ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Pyroomacoustics: A Python package for audio room simulations and array processing algorithms

Robin Scheibler[†], *Eric Bezzam*[‡], *and Ivan Dokmanić*[*]

[†]*Tokyo Metropolitan University, Japan* [‡]*École Polytechnique Fédérale de Lausanne, Switzerland* [*]*University of Illinois Urbana-Champaign, USA*

LCAV

Overview

As Python has become the go-to language for machine learning, having a package to streamline the prototyping of microphone array algorithms and for accurate room impulse response (RIR) generation is important. We present **pyroomacoustics**. Three key features include:

Object-oriented interface

- Conveniently construct different simulation scenarios in 2D/3D.

Room impulse response generator

- C implementation of the *image source model* to generate RIRs and simulate propagation between sources and microphones.

Reference algorithms

- Beamforming, direction-of-arrival (DOA), adaptive filtering, Short-Time Fourier Transform (STFT).

And it's **free**!

```
pip install pyroomacoustics
```

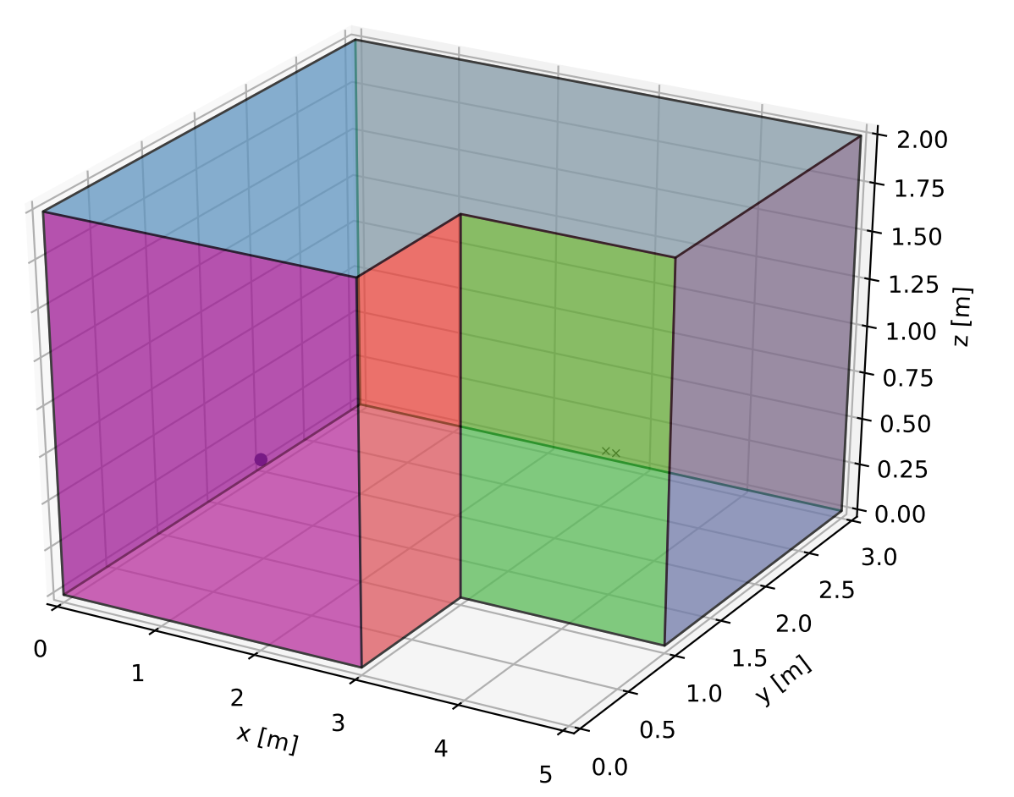
Constructing a scene

With *pyroomacoustics*, we can build a 2D or 3D room in a very intuitive fashion and conveniently add a source and microphone array.

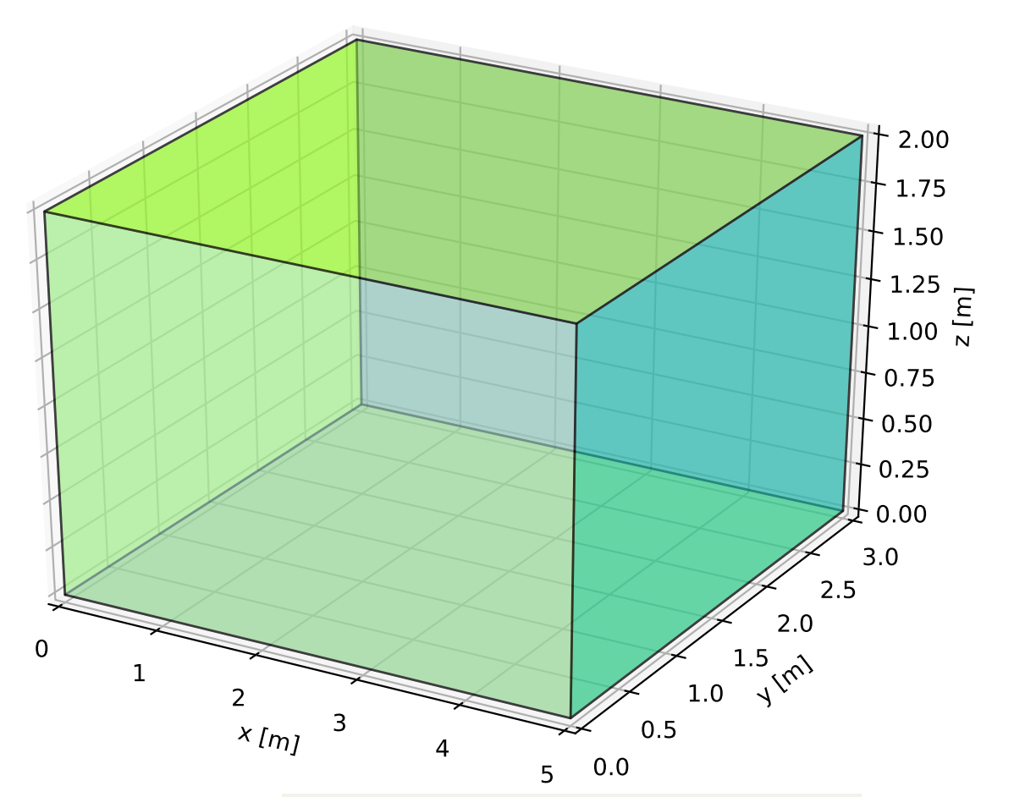
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.io import wavfile
import pyroomacoustics as pra

# specify signal source
fs, signal = wavfile.read(<FILEPATH>)

# Create a 2D L-shaped room from the specified coordinates
corners = np.array([[0,0], [0,3], [5,3], [5,1], [3,1], [3,0]]).T  # [x,y]
room = pra.Room.from_corners(corners, fs=fs, max_order=12, absorption=0.15)

# Create the 3D room by extruding the 2D shape
room.extrude(2.)

# add source and set the signal to WAV file content
room.add_source([1., 1., 0.5], signal=signal)

# add two-microphone array
R = np.array([[3.5, 3.6], [2., 2.], [0.5,  0.5]])  # [[x], [y], [z]]
room.add_microphone_array(pra.MicrophoneArray(R, room.fs))
```

Moreover, we can plot the constructed room using standard plotting libraries:

```
room.plot()
# plotting customization on current figure: ax = plt.gca()
plt.show()
```



(a) Room generated by code above.



(b) Shoebox - `room = pra.ShoeBox([x, y, z])`.

Room impulse response (RIR) generator

The RIR generator is based on the *image source model* (ISM).

For a microphone placed at $\mathbf{r}$ and a source at $\mathbf{s}_0$, we can define the set of *image* sources visible to $\mathbf{r}$ as $\mathcal{V}_{\mathbf{r}}(\mathbf{s}_0)$. The impulse response between $\mathbf{r}$ and $\mathbf{s}_0$, sampled at F_s, is thus given by:

$$a_{\mathbf{r}}(\mathbf{s}_0, n) = \sum_{\mathbf{s}\in\mathcal{V}_{\mathbf{r}}(\mathbf{s}_0)} \frac{(1-\alpha)^{\text{gen}(\mathbf{s})}}{4\pi\|\mathbf{r}-\mathbf{s}\|}\,\delta_{\text{LP}}\left(n - F_s\frac{\|\mathbf{r}-\mathbf{s}\|}{c}\right), \text{ where } \delta_{\text{LP}}(t) = \begin{cases} \frac{1}{2}\left(1+\cos\left(\frac{2\pi t}{T_w}\right)\right)\text{sinc}(t) & \text{if } -\frac{T_w}{2} \le t \le \frac{T_w}{2}, \\ 0 & \text{otherwise.}\end{cases}$$

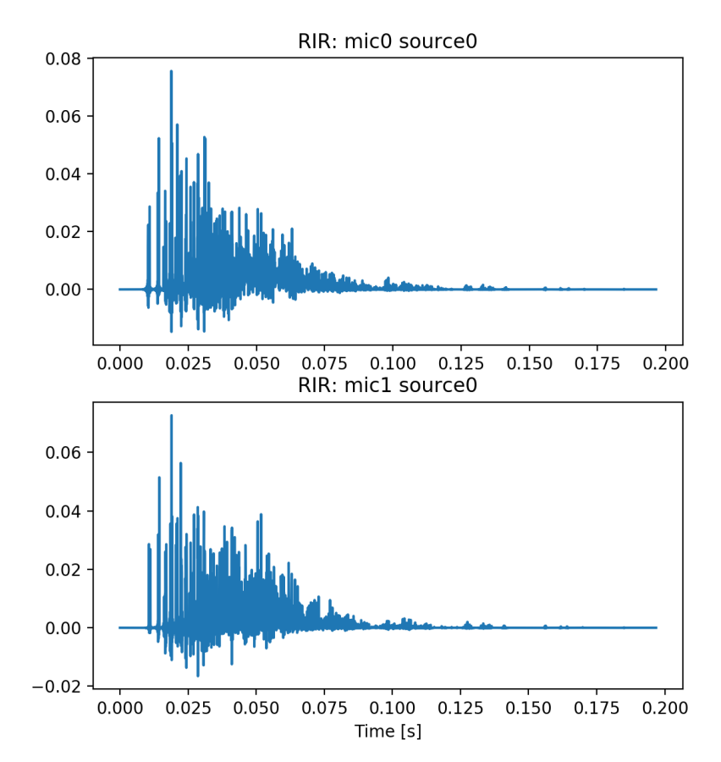
Notation: $\text{gen}(\mathbf{s})$ denotes the reflection order of source $\mathbf{s}$, $\alpha \in [0,1]$ is the absorption factor of the walls, c is the speed of sound, and δ_{LP} is the windowed sinc function where T_w sets the window's width.

The sampling frequency, maximum ISM order and absorption factor can be set when constructing the room.

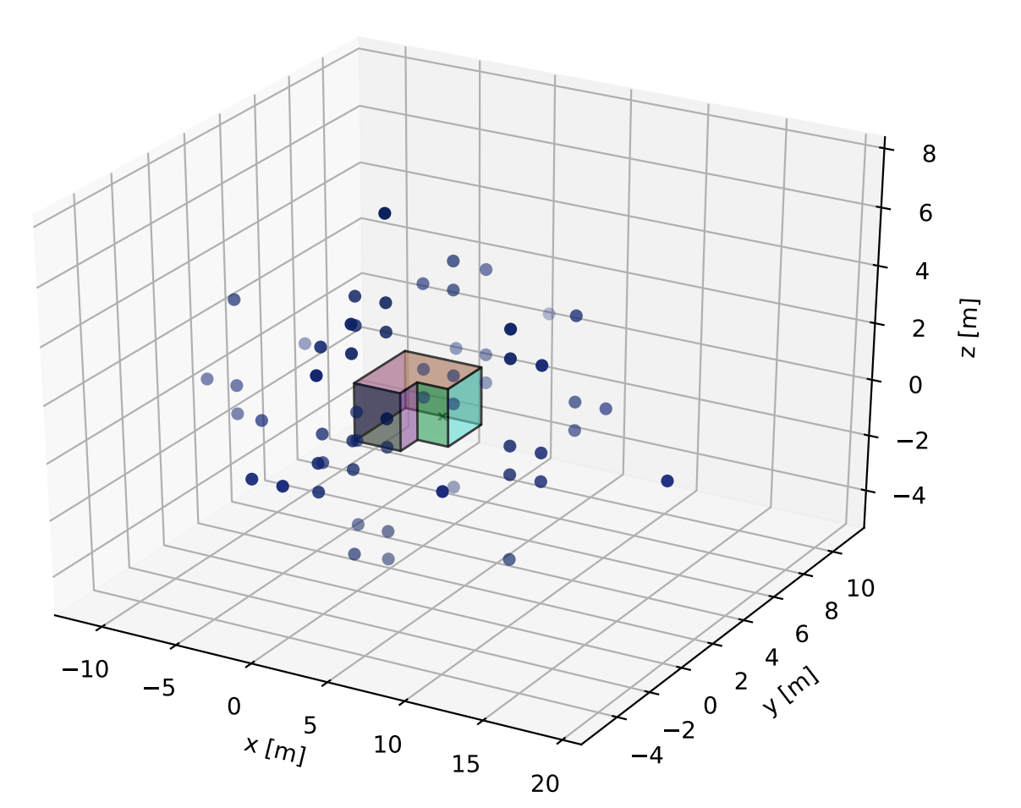
```
room = pra.Room.from_corners(corners, fs=fs, max_order=12, absorption=0.15)
```

The RIRs can be computed and plotted with the following commands:

```
room.compute_rir(); room.plot_rir()
```



(a) Generated room impulse responses for the room specified in "Constructing a scene".



(b) Vizualize the first K image sources after performing `room.compute_rir()` or `room.image_source_model()` by running `room.plot(img_order=K)`.

To simulate the recording of the source with the specified microphone array:

```
room.simulate()  # resulting recordings in 'room.mic_array.signals'
```

Most available software can only compute the RIR for *shoebox* rooms. With *pyroomacoustics*, it is possible to compute the RIR for arbitrary *polyhedral* rooms!

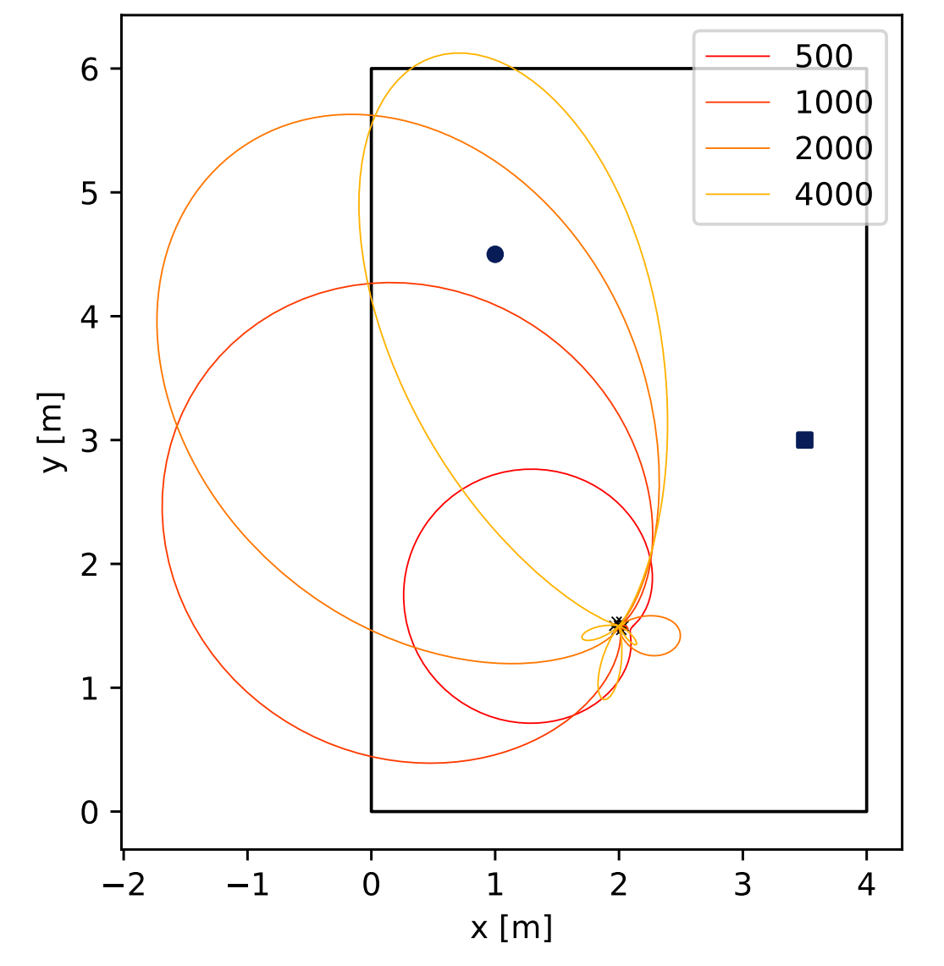
Beamforming (time and frequency)

Classic beamforming algorithms (DAS and MVDR) are included as special cases of the *acoustic rake receivers*.

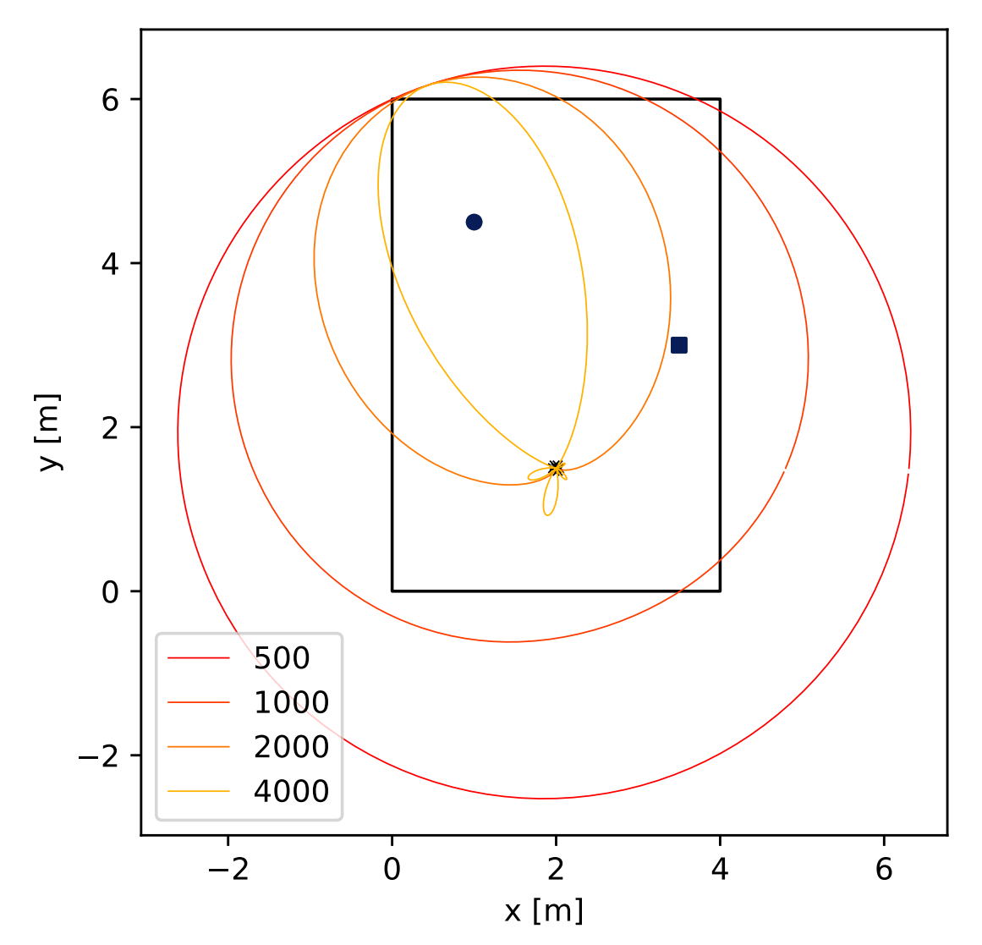
```
# Create array
center = [2, 1.5]; radius = 37.5e-3
fft_len = 512
array = pra.circular_2D_array(center=center, M=6, phi0=0, radius=radius)
array = np.concatenate((array, np.array(center, ndmin=2).T), axis=1)
room.add_microphone_array(pra.Beamformer(array, room.fs, N=fft_len))

# Compute and plot weights for the beamformer
mic_noise = 30  # db SPL
R_n = 10**((mic_noise-94)/20) * np.eye(fft_len*room.mic_array.M)
room.mic_array.rake_mvdr_filters(room.sources[0][:1], interferer=room.sources[1][:1], R_n=R_n)
room.plot(freq=[500, 1000, 2000, 4000], img_order=0)
```

Shoebox room (4x6) with one source (circle) and one interferer (square) and a 6 mic circular (37.5 mm radius) array with a center mic.

The following command can be used to beamform the simulated microphone signals: `room.mic_array.process(FD=False)`



(a) Rake MVDR with interferer.



(b) `rake_delay_and_sum_weights(source)`

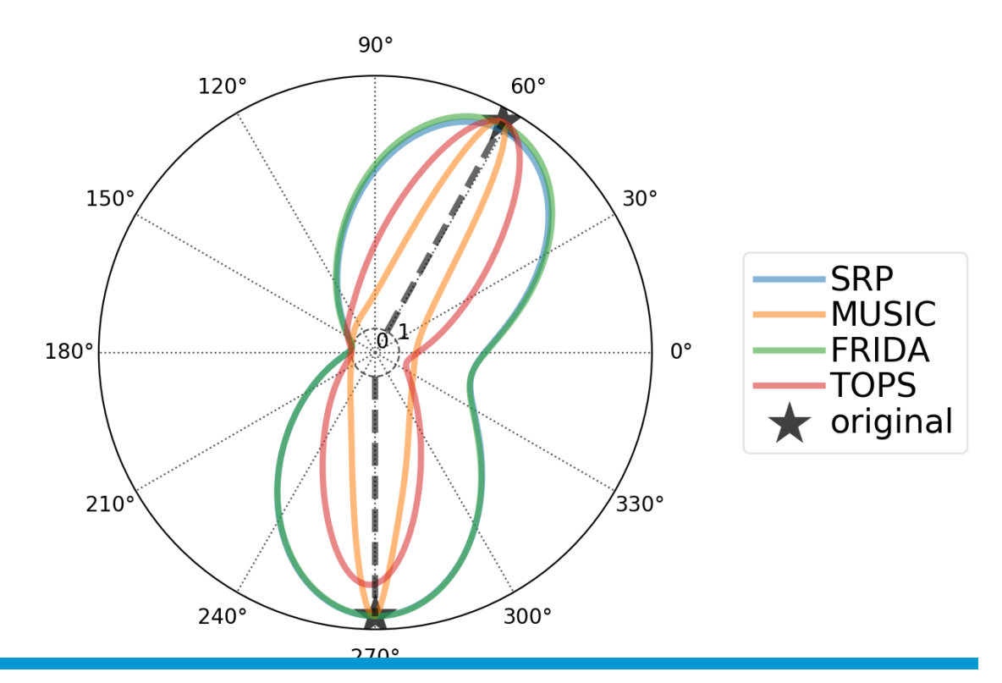
Direction-of-arrival (DOA)

DOA syntax where `<ALGO>` can be one of [SRP, MUSIC, CSSM, WAVES, FRIDA].

```
doa = pra.doa.<ALGO>(mic_array, fs, nfft, c, num_src, dim=<2,3>, mode=<'far','near'>)
doa.locate_sources(STFT, freq_range)
doa.polar_plt_dirac()
```

Spatial spectrum stored in `doa.grid.values` and estimated directions in `doa.azimuth_recon` (and `doa.colatitude_recon` for 3D.)

(Right) far-field DOA with the same array geometry as in "Beamforming". Signal consists of two 1-second long white noise sources at 61° and at 270° with an SNR of 0 dB. DOA is performed within the frequency range of [300, 3500] Hz, i.e. range of human speech.



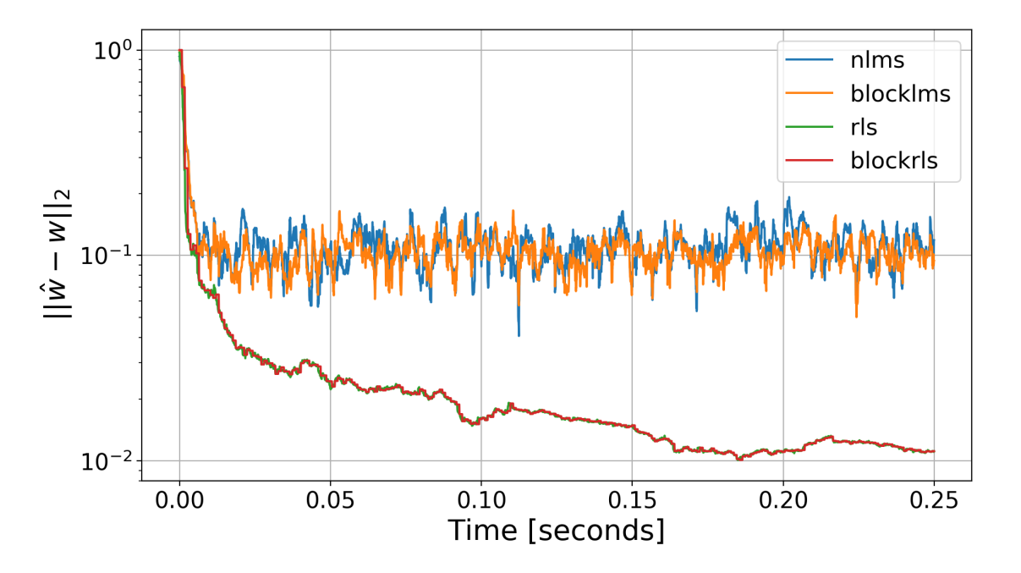
Adaptive filtering

Adaptive filt. syntax where `<ALGO>` can be one of [NLMS, BlockLMS, RLS, BlockRLS].

```
# create a known driving signal (reference) and convolve with unknown filter 'w'
x = numpy.random.randn(n_samples)
d_clean = scipy.signal.fftconvolve(x, w)[:n_samples]
d = d_clean + numpy.random.randn(n_samples) * 10**(-SNR / 20.)

# apply
adap_filt = pra.adaptive.<ALGO>(length)
for i in range(n_samples):
  adap_filt.update(x[i], d[i])
```

Estimated filter $\hat{\boldsymbol{w}}$ is stored in `adap_filt.w`.

(Right) Convergence of different adaptive filtering methods for an SNR of 15 dB.



STFT engine and real-time processing

While there is an STFT module supporting overlap-add, zero-padding, and various analysis/synthesis windows:

- `pra.stft.stft(x, L, hop, transform=np.fft.fft, win=None, zp_back=0, zp_front=0)`
- `pra.stft.istft(X, L, hop, transform=np.fft.ifft, win=None, zp_back=0, zp_front=0)`

It performs the *analysis* and *synthesis* operations on the entire signal. The `pra.realtime.STFT` class is more suitable for streaming / real-time data and is applicable to multi-channel.

```
stft = pra.realtime.STFT(block_size, hop, analysis_window, channels, transform=<numpy,pyfftw,mkl>)

while(<full blocks available>):
  stft.analysis(input_audio)
  stft.process()  # option to apply filter in frequency domain
  processed_audio = stft.synthesis()
```

Related publications

J. B. Allen and D. A. Berkley, *Image method for efficiently simulating small-room acoustics*, 1979.
I. Dokmanić, R. Scheibler, M. Vetterli, *Raking the Cocktail Party*, 2015.
R. Scheibler, M. Vetterli, *The Recursive Hessian Sketch for Adaptive Filtering*, 2016.
H. Pan et al., *FRIDA: FRI-based DOA Estimation with Arbitrary Array Layout*, 2017.

Check the paper for references to all algorithms implemented in *pyroomacoustics*!

May the Fork be with you!

New features on the horizon! If you would like to make a contribution, feel free to make a pull request by navigating to the link below ☺

`https://github.com/LCAV/pyroomacoustics`