PROCEEDINGS of the
23rd International Congress on Acoustics
9 to 13 September 2019 in Aachen, Germany

PyTTa: Open source toolbox for acoustic measurement and signal processing

William D'Andrea FONSECA[(1)], João Vitor PAES[(1)], Matheus LAZARIN[(1)],
Marcos Vinicius REIS[(1)], Paulo Henrique MAREZE[(1)], Eric BRANDÃO[(1)]

[(1)]Federal University of Santa Maria, Acoustical Engineering, Santa Maria, RS, Brazil,
{will.fonseca, joao.paes, matheus.lazarin, marcos.reis, paulo.mareze, eric.brandao}@eac.ufsm.br

Abstract

Python is a high-level programming language that has gained strength in the international community. This mainly occurs due to its open-source nature, that is, any user is entitled to its use, whether for scientific, commercial or hobby purposes. Another important feature is that Python is OS-independent. This means that no matter which OS the code was written in, it can be run on any operating system. In addition, access to AD/DA acquisition interfaces allows interaction with real systems for their estimation and control. These features enable its use as a powerful signal processing tool, whether for acoustics or general purposes. The Federal University of Santa Maria (UFSM), in Brazil, has created a group of developers who are working cooperatively on a toolbox for acoustic, audio and vibration signal processing. The project is named PyTTa, standing for Python in Technical Acoustics. It is object-oriented programmed, enabling the easy use of variables, classes, and functions. The collection of codes inside the toolbox creates an environment that facilitates acquisition/measurement, pre/post-processing, I/O operation and plotting/documenting. This work is ongoing and relies on collaborative effort to provide free access to information and work/research tools.

Keywords: Signal Processing, Toolbox, Computer Science, Acoustics.

1 INTRODUCTION

Aiming to create acoustics, audio, vibration and general signal processing tools, the Acoustical Engineering Department, at the Federal University of Santa Maria (UFSM), in Brazil, has fostered a group of developers who are working cooperatively on a toolbox. The project is named PyTTa, standing for **`Python in Technical Acoustics`** (since Python [1] is the chosen programming language).). PyTTa is an ongoing research project that relies on collaborative efforts to provide free access to information and work/research tools.

The collection of codes inside the toolbox creates an environment that facilitates the acquisition/measurement, pre/post-processing, I/O operations and plotting/documenting. This work presents the ideas and developments carried out during the first two years of the PyTTa project.

1.1 Motivation

Engineering pre- and/or post-processing tools typically rely on paid software. This yields the problem that enthusiasts and students may face costly situations to start studying/testing. Thus, looking forward to circumventing this circumstance and motivated by the excellent experience of using ITA Toolbox [2] (written in Matlab), students, professors and coders have started the collaborative work towards freeware and open source tools.

2 PYTHON

Python[1] is an already established high-level programming language. It was published during the 1990s by Guido van Rossum [4]. He released the first public version of the code in the forum *alt.sources* (this version

[1]Excerpt fom General Python FAQ [3]: "*When he began implementing Python, Guido van Rossum was also reading the published scripts from 'Monty Python's Flying Circus', a BBC comedy series from the 1970s. Van Rossum thought he needed a name that was short, unique, and slightly mysterious, so he decided to call the language Python*".

was known as 0.9.0). Version 1.0 was released in 1994 and already in 1995 (version 1.4) [5] the language won native support for complex numbers. In version 2.2, Python types and classes were unified under only one hierarchy. This feature has enabled the Python object model to be consistently object-oriented. For comparison, Python 3 has new syntax and keywords; fresh modules in the standard library; reliability improvements and dozens of fixes that made it easier to develop high-quality software. Version 3.73 is the most recent, until the closing of this paper.

Python is a free and open source language. That is, any user may use it for scientific, commercial or hobby applications. It is classified as a *scripting language*[2], i.e. it does not require the compilation step and is rather interpreted. Another important point is that Python is *platform independent*. This means that whatever operating system (OS) on which the code was written, it will probably run smoothly.

The language is commonly used for web applications, game development and database interface. Recently, it has expanded to include scientific and statistical analysis. This has occurred due to a collaborative community that develops and delivers new modules for these purposes. These modules are libraries with implemented functions, which add features/resources to the user/programmer. Some notable ones are NumPy [6] (for data manipulation of homogeneous vectors); Pandas [7] (for data manipulation of heterogeneous vectors), SciPy [8] (for daily scientific applications), and Matplotlib [9] (for quality plots), among others.

Developed to be an easy-to-read language, the indentation is mandatory. Given that aspects and presenting an easy syntax, it is considered as an easy learning language. It is used by universities, private companies and computer coders. Furthermore, Python may be an excellent choice in order to learn about programming logic [10].

2.1 Python Libraries

A huge collaborative community has arisen around the Python language. Developers and coders (who share their creations) help to fix and optimize routines for other peers. This community is mostly connected by two portals: Stack Overflow [11] and GitHub [12].

The community is also responsible for implementing a variety of Python tools. By being an object-oriented programming language, it is quite accessible to create functions. The source codes of the libraries may be found on the GitHub platform. According to the Wikipedia article [13]: "*GitHub is a source code hosting platform with version control using Git*[3]". It allows programmers or any user in the platform to contribute to private and/or open source projects.

Libraries commonly used in *signal processing*[4] are SciPy, NumPy, Matplotlib and SoundDevice/PortAudio [15]. To use a library, at first it must be imported. This is achieved by using the native Python command **`import`**. When accomplishing this step, it is possible to assign any name someone would like to that library, as can be observed in the example of Code 1.

The **`scipy`** library, according to its own site, "*is a Python-based ecosystem of open-source software for mathematics, science, and engineering*" [8]. Some of the packages contained in the **`scipy`** library are **`numpy`**, **`matplotlib`** and the **`signal`** module (used to create signals). The **`sounddevice`** [15] library provides links to the PortAudio [16] library and some convenient functions for playing and registering **`numpy`** arrays containing audio signals.

Code 1: Example of Python programming.

```
>>> import numpy as np
>>> import scipy.signal as sig
>>> import sounddevice as sd
>>> x = sig.chirp(t, F[0], T, F[1], 'logarithmic', phi=90)
>>> y = sd.playrec(xt, fs, input_mapping=[1], output_mapping=[1])
```

2.2 Object-Oriented Programming

As a scripted language, Python provides several sets of built-in functionalities that allow the developer to use a paradigm that fits better on the code crafting. Since its birth, PyTTa has intended to provide a class that holds all sorts of data about audio signals, thus it is written using the Object-Oriented Programming (OOP) paradigm.

The main advantages of this choice are the modularity and integration capabilities, making the code (once written) available as a core (or a base) for future applications. The fundamentals of OOP are that everything is an object, and each object is based on a class. Every information about the object is a class attribute and every action it performs (or can be performed with) is a class method.

[2]For the record, all scripting languages are programming languages. For example, Matlab is also a scripting language.
[3]Git is a distributed version control system.
[4]For example, Impulse Response (IR) measurement [14] or Frequency Response Function (FRF), see Section 4.1.

The attributes in a class can be other classes, indeed Python sees everything, even the numerical types as a class instance. This is called a *composition* and it is the preferable way of combining different classes and their attributes and methods. Another way of sharing them is inheriting a class, which consists of creating another class, often called a subclass, that holds its superclass attributes and methods but can also have its own.

Inheritance brings to light another feature of the OOP, the polymorphism. Any class can have the same attributes or methods, and they all can behave differently from class to class. A subclass can reimplement any method provided by the superclass and can change the values of the attributes without affecting the superclass. Finally, there is the encapsulation, which is to hide the actual data and provide only methods that either show or change the data – in a way to check if the new data suits the role that attributes have within the class or the code.

3 PyTTa FEATURES

By now, PyTTa has several features and can be seen as a core package for measurement applications. It provides ways to generate and handle audio signals for playback, recording, visualization and processing.

The codebase is separated into modules, python scripts, that were made as the code grew in lines. Thus they are not perfectly designed to have information interchangeability, but everything created to date works. There are four modules that compose the measurement core, and the fifth is the room acoustics application (under development). They can be observed in Figure 1 and are presented in the following sections.

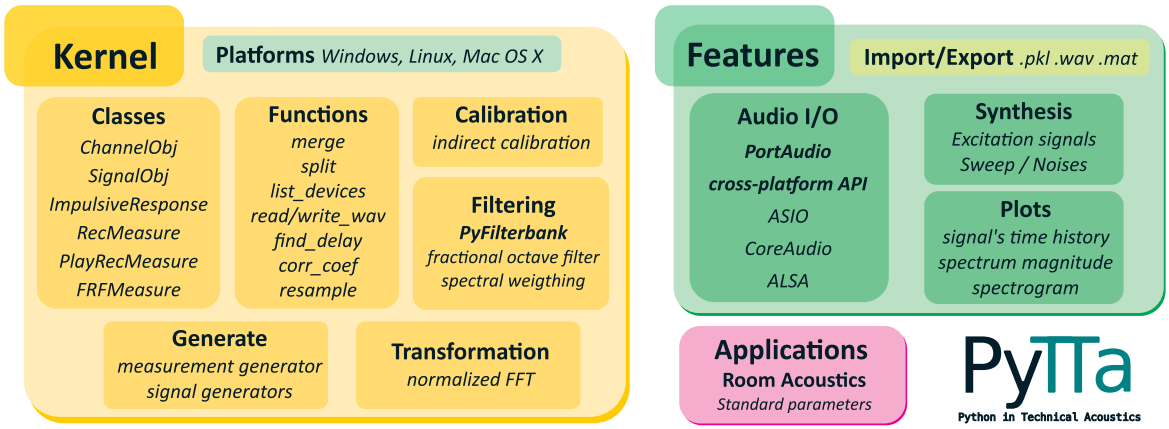


Figure 1. PyTTa Toolbox diagram showing modules and features.

3.1 Classes

Classes is the first module created. All the PyTTa classes are written in this file. They are:

- `SignalObj`;
- `ChannelObj`;
- `ImpulsiveResponse`;
- `RecMeasure`;
- `PlayRecMeasure`;
- `FRFMeasure`.

The `SignalObj` class is the main audio handler. It holds data on the discrete time amplitudes; discrete magnitudes and phases in the frequency domain; a list of active channels for audio acquisition (or reproduction) in the form of `ChannelObj` (which holds information about each channel's identification, calibration factor and units). The `ImpulsiveResponse` is a container class that holds three `SignalObj`, one for the excitation audio, one for the recorded audios and the third for the processed impulsive responses (IRs).

On the measurement side, the classes' roles are identified by their names. The `RecMeasure` class is used for record-only purposes. Both the `PlayRecMeasure` and the `FRFMeasure` are used for playback and record. The main difference is that the `FRFMeasure` is being developed to return `ImpulsiveResponse` objects, assuming that the out/in relation will be analyzed.

3.2 Properties

Every PyTTa class has several attributes that are used for measurement or signal configuration. The package comes with a set of default values that are held by the *Default class* (which is inside the properties module). These parameters are passed down to every function that needs some sort of PyTTa-like object parameters. They range from sampling rate to the number of channels, to commentaries about the audios or measurements.

3.3 Generate

This module provides user-end functions that are fully default parameterized to return pre-configured signals and measurements (setup to run out of the box). There are two principle types of functions inside the module: a group of signal generators and a unique measurement generator. The signal generators are named after the signals they generate (e.g., sweep, impulse, noise). The measurement generator is named *measurement* and a parameter is passed down to choose which of the measure classes will be used.

3.4 Functions

The functions module has several utility functions like merging N signal objects (`SignalObj`) of M channels into one (containing the total number of channels); estimating the delay between two signals; finding the signal's peak time from each channel; listing available audio I/O[5] devices; export and import audio files; resampling; correlation and convolution.

3.5 Rooms

This is the latest addition and is still under development. The module is being designed to provide a bundle of room acoustic parameters like reverberation time, center time, clarity, definition, interaural cross-correlation, speech transmission index and many others. This module will also be the first application of the package.

4 Application in Acoustics, Audio, Vibration and Signal Processing

As expected for an open-source toolbox, PyTTa may have in the future new collaborators who will introduce features based upon their needs. The basic measurement types implemented up to this point open possibilities for more specific data acquisition. From the fundamental task of measuring Frequency Response Functions (FRFs) some applications can be developed, such as sound reinforcement systems or headphones evaluation in the audio field; the response of vibrating structures (if accelerometers are employed) and/or parameters for room acoustics.

4.1 Script Example: Frequency Response Function (FRF)

An example of a simple FRF measurement is shown in Code 2. First the data is acquired by the `FRFMeasure` class run method, then it returns an `ImpulsiveResponse` class object with the calculated impulse response (IR). Time and frequency domain plots can be easily extracted from the object, per Figure 2.

Code 2: Example of a PyTTa FRF measurement.

```
import pytta
fs = 44100
sweep = pytta.generate.sweep(freqMin=20,freqMax=20000,samplingRate=fs,
                             fftDegree=18,startMargin=0.15,stopMargin=0.8,
                             method='logarithmic',windowing='hann')
ms = pytta.generate.measurement(kind='frf',excitation=sweep,samplingRate=fs,
                                freqMin=20,freqMax=20000,device=[0,1],
                                inChannel=[1,2],outChannel=[2,3],
                                comment='Example FRF measurement')
m1 = ms.run()
m1.plot_freq(smooth=True)
```

[5] I/O: input and output.

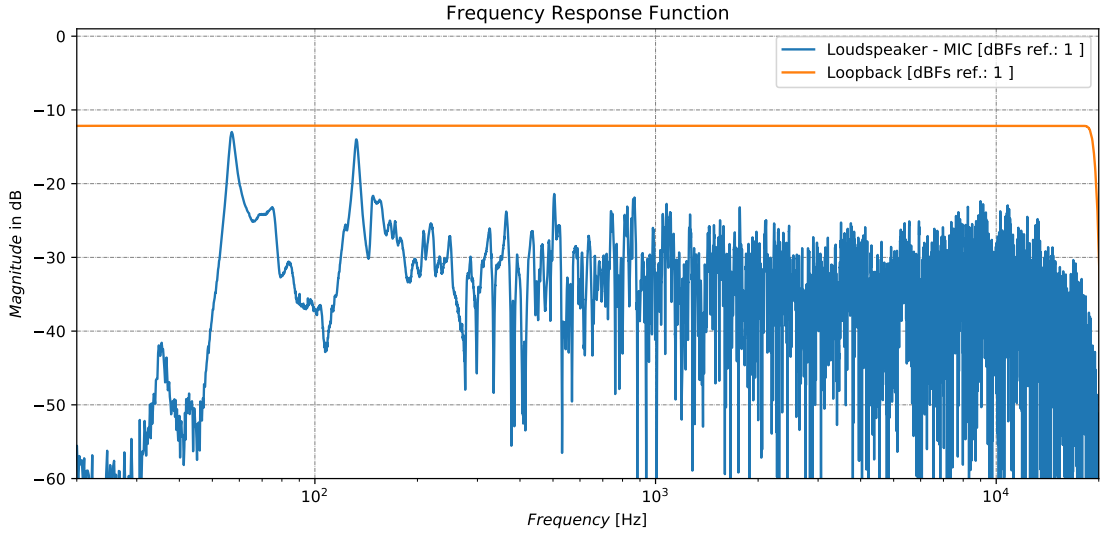
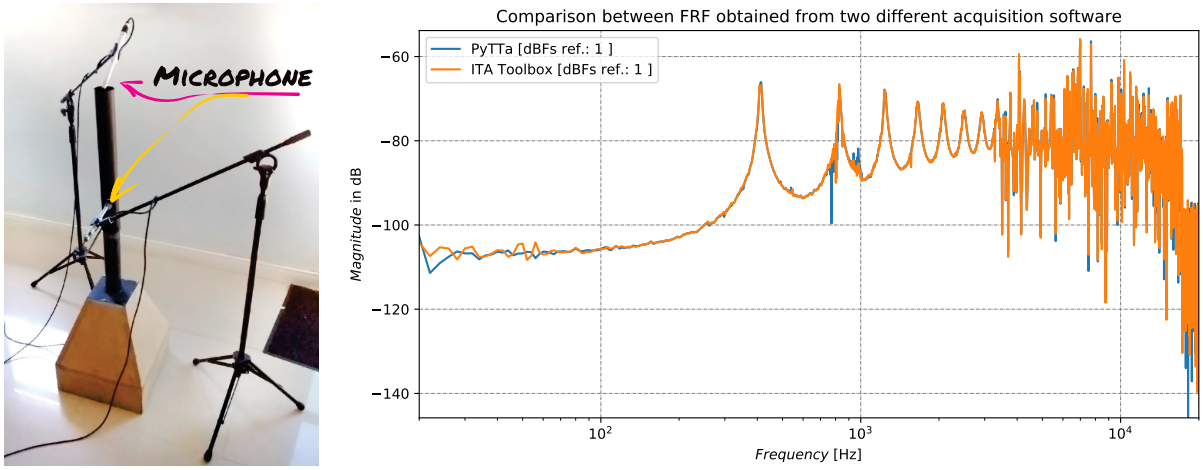


Figure 2. Frequency Response Function (FRF) between a loudspeaker and a microphone, including a loopback between input and output.

4.2 Acquisition and post-processing comparison

To validate the core features, some tests were carried out. One of the most recent results, intended to validate the calculation of transfer-functions by statistical methods, is presented in the Figure 3.



(a) Test bench

(b) FRF: `PyTTa` vs. `ITA Toolbox`

Figure 3. Testing audio acquisition and processing using `PyTTa` vs. `ITA` on muffler insertion loss test bench.

The test was performed using a test bench for insertion loss. The tested object is a reference tube approximately 42 cm long. The microphone that the yellow arrow points to (see Figure 3 (a)) is placed into an opening at the beginning of the tube (closer to the sound source and linked to the first channel of the audio I/O device). The microphone that the pink arrow points to (see Figure 3 (a)) is aiming to the center point of the tube outlet (and is linked to the second channel of the audio I/O device).

The evaluation of the tube transfer-function is the ratio between microphone at channel 2 over microphone at channel 1. Two data sets were acquired, one using `ITA Toolbox` via Matlab, and another using `PyTTa` via Anaconda Python. The data post- processing was accomplished with Matlab and Python for both acquisitions. Figure 3 (b) depicts only the comparison between the ITA Toolbox measurement and the PyTTa measurement, both processed, in this case[6], by PyTTa **`ImpulsiveResponse`** class, using the H1 statistical method – as proposed by Shin and Hammond [17]. One notices that the differences are quite small, even considering a statistical method and for different measurements.

[6]The data measured by ITA Toolbox was imported to PyTTa.

5 STARTING TO USE PyTTa

To begin, the authors recommend installing Anaconda [18], a Python distribution with important packages included such as NumPy, SciPy, and Matplotlib. It also includes other Python applications, like the Spyder IDE [19], per Figure 4. The PyTTa dependencies must be accordingly installed. They are NumPy, SciPy, Matplotlib, Sounddevice and Pyfilterbank.

The toolbox source code can be cloned from GitHub. For this task, the repository path may be included in the `PYTHONPATH` environment variable. Alternatively, it can be installed via **`pip`** [20] – a package installer for Python. To install from GitHub Code 3 must be followed.

Code 3: Installing PyTTa from GitHub with **`pip`** package.

```
pip install git+https://www.github.com/pyttamaster/pytta@master
```

To begin, one can try:

```
import pytta
pytta.list_devices()
pytta.default()
```

This set of commands will print the available audio I/O devices (such as sound cards) and the default parameters (as they are in the **`default`** class object).

To read everything present inside the package, assuming the use of Spyder IDE (see Figure 4), the user can press `ctrl+i` with the cursor in front of the module, submodule, class, methods or function names (see Code 4 below). This action will open the help menu with the documentation for the respective item.

Code 4: PyTTA installed package.

```
pytta|
pytta.properties|
pytta.generate|
pytta.functions|
pytta.classes|
```

The " | " sign represents the cursor position to press `ctrl+i` in order to use the Spyder help widget. Inside each submodule the user will find instructions on the available tools, and how to access them.

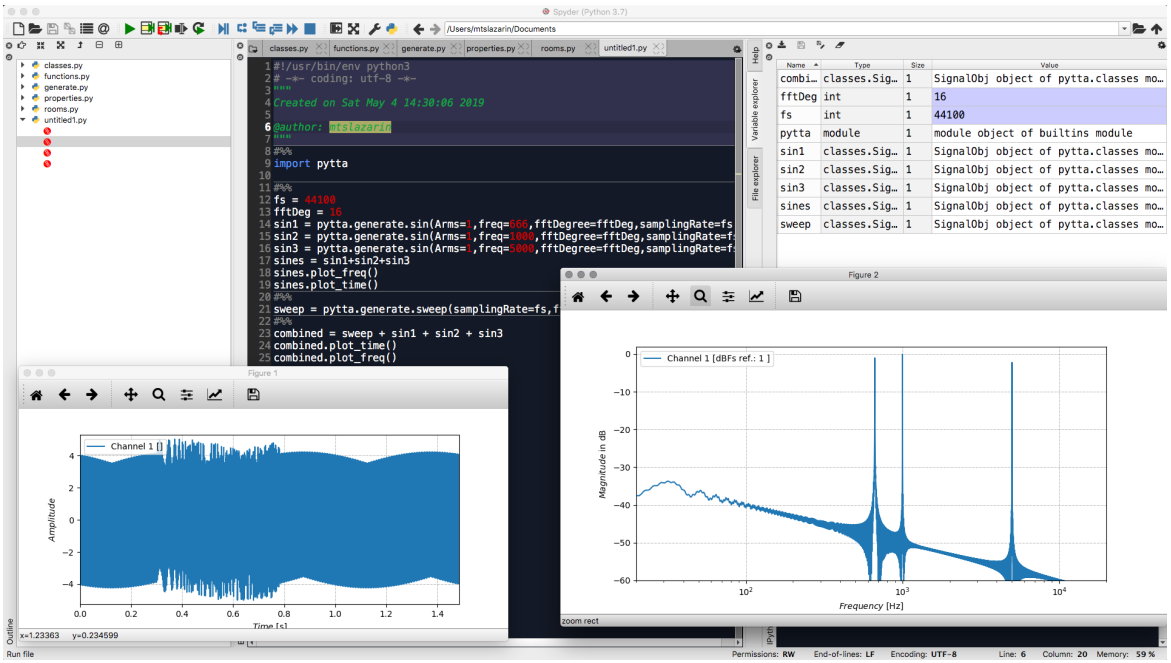


Figure 4. PyTTa usage example into Spyder IDE.

6 OPEN SCIENCE

PyTTA is being developed under an open-source concept and in a collaborative manner. The goal is to encourage people to get in touch with science by welcoming users and developers. As a counterpart, PyTTA offers the opportunity to engage in a project developed by the scientific community, integrating students, professionals, professors and enthusiasts. As a global project, it is gathering researchers and acousticians around the world.

The tool is free of use. As the concept of open-source software states, the right to use and to distribute for any purpose is granted. It will be available as a customizable tool to attend the needs of researchers and consultants in vibration and acoustics, mainly.

Integrating the curiosity of developers and researchers with the demands of professional consultants, the platform aims to absorb diverse functionalities beyond the initial step given until now (as an acquisition and processing tool). As a Federal University product, it fulfills its pedagogical and civic duty, providing an accessible tool for the development of acoustic science.

6.1 Getting Involved

PyTTa is stored in GitHub and therefore, for everyone willing to contribute, it is important to be familiar with this platform. The user can find PyTTa Master at GitHub accessing the link [21] in order to clone the module and start to use and collaborate. Some knowledge in programming, acoustics, interest in audio, engineering and open platforms is expected. For questions and feedback, contact the project at pytta@eac.ufsm.br or via GitHub.

7 CONCLUSIONS

PyTTa is a collaborative effort and a tool under development to aid acousticians to measure, achieve results, do analyze and find solutions with a trusted, easy to run, community software capable of expansion and customization. Every class, module and application is in constant improvement to be more user and developer friendly. A long journey to walk alone, the engagement of users on helping to improve and create more tools is essential for the continuity of the work.

The next steps to be followed are the development of the room acoustics module (and application), an environmental noise application, a loudspeaker alignment application, and the means to provide an auralization application. An end-user application that can be setup and started using a graphical user interface (GUI), returning fully processed data and real-time monitoring is also in development. In addition, a full documentation covering all the end user functionalities and a reference guide for developers is projected. All this development is based upon the research group at the Federal University of Santa Maria, but also includes and seeks further community contribution and user feedback.

ACKNOWLEDGEMENTS

The authors would like to express extensive gratitude to the Federal University of Santa Maria [22]; Acoustic Engineering Department, students and professors [23]; and the Institute of Technical Acoustics of the RWTH Aachen University [24] for the countless support and inspiration. Finally, we wish to acknowledge the help provided by the Python developer community and the people who maintain websites, forums, blogs and YouTube channels with tutorials.

REFERENCES

[1] Python Programming Language. https://www.python.org, accessed March 2019.

[2] P. Dietrich, B. Masiero, M. Müller-Trapet, M. Pollow, and R. Scharrer. Matlab toolbox for the comprehension of acoustic measurement and signal processing. In *Fortschritte der Akustik – DAGA*, 2010.

[3] Python - General Python FAQ. https://docs.python.org/3/faq/general.html, accessed May 2019.

[4] John V. Guttag. *Introduction to Computation and Programming Using Python*. The MIT Press, 2 edition, 2016. ISBN 9780262525008.

[5] G. van Rossum. *Python tutorial - CWI Report CS-R9525*. 1995.

[6] NumPy. http://www.numpy.org, accessed April 2019.

[7] Pandas - Python Data Analysis Library. https://pandas.pydata.org/, accessed April 2019.

[8] SciPy. https://scipy.org, accessed April 2019.

[9] Matplotlib: Python plotting. https://matplotlib.org, accessed April 2019.

[10] edX | Introduction to Python: Fundamentals (by Microsoft). https://www.edx.org/course/introduction-to-python-fundamentals-0, accessed April 2019.

[11] Jeff Atwood and Joel Spolsky. Stack Overflow http://pt.stackoverflow.com, accessed April 2019.

[12] GitHub. http://github.com, accessed March 2019.

[13] Wikipedia - GitHub. https://pt.wikipedia.org/wiki/GitHub, accessed March 2019.

[14] Swen Müller and Paulo Massarani. Transfer-function Measurement with Sweeps (Director's Cut Including Previously Unreleased Material and Some Corrections). *Original published in Journal of the Audio Engineering Society*, 49(6):443–471, 2001. URL http://www.aes.org/e-lib/browse.cfm?elib=10189.

[15] Sounddevice - Play and Record Sound with Python. http://pypi.org/project/sounddevice, accessed May 2019.

[16] PortAudio - an Open-Source Cross-Platform Audio API. http://www.portaudio.com, accessed March 2019.

[17] Kihong Shin and Joseph Hammond. *Fundamentals of signal processing for sound and vibration engineers*. John Wiley & Sons, 2008.

[18] Anaconda Distribution. https://www.anaconda.com/distribution/, accessed April 2019.

[19] Spyder IDE. http://pythonhosted.org/spyder, accessed April 2019.

[20] PyPI - Python Package Index. https://pypi.org/project/pip/, accessed May 2019.

[21] PyTTa Master. GitHub - PyTTA https://github.com/pyttamaster/pytta, accessed May 2019.

[22] Federal University of Santa Maria. Official website https://ufsm.br, 2019. acessed May 2019.

[23] Course of Acoustical Engineering (UFSM). Official website https://eac.ufsm.br, 2019. acessed May 2019.

[24] ITA Toolbox. http://www.ita-toolbox.org and http://git.rwth-aachen.de/ita/toolbox, accessed April 2019.