1 Requirements

Obviously, you’ll need Python\textsuperscript{2}. We normally use Python 3.x, but it should also work with Python 2.x. NumPy\textsuperscript{3} and SciPy\textsuperscript{4} are needed for the calculations. If you also want to plot the resulting sound fields, you’ll need matplotlib\textsuperscript{5}.

\footnote{1http://github.com/sfstoolbox/sfs/}
\footnote{2http://www.python.org/}
\footnote{3http://www.numpy.org/}
\footnote{4http://www.scipy.org/scipylib/}
\footnote{5http://matplotlib.org/}

Documentation: http://sfs.rtfd.org/
Code: http://github.com/sfstoolbox/sfs-python/

1 Requirements

Obviously, you’ll need Python\textsuperscript{2}. We normally use Python 3.x, but it should also work with Python 2.x. NumPy\textsuperscript{3} and SciPy\textsuperscript{4} are needed for the calculations. If you also want to plot the resulting sound fields, you’ll need matplotlib\textsuperscript{5}.

\footnote{1http://github.com/sfstoolbox/sfs/}
\footnote{2http://www.python.org/}
\footnote{3http://www.numpy.org/}
\footnote{4http://www.scipy.org/scipylib/}
\footnote{5http://matplotlib.org/}
Instead of installing all of them separately, you should probably get a Python distribution that already includes everything, e.g. Anaconda.

## 2 Installation

Currently, the package is not yet available on PyPI (but coming soon!), for now you should get it from Github:

```bash
git clone https://github.com/sfstoolbox/sfs-python.git
cd sfs-python
python setup.py develop --user
```

## 3 How to Get Started

Various examples are located in the directory examples/

- **sound_field_synthesis.py**: Illustrates the general usage of the toolbox
- **horizontal_plane_arrays.py**: Computes the sound fields for various techniques, virtual sources and loudspeaker array configurations
- **soundfigures.py**: Illustrates the synthesis of sound figures with Wave Field Synthesis

## 4 Further Reading

### 4.1 Different Types of Sound Sources

This page shows some fundamental types of sound sources. At the same time, it also shows how to calculate and plot sound fields.

First we have to import the sfs library. Let’s also import NumPy and matplotlib, as we’ll need them later.

```python
In [1]: import sfs
In [2]: import numpy as np
In [3]: import matplotlib.pyplot as plt
In [4]: plt.rcParams['figure.figsize'] = 8, 4  # inch
```

We specify the position of the sound source in meters and its frequency in Hertz, but we immediately convert this to an angular frequency in radians:

```python
In [5]: x0 = [1.5, 1, 0]
In [6]: f = 500  # Hz
In [7]: omega = 2 * np.pi * f
```

Now we create a spatial grid on which we will evaluate (and later plot) the sound field. For the x and y component, we give a minimum and maximum value, for the z component we only specify a single scalar 0, i.e. we’re interested in the horizontal plane at $z = 0$. For more options, have a look at the documentation of `xyz_grid()`.
In [8]: grid = sfs.util.xyz_grid([-2, 3], [-1, 2], 0, spacing=0.02)

Now we have all we need to calculate our first sound source. Let’s use the function `sfs.mono.source.point()` to create a point source:

In [9]: p_point = sfs.mono.source.point(omega, x0, None, grid)

In [10]: p_point.shape
Out[10]: (151, 251)

In [11]: p_point
Out[11]:
\[
\begin{array}{llllll}
1.40707234e-02+0.013846j, & 1.17526081e-02+0.01596715j, \\
9.12077963e-03+0.01770031j, & \ldots, & -2.48093081e-02+0.02042731j, \\
-2.2348126e-02+0.02288397j, & -1.96175283e-02+0.0250672j, \\
1.27925424e-02+0.01509854j, & 1.02795317e-02+0.01701049j, \\
7.48715084e-03+0.01850454j, & \ldots, & -2.77258311e-02+0.0166625j, \\
-2.56206914e-02+0.01949031j, & -2.32100825e-02+0.02208109j, \\
1.14139930e-02+0.01622509j, & 8.72619816e-03+0.01791174j, \\
5.79591834e-03+0.01953813j, & \ldots, & -3.00658868e-02+0.01249827j, \\
-2.83742417e-02+0.01564392j, & -2.63995316e-02+0.01859536j, \\
\ldots,
\end{array}
\]

As we can see, the result is a two-dimensional array (because we asked for only one z value) of complex numbers. Those numbers look already quite impressive, but maybe it’s even better if we plot them?

In [12]: sfs.plot.soundfield(p_point, grid);

In [13]: plt.title("Point Source at {} m".format(x0));

Normalization ... multiply by $4\pi$ ...

In [14]: p_point *= 4 * np.pi

In [15]: sfs.plot.soundfield(p_point, grid);

In [16]: plt.title("Point Source at {} m".format(x0));
Now it looks nice!

For more plotting options, have a look at the documentation of `sfs.plot.soundfield()`.

Now we create a line source using the function `sfs.mono.source.line()`:

```python
In [17]: p_line = sfs.mono.source.line(omega, x0, None, grid)
In [18]: p_line *= np.exp(-1j*7*np.pi/4) / np.sqrt(1/(8*np.pi*omega/sfs.defs.c))
In [19]: sfs.plot.soundfield(p_line, grid);
In [20]: plt.title("Line Source at {} m\".format(x0[:2]));
```

In order to compare the two, let’s try to plot them side-by-side.

```python
In [21]: f, (ax1, ax2) = plt.subplots(ncols=2, sharex=True, sharey=True)
In [22]: f.set_figwidth(f.get_figwidth() * 2)
In [23]: f.subplots_adjust(wspace=0.05)
In [24]: sfs.plot.soundfield(p_point, grid, ax=ax1, colorbar=False);
In [25]: ax1.set_title("Point Source");
In [26]: im = sfs.plot.soundfield(p_line, grid, ax=ax2, colorbar=False, ylabel="")
In [27]: ax2.set_title("Line Source");
In [28]: f.colorbar(im, ax=[ax1, ax2]);
```
Finally, let’s have a look at a plane wave, which can be created with `sfs.mono.source.plane()`.

```python
In [29]: plt.close()  # get rid of the double-width figure from above
In [30]: direction = 45  # degree
In [31]: n0 = sfs.util.normal(np.radians(direction), np.radians(90))
In [32]: p_plane = sfs.mono.source.plane(omega, x0, n0, grid)
In [33]: sfs.plot.soundfield(p_plane, grid);
In [34]: plt.title("Plane wave with direction {} degree".format(direction));
```

4.2 Different Types of Loudspeaker Arrays

```python
In [1]: import sfs
In [2]: import numpy as np
In [3]: import matplotlib.pyplot as plt
In [4]: plt.rcParams['figure.figsize'] = 8, 4  # inch
In [5]: dx = 0.1  # secondary source spacing
In [6]: N = 16  # number of secondary sources
In [7]: center = 0.3, 0.7, 0
In [8]: normal = sfs.util.normal(np.radians(-70), np.radians(90))
In [9]: x0, n0, a0 = sfs.array.linear(N, dx, center=center, n0=normal)
In [10]: sfs.plot.loudspeaker_2d(x0, n0, a0)
In [11]: plt.axis('equal');
```
4.3 API Documentation

Loudspeaker Arrays

Compute positions of various secondary source distributions.

```python
sfs.array.linear(N, dx, center=[0, 0, 0], n0=[1, 0, 0])
```
Linear secondary source distribution.

```python
sfs.array.linear_nested(N, dx1, dx2, center=[0, 0, 0], n0=[1, 0, 0])
```
Nested linear secondary source distribution.

```python
sfs.array.linear_random(N, dy1, dy2, center=[0, 0, 0], n0=[1, 0, 0])
```
Randomly sampled linear array.

```python
sfs.array.circular(N, R, center=[0, 0, 0])
```
Circular secondary source distribution parallel to the xy-plane.

```python
sfs.array.rectangular(Nx, dx, Ny, dy, center=[0, 0, 0], n0=None)
```
Rectangular secondary source distribution.

```python
sfs.array.rounded_edge(Nxy, Nr, dx, center=[0, 0, 0], n0=None)
```
Array along the xy-axis with rounded edge at the origin.

**Parameters**

- `Nxy` (**integer**) – Number of secondary sources along x- and y-axis.
- `Nr` (**integer**) – Number of secondary sources in rounded edge. Radius of edge is adjusted to equidistant sampling along entire array.
- `center` (**triple of floats**) – Position of edge.
- `n0` (**triple of floats**) – Normal vector of array. Default orientation is along xy-axis.

**Returns**

- `positions` (**list of triplets of floats**) – positions of secondary sources
- `directions` (**list of triplets of floats**) – orientations (normal vectors) of secondary sources
- `weights` (**list of floats**) – integration weights of secondary sources

```python
sfs.array.planar(Ny, dy, Nz, dz, center=[0, 0, 0], n0=None)
```
Planar secondary source distribution.

```python
sfs.array.cube(Nx, dx, Ny, dy, Nz, dz, center=[0, 0, 0], n0=None)
```
Cube shaped secondary source distribution.

```python
sfs.array.sphere_load(fname, radius, center=[0, 0, 0])
```
Spherical secondary source distribution loaded from datafile.

ASCII Format (see MATLAB SFS Toolbox) with 4 numbers (3 position, 1 weight) per secondary source located on the unit circle.

```python
sfs.array.load(fname, center=[0, 0, 0], n0=None)
```
Load secondary source positions from datafile.

Comma Seperated Values (CSV) format with 7 values (3 positions, 3 directions, 1 weight) per secondary source

```python
sfs.array.weights_linear(positions)
```
Calculate loudspeaker weights for a linear array.

The linear array has to be parallel to the y-axis.

```python
sfs.array.weights_closed(positions)
```
Calculate loudspeaker weights for a simply connected array.

The weights are calculated according to the midpoint rule
Note: The loudspeaker positions have to be ordered on the closed contour

**Tapering**

Weights (tapering) for the driving function.

- **sfs.tapering.none** *(active)*
  - No tapering window.

- **sfs.tapering.kaiser** *(active)*
  - Kaiser tapering window.

- **sfs.tapering.tukey** *(active, alpha)*
  - Tukey tapering window.

**Monochromatic Sources**

Compute the sound field generated by a sound source

- **sfs.mono.source.point** *(omega, x0, n0, grid, c=None)*
  - Point source.
  
  \[
  G(x-x0, w) = \frac{1}{4\pi} \frac{e^{-j \omega/c |x-x0|}}{|x-x0|}
  \]

- **sfs.mono.source.line** *(omega, x0, n0, grid, c=None)*
  - Line source parallel to the z-axis.
  
  Note: third component of x0 is ignored.

  \[
  G(x-x0, w) = -j/4 \frac{H0(\omega/c |x-x0|)}{|x-x0|}
  \]

- **sfs.mono.source.line_dipole** *(omega, x0, n0, grid, c=None)*
  - Line source with dipole characteristics parallel to the z-axis.
  
  Note: third component of x0 is ignored.

  \[
  G(x-x0, w) = jk/4 \frac{H1(\omega/c |x-x0|) \cos(\phi)}{|x-x0|^{3/2}}
  \]

- **sfs.mono.source.plane** *(omega, x0, n0, grid, c=None)*
  - Plane wave.

  \[
  G(x, w) = e^{-i \omega/n x}
  \]

**Monochromatic Driving Functions**

Compute driving functions for various systems.

- **sfs.mono.drivingfunction.wfs_2d_line** *(omega, x0, n0, xs, c=None)*
  - Line source by 2-dimensional WFS.

  \[
  D(x0,k) = j k (x0-xs) n0 / |x0-xs| + H1(k |x0-xs|)
  \]

- **sfs.mono.drivingfunction.wfs_2d_point** *(omega, x0, n0, xs, c=None)*
  - Point source by two- or three-dimensional WFS.

  \[
  D(x0,k) = j k \frac{e^{-j k |x0-xs|}}{|x0-xs|^{3/2}}
  \]
sfs.mono.drivingfunction.wfs_25d_point(omega, x0, n0, xs, xref=[0, 0, 0], c=None, omalias=None)

Point source by 2.5-dimensional WFS.

\[
D(x_0, k) = \frac{(x_0-x_s) n_0}{|x_0-x_s|^{3/2}} e^{-j k |x_0-x_s|}
\]

sfs.mono.drivingfunction.wfs_3d_point(omega, x0, n0, xs, c=None)

Point source by two- or three-dimensional WFS.

\[
D(x_0, k) = j k \frac{(x_0-x_s) n_0}{|x_0-x_s|^{3/2}} e^{-j k |x_0-x_s|}
\]

sfs.mono.drivingfunction.wfs_2d_plane(omega, x0, n0, n=[0, 1, 0], c=None)

Plane wave by two- or three-dimensional WFS.

Eq.(17) from [Spors et al, 2008]:

\[
D(x_0, k) = j k n n_0 e^{-j k n x_0}
\]

sfs.mono.drivingfunction.wfs_25d_plane(omega, x0, n0, n=[0, 1, 0], xref=[0, 0, 0], c=None, omalias=None)

Plane wave by 2.5-dimensional WFS.

\[
D_{2.5D}(x_0, w) = \frac{1}{2\pi r_0} \sum_{m=-N}^{N} h_m (w/c r_0) e^{i m (\phi_0-\phi)}
\]

sfs.mono.drivingfunction.delay_3d_plane(omega, x0, n0, n=[0, 1, 0], c=None)

Plane wave by simple delay of secondary sources.

sfs.mono.drivingfunction.source_selection_plane(n0, n)

Secondary source selection for a plane wave.

Eq.(13) from [Spors et al, 2008]

sfs.mono.drivingfunction.source_selection_point(n0, x0, xs)

Secondary source selection for a point source.

Eq.(15) from [Spors et al, 2008]

sfs.mono.drivingfunction.source_selection_all(N)

Select all secondary sources.

sfs.mono.drivingfunction.nfchoa_2d_plane(omega, x0, r0, n=[0, 1, 0], c=None)

Point source by 2.5-dimensional WFS.

sfs.mono.drivingfunction.nfchoa_25d_point(omega, x0, r0, xs, c=None)

Point source by 2.5-dimensional WFS.
Monochromatic Sound Fields

Computation of synthesized sound fields.

sfs.mono.synthesized.generic(omega, x0, n0, d, grid, c=None, source=<function point>)
Compute sound field for a generic driving function

sfs.mono.synthesized.shiftphase(p, phase)
Shift phase of a sound field.

Plotting

Plot sound fields etc

sfs.plot.virtualsource_2d(xs, ns=None, type='point', ax=None)
Draw position/orientation of virtual source.

sfs.plot.reference_2d(xref, size=0.1, ax=None)
Draw reference/normalization point

sfs.plot.secondarysource_2d(x0, n0, grid=None)
Simple plot of secondary source locations.
sfs.plot.loudspeaker_2d(x0, n0, a0=None, w=0.08, h=0.08, index=False, grid=None)

Draw loudspeaker symbols at given locations, angles.

sfs.plot.loudspeaker_3d(x0, n0, a0=None, w=0.08, h=0.08)

Plot positions and normal vectors of a 3D secondary source distribution.

sfs.plot.soundfield(p, grid, xnorm=None, colorbar=True, cmap='coolwarm_clip', ax=None, xlabel='x (m)', ylabel='y (m)', vmax=2.0, vmin=-2.0, **kwargs)

Two-dimensional plot of sound field.

sfs.plot.level(p, grid, xnorm=None, colorbar=True, cmap='coolwarm_clip', ax=None, xlabel='x (m)', ylabel='y (m)', vmax=3.0, vmin=-50, **kwargs)

Two-dimensional plot of level (dB) of sound field.

Utilities

Various utility functions.

sfs.util.rotation_matrix(n1, n2)

Compute rotation matrix for rotation from n1 to n2.

sfs.util.wavenumber(omega, c=None)

Compute the wavenumber for a given radial frequency.

sfs.util.normal(alpha, beta)

Compute normal vector from azimuth, colatitude.

sfs.util.sph2cart(alpha, beta, r)

Spherical to cartesian coordinates.

sfs.util.cart2sph(x, y, z)

Cartesian to spherical coordinates.

sfs.util.asarray_1d(a, **kwargs)

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

Returns a converted to a numpy.array and stripped of all singleton dimensions. The result must have exactly one dimension. If not, an error is raised.

sfs.util.asarray_of_arrays(a, **kwargs)

Convert the input to an array consisting of arrays.

A one-dimensional array with dtype=object is returned, containing the elements of a as arrays (whose dtype and other options can be specified with **kwargs).

sfs.util.strict_arange(start, stop, step=1, endpoint=False, dtype=None, **kwargs)

Like numpy.arange()11, but compensating numeric errors.

Unlike numpy.arange()12, but similar to numpy.linspace()13, providing endpoint=True includes both endpoints.

Parameters

• start, stop, step, dtype – See numpy.arange()14.

• endpoint – See numpy.linspace()15.

Note: With endpoint=True, the difference between start and end value must be an integer multiple of the corresponding spacing value!

• **kwargs – All further arguments are forwarded to numpy.isclose()16.

11http://docs.scipy.org/doc/numpy/reference/generated/numpy.arange.html#numpy.arange
12http://docs.scipy.org/doc/numpy/reference/generated/numpy.arange.html#numpy.arange
13http://docs.scipy.org/doc/numpy/reference/generated/numpy.linspace.html#numpy.linspace
14http://docs.scipy.org/doc/numpy/reference/generated/numpy.linspace.html#numpy.linspace
15http://docs.scipy.org/doc/numpy/reference/generated/numpy.linspace.html#numpy.linspace
16http://docs.scipy.org/doc/numpy/reference/generated/numpy.isclose.html#numpy.isclose
sfs.util.xyz_grid(x, y, z, spacing, endpoint=True, **kwargs)
Create a grid with given range and spacing.

Parameters

- **x, y, z** (float or pair of floats) – Inclusive range of the respective coordinate or a single value if only a slice along this dimension is needed.
- **spacing** (float or triple of floats) – Grid spacing. If a single value is specified, it is used for all dimensions, if multiple values are given, one value is used per dimension. If a dimension (x, y or z) has only a single value, the corresponding spacing is ignored.
- **endpoint** (bool) – If True (the default), the endpoint of each range is included in the grid. Use False to get a result similar to numpy.arange(). See strict_arange().
- ****kwargs – All further arguments are forwarded to strict_arange().

Returns

- list of numpy.ndarrays – A grid that can be used for sound field calculations.

See also:

strict_arange(), numpy.meshgrid()  
sfs.util.normalize(p, grid, xnorm)
Normalize sound field wrt position xnorm.

sfs.util.level(p, grid, x)
Determine level at position x in the sound field p.

5 Indices

- genindex
- modindex

---

17http://docs.scipy.org/doc/numpy/reference/generated/numpy.arange.html#numpy.arange
18http://docs.scipy.org/doc/numpy/reference/generated/numpy.arange.html#numpy.arange
19http://docs.scipy.org/doc/numpy/reference/generated/numpy.meshgrid.html#numpy.meshgrid
Python Module Index

S

sfs.array, 5
sfs.mono.drivingfunction, 7
sfs.mono.source, 6
sfs.mono.synthesized, 9
sfs.plot, 9
sfs.tapering, 6
sfs.util, 9
Index

A
asarray_1d() (in module sfs.util), 10
asarray_of_arrays() (in module sfs.util), 10

C
cart2sph() (in module sfs.util), 10
circular() (in module sfs.array), 5
cube() (in module sfs.array), 6

delay_3d_plane() (in module sfs.mono.drivingfunction), 8
delay_3d_plane() (in module sfs.mono.drivingfunction), 8

generic() (in module sfs.mono.synthesized), 9

K
kaiser() (in module sfs.tapering), 6

L
level() (in module sfs.plot), 9
level() (in module sfs.util), 11
line() (in module sfs.mono.source), 7
line_dipole() (in module sfs.mono.source), 7
linear() (in module sfs.array), 5
linear_nested() (in module sfs.array), 5
linear_random() (in module sfs.array), 5
load() (in module sfs.array), 6
loudspeaker_2d() (in module sfs.plot), 9
loudspeaker_3d() (in module sfs.plot), 9

N
nfchoa_2d_point() (in module sfs.mono.drivingfunction), 8
nfchoa_2d_point() (in module sfs.mono.drivingfunction), 8

P
planar() (in module sfs.array), 6
plane() (in module sfs.mono.source), 7
point() (in module sfs.mono.source), 6

R
rectangular() (in module sfs.array), 5
reference_2d() (in module sfs.plot), 9
rotation_matrix() (in module sfs.util), 9
rounded_edge() (in module sfs.array), 5

S
sdm_2d_point() (in module sfs.mono.drivingfunction), 8
sdm_2d_point() (in module sfs.mono.drivingfunction), 9

T
tukey() (in module sfs.tapering), 6

V
virtualsource_2d() (in module sfs.plot), 9

W
wavenumber() (in module sfs.util), 9
weights_closed() (in module sfs.array), 6
weights_linear() (in module sfs.array), 6
wfs_2d_point() (in module sfs.mono.drivingfunction), 7
wfs_2d_point() (in module sfs.mono.drivingfunction), 7
wfs_2d_point() (in module sfs.mono.drivingfunction), 7
wfs_2d_point() (in module sfs.mono.drivingfunction), 7
X

xyz_grid() (in module sfs.util), 10