spaudiopy Documentation

Release v0.1.6

Chris Hold

Feb 05, 2024

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 . . . 11
- 2.4 spaudiopy.decoder . . . 28
- 2.5 spaudiopy.process . . . 52
- 2.6 spaudiopy.utils . . . 60
- 2.7 spaudiopy.grids . . . 62
- 2.8 spaudiopy.parsa . . . 71
- 2.9 spaudiopy.plot . . . 81
- 2.10 Multiprocessing . . . 87

3 Indices and tables **89**

Python Module Index **91**

Index **93**

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.
parsa	Parametric Spatial Audio (PARSA).
plot	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_zen, ...])	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, jobs_count])	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*, *jobs_count=None*)

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

Parameters

- **fs** (*int*) – fs(t).
- **filename** (*string, optional*) – HRTF.mat file or default set, or 'dummy' for debugging.
- **jobs_count** (*int or None, optional*) – Number of parallel jobs for resample_hrirs() in get_default_hrirs(), 'None' employs 'cpu_count'.

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.

azi
[(g,) array_like] grid azimuth.

zen
[(g,) array_like] grid zenith / colatitude.

fs
[int] fs(t).

spaudiopy.io.**get_default_hrirs**(*grid_azi=None*, *grid_zen=None*, *jobs_count=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_zen** (*array_like, optional*)
- **jobs_count** (*int or None, optional*) – Number of parallel jobs for resample_hrirs(), 'None' employs 'cpu_count'.

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.

azi
[(g,) array_like] grid azimuth.

zen
[(g,) array_like] grid zenith / colatitude.

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_zen' 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_zen** (*(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, azi, zen, 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 instance.

save(*filename*, *subtype='FLOAT'*)

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.

resample(*fs_new*)

Resample signal to new sampling rate fs_new.

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

Play sound signal. Adjust gain and wait until finished.

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

Bases: *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 (nCH, nSmps).

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.

resample(*fs_new*)

Resample signal to new sampling rate fs_new.

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

Play sound signal. Adjust gain and wait until finished.

copy()

Return an independent (deep) copy of the instance.

save(*filename*, *subtype='FLOAT'*)

Save to file.

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

Bases: *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 instance.

get_signals()

Return ndarray of signals, stacked along rows (nCH, nSmps).

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

Play sound signal. Adjust gain and wait until finished.

resample(*fs_new*)

Resample signal to new sampling rate fs_new.

save(*filename*, *subtype='FLOAT'*)

Save to file.

trim(*start*, *stop*)

Trim all channels to start and stop in seconds.

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

Bases: object

Signal class for head-related impulse responses.

__init__(*left*, *right*, *azi*, *zen*, *fs*)

Constructor.

Parameters

- **left** (*(numDirs, numTaps) ndarray*) – Left ear HRIRs.
- **right** (*(numDirs, numTaps) ndarray*) – Right ear HRIRs.
- **azi** (*(numDirs,) array_like, in rad*)
- **fs** (*int*)

copy()

Return an independent (deep) copy of the instance.

update_hrirs(*left*, *right*)

Update and replace HRIRs in place.

Parameters

- **left** (*(numDirs, numTaps) ndarray*) – Left ear HRIRs.
- **right** (*numDirs, numTaps ndarray*) – Right ear HRIRs.

Returns

None.

nearest_hrirs(*azi*, *zen*)

For a point on the sphere, select closest HRIR defined on grid.

Based on the haversine distance.

Parameters

- **azi** (*float*) – Azimuth.
- **zen** (*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_idx(*azi*, *zen*)

Index of nearest HRIR grid point based on dot product.

Parameters

- **azi** (*float, array_like*) – Azimuth.
- **zen** (*float, array_like*) – Zenith / Colatitude.

Returns

idx (*int, np.ndarray*) – Index.

apply_ctf_eq(*eq_taps=None*, *mode='full'*)

Equalize common transfer function (CTF) of HRIRs.

Parameters

- **eq_taps** (*array_like, optional*) – FIR filter, *None* will calculate. The default is None.
- **mode** (*string, optional*) – Forwarded to scipy.signal.convolve(). The default is 'full'.

Returns
None.

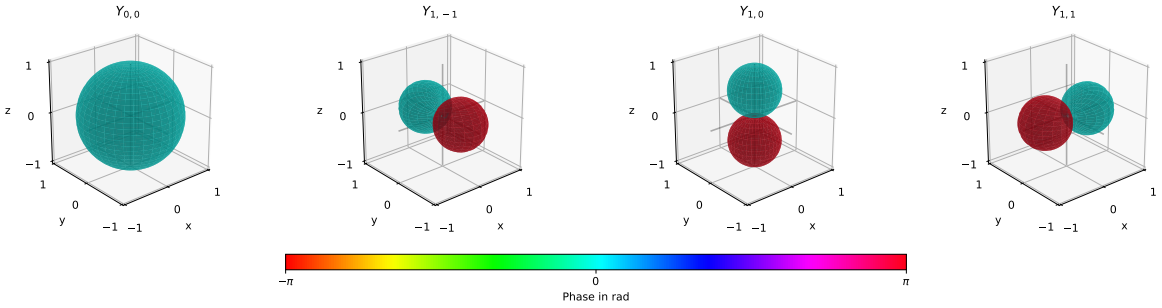
spaudiopy.sig.**trim_audio**(*A*, *start*, *stop*)

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.plot.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_sph, d[, w_n])	Order N spatially bandlimited Dirac pulse at central angle d.
binaural_coloration_compensation(N_sph, f[, ...])	Spectral equalization gain G(kr)|N for diffuse field of order N_sph.
butterworth_modal_weights(N_sph, k, n_c[, ...])	Modal weights for spatial butterworth filter / beamformer.
cardioid_modal_weights(N_sph)	Modal weights for beamformer resulting in a cardioid.
check_cond_sht(N_sph, azi, zen, sh_type[, lim])	Check if condition number for a least-squares SHT(N_sph) is high.

continues on next page

Table 1 – continued from previous page

design_sph_filterbank(N_sph, sec_azi, ...)	Design spherical filter bank analysis and reconstruction matrix.
hypercardioid_modal_weights(N_sph)	Modal weights for beamformer resulting in a hypercardioid.
inverse_sht(F_nm, azi, zen, sh_type[, ...])	Perform the inverse spherical harmonics transform.
max_rE_weights(N_sph)	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_sph, 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.
rotate_sh(F_nm, yaw, pitch, roll[, sh_type])	Rotate spherical harmonics coefficients.
sh_matrix(N_sph, azi, zen[, sh_type])	Evaluates the spherical harmonics up to order *N_sph* for given angles.
sh_mult(a_nm, b_nm, sh_type)	Multiply SH vector a_nm and b_nm in (discrete) space.
sh_rotation_matrix(N_sph, yaw, pitch, roll)	Computes a Wigner-D matrix for the rotation of spherical harmonics.
sh_to_b(F_nm[, W_weight])	Convert first order SH input signals to B-format.
sht(f, N_sph, azi, zen, sh_type[, weights, Y_nm])	Spherical harmonics transform of f for appropriate point sets.
sht_lstsq(f, N_sph, azi, zen, 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.
sph_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_zen)	Get B format signal channels for source in direction azi/zen.
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_sph*, *azi*, *zen*, *sh_type='real'*)

Evaluates the spherical harmonics up to order *N_sph* for given angles.

Matrix returns spherical harmonics up to order N evaluated at 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_sph** (*int*) – Maximum SH order.
- **azi** (*(Q,) array_like*) – Azimuth.
- **zen** (*(Q,) array_like*) – Colatitude / Zenith.
- **sh_type** (*'complex' or 'real' spherical harmonics.*)

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_sph*, *azi*, *zen*, *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/zen'.
- **N_sph** (*int*) – Maximum SH order.
- **azi** (*(Q,) array_like*) – Azimuth.
- **zen** (*(Q,) array_like*) – Colatitude / Zenith.
- **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_sph*, *azi*, *zen*, *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/zen'.
- **N_sph** (*int*) – Maximum SH order.
- **azi** (*(Q,) array_like*) – Azimuth.
- **zen** (*(Q,) array_like*) – Colatitude / Zenith.
- **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*, *zen*, *sh_type*, *N_sph=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.
- **zen** (*(Q,) array_like*) – Colatitude / Zenith.
- **sh_type** (*'complex' or 'real' spherical harmonics.*)
- **N_sph** (*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/zen'.

spaudiopy.sph.**sh_rotation_matrix**(*N_sph*, *yaw*, *pitch*, *roll*, *sh_type='real'*, *return_as_blocks=False*)

Computes a Wigner-D matrix for the rotation of spherical harmonics.

Parameters

- **N_sph** (*int*) – Maximum SH order.
- **yaw** (*float*) – Rotation around Z axis.
- **pitch** (*float*) – Rotation around Y axis.
- **roll** (*float*) – Rotation around X axis.
- **sh_type** (*'complex' or 'real' spherical harmonics.*) – Currently only 'real' is supported.
- **return_as_blocks** (*optional, default is False.*) – Return full block diagonal matrix, or a list of blocks if True.

Returns

R (*(..., (N_sph+1)**2, (N_sph+1)**2) numpy.ndarray*) – A block diagonal matrix R with shape (..., (N_sph+1)**2, (N_sph+1)**2), or a list r_blocks of numpy arrays [r(n) for n in range(N_sph)], where the shape of r is (..., 2*n-1, 2*n-1).

See also:

spaudiopy.sph.rotate_sh()

Apply rotation to SH signals.

References

Implemented according to: Ivanic, Joseph, and Klaus Ruedenberg. "Rotation matrices for real spherical harmonics. Direct determination by recursion." The Journal of Physical Chemistry 100.15 (1996): 6342-6347.

Ported from https://git.iem.at/audioplugins/IEMPluginSuite .

spaudiopy.sph.**rotate_sh**(*F_nm*, *yaw*, *pitch*, *roll*, *sh_type='real'*)

Rotate spherical harmonics coefficients.

Parameters

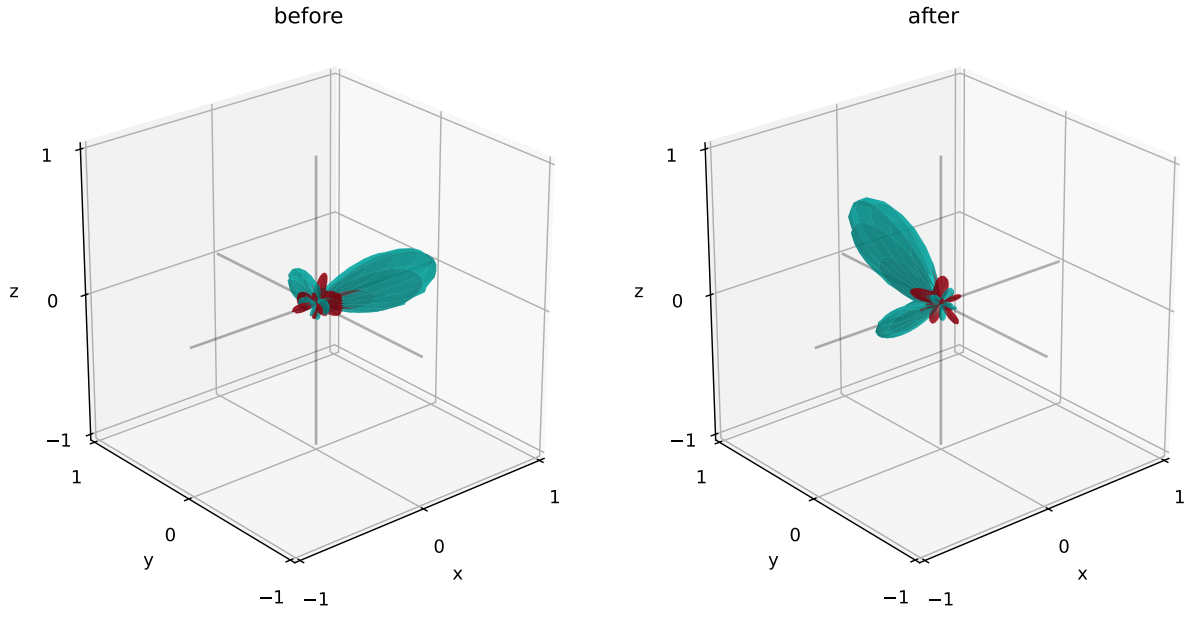
- **F_nm** (*(..., (N_sph+1)**2) numpy.ndarray*) – Spherical harmonics coefficients
- **yaw** (*float*) – Rotation around Z axis.
- **pitch** (*float*) – Rotation around Y axis.
- **roll** (*float*) – Rotation around X axis.
- **sh_type** (*'complex' or 'real' spherical harmonics.*) – Currently only 'real' is supported.

Returns

F_nm_rot (*(..., (N_sph+1)**2) numpy.ndarray*) – Rotated spherical harmonics coefficients.

Examples

```
N_sph = 5
y1 = spa.sph.sh_matrix(N_sph, 0, np.pi/2, 'real')
y2 = 0.5 * spa.sph.sh_matrix(N_sph, np.pi/3, np.pi/2, 'real')
y_org = (4 * np.pi) / (N_sph + 1)**2 * (y1 + y2)
y_rot = spa.sph.rotate_sh(y_org, -np.pi/2, np.pi/8, np.pi/4)

spa.plot.sh_coeffs_subplot([y_org, y_rot],
                           titles=['before', 'after'], cbar=False)
```



spaudiopy.sph.**check_cond_sht**(*N_sph*, *azi*, *zen*, *sh_type*, *lim=None*)

Check if condition number for a least-squares SHT(N_sph) is high.

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_zen*)

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

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.plot.sh_rms_map(sig_nm)
```

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

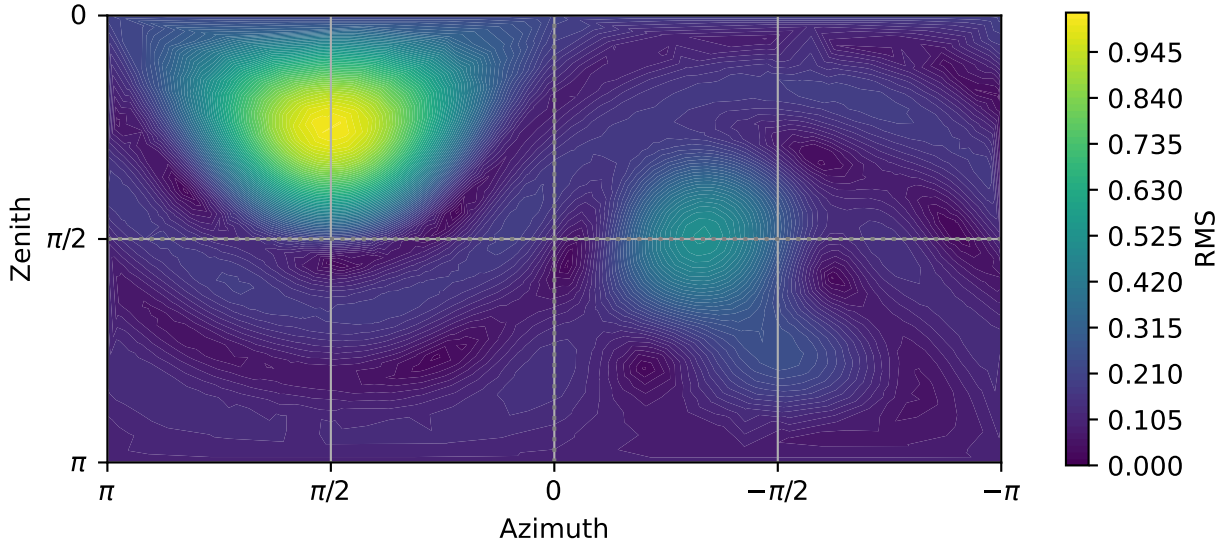
Order N spatially bandlimited Dirac pulse at central angle d.

Parameters

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

Returns

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


0
π/2
π
Zenith
π
π/2
0
−π/2
−π
Azimuth
0.945
0.840
0.735
0.630
0.525
0.420
0.315
0.210
0.105
0.000
RMS

Notes

Normalize with

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

References

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

Examples

```
dirac_azi = np.deg2rad(0)
dirac_zen = np.deg2rad(90)
N_sph = 5

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

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

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

spaudiopy.sph.**max_rE_weights**(*N_sph*)

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

See also:

spaudiopy.sph.unity_gain()
Unit amplitude compensation.

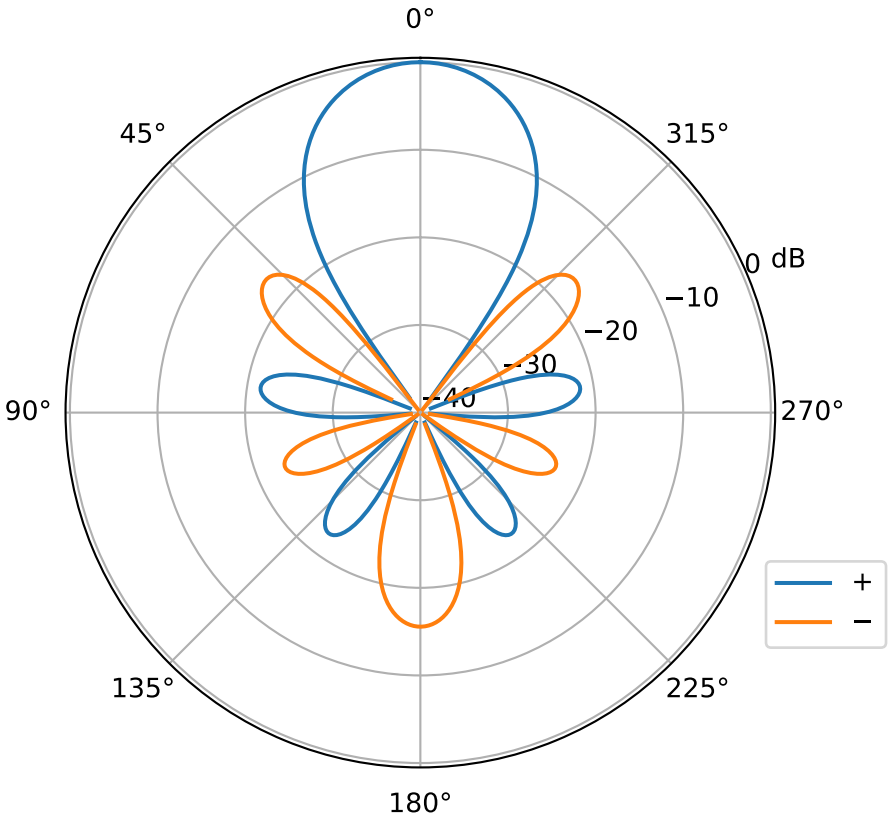
References

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

Examples

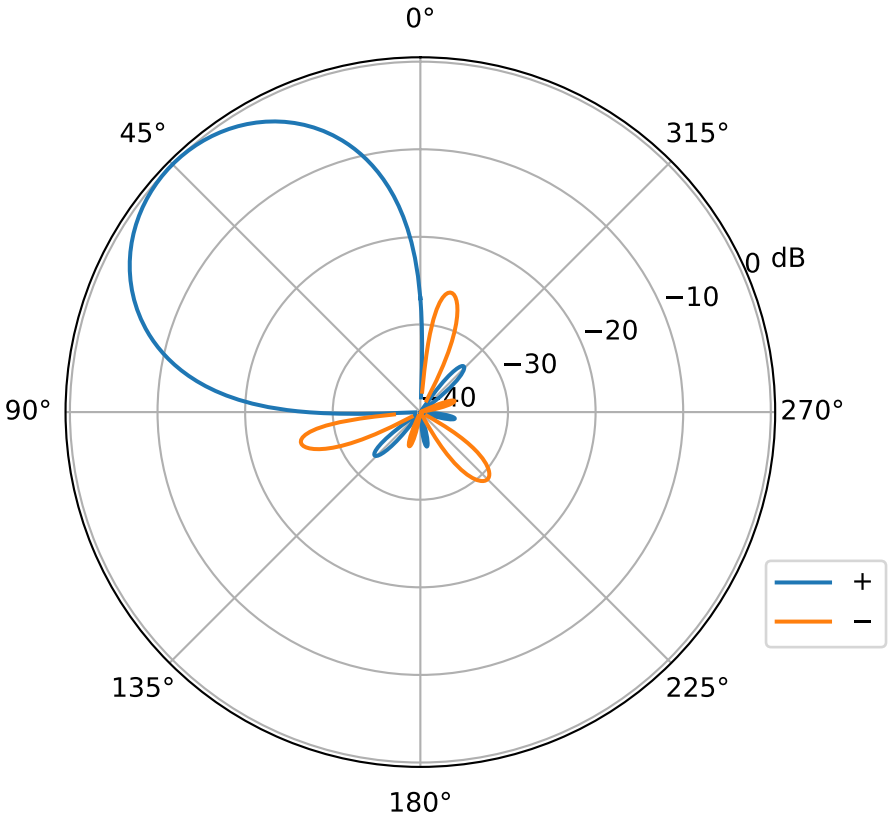
```
dirac_azi = np.deg2rad(45)
dirac_zen = np.deg2rad(45)
N = 5

# 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)
```

(continues on next page)


0°
45°
315°
90°
270°
135°
225°
180°
0 dB
−10
−20
−30
−40
+
−

(continued from previous page)

```
w_n = spa.sph.unity_gain(w_n)
dirac_tapered = spa.sph.bandlimited_dirac(N, azi - dirac_azi, w_n=w_n)

spa.plot.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_sph*, *kr*, *weights=None*)

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

Parameters

- **N_sph** (*int*) – SH order.
- **kr** (*array_like*) – kr vector, product of wavenumber k and radius r_0.
- **weights** (*(N_sph+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_sph*, *f*, *r_0=0.0875*, *w_taper=None*)

Spectral equalization gain G(kr)|N for diffuse field of order N_sph. 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_sph** (*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:

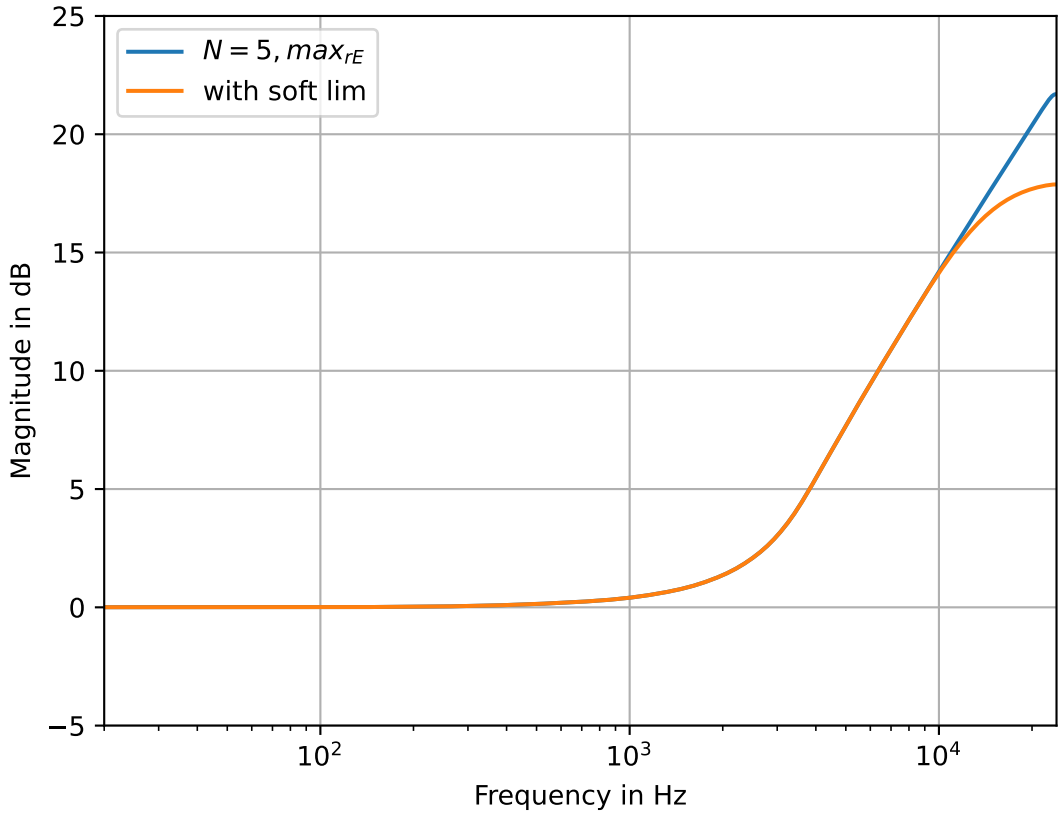
spaudiopy.process.gain_clipping()
Limit maximum gain.

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_sph = 5
# tapering window
w_rE = spa.sph.max_rE_weights(N)

compensation_tapered = spa.sph.binaural_coloration_compensation(
                            N_sph, f, w_taper=w_rE)
compensation_tapered_lim = spa.process.gain_clipping(
                            compensation_tapered,
                            spa.utils.from_db(12))
spa.plot.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_sph = 5
w_n = spa.sph.hypercardioid_modal_weights(N_sph)
w_nm = spa.sph.repeat_per_order(w_n) *              spa.sph.sh_matrix(N_sph, np.pi/4,
↪ np.pi/4, 'real')
spa.plot.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.

Notes

Also called *in-phase weights*, where $w_{n,cardioid} = 4\pi * w_{n,inphase}/(N+1)$.

Examples

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

Notes

Can be compensated for unit amplitude.

Examples

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

spaudiopy.sph.**sph_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 SH beamformers.
- **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_sph_filterbank**(*N_sph*, *sec_azi*, *sec_zen*, *c_n*, *sh_type*, *mode*)

Design spherical 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** (*((N+1)**2, J) 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_sph_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 @ s_sec

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

spaudiopy.sph.**sh_mult**(*a_nm*, *b_nm*, *sh_type*)

Multiply SH vector a_nm and b_nm in (discrete) space.

Parameters

- **a_nm** (*((N1+1)**2,), array_like*) – SH coefficients.
- **b_nm** (*((N2+1)**2,), array_like*) – SH coefficients.
- **sh_type** (*'real' or 'complex'*)

Returns

c_nm (*((N1+N2+1)**2,), array_like*) – SH coefficients.

Examples

```
spa.plot.sh_coeffs(4*spa.sph.sh_mult([1, 0, 1, 0], [0, 1, 0, 0],
                                     'real'))
```

2.4 spaudiopy.decoder

Loudspeaker decoders.

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

import spaudiopy as spa
```

(continues on next page)

(continued from previous page)

```
# 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.
magls_bin(hrirs, N_sph[, f_trans, hf_cont, ...])	Magnitude Least-Squares (magLS) binaural 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

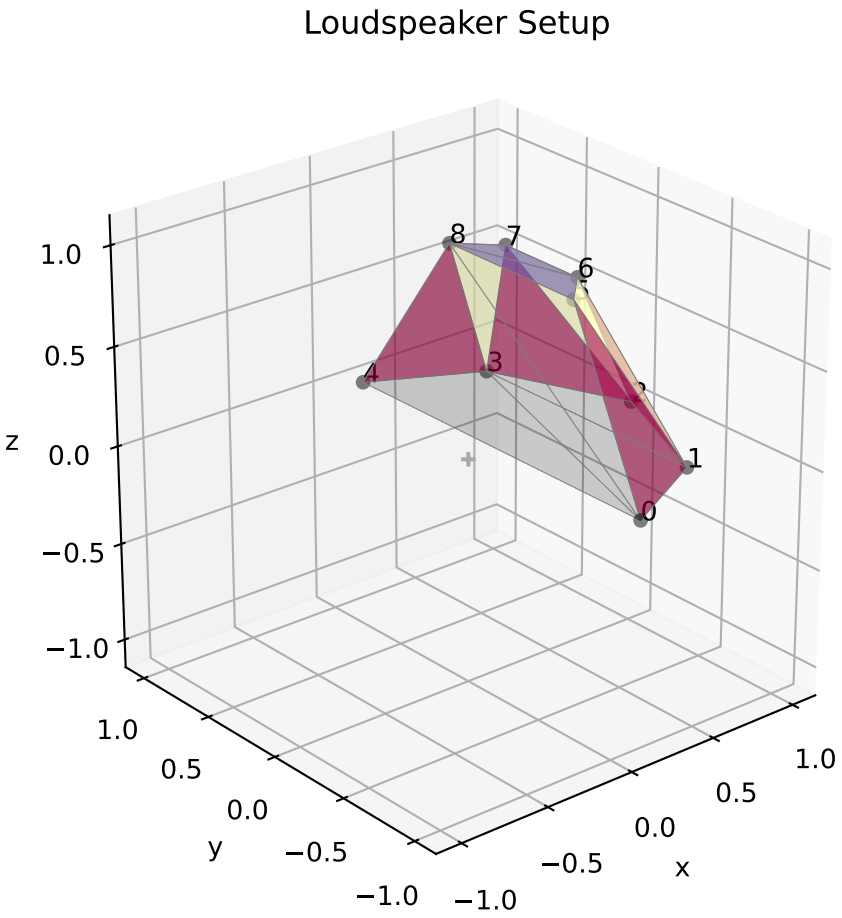
LoudspeakerSetup(x, y, z[, listener_position])	Creates a 'hull' object containing all information for further decoding.

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*, *zen*, *r=1*, *listener_position=None*)

Alternative constructor, using spherical coordinates in rad.

Parameters

- **azi** (*array_like, spherical*)
- **zen** (*array_like, spherical*)
- **r** (*array_like, spherical*)
- **listener_position** (*(azi, zen, 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`(*update_hull=False*, *imaginary_ls=None*, *characteristic_order=None*, *N_kernel=50*)

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

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

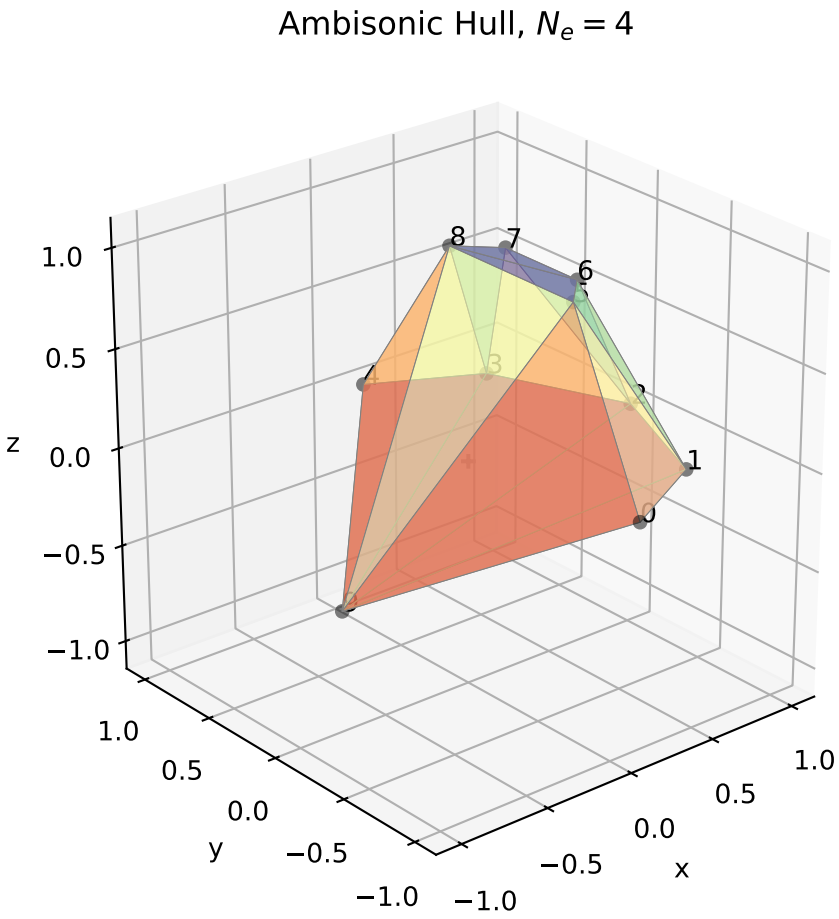
Examples

```
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


Ambisonic Hull, $N_e = 4$
8
7
6
1
0
1.0
0.5
0.0
−0.5
−1.0
z
y
x

- **ls_signals** (*(L, S) np.ndarray*) – Loudspeaker signals.
- **fs** (*int*)
- **orientation** (*(azi, zen) tuple, optional*) – Listener orientation offset (azimuth, zenith) 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 plot.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

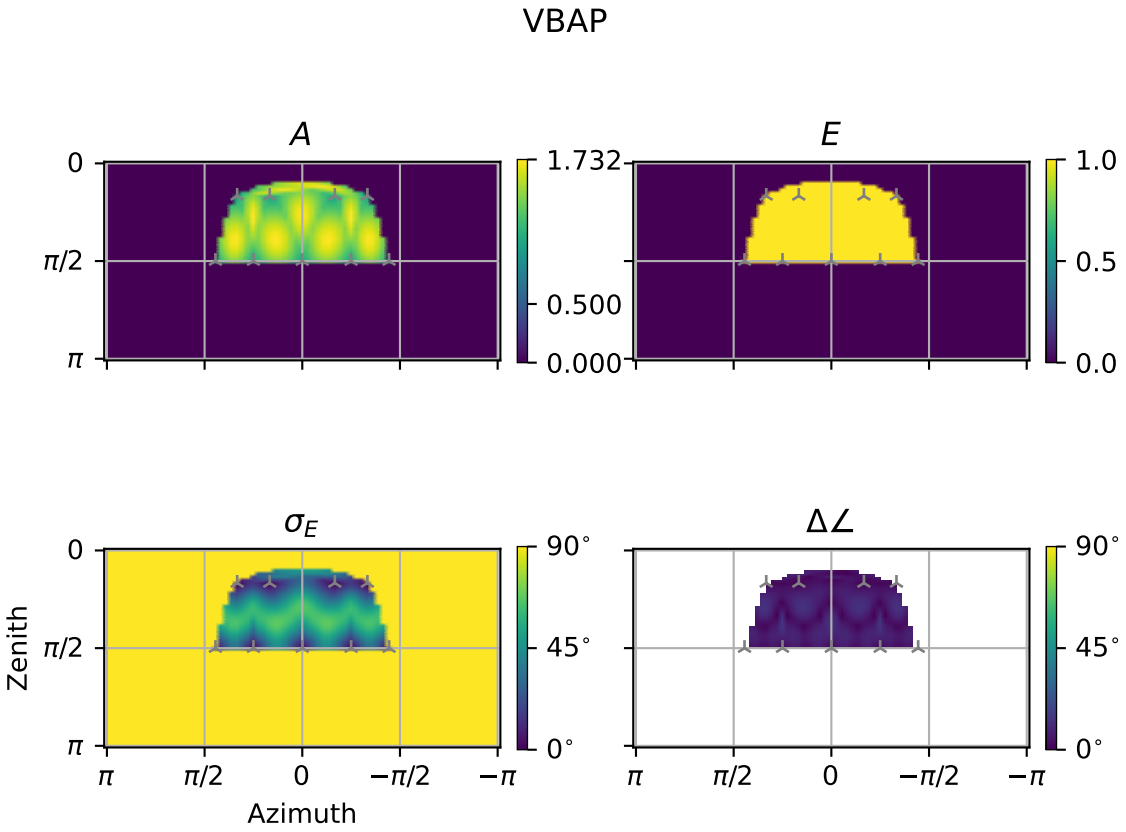
```
ls_setup = spa.decoder.LoudspeakerSetup(ls_x, ls_y, ls_z)
ls_setup.pop_triangles(normal_limit=85, aperture_limit=90,
                       opening_limit=150)

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

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

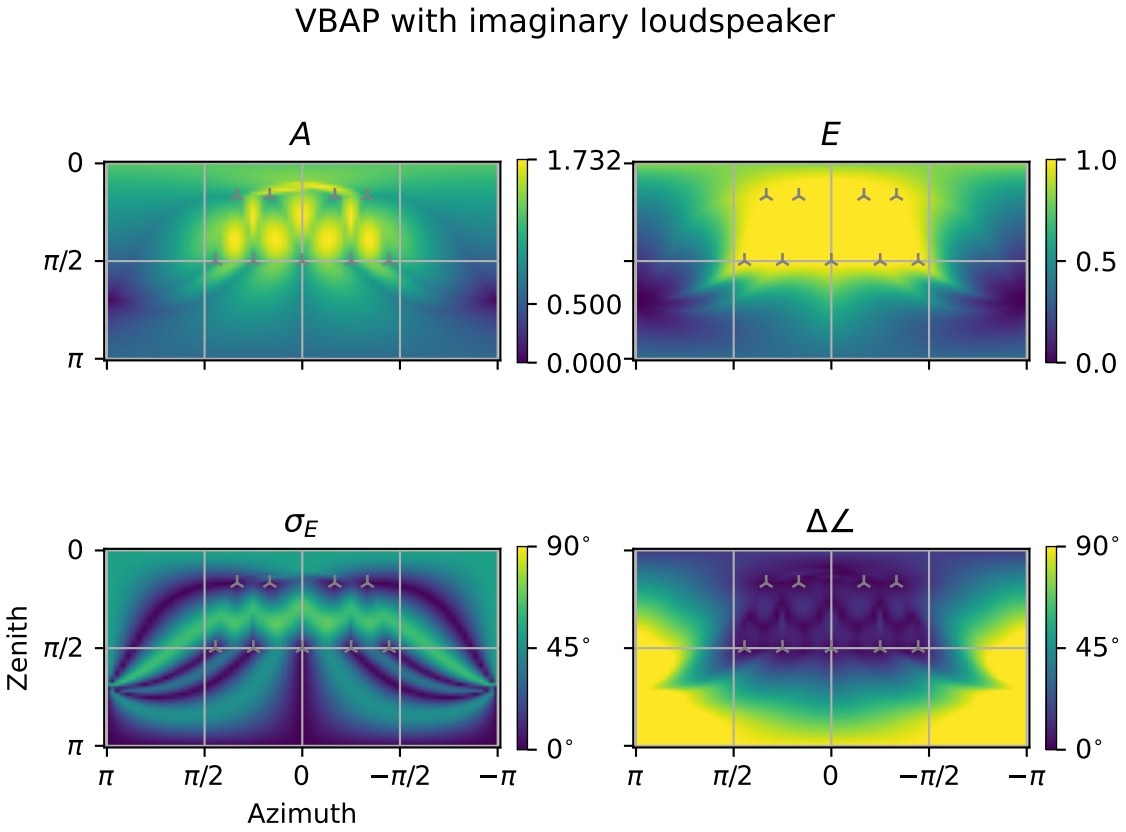
spaudiopy.decoder.**vbip**(*src*, *hull*, *norm=2*, *valid_simplices=None*, *retain_outside=False*, *jobs_count=1*)

Loudspeaker gains for Vector Base Intensity Panning decoding.

Parameters

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


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


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

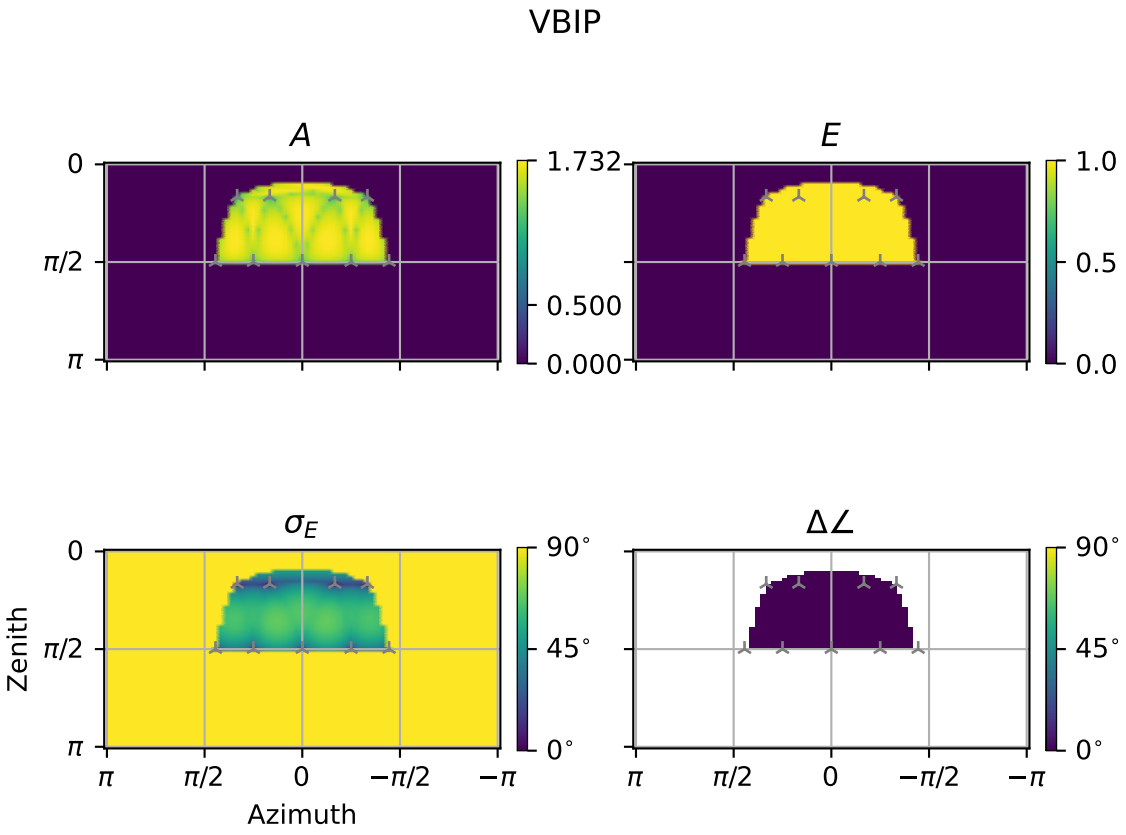
- **norm** (*non-zero int, float*) – Gain normalization norm, e.g. 1: anechoic, 2: reverberant
- **valid_simplices** (*(nsimplex, 3) numpy.ndarray*) – Valid simplices employed for rendering, defaults hull.valid_simplices.
- **retain_outside** (*bool, optional*) – Render on the 'ambisonic hull', amplitude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

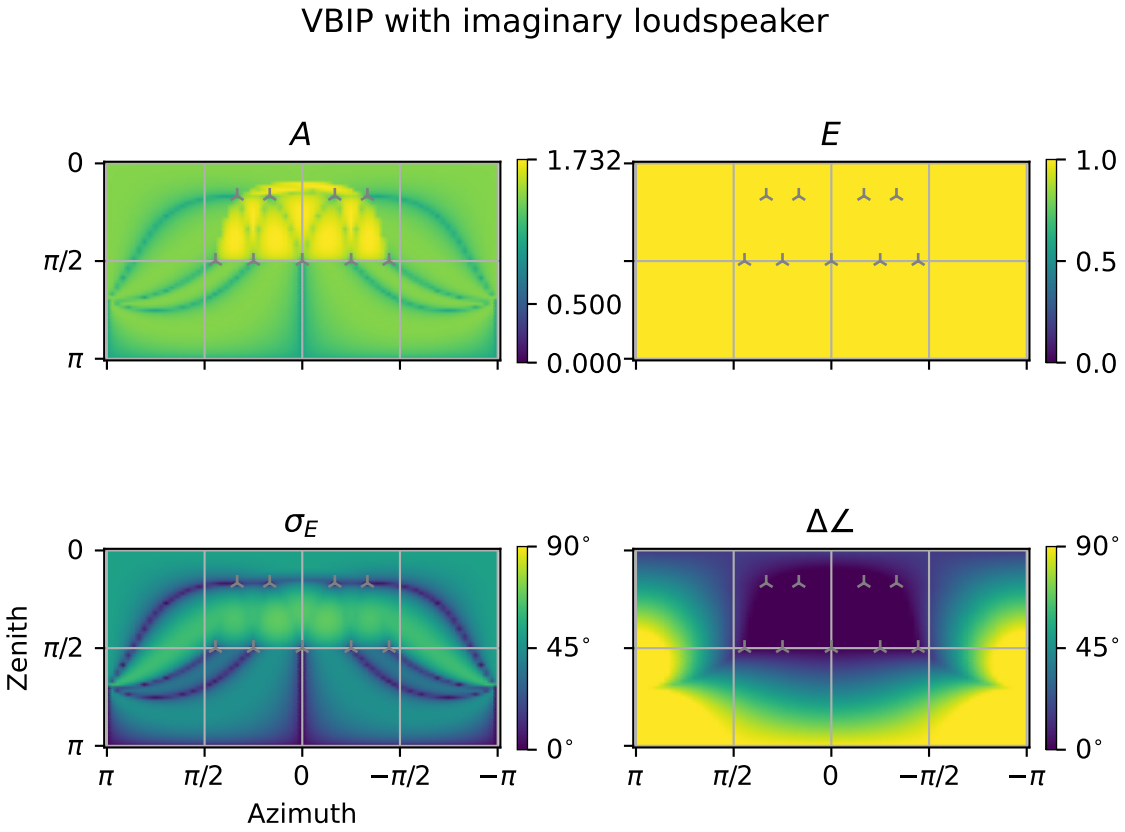
Returns

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.plot.decoder_performance(ls_setup, 'VBIP')

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




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

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

Find the characteristic order for specified loudspeaker layout.

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.plot.decoder_performance(ls_setup, 'ALLRAP')
```

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

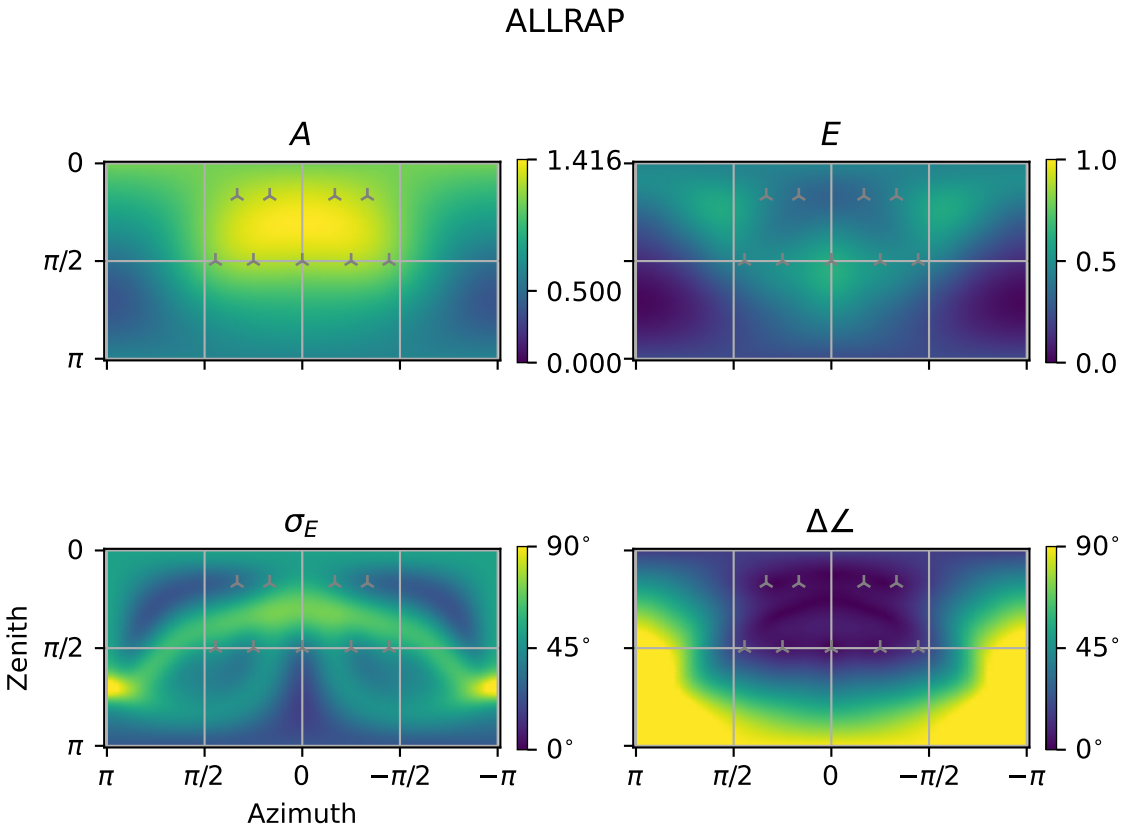
Loudspeaker gains for All-Round Ambisonic Panning 2.

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.

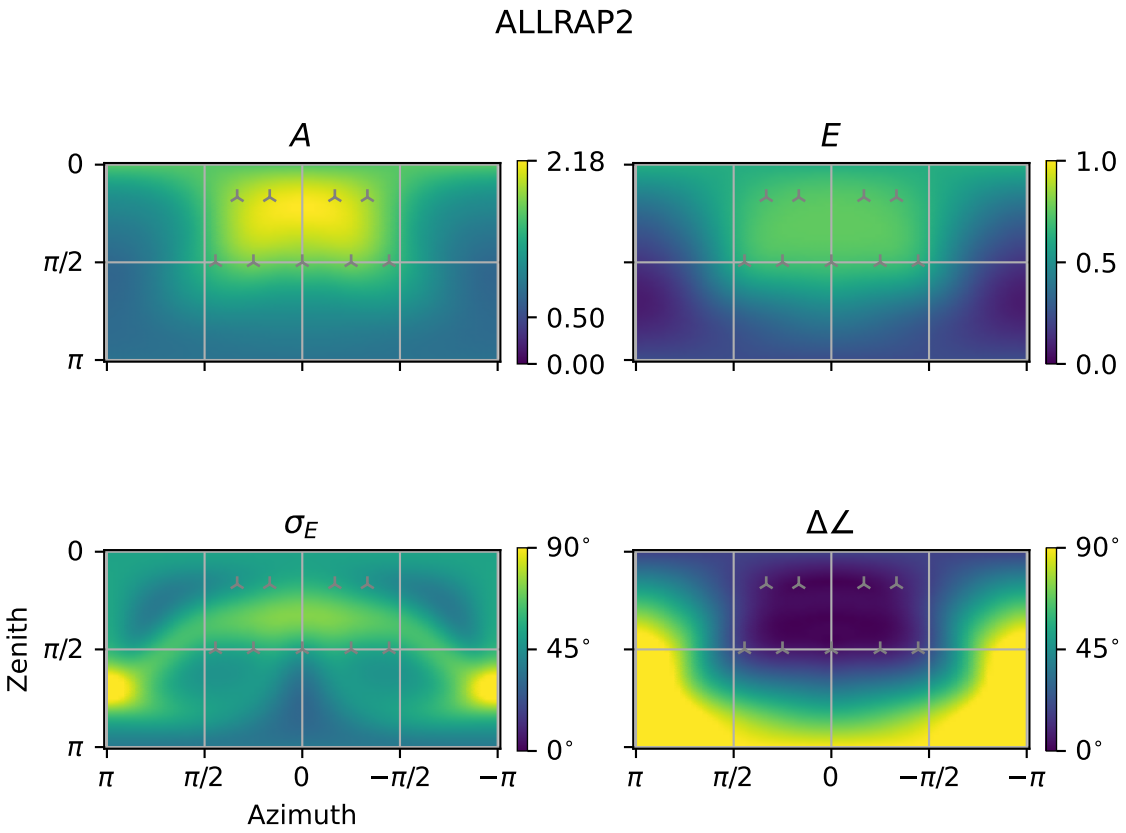

ALLRAP
A
E
σ_E
$\Delta\angle$
0
π/2
π
1.416
0.500
0.000
1.0
0.5
0.0
90°
45°
0°
Zenith
Azimuth
π
π/2
0
−π/2
−π

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.plot.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

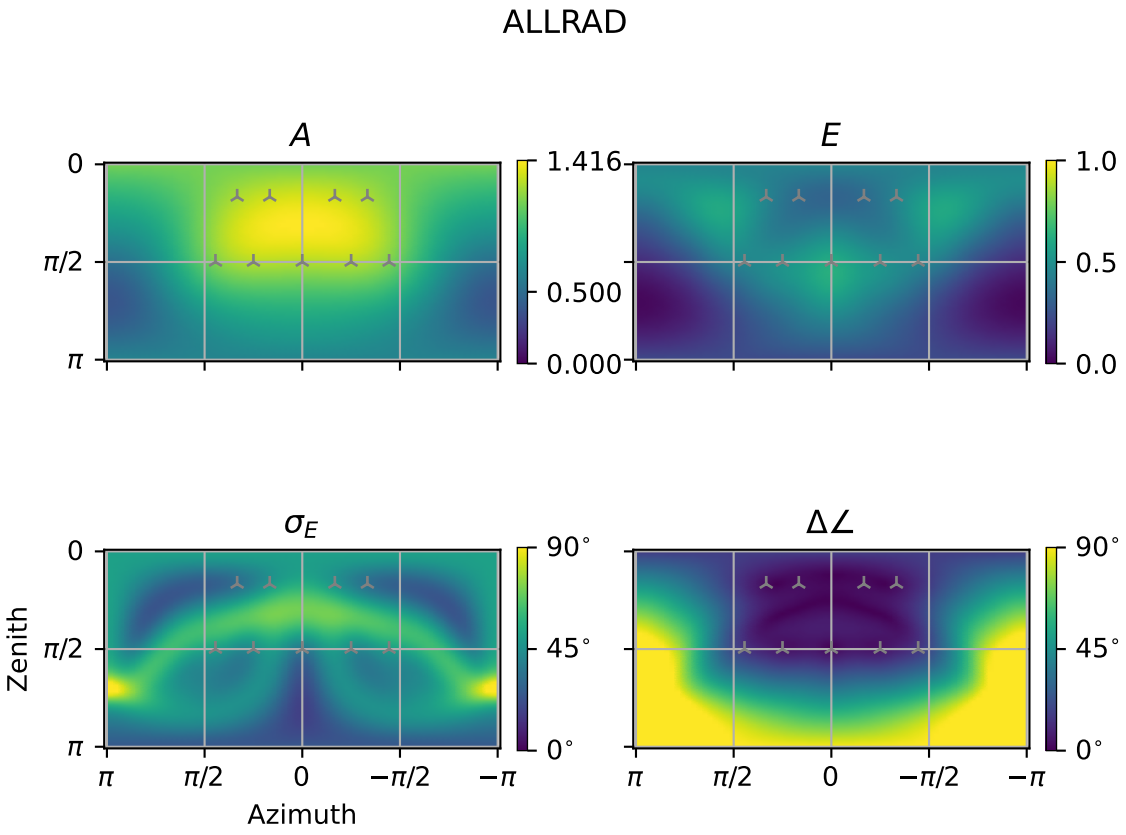
ls_sig (*(L, S) numpy.ndarray*) – Loudspeaker L output signal S.

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.plot.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.plot.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

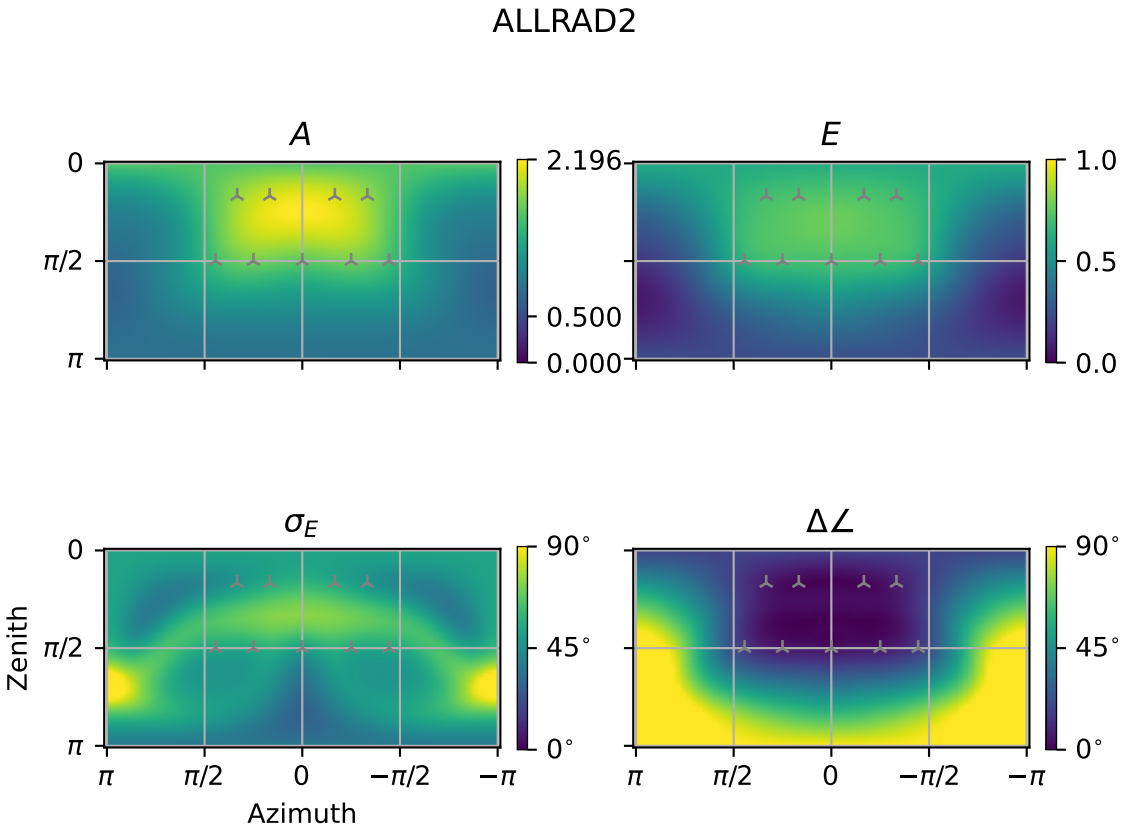
ls_sig (*(L, S) numpy.ndarray*) – Loudspeaker L output signal S.

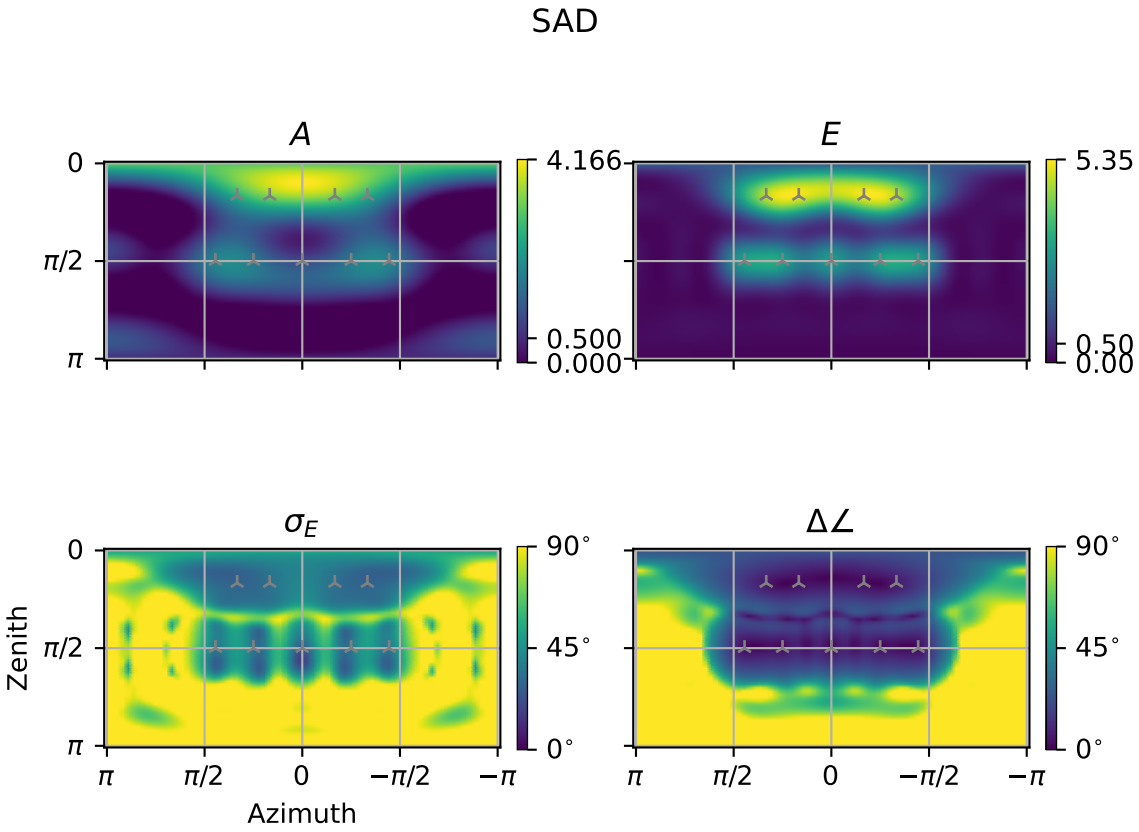
References

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.plot.decoder_performance(ls_setup, 'SAD')
```


ALLRAD2
A
E
σ_E
$\Delta\angle$
0
$\pi/2$
π
2.196
0.500
0.000
1.0
0.5
0.0
90°
45°
0°
Zenith
Azimuth
π
$\pi/2$
0
$-\pi/2$
$-\pi$


SAD
A
E
0
π/2
π
4.166
0.500
0.000
5.35
0.50
0.00
σ_E
$\Delta\angle$
Zenith
Azimuth
90°
45°
0°
π
π/2
0
−π/2
−π

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.plot.decoder_performance(ls_setup, 'MAD')
```

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.

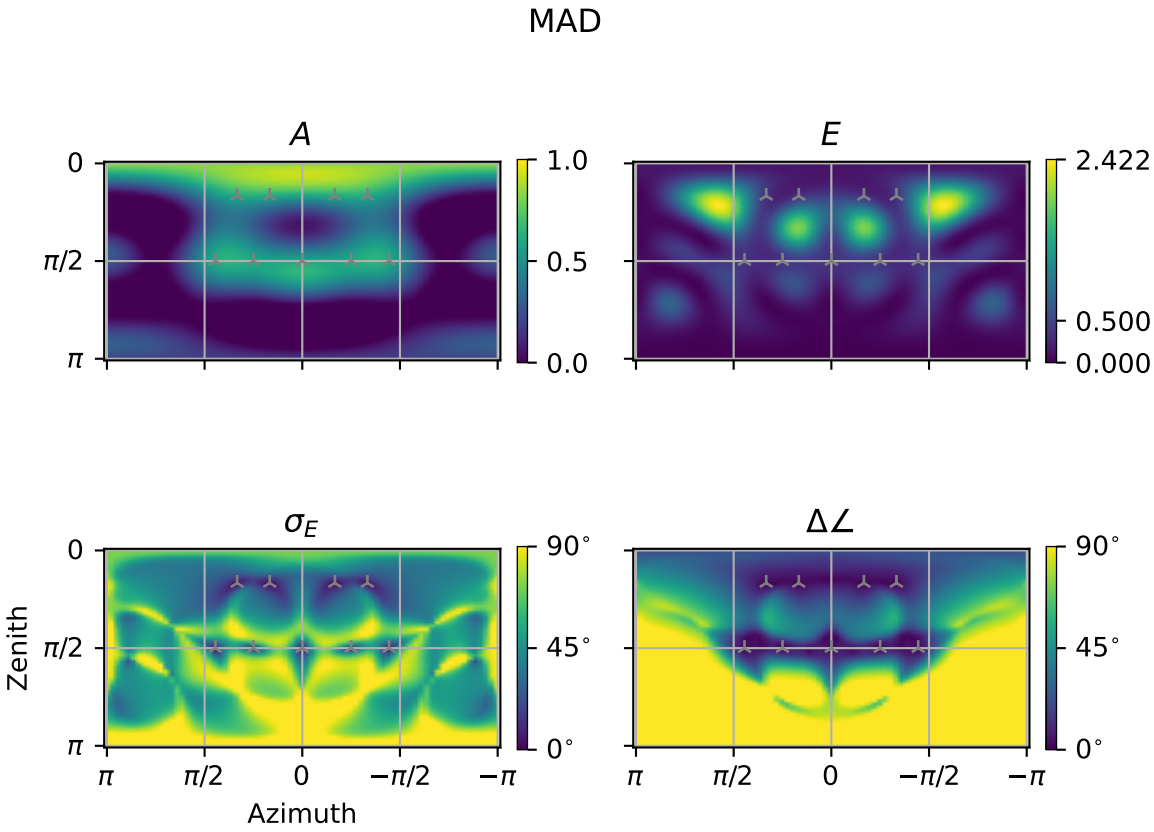
Returns

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.

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

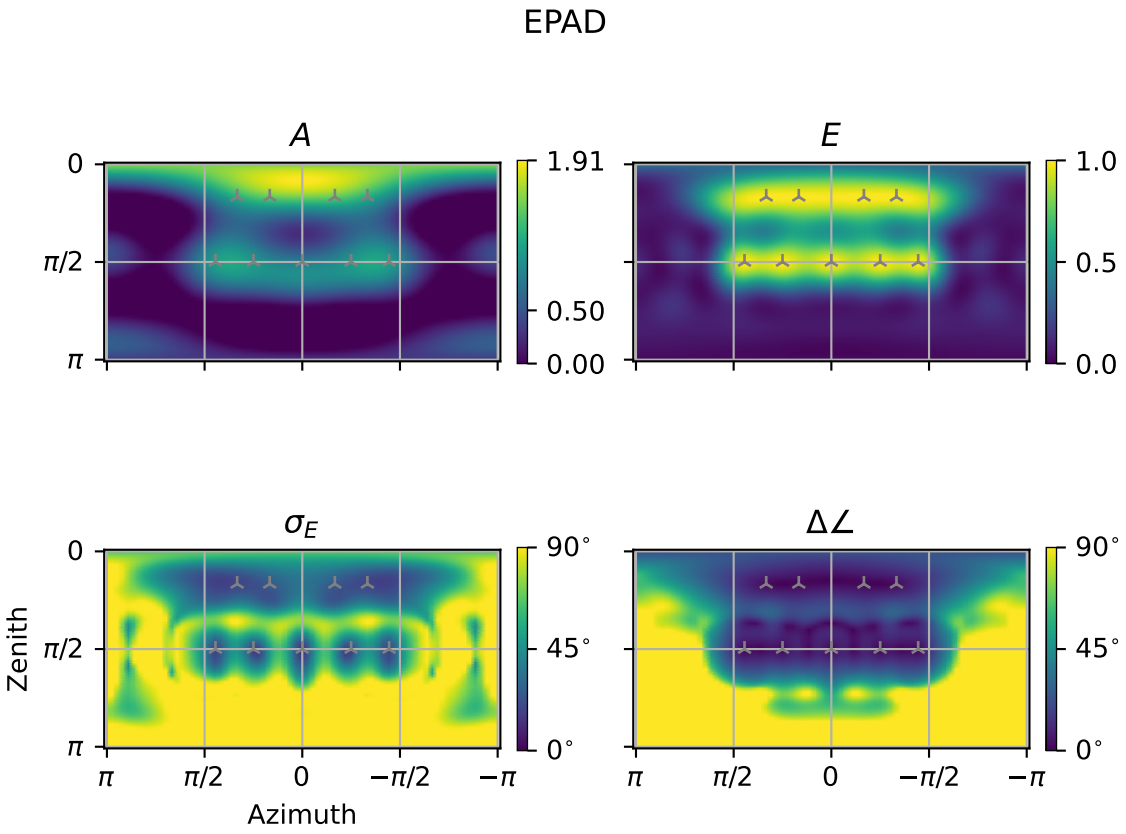

MAD
A
E
σ_E
$\Delta\angle$
0
π/2
π
1.0
0.5
0.0
2.422
0.500
0.000
90°
45°
0°
Zenith
Azimuth
π
π/2
0
−π/2
−π

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.plot.decoder_performance(ls_setup, 'EPAD')

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

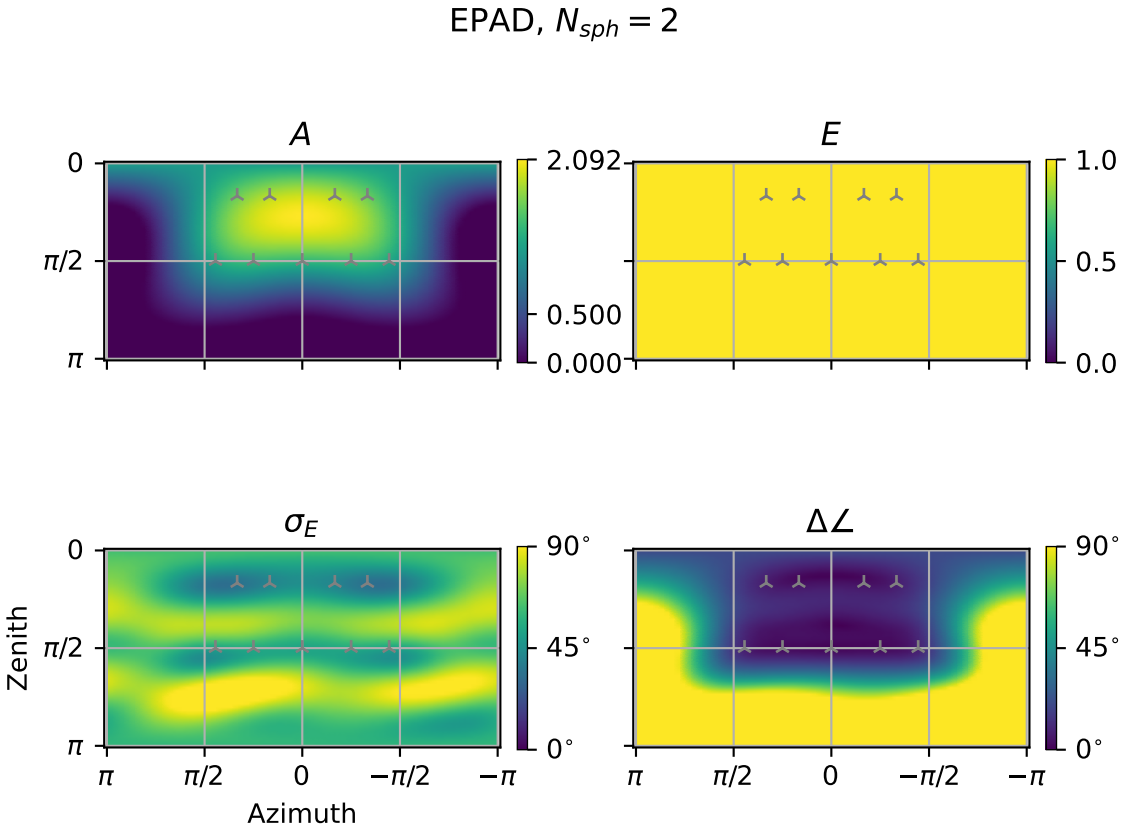


spaudiopy.decoder.**nearest_loudspeaker**(*src*, *hull*)

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*)

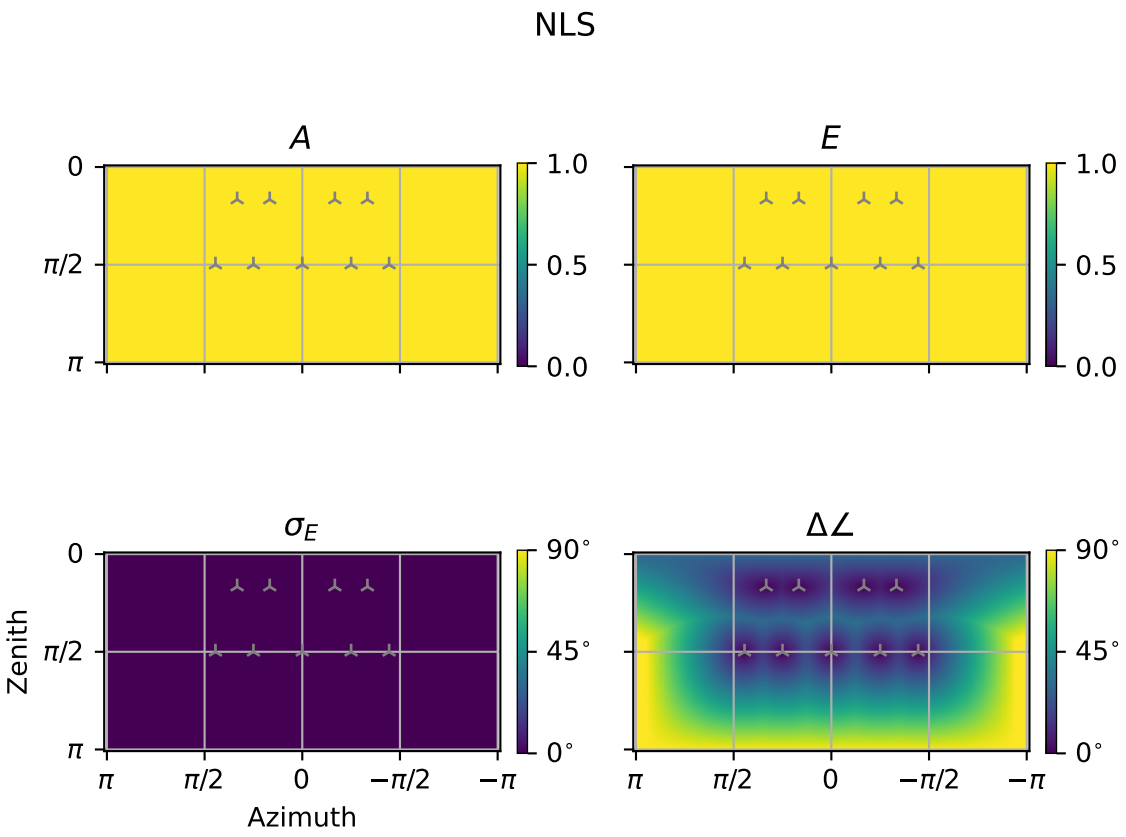

EPAD, $N_{sph} = 2$
A
E
σ_E
$\Delta\angle$
0
$\pi/2$
π
2.092
0.500
0.000
1.0
0.5
0.0
90°
45°
0°
Zenith
Azimuth
π
$\pi/2$
0
$-\pi/2$
$-\pi$

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.plot.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.

See also:

spaudiopy.decoder.magls_bin()
MagLS binaural decoder.

spaudiopy.decoder.**magls_bin**(*hrirs*, *N_sph*, *f_trans=None*, *hf_cont='avg'*, *hf_delay=(0, 0)*)

Magnitude Least-Squares (magLS) binaural decoder.

This binaural decoder renders the (least squares) binaural output below *f_trans*, while rendering a magnitude solution above.

Parameters

- **hrirs** (*sig.HRIRs*) – HRIR set.
- **N_sph** (*int*) – Ambisonic (SH) order.
- **f_trans** (*float, optional*) – Transition frequency between linear and magLS handling. The default is None, which sets it to 'N_sph * 500'.
- **hf_cont** (*['delay', 'avg', 'angle'], optional*) – High Frequency phase continuation method . The default is 'avg'.
- **hf_delay** (*(2,), optional*) – High frequency (additional) group delay in smpls. The default is (0, 0).

Raises
ValueError – When passing not supported option.

Returns
hrirs_mls_nm (*(2, (N_sph+1)**2, L)*) – Decoding IRs matrix, 2: left,right (stacked), real coeffs, L taps.

Notes

Details can be found in [1]. The iterative procedure in [2] suffers form HF dispersion (available by *hf_cont='delay'* and *hf_delay=(0, 0)*). This function offers multiple options to mitigate this issue. E.g. manually estimating and setting *hf_delay*, or estimating a phase difference on previous frequency bins which are then used to predict. This delta can be the spherical average *hf_cont='avg'*, which offers an algorithmic way to estimate the global group delay. It can also be estimated per direction when *hf_cont='angle'*, which is able to preserve group delay changes over the angle, however, might reintroduce more complexity again. For very low orders, there might be a slight tradeoff.

References

[1] Hold, C., Meyer-Kahlen, N., & Pulkki, V. (2023). Magnitude-Least-Squares Binaural Ambisonic Rendering with Phase Continuation. Fortschritte Der Akustik - DAGA. [2] Zotter, F., & Frank, M. (2019). Ambisonics. Springer Topics in Signal Processing.

See also:

spaudiopy.decoder.sh2bin()
Decode Ambisonic streams to binaural.

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.
frac_octave_smoothing(a, smoothing_n[, WEIGHTED])	Fractional octave (weighted) smoothing.
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).
hrirs_ctf(hrirs[, MIN_PHASE, freq_lims, ...])	Get common transfer function (CTF) EQ for HRIRs.
ilds_from_hrirs(hrirs[, f_cut, TODB])	Calculate ILDs from HRIRs by high/band-passed broadband RMS.
itds_from_hrirs(hrirs[, f_cut, upsample])	Calculate ITDs from HRIRs inter-aural cross-correlation (IACC).
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])	Compute 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.**hrirs_ctf**(*hrirs*, *MIN_PHASE=True*, *freq_lims=(125, 10000.0)*, *grid_weights=None*)

Get common transfer function (CTF) EQ for HRIRs.

Often used to equalize the direction independent coloration of a measurement. Can be used to replace headphone EQ.

Parameters

- **hrirs** (*sig.HRIRs*)
- **MIN_PHASE** (*bool, optional*) – Minimum phase EQ. The default is True.
- **freq_lims** (*tuple, optional*) – Frequency limits of inversion. The default is (125, 10e3).
- **grid_weights** (*array_like, optional*) – Grid weights of hrirs, *None* will calculate them. The default is None.

Returns

eq_taps (*np.ndarray*) – EQ filter taps, same length as HRIRs.

spaudiopy.process.**ilds_from_hrirs**(*hrirs*, *f_cut=(1000.0, 20000.0)*, *TODB=True*)

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

Parameters

- **hrirs** (*sig.HRIRs*)
- **f_cut** (*float (2,), optional*) – Band-pass cutoff frequencies. The default is (1000, 20000).
- **TODB** (*bool, optional*) – ILD in dB RMS ratio, otherwise as RMS difference. The default is TRUE.

Returns

ild (*array_like*) – ILD per grid point, positive value indicates left ear louder.

spaudiopy.process.**itds_from_hrirs**(*hrirs*, *f_cut=(100, 1500.0)*, *upsample=4*)

Calculate ITDs from HRIRs inter-aural cross-correlation (IACC).

The method calculates IACC on energy of upsampled, filtered HRIRs.

Parameters

- **hrirs** (*sig.HRIRs*)
- **f_cut** (*float (2,), optional*) – Band-pass cutoff frequencies. The default is (100, 1500).
- **upsample** (*int, optional*) – Upsampling factor. The default is 4.

Returns

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

References

Andreopoulou, A., & Katz, B. F. G. (2017). Identification of perceptually relevant methods of inter-aural time difference estimation. JASA.

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_smoothing**(*a*, *smoothing_n*, *WEIGHTED=True*)

Fractional octave (weighted) smoothing.

Parameters

- **a** (*(n,) array_like*) – Input spectrum.
- **smoothing_n** (*int*) – 1 / smoothing_n octave band.
- **WEIGHTED** (*bool, optional*) – Use (hamming) weighting on mean around center. The default is True.

Returns
smoothed_a (*(n,) np.array*)

spaudiopy.process.**frac_octave_filterbank**(*n*, *N_out*, *fs*, *f_low*, *f_high=None*, *mode='energy'*, *overlap=0.5*, *slope_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].
- **slope_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')
```

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

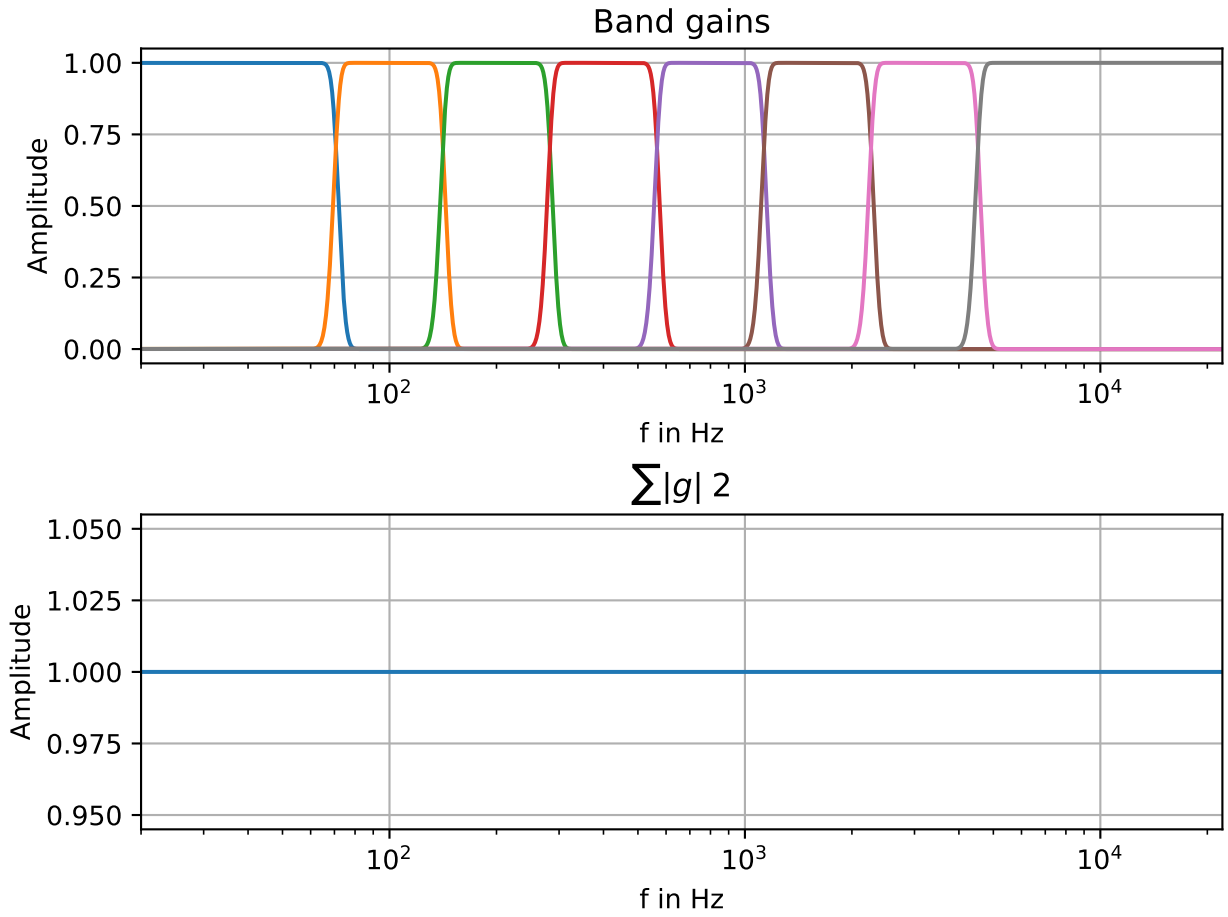
Compute the level/power in each subband of subband signals.

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

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

Parameters
p (*array_like*)

Returns
rd (*array_like*)


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.**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*)

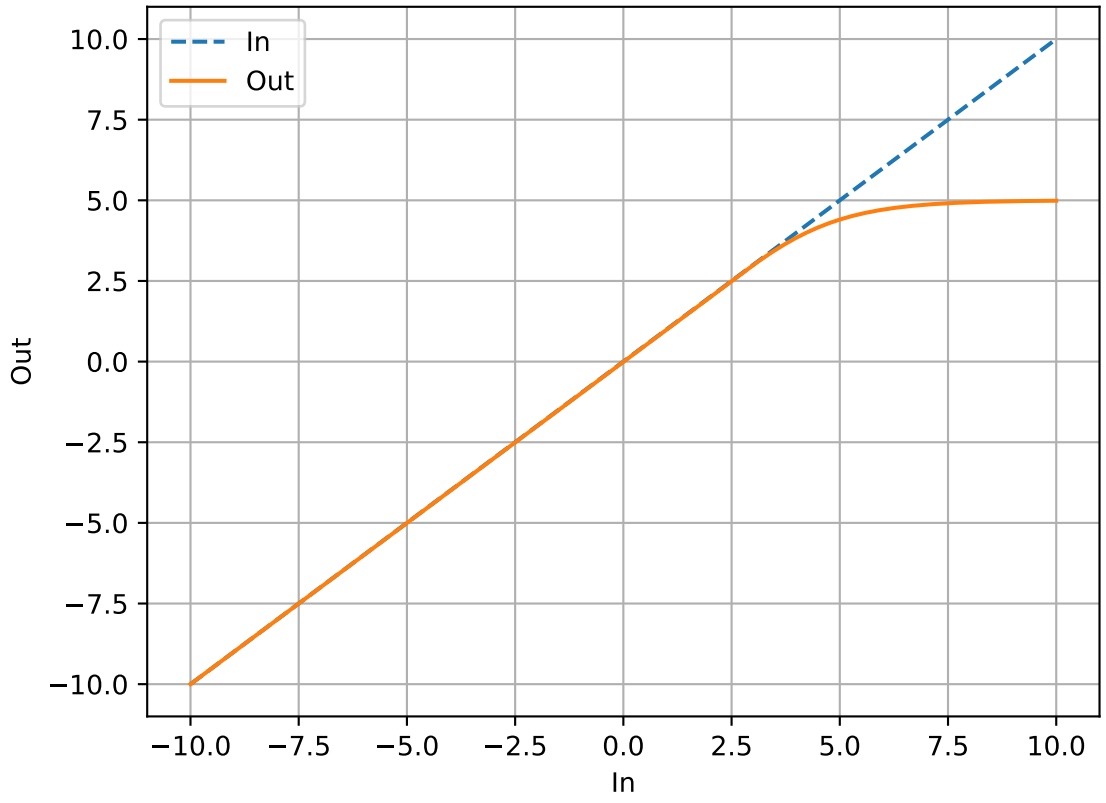
Pulsed noise train, pink or white.

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.
cart2dir(x, y, z)	Vectorized conversion of cartesian coordinates to (azi, zen).
cart2sph(x, y, z[, positive_azi, steady_zen])	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).
dir2cart(azi, zen)	Vectorized conversion of direction to cartesian coordinates.
from_db(db[, power])	Convert decibel back to ratio.
haversine(azi1, zen1, azi2, zen2[, 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.
rotation_euler([yaw, pitch, roll])	Matrix rotating by Yaw (around z), pitch (around y), roll (around x).
rotation_rodrigues(k, theta)	Matrix rotating around axis defined by unit vector k, by angle theta.
rotation_vecvec(f, t)	Matrix rotating from vector f to vector t, forces unit length.
sph2cart(azi, zen[, r])	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.
vec2dir(vec)	Convert (along last axis) vec: [x, y, z] to dir: [azi, zen].

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*, *positive_azi=False*, *steady_zen=False*)

Conversion of cartesian to spherical coordinates.

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

Conversion of spherical to cartesian coordinates.

spaudiopy.utils.**matlab_sph2cart**(*az*, *elev*, *r=1*)

Matlab port with ELEVATION.

spaudiopy.utils.**cart2dir**(*x*, *y*, *z*)

Vectorized conversion of cartesian coordinates to (azi, zen).

spaudiopy.utils.**dir2cart**(*azi*, *zen*)

Vectorized conversion of direction to cartesian coordinates.

spaudiopy.utils.**vec2dir**(*vec*)

Convert (along last axis) vec: [x, y, z] to dir: [azi, zen].

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

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

spaudiopy.utils.**rotation_euler**(*yaw=0*, *pitch=0*, *roll=0*)

Matrix rotating by Yaw (around z), pitch (around y), roll (around x). See https://mathworld.wolfram.com/RotationMatrix.html

spaudiopy.utils.**rotation_rodrigues**(*k*, *theta*)

Matrix rotating around axis defined by unit vector k, by angle theta. See https://mathworld.wolfram.com/RodriguesRotationFormula.html

spaudiopy.utils.**rotation_vecvec**(*f*, *t*)

Matrix rotating from vector f to vector t, forces unit length.

spaudiopy.utils.**haversine**(*azi1*, *zen1*, *azi2*, *zen2*, *r=1*)

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

Parameters

- **azi1** (*(n,) float, array_like*)
- **zen1** (*(n,) float, array_like*)
- **azi2** (*(n,) float, array_like*)
- **zen2** (*(n,) float, array_like*)
- **r** (*float, optional.*)

Returns

c (*(n,) array_like*) – Haversine distance between pairs of points.

References

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

calculate_grid_weights(azi, zen[, order])	Approximate quadrature weights by pseudo-inverse.
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.**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.grids.**load_t_design**(*degree*)

Return the unit coordinates of minimal T-designs.

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).

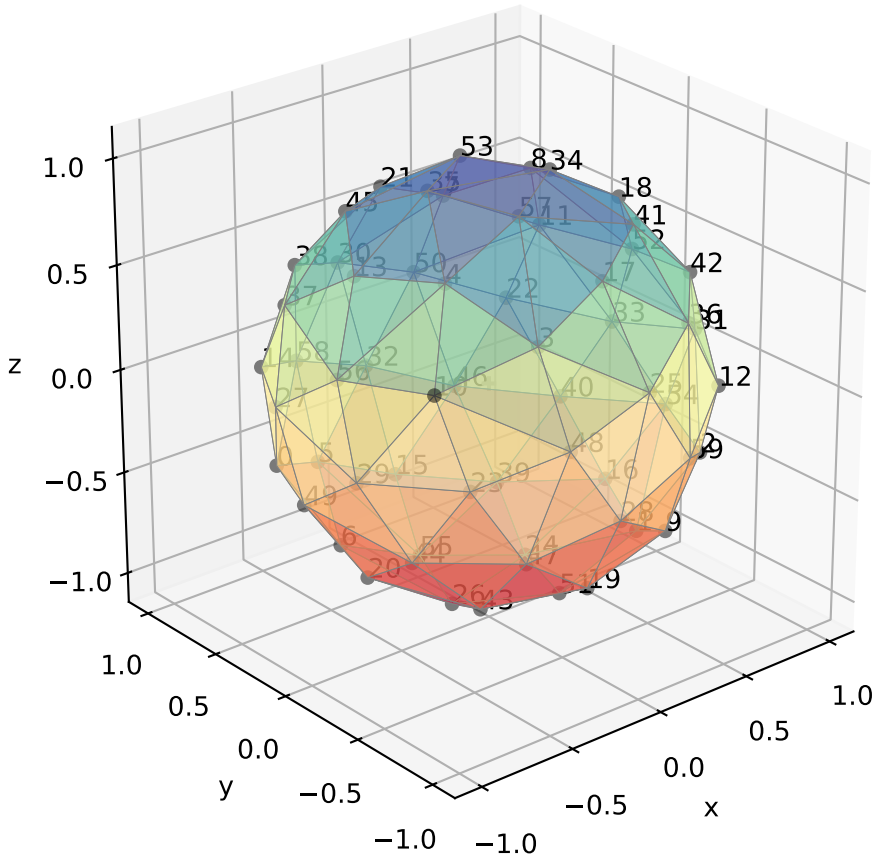
References

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.

Examples

```
vecs = spa.grids.load_t_design(degree=2*5)
spa.plot.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.

Parameters

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

Returns

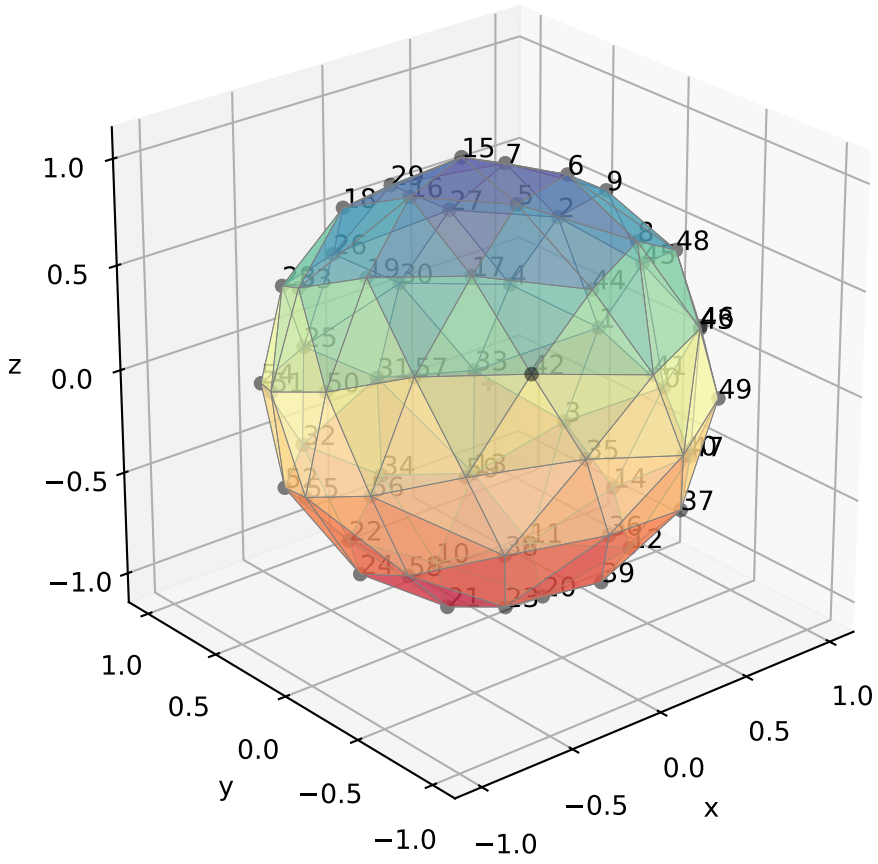
vecs (*(M, 3) numpy.ndarray*) – Coordinates of points.

References

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

Examples

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



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

Return the unit coordinates of Lebedev grid.

Parameters

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

Returns

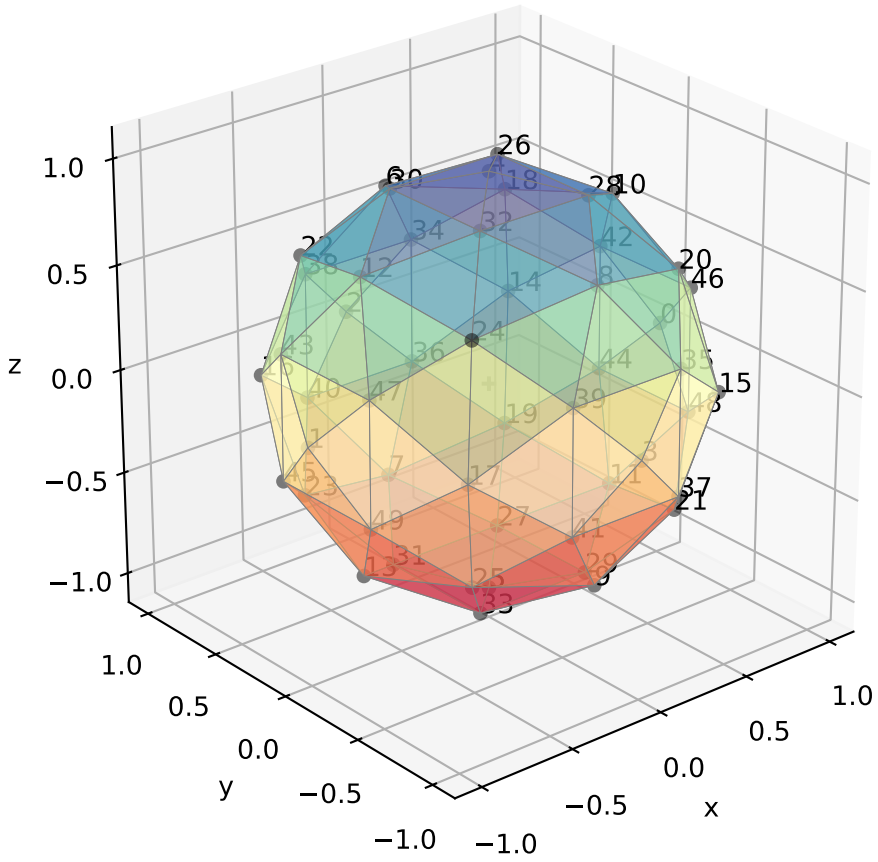
- **vecs** (*(M, 3) numpy.ndarray*) – Coordinates of points.
- **weights** (*array_like*) – Quadrature weights.

References

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

Examples

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



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

Return Fliege-Maier grid nodes with associated weights.

Parameters

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

Returns

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

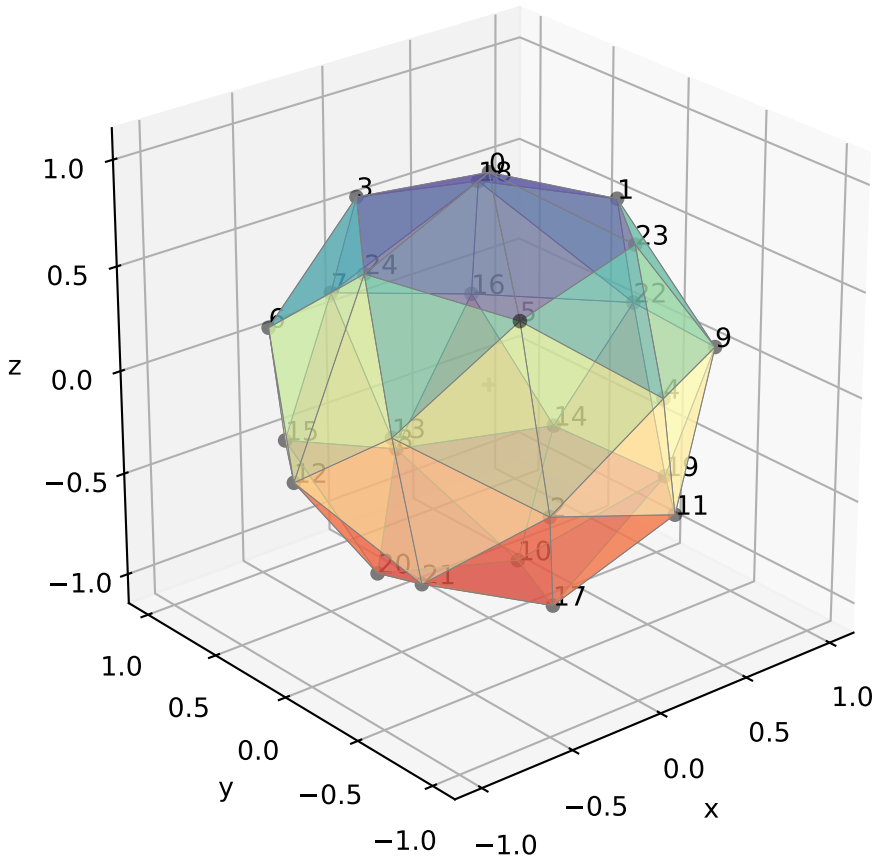
References

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.

Examples

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



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

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

Parameters

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

Returns

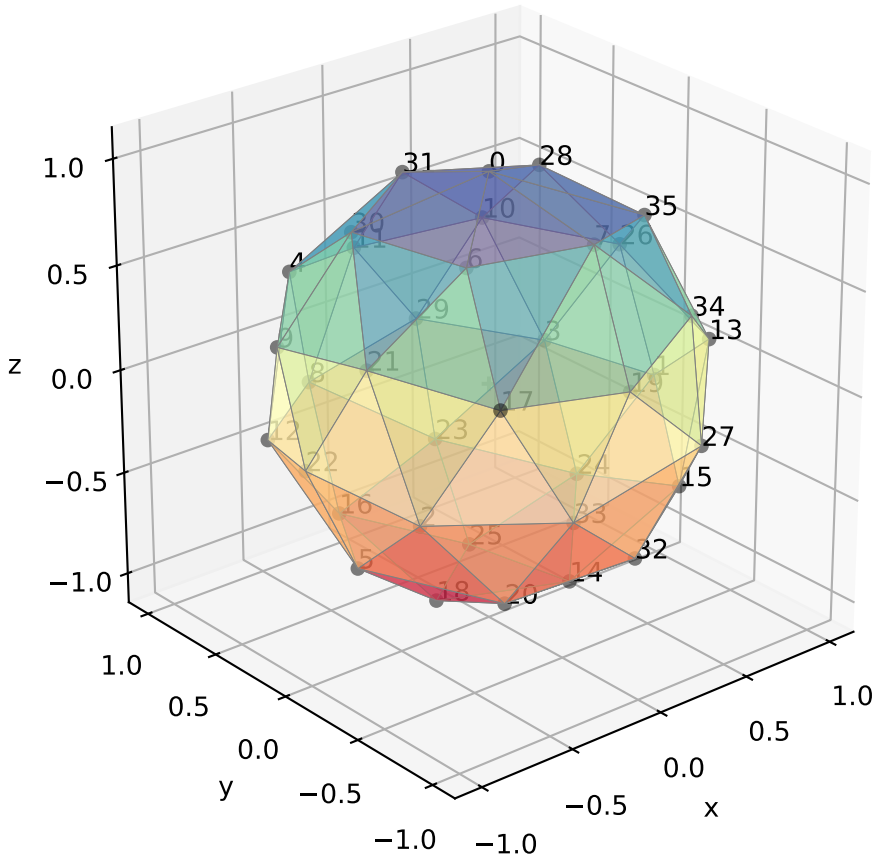
- **vecs** (*(M, 3) numpy.ndarray*) – Coordinates of points.
- **weights** (*array_like*) – Quadrature weights.

References

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

Examples

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



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

Equi-angular sampling points on a sphere.

Parameters

n (*int*) – Maximum order.

Returns

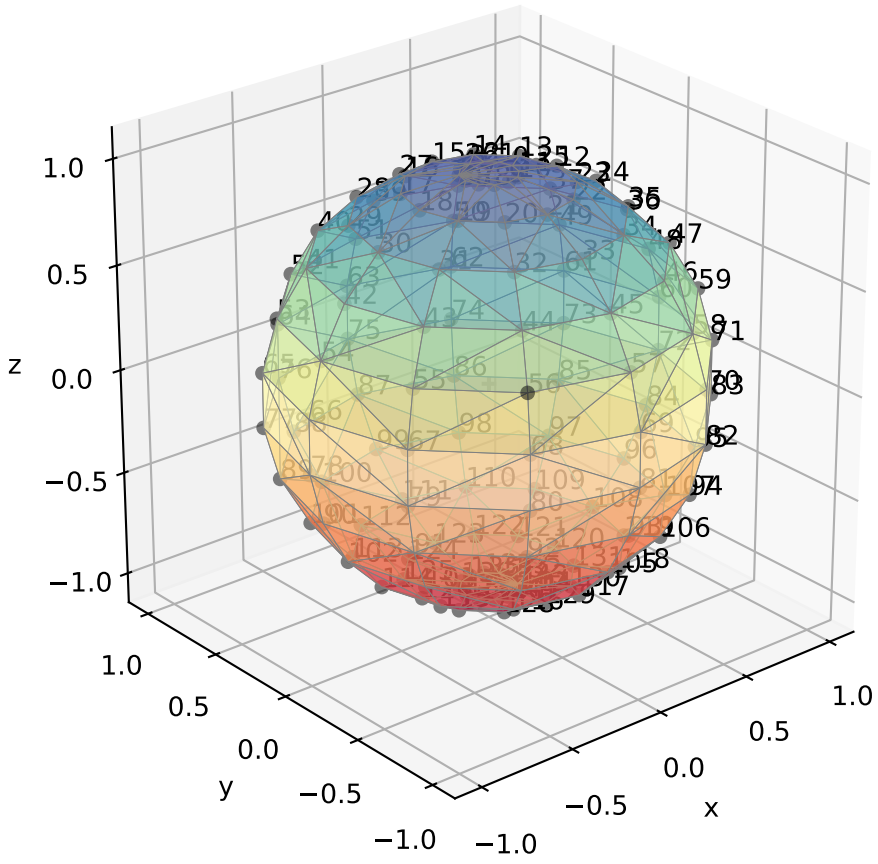
- **azi** (*array_like*) – Azimuth.
- **zen** (*array_like*) – Colatitude.
- **weights** (*array_like*) – Quadrature weights.

References

Rafaely, B. (2015). Fundamentals of Spherical Array Processing., sec.3.2

Examples

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



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

Gauss-Legendre sampling points on sphere.

Parameters

n (*int*) – Maximum order.

Returns

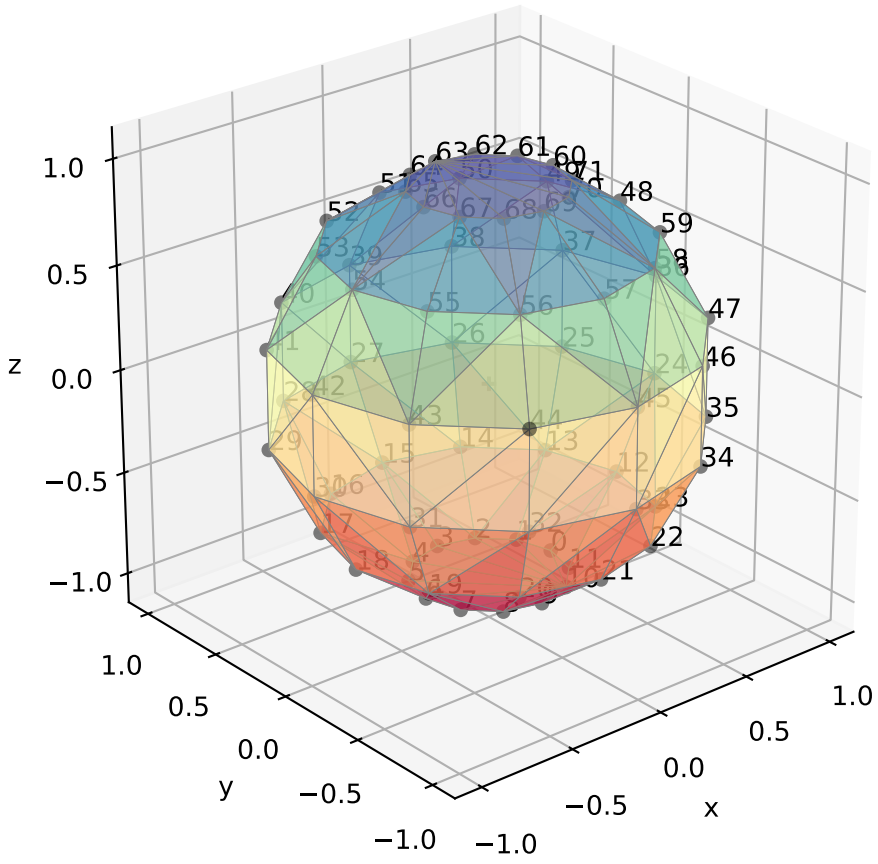
- **azi** (*array_like*) – Azimuth.
- **zen** (*array_like*) – Colatitude.
- **weights** (*array_like*) – Quadrature weights.

References

Rafaely, B. (2015). Fundamentals of Spherical Array Processing., sec.3.3

Examples

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



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

Equi-angular sampling points on a circle.

Parameters

n (*int*) – Maximum order

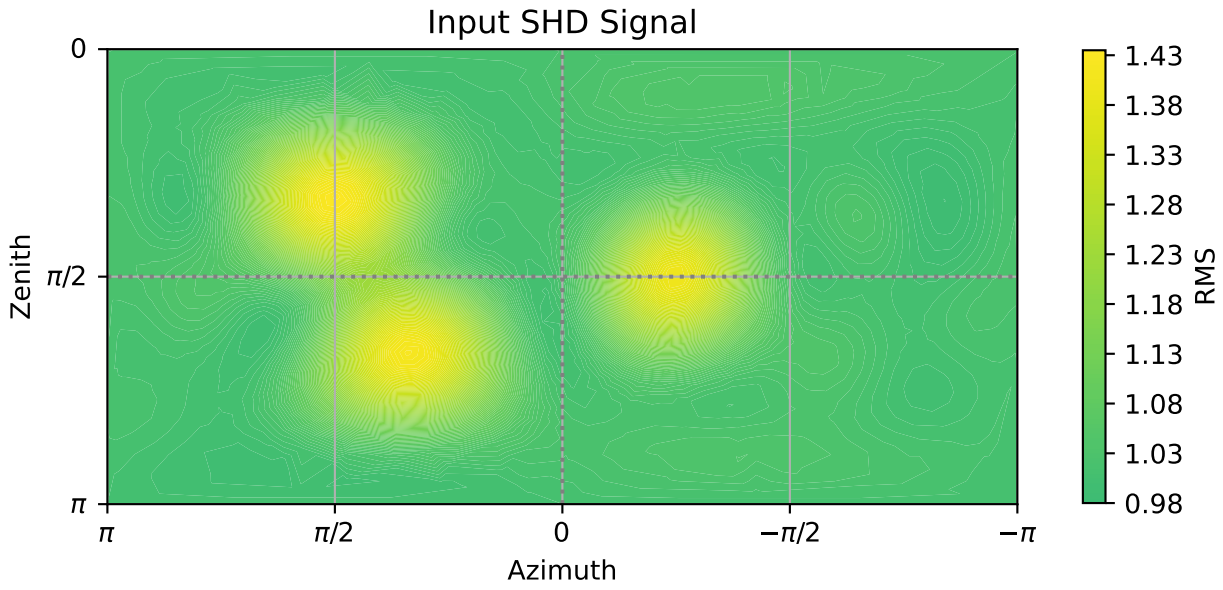
Returns

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

2.8 spaudiopy.parsa

Parametric Spatial Audio (PARSA).

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

import spaudiopy as spa

N_sph = 3
# Three sources
x_nm = spa.sph.src_to_sh(np.random.randn(3, 10000),
                         [np.pi/2, -np.pi/4, np.pi/3],
                         [np.pi/3, np.pi/2, 2/3 * np.pi], N_sph)
# Diffuse noise
x_nm += np.sqrt(16/(4*np.pi)) * np.random.randn(16, 10000)
spa.plot.sh_rms_map(x_nm, title="Input SHD Signal")
```



Memory cached functions

spaudiopy.parsa.**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_zen, I_r (*array_like*) – Pseudo intensity vector for each time sample.

spaudiopy.parsa.**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

estimate_num_sources(cov_x[, a, w])	Active source count estimate from signal covariance.
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.
sdm_post_equalization(ls_sigs, sdm_p, fs, ...)	Post equalization to compensate spectral whitening.
sdm_post_equalization2(ls_sigs, sdm_p, fs, ...)	Post equalization to compensate spectral whitening.
separate_cov(cov_x[, num_cut])	Separate Covariance matrix in signal and noise components.
sh_beamform(w_nm, sig_nm)	Apply spherical harmonics domain (SHD) beamformer.
sh_beamformer_from_pattern(pattern, N_sph, ...)	Get spherical harmonics domain (SHD) beamformer coefficients.
sh_lcmv(cov_x, dirs_azi_c, dirs_zen_c, c_gain)	Spherical Harmonics domain LCMV beamformer.
sh_music(cov_x, num_src, dirs_azi, dirs_zen)	SH domain / Eigenbeam Multiple Signal Classification (EB-MUSIC).
sh_mvdr(cov_x, dirs_azi, dirs_zen)	Spherical Harmonics domain MVDR beamformer.
sh_sector_beamformer(A_nm)	Get sector pressure and intensity beamformers.

spaudiopy.parsa.**sh_beamformer_from_pattern**(*pattern*, *N_sph*, *azi_steer*, *zen_steer*)

Get spherical harmonics domain (SHD) beamformer coefficients.

Parameters

- **pattern** (*string , or (N+1,) array_like*) – Pattern description, e.g. *'cardioid'* or modal weights.
- **N_sph** (*int*) – SH order.
- **azi_steer** (*(J,) array_like*) – Azimuth steering directions.
- **zen_steer** (*(J,) array_like*) – Zenith/colatitude steering directions.

Returns

w_nm (*(J, (N+1)**2) numpy.ndarray*) – SHD Beamformer weights.

Examples

See *spaudiopy.parsa.sh_beamform()*.

spaudiopy.parsa.**sh_beamform**(*w_nm*, *sig_nm*)

Apply spherical harmonics domain (SHD) beamformer.

Parameters

- **w_nm** (*((N+1)**2,) array_like, or (J, (N+1)**2) np.ndarray*) – SHD beamformer weights (for *J* beamformers)
- **sig_nm** (*((N+1)**2, l) np.ndarray*) – SHD signal of length l.

Returns

y (*(J, l) np.ndarray*) – Beamformer output signals.

Examples

```
vecs, _ = spa.grids.load_maxDet(50)
dirs = spa.utils.vec2dir(vecs)
w_nm = spa.parsa.sh_beamformer_from_pattern('cardioid', N_sph,
                                            dirs[:,0], dirs[:,1])
y = spa.parsa.sh_beamform(w_nm, x_nm)
spa.plot.spherical_function_map(spa.utils.rms(y), dirs[:,0], dirs[:,1],
                                TODB=True, title="Output RMS")
```

spaudiopy.parsa.**estimate_num_sources**(*cov_x*, *a=None*, *w=None*)

Active source count estimate from signal covariance.

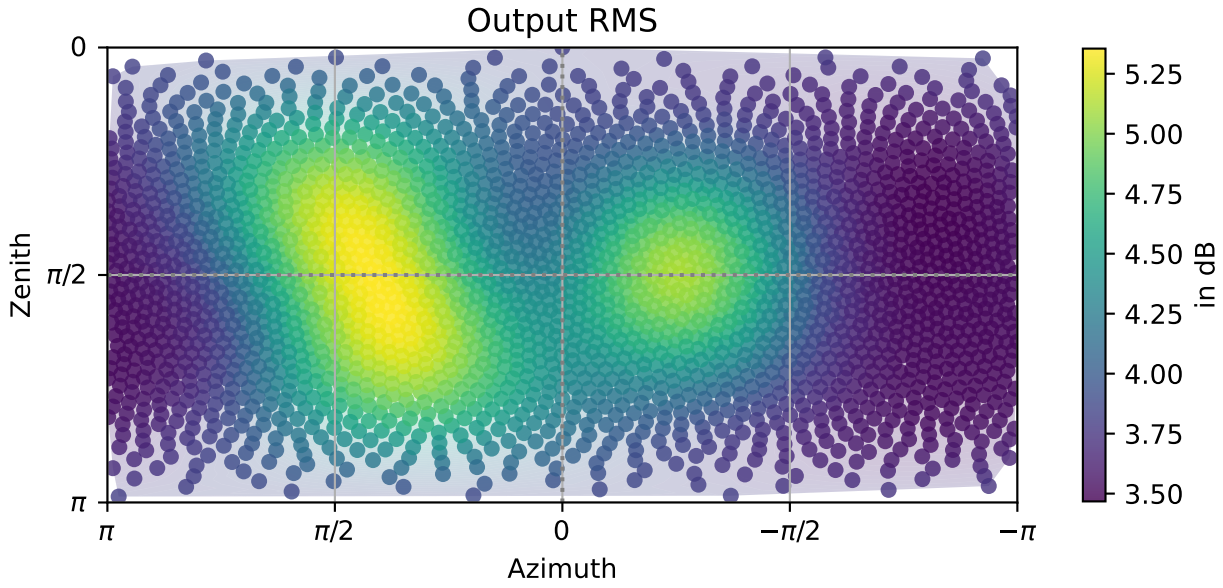
Based on the relation of consecutive eigenvalues.

Parameters

- **cov_x** (*(L, L) numpy.2darray*) – Signal covariance.
- **a** (*float, optional*) – Threshold condition (ratio), defaults to *1 + 2/len(cov_x)*
- **w** (*(L,) array_like, optional*) – Eigenvalues in ascending order, not using *cov_x* if available.

Returns

num_src_est (*int*) – Number of active sources estimate.


Output RMS
0
π/2
π
Zenith
π
π/2
0
−π/2
−π
Azimuth
5.25
5.00
4.75
4.50
4.25
4.00
3.75
3.50
in dB

Examples

See *spaudiopy.parsa.sh_music()*.

spaudiopy.parsa.**separate_cov**(*cov_x*, *num_cut=None*)

Separate Covariance matrix in signal and noise components.

Parameters

- **S_xx** (*(L, L) numpy.2darray*) – Covariance.
- **num_cut** (*int, optional*) – Split point of Eigenvalues, default: *parsa.estimate_num_sources()*.

Returns

- **S_pp** (*(L, L) numpy.2darray*) – Signal covariance.
- **S_nn** (*(L, L) numpy.2darray*) – Noise (residual) covariance.

Notes

Signal model is $S_x = S_p + S_n$.

Examples

```
S_xx = x_nm @ x_nm.T
S_pp, S_nn = spa.parsa.separate_cov(S_xx, num_cut=3)
fig, axs = plt.subplots(1, 3, constrained_layout=True)
axs[0].matshow(S_xx)
axs[0].set_title("X")
axs[1].matshow(S_pp)
axs[1].set_title("S")
axs[2].matshow(S_nn)
axs[2].set_title("N")
```

spaudiopy.parsa.**sh_music**(*cov_x*, *num_src*, *dirs_azi*, *dirs_zen*)

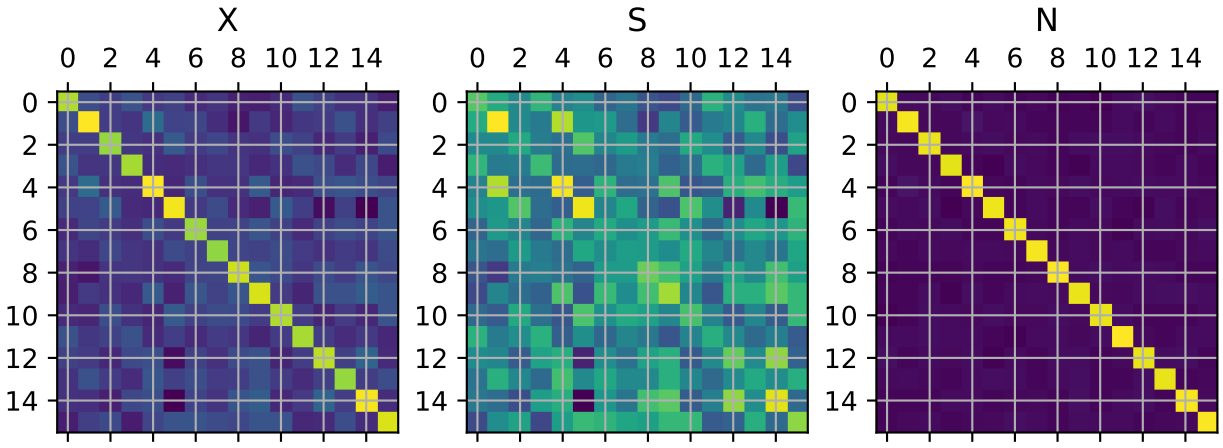
SH domain / Eigenbeam Multiple Signal Classification (EB-MUSIC).

Parameters

- **cov_x** (*(L, L) numpy.2darray*) – SH signal covariance.
- **num_src** (*int*) – Number of sources.
- **dirs_azi** (*(g,) array_like*)
- **dirs_zen** (*(g,) array_like*)

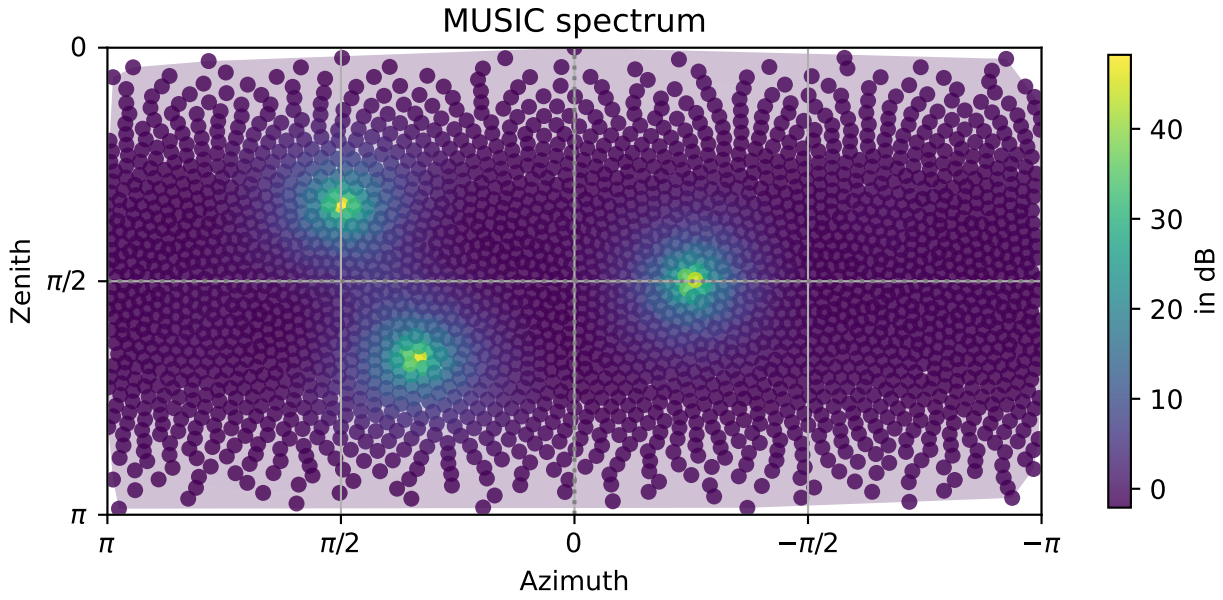
Returns

P_music (*(g,) array_like*) – MUSIC (psuedo-) spectrum.


X
S
N
0 2 4 6 8 10 12 14
0
2
4
6
8
10
12
14

Examples

```
S_xx = x_nm @ x_nm.T
num_src_est = spa.parsa.estimate_num_sources(S_xx)

vecs, _ = spa.grids.load_maxDet(50)
dirs = spa.utils.vec2dir(vecs)
P_music = spa.parsa.sh_music(S_xx, num_src_est, dirs[:,0], dirs[:,1])
spa.plot.spherical_function_map(P_music, dirs[:,0], dirs[:,1],
                                TODB=True, title="MUSIC spectrum")
```



spaudiopy.parsa.**sh_mvdr**(*cov_x*, *dirs_azi*, *dirs_zen*)

Spherical Harmonics domain MVDR beamformer. SH / Eigenbeam domain minimum variance distortionless response (EB-MVDR). Often employed on signal *cov_x* = *S_xx*, instead of noise *cov_x* = *S_nn*, then called minimum power distortionless response (MPDR) beamformer.

Parameters

- **cov_x** (*(L, L) numpy.2darray*) – SH signal (noise) covariance.
- **dirs_azi** (*(g,) array_like*)
- **dirs_zen** (*(g,) array_like*)

Returns

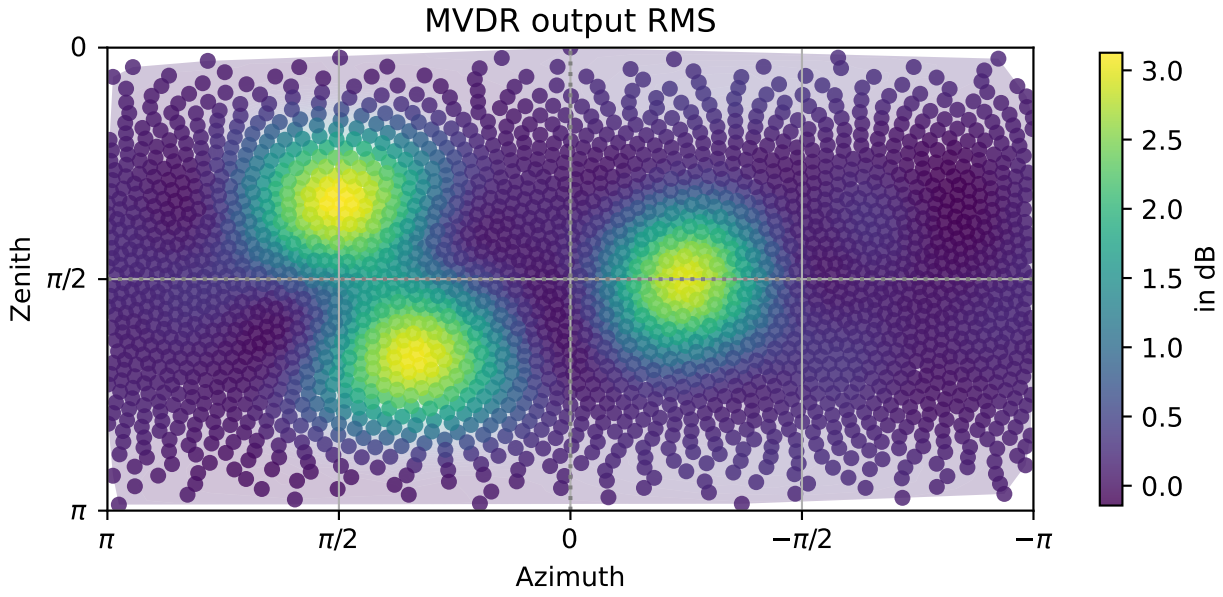
W_nm (*(g, L) numpy.2darray*) – MVDR beampattern weights.

References

Rafaely, B. (2015). Fundamentals of Spherical Array Processing. Springer. ch. 7.2.

Examples

```
S_xx = x_nm @ x_nm.T
num_src_est = spa.parsa.estimate_num_sources(S_xx)
_, S_nn = spa.parsa.separate_cov(S_xx, num_cut=num_src_est)

vecs, _ = spa.grids.load_maxDet(50)
dirs = spa.utils.vec2dir(vecs)
W_nm = spa.parsa.sh_mvdr(S_nn, dirs[:,0], dirs[:,1])
y = spa.parsa.sh_beamform(W_nm, x_nm)
spa.plot.spherical_function_map(spa.utils.rms(y), dirs[:,0], dirs[:,1],
                                TODB=True, title="MVDR output RMS")
```



spaudiopy.parsa.**sh_lcmv**(*cov_x*, *dirs_azi_c*, *dirs_zen_c*, *c_gain*)

Spherical Harmonics domain LCMV beamformer. SH / Eigenbeam domain Linearly Constrained Minimum Variance (LCMV) beamformer. Often employed on signal *cov_x* = *S_xx*, instead of noise *cov_x* = *S_nn*, then called linearly constrained minimum power (LCMP) beamformer.

Parameters

- **cov_x** (*(L, L) numpy.2darray*) – SH signal (noise) covariance.
- **dirs_azi** (*(g,) array_like*)
- **dirs_zen** (*(g,) array_like*)
- **c_gain** (*(g,) array_like*) – Constraints (gain) on points *[dirs_azi, dirs_zen]*.

Returns

w_nm (*(L,) array_like*) – LCMV beampattern weights.

References

Rafaely, B. (2015). Fundamentals of Spherical Array Processing. Springer. ch. 7.5.

Examples

```
S_xx = x_nm @ x_nm.T
num_src_est = spa.parsa.estimate_num_sources(S_xx)
_, S_nn = spa.parsa.separate_cov(S_xx, num_cut=num_src_est)

dirs_azi_c = [np.pi/2, 0., np.pi]
dirs_zen_c = [np.pi/2, np.pi/2, np.pi/4]
c = [1, 0.5, 0]
w_nm = spa.parsa.sh_lcmv(S_nn, dirs_azi_c, dirs_zen_c, c)
spa.plot.sh_coeffs(w_nm)
```

spaudiopy.parsa.**sh_sector_beamformer**(*A_nm*)

Get sector pressure and intensity beamformers.

Parameters

A_nm (*(J, (N+1)**2), np.ndarray*) – SH beamformer matrix, see spa.sph.design_sph_filterbank().

Returns

A_wxyz (*((4*J), (N+2)**2)*) – SH sector pattern beamformers.

spaudiopy.parsa.**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.parsa.**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 'parsa.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.parsa.**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.parsa.**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.plot

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, zen[, p, size, c, alpha, fs, ...])	Direction of Arrival, with optional p(t) scaling the size.
freq_resp(freq, amp[, TODB, smoothing_n, ...])	Plot magnitude of frequency response over time frequency f.
hrirs_ild_itd(hrirs[, plevels, pclims, ...])	Plot ILDs and ITDs of HRIRs.
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[, TODB, 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_bar(x_nm[, TODB, centered, num_groups, ...])	Barplot over SH channels.
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 plot.
sh_coeffs_subplot(F_nm_list[, titles, fig])	Plot spherical harmonics coefficients list as function on the sphere.
sh_rms_map(F_nm[, TODB, 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, zen[, title, fig])	Plot function 1D vector f over azi and zen.
spherical_function_map(f, azi, zen[, TODB, ...])	Plot function 1D vector f over azi and zen, can also convert to dB.
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.plot.**spectrum**(*x*, *fs*, *ylim=None*, *scale_mag=False*, ***kwargs*)

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

Parameters

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

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

Plot magnitude of frequency response over time frequency f.

Parameters

- **f** (*frequency array*)
- **amp** (*array_like, list of array_like*)
- **TODB** (*bool*) – Plot in dB.
- **smoothing_n** (*int*) – Forwarded to process.frac_octave_smoothing()

Examples

See *spaudiopy.sph.binaural_coloration_compensation()*

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

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

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

Plot Zero Pole diagram from a and b coefficients.

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

Compare A and B format signals.

spaudiopy.plot.**spherical_function**(*f*, *azi*, *zen*, *title=None*, *fig=None*)

Plot function 1D vector f over azi and zen.

spaudiopy.plot.**sh_coeffs**(*F_nm*, *sh_type=None*, *azi_steps=5*, *el_steps=3*, *title=None*, *fig=None*, *ax=None*, *cbar=True*)

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

Examples

See *spaudiopy.sph*

spaudiopy.plot.**sh_coeffs_subplot**(*F_nm_list*, *titles=None*, *fig=None*, ***kwargs*)

Plot spherical harmonics coefficients list as function on the sphere. *kwargs* are forwarded to `spaudiopy.plt.sh_coeffs`.

Examples

See *spaudiopy.sph*

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

Overlay spherical harmonics coefficients plot.

Examples

See *spaudiopy.plot.sh_coeffs*

spaudiopy.plot.**sh_rms_map**(*F_nm*, *TODB=False*, *w_n=None*, *sh_type=None*, *n_plot=50*, *title=None*, *clim=[0, 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.
- **TODB** (*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.*)
- **n_plot** (*int*) – Plotting precision (grid degree).

Examples

See *spaudiopy.sph.src_to_sh*

spaudiopy.plot.**spherical_function_map**(*f*, *azi*, *zen*, *TODB=False*, *title=None*, *clim=(None, None)*, *fig=None*)

Plot function 1D vector f over azi and zen, can also convert to dB.

Examples

See *spaudiopy.parsa.sh_beamform*

spaudiopy.plot.**sh_bar**(*x_nm*, *TODB=True*, *centered=False*, *num_groups=1*, *s=250*, *vf=4*, *clim=None*, *xticklabels=None*, *title=None*, *fig=None*)

Barplot over SH channels.

Parameters

- **x_nm** (*array_like*) – C x L.
- **TODB** (*TYPE, optional*) – DESCRIPTION. The default is True.
- **centered** (*TYPE, optional*) – DESCRIPTION. The default is False.
- **num_groups** (*TYPE, optional*) – Plot gourps. The default is 1.
- **s** (*TYPE, optional*) – Scatter plot size. The default is 250.
- **vf** (*TYPE, optional*) – Vertical ratio. The default is 4.
- **clim** (*TYPE, optional*) – DESCRIPTION. The default is None.
- **xticklabels** (*TYPE, optional*) – DESCRIPTION. The default is None.
- **title** (*TYPE, optional*) – DESCRIPTION. The default is None.
- **fig** (*TYPE, optional*) – DESCRIPTION. The default is None.

Returns

None.

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

Plot loudspeaker setup with vertex and face normals.

spaudiopy.plot.**polar**(*theta*, *r*, *TODB=True*, *rlim=None*, *title=None*, *ax=None*)

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

Examples

See *spaudiopy.sph.bandlimited_dirac()*

spaudiopy.plot.**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

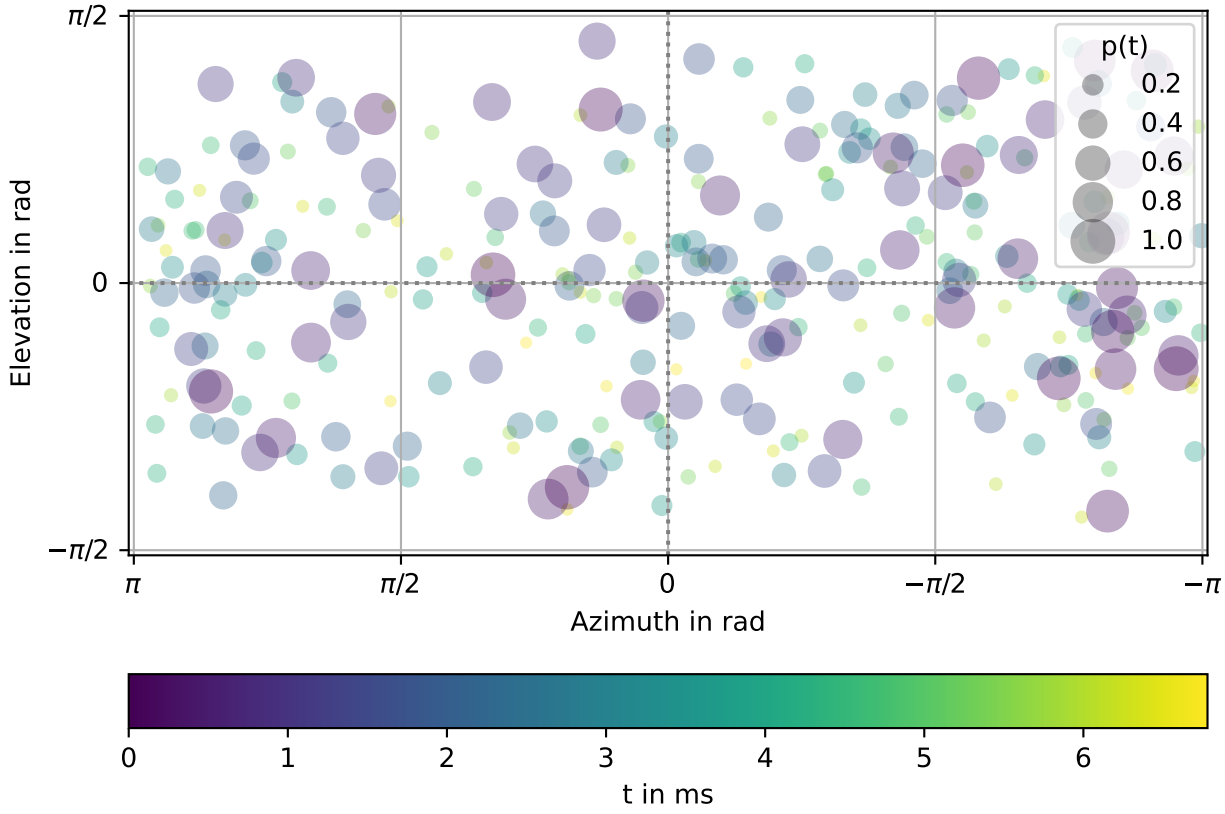
See *spaudiopy.decoder*

spaudiopy.plot.**doa**(*azi*, *zen*, *p=None*, *size=250*, *c=None*, *alpha=None*, *fs=None*, *title=None*, *ltitle=None*)

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

Examples

```
n = 300
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, zen, r = spa.utils.cart2sph(x, y, z)

ps = 1 / np.exp(np.linspace(0, 3, n))
spa.plot.doa(azi, zen, ps, fs=fs, ltitle="p(t)")
```



spaudiopy.plot.**hrirs_ild_itd**(*hrirs*, *plevels=50*, *pclims=(None, None)*, *title=None*, *fig=None*)

Plot ILDs and ITDs of HRIRs.

Parameters

- **hrirs** (*sig.HRIRs*)
- **plevels** (*int, optional*) – Contour levels. The default is 50.
- **pclims** (*(2,), optional*) – Set the plot color limits for ild and itd, e.g. (20, 0.75)
- **title** (*string, optional.*)

- **fig** (*plt.figure, optional*)

Returns
None.

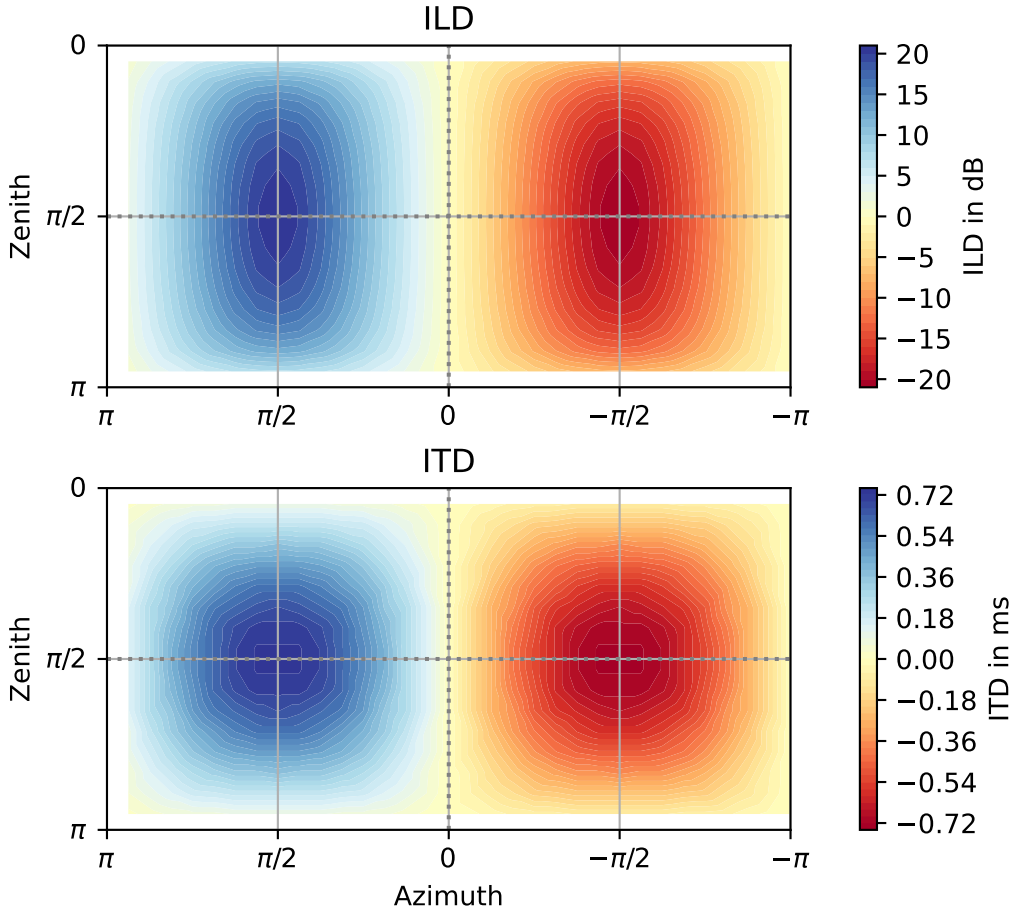
See also:

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

spaudiopy.process.itds_from_hrirs
Calculating ITDs with defaults.

Examples

```
dummy_hrirs = spa.io.load_hrirs(48000, 'dummy')
spa.plot.hrirs_ild_itd(dummy_hrirs)
```



spaudiopy.plot.**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, 62
spaudiopy.io, 3
spaudiopy.parsa, 71
spaudiopy.plot, 81
spaudiopy.process, 52
spaudiopy.sig, 7
spaudiopy.sph, 11
spaudiopy.utils, 60

INDEX

Symbols

__init__() (*spaudiopy.decoder.LoudspeakerSetup method*), 30
__init__() (*spaudiopy.sig.AmbiBSignal method*), 9
__init__() (*spaudiopy.sig.HRIRs method*), 10
__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*), 55
AmbiBSignal (*class in spaudiopy.sig*), 9
ambisonics_setup() (*spaudiopy.decoder.LoudspeakerSetup method*), 31
angle_between() (*in module spaudiopy.utils*), 61
apply() (*spaudiopy.sig.AmbiBSignal method*), 9
apply() (*spaudiopy.sig.MonoSignal method*), 8
apply() (*spaudiopy.sig.MultiSignal method*), 8
apply_blacklist() (*in module spaudiopy.decoder*), 33
apply_ctf_eq() (*spaudiopy.sig.HRIRs method*), 10
area_triangle() (*in module spaudiopy.utils*), 62
asarray_1d() (*in module spaudiopy.utils*), 60

B

b_to_sh() (*in module spaudiopy.sph*), 16
b_to_stereo() (*in module spaudiopy.process*), 55
bandlimited_dirac() (*in module spaudiopy.sph*), 17
binaural_coloration_compensation() (*in module spaudiopy.sph*), 23
binauralize() (*spaudiopy.decoder.LoudspeakerSetup method*), 31
butterworth_modal_weights() (*in module spaudiopy.sph*), 26

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.grids*), 63
calculate_vertex_normals() (*in module spaudiopy.decoder*), 33
cardioid_modal_weights() (*in module spaudiopy.sph*), 25
cart2dir() (*in module spaudiopy.utils*), 61
cart2sph() (*in module spaudiopy.utils*), 61
characteristic_ambisonic_order() (*in module spaudiopy.decoder*), 37
check_aperture() (*in module spaudiopy.decoder*), 33
check_cond_sht() (*in module spaudiopy.sph*), 15
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.plot*), 82
conv() (*spaudiopy.sig.AmbiBSignal method*), 9
conv() (*spaudiopy.sig.MonoSignal method*), 8
conv() (*spaudiopy.sig.MultiSignal method*), 8
copy() (*spaudiopy.sig.AmbiBSignal method*), 9
copy() (*spaudiopy.sig.HRIRs method*), 10
copy() (*spaudiopy.sig.MonoSignal method*), 8
copy() (*spaudiopy.sig.MultiSignal method*), 9

D

db() (*in module spaudiopy.utils*), 62
decoder_performance() (*in module spaudiopy.plot*), 84
deg2rad() (*in module spaudiopy.utils*), 60
design_sph_filterbank() (*in module spaudiopy.sph*), 27
dir2cart() (*in module spaudiopy.utils*), 61
doa() (*in module spaudiopy.plot*), 84

E

energy_decay() (*in module spaudiopy.process*), 56
epad() (*in module spaudiopy.decoder*), 46
equal_angle() (*in module spaudiopy.grids*), 68
equal_polar_angle() (*in module spaudiopy.grids*), 70
estimate_num_sources() (*in module spaudiopy.parsa*), 73

F

find_imaginary_loudspeaker() (*in module spaudiopy.decoder*), 34
frac_octave_filterbank() (*in module spaudiopy.process*), 55
frac_octave_smoothing() (*in module spaudiopy.process*), 55
freq_resp() (*in module spaudiopy.plot*), 81
from_db() (*in module spaudiopy.utils*), 62
from_file() (*spaudiopy.sig.AmbiBSignal class method*), 9
from_file() (*spaudiopy.sig.MonoSignal class method*), 8
from_file() (*spaudiopy.sig.MultiSignal class method*), 8
from_sph() (*spaudiopy.decoder.LoudspeakerSetup class method*), 31

G

gain_clipping() (*in module spaudiopy.process*), 58
gauss() (*in module spaudiopy.grids*), 69
get_characteristic_order() (*spaudiopy.decoder.LoudspeakerSetup method*), 31
get_default_hrirs() (*in module spaudiopy.io*), 5
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*), 61
HRIRs (*class in spaudiopy.sig*), 9
hrirs_ctf() (*in module spaudiopy.process*), 54
hrirs_ild_itd() (*in module spaudiopy.plot*), 85
hull() (*in module spaudiopy.plot*), 83
hull_normals() (*in module spaudiopy.plot*), 84
hypercardioid_modal_weights() (*in module spaudiopy.sph*), 25

I

ilds_from_hrirs() (*in module spaudiopy.process*), 54
interleave_channels() (*in module spaudiopy.utils*), 62
inverse_sht() (*in module spaudiopy.sph*), 14
is_simplex_valid() (*spaudiopy.decoder.LoudspeakerSetup method*), 31
itds_from_hrirs() (*in module spaudiopy.process*), 54

L

lagrange_delay() (*in module spaudiopy.process*), 55
load_audio() (*in module spaudiopy.io*), 4
load_Fliege_Maier_nodes() (*in module spaudiopy.grids*), 66
load_hrirs() (*in module spaudiopy.io*), 4
load_layout() (*in module spaudiopy.io*), 7
load_lebedev() (*in module spaudiopy.grids*), 65
load_maxDet() (*in module spaudiopy.grids*), 67
load_n_design() (*in module spaudiopy.grids*), 64
load_sdm() (*in module spaudiopy.io*), 6
load_sofa_data() (*in module spaudiopy.io*), 5
load_sofa_hrirs() (*in module spaudiopy.io*), 5
load_t_design() (*in module spaudiopy.grids*), 63
loudspeaker_signals() (*spaudiopy.decoder.LoudspeakerSetup method*), 33
LoudspeakerSetup (*class in spaudiopy.decoder*), 30

M

mad() (*in module spaudiopy.decoder*), 43
magls_bin() (*in module spaudiopy.decoder*), 51
match_loudness() (*in module spaudiopy.process*), 54
matlab_sph2cart() (*in module spaudiopy.utils*), 61
max_rE_weights() (*in module spaudiopy.sph*), 19
maxre_modal_weights() (*in module spaudiopy.sph*), 25
mode_strength() (*in module spaudiopy.sph*), 22
module
    spaudiopy, 3
    spaudiopy.decoder, 28
    spaudiopy.grids, 62
    spaudiopy.io, 3
    spaudiopy.parsa, 71
    spaudiopy.plot, 81
    spaudiopy.process, 52
    spaudiopy.sig, 7
    spaudiopy.sph, 11
    spaudiopy.utils, 60
MonoSignal (*class in spaudiopy.sig*), 7
MultiSignal (*class in spaudiopy.sig*), 8

N

n3d_to_sn3d() (*in module spaudiopy.sph*), 16
nearest_hrirs() (*spaudiopy.sig.HRIRs method*), 10
nearest_idx() (*spaudiopy.sig.HRIRs method*), 10
nearest_loudspeaker() (*in module spaudiopy.decoder*), 48

P

platonic_solid() (*in module spaudiopy.sph*), 16
play() (*spaudiopy.sig.AmbiBSignal method*), 9
play() (*spaudiopy.sig.MonoSignal method*), 8
play() (*spaudiopy.sig.MultiSignal method*), 9
polar() (*in module spaudiopy.plot*), 84
pop_triangles() (*spaudiopy.decoder.LoudspeakerSetup method*), 31
pressure_on_sphere() (*in module spaudiopy.sph*), 23
project_on_sphere() (*in module spaudiopy.sph*), 22
pseudo_intensity() (*in module spaudiopy.parsa*), 71
pulsed_noise() (*in module spaudiopy.process*), 58

R

r_E() (*in module spaudiopy.sph*), 21
rad2deg() (*in module spaudiopy.utils*), 61
render_binaural_loudspeaker_sdm() (*in module spaudiopy.parsa*), 79
render_bsdm() (*in module spaudiopy.parsa*), 72
render_stereo_sdm() (*in module spaudiopy.parsa*), 79
repeat_per_order() (*in module spaudiopy.sph*), 22
resample() (*spaudiopy.sig.AmbiBSignal method*), 9
resample() (*spaudiopy.sig.MonoSignal method*), 8
resample() (*spaudiopy.sig.MultiSignal method*), 8
resample_hrirs() (*in module spaudiopy.process*), 52
resample_signal() (*in module spaudiopy.process*), 53
resample_spectrum() (*in module spaudiopy.process*), 53
rms() (*in module spaudiopy.utils*), 62
rotate_sh() (*in module spaudiopy.sph*), 15
rotation_euler() (*in module spaudiopy.utils*), 61
rotation_rodrigues() (*in module spaudiopy.utils*), 61
rotation_vecvec() (*in module spaudiopy.utils*), 61

S

sad() (*in module spaudiopy.decoder*), 43
save() (*spaudiopy.sig.AmbiBSignal method*), 9
save() (*spaudiopy.sig.MonoSignal method*), 8
save() (*spaudiopy.sig.MultiSignal method*), 9
save_audio() (*in module spaudiopy.io*), 4
save_layout() (*in module spaudiopy.io*), 7
sdm_post_equalization() (*in module spaudiopy.parsa*), 80
sdm_post_equalization2() (*in module spaudiopy.parsa*), 80
separate_cov() (*in module spaudiopy.parsa*), 75
set_aspect_equal3d() (*in module spaudiopy.plot*), 86
sh2bin() (*in module spaudiopy.decoder*), 50
sh_bar() (*in module spaudiopy.plot*), 83
sh_beamform() (*in module spaudiopy.parsa*), 73
sh_beamformer_from_pattern() (*in module spaudiopy.parsa*), 72
sh_coeffs() (*in module spaudiopy.plot*), 82
sh_coeffs_overlay() (*in module spaudiopy.plot*), 82
sh_coeffs_subplot() (*in module spaudiopy.plot*), 82
sh_lcmv() (*in module spaudiopy.parsa*), 78
sh_matrix() (*in module spaudiopy.sph*), 12
sh_mult() (*in module spaudiopy.sph*), 28
sh_music() (*in module spaudiopy.parsa*), 75
sh_mvdr() (*in module spaudiopy.parsa*), 77
sh_rms_map() (*in module spaudiopy.plot*), 83
sh_rotation_matrix() (*in module spaudiopy.sph*), 14
sh_sector_beamformer() (*in module spaudiopy.parsa*), 79
sh_to_b() (*in module spaudiopy.sph*), 16
sh_to_b() (*spaudiopy.sig.AmbiBSignal class method*), 9
show() (*spaudiopy.decoder.LoudspeakerSetup method*), 33
sht() (*in module spaudiopy.sph*), 13
sht_lstsq() (*in module spaudiopy.sph*), 13
sn3d_to_n3d() (*in module spaudiopy.sph*), 16
sofa_to_sh() (*in module spaudiopy.io*), 5
sort_vertices() (*in module spaudiopy.decoder*), 33
soundfield_to_b() (*in module spaudiopy.sph*), 16
spaudiopy
    module, 3
spaudiopy.decoder
    module, 28
spaudiopy.grids
    module, 62
spaudiopy.io
    module, 3
spaudiopy.parsa
    module, 71
spaudiopy.plot
    module, 81
spaudiopy.process
    module, 52
spaudiopy.sig
    module, 7
spaudiopy.sph
    module, 11
spaudiopy.utils
    module, 60
spectrum() (*in module spaudiopy.plot*), 81
sph2cart() (*in module spaudiopy.utils*), 61
sph_filterbank_reconstruction_factor() (*in module spaudiopy.sph*), 27
spherical_function() (*in module spaudiopy.plot*), 82
spherical_function_map() (*in module spaudiopy.plot*), 83
spherical_hn2() (*in module spaudiopy.sph*), 22
src_to_b() (*in module spaudiopy.sph*), 17
src_to_sh() (*in module spaudiopy.sph*), 17
stack() (*in module spaudiopy.utils*), 62
subband_levels() (*in module spaudiopy.process*), 56

T

test_diff() (*in module spaudiopy.utils*), 62
transfer_function() (*in module spaudiopy.plot*), 82
trim() (*spaudiopy.sig.AmbiBSignal method*), 9
trim() (*spaudiopy.sig.MonoSignal method*), 8
trim() (*spaudiopy.sig.MultiSignal method*), 8
trim_audio() (*in module spaudiopy.sig*), 11

U

unity_gain() (*in module spaudiopy.sph*), 24
update_hrirs() (*spaudiopy.sig.HRIRs method*), 10

V

vbap() (*in module spaudiopy.decoder*), 34
vbip() (*in module spaudiopy.decoder*), 34
vec2dir() (*in module spaudiopy.utils*), 61

W

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

Z

zeropole() (*in module spaudiopy.plot*), 82