Von der Messung zum Ergebnis - Reproduzierbare Wissenschaft anhand von Mikrofonarray-Datenverarbeitung

Simon Jekosch, Ennes Sarradj und Gert Herold
Technische Universität Berlin, Einsteinufer 25, 10587 Berlin, Deutschland,
Email: s.jekosch@tu-berlin.de, ennes.sarradj@tu-berlin.de und gert.herold@tu-berlin.de

Einleitung

Die Reproduzierbarkeit von Forschungsergebnissen ist ein zentrales Element der Wissenschaft. Die Abhängigkeit der Ergebnisse steigt dabei zunehmend mit der Menge der Messdaten, aber auch von den verwendeten Auswertemethoden. Eine genaue Beschreibung der benutzen Algorithmen ist daher essentiell für die Reproduzierbarkeit.

Am Beispiel einer Datenauswertung mit dem freien Softwarepaket Acoular werden Probleme und Lösungsvorschläge vorgestellt, um die Reproduzierbarkeit einer Messdatenauswertung zu gewährleisten. Acoular liefert hierfür einheitliche Auswertungsabläufe und Algorithmen für Datenauswertung von Mikrofonarrayverfahren. Diese Algorithmen sind objektorientiert aufgebaut und ermöglichen eine Erweiterung der Methoden mit minimalem Aufwand.

Die Struktur des Programms erlaubt einfaches skriptbasiertes Auswerten von Messdaten. Der Beitrag zeigt Möglichkeiten auf, wie Open Source Software gestaltet werden kann, damit die Nutzbarkeit und Nachvollziehbarkeit für Wissenschaftliches Arbeiten möglich ist.

Anforderungen an eine reproduzierbare Mikrofonarray Auswertung

Um eine reproduzierbare Messdatenauswertung zu gewährleisten, müssen sämtliche relevanten Informationen übersichtlich erfasst werden und öffentlich zur Verfügung gestellt werden.

Für die Messung müssen neben den Umgebungsparametern vor allem die Anzahl der Mikrofone sowie deren Geometrie und die Kalibrationsfaktoren der Mikrofone bekannt sein. Außerdem sollten Samplerate, Messdauer und die zeitlichen Schalldruckverläufe der Mikrofone bekannt sein.

Für die Datenverarbeitung der Messdaten sind die Parameter für die Übertragung der Signale in den Frequenzbereich zu dokumentieren. Hierzu zählen die Algorithmen zur Berechnung der Fouriertransformation, Fensterfunktionen, Größe der Signal Blocks für die FFT und deren Überlappung. Des weiteren werden auch Informationen über benutzte Filter und Mittlungen benötigt.

Für das eigentliche Berechnung der akustischen Quellverteilung werden Angaben über das benutzte Berechungsgebiet und -gitter, sowie die benutzen Auswertealgorithmen benötigt.

Datenauswertung mit Acoular

Acoular ist eine Open Source Python-Bibliothek und für alle gängigen Betriebssysteme frei verfügbar unter BSD Lizenz [1],[2]. Es bietet ein Framework für einfache Skript-basierte Auswertung von Mikrofonarray Messungen und gleichzeitiges einbinden von verfügbaren Python-Bibliotheken zur Datenverarbeitung und -darstellung.

Um rechen aufwändige Operationen zu beschleunigen, wurden einige Funktionen des Codes mit Numba [4] geschrieben, um die Rechenzeit zu minimieren. Des weiteren nutzt Acoular ein Caching System, welches wichtige Zwischenergebnisse, wie Kreuzspektralmatrizen und Pointspreadfunctions, sowie die Schalldruckverteilungen der Quellkarten abspeichert. Dies hilft unnötige wiederholte Berechnungen zu vermeiden . Da nicht immer alle Berechnungen jeder Klasse für das Endergebnis relevant sind, wurde eine 'Lazy' Evaluation implementiert. Diese sorgt dafür, dass Berechnungen erst gestartet werden, sobald ein Ergebnis vom Nutzer abgefragt wird. Außerdem arbeitet Acoular mit einer Sequenzielle Datenverarbeitung, welche die Messdaten nur Paketen verarbeiten, die die Größe des Arbeitsspeicher nicht überschreiten. dadurch ist es möglich Messdaten zu verarbeiten, die nicht in den Arbeitsspeicher des Computers passen.

Struktur und Funktionen

Als Grundstruktur von Acoular wurde eine Objektorientierte Blockbauweise gewählt. Die Abbildung 1 zeigt die Hauptklassen des Software Pakets Acoular in einem UML-Diagramm. Weitere Klassen werden je nach Funktionalität von den Hauptklassen abgeleitet und können über die gleichen Schnittstellen mit den anderen Klassen kommunizieren. Die Klasse SamplesGenerator produziert aus den Zeitsignalen der Messdaten Blöcke mit einer festen Anzahl an Samples. Diese können von den Instanzen TimeInOut und PowerSpectra weiter verarbeitet werden. TimeInOut Instanzen können diese Signalblöcke im Zeitbereich prozessieren. Neben Filterungen, quadrieren und Mittlungen kann TimeInOut die Daten an BeamformerTime weitergeben, welches Beamforming im Zeitbereich anwendet.

Die Klasse PowerSpectra dient hauptsächlich zur Berechnung der Kreuzspektralmatrix, sowie Eigenwert- und Eigenvektorzerlegung. Dafür wird das komplette Zeitsignal vom SamplesGenerator benötigt, bevor die Ergebnisse an den Frequenzbereichs Beamformer weiter gegeben können. BeamformerBase ist die Basisklasse für sämtliche Frequenzbereichs Beamforming Algorithmen.

Die Klassen Zeit- und Frequenzbereichs Beamforming benötigen zusätzliche Informationen über das Ausbreitungsmodell, die Mikrofonanordnung und das Berechnungsgitter. Die Informationen über das Ausbreitungsmodell werden von der Klasse Environment bereitgestellt. Neben einfachen akustische Umgebungen sind hier auch Ausbreitungsmodelle für verschieden Strömungen für offene und geschlossene Messstrecken implementiert. Für die Mikrofonanordnung wird die Klasse MicGeom benutzt. Über diese kann man die Positionen entweder direkt eingeben oder aus einer externen Datei einlesen. Die Berechnungsgitter werden von der Grid Instanz vorgegeben. Für die Berechnung sind zwei- und dreidimensionale Gitter implementiert.

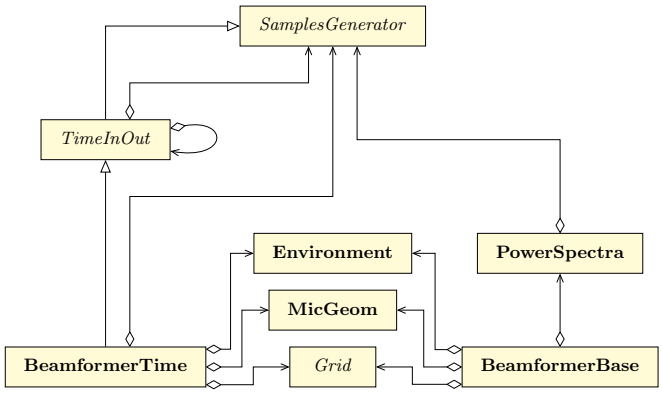

Abbildung 1: UML-Diagramm des Software Pakets Acoular.

Datenauswertung

In diesem Abschnitt wird an einem Beispiel erläutert, wie mit Acoular Python Skripte für eine Reproduzierbare Datenauswertung genutzt werden können. Die Abbildung 2 zeigt ein solches Python Skript. Zu Beginn des Skriptes wird in Zeile 1 das Acoular Modul geladen. In Zeile 3 wird ein Objekt ts aus der Klasse SamplesGenerator erzeugt, welches die Messdaten einliest. Die Messdaten 'three_sources.h5' beinhalten hierbei drei unterschiedlich stark abstrahlenden Monopol Schallquellen mit unkorreliertem weißes Rauschen. Diese sind im Datenformat HDF5 [3] abgespeichert. Die Geometrie des Mikrofonarrays wird über das Objekt mg eingelesen und von der Klasse MicGeom erzeugt. Das Array besteht in diesem Fall aus 64 Mikrofonen und hat eine Apertur von 38 cm. Die Datei ist im XML Format gespeichert und enthält die kartesischen x,y und z Koordinaten der Mikrofone. Das Objekt ps wird in Zeile 5 - 7 von der Klasse PowerSpectra erstellt. Das Objekt benötigt Informationen über das Zeitsignal ts, sowie Parameter für die FFT - in diesem Beispiel die Fensterfunktion und die Blockgröße. Anschließend wird das Berechnungsgitter definiert. Das Objekt rg wird von einem 2-D Rechteckgitter abgeleitet und enthält die Ausdehnung des Berechnungsgebiets, sowie den Abstand zum Mikrofonarray und dem Abstand zwischen den Gitterpunkten.

Der Algorithmus für das Beamforming wird in Zeile 11 definiert. Hier wird das klassische Delay-and-Sum-Beamforming im Frequenzbereich angewendet. Das Objekt bf bekommt sämtliche Informationen über die Mikrofongeometrie, Rechengitter und das PowerSpectra-Objekt übergeben. Da keine besondere Messumgebung mit der Environment Klasse definiert wurde, geht der Beamforming Algorithmus von einem ruhendem Medium aus. Die eigentliche Berechnung startet erst in Zeile 13. Die Unterklasse synthetic der Klasse BeamformerBase berechnet die Schalldruckquadrate für jeden Gitterpunkt. In diesem Fall für alle FFT Linien innerhalb der 8000 Hz Terz. Die Schalldruckquadrate werden von der L_p Funktion in Schalldruckpegel umgerechnet und dann in Lm abgespeichert. Die Abbildung 3 zeigt das Ergebnis der Berechnung als grafische Ausgabe.

```
1  import acoular
2
3  ts = acoular.TimeSamples( name = 'three_sources.h5' )
4  mg = acoular.MicGeom( from_file = 'array_64.xml' )
5  ps = acoular.PowerSpectra( time_data = ts,
6                             block_size = 128,
7                             window = 'Hanning' )
8  rg = acoular.RectGrid( x_min = -0.2, x_max = 0.2,
9                         y_min = -0.2, y_max = 0.2,
10                        z = 0.3, increment = 0.01 )
11 bf = acoular.BeamformerBase( freq_data = ps,
12                               grid = rg, mpos = mg )
13 Lm = acoular.L_p( bf.synthetic( 8000, 3 ) )
```

Abbildung 2: Beispiel zur Auswertung einer Mikrofonarray Datenauswertung mit Acoular in Python.

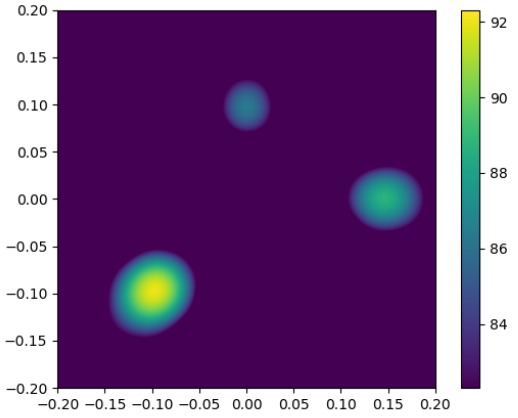

Abbildung 3: Grafische Ausgabe der Beispielauswertung zeigt drei verteilte Quellen.

Erweiterbarkeit

Als Beispiel für die Erweiterung einer Bestehenden Klasse soll hier der Beamforming Algorithmus Functional Beamforming von Dougerthy [5] implementiert werden. Der Algorithmus wird im Frequenzbereich angewendet und unterscheidet sich vom Delay-and-sum-Beamforming von einem Exponent γ. Die Schalldruckquadrate an den Gitterpunkten wird in diesem Fall nicht mehr, wie beim BeamformerBase durch

$$\mathbf{p}_m^2 = \left(h_m^H\, \mathbf{G} h_m\right) \tag{1}$$

sondern durch

$$\mathbf{p}_m^2 = \left(h_m^H \, \mathbf{G}^{1/\gamma} h_m\right)^\gamma \tag{2}$$

berechnet. Die Abbildung 4 zeigt den Quellcode für die Implementierung. Die neu erstellte Klasse BeamformerFunctional ist eine Unterklasse von der Hauptklasse BeamformerBase. BeamformerFunctional übernimmt alle Parameter aus der Hauptklasse, so dass nur der Functional Exponent γ als neuer Parameter definiert werden muss. Außerdem werden die Eigenvektoren und Eigenwerte für die Matrixwurzel Berechnung $\mathbf{G}^{1/\gamma}$ benötigt. Zusätzlich muss nur die Berechnungsfunktion calc neu implementiert werden. Dazu werden in den Zeilen 8-12 variablen definiert, die für die Berechnung notwendig sind. Die Zeilen 13 und 14 iterieren über alle zu berechnender Frequenzstützstellen. Zeile 15, 16 und berechnen und speichern jeweils die Eigenwert Wurzeln ,die Eigenvektoren und die Wellenzahl. Die Matrix-Vektor Multiplikation aus 2 wird von der mit Numba implementierten Subroutine beamfunc berechnen. Zuletzt werden die berechneten Schalldruckquadrate in Zeile 19 gespeichert.

```
1  class BeamformerFunctional( BeamformerBase ):
2
3    gamma = Float(1, desc="functional exponent")
4
5    freq_data = Trait(EigSpectra,
6                      desc="freq data object")
7
8    def calc(self, ac, fr):
9      kj = 2j*pi*self.freq_data.fftfreq()/self.c
10     numchannels = int(self.freq_data.numchannels)
11     e = zeros((numchannels), 'D')
12     h = empty((1, self.grid.size), 'd')
13     beamfunc = self.get_beamfunc('_os')
14     for i in self.freq_data.indices:
15       if not fr[i]:
16         eva = array(self.freq_data.eva[i][newaxis],
17                     dtype='float64')**(1.0/self.gamma)
18         eve = array(self.freq_data.eve[i][newaxis],
19                     dtype='complex128')
20         kji = kj[i, newaxis]
21         beamfunc(e, h, self.r0, self.rm, kji,
22                  eva, eve, 0, numchannels)
23         ac[i] = h**self.gamma
24         fr[i] = True
```

Abbildung 4: Minimal Beispiel zur Auswertung einer Mikrofonarray Datenauswertung mit Acoular in Python.

Die Abbildung 5 zeigt das Ergebnis der Berechnung als grafische Ausgabe, wenn das Skript aus 2 ausgeführt wird, jedoch als Beamforming Algorithmus anstatt BeamformerBase der neu implementierte BeamformerFuncional mit einem γ von 20 benutzt wird.

Vergleich von Algorithmen mit Benchmarking

Um verschiedene Algorithmen, aber auch gleiche Algorithmen, die von verschiedenen Instituten implementiert wurden, zu vergleichen sind Messdaten mit bekannten Lösungen nötig. Zu diesem Zweck wurde der Mikrofon Array Benchmark der AIAA erstellt [6]. Der Benchmark besteht aus insgesamt 9 Testfällen, davon 5 Tests

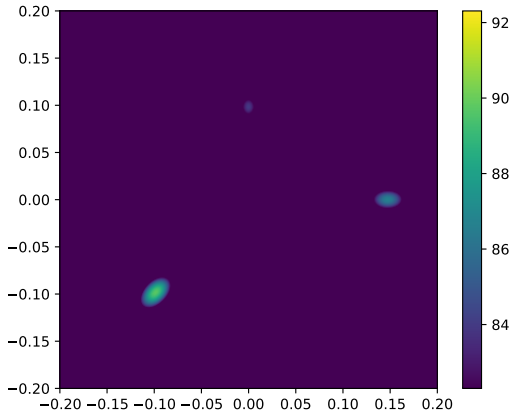

Abbildung 5: Grafische Ausgabe der Beispielauswertung zeigt drei verteilte Quellen und dem Functional Beamforming Algorithmus mit $\gamma = 20$.

mit Simulierten Daten und 4 Tests mit Messdaten. Die Tests behandeln unterschiedliche Messumgebung - Freifeld, Strömungskanäle und rotierende Felder und diverse Quellarten - Monopolquellen, Linienquellen und räumlich verteilte Quellen. Um eine Reproduzierbarkeit der Ergebnisse zu gewährleisten, stehen alle für die Berechnung nötigen Parameter, wie auch die Messdaten, zur Verfügung.

Erste Ergebnisse zu verschiedenen Benchmarks wurden bereits 2017 auf der AIAA vorgestellt [7],[8]. Die Tests zeigen deutliche Unterschiede zwischen verschiedenen Algorithmen, aber auch zwischen verschiedenen Implementationen des gleichen Algorithmus aus. Dadurch wird ersichtlich, dass selbst bei Angabe aller Messdaten und relevanten Parametern keine Reproduzierbarkeit der Auswertung gewährleistet ist, solange die Algorithmen für die Auswqertung nicht mit angegeben werden.

Zusammenfassung

Acoular bietet ein Open Source Framework für Skriptbasierte Mikrofonarray Datenauswertung in Python. Die Skripte bieten ein Übersicht über alle Berechnungsparameter, welche zur Reproduzierbarkeit der Messung beitragen. Es existiert eine Vielzahl möglicher Auswerte Algorithmen, die leicht Erweiterbar sind.

Der Mikrofon Array Benchmark der AIAA bietet ein Open Data Projekt, um Algorithmen auf Reproduzierbarkeit zu überprüfen. Außerdem können Implementationen von Algorithmen mit denen anderer Institute vergleichen werden. Unterschiedliche Ergebnisse von verschiedenen Implementationen des gleichen Algorithmus zeigen, dass die Angabe der verwendeten Implementation für die Reproduzierbarkeit eine entscheidende Rolle spielt.

Literatur

[1] Sarradj, E., Herold, G.: A Python framework for microphone array data processing,

(2017), Applied Acoustics, 116, 50–58. https://doi.org/10.1016/j.apacoust.2016.09.015

[2] Acoular – Acoustic testing and source mapping software, URL: `http://www.acoular.org`

[3] Hierarchical Data Format, version 5, The HDF Group,1997-2018, URL: `http://www.hdfgroup.org/HDF5/`

[4] Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. 2015. Numba: a LLVM-based Python JIT compiler. In Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC (LLVM '15), DOI: https://doi.org/10.1145/2833157.2833162

[5] Dougherty, R. P.: Functional beamforming. In Proceedings of the 5th Berlin Beamforming Conference 2014 (pp. 1–25)

[6] AIAA - Benchmarking Array Analysis Methods URL: `https://www.b-tu.de/fg-akustik/lehre/aktuelles/arraybenchmark`

[7] A Comparison of Microphone Phased Array Methods Applied to the Study of Airframe Noise in Wind Tunnel Testing, Christopher J. Bahr, William M. Humphreys, Daniel Ernst, Thomas Ahlefeldt, Carsten Spehr, Antonio Pereira, Quentin Leclère, Christophe Picard, Ric Porteous, Danielle Moreau, Jeoffrey R. Fischer, and Con J. Doolan 23rd AIAA/CEAS Aeroacoustics Conference, 2017, Denver, Colorado

[8] A Microphone Array Method Benchmarking Exercise using Synthesized Input Data Ennes Sarradj, Gert Herold, Pieter Sijtsma, Roberto Merino Martinez, Thomas F. Geyer, Christopher J. Bahr, Ric Porteous, Danielle Moreau, and Con J. Doolan 23rd AIAA/CEAS Aeroacoustics Conference, 2017, Denver, Colorado.