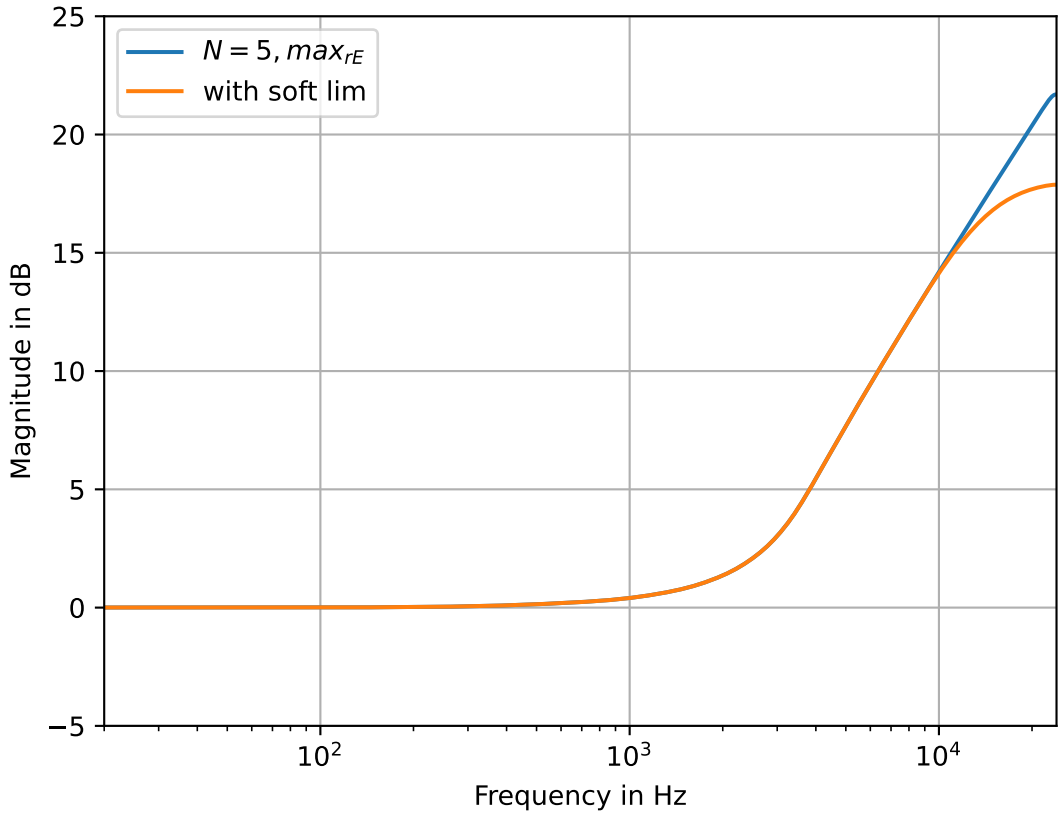
spaudiopy.process.gain_clipping() Limit maximum gain.

References

Hold, C., Gamper, H., Pulkki, V., Raghuvanshi, N., & Tashev, I. J. (2019). Improving Binaural Ambisonics Decoding by Spherical Harmonics Domain Tapering and Coloration Compensation. In IEEE International Conference on Acoustics, Speech and Signal Processing.

Examples

```
fs = 48000
f = np.linspace(0, fs / 2, 1000)
# target spherical harmonics order N (>= 3)
N = 5
# tapering window
w_rE = spa.sph.max_rE_weights(N)

compensation_tapered = spa.sph.binaural_coloration_compensation(
                        N, f, w_taper=w_rE)
compensation_tapered_lim = spa.process.gain_clipping(
                            compensation_tapered,
                            spa.utils.from_db(12))
spa.plots.freq_resp(f, [compensation_tapered,
                        compensation_tapered_lim],
                    ylim=(-5, 25),
                    labels=[r'$N=5, max_{rE}$', 'with soft lim'])
```



spaudiopy.sph.**unity_gain**(*w_n*)

Make modal weighting / tapering unit amplitude in steering direction.

Parameters **w_n** (*(N+1,) array_like*) – Modal weighting factors.

Returns **w_n** (*(N+1,) array_like*) – Modal weighting factors, adjusted for unit amplitude.

Examples

See *spaudiopy.sph.max_rE_weights()*.

spaudiopy.sph.**hypercardioid_modal_weights**(*N_sph*)

Modal weights for beamformer resulting in a hyper-cardioid.

Parameters **N_sph** (*int*) – SH order.

Returns **w_n** (*(N+1,) array_like*) – Modal weighting factors.

Notes

Also called max-DI or normalized PWD.

Examples

```
N = 5
w_n = spa.sph.hypercardioid_modal_weights(N)
w_nm = spa.sph.repeat_per_order(w_n) *              spa.sph.sh_matrix(N, np.pi/4, np.
↪pi/4, 'real')
spa.plots.sh_coeffs(w_nm)
```

spaudiopy.sph.**cardioid_modal_weights**(*N_sph*)

Modal weights for beamformer resulting in a cardioid.

Parameters **N_sph** (*int*) – SH order.

Returns **w_n** (*(N+1,) array_like*) – Modal weighting factors.

Examples

```
N = 5
w_n = spa.sph.cardioid_modal_weights(N)
w_nm = spa.sph.repeat_per_order(w_n) *              spa.sph.sh_matrix(N, np.pi/4, np.
↪pi/4, 'real')
spa.plots.sh_coeffs(w_nm)
```

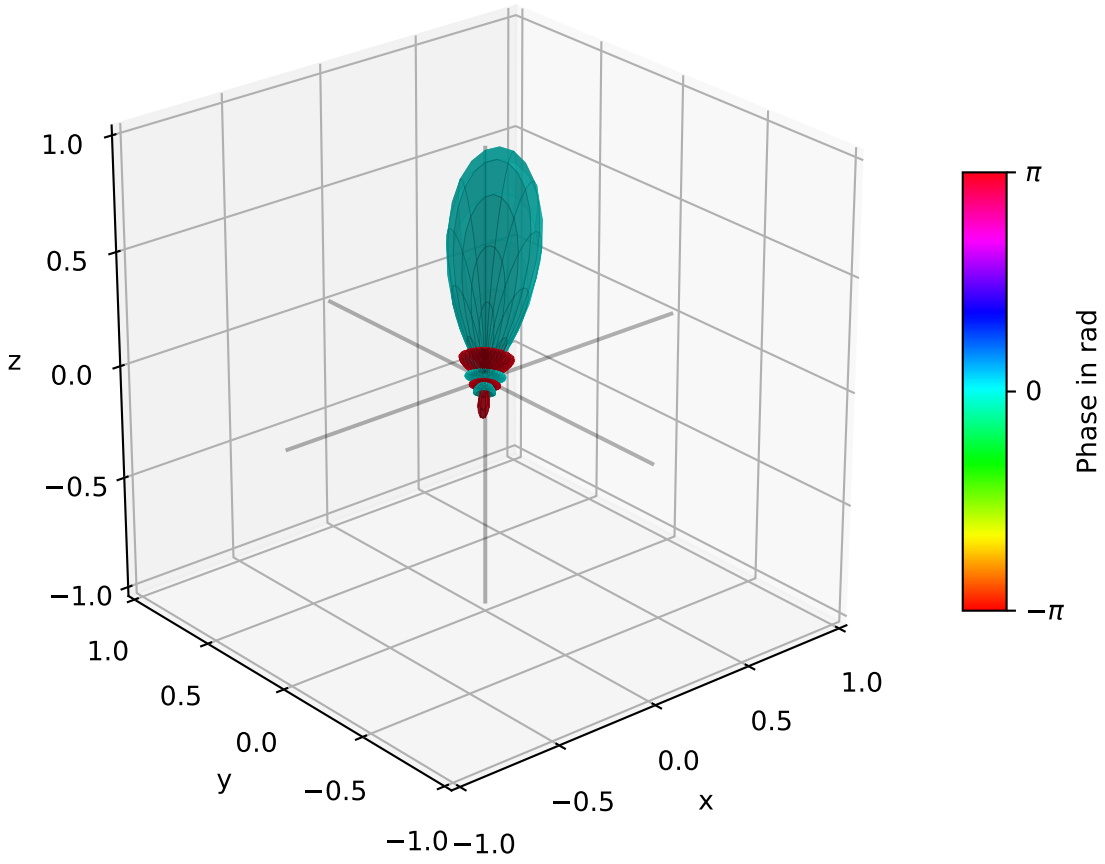
spaudiopy.sph.**maxre_modal_weights**(*N_sph*, *UNITAMP=True*)

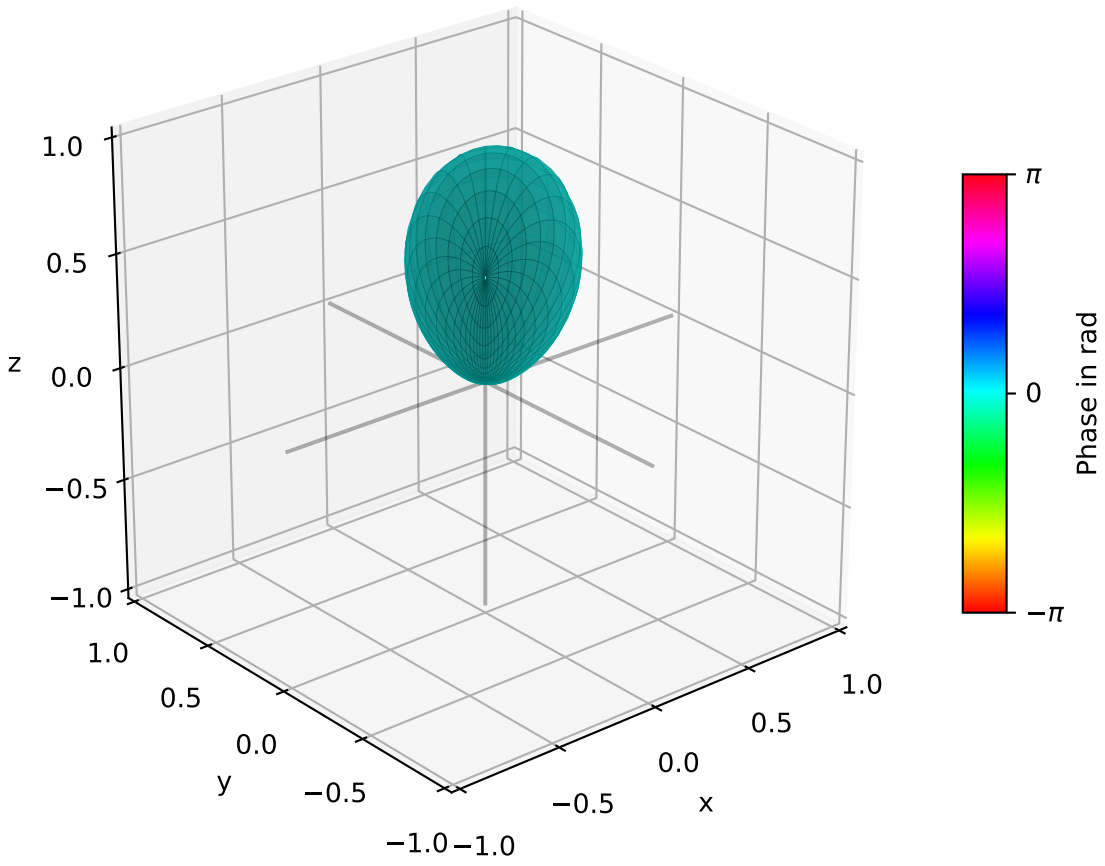
Modal weights for beamformer resulting with max-rE weighting.

Parameters

- **N_sph** (*int*) – SH order.
- **UNITAMP** (*bool, optional (default:True)*)

Returns **w_n** (*(N+1,) array_like*) – Modal weighting factors.


1.0
0.5
0.0
−0.5
−1.0
z
1.0
0.5
0.0
−0.5
−1.0
y
1.0
0.5
0.0
−0.5
−1.0
x
π
0
−π
Phase in rad

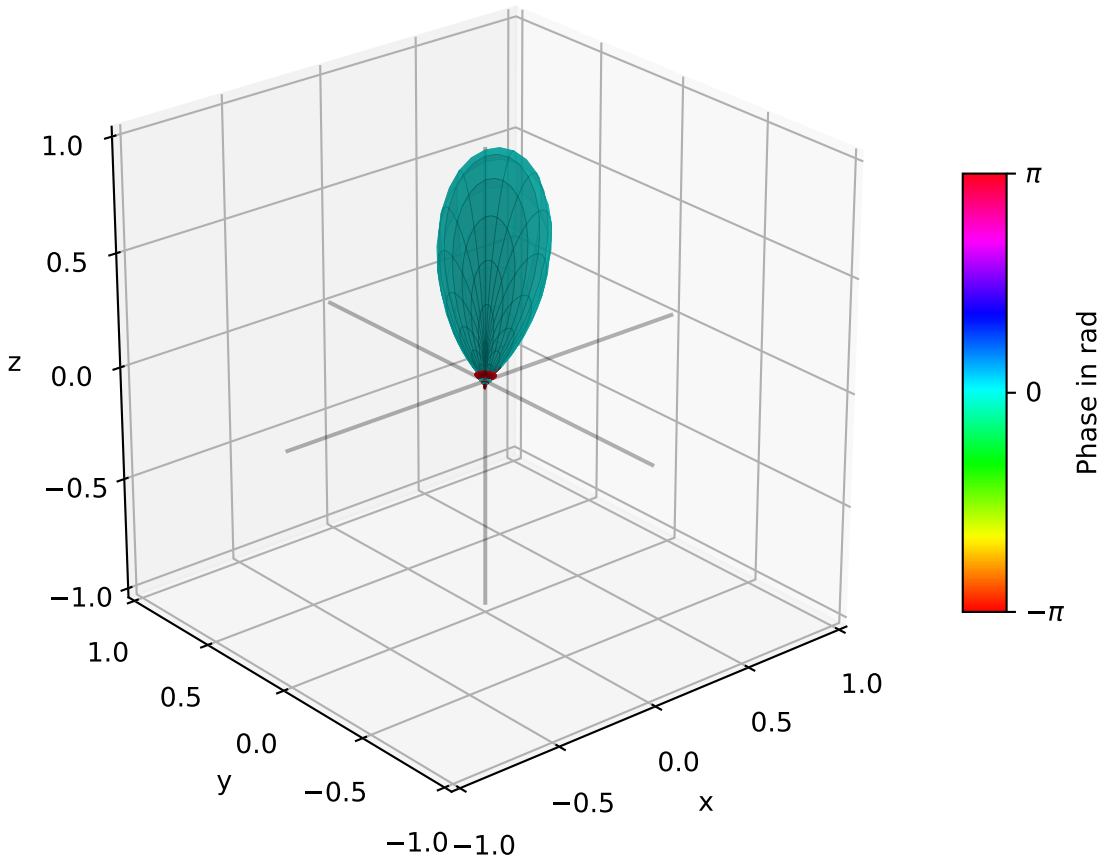

1.0
0.5
0.0
−0.5
−1.0
z
1.0
0.5
0.0
−0.5
−1.0
y
1.0
0.5
0.0
−0.5
−1.0
x
π
0
−π
Phase in rad

Notes

Can be compensated for unit amplitude.

Examples

```
N = 5
w_n = spa.sph.maxre_modal_weights(N)
w_nm = spa.sph.repeat_per_order(w_n) *              spa.sph.sh_matrix(N, np.pi/4, np.
↪pi/4, 'real')
spa.plots.sh_coeffs(w_nm)
```



spaudiopy.sph.**butterworth_modal_weights**(*N_sph*, *k*, *n_c*, *UNITAMP=True*)

Modal weights for spatial butterworth filter / beamformer.

Parameters

- **N_sph** (*int*) – SH order.
- **k** (*int (float)*) – Filter order
- **n_c** (*int (float)*) – Cut-on SH order.
- **UNITAMP** (*bool, optional (default:True)*)

Returns **w_n** (*(N+1,) array_like*) – Modal weighting factors.

Notes

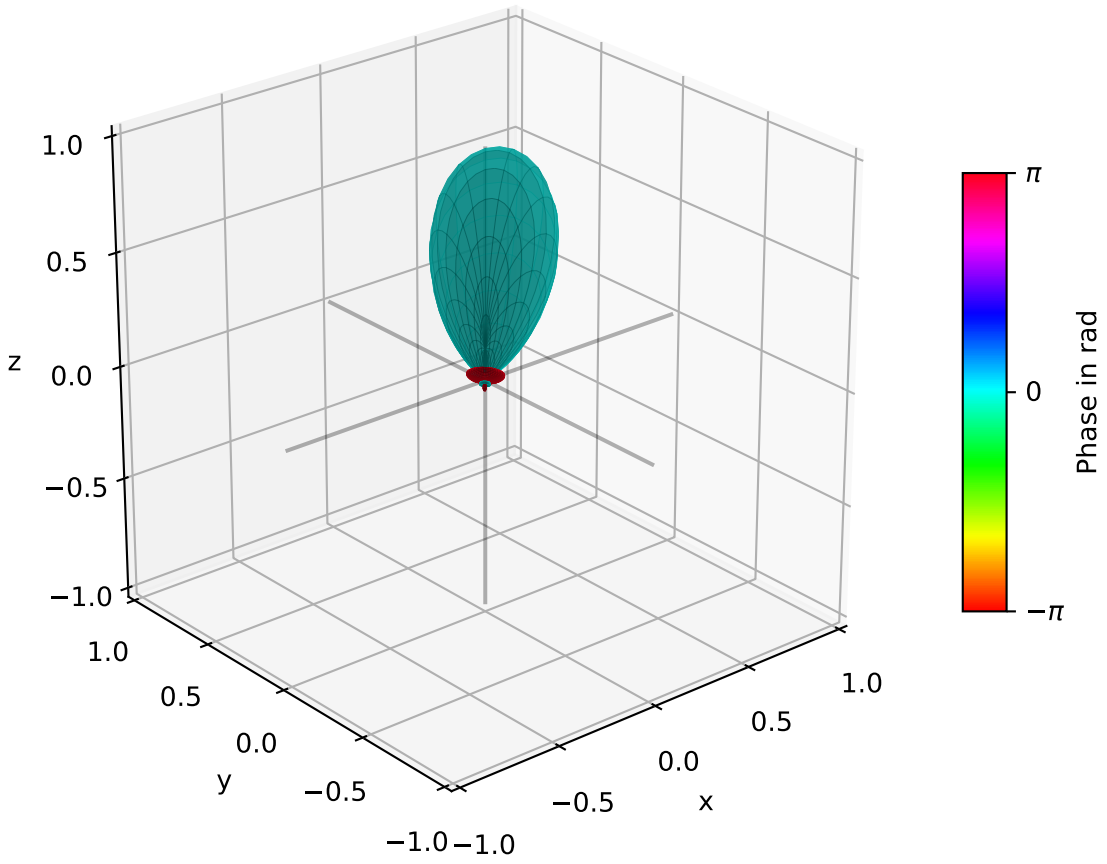
Can be compensated for unit amplitude.

References

Devaraju, B. (2015). Understanding filtering on the sphere.

Examples

```
N = 5
w_n = spa.sph.butterworth_modal_weights(N, 5, 3)
w_nm = spa.sph.repeat_per_order(w_n) *              spa.sph.sh_matrix(N, np.pi/4, np.
↪pi/4, 'real')
spa.plots.sh_coeffs(w_nm)
```



spaudiopy.sph.**spat_filterbank_reconstruction_factor**(*w_nm*, *num_secs*, *mode=None*)

Reconstruction factor for restoring amplitude/energy preservation.

Parameters

- **w_nm** (*((N+1)**2,), array_like*) – SH beam coefficients.

- **num_secs** (*int*) – Number of spatial filters.
- **mode** (*'amplitude' or 'energy'*)

Raises `ValueError` – If mode is not specified.

Returns **beta** (*float*) – Reconstruction factor.

References

Hold, C., Politis, A., Mc Cormack, L., & Pulkki, V. (2021). Spatial Filter Bank Design in the Spherical Harmonic Domain. EUSIPCO 2021.

spaudiopy.sph.**design_spat_filterbank**(*N_sph*, *sec_azi*, *sec_zen*, *c_n*, *SH_type*, *mode*)

Design spatial filter bank analysis and reconstruction matrix.

Parameters

- **N_sph** (*int*) – SH order.
- **sec_azi** (*(J,) array_like*) – Sector azimuth steering directions.
- **sec_zen** (*(J,) array_like*) – Sector zenith/colatitude steering directions.
- **c_n** (*(N,) array_like*) – SH Modal weights, describing (axisymmetric) pattern.
- **SH_type** (*'real' or 'complex'*)
- **mode** (*'perfect' or 'energy'*) – Design achieves perfect reconstruction or energy reconstruction.

Raises `ValueError` – If mode not specified.

Returns

- **A** (*(J, (N+1)**2) numpy.ndarray*) – Analysis matrix.
- **B** (*(J, (N+1)**2) numpy.ndarray*) – Resynthesis matrix.

References

Hold, C., Schlecht, S. J., Politis, A., & Pulkki, V. (2021). Spatial Filter Bank in the Spherical Harmonic Domain : Reconstruction and Application. WASPAA 2021.

Examples

```
N_sph = 3
sec_dirs = spa.utils.cart2sph(*spa.grids.load_t_design(2*N_sph).T)
c_n = spa.sph.maxre_modal_weights(N_sph)
[A, B] = spa.sph.design_spat_filterbank(N_sph, sec_dirs[0], sec_dirs[1],
                                        c_n, 'real', 'perfect')
# diffuse input SH signal
in_nm = np.random.randn((N_sph+1)**2, 1000)
# Sector signals (Analysis)
s_sec = A @ in_nm
# Reconstruction to SH domain
out_nm = B.conj().T @ s_sec
```

(continues on next page)

(continued from previous page)

```
# Test perfect reconstruction
print(spa.utils.test_diff(in_nm, out_nm))
```

2.4 spaudiopy.decoder

Loudspeaker decoders.

```
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['axes.grid'] = True

import spaudiopy as spa

# Loudspeaker Setup
ls_dirs = np.array([[-80, -45, 0, 45, 80, -60, -30, 30, 60],
                    [0, 0, 0, 0, 0, 60, 60, 60, 60]])
ls_x, ls_y, ls_z = spa.utils.sph2cart(spa.utils.deg2rad(ls_dirs[0, :]),
                                      spa.utils.deg2rad(90 - ls_dirs[1, :]))
```

Functions

allrad(F_nm, hull[, N_sph, jobs_count])	Loudspeaker signals of All-Round Ambisonic Decoder.
allrad2(F_nm, hull[, N_sph, jobs_count])	Loudspeaker signals of All-Round Ambisonic Decoder 2.
allrap(src, hull[, N_sph, jobs_count])	Loudspeaker gains for All-Round Ambisonic Panning.
allrap2(src, hull[, N_sph, jobs_count])	Loudspeaker gains for All-Round Ambisonic Panning 2.
apply_blacklist(hull[, blacklist])	Specify a blacklist to exclude simplices from valid simplices.
calculate_barycenter(hull)	Barycenter of hull object.
calculate_centroids(hull)	Calculate centroid for each simplex.
calculate_face_areas(hull)	Calculate area for each simplex.
calculate_face_normals(hull[, eps, normalize])	Calculate outwards pointing normal for each simplex.
calculate_vertex_normals(hull[, normalize])	Calculate normal for each vertex from simplices normals.
characteristic_ambisonic_order(hull)	Find the characteristic order for specified loudspeaker layout.
check_aperture(hull[, aperture_limit, ...])	Return valid simplices, where the aperture form the listener is small.
check_listener_inside(hull[, listener_position])	Return valid simplices for which the listener is inside the hull.
check_normals(hull[, normal_limit, ...])	Return valid simplices that point towards listener.
check_opening(hull[, opening_limit])	Return valid simplices with all opening angles within simplex > limit.
epad(F_nm, hull[, N_sph])	Loudspeaker signals of Energy-Preserving Ambisonic Decoder.
find_imaginary_loudspeaker(hull)	Find imaginary loudspeaker coordinates for smoother hull.
get_hull(x, y, z)	Wrapper for scipy.spatial.ConvexHull.
mad(F_nm, hull[, N_sph])	Loudspeaker signals of Mode-Matching Ambisonic Decoder.
nearest_loudspeaker(src, hull)	Loudspeaker gains for nearest loudspeaker selection (NLS) decoding, based on euclidean distance.
sad(F_nm, hull[, N_sph])	Loudspeaker signals of Sampling Ambisonic Decoder.
sh2bin(sig_nm, hrirs_nm)	Spherical Harmonic Domain signals to binaural renderer.
sort_vertices(simplices)	Start the simplices with smallest vertex entry.
vbap(src, hull[, norm, valid_simplices, ...])	Loudspeaker gains for Vector Base Amplitude Panning decoding.
vbip(src, hull[, norm, valid_simplices, ...])	Loudspeaker gains for Vector Base Intensity Panning decoding.

Classes

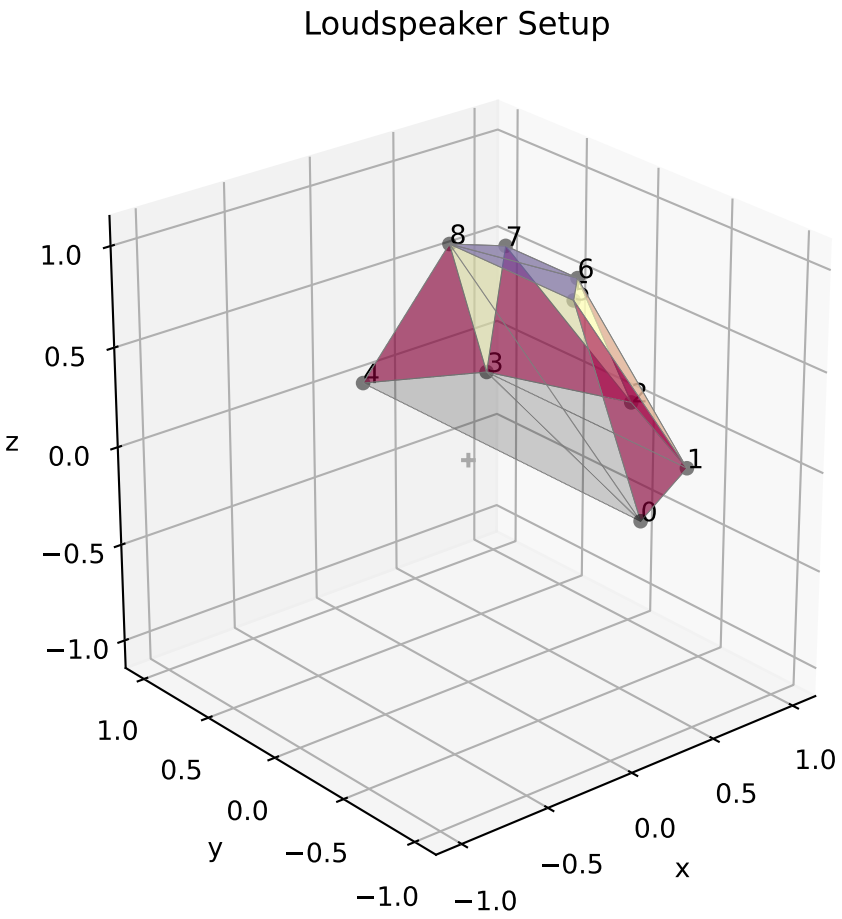
LoudspeakerSetup(x, y, z[, listener_position])	Creates a 'hull' object containing all information for further decoding.

class spaudiopy.decoder.**LoudspeakerSetup**(*x*, *y*, *z*, *listener_position=None*)

Bases: object

Creates a 'hull' object containing all information for further decoding.

```
ls_setup = spa.decoder.LoudspeakerSetup(ls_x, ls_y, ls_z)
ls_setup.pop_triangles(normal_limit=85, aperture_limit=90,
                       opening_limit=150)
ls_setup.show()
```



__init__(*x*, *y*, *z*, *listener_position=None*)

Parameters

- **x** (*array_like*)
- **y** (*array_like*)
- **z** (*array_like*)

- **listener_position** (*(3,), cartesian, optional*) – Offset, will be substracted from the loudspeaker positions.

`classmethod from_sph`(*azi*, *colat*, *r=1*, *listener_position=None*)

Alternative constructor, using spherical coordinates in rad.

Parameters

- **azi** (*array_like, spherical*)
- **colat** (*array_like, spherical*)
- **r** (*array_like, spherical*)
- **listener_position** (*(azi, colat, r), spherical, optional*) – Offset, will be substracted from the loudspeaker positions.

`is_simplex_valid`(*simplex*)

Tests if simplex is in valid simplices (independent of orientation).

`pop_triangles`(*normal_limit=85*, *aperture_limit=None*, *opening_limit=None*, *blacklist=None*)

Refine triangulation by removing them from valid simplices. Bypass by passing 'None'.

Parameters

- **normal_limit** (*float, optional*)
- **aperture_limit** (*float, optional*)
- **opening_limit** (*float, optional*)
- **blacklist** (*list, optional*)

`get_characteristic_order`()

Characteristic Ambisonics order.

`ambisonics_setup`(*N_kernel=50*, *update_hull=False*, *imaginary_ls=None*)

Prepare loudspeaker hull for ambisonic rendering. Sets the *kernel_hull* as an n-design of twice *N_kernel*, and updates the ambisonic hull with an additional imaginary loudspeaker, if desired.

Parameters

- **N_kernel** (*int, optional*)
- **update_hull** (*bool, optional*)
- **imaginary_ls** (*(L, 3), cartesian, optional*) – Imaginary loudspeaker positions, if set to 'None' calls 'find_imaginary_loudspeaker()' for 'update_hull'.

Examples

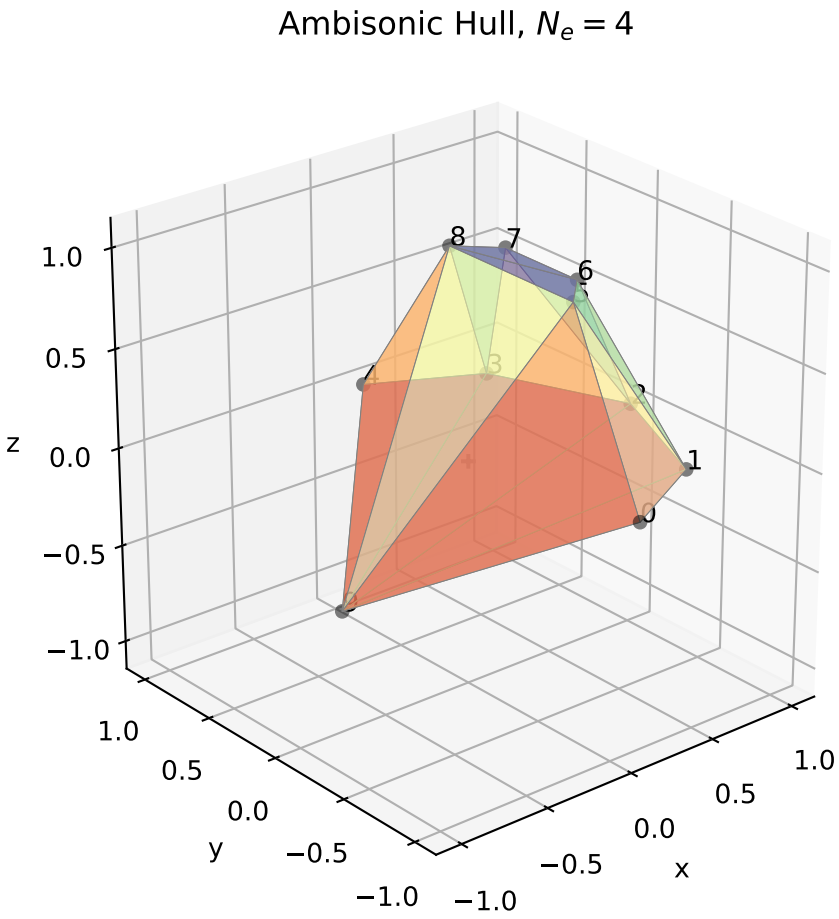
```
ls_setup.ambisonics_setup(update_hull=True)
N_e = ls_setup.characteristic_order
ls_setup.ambisonics_hull.show(title=f"Ambisonic Hull, $N_e={N_e}$")
```

`binauralize`(*ls_signals*, *fs*, *orientation=(0, 0)*, *hrirs=None*)

Create binaural signals that the loudspeaker signals produce on this setup (no delays).

Parameters

- **ls_signals** (*(L, S) np.ndarray*) – Loudspeaker signals.
- **fs** (*int*)


Ambisonic Hull, $N_e = 4$
8
7
6
1
z
1.0
0.5
0.0
−0.5
−1.0
y
x

- **orientation** (*(azi, colat) tuple, optional*) – Listener orientation offset (azimuth, colatitude) in rad.
- **hrirs** (*sig.HRIRs, optional*)

Returns

- **l_sig** (*array_like*)
- **r_sig** (*array_like*)

loudspeaker_signals(*ls_gains*, *sig_in=None*)

Render loudspeaker signals.

Parameters

- **ls_gains** (*(S, L) np.ndarray*)
- **sig_in** (*(S,) array like, optional*)

Returns **sig_out** (*(L, S) np.ndarray*)

show(*title='Loudspeaker Setup'*, ***kwargs*)

Plot hull object, calls plots.hull().

spaudiopy.decoder.**get_hull**(*x*, *y*, *z*)

Wrapper for scipy.spatial.ConvexHull.

spaudiopy.decoder.**calculate_centroids**(*hull*)

Calculate centroid for each simplex.

spaudiopy.decoder.**calculate_face_areas**(*hull*)

Calculate area for each simplex.

spaudiopy.decoder.**calculate_face_normals**(*hull*, *eps=1e-05*, *normalize=False*)

Calculate outwards pointing normal for each simplex.

spaudiopy.decoder.**calculate_vertex_normals**(*hull*, *normalize=False*)

Calculate normal for each vertex from simplices normals.

spaudiopy.decoder.**calculate_barycenter**(*hull*)

Barycenter of hull object.

spaudiopy.decoder.**check_listener_inside**(*hull*, *listener_position=None*)

Return valid simplices for which the listener is inside the hull.

spaudiopy.decoder.**check_normals**(*hull*, *normal_limit=85*, *listener_position=None*)

Return valid simplices that point towards listener.

spaudiopy.decoder.**check_aperture**(*hull*, *aperture_limit=90*, *listener_position=None*)

Return valid simplices, where the aperture form the listener is small.

spaudiopy.decoder.**check_opening**(*hull*, *opening_limit=135*)

Return valid simplices with all opening angles within simplex > limit.

spaudiopy.decoder.**apply_blacklist**(*hull*, *blacklist=None*)

Specify a blacklist to exclude simplices from valid simplices.

spaudiopy.decoder.**sort_vertices**(*simplices*)

Start the simplices with smallest vertex entry.

spaudiopy.decoder.**find_imaginary_loudspeaker**(*hull*)

Find imaginary loudspeaker coordinates for smoother hull.

References

Zotter, F., & Frank, M. (2012). All-Round Ambisonic Panning and Decoding. Journal of Audio Engineering Society, Sec. 1.1.

spaudiopy.decoder.**vbap**(*src*, *hull*, *norm=2*, *valid_simplices=None*, *retain_outside=False*, *jobs_count=1*)

Loudspeaker gains for Vector Base Amplitude Panning decoding.

Parameters

- **src** (*(n, 3) numpy.ndarray*) – Cartesian coordinates of n sources to be rendered.
- **hull** (*LoudspeakerSetup*)
- **norm** (*non-zero int, float*) – Gain normalization norm, e.g. 1: anechoic, 2: reverberant
- **valid_simplices** (*(nsimplex, 3) numpy.ndarray*) – Valid simplices employed for rendering, defaults hull.valid_simplices.
- **retain_outside** (*bool, optional*) – Render on the 'ambisonic hull' to fade out amplitude.
- **jobs_count** (*int or None, optional*) – Number of parallel jobs, 'None' employs 'cpu_count'.

Returns **gains** (*(n, L) numpy.ndarray*) – Panning gains for L loudspeakers to render n sources.

References

Pulkki, V. (1997). Virtual Sound Source Positioning Using Vector Base Amplitude Panning. AES, 144(5), 357–360.

Examples

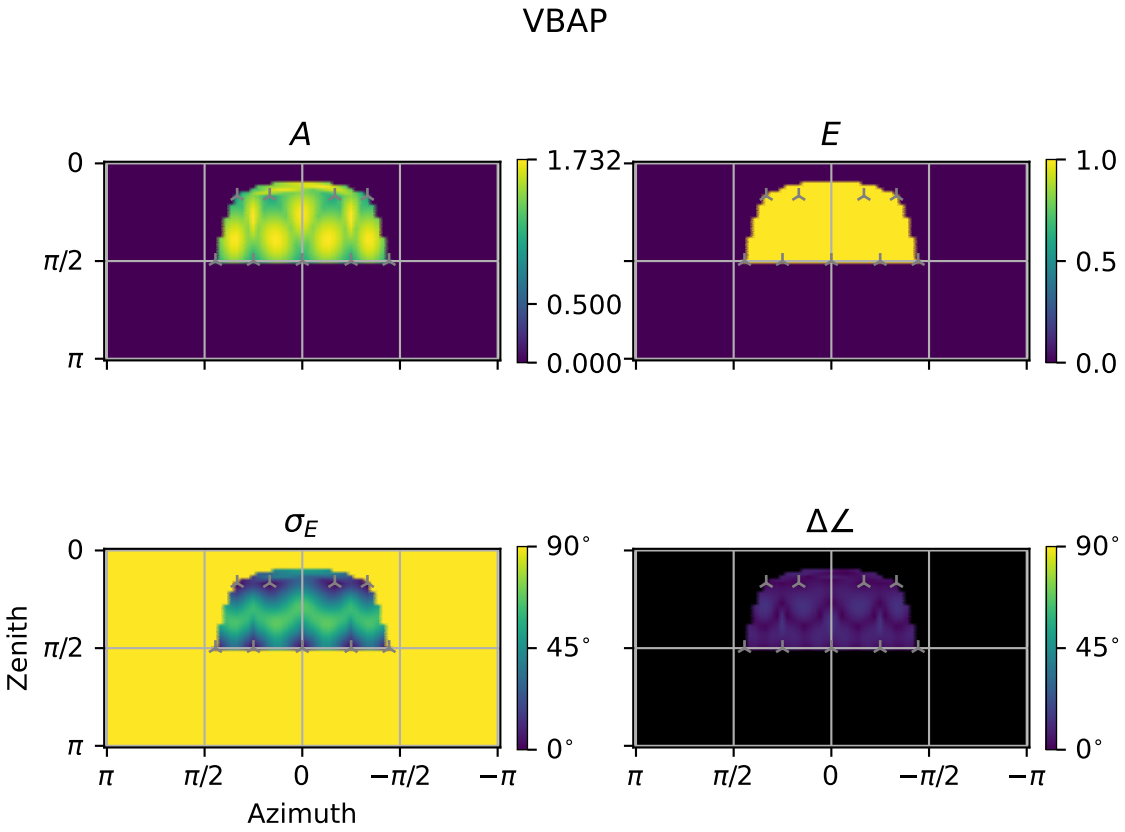
```
ls_setup = spa.decoder.LoudspeakerSetup(ls_x, ls_y, ls_z)
ls_setup.pop_triangles(normal_limit=85, aperture_limit=90,
                       opening_limit=150)

spa.plots.decoder_performance(ls_setup, 'VBAP')

ls_setup.ambisonics_setup(update_hull=True)
spa.plots.decoder_performance(ls_setup, 'VBAP', retain_outside=True)
plt.suptitle('VBAP with imaginary loudspeaker')
```

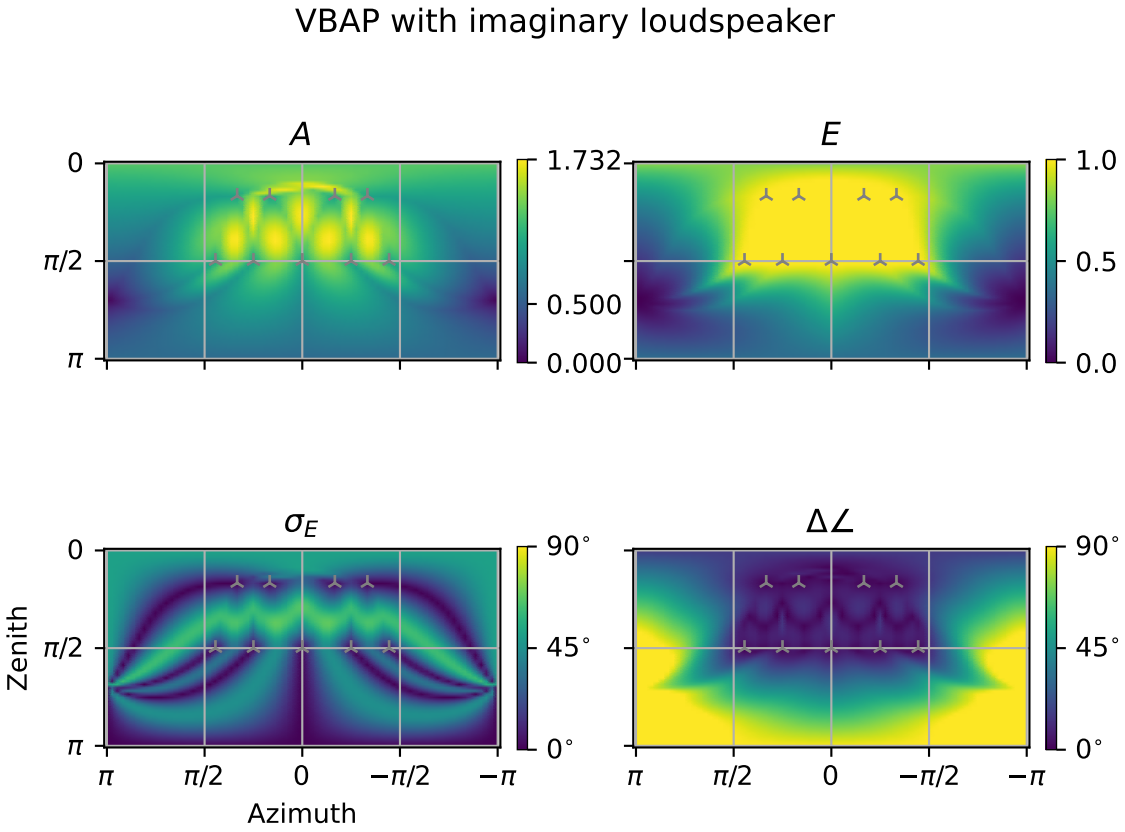
spaudiopy.decoder.**vbip**(*src*, *hull*, *norm=2*, *valid_simplices=None*, *retain_outside=False*, *jobs_count=1*)

Loudspeaker gains for Vector Base Intensity Panning decoding.

Parameters

- **src** (*(n, 3) numpy.ndarray*) – Cartesian coordinates of n sources to be rendered.
- **hull** (*LoudspeakerSetup*)
- **norm** (*non-zero int, float*) – Gain normalization norm, e.g. 1: anechoic, 2: reverberant
- **valid_simplices** (*(nsimplex, 3) numpy.ndarray*) – Valid simplices employed for rendering, defaults hull.valid_simplices.
- **retain_outside** (*bool, optional*) – Render on the 'ambisonic hull', amplitude will not fade out with VBIP.
- **jobs_count** (*int or None, optional*) – Number of parallel jobs, 'None' employs 'cpu_count'.

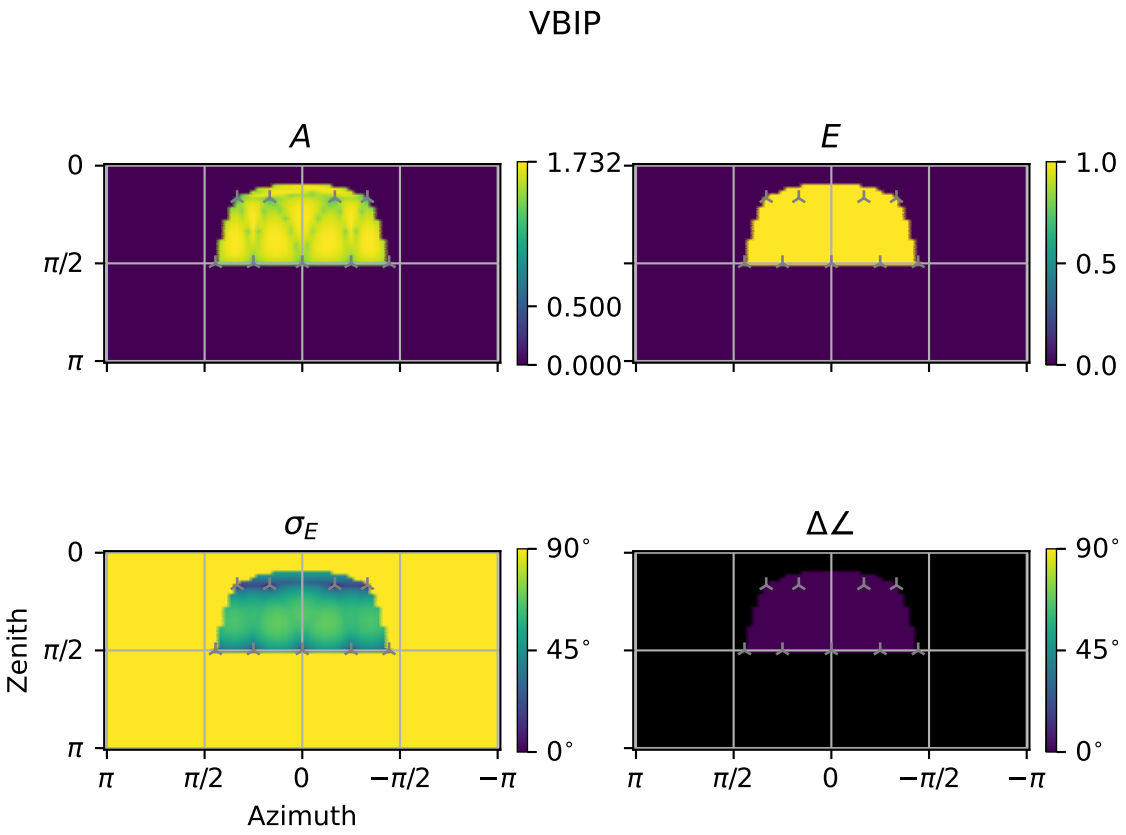

VBAP
A
E
σE
ΔL
0
π/2
π
1.732
0.500
0.000
1.0
0.5
0.0
90°
45°
0°
Zenith
Azimuth
π
π/2
0
−π/2
−π


VBAP with imaginary loudspeaker
A
E
σ_E
$\Delta\angle$
0
π/2
π
1.732
0.500
0.000
1.0
0.5
0.0
90°
45°
0°
Zenith
Azimuth
π
π/2
0
−π/2
−π

Returns **gains** (*(n, L) numpy.ndarray*) – Panning gains for L loudspeakers to render n sources.

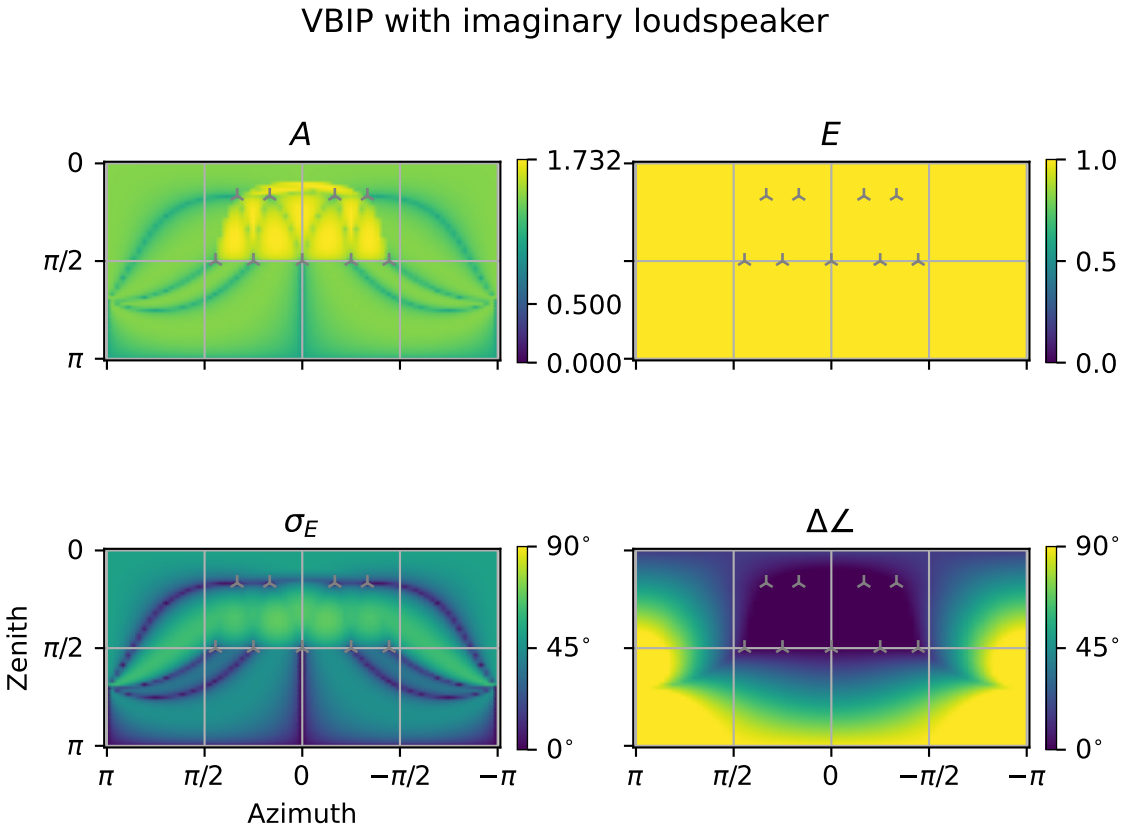
Examples

```
ls_setup = spa.decoder.LoudspeakerSetup(ls_x, ls_y, ls_z)
ls_setup.pop_triangles(normal_limit=85, aperture_limit=90,
                       opening_limit=150)

spa.plots.decoder_performance(ls_setup, 'VBIP')

ls_setup.ambisonics_setup(update_hull=True)
spa.plots.decoder_performance(ls_setup, 'VBIP', retain_outside=True)
plt.suptitle('VBIP with imaginary loudspeaker')
```



spaudiopy.decoder.**characteristic_ambisonic_order**(*hull*)

Find the characteristic order for specified loudspeaker layout.


VBIP with imaginary loudspeaker
A
E
σ_E
$\Delta\angle$
0
π/2
π
1.732
0.500
0.000
1.0
0.5
0.0
90°
45°
0°
Zenith
π
π/2
0
−π/2
−π
Azimuth

References

Zotter, F., & Frank, M. (2012). All-Round Ambisonic Panning and Decoding. Journal of Audio Engineering Society, Sec. 7.

spaudiopy.decoder.**allrap**(*src*, *hull*, *N_sph=None*, *jobs_count=1*)

Loudspeaker gains for All-Round Ambisonic Panning.

Parameters

- **src** (*(N, 3)*) – Cartesian coordinates of N sources to be rendered.
- **hull** (*LoudspeakerSetup*)
- **N_sph** (*int*) – Decoding order, defaults to hull.characteristic_order.
- **jobs_count** (*int or None, optional*) – Number of parallel jobs, 'None' employs 'cpu_count'.

Returns **gains** (*(N, L) numpy.ndarray*) – Panning gains for L loudspeakers to render N sources.

References

Zotter, F., & Frank, M. (2012). All-Round Ambisonic Panning and Decoding. Journal of Audio Engineering Society, Sec. 4.

Examples

```
ls_setup = spa.decoder.LoudspeakerSetup(ls_x, ls_y, ls_z)
ls_setup.pop_triangles(normal_limit=85, aperture_limit=90,
                       opening_limit=150)
ls_setup.ambisonics_setup(update_hull=True)

spa.plots.decoder_performance(ls_setup, 'ALLRAP')
```

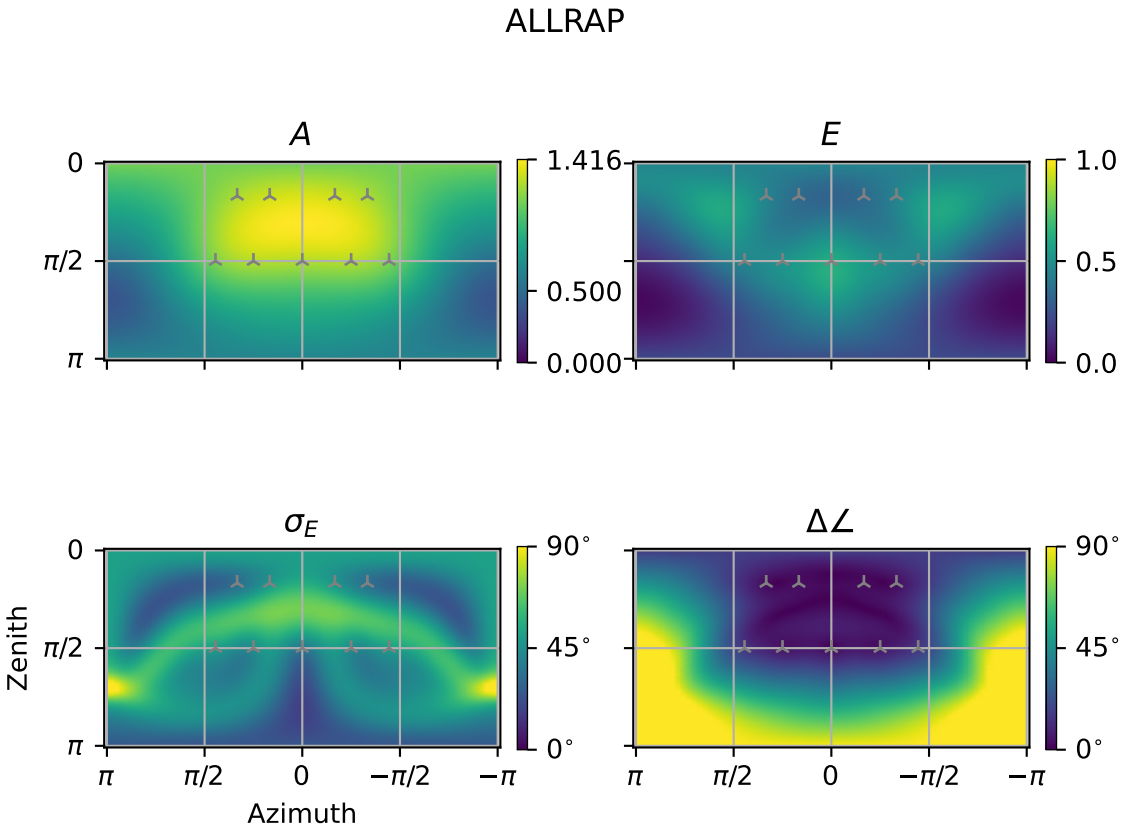
spaudiopy.decoder.**allrap2**(*src*, *hull*, *N_sph=None*, *jobs_count=1*)

Loudspeaker gains for All-Round Ambisonic Panning 2.

Parameters

- **src** (*(N, 3)*) – Cartesian coordinates of N sources to be rendered.
- **hull** (*LoudspeakerSetup*)
- **N_sph** (*int*) – Decoding order, defaults to hull.characteristic_order.
- **jobs_count** (*int or None, optional*) – Number of parallel jobs, 'None' employs 'cpu_count'.

Returns **gains** (*(N, L) numpy.ndarray*) – Panning gains for L loudspeakers to render N sources.

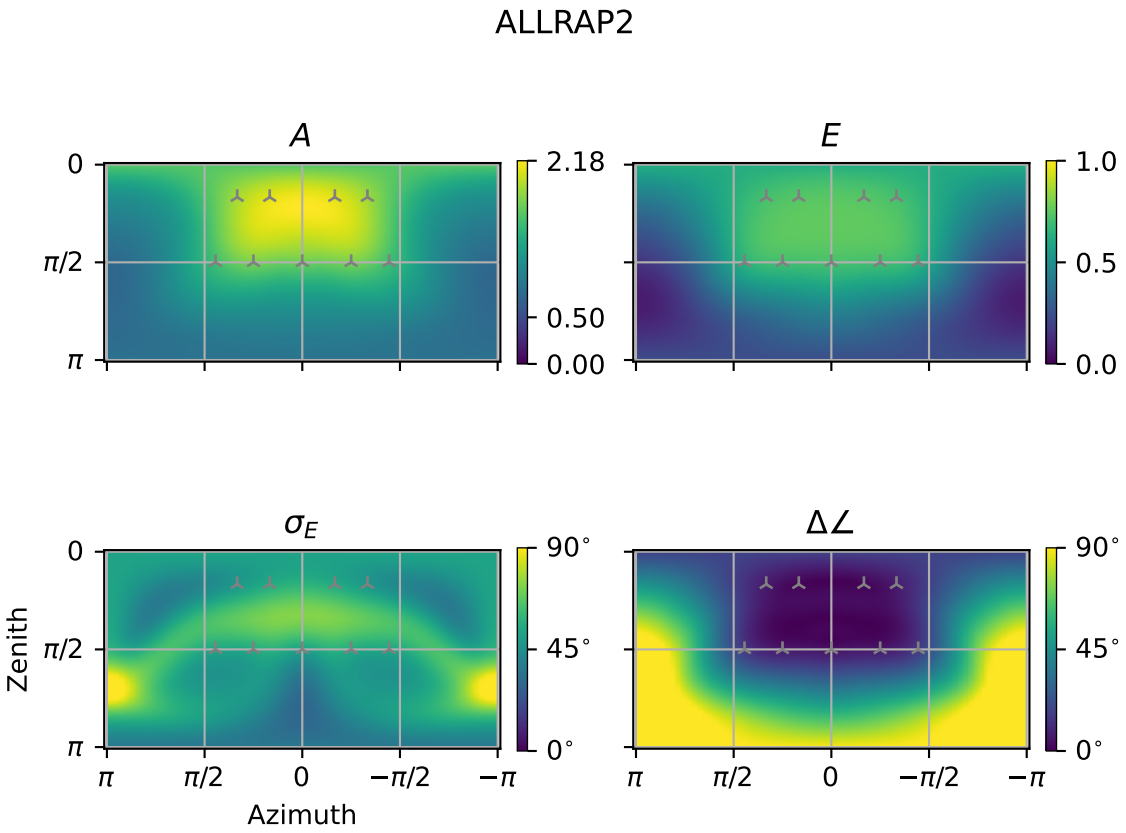

ALLRAP
A
E
σ_E
$\Delta\angle$
0
π/2
π
1.416
0.500
0.000
1.0
0.5
0.0
90°
45°
0°
Zenith
π
π/2
0
−π/2
−π
Azimuth

References

Zotter, F., & Frank, M. (2018). Ambisonic decoding with panning-invariant loudness on small layouts (AllRAD2). In 144th AES Convention.

Examples

```
ls_setup = spa.decoder.LoudspeakerSetup(ls_x, ls_y, ls_z)
ls_setup.pop_triangles(normal_limit=85, aperture_limit=90,
                       opening_limit=150)
ls_setup.ambisonics_setup(update_hull=True)

spa.plots.decoder_performance(ls_setup, 'ALLRAP2')
```



spaudiopy.decoder.**allrad**(*F_nm*, *hull*, *N_sph=None*, *jobs_count=1*)

Loudspeaker signals of All-Round Ambisonic Decoder.

Parameters

- **F_nm** (*((N_sph+1)**2, S) numpy.ndarray*) – Matrix of spherical harmonics coefficients of spherical function(S).
- **hull** (*LoudspeakerSetup*)
- **N_sph** (*int*) – Decoding order.

- **jobs_count** (*int or None, optional*) – Number of parallel jobs, 'None' employs 'cpu_count'.

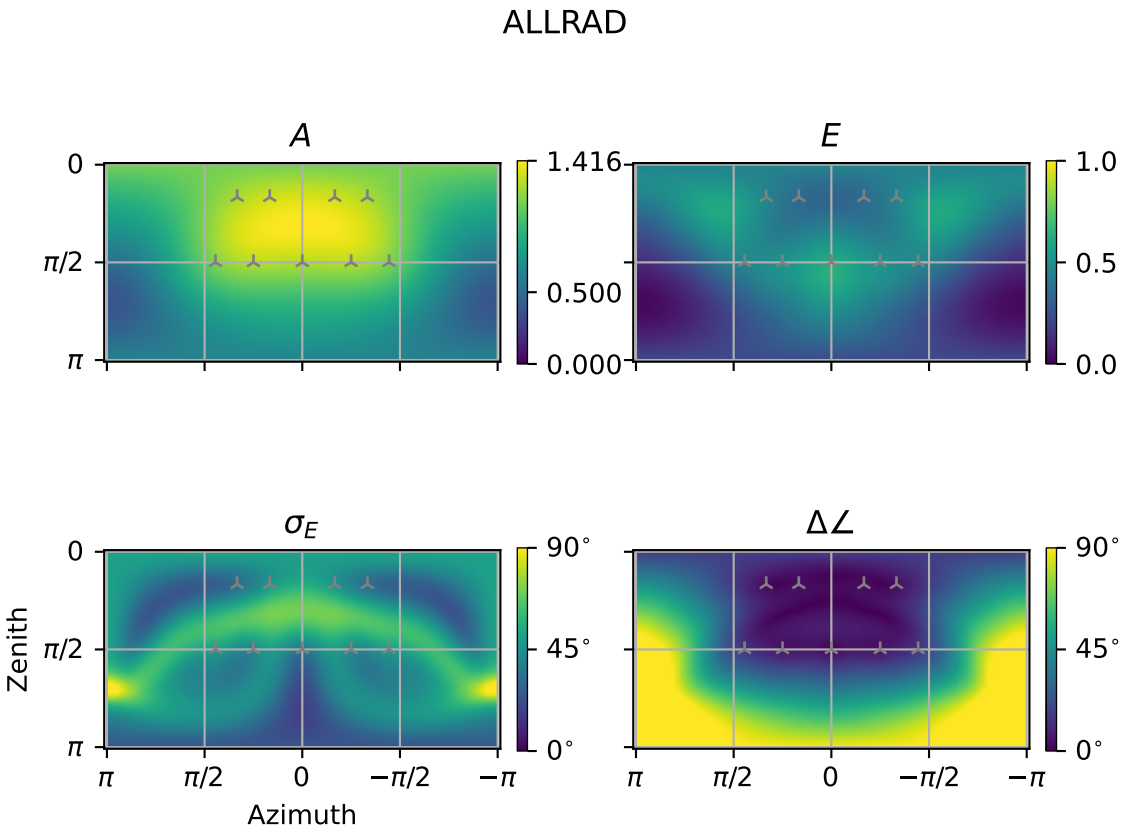
Returns **ls_sig** (*(L, S) numpy.ndarray*) – Loudspeaker L output signal S.

References

Zotter, F., & Frank, M. (2012). All-Round Ambisonic Panning and Decoding. Journal of Audio Engineering Society, Sec. 6.

Examples

```
ls_setup = spa.decoder.LoudspeakerSetup(ls_x, ls_y, ls_z)
ls_setup.pop_triangles(normal_limit=85, aperture_limit=90,
                       opening_limit=150)
ls_setup.ambisonics_setup(update_hull=True)

spa.plots.decoder_performance(ls_setup, 'ALLRAD')
```



spaudiopy.decoder.**allrad2**(*F_nm*, *hull*, *N_sph=None*, *jobs_count=1*)

Loudspeaker signals of All-Round Ambisonic Decoder 2.

Parameters

- **F_nm** (*((N_sph+1)**2, S) numpy.ndarray*) – Matrix of spherical harmonics coefficients of spherical function(S).
- **hull** (*LoudspeakerSetup*)
- **N_sph** (*int*) – Decoding order, defaults to hull.characteristic_order.
- **jobs_count** (*int or None, optional*) – Number of parallel jobs, 'None' employs 'cpu_count'.

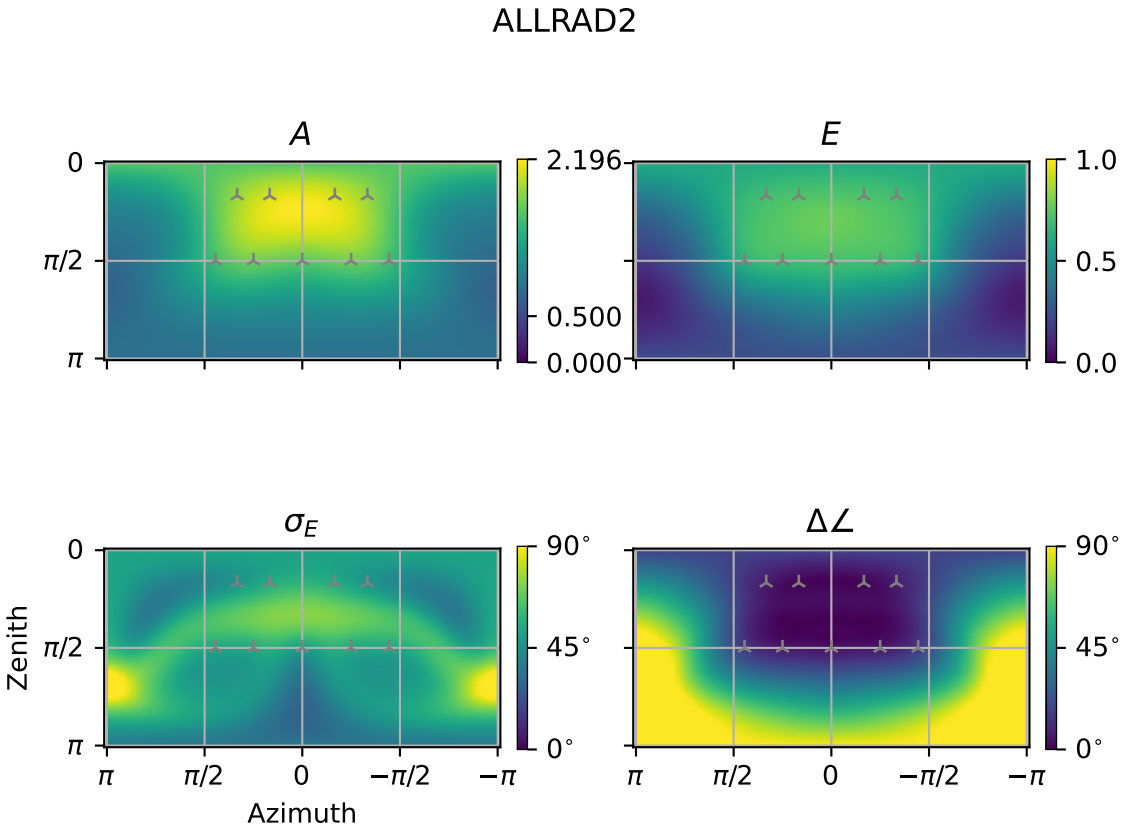
Returns **ls_sig** (*(L, S) numpy.ndarray*) – Loudspeaker L output signal S.

References

Zotter, F., & Frank, M. (2018). Ambisonic decoding with panning-invariant loudness on small layouts (AllRAD2). In 144th AES Convention.

Examples

```
ls_setup = spa.decoder.LoudspeakerSetup(ls_x, ls_y, ls_z)
ls_setup.pop_triangles(normal_limit=85, aperture_limit=90,
                       opening_limit=150)
ls_setup.ambisonics_setup(update_hull=True)

spa.plots.decoder_performance(ls_setup, 'ALLRAD2')
```



spaudiopy.decoder.**sad**(*F_nm*, *hull*, *N_sph=None*)

Loudspeaker signals of Sampling Ambisonic Decoder.

Parameters

- **F_nm** (*((N_sph+1)**2, S) numpy.ndarray*) – Matrix of spherical harmonics coefficients of spherical function(S).
- **hull** (*LoudspeakerSetup*)
- **N_sph** (*int*) – Decoding order, defaults to hull.characteristic_order.

Returns **ls_sig** (*(L, S) numpy.ndarray*) – Loudspeaker L output signal S.

References

ch. 4.9.1, Zotter, F., & Frank, M. (2019). Ambisonics. Springer Topics in Signal Processing.

Examples

```
ls_setup = spa.decoder.LoudspeakerSetup(ls_x, ls_y, ls_z)
ls_setup.pop_triangles(normal_limit=85, aperture_limit=90,
                       opening_limit=150)

spa.plots.decoder_performance(ls_setup, 'SAD')
```

spaudiopy.decoder.**mad**(*F_nm*, *hull*, *N_sph=None*)

Loudspeaker signals of Mode-Matching Ambisonic Decoder.

Parameters

- **F_nm** (*((N_sph+1)**2, S) numpy.ndarray*) – Matrix of spherical harmonics coefficients of spherical function(S).
- **hull** (*LoudspeakerSetup*)
- **N_sph** (*int*) – Decoding order, defaults to hull.characteristic_order.

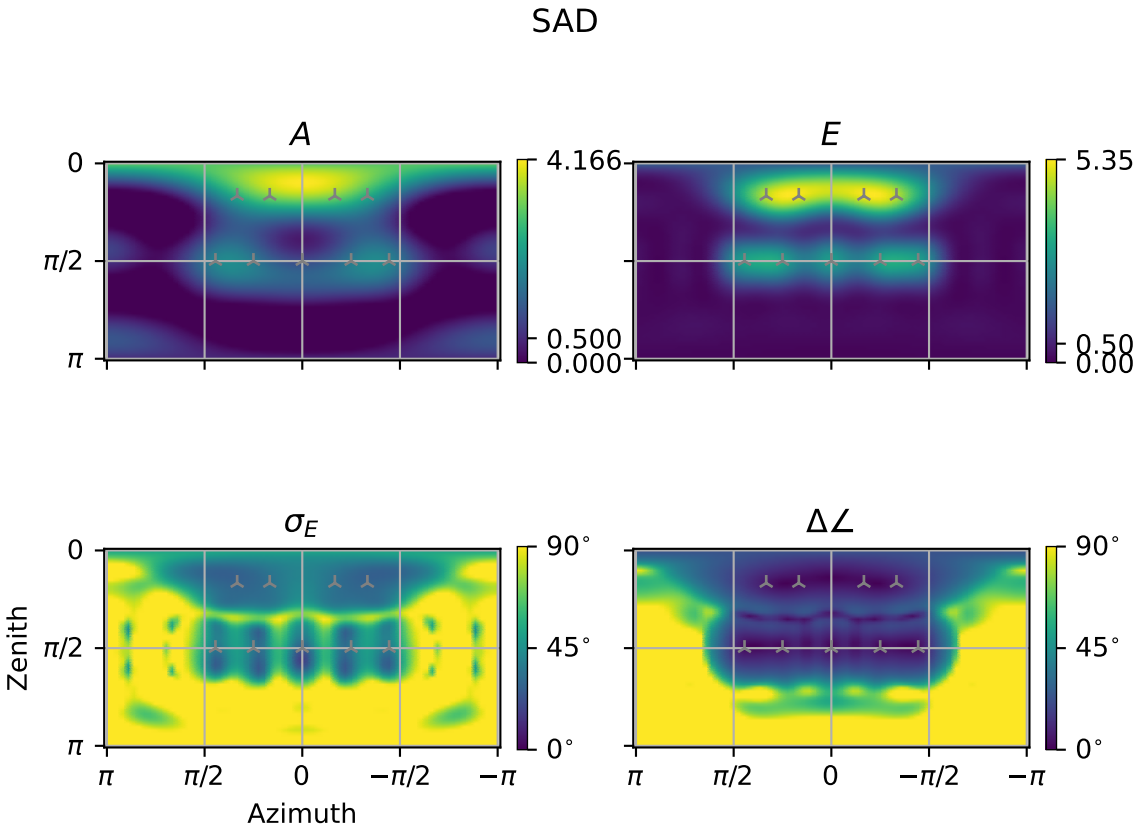
Returns **ls_sig** (*(L, S) numpy.ndarray*) – Loudspeaker L output signal S.

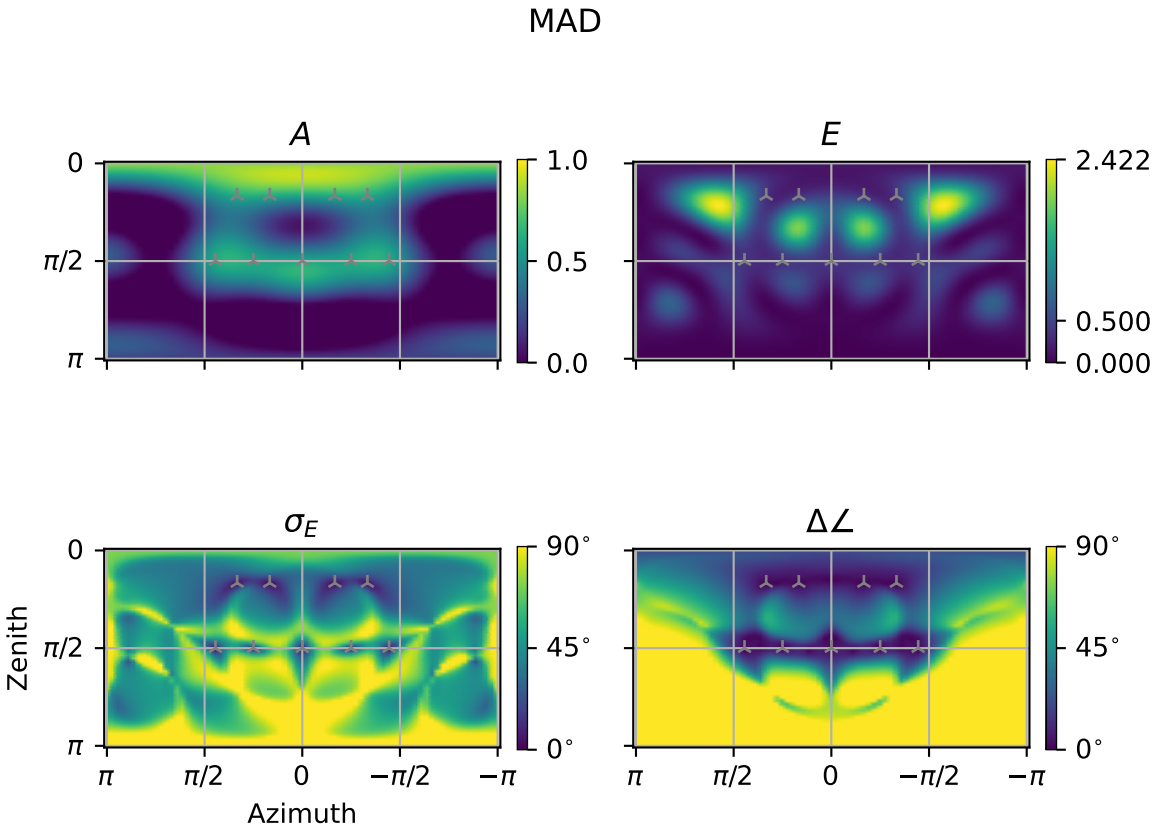
References

ch. 4.9.2, Zotter, F., & Frank, M. (2019). Ambisonics. Springer Topics in Signal Processing.

Examples

```
ls_setup = spa.decoder.LoudspeakerSetup(ls_x, ls_y, ls_z)
ls_setup.pop_triangles(normal_limit=85, aperture_limit=90,
                       opening_limit=150)

spa.plots.decoder_performance(ls_setup, 'MAD')
```


SAD
A
E
0
π/2
π
4.166
0.500
0.000
5.35
0.50
0.00
σE
ΔΔ
Zenith
90°
45°
0°
π
π/2
0
−π/2
−π
Azimuth


MAD
A
E
σ_E
$\Delta\angle$
0
π/2
π
1.0
0.5
0.0
2.422
0.500
0.000
90°
45°
0°
Zenith
Azimuth
π
π/2
0
−π/2
−π

spaudiopy.decoder.**epad**(*F_nm*, *hull*, *N_sph=None*)

Loudspeaker signals of Energy-Preserving Ambisonic Decoder.

Parameters

- **F_nm** (*((N_sph+1)**2, S) numpy.ndarray*) – Matrix of spherical harmonics coefficients of spherical function(S).
- **hull** (*LoudspeakerSetup*)
- **N_sph** (*int*) – Decoding order, defaults to hull.characteristic_order.

Returns **ls_sig** (*(L, S) numpy.ndarray*) – Loudspeaker L output signal S.

Notes

Number of loudspeakers should be greater or equal than SH channels, i.e.

$$L \geq (N_{sph} + 1)^2.$$

References

Zotter, F., Pomberger, H., & Noisternig, M. (2012). Energy-preserving ambisonic decoding. Acta Acustica United with Acustica, 98(1), 37–47.

Examples

```
ls_setup = spa.decoder.LoudspeakerSetup(ls_x, ls_y, ls_z)
ls_setup.pop_triangles(normal_limit=85, aperture_limit=90,
                       opening_limit=150)

spa.plots.decoder_performance(ls_setup, 'EPAD')

spa.plots.decoder_performance(ls_setup, 'EPAD', N_sph=2,
                              title='$N_{sph}=2$')
```

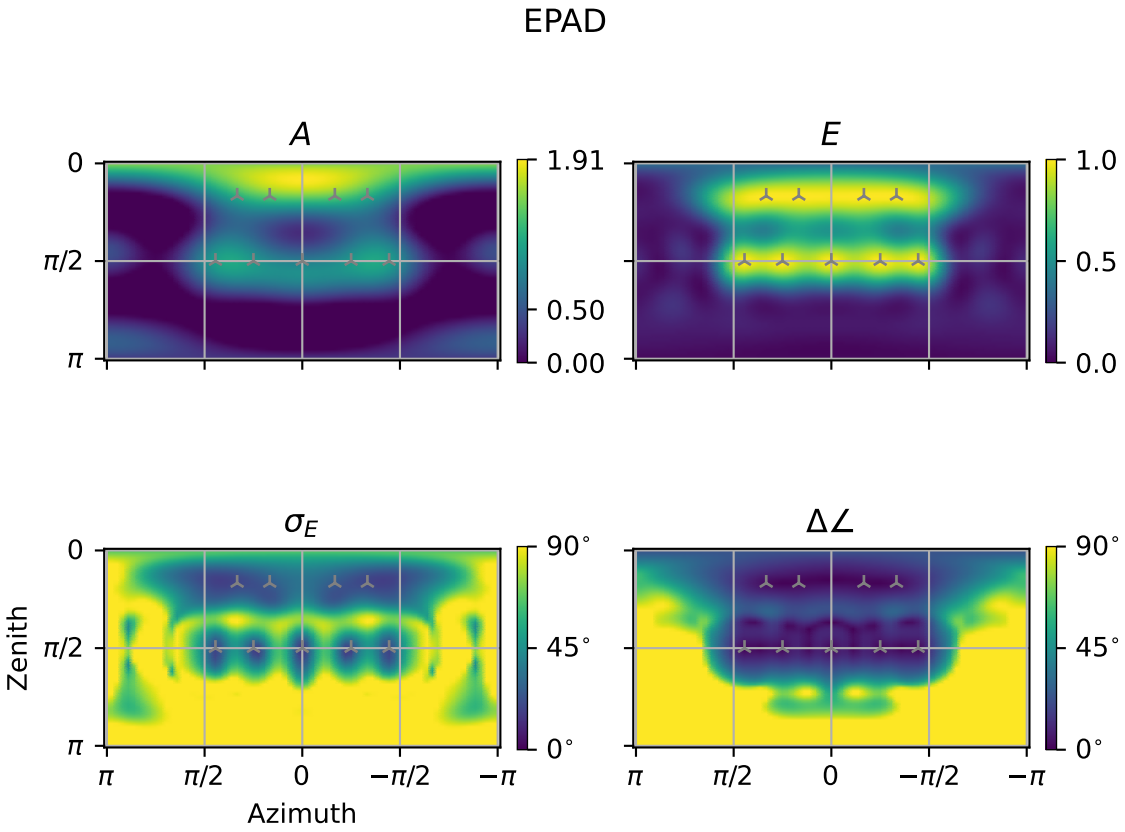
spaudiopy.decoder.**nearest_loudspeaker**(*src*, *hull*)

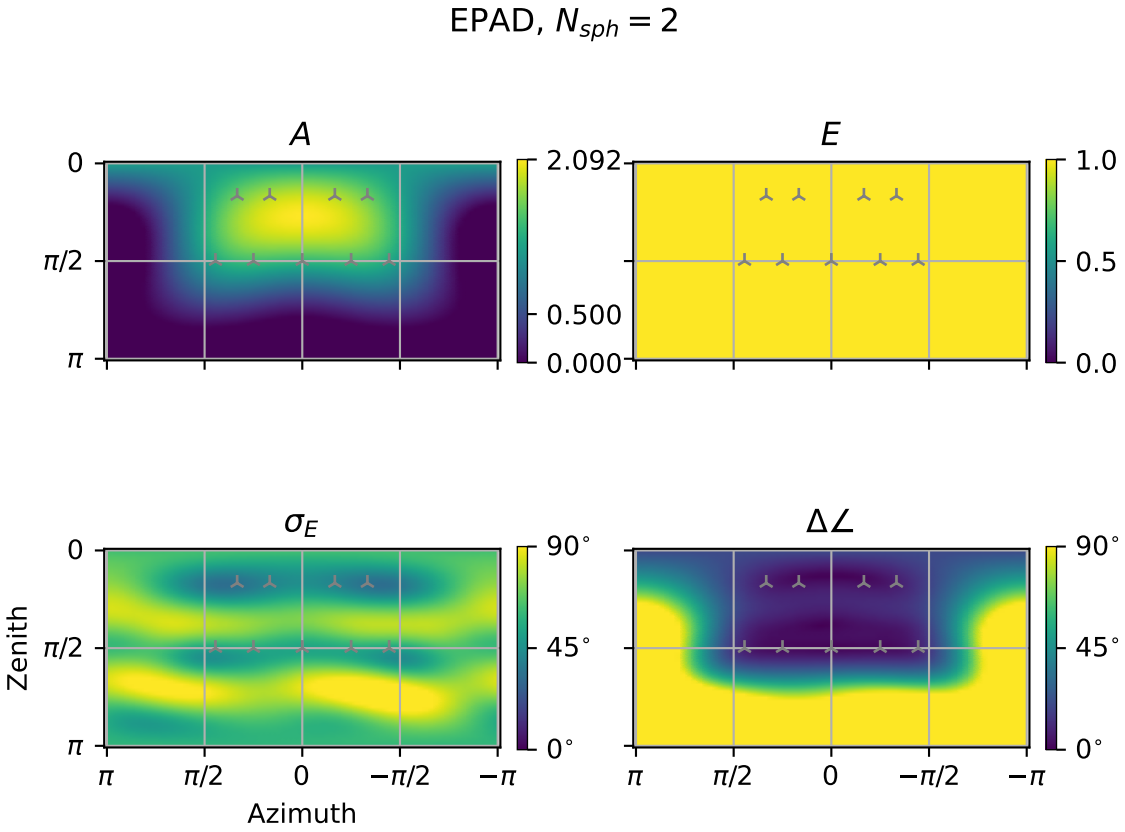
Loudspeaker gains for nearest loudspeaker selection (NLS) decoding, based on euclidean distance.

Parameters

- **src** (*(N, 3)*) – Cartesian coordinates of N sources to be rendered.
- **hull** (*LoudspeakerSetup*)

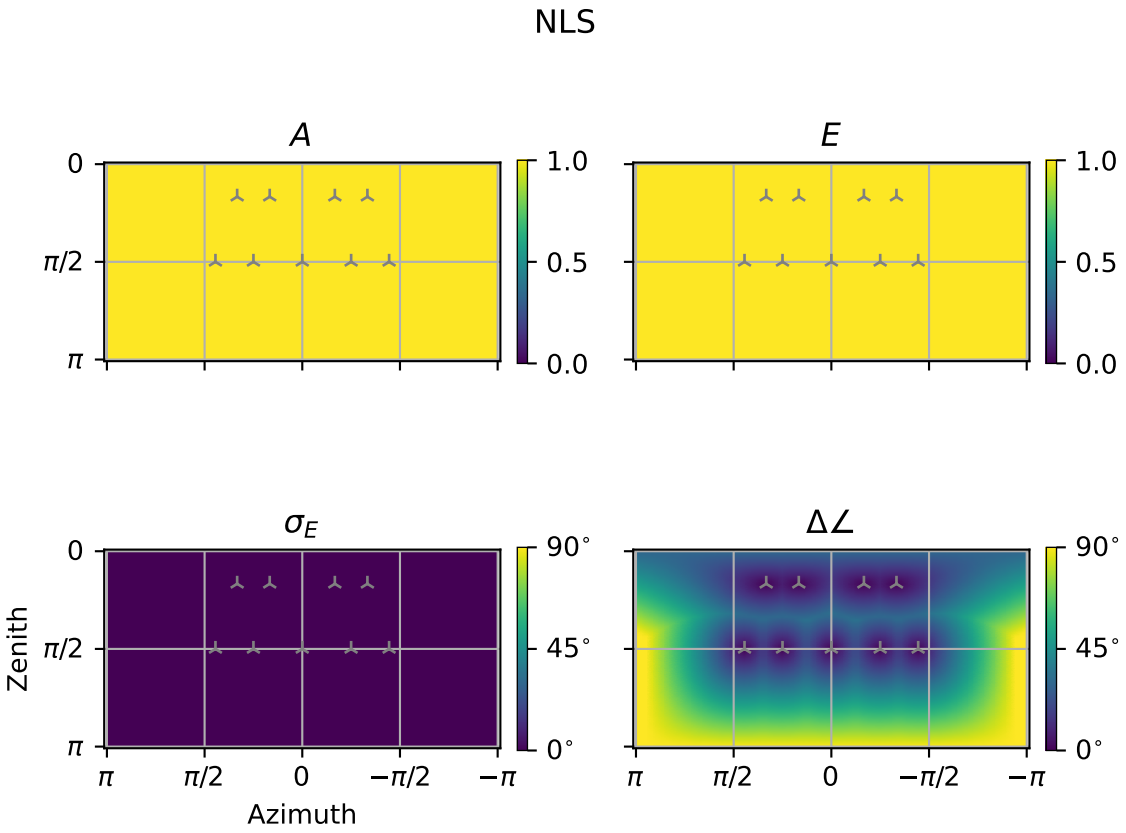
Returns **gains** (*(N, L) numpy.ndarray*) – Panning gains for L loudspeakers to render N sources.


EPAD
A
E
σ_E
$\Delta\angle$
0
π/2
π
1.91
0.50
0.00
1.0
0.5
0.0
90°
45°
0°
Zenith
Azimuth
π
π/2
0
−π/2
−π


EPAD, $N_{sph} = 2$
A
E
σ_E
$\Delta\angle$
0
$\pi/2$
π
2.092
0.500
0.000
1.0
0.5
0.0
90°
45°
0°
π
$\pi/2$
0
$-\pi/2$
$-\pi$
Zenith
Azimuth

Examples

```
ls_setup = spa.decoder.LoudspeakerSetup(ls_x, ls_y, ls_z)
ls_setup.pop_triangles(normal_limit=85, aperture_limit=90,
                       opening_limit=150)

spa.plots.decoder_performance(ls_setup, 'NLS')
```



spaudiopy.decoder.**sh2bin**(*sig_nm*, *hrirs_nm*)

Spherical Harmonic Domain signals to binaural renderer. Ambisonic signals as N3D - ACN, i.e. real valued SH (time) signals.

Parameters

- **sig_nm** (*((N_sph+1)**2, S) numpy.ndarray*) – Input signal (SHD / Ambisonics).
- **hrirs_nm** (*(2, (N_sph+1)**2, L)*) – Decoding IRs matrix, 2: left,right (stacked), real coeffs, L taps.

Returns *(2, S+L-1) numpy.ndarray* – Left and Right (stacked) binaural output signals.

2.5 spaudiopy.process

Collection of audio processing tools.

```
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['axes.grid'] = True

import spaudiopy as spa
```

Memory cached functions

spaudiopy.process.**resample_hrirs**(*hrir_l*, *hrir_r*, *fs_hrir*, *fs_target*, *jobs_count=None*)

Memoized version of resample_hrirs(hrir_l, hrir_r, fs_hrir, fs_target, jobs_count=None)

Resample HRIRs to new SamplingRate(t), using multiprocessing.

Parameters

- **hrir_l** (*(g, h) numpy.ndarray*) – h(t) for grid position g.
- **hrir_r** (*(g, h) numpy.ndarray*) – h(t) for grid position g.
- **fs_hrir** (*int*) – Current fs(t) of hrirs.
- **fs_target** (*int*) – Target fs(t) of hrirs.
- **jobs_count** (*int or None, optional*) – Number of parallel jobs, 'None' employs 'cpu_count'.

Returns

- **hrir_l_resampled** (*(g, h_n) numpy.ndarray*) – h_n(t) resampled for grid position g.
- **hrir_r_resampled** (*(g, h_n) numpy.ndarray*) – h_n(t) resampled for grid position g.
- **fs_hrir** (*int*) – New fs(t) of hrirs.

Functions

ambeo_a2b(Ambi_A[, filter_coeffs])	Convert A 'MultiSignal' (type I: FLU, FRD, BLD, BRU) to B AmbiBSignal.
b_to_stereo(Ambi_B)	Downmix B format first order Ambisonics to Stereo.
energy_decay(p)	Energy decay curve (EDC) in dB by Schroeder backwards integration.
frac_octave_filterbank(n, N_out, fs, f_low)	Fractional octave band filterbank.
gain_clipping(gain, threshold)	Limit gain factor by soft clipping function.
half_sided_Hann(N)	Design half-sided Hann tapering window of order N (>=3).
ilds_from_hrirs(hrirs[, f_cut, INDB])	Calculate ILDs from HRIRs by high-passed broad-band RMS.
itds_from_hrirs(hrirs[, f_cut, upsample])	Calculate ITDs from HRIRs by upsampled and filtered cross-correlation.
lagrange_delay(N, delay)	Return fractional delay filter using lagrange interpolation.
match_loudness(sig_in, sig_target)	Match loundess of input to target, based on RMS and avoid clipping.
pulsed_noise(t_noise, t_pause, fs[, reps, ...])	Pulsed noise train, pink or white.
resample_signal(s_time, fs_current, fs_target)	Resample time signal.
resample_spectrum(single_spec, fs_current, ...)	Resample single sided spectrum, as e.g.
subband_levels(x, width, fs[, power, axis])	Computes the level/power in each subband of subband signals.

spaudiopy.process.**resample_signal**(*s_time*, *fs_current*, *fs_target*, *axis=- 1*)

Resample time signal.

Parameters

- **s_time** (*numpy.ndarray*) – Time signal, or signals stacked.
- **fs_current** (*int*)
- **fs_target** (*int*)
- **axis** (*int, optional*) – Axis along which to resample. The default is -1.

Returns **single_spec_resamp** (*numpy.ndarray.*)

spaudiopy.process.**resample_spectrum**(*single_spec*, *fs_current*, *fs_target*, *axis=- 1*)

Resample single sided spectrum, as e.g. from np.fft.rfft().

Parameters

- **single_spec** (*numpy.ndarray*) – Single sided spectrum, or spectra stacked.
- **fs_current** (*int*)
- **fs_target** (*int*)
- **axis** (*int, optional*) – Axis along which to resample. The default is -1.

Returns **single_spec_resamp** (*numpy.ndarray.*)

spaudiopy.process.**ilds_from_hrirs**(*hrirs*, *f_cut=1000*, *INDB=True*)

Calculate ILDs from HRIRs by high-passed broad-band RMS.

Parameters

- **hrirs** (*sig.HRIRs*)
- **f_cut** (*float, optional*) – Low-pass cutoff frequency. The default is 1000.

Returns

- **ild** (*array_like*) – ILD per grid point, positive value indicates left ear louder.
- **INDB** (*bool, optional*) – ILD in dB RMS ratio, otherwise as RMS difference. The default is TRUE.

spaudiopy.process.**itds_from_hrirs**(*hrirs*, *f_cut=1000*, *upsample=8*)

Calculate ITDs from HRIRs by upsampled and filtered cross-correlation.

Parameters

- **hrirs** (*sig.HRIRs*)
- **f_cut** (*float, optional*) – Low-pass cutoff frequency. The default is 1000.
- **upsample** (*int, optional*) – Upsampling factor. The default is 8.

Returns **itd** (*array_like*) – ITD in seconds per grid point, positive value indicates left ear first.

spaudiopy.process.**match_loudness**(*sig_in*, *sig_target*)

Match loundess of input to target, based on RMS and avoid clipping.

Parameters

- **sig_in** (*(n, c) array_like*) – Input(t) samples n, channel c.
- **sig_target** (*(n, c) array_like*) – Target(t) samples n, channel c.

Returns **sig_out** (*(n, c) array_like*) – Output(t) samples n, channel c.

spaudiopy.process.**ambeo_a2b**(*Ambi_A*, *filter_coeffs=None*)

Convert A 'MultiSignal' (type I: FLU, FRD, BLD, BRU) to B AmbiBSignal.

Parameters

- **Ambi_A** (*sig.MultiSignal*) – Input signal.
- **filter_coeffs** (*string*) – Picklable file that contains b0_d, a0_d, b1_d, a1_d.

Returns **Ambi_B** (*sig.AmbiBSignal*) – B-format output signal.

spaudiopy.process.**b_to_stereo**(*Ambi_B*)

Downmix B format first order Ambisonics to Stereo.

Parameters **Ambi_B** (*sig.AmbiBSignal*) – B-format output signal.

Returns **L, R** (*array_like*)

spaudiopy.process.**lagrange_delay**(*N*, *delay*)

Return fractional delay filter using lagrange interpolation. For best results, delay should be near N/2 +/- 1.

Parameters

- **N** (*int*) – Filter order.
- **delay** (*float*) – Delay in samples.

Returns **h** (*(N+1,) array_like*) – FIR Filter.

spaudiopy.process.**frac_octave_filterbank**(*n*, *N_out*, *fs*, *f_low*, *f_high=None*, *mode='energy'*, *overlap=0.5*, *l=3*)

Fractional octave band filterbank. Design of digital fractional-octave-band filters with energy conservation and perfect reconstruction.

Parameters

- **n** (*int*) – Octave fraction, e.g. n=3 third-octave bands.
- **N_out** (*int*) – Number of non-negative frequency bins [0, fs/2].
- **fs** (*int*) – Sampling frequency in Hz.
- **f_low** (*int*) – Center frequency of first full band in Hz.
- **f_high** (*int*) – Cutoff frequency in Hz, above which no further bands are generated.
- **mode** (*'energy' or 'amplitude'*) – 'energy' produces -3dB at crossover, 'amplitude' -6dB.
- **overlap** (*float*) – Band overlap, should be between [0, 0.5].
- **l** (*int*) – Band transition slope, implemented as recursion order *l*.

Returns

- **g** (*(b, N) np.ndarray*) – Band gains for non-negative frequency bins.
- **ff** (*(b, 3) np.ndarray*) – Filter frequencies as [f_lo, f_c, f_hi].

Notes

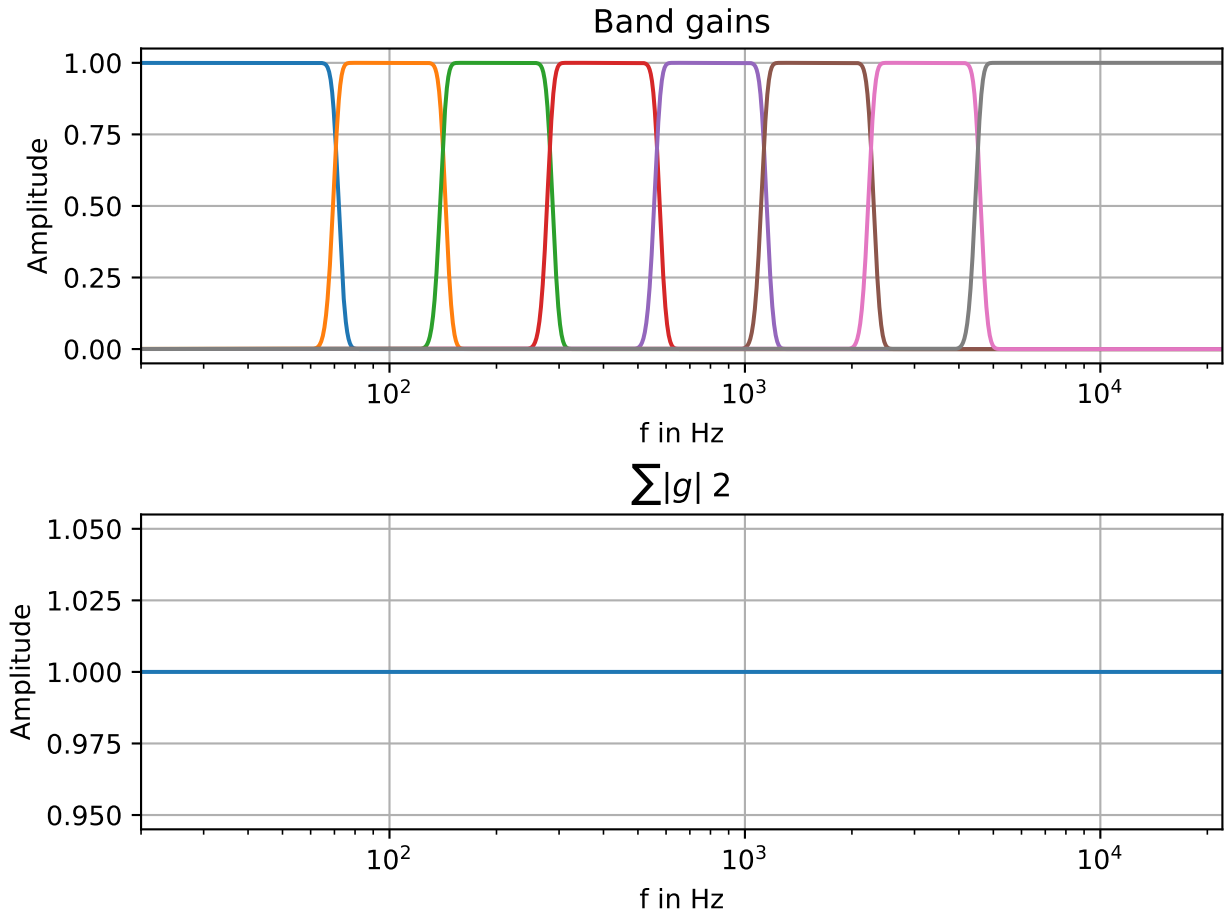
This filterbank is originally designed such that the sum of gains squared sums to unity. The alternative 'amplitude' mode ensures that the gains sum directly to unity.

References

Antoni, J. (2010). Orthogonal-like fractional-octave-band filters. The Journal of the Acoustical Society of America, 127(2), 884–895.

Examples

```
fs = 44100
N = 2**16
gs, ff = spa.process.frac_octave_filterbank(n=1, N_out=N, fs=fs,
                                            f_low=100, f_high=8000)
f = np.linspace(0, fs//2, N)
fig, ax = plt.subplots(2, 1, constrained_layout=True)
ax[0].semilogx(f, gs.T)
ax[0].set_title('Band gains')
ax[1].semilogx(f, np.sum(np.abs(gs)**2, axis=0))
ax[1].set_title(r'$\sum |g| ^ 2$')
for a_idx in ax:
    a_idx.grid(True)
    a_idx.set_xlim([20, fs//2])
    a_idx.set_xlabel('f in Hz')
    a_idx.set_ylabel('Amplitude')
```


Band gains
Amplitude
1.00
0.75
0.50
0.25
0.00
10^2
10^3
10^4
f in Hz
$\sum|g|^2$
Amplitude
1.050
1.025
1.000
0.975
0.950
10^2
10^3
10^4
f in Hz

spaudiopy.process.**subband_levels**(*x*, *width*, *fs*, *power=False*, *axis=- 1*)

Computes the level/power in each subband of subband signals.

spaudiopy.process.**energy_decay**(*p*)

Energy decay curve (EDC) in dB by Schroeder backwards integration.

Parameters **p** (*array_like*)

Returns **rd** (*array_like*)

spaudiopy.process.**half_sided_Hann**(*N*)

Design half-sided Hann tapering window of order N (>=3).

spaudiopy.process.**gain_clipping**(*gain*, *threshold*)

Limit gain factor by soft clipping function. Limits gain factor to +6dB beyond threshold point. (Pass values as factors/ratios, not dB!)

Parameters

- **gain** (*array_like*)
- **threshold** (*float*)

Returns **gain_clipped** (*array_like*)

Examples

```
x = np.linspace(-10, 10, 1000)
lim_threshold = 2.5
y = spa.process.gain_clipping(x, lim_threshold)
plt.figure()
plt.plot(x, x, '--', label='In')
plt.plot(x, y, label='Out')
plt.legend()
plt.xlabel('In')
plt.ylabel('Out')
plt.grid(True)
```

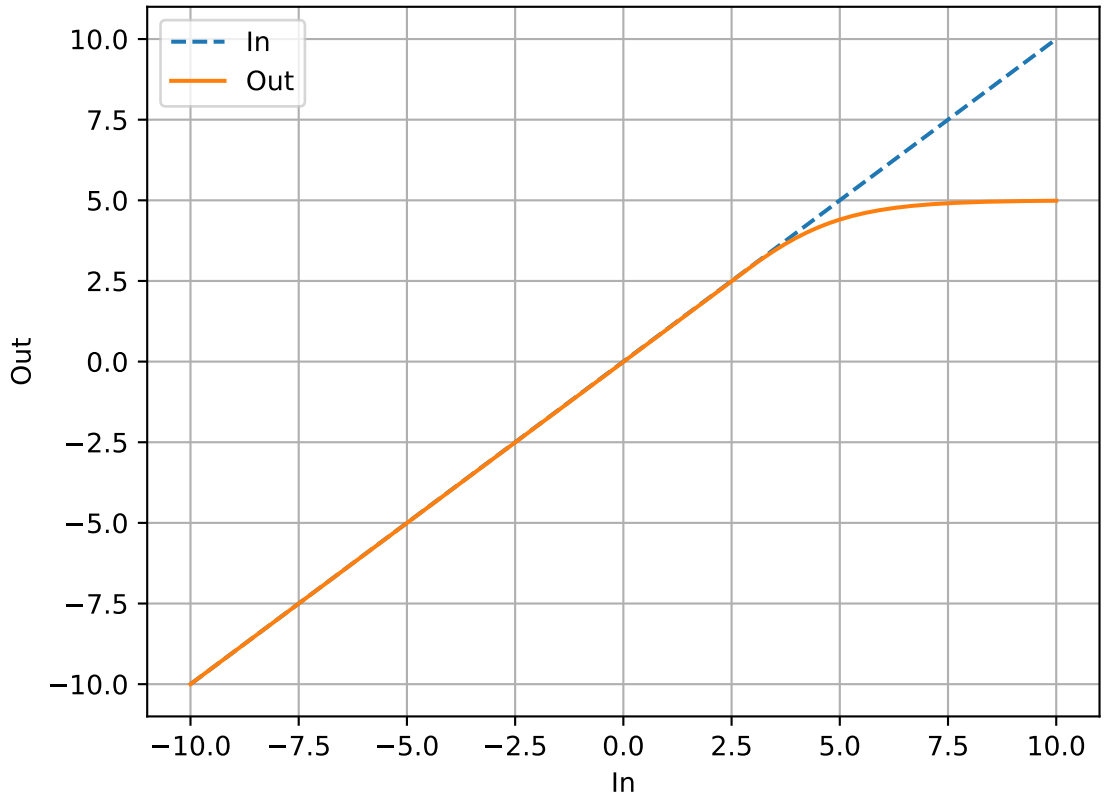
spaudiopy.process.**pulsed_noise**(*t_noise*, *t_pause*, *fs*, *reps=10*, *t_fade=0.02*, *pink_noise=True*, *normalize=True*)

Pulsed noise train, pink or white.

Parameters

- **t_noise** (*float*) – t in s for pulse.
- **t_pause** (*float*) – t in s between pulses.
- **fs** (*int*) – Sampling frequency.
- **reps** (*int, optional*) – Repetitions (independent). The default is 10.
- **t_fade** (*float, optional*) – t in s for fade in and out. The default is 0.02.
- **pink_noise** (*bool, optional*) – Use 'pink' (1/f) noise. The default is True
- **normalize** (*bool, optional*) – Normalize output. The default is True.

Returns **s_out** (*array_like*) – output signal.


In
Out
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
Out
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
In

2.6 spaudiopy.utils

A few helpers.

```
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['axes.grid'] = True

import spaudiopy as spa
```

Functions

angle_between(v1, v2[, vi])	Angle between point v1 and v2(s) with initial point vi.
area_triangle(p1, p2, p3)	calculate area of any triangle given coordinates of its corners p.
asarray_1d(a, **kwargs)	Squeeze the input and check if the result is one-dimensional.
cart2sph(x, y, z[, steady_colat])	Vectorized conversion of cartesian to spherical coordinates.
db(x[, power])	Convert ratio *x* to decibel.
deg2rad(deg)	Convert from degree [0, 360) to radiant [0, 2*pi).
from_db(db[, power])	Convert decibel back to ratio.
haversine(azi1, colat1, azi2, colat2[, r])	Calculate the spherical distance between two points on the sphere.
interleave_channels(left_channel, right_channel)	Interleave left and right channels (Nchannel x Nsamples).
matlab_sph2cart(az, elev, r)	Matlab port with ELEVATION.
rad2deg(rad)	Convert from radiant [0, 2*pi) to degree [0, 360).
rms(x[, axis])	RMS (root-mean-squared) along given axis.
sph2cart(azi, colat[, r])	Vectorized conversion of spherical to cartesian coordinates.
stack(vector_1, vector_2)	Stack two 2D vectors along the same-sized or the smaller dimension.
test_diff(v1, v2[, msg, axis, test_lim, VERBOSE])	Test if the cumulative element-wise difference between v1 and v2.
vecs2dirs(vecs[, positive_azi])	Helper to convert [x, y, z] to [azi, colat].

spaudiopy.utils.**asarray_1d**(*a*, ***kwargs*)

Squeeze the input and check if the result is one-dimensional.

Returns *a* converted to a *numpy.ndarray* and stripped of all singleton dimensions. Scalars are "upgraded" to 1D arrays. The result must have exactly one dimension. If not, an error is raised.

spaudiopy.utils.**deg2rad**(*deg*)

Convert from degree [0, 360) to radiant [0, 2*pi).

spaudiopy.utils.**rad2deg**(*rad*)

Convert from radiant [0, 2*pi) to degree [0, 360).

spaudiopy.utils.**cart2sph**(*x*, *y*, *z*, *steady_colat=False*)

Vectorized conversion of cartesian to spherical coordinates.

spaudiopy.utils.**sph2cart**(*azi*, *colat*, *r=1*)

Vectorized conversion of spherical to cartesian coordinates.

spaudiopy.utils.**matlab_sph2cart**(*az*, *elev*, *r*)

Matlab port with ELEVATION.

spaudiopy.utils.**vecs2dirs**(*vecs*, *positive_azi=True*)

Helper to convert [x, y, z] to [azi, colat].

spaudiopy.utils.**angle_between**(*v1*, *v2*, *vi=None*)

Angle between point v1 and v2(s) with initial point vi.

spaudiopy.utils.**haversine**(*azi1*, *colat1*, *azi2*, *colat2*, *r=1*)

Calculate the spherical distance between two points on the sphere. The spherical distance is central angle for r=1.

Parameters

- **azi1** (*(n,) array_like*)
- **colat1** (*(n,) array_like.*)
- **azi2** (*(n,) array_like*)
- **colat2** (*(n,) array_like*)
- **r** (*float, optional.*)

Returns

- **c** (*(n,) array_like*) – Haversine distance between pairs of points.
- *Reference*
- ———
- **https** (*//en.wikipedia.org/wiki/Haversine_formula*)

spaudiopy.utils.**area_triangle**(*p1*, *p2*, *p3*)

calculate area of any triangle given coordinates of its corners p.

spaudiopy.utils.**db**(*x*, *power=False*)

Convert ratio *x* to decibel.

Parameters

- **x** (*array_like*) – Input data. Values of 0 lead to negative infinity.
- **power** (*bool, optional*) – If `power=False` (the default), *x* is squared before conversion.

spaudiopy.utils.**from_db**(*db*, *power=False*)

Convert decibel back to ratio.

Parameters

- **db** (*array_like*) – Input data.
- **power** (*bool, optional*) – If `power=False` (the default), was used for conversion to dB.

spaudiopy.utils.**rms**(*x*, *axis=- 1*)

RMS (root-mean-squared) along given axis.

Parameters

- **x** (*array_like*) – Input data.

- **axis** (*int, optional*) – Axis along which RMS is calculated

spaudiopy.utils.**stack**(*vector_1*, *vector_2*)

Stack two 2D vectors along the same-sized or the smaller dimension.

spaudiopy.utils.**test_diff**(*v1*, *v2*, *msg=None*, *axis=None*, *test_lim=1e-06*, *VERBOSE=True*)

Test if the cumulative element-wise difference between v1 and v2. Return difference and be verbose if is greater *test_lim*.

spaudiopy.utils.**interleave_channels**(*left_channel*, *right_channel*, *style=None*)

Interleave left and right channels (Nchannel x Nsamples). Style = 'SSR' checks if we total 360 channels.

2.7 spaudiopy.grids

Sampling grids.

```
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['axes.grid'] = True

import spaudiopy as spa
```

Functions

equal_angle(n)	Equi-angular sampling points on a sphere.
equal_polar_angle(n)	Equi-angular sampling points on a circle.
gauss(n)	Gauss-Legendre sampling points on sphere.
load_Fliege_Maier_nodes(grid_order)	Return Fliege-Maier grid nodes with associated weights.
load_lebedev(degree)	Return the unit coordinates of Lebedev grid.
load_maxDet(degree)	Return Maximum Determinant (Fekete, Extremal) points on the sphere.
load_n_design(degree)	Return the unit coordinates of spherical N-design (Chebyshev-type quadrature rules).
load_t_design(degree)	Return the unit coordinates of minimal T-designs.

spaudiopy.grids.**load_t_design**(*degree*)

Return the unit coordinates of minimal T-designs.

The designs have been copied from: http://neilsloane.com/sphdesigns/ and should be referenced as:

> "McLaren's Improved Snub Cube and Other New Spherical Designs in Three Dimensions", R. H. Hardin and N. J. A. Sloane, Discrete and Computational Geometry, 15 (1996), pp. 429-441.

Parameters **degree** (*int*) – T-design degree between 1 and 21.

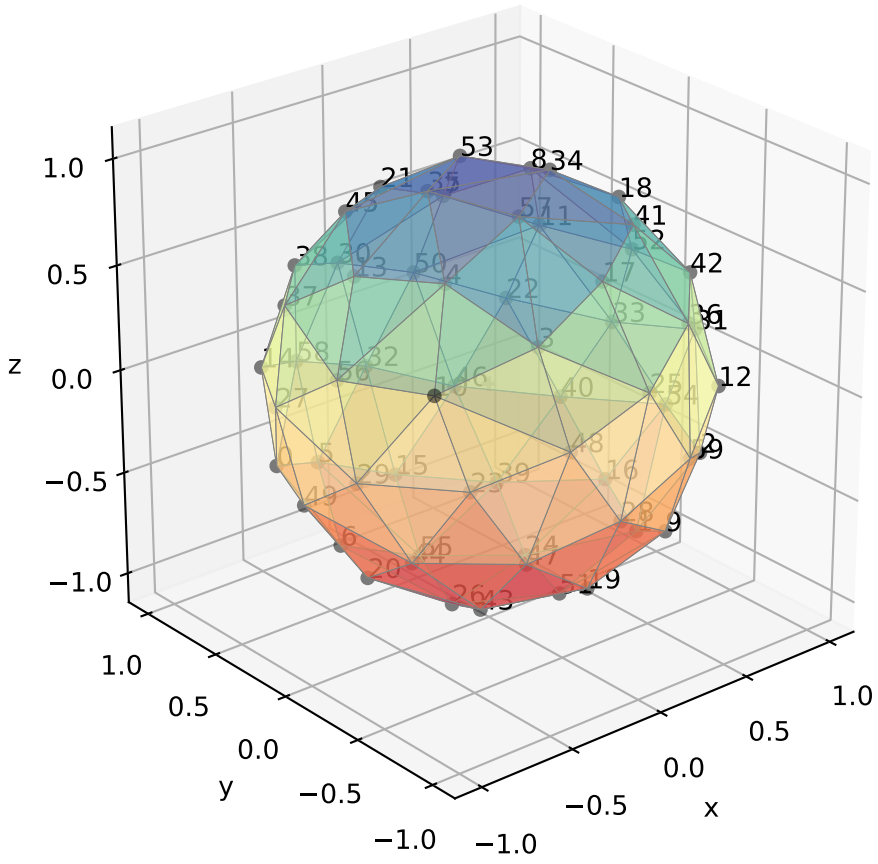
Returns **vecs** (*(M, 3) numpy.ndarray*) – Coordinates of points.

Notes

Degree must be >= 2 * SH_order for spherical harmonic transform (SHT).

Examples

```
vecs = spa.grids.load_t_design(degree=2*5)
spa.plots.hull(spa.decoder.get_hull(*vecs.T))
```



spaudiopy.grids.**load_n_design**(*degree*)

Return the unit coordinates of spherical N-design (Chebyshev-type quadrature rules). Seem to be equivalent but more modern t-designs.

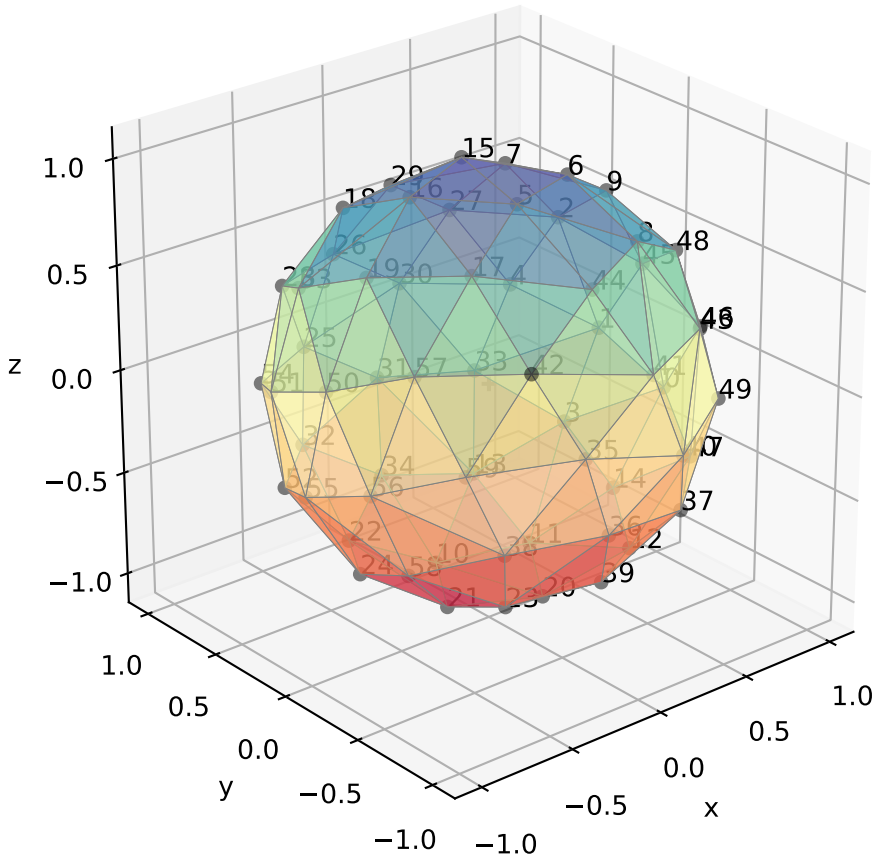
The designs have been copied from: https://homepage.univie.ac.at/manuel.graef/quadrature.php

Parameters **degree** (*int*) – Degree of exactness N between 1 and 124.

Returns **vecs** (*(M, 3) numpy.ndarray*) – Coordinates of points.

Examples

```
vecs = spa.grids.load_n_design(degree=2*5)
spa.plots.hull(spa.decoder.get_hull(*vecs.T))
```



spaudiopy.grids.**load_lebedev**(*degree*)

Return the unit coordinates of Lebedev grid.

The designs have been copied from: https://people.sc.fsu.edu/~jburkardt/datasets/sphere_lebedev_rule/sphere_lebedev_rule.html

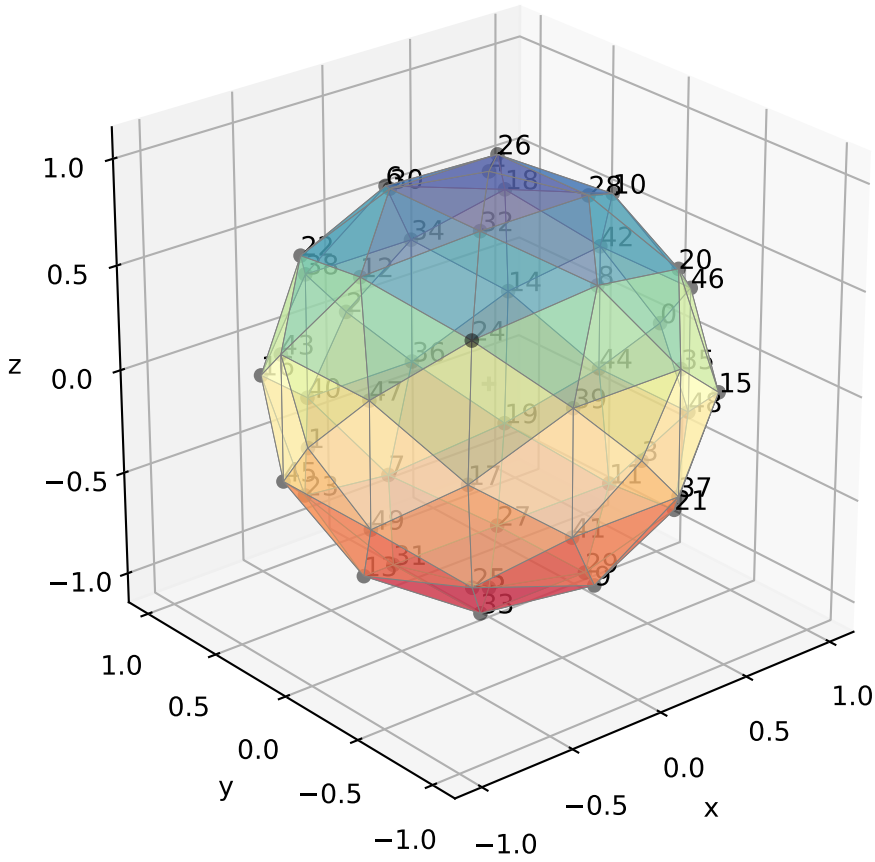
Parameters **degree** (*int*) – Degree of precision p between 3 and 131.

Returns

- **vecs** (*(M, 3) numpy.ndarray*) – Coordinates of points.
- **weights** (*array_like*) – Quadrature weights.

Examples

```
vecs, weights = spa.grids.load_lebedev(degree=2*5)
spa.plots.hull(spa.decoder.get_hull(*vecs.T))
```



spaudiopy.grids.**load_Fliege_Maier_nodes**(*grid_order*)

Return Fliege-Maier grid nodes with associated weights.

The designs have been copied from: http://www.personal.soton.ac.uk/jf1w07/nodes/nodes.html and should be referenced as:

> "A two-stage approach for computing cubature formulae for the sphere.", Jorg Fliege and Ulrike Maier, Mathematik 139T, Universitat Dortmund, Fachbereich Mathematik, Universitat Dortmund, 44221. 1996.

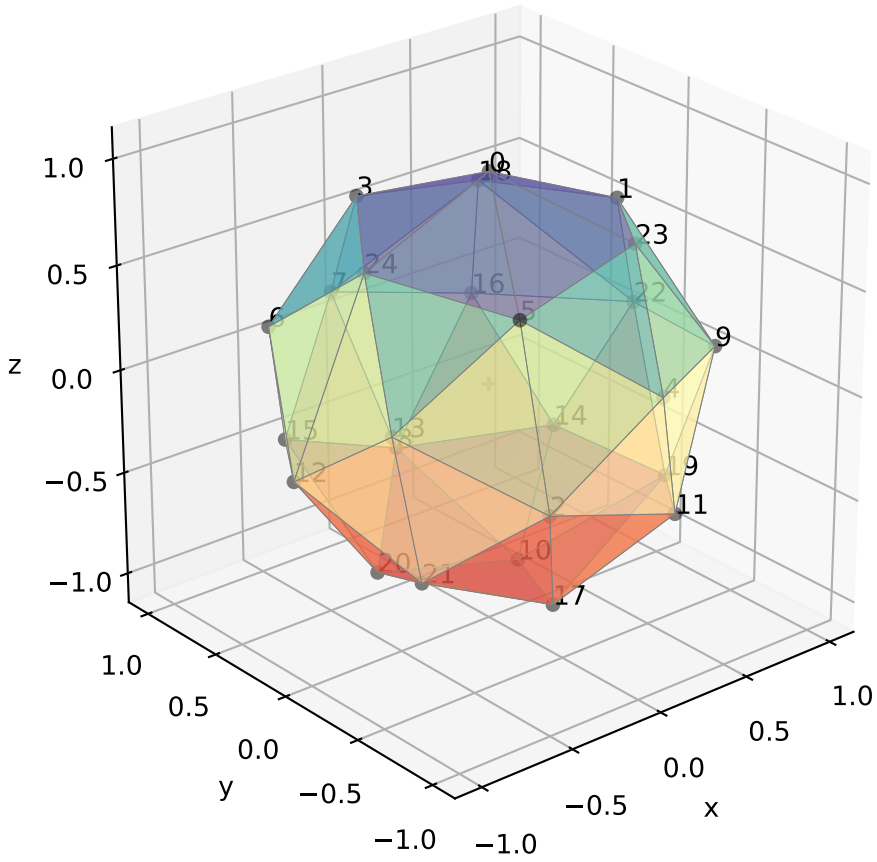
Parameters **grid_order** (*int*) – Grid order between 2 and 30

Returns

- **vecs** (*(M, 3) numpy.ndarray*) – Coordinates of points.
- **weights** (*array_like*) – Quadrature weights.

Examples

```
vecs, weights = spa.grids.load_Fliege_Maier_nodes(grid_order=5)
spa.plots.hull(spa.decoder.get_hull(*vecs.T))
```



spaudiopy.grids.**load_maxDet**(*degree*)

Return Maximum Determinant (Fekete, Extremal) points on the sphere.

The designs have been copied from: https://web.maths.unsw.edu.au/~rsw/Sphere/MaxDet/

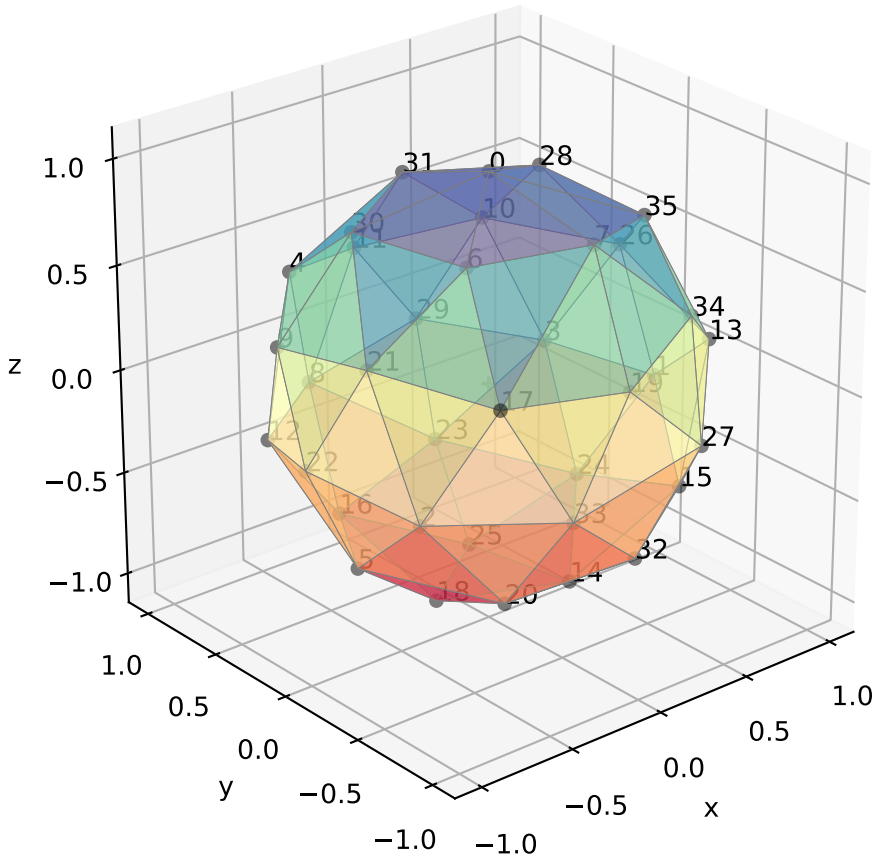
Parameters **degree** (*int*) – Degree between 1 and 200.

Returns

- **vecs** (*(M, 3) numpy.ndarray*) – Coordinates of points.
- **weights** (*array_like*) – Quadrature weights.

Examples

```
vecs, weights = spa.grids.load_maxDet(degree=5)
spa.plots.hull(spa.decoder.get_hull(*vecs.T))
```



spaudiopy.grids.**equal_angle**(*n*)

Equi-angular sampling points on a sphere.

According to (cf. Rafaely book, sec.3.2)

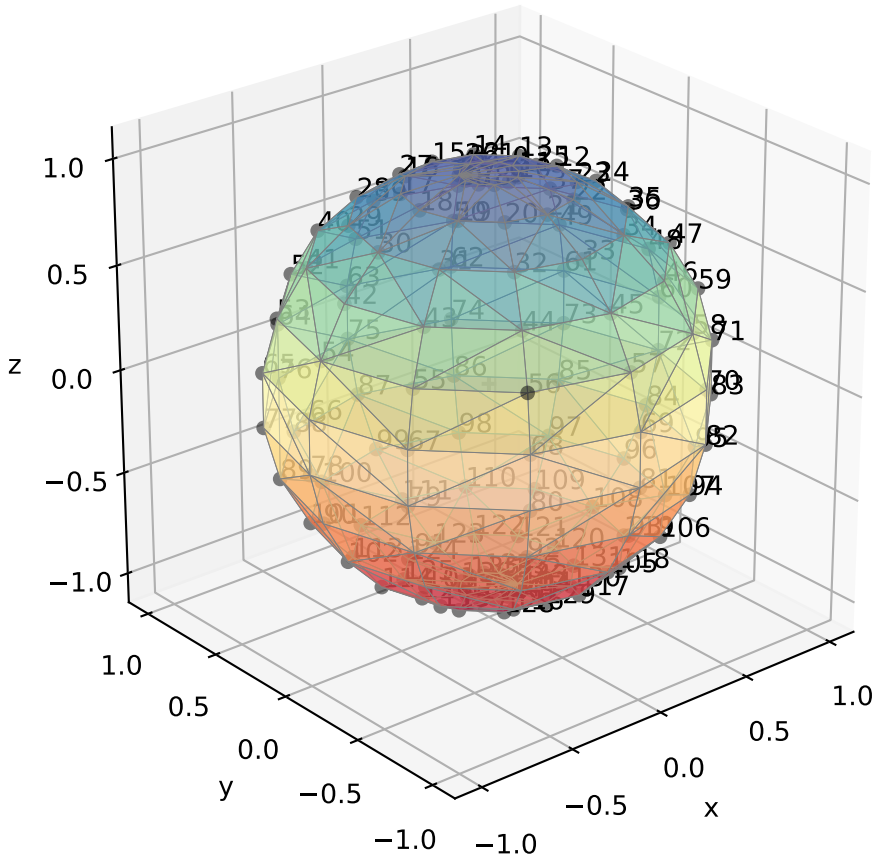
Parameters **n** (*int*) – Maximum order.

Returns

- **azi** (*array_like*) – Azimuth.
- **colat** (*array_like*) – Colatitude.
- **weights** (*array_like*) – Quadrature weights.

Examples

```
azi, colat, weights = spa.grids.equal_angle(n=5)
spa.plots.hull(spa.decoder.get_hull(*spa.utils.sph2cart(azi, colat)))
```



spaudiopy.grids.**gauss**(*n*)

Gauss-Legendre sampling points on sphere.

According to (cf. Rafaely book, sec.3.3)

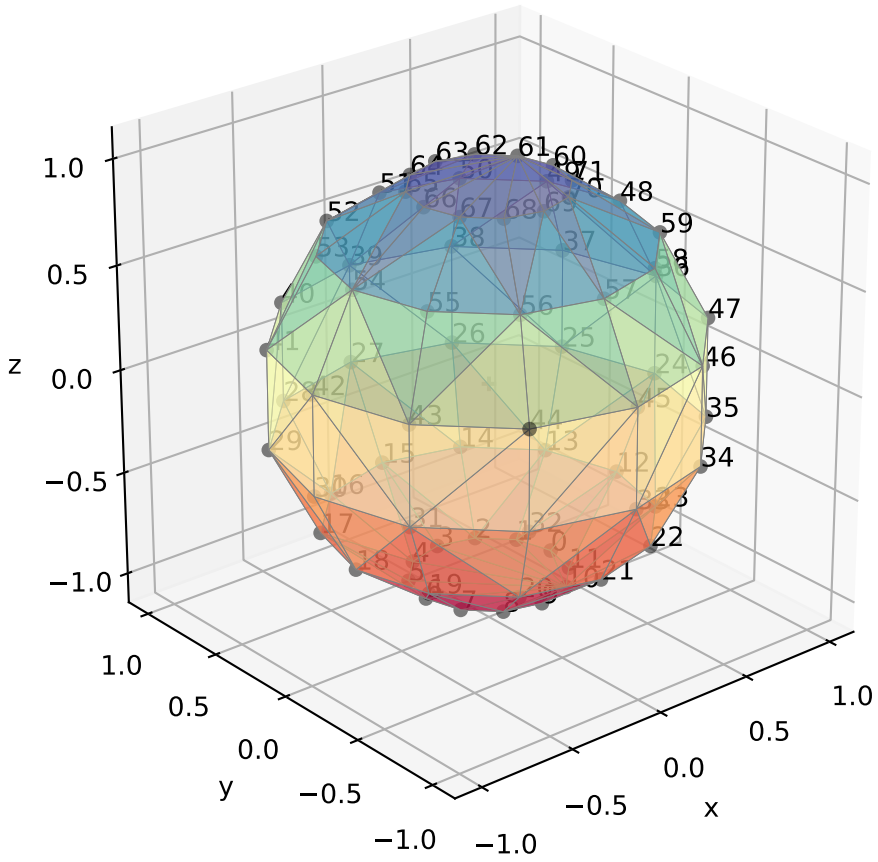
Parameters **n** (*int*) – Maximum order.

Returns

- **azi** (*array_like*) – Azimuth.
- **colat** (*array_like*) – Colatitude.
- **weights** (*array_like*) – Quadrature weights.

Examples

```
azi, colat, weights = spa.grids.gauss(n=5)
spa.plots.hull(spa.decoder.get_hull(*spa.utils.sph2cart(azi, colat)))
```



spaudiopy.grids.**equal_polar_angle**(*n*)

Equi-angular sampling points on a circle.

Parameters **n** (*int*) – Maximum order

Returns

- **pol** (*array_like*) – Polar angle.
- **weights** (*array_like*) – Weights.

2.8 spaudiopy.sdm

Spatial Decomposition Method (SDM).

```
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['axes.grid'] = True

import spaudiopy as spa
```

Memory cached functions

spaudiopy.sdm.**pseudo_intensity**(*ambi_b*, *win_len=33*, *f_bp=None*, *smoothing_order=5*, *jobs_count=1*)

Memoized version of pseudo_intensity(ambi_b, win_len=33, f_bp=None, smoothing_order=5, jobs_count=1)

Direction of arrival (DOA) for each time sample from pseudo-intensity.

Parameters

- **ambi_b** (*sig.AmbiBSignal*) – Input signal, B-format.
- **win_len** (*int optional*) – Sliding window length.
- **f_bp** (*tuple(f_lo, f_hi), optional*) – Cutoff frequencies for bandpass, 'None' to disable.
- **smoothing_order** (*int, optional*) – Apply hanning(smoothing_order) smoothing to output.
- **jobs_count** (*int or None, optional*) – Number of parallel jobs, 'None' employs 'cpu_count'.

Returns **I_azi, I_colat, I_r** (*array_like*) – Pseudo intensity vector for each time sample.

spaudiopy.sdm.**render_bsdm**(*sdm_p*, *sdm_phi*, *sdm_theta*, *hrirs*, *jobs_count=None*)

Memoized version of render_bsdm(sdm_p, sdm_phi, sdm_theta, hrirs, jobs_count=1)

Binaural SDM Render. Convolves each sample with corresponding hrir. No Post-EQ.

Parameters

- **sdm_p** (*(n,) array_like*) – Pressure p(t).
- **sdm_phi** (*(n,) array_like*) – Azimuth phi(t).
- **sdm_theta** (*(n,) array_like*) – Colatitude theta(t).
- **hrirs** (*sig.HRIRs*)
- **jobs_count** (*int or None, optional*) – Number of parallel jobs, 'None' employs 'cpu_count'.

Returns

- **bsdm_l** (*array_like*) – Left binaural impulse response.
- **bsdm_r** (*array_like*) – Right binaural impulse response.

Functions

post_equalization(ls_sigs, sdm_p, fs, ls_setup)	Post equalization to compensate spectral whitening.
post_equalization2(ls_sigs, sdm_p, fs, ls_setup)	Post equalization to compensate spectral whitening.
render_binaural_loudspeaker_sdm(sdm_p, ...)	Render sdm signal on loudspeaker setup as binaural synthesis.
render_stereo_sdm(sdm_p, sdm_phi, sdm_theta)	Stereophonic SDM Render IR, with a cos(phi) pannign law.

spaudiopy.sdm.**render_stereo_sdm**(*sdm_p*, *sdm_phi*, *sdm_theta*)

Stereophonic SDM Render IR, with a cos(phi) pannign law. This is only meant for quick testing.

Parameters

- **sdm_p** (*(n,) array_like*) – Pressure p(t).
- **sdm_phi** (*(n,) array_like*) – Azimuth phi(t).
- **sdm_theta** (*(n,) array_like*) – Colatitude theta(t).

Returns

- **ir_l** (*array_like*) – Left impulse response.
- **ir_r** (*array_like*) – Right impulse response.

spaudiopy.sdm.**render_binaural_loudspeaker_sdm**(*sdm_p*, *ls_gains*, *ls_setup*, *fs*, *post_eq_func='default'*, ***kwargs*)

Render sdm signal on loudspeaker setup as binaural synthesis.

Parameters

- **sdm_p** (*(n,) array_like*) – Pressure p(t).
- **ls_gains** (*(n, l)*) – Loudspeaker (l) gains.
- **ls_setup** (*decoder.LoudspeakerSetup*)
- **fs** (*int*)
- **post_eq_func** (*None, 'default' or function*) – Post EQ applied to the loudspeaker signals. 'default' calls 'sdm.post_equalization', 'None' disables (not recommended). You can also provide your custom post-eq-function with the signature *post_eq_func(ls_sigs, sdm_p, fs, ls_setup, **kwargs)*.

Returns

- **ir_l** (*array_like*) – Left binaural impulse response.
- **ir_r** (*array_like*) – Right binaural impulse response.

spaudiopy.sdm.**post_equalization**(*ls_sigs*, *sdm_p*, *fs*, *ls_setup*, *soft_clip=True*)

Post equalization to compensate spectral whitening.

Parameters

- **ls_sigs** (*(L, S) np.ndarray*) – Input loudspeaker signals.
- **sdm_p** (*array_like*) – Reference (sdm) pressure signal.
- **fs** (*int*)
- **ls_setup** (*decoder.LoudspeakerSetup*)

- **soft_clip** (*bool, optional*) – Limit the compensation boost to +6dB.

Returns **ls_sigs_compensated** (*(L, S) np.ndarray*) – Compensated loudspeaker signals.

References

Tervo, S., et. al. (2015). Spatial Analysis and Synthesis of Car Audio System and Car Cabin Acoustics with a Compact Microphone Array. Journal of the Audio Engineering Society.

spaudiopy.sdm.**post_equalization2**(*ls_sigs*, *sdm_p*, *fs*, *ls_setup*, *blocksize=4096*, *smoothing_order=5*)

Post equalization to compensate spectral whitening. This alternative version works on fixed blocksizes with octave band gain smoothing. Sonically, this seems not the preferred version, but it can gain some insight through the band gains which are returned.

Parameters

- **ls_sigs** (*(L, S) np.ndarray*) – Input loudspeaker signals.
- **sdm_p** (*array_like*) – Reference (sdm) pressure signal.
- **fs** (*int*)
- **ls_setup** (*decoder.LoudspeakerSetup*)
- **blocksize** (*int*)
- **smoothing_order** (*int*) – Block smoothing, increasing Hanning window up to this order.

Returns

- **ls_sigs_compensated** (*(L, S) np.ndarray*) – Compensated loudspeaker signals.
- **band_gains_list** (*list*) – Each element contains the octave band gain applied as post eq.

2.9 spaudiopy.plots

Plotting helpers.

```
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['axes.grid'] = True

import spaudiopy as spa
```

Functions

compare_ambi(Ambi_A, Ambi_B)	Compare A and B format signals.
decoder_performance(hull, renderer_type[, ...])	Shows amplitude, energy, spread and angular error measures on grid.
doa(azi, colat, fs[, p, size])	Direction of Arrival, with optional p(t) scaling the size.
freq_resp(freq, amp[, to_db, smoothing_n, ...])	Plot magnitude of frequency response over time frequency f.
hrir_ild_itd(hrirs[, plevels, fig])	Plot HRIR ILDs and ITDs.
hull(hull[, simplices, mark_invalid, title, ...])	Plot loudspeaker setup and valid simplices from its hull object.
hull_normals(hull[, plot_face_normals, ...])	Plot loudspeaker setup with vertex and face normals.
polar(theta, r[, INDB, rlim, title, ax])	Polar plot (in dB) that allows negative values for *r*.
set_aspect_equal3d([ax, XYZlim])	Set 3D axis to equal aspect.
sh_coeffs(F_nm[, SH_type, azi_steps, ...])	Plot spherical harmonics coefficients as function on the sphere.
sh_coeffs_overlay(F_nm_list[, SH_type, ...])	Overlay spherical harmonics coefficients plots.
sh_coeffs_subplot(F_nm_list[, SH_type, ...])	Plot spherical harmonics coefficients list as function on the sphere.
sh_rms_map(F_nm[, INDB, w_n, SH_type, ...])	Plot spherical harmonic signal RMS as function on the sphere.
spectrum(x, fs[, ylim, scale_mag])	Positive (single sided) amplitude spectrum of time signal x.
spherical_function(f, azi, colat[, title, fig])	Plot function 1D vector f over azi and colat.
transfer_function(freq, H[, title, xlim])	Plot transfer function H (magnitude and phase) over time frequency f.
zeropole(b, a[, zPlane, title])	Plot Zero Pole diagram from a and b coefficients.

spaudiopy.plots.**spectrum**(*x*, *fs*, *ylim=None*, *scale_mag=False*, ***kwargs*)

Positive (single sided) amplitude spectrum of time signal x. kwargs are forwarded to plots.freq_resp().

Parameters

- **x** (*np.array, list of np.array*) – Time domain signal.
- **fs** (*int*) – Sampling frequency.

spaudiopy.plots.**freq_resp**(*freq*, *amp*, *to_db=True*, *smoothing_n=None*, *title=None*, *labels=None*, *xlim=(20, 24000)*, *ylim=(- 30, None)*)

Plot magnitude of frequency response over time frequency f.

Parameters

- **f** (*frequency array*)
- **amp** (*array_like, list of array_like*)

Examples

See *spaudiopy.sph.binaural_coloration_compensation()*

spaudiopy.plots.**transfer_function**(*freq*, *H*, *title=None*, *xlim=(10, 25000)*)

Plot transfer function H (magnitude and phase) over time frequency f.

spaudiopy.plots.**zeropole**(*b*, *a*, *zPlane=False*, *title=None*)

Plot Zero Pole diagram from a and b coefficients.

spaudiopy.plots.**compare_ambi**(*Ambi_A*, *Ambi_B*)

Compare A and B format signals.

spaudiopy.plots.**spherical_function**(*f*, *azi*, *colat*, *title=None*, *fig=None*)

Plot function 1D vector f over azi and colat.

spaudiopy.plots.**sh_coeffs**(*F_nm*, *SH_type=None*, *azi_steps=5*, *el_steps=3*, *title=None*, *fig=None*)

Plot spherical harmonics coefficients as function on the sphere. Evaluates the inverse SHT.

Examples

See *spaudiopy.sph*

spaudiopy.plots.**sh_coeffs_overlay**(*F_nm_list*, *SH_type=None*, *azi_steps=5*, *el_steps=3*, *title=None*, *fig=None*)

Overlay spherical harmonics coefficients plots.

Examples

See *spaudiopy.plots.sh_coeffs*

spaudiopy.plots.**sh_coeffs_subplot**(*F_nm_list*, *SH_type=None*, *azi_steps=5*, *el_steps=3*, *titles=None*, *fig=None*)

Plot spherical harmonics coefficients list as function on the sphere.

Examples

See *spaudiopy.sph*

spaudiopy.plots.**sh_rms_map**(*F_nm*, *INDB=False*, *w_n=None*, *SH_type=None*, *azi_steps=5*, *zen_steps=3*, *title=None*, *fig=None*)

Plot spherical harmonic signal RMS as function on the sphere. Evaluates the maxDI beamformer, if w_n is None.

Parameters

- **F_nm** (*((N+1)**2, S) numpy.ndarray*) – Matrix of spherical harmonics coefficients, Ambisonic signal.
- **INDB** (*bool*) – Plot in dB.
- **w_n** (*array_like*) – Modal weighting of beamformers that are evaluated on the grid.
- **SH_type** (*'complex' or 'real' spherical harmonics.*)

Examples

See *spaudiopy.sph.src_to_sh*

spaudiopy.plots.**hull**(*hull*, *simplices=None*, *mark_invalid=True*, *title=None*, *draw_ls=True*, *ax_lim=None*, *color=None*, *clim=None*, *fig=None*)

Plot loudspeaker setup and valid simplices from its hull object.

Parameters

- **hull** (*decoder.LoudspeakerSetup*)
- **simplices** (*optional*)
- **mark_invalid** (*bool, optional*) – mark invalid simplices from hull object.
- **title** (*string, optional*)
- **draw_ls** (*bool, optional*)
- **ax_lim** (*float, optional*) – Axis limits in m.
- **color** (*array_like, optional*) – Custom colors for simplices.
- **clim** (*(2,), optional*) – *vmin* and *vmax* for colors.

Examples

See *spaudiopy.decoder*

spaudiopy.plots.**hull_normals**(*hull*, *plot_face_normals=True*, *plot_vertex_normals=True*)

Plot loudspeaker setup with vertex and face normals.

spaudiopy.plots.**polar**(*theta*, *r*, *INDB=True*, *rlim=None*, *title=None*, *ax=None*)

Polar plot (in dB) that allows negative values for *r*.

Examples

See *spaudiopy.sph.bandlimited_dirac()*

spaudiopy.plots.**decoder_performance**(*hull*, *renderer_type*, *azi_steps=5*, *ele_steps=3*, *show_ls=True*, *title=None*, ***kwargs*)

Shows amplitude, energy, spread and angular error measures on grid. For renderer_type={'VBAP', 'VBIP', 'ALLRAP', 'NLS'}, as well as {'ALLRAD', 'ALLRAD2', 'EPAD', 'MAD', 'SAD'}. All kwargs are forwarded to the decoder function.

References

Zotter, F., & Frank, M. (2019). Ambisonics. Springer Topics in Signal Processing.

Examples

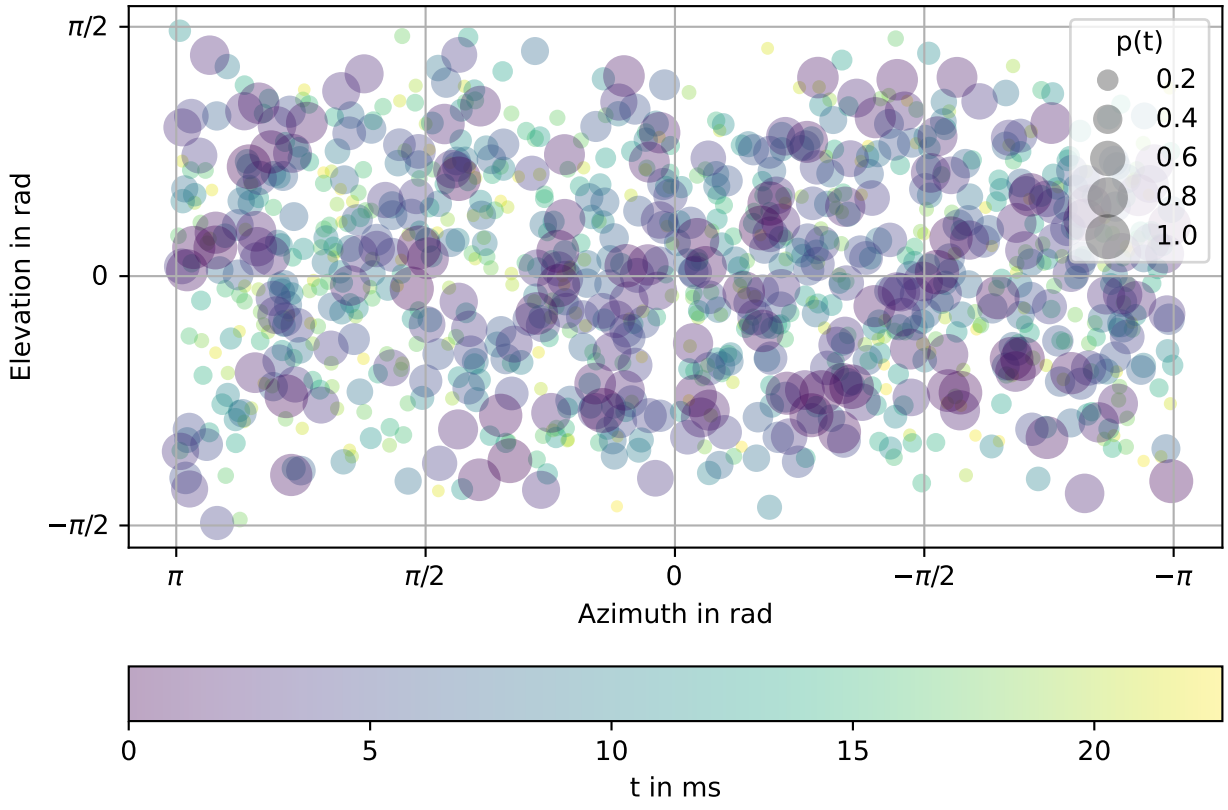
See *spaudiopy.decoder*

spaudiopy.plots.**doa**(*azi*, *colat*, *fs*, *p=None*, *size=250*)

Direction of Arrival, with optional p(t) scaling the size.

Examples

```
n = 1000
fs = 44100
t_ms = np.linspace(0, n/fs, n, endpoint=False) * 1000  # t in ms

x = np.random.randn(n)
y = np.random.randn(n)
z = np.random.randn(n)
azi, colat, r = spa.utils.cart2sph(x, y, z)

ps = 1 / np.exp(np.linspace(0, 3, n))
spa.plots.doa(azi, colat, fs, ps)
```



spaudiopy.plots.**hrir_ild_itd**(*hrirs*, *plevels=50*, *fig=None*)

Plot HRIR ILDs and ITDs.

Parameters

- **hrirs** (*sig.HRIRs*)
- **plevels** (*int, optional*) – Contour levels. The default is 50.
- **fig** (*plt.figure, optional*)

Returns *None.*

See also:

spaudiopy.process.ilds_from_hrirs Calculating ILDs with defaults (in dB).

spaudiopy.process.itds_from_hrirs Calculating ITDs with defaults.

spaudiopy.plots.**set_aspect_equal3d**(*ax=None*, *XYZlim=None*)

Set 3D axis to equal aspect.

Parameters

- **ax** (*axis, optional*) – ax object. The default is None.
- **XYZlim** (*[min, max], optional*) – min and max in m. The default is None.

Returns *None.*

2.10 Multiprocessing

If the function has an argument called *jobs_count* the implementation allows launching multiple processing jobs. Keep in mind that there is always a certain computational overhead involved, and more jobs is not always faster.

Especially on Windows, you then have to protect your *main()* function with:

```
if __name__ == '__main__':
    # Put your code here
```

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

S

spaudiopy, 3
spaudiopy.decoder, 28
spaudiopy.grids, 60
spaudiopy.IO, 3
spaudiopy.plots, 70
spaudiopy.process, 51
spaudiopy.sdm, 68
spaudiopy.sig, 7
spaudiopy.sph, 10
spaudiopy.utils, 58

INDEX

Symbols

__init__() (*spaudiopy.decoder.LoudspeakerSetup method*), 30
__init__() (*spaudiopy.sig.AmbiBSignal method*), 8
__init__() (*spaudiopy.sig.HRIRs method*), 9
__init__() (*spaudiopy.sig.MonoSignal method*), 7
__init__() (*spaudiopy.sig.MultiSignal method*), 8

A

allrad() (*in module spaudiopy.decoder*), 41
allrad2() (*in module spaudiopy.decoder*), 42
allrap() (*in module spaudiopy.decoder*), 39
allrap2() (*in module spaudiopy.decoder*), 39
ambeo_a2b() (*in module spaudiopy.process*), 53
AmbiBSignal (*class in spaudiopy.sig*), 8
ambisonics_setup() (*spaudiopy.decoder.LoudspeakerSetup method*), 31
angle_between() (*in module spaudiopy.utils*), 59
apply() (*spaudiopy.sig.AmbiBSignal method*), 8
apply() (*spaudiopy.sig.MonoSignal method*), 7
apply() (*spaudiopy.sig.MultiSignal method*), 8
apply_blacklist() (*in module spaudiopy.decoder*), 33
area_triangle() (*in module spaudiopy.utils*), 59
asarray_1d() (*in module spaudiopy.utils*), 58

B

b_to_sh() (*in module spaudiopy.sph*), 14
b_to_stereo() (*in module spaudiopy.process*), 53
bandlimited_dirac() (*in module spaudiopy.sph*), 15
binaural_coloration_compensation() (*in module spaudiopy.sph*), 20
binauralize() (*spaudiopy.decoder.LoudspeakerSetup method*), 31
butterworth_modal_weights() (*in module spaudiopy.sph*), 25

C

calculate_barycenter() (*in module spaudiopy.decoder*), 33
calculate_centroids() (*in module spaudiopy.decoder*), 33
calculate_face_areas() (*in module spaudiopy.decoder*), 33
calculate_face_normals() (*in module spaudiopy.decoder*), 33
calculate_grid_weights() (*in module spaudiopy.sph*), 13
calculate_vertex_normals() (*in module spaudiopy.decoder*), 33
cardioid_modal_weights() (*in module spaudiopy.sph*), 22
cart2sph() (*in module spaudiopy.utils*), 58
characteristic_ambisonic_order() (*in module spaudiopy.decoder*), 37
check_aperture() (*in module spaudiopy.decoder*), 33
check_cond_sht() (*in module spaudiopy.sph*), 13
check_listener_inside() (*in module spaudiopy.decoder*), 33
check_normals() (*in module spaudiopy.decoder*), 33
check_opening() (*in module spaudiopy.decoder*), 33
compare_ambi() (*in module spaudiopy.plots*), 72
conv() (*spaudiopy.sig.AmbiBSignal method*), 9
conv() (*spaudiopy.sig.MonoSignal method*), 7
conv() (*spaudiopy.sig.MultiSignal method*), 8
copy() (*spaudiopy.sig.AmbiBSignal method*), 9
copy() (*spaudiopy.sig.MonoSignal method*), 7
copy() (*spaudiopy.sig.MultiSignal method*), 8

D

db() (*in module spaudiopy.utils*), 59
decoder_performance() (*in module spaudiopy.plots*), 73
deg2rad() (*in module spaudiopy.utils*), 58
design_spat_filterbank() (*in module spaudiopy.sph*), 27
doa() (*in module spaudiopy.plots*), 74

E

energy_decay() (*in module spaudiopy.process*), 56
epad() (*in module spaudiopy.decoder*), 44
equal_angle() (*in module spaudiopy.grids*), 65
equal_polar_angle() (*in module spaudiopy.grids*), 67

F

find_imaginary_loudspeaker() (*in module spaudiopy.decoder*), 33
frac_octave_filterbank() (*in module spaudiopy.process*), 53
freq_resp() (*in module spaudiopy.plots*), 71
from_db() (*in module spaudiopy.utils*), 59
from_file() (*spaudiopy.sig.AmbiBSignal class method*), 8
from_file() (*spaudiopy.sig.MonoSignal class method*), 7
from_file() (*spaudiopy.sig.MultiSignal class method*), 8
from_sph() (*spaudiopy.decoder.LoudspeakerSetup class method*), 31

G

gain_clipping() (*in module spaudiopy.process*), 56
gauss() (*in module spaudiopy.grids*), 66
get_characteristic_order() (*spaudiopy.decoder.LoudspeakerSetup method*), 31
get_default_hrirs() (*in module spaudiopy.IO*), 4
get_hull() (*in module spaudiopy.decoder*), 33
get_signals() (*spaudiopy.sig.AmbiBSignal method*), 9
get_signals() (*spaudiopy.sig.MultiSignal method*), 8

H

half_sided_Hann() (*in module spaudiopy.process*), 56
haversine() (*in module spaudiopy.utils*), 59
hrir_ild_itd() (*in module spaudiopy.plots*), 74
HRIRs (*class in spaudiopy.sig*), 9
hull() (*in module spaudiopy.plots*), 73
hull_normals() (*in module spaudiopy.plots*), 73
hypercardioid_modal_weights() (*in module spaudiopy.sph*), 22

I

ilds_from_hrirs() (*in module spaudiopy.process*), 52
interleave_channels() (*in module spaudiopy.utils*), 60
inverse_sht() (*in module spaudiopy.sph*), 13
is_simplex_valid() (*spaudiopy.decoder.LoudspeakerSetup method*), 31
itds_from_hrirs() (*in module spaudiopy.process*), 53

L

lagrange_delay() (*in module spaudiopy.process*), 53
load_audio() (*in module spaudiopy.IO*), 4
load_Fliege_Maier_nodes() (*in module spaudiopy.grids*), 63
load_hrirs() (*in module spaudiopy.IO*), 4
load_layout() (*in module spaudiopy.IO*), 6
load_lebedev() (*in module spaudiopy.grids*), 62
load_maxDet() (*in module spaudiopy.grids*), 64
load_n_design() (*in module spaudiopy.grids*), 61
load_sdm() (*in module spaudiopy.IO*), 5
load_sofa_data() (*in module spaudiopy.IO*), 5
load_sofa_hrirs() (*in module spaudiopy.IO*), 5
load_t_design() (*in module spaudiopy.grids*), 60
loudspeaker_signals() (*spaudiopy.decoder.LoudspeakerSetup method*), 33
LoudspeakerSetup (*class in spaudiopy.decoder*), 30

M

mad() (*in module spaudiopy.decoder*), 44
match_loudness() (*in module spaudiopy.process*), 53
matlab_sph2cart() (*in module spaudiopy.utils*), 59
max_rE_weights() (*in module spaudiopy.sph*), 17
maxre_modal_weights() (*in module spaudiopy.sph*), 22
mode_strength() (*in module spaudiopy.sph*), 19
module
    spaudiopy, 3
    spaudiopy.decoder, 28
    spaudiopy.grids, 60
    spaudiopy.IO, 3
    spaudiopy.plots, 70
    spaudiopy.process, 51
    spaudiopy.sdm, 68
    spaudiopy.sig, 7
    spaudiopy.sph, 10
    spaudiopy.utils, 58
MonoSignal (*class in spaudiopy.sig*), 7
MultiSignal (*class in spaudiopy.sig*), 8

N

n3d_to_sn3d() (*in module spaudiopy.sph*), 14
nearest() (*spaudiopy.sig.HRIRs method*), 9
nearest_hrirs() (*spaudiopy.sig.HRIRs method*), 9
nearest_loudspeaker() (*in module spaudiopy.decoder*), 47

P

platonic_solid() (*in module spaudiopy.sph*), 14
play() (*spaudiopy.sig.AmbiBSignal method*), 9
play() (*spaudiopy.sig.MonoSignal method*), 7
play() (*spaudiopy.sig.MultiSignal method*), 8
polar() (*in module spaudiopy.plots*), 73
pop_triangles() (*spaudiopy.decoder.LoudspeakerSetup method*), 31
post_equalization() (*in module spaudiopy.sdm*), 69
post_equalization2() (*in module spaudiopy.sdm*), 70

pressure_on_sphere() (*in module spaudiopy.sph*), 20
project_on_sphere() (*in module spaudiopy.sph*), 19
pseudo_intensity() (*in module spaudiopy.sdm*), 68
pulsed_noise() (*in module spaudiopy.process*), 56

R

r_E() (*in module spaudiopy.sph*), 18
rad2deg() (*in module spaudiopy.utils*), 58
render_binaural_loudspeaker_sdm() (*in module spaudiopy.sdm*), 69
render_bsdm() (*in module spaudiopy.sdm*), 68
render_stereo_sdm() (*in module spaudiopy.sdm*), 69
repeat_per_order() (*in module spaudiopy.sph*), 19
resample_hrirs() (*in module spaudiopy.process*), 51
resample_signal() (*in module spaudiopy.process*), 52
resample_spectrum() (*in module spaudiopy.process*), 52
rms() (*in module spaudiopy.utils*), 59

S

sad() (*in module spaudiopy.decoder*), 43
save() (*spaudiopy.sig.AmbiBSignal method*), 9
save() (*spaudiopy.sig.MonoSignal method*), 7
save() (*spaudiopy.sig.MultiSignal method*), 8
save_audio() (*in module spaudiopy.IO*), 4
save_layout() (*in module spaudiopy.IO*), 6
set_aspect_equal3d() (*in module spaudiopy.plots*), 75
sh2bin() (*in module spaudiopy.decoder*), 50
sh_coeffs() (*in module spaudiopy.plots*), 72
sh_coeffs_overlay() (*in module spaudiopy.plots*), 72
sh_coeffs_subplot() (*in module spaudiopy.plots*), 72
sh_matrix() (*in module spaudiopy.sph*), 12
sh_rms_map() (*in module spaudiopy.plots*), 72
sh_to_b() (*in module spaudiopy.sph*), 14
sh_to_b() (*spaudiopy.sig.AmbiBSignal class method*), 8
show() (*spaudiopy.decoder.LoudspeakerSetup method*), 33
sht() (*in module spaudiopy.sph*), 12
sht_lstsq() (*in module spaudiopy.sph*), 12
sn3d_to_n3d() (*in module spaudiopy.sph*), 14
sofa_to_sh() (*in module spaudiopy.IO*), 5
sort_vertices() (*in module spaudiopy.decoder*), 33
soundfield_to_b() (*in module spaudiopy.sph*), 14
spat_filterbank_reconstruction_factor() (*in module spaudiopy.sph*), 26
spaudiopy
    module, 3
spaudiopy.decoder
    module, 28
spaudiopy.grids
    module, 60
spaudiopy.IO
    module, 3
spaudiopy.plots
    module, 70
spaudiopy.process
    module, 51
spaudiopy.sdm
    module, 68
spaudiopy.sig
    module, 7
spaudiopy.sph
    module, 10
spaudiopy.utils
    module, 58
spectrum() (*in module spaudiopy.plots*), 71
sph2cart() (*in module spaudiopy.utils*), 58
spherical_function() (*in module spaudiopy.plots*), 72
spherical_hn2() (*in module spaudiopy.sph*), 19
src_to_b() (*in module spaudiopy.sph*), 15
src_to_sh() (*in module spaudiopy.sph*), 15
stack() (*in module spaudiopy.utils*), 60
subband_levels() (*in module spaudiopy.process*), 54

T

test_diff() (*in module spaudiopy.utils*), 60
transfer_function() (*in module spaudiopy.plots*), 72
trim() (*spaudiopy.sig.AmbiBSignal method*), 9
trim() (*spaudiopy.sig.MonoSignal method*), 7
trim() (*spaudiopy.sig.MultiSignal method*), 8
trim_audio() (*in module spaudiopy.sig*), 9

U

unity_gain() (*in module spaudiopy.sph*), 21

V

vbap() (*in module spaudiopy.decoder*), 34
vbip() (*in module spaudiopy.decoder*), 34
vecs2dirs() (*in module spaudiopy.utils*), 59

W

write_ssr_brirs_loudspeaker() (*in module spaudiopy.IO*), 6
write_ssr_brirs_sdm() (*in module spaudiopy.IO*), 6

Z

zeropole() (*in module spaudiopy.plots*), 72