Sound Field Analysis Toolbox

Release 474b106

SFA Toolbox Developers

Sep 22, 2017

Contents

1 Usage **1**
  1.1 Requirements 1
  1.2 How to Get Started 1

2 Modal Beamforming **2**
  2.1 Angular 2
  2.2 Radial 3

3 Utilities **4**

The Sound Field Analysis Toolbox for NumPy/Python provides implementations of various techniques for the analysis of sound fields and beamforming using microphone arrays.

Source code and issue tracker: https://github.com/spatialaudio/sfa-numpy

License: MIT – see the file `LICENSE` for details.

Quick start:

- Install NumPy, SciPy and for the examples Matplotlib
- `git clone https://github.com/spatialaudio/sfa-numpy.git`
- `cd sfa-numpy`
- `python setup.py install --user`

---

1 Usage

1.1 Requirements

Obviously, you'll need Python[1]. We normally use Python 3.x, but it *should* also work with Python 2.x. NumPy[2] and SciPy[3] are needed for the calculations. If you also want to plot the results in the examples, you'll need matplotlib[4].

[1] http://www.python.org/
[2] http://www.numpy.org/
[3] http://www.scipy.org/scipylib/
[4] http://matplotlib.org/

Instead of installing all of them separately, you should probably get a Python distribution that already includes everything, e.g. Anaconda[5].

1.2 How to Get Started

Various examples are located in the examples directory.

2 Modal Beamforming

Submodules for modal beamforming

2.1 Angular

`micarray.modal.angular.sht_matrix(N, azi, elev, weights=None)`

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

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

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

where

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

Parameters

- **N** (*int*) – Maximum order.
- **azi** (*(Q,) array_like*) – Azimuth.
- **elev** (*(Q,) array_like*) – Elevation.
- **weights** (*(Q,) array_like, optional*) – Quadrature weights.

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

`micarray.modal.angular.Legendre_matrix(N, ctheta)`

Matrix of weighted Legendre Polynominals.

Computes a matrix of weighted Legendre polynominals up to order N for the given angles

$$L_n(\theta) = \frac{2n+1}{4\pi} P_n(\theta)$$

Parameters

- **N** (*int*) – Maximum order.
- **ctheta** (*(Q,) array_like*) – Angles.

Returns **Lmn** (*((N+1), Q) numpy.ndarray*) – Matrix containing Legendre polynominals.

`micarray.modal.angular.grid_equal_angle(n)`

Equi-angular sampling points on a sphere.

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

[5] http://docs.continuum.io/anaconda/

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

Returns

- **azi** (*array_like*) – Azimuth.
- **elev** (*array_like*) – Elevation.
- **weights** (*array_like*) – Quadrature weights.

`micarray.modal.angular.`**`grid_gauss`**`(`*n*`)`

Gauss-Legendre sampling points on sphere.

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

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

Returns

- **azi** (*array_like*) – Azimuth.
- **elev** (*array_like*) – Elevation.
- **weights** (*array_like*) – Quadrature weights.

2.2 Radial

`micarray.modal.radial.`**`spherical_pw`**`(`*N*, *k*, *r*, *setup*`)`

Radial coefficients for a plane wave.

Computes the radial component of the spherical harmonics expansion of a plane wave impinging on a spherical array.

$$\mathring{P}_n(k) = 4\pi i^n b_n(kr)$$

Parameters

- **N** (*int*) – Maximum order.
- **k** (*(M,) array_like*) – Wavenumber.
- **r** (*float*) – Radius of microphone array.
- **setup** (*{'open', 'card', 'rigid'}*) – Array configuration (open, cardioids, rigid).

Returns **bn** (*(M, N+1) numpy.ndarray*) – Radial weights for all orders up to N and the given wavenumbers.

`micarray.modal.radial.`**`spherical_ps`**`(`*N*, *k*, *r*, *rs*, *setup*`)`

Radial coefficients for a point source.

Computes the radial component of the spherical harmonics expansion of a point source impinging on a spherical array.

$$\mathring{P}_n(k) = 4\pi(-i)kh_n^{(2)}(kr_s)b_n(kr)$$

Parameters

- **N** (*int*) – Maximum order.
- **k** (*(M,) array_like*) – Wavenumber.
- **r** (*float*) – Radius of microphone array.
- **rs** (*float*) – Distance of source.
- **setup** (*{'open', 'card', 'rigid'}*) – Array configuration (open, cardioids, rigid).

Returns **bn** (*(M, N+1) numpy.ndarray*) – Radial weights for all orders up to N and the given wavenumbers.

`micarray.modal.radial.`**`weights`**(*N*, *kr*, *setup*)

Radial weighing functions.

Computes the radial weighting functions for diferent array types (cf. eq.(2.62), Rafaely 2015).

For instance for an rigid array

$$b_n(kr) = j_n(kr) - \frac{j_n'(kr)}{h_n^{(2)\prime}(kr)} h_n^{(2)}(kr)$$

Parameters

- **N** (*int*) – Maximum order.
- **kr** (*(M,) array_like*) – Wavenumber * radius.
- **setup** (*{'open', 'card', 'rigid'}*) – Array configuration (open, cardioids, rigid).

Returns **bn** (*(M, N+1) numpy.ndarray*) – Radial weights for all orders up to N and the given wavenumbers.

`micarray.modal.radial.`**`regularize`**(*dn*, *a0*, *method*)

Regularization (amplitude limitation) of radial filters.

Amplitude limitation of radial filter coefficients, methods according to (cf. Rettberg, Spors : DAGA 2014)

Parameters

- **dn** (*numpy.ndarray*) – Values to be regularized
- **a0** (*float*) – Parameter for regularization (not required for all methods)
- **method** (*{'none', 'discard', 'softclip', 'Tikh', 'wng'}*) – Method used for regularization/amplitude limitation (none, discard, hardclip, Tikhonov, White Noise Gain).

Returns

- **dn** (*numpy.ndarray*) – Regularized values.
- **hn** (*array_like*)

`micarray.modal.radial.`**`diagonal_mode_mat`**(*bk*)

Diagonal matrix of radial coefficients for all modes/wavenumbers.

Parameters **bk** (*(M, N+1) numpy.ndarray*) – Vector containing values for all wavenumbers M and modes up to order N

Returns **Bk** (*(M, (N+1)**2, (N+1)**2) numpy.ndarray*) – Multidimensional array containing diagnonal matrices with input vector on main diagonal.

3 Utilities

`micarray.util.`**`norm_of_columns`**(*A*, *p=2*)

Vector p-norm of each column of a matrix.

Parameters

- **A** (*array_like*) – Input matrix.
- **p** (*int, optional*) – p-th norm.

Returns *array_like* – p-norm of each column of A.

`micarray.util.`**`coherence_of_columns`**(*A*)

Mutual coherence of columns of A.

Parameters

- **A** (*array_like*) – Input matrix.

- **p** (*int, optional*) – p-th norm.

Returns *array_like* – Mutual coherence of columns of A.

`micarray.util.`**`asarray_1d`**(*a*, ***kwargs*)

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

Returns *a* converted to a `numpy.ndarray`[6] 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.

[6] https://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.html#numpy.ndarray