Python implementation of the Sound Field Synthesis Toolbox\(^1\).

**Documentation:** [http://sfs.rtfd.org/](http://sfs.rtfd.org/)


**Python Package Index:** [http://pypi.python.org/pypi/sfs/](http://pypi.python.org/pypi/sfs/)

## 1 Requirements

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

---

\(^1\) [github.com/sfstoolbox/sfs/](http://github.com/sfstoolbox/sfs/)

\(^2\) [http://www.python.org/](http://www.python.org/)

\(^3\) [http://www.numpy.org/](http://www.numpy.org/)

\(^4\) [http://www.scipy.org/scipylib](http://www.scipy.org/scipylib)

\(^5\) [http://matplotlib.org/](http://matplotlib.org/)
Instead of installing all of them separately, you should probably get a Python distribution that already includes everything, e.g., Anaconda6.

## 2 Installation

Once you have installed the above-mentioned dependencies, you can use pip7 to download and install the latest release with a single command:

```
pip install sfs --user
```

If you want to install it system-wide for all users (assuming you have the necessary rights), you can just drop the `--user` option.

To un-install, use:

```
pip uninstall sfs
```

If you want to keep up with the latest and greatest development version you should get it from Github8:

```
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 API Documentation

### 4.1 Loudspeaker Arrays

Compute positions of various secondary source distributions.

```python
import sfs
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = 8, 4 # inch
plt.rcParams['axes.grid'] = True
sfs.array.linear(N, spacing, center=[0, 0, 0], n0=[1, 0, 0])
```

Linear secondary source distribution.

**Parameters**

- **N** (int) – Number of loudspeakers.
- **spacing** (float) – Distance (in metres) between loudspeakers.
- **center** ((3,) array_like, optional) – Coordinates of array center.

---

6 http://docs.continuum.io/anaconda/
8 http://github.com/sfstoolbox/sfs-python/
- **n0** ((3,) array_like, optional) – Normal vector of array.

Returns

- **positions** ((N, 3) numpy.ndarray) – Positions of secondary sources
- **directions** ((N, 3) numpy.ndarray) – Orientations (normal vectors) of secondary sources
- **weights** ((N,) numpy.ndarray) – Weights of secondary sources

**Example**

```python
x0, n0, a0 = sfs.array.linear(16, 0.2, n0=[0, -1, 0])
sfs.plot.loudspeaker_2d(x0, n0, a0)
plt.axis('equal')
```

---

**sfs.array.linear_nested**(N, dx1, dx2, center=[0, 0, 0], n0=[1, 0, 0])

 Nested linear secondary source distribution.

**Example**

```python
x0, n0, a0 = sfs.array.linear_nested(16, 0.15, 0.2, n0=[0, -1, 0])
sfs.plot.loudspeaker_2d(x0, n0, a0)
plt.axis('equal')
```
sfs.array.linear_random(N, dy1, dy2, center=[0, 0, 0], n0=[1, 0, 0])
Randomly sampled linear array.

Example

```python
x0, n0, a0 = sfs.array.linear_random(12, 0.15, 0.4, n0=[0, -1, 0])
sfs.plot.loudspeaker_2d(x0, n0, a0)
plt.axis('equal')
```

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

Example

```python
x0, n0, a0 = sfs.array.circular(16, 1)
sfs.plot.loudspeaker_2d(x0, n0, a0, size=0.2, show_numbers=True)
plt.axis('equal')
```
sfs.array.rectangular \((N_x, d_x, N_y, d_y, center=[0, 0, 0], n0=None)\)
Rectangular secondary source distribution.

Example

```python
x0, n0, a0 = sfs.array.rectangular(8, 0.2, 4, 0.2)
sfs.plot.loudspeaker_2d(x0, n0, a0, show_numbers=True)
plt.axis('equal')
```

sfs.array.rounded_edge \((N_{xy}, N_r, d_x, center=[0, 0, 0], n0=None)\)
Array along the xy-axis with rounded edge at the origin.

Parameters

- **N_{xy} (int)** – Number of secondary sources along x- and y-axis.
- **N_r (int)** – Number of secondary sources in rounded edge. Radius of edge is adjusted to equidistant sampling along entire array.
- **center ((3,) array_like, optional)** – Position of edge.
- **n0 ((3,) array_like, optional)** – Normal vector of array. Default orientation is along xy-axis.
Returns

- **positions** ((N, 3) numpy.ndarray) – Positions of secondary sources
- **directions** ((N, 3) numpy.ndarray) – Orientations (normal vectors) of secondary sources
- **weights** ((N,) numpy.ndarray) – Integration weights of secondary sources

Example

```python
x0, n0, a0 = sfs.array.rounded_edge(8, 5, 0.2)
sfs.plot.loudspeaker_2d(x0, n0, a0)
plt.axis('equal')
```

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

Planar secondary source distribution.

sfs.array.cube(Nx, dx, Ny, dy, Nz, dz, center=[0, 0, 0], n0=None)

Cube shaped secondary source distribution.

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.

sfs.array.load(fname, center=[0, 0, 0], n0=None)

Load secondary source positions from datafile.

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

sfs.array.weights_linear(positions)

Calculate loudspeaker weights for a linear array.

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

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.
4.2 Tapering

Weights (tapering) for the driving function.

\[ \text{sfs.tapering.}\text{none} (\text{active}) \]
No tapering window.

\[ \text{sfs.tapering.}\text{kaiser} (\text{active}) \]
Kaiser tapering window.

\[ \text{sfs.tapering.}\text{tukey} (\text{active, alpha}) \]
Tukey tapering window.

4.3 Monochromatic Sources

Compute the sound field generated by a sound source.

```python
import sfs
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = 8, 4  # inch
x0 = 1.5, 1, 0
f = 500  # Hz
omega = 2 * np.pi * f
grid = sfs.util.xyz_grid([-2, 3], [-1, 2], 0, spacing=0.02)

sfs.mono.source.\text{point} (omega, x0, n0, grid, c=None)
Point source.

\[
G(x-x0, w) = \frac{1}{4\pi|x-x0|} e^{-j w/c |x-x0|}
\]

Examples

\[
p_{\text{point}} = \text{sfs.mono.source.point(omega, x0, None, grid)}
\]
\[
\text{sfs.plot.soundfield(p_point, grid)}
\]
\[
\text{plt.title("Point Source at {} m".format(x0))}
\]

Normalization ... multiply by \(4\pi\) ...
p_point *= 4 * np.pi
sfs.plot.soundfield(p_point, grid)
plt.title("Point Source at {} m (normalized)").format(x0))

\[
p(x-x0, w) = -j/4 \ H_0(w/c |x-x0|)
\]

\(sfs.mono.source.point_modal(\omega, x0, n0, grid, L, N=None, deltan=0, c=None)\)

Point source in a rectangular room using a modal room model.

**Parameters**

- **\(\omega\) (float)** – Frequency of source.
- **\(x0\) ((3,) array_like)** – Position of source.
- **\(n0\) ((3,) array_like)** – Normal vector (direction) of source (only required for compatibility).
- **\(grid\) (triple of numpy.ndarray)** – The grid that is used for the sound field calculations.
- **\(L\) ((3,) array_like)** – Dimensions of the rectangular room.
- **\(N\) ((3,) array_like or int, optional)** – Combination of modal orders in the three-spatial dimensions to calculate the sound field for or maximum order for all dimensions. If not given, the maximum modal order is approximately determined and the sound field is computed up to this maximum order.
- **\(deltan\) (float, optional)** – Absorption coefficient of the walls.
- **\(c\) (float, optional)** – Speed of sound.

**Returns** `numpy.ndarray` – Sound pressure at positions given by `grid`.

\(sfs.mono.source.line(\omega, x0, n0, grid, c=None)\)

Line source parallel to the z-axis.

Note: third component of \(x0\) is ignored.

\[
G(x-x0, w) = -j/4 \ H_0(w/c |x-x0|)
\]

**Examples**

```python
p_line = sfs.mono.source.line(omega, x0, None, grid)
sfs.plot.soundfield(p_line, grid)
plt.title("Line Source at {} m".format(x0[1:2]))
```
Normalization ...

```
p_line = np.exp(-1j*7*np.pi/4) / np.sqrt(1/(8*np.pi*omega/sfs.defs.c))
sfs.plot.soundfield(p_line, grid)
plt.title("Line Source at {} m (normalized)".format(x0[:2]))
```

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.

\( G(x-x0, w) = \frac{jk}{4} H_1 \left( \frac{w}{c} |x-x0| \right) \cos(\phi) \)

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

\( G(x, w) = e^{-i \frac{w}{c} n x} \)

Example
direction = 45  # degree
n0 = sfs.util.direction_vector(np.radians(direction))
p_plane = sfs.mono.source.plane(omega, x0, n0, grid)
sfs.plot.soundfield(p_plane, grid);
plt.title("Plane wave with direction {} degree".format(direction))

4.4 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.

\[ D(x_0, k) = j k \frac{(x_0-x_s) n_0}{|x_0-x_s|} + H_1(k |x_0-x_s|) \]

sfs.mono.drivingfunction.wfs_2d_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_25d_point(omega, x0, n0, xs, xref=[0, 0, 0], c=None, omalias=None)
Point source by 2.5-dimensional WFS.

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

sfs.mono.drivingfunction.wfs_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]:
**Equation (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.

\[ D_{2.5D}(x_0, w) = \sqrt{|j |xref-x_0| \ n \ n_0 \ e^{-j \ k \ n \ x_0}} \]

**sfs.mono.drivingfunction.wfs_3d_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_preeq** (omega, omalias, c)

Preemphasis for 2.5D WFS.

**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.

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

**sfs.mono.drivingfunction.nfchoa_25d_plane** (omega, x0, r0, n=[0, 1, 0], c=None)

Plane wave by 2.5-dimensional WFS.

\[
D_{25D}(\phi_0, w) = \frac{2i}{\sqrt{2\pi r_0}} \sum_{m=-N}^{N} (\frac{w}{c} m) e^{i m (\phi_0 - \phi_{pw})}
\]

**sfs.mono.drivingfunction.sdm_2d_line** (omega, x0, n0, xs, c=None)

Line source by two-dimensional SDM.

The secondary sources have to be located on the x-axis (y0=0). Derived from [Spors 2009, 126th AES Convention], Eq.(9), Eq.(4):

\[ D(x_0, k) = \]

**sfs.mono.drivingfunction.sdm_2d_plane** (omega, x0, n0, n=[0, 1, 0], c=None)

Plane wave by two-dimensional SDM.
The secondary sources have to be located on the x-axis (y0=0). Derived from [Ahrens 2011, Springer], Eq.(3.73), Eq.(C.5), Eq.(C.11):

\[ D(x_0, k) = k_{pw,y} \ast e^{-j \cdot k_{pw,x} \cdot x} \]

\[ sfs.mono.drivingfunction.sdm_{2.5d\ plane}(omega, x_0, n_0, n=[0, 1, 0], xref=[0, 0, 0], c=None) \]

Plane wave by 2.5-dimensional SDM.

The secondary sources have to be located on the x-axis (y0=0). Eq.(3.79) from [Ahrens 2011, Springer]:

\[ D_{2.5D}(x_0, \omega) = \]

\[ sfs.mono.drivingfunction.sdm_{2.5d\ point}(omega, x_0, n_0, x_s, xref=[0, 0, 0], c=None) \]

Point source by 2.5-dimensional SDM.

The secondary sources have to be located on the x-axis (y0=0). Driving function from [Spors 2010, 128th AES Convention], Eq.(24):

\[ D(x_0, k) = \]

\[ 4.5 \text{ Monochromatic Sound Fields} \]

Computation of synthesized sound fields.

\[ sfs.mono.synthesized.generic(omega, x_0, n_0, d, grid, c=None, source='<function point>') \]

Compute sound field for a generic driving function.

\[ sfs.mono.synthesized.shiftphase(p, phase) \]

Shift phase of a sound field.

\[ 4.6 \text{ Plotting} \]

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.

\[ sfs.plot.secondarysource_2d(x_0, n_0, grid=None) \]

Simple plot of secondary source locations.

\[ sfs.plot.loudspeaker_2d(x_0, n_0, a_0=0.5, size=0.08, show_numbers=False, ax=None) \]

Draw loudspeaker symbols at given locations and angles.

\textbf{Parameters}

- \textbf{x0} ((N, 3) \text{ array_like}) – Loudspeaker positions.
- \textbf{n0} ((N, 3) or (3,) \text{ array_like}) – Normal vector(s) of loudspeakers.
- \textbf{a0} (float or (N,) \text{ array_like}, optional) – Weighting factor(s) of loudspeakers.
- \textbf{size} (float, optional) – Size of loudspeakers in metres.
- \textbf{show_numbers} (bool, optional) – If True, loudspeaker numbers are shown.
- \textbf{grid} (triple of numpy.ndarray, optional) – If specified, only loudspeakers within the grid are shown.
- \textbf{ax} (Axes object, optional) – The loudspeakers are plotted into this Axes\(^9\) object or – if not specified – into the current axes.

\(^9\) http://matplotlib.sourceforge.net/api/axes_api.html#matplotlib.axes.Axes
sfs.plot.loudspeaker_3d(x0, n0, a0=None, w=0.08, h=0.08)
Plot positions and normals 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.

4.7 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.direction_vector(alpha, beta=1.5707963267948966)
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.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.

sfs.util.asarray_of_rows(a, **kwargs)
Convert to 2D array, turn column vector into row vector.

Returns a converted to a numpy.ndarray and stripped of all singleton dimensions. If the result has exactly one dimension, it is re-shaped into a 2D row vector.

sfs.util.asarray_of_arrays(a, **kwargs)
Convert the input to an array consisting of arrays. A one-dimensional numpy.ndarray with dtype=object is returned, containing the elements of a as numpy.ndarray (whose dtype and other options can be specified with **kwargs).

sfs.util.strict_range(start, stop, step=1, endpoint=False, dtype=None, **kwargs)
Like numpy.arange() 14, but compensating numeric errors.

Unlike numpy.arange() 15, but similar to numpy.linspace() 16, providing endpoint=True includes both endpoints.

Parameters

• start, stop, step, dtype – See numpy.arange() 17.

---

10 http://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.html#numpy.ndarray
11 http://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.html#numpy.ndarray
12 http://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.html#numpy.ndarray
13 http://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.html#numpy.ndarray
14 http://docs.scipy.org/doc/numpy/reference/generated/numpy.arange.html#numpy.arange
15 http://docs.scipy.org/doc/numpy/reference/generated/numpy.arange.html#numpy.arange
16 http://docs.scipy.org/doc/numpy/reference/generated/numpy.linspace.html#numpy.linspace
17 http://docs.scipy.org/doc/numpy/reference/generated/numpy.arange.html#numpy.arange
• **kwargs – All further arguments are forwarded to `numpy.isclose()`\(^{19}\).

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

---

Returns `numpy.ndarray` – Array of evenly spaced values. See `numpy.arange()`\(^{20}\).

`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 float) – Inclusive range of the respective coordinate or a single value if only a slice along this dimension is needed.
- **spacing** (float or triple of float) – 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, optional) – If `True` (the default), the endpoint of each range is included in the grid. Use `False` to get a result similar to `numpy.arange()`\(^{21}\). 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()`\(^{22}\)

`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`.

`sfs.util.broadcast_zip(*args)`

Broadcast arguments to the same shape and then use `zip()`\(^{23}\).

---

5 Indices

- genindex
- modindex

---


\(^{19}\) [http://docs.scipy.org/doc/numpy/reference/generated/numpy.isclose.html#numpy.isclose](http://docs.scipy.org/doc/numpy/reference/generated/numpy.isclose.html#numpy.isclose)


\(^{23}\) [http://docs.python.org/3/library/functions.html#zip](http://docs.python.org/3/library/functions.html#zip)
Python Module Index

S
sfs.array, 2
sfs.mono.drivingfunction, 10
sfs.mono.source, 7
sfs.mono.synthesized, 12
sfs.plot, 12
sfs.tapering, 6
sfs.util, 13
wfs_2d_point() (in module sfs.mono.drivingfunction), 10
wfs_3d_plane() (in module sfs.mono.drivingfunction), 11
wfs_3d_point() (in module sfs.mono.drivingfunction), 10

X

xyz_grid() (in module sfs.util), 14