Ultrasonic measurement systems with diffraction correction for gas

Andreas Hagen

Master thesis in acoustics

Department of Physics and Technology
University of Bergen

August 31, 2017

Contents

Chapter 1

Introduction

1.1 Background and motivation

Indirect measurement of quantities like density, distance, composition, and more can be done with acoustic techniques. This means that many measurements can be done with little to no disturbance to the object measured by utilising these techniques over more direct approaches. Examples of this include flow measurement, where full or partial removal of the measurement equipment from the flow minimises the pressure drop over the measurement equipment, and medical imaging, where internal images can be captured without irradiating or performing surgery on patients.

The electroacoustic transducers used in these applications necessarily exhibit diffraction effects due to their finite extent, which depending on the application can be a significant source of error if not properly accounted for, especially for measurements over a short distance (i.e. inside the Fresnel region; $d \ll ka^2$ for distance from transducer d, wavenumber k, and transducer radius a). An example of this is the apparent rise in the propagation speed of sound in the near field of a transducer observed by Hebb [1] and Reid [2]; however this was not identified by the authors as diffraction at the time.

The earliest attempt of diffraction correction in an acoustic system known to the current author is by Huntington et al. [3], who while constructing a model for an ultrasonic delay line included diffraction effects by considering special cases of the Helmholtz-Kirchhoff equation (discussed in section 2.6) for a uniformly vibrating circular piston mounted flush in a rigid baffle of infinite extent subject to Fresnel and Fraunhofer approximations [4, Chapter 8, pp. 382-386]. In regions where the model proved difficult to solve they resorted to using tabular data after Lommel [5]. An exact integral form of this diffraction correction was later given by Rogers and Van Bruen [6].

Notable developments on the idealised baffled piston diffraction correction model after the above initial work is William's exact solution in all regions for a transducer pair of identical dimensions [7] based on earlier work by King [8], and tabulated by Khimunin [9; 10], and Sabin's more general solution for transducers of differing dimensions [11].

Recent work on diffraction correction include a proposal by Lunde et al. [12] to augment the far-field solution of the idealised baffled piston model with data from finite element analysis of non-uniformly vibrating transducers and their immediate vicinity.

Only partially simulating the acoustic system where the diffraction occurs as proposed might seem like a suboptimal approach as opposed to simulating the whole system and computing the diffraction correction directly, but finite element analysis at the resolutions required for the proposed purpose is a computationally expensive process; Hence it is of interest to check the validity of the partial model. Some work on this has been done at the University of Bergen [13; 14; 15; 16; 17], but due to difficulties pertaining to coherent electrical noise (crosstalk) the model has thus far evaded comparisons at shorter separation distances.

1.2 Objectives

The current work is focused on general improvements of the measurement system left after Andersen [17] to facilitate measurements of transfer functions of the acoustic system with shorter distances between the electroacoustic transducers. These transfer functions are then to be compared with transfer functions predicted by the reciprocity based transfer function model from section 2.3.4 corrected for diffraction with the proposed finite element based diffraction correction by Lunde et al. [12] to explore the viability of this diffraction correction model at shorter separation distances.

To improve the measured signal used to calculate transfer functions the grounding and shielding of different parts of the experimental setup and other electronics in its immediate vicinity is investigated. Additional shielding of some exposed wiring on the transducers are also introduced and the effects of this is explored.

A method of removing residual coherent noise by subtracting its spectrum from measured signals is also explored. The coherent noise spectrum is obtained by blocking out the acoustic part of the signal with a screen, thus measuring the electric part separately. This part of the current work should be seen as preliminary tests as material parameters of the screen filtering out the acoustic part of the signal has been given no attention and can surely be optimised.

In the current work the software COMSOL Multiphysics [18] is used to simulate piezoelectric discs immersed in air in order to calculate their electric and acoustic properties for use in the aforementioned reciprocity based transfer function model and diffraction correction model. This is a departure from previous work done at University of Bergen, which all primarily have been using FEMP [19], and as such a rudimentary comparison is performed to show that both pieces of software yield similar results for the problem at hand.

As the measurement setup used in the current work is a semi-permanent setup at the University of Bergen, being used to the authors best knowledge for at least six theses, an effort to make a coherent codebase for obtaining and post-processing measurement data is also included, but relegated to an appendix as it is not a primary focus of the current work.

1.3 Outline

For the readers convenience the following short outline will present the general structure of the present work.

Chapter 2 gives derivations and explanations of the theoretical framework the current work is based on. Full derivations of equations used are preferred over plain explanations as it is thought that this gives the reader an easier time understanding the uses of the principles discussed.

Chapter 3 explains the setup used for simulating the piezoelectric discs and post-processing done to the resulting data.

Chapter 4 explains the measurement system used and the procedures used with it to preform the measurements used in the current work. This also includes post-processing of the data to remove spurious contributions to the measured value.

Chapter 5 walks the reader through the acquired results from simulations and measurements. A comparison between the two will also be included in the end.

Chapter 6 will discuss the results from the previous chapter and argue conclusions to draw from the presented data. A section suggesting future work on the topic is also included.

Chapter 2

Theory

2.1 Governing principles

2.1.1 Waves in fluids

One of the types of media dealt with in acoustics are fluids, which generally are described by the Navier-Stokes equations, which can be formulated as

$$\rho\left(\frac{\partial\vec{v}}{\partial t}+(\vec{v}\cdot\nabla)\vec{v}\right)=-\nabla p+\left(\frac{4}{3}\eta+\eta_B\right)\nabla(\nabla\cdot\vec{v})-\eta\nabla\times\nabla\times\vec{v}\,. \tag{2.1}$$

The non-linear terms of the above equation makes it unwieldy when used for non-trivial cases and as such should be linearised. This can be accomplished by assuming that the velocity field is not turbulent, which yields $\eta\nabla\times\nabla\times\vec{v}\approx 0$, and that each differential volume of fluid isn't over time affected much by travelling through the velocity field, either because the velocity field is not a strong function of space, or because differential volumes of fluid doesn't move far from their starting location, both of which yields $(\vec{v}\cdot\nabla)\vec{v}\approx 0$.

By taking the divergence of the linearised Navier-Stokes equation and substituting in the linearised continuity equation, $\nabla\cdot\vec{v}=-\frac{\partial s}{\partial t}$, and the adiabatic relation $p=\rho_0 c^2 s$ one can derive the lossy wave equation (as shown in [20, Chapter 8, pp. 211-212])

$$\frac{1}{c^2}\frac{\partial^2 p}{\partial t^2}=\left(1+\tau_S\frac{\partial}{\partial t}\right)\nabla^2 p\,, \tag{2.2}$$

where the time constant $\tau_S=\left(\frac{4}{3}\eta+\eta_B\right)/\rho_0 c^2$.

By assuming harmonic solutions, $p=p_0 e^{i\omega t}$, the above wave equation reduces to Helmholtz equation,

$$(\nabla^2+k^2)p=0\,, \tag{2.3}$$

with $k=\frac{\omega}{c}/\sqrt{1+i\omega\tau_S}$. This equation is used to model acoustical pressure waves, i.e. waves in gasses and liquids, and while constrained to mono-frequency solutions when used alone can model all sorts of signals when combined with Fourier analysis.

2.1.2 Waves in solids

In solid media the Navier-Stokes equations used in the previous section does not apply and as such a different way of obtaining a wave equation has to be used. The Cauchy stress tensor from continuum mechanics is defined by the net propagation of forces in a body, given by $\Delta\vec{F}=(T\cdot\vec{n})\Delta A$, which implicitly yields the equation for traction

$$\vec{t}=\frac{d\vec{F}}{dA}=T\cdot\vec{n}\,. \tag{2.4}$$

Using the above equation together with the divergence theorem on a volume small enough so that $\nabla \cdot T$ can be approximated as constant over the whole volume yields

$$\rho \frac{\partial^2 \vec{u}}{\partial t^2} = \nabla \cdot T \tag{2.5}$$

when said volume is divided through. This relation is essentially just Newton's second law for the insides of solid bodies as stated in terms of stresses instead of forces. A more detailed derivation of this relation can be found in [21, Chapter 5].

Since stresses are just an analogue of forces in continuum mechanics it should come as no surprise that Hooke's law also has an analogue in the form of

$$T_H = c_E : S \,. \tag{2.6}$$

This linear representation accounts for all stress linearly induced by strains, but other sources of stresses might also contribute to the total stress, and thus this component is here subscripted by H.

It is usually easier to express deformations in terms of displacements instead of strains, and thus it is desirable to have a relation between the two. Assuming that the displacement $\vec{u}$ is given as an affine transformation of the position $\vec{x}$, meaning the solid is restricted to a translation $\vec{b}$ and a linear deformation, it's complete Taylor series is given as

$$\vec{u} = M\vec{x} + \vec{b} = (\nabla \vec{u})\, \vec{x} + \vec{b} \,. \tag{2.7}$$

Since a strain is just the change in length per length, i.e. percentage change, it is natural to assume from the above equation that $S = M = \nabla \vec{u}$.

While this works as long as no rotation is induced, it fails otherwise as $\nabla \vec{u}$ also includes these rotations, and thus rotations would be interpreted as spurious strains. As the curl $\nabla \times \vec{u}$ is zero only when M is symmetric [22, Chapter 6, pp. 210-211], S has to be defined as the symmetric part of M,

$$S = \nabla_S \vec{u} = \frac{1}{2} \left(\nabla \vec{u} + (\nabla \vec{u})^T \right) \,. \tag{2.8}$$

Combining the above relations, adding in a term T_E to account for external stresses not due to strain of the material, one can form a wave equation of the form

$$\rho \frac{\partial^2 \vec{u}}{\partial t^2} = \nabla \cdot \left(c_E : \nabla_S \vec{u} + T_E \right) \,. \tag{2.9}$$

As in the previous section the wave equation can be simplified by assuming harmonic solutions of the form $\vec{u} = \vec{u}_0 e^{i\omega t}$, yielding

$$-\omega^2 \rho \vec{u} = \nabla \cdot \left(c_E : \nabla_S \vec{u} + T_E \right) \,. \tag{2.10}$$

2.1.3 Waves in piezoelectric materials

As piezoelectric materials are generally crystalline structures the governing equation for waves will be the same as in the general solid case. In addition to the stress-strain relationship found in all solids piezoelectric materials also exhibit the piezoelectric and the reverse piezoelectric effect, which is generally characterized by the generation of voltage from forced strains to the material, and the generation of stresses from induced voltages in the material.

These effects are usually modelled by the constitutive equations for piezoelectric materials [23, Chapter 2], which adds a term involving the material strain to the electric displacement

field equations, $\vec{D}_E = e\colon S$, and a term involving the electric field to Hooke's law, $T_E = -e^T \cdot \vec{E}$, yielding

$$\begin{aligned} T &= c_E\colon S - e^T \cdot \vec{E} \\ \vec{D} &= e\colon S + \epsilon_S \cdot \vec{E} \end{aligned} . \tag{2.11}$$

As piezoelectric materials don't tend to have unbound charges in them there will be approximately no sources in the electric displacement field inside them. Thus $\nabla \cdot \vec{D} = 0$.

Assuming that the electric field is irrotational it can be expressed as the negative gradient of the electric potential in the material, $\vec{E} = -\nabla V$. Making this substitution reduces the remaining unknown variable to a scalar.

The above relations can be used together with the equation for waves in solids to give an equation for acoustic waves in piezoelectric materials together with an equilibrium equation for the corresponding electric field which have to be solved simultaneously.

$$\begin{aligned} -\omega^2 \rho \vec{u} &= \nabla \cdot \left(c_E\colon \nabla_S \vec{u} + e^T \cdot \nabla V \right) \\ 0 &= \nabla \cdot \left(e\colon \nabla_S \vec{u} - \epsilon_S \cdot \nabla V \right) \end{aligned} . \tag{2.12}$$

2.2 Fourier analysis

In nature no signal is a clean sine or cosine function, but instead has more features. A signal might have some start or stop, i.e. if what generated the signal is turned on or off; might increase or decrease in power, i.e. if the source of the signal moves closer or further away; or any number of other complications.

A useful abstraction is to think of these signals as composed of a possibly infinite number of simpler mono-frequency signals, like how music might be built up from many single instruments playing different notes.

2.2.1 Problem

The problem solved by Fourier analysis is to decompose arbitrary functions $h(\cdot)$ on the domain $[a,\, b] \in \mathbb{R}$ into sets of simple mono-frequency functions. This sort of decomposition works if the function being decomposed is sufficiently "nice", but for general functions this might not work. For this reason one often restrict the scope of the analysis to square integrable functions for which

$$\int_a^b |h(t)|^2 \, dt < \infty \,. \tag{2.13}$$

If the function $h(\cdot)$ represents a linear wave, i.e. the wave phenomena is governed by Hooke's law, then $|h(\cdot)|^2$ is proportional to the energy carried by the wave. Thus the above restriction can be seen as a restriction to waves or signals carrying a finite amount of energy. The absolute value is included in case of a complex valued functions to ensure amplitude is integrated over.

The above restriction is however slightly too strict because there exists functions that can be decomposed into mono-frequency functions but which is not square integrable. A trivial example of this is an infinite sine wave.

To dodge this problem instead define the domain of Fourier analysis to be all functions that can be composed from summing complex exponential waves Ae^{it}. Borrowing from linear algebra this is the same as saying that the domain is the abstract vector space of functions spanned by the above exponentials.

A vector space can be equipped with an inner product which adds the possibility of projecting vectors in the space onto each other. A natural definition for the inner product for the above

vector space would be a generalization of the dot product for $\mathbb{C}^n$ as complex valued functions can be seen as infinite dimensional vectors of the complex numbers they produce.

$$\langle h,\, g\rangle = \int_a^b h(t)\overline{g(t)}\,dt \tag{2.14}$$

The projection of $h(\cdot)$ onto $g(\cdot)$ is then given by

$$\mathrm{proj}_g(h) = \langle h,\, g\rangle \frac{g}{\|g\|} \tag{2.15}$$

where the norm of $g(\cdot)$ is just the square root of its inner product with itself, $\|g\| = \sqrt{\langle g,\, g\rangle}$.

2.2.2 Fourier transform

The previous section defined the domain of Fourier analysis to be the vector space of functions spanned by the complex exponentials. It is useful to find an orthogonal basis for this vector space as it enables construction and decomposition of arbitrary vectors in the space. It is also convenient if the basis is already normalized as this removes normalization terms when the basis is used. It is possible to show that

$$b_k(t) = \frac{1}{\sqrt{2l}} e^{\frac{i\pi kt}{l}},\ k \in \mathbb{Z} \tag{2.16}$$

forms such a basis for functions defined on the interval $[-l,\, l] \in \mathbb{R}$ [24, Chapter 1, p. 60]. Since projecting a vector onto a basis vector from an orthonormal basis essentially just gets one coefficient of the vector as expressed in that basis it follows that $h(\cdot)$ can be decomposed and reconstructed as

$$h(t) = \sum_{k=-\infty}^{\infty} \langle h,\, b_k\rangle b_k(t) = \sum_{k=-\infty}^{\infty} \alpha_k e^{\frac{i\pi kt}{l}}\,, \tag{2.17}$$

where

$$\alpha_k = \frac{1}{2l}\int_{-l}^{l} h(t) e^{-\frac{i\pi kt}{l}}\,dt\,. \tag{2.18}$$

By making the substitution $2f_k = {}^{k}/_{l}$, noting that $\Delta f = f_{k+1} - f_k = {}^{1}/_{2l}$ (as show in [24, Chapter 2, p. 93]),

$$h(t) = \sum_{k=-\infty}^{\infty} H_l(f_k)\Delta f = \sum_{k=-\infty}^{\infty} \left(\int_{-l}^{l} h(t) e^{-2\pi i f_k t}\,dt\right) e^{2\pi i f_k t}\Delta f\,, \tag{2.19}$$

one can easily see that as l goes to infinity the decomposition and reconstruction of $f(\cdot)$ can be separated as the Fourier transform

$$\mathcal{F}\{h(t)\}(f) = \int_{-\infty}^{\infty} f(t) e^{-2\pi i f t}\,dt\,, \tag{2.20}$$

and it's inverse

$$h(t) = \int_{-\infty}^{\infty} \mathcal{F}\{f(t)\}(f) e^{2\pi i f t}\,df\,. \tag{2.21}$$

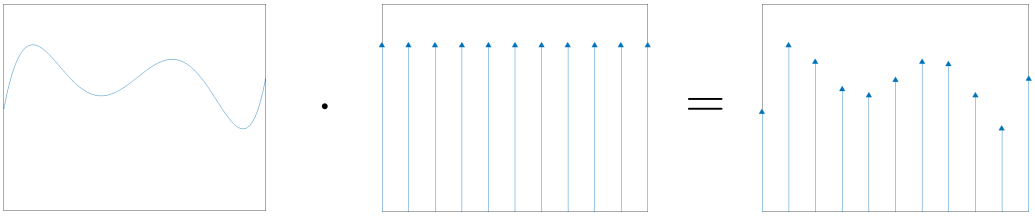


Figure 2.1: Multiplication of an example function with a dirac comb to discretize it.

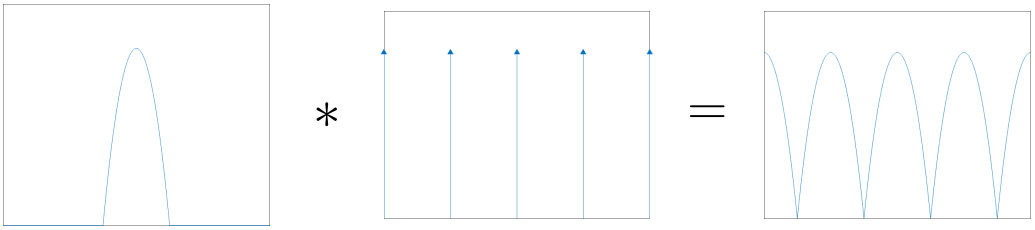


Figure 2.2: Convolution of an example function with a dirac comb to periodize it.

2.2.3 Discretizing signals

When working with the Fourier transform in a digital environment a discretized version of the transform is needed. It is desirable to discretize both the time domain and the frequency domain as this makes both signal representations fit in a finite amount of memory.

This discretization can be done by multiplying the function under analysis with the Dirac comb function

$$h(t) \cdot \text{Ш}_T(t) = \sum_{k=-\infty}^{\infty} h(kT)\delta(t - kT)\,, \tag{2.22}$$

which essentially just samples the function at time intervals T as illustrated in fig. 2.1. When evaluating expressions where the delta function is used like this it is assumed that one needs to integrate over a small interval containing the evaluation point since evaluating the delta function is not well defined.

While multiplying with a Dirac comb discretizes a function convolving with it periodizes a function by repeatedly superposing every point with all points a distance kT, $k \in \mathbb{Z}$ away.

$$h(t) * \text{Ш}_T(t) = \sum_{k=-\infty}^{\infty} h(t - kT) \tag{2.23}$$

This is shown for a function with support on an interval of length T in fig. 2.2.

Combining the fact that the Dirac comb is the Fourier transform of itself,

$$\mathcal{F}\{\text{Ш}_T(t)\}(f) = \frac{1}{T}\text{Ш}_{1/T}(f)\,, \tag{2.24}$$

with the convolution theorem,

$$\begin{aligned} \mathcal{F}\{h(t) \cdot g(t)\}(f) &= \mathcal{F}\{h(t)\}(f) * \mathcal{F}\{g(t)\}(f) \\ \mathcal{F}\{h(t) * g(t)\}(f) &= \mathcal{F}\{h(t)\}(f) \cdot \mathcal{F}\{g(t)\}(f) \end{aligned}\,, \tag{2.25}$$

one can easily see that for a discrete signal to have a discrete Fourier transform the signal has to be periodic. This relation works both ways as the convolution theorem is symmetric, and discrete signals has to have a periodic Fourier transform as well.

Because of the above restriction on signals with a discrete Fourier transform one is forced to assume that signals sampled during an interval T are of the form

$$\hat{h}(t) = h(t) = \text{Ш}_T(t) * (\Pi_T(t)h(t)) = h\left(t - \left\lfloor \frac{t}{T} \right\rfloor T\right)\,, \tag{2.26}$$

where

$$\Pi_T(t) = \begin{cases} 1 & \text{if } 0 \leq t < T \\ 0 & else \end{cases}, \tag{2.27}$$

meaning the sampled signal is a whole number of periods of a periodic signal. As such signals would have to go on forever this is not a realistic assumption and thus a discrete Fourier transform will never be better than approximately equal to the continuous Fourier transform of a natural signal.

If the signal is not periodic the forced periodicity from the above formula will cause spectral leakage, which is when the Dirac comb convolution and the rectangle function's Fourier transform causes artefacts in the computed Fourier transform that is not present in the actual Fourier transform of the signal.

2.2.4 Discrete Fourier transform

As the previous section demonstrated that a signal with a discrete Fourier transform has to be periodic, one has to assume that this is the case for a sampled signal $\hat{h}(t) = \text{Ш}_{T_s}(t) \cdot h(t)$ to derive the discrete Fourier transform. For simplicity assume that the signal is periodic with an interval T a multiple of the sampling interval T_s such that the signal repeats right after the last sample taken. For N samples taken in $t \in [0, (N-1)T_s]$ this yields the relation $T = NT_s$.

Inserting the signal $h(t)$ into the Fourier transform formula, equation (2.20), utilizing the convolution theorem yields

$$\mathcal{F}\{\hat{h}(t)\}(f) = \frac{1}{T}\text{Ш}_{1/T}(f) \int_0^{T^-} \sum_{n=-\infty}^{\infty} h(nT_s)\delta(t - nT_s)e^{-2\pi i f n T_s}\, dt\,. \tag{2.28}$$

Since $\text{Ш}_{1/T}(f)$ non-zero only when $f = {}^{k}/_{T}$, $k \in \mathbb{Z}$ one can restrict analysis to these discrete frequencies by substituting it for f.

The sum under the integral only produces non-zero values when $0 \leq n \leq N-1$ due to the sifting property of the delta function. Therefore one can restrict the sum to this range while moving it outside the integral to produce an integral that sifts out a single value for each n.

$$\mathcal{F}\{\hat{h}(t)\}\left(\frac{k}{NT_s}\right) = \frac{1}{NT_s}\sum_{n=0}^{N-1} \int_0^{T^-} h(nT_s)\delta(t - nT_s)e^{-2\pi i k \frac{n}{N}}\, dt \tag{2.29}$$

Defining $h[n] = h(nT_s)$ and evaluating the integral yields

$$\mathcal{F}\{\hat{h}(t)\}\left(\frac{k}{NT_s}\right) = \frac{1}{NT_s}\sum_{n=0}^{N-1} h[n]e^{-2\pi i k \frac{n}{N}}\,, \tag{2.30}$$

which scaled by a factor T_s, to have a scale independent definition, is the discrete Fourier transform

$$\text{DFT}\{h[n]\}[k] = T_s\mathcal{F}\{\hat{h}(t)\}\left(\frac{k}{NT_s}\right) = \frac{1}{N}\sum_{n=0}^{N-1} h[n]e^{-2\pi i k \frac{n}{N}}\,. \tag{2.31}$$

As the factor ${}^{1}/_{N}$ is frequency independent it is irrelevant when doing relative computations, and is often moved to the inverse transform to save computation time when inverting the signal is not of interest.

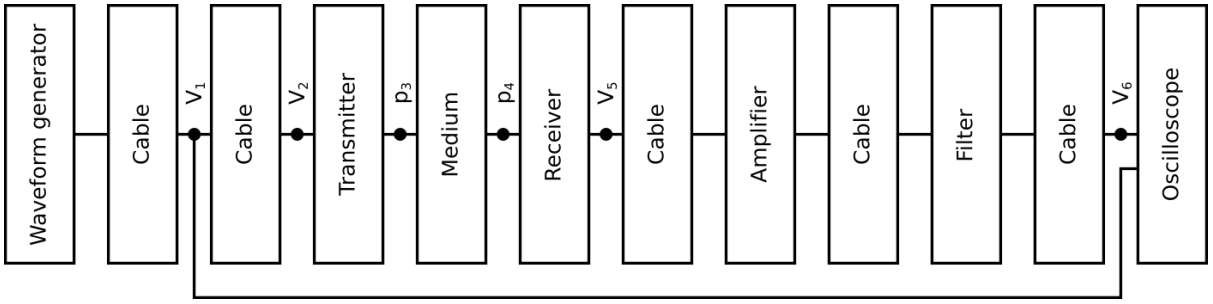


Figure 2.3: Diagram of the path taken by signals propagated through the system.

The above equation can be rewritten as a matrix equation with $w = e^{\frac{-2\pi i}{N}}$, yielding

$$\mathrm{DFT}\{h[n]\} = \frac{1}{N}\begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & w & w^2 & \cdots & w^{N-1} \\ 1 & w^2 & w^4 & \cdots & w^{2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & w^{N-1} & w^{2(N-1)} & \cdots & w^{(N-1)^2} \end{pmatrix} \begin{pmatrix} h[0] \\ h[1] \\ h[2] \\ \vdots \\ h[N-1] \end{pmatrix}. \tag{2.32}$$

Due to the simplicity of this matrix there exists several optimized algorithms to efficiently compute partial or full DFTs of sampled signals. Examples including the Cooley-Tukey algorithm, which exploits the similarity between different coefficients in the matrix to re-express the above equation as a recursive problem with smaller and smaller matrices, and the Goertzel algorithm, which re-expresses the expression for one DFT bin as an IIR filter.

2.3 System model

To be able to treat the measurement setup as individual parts whose effect can be accounted for with a single function it is assumed that the system is linear and time invariant. This is obviously just an approximation as no physical systems are linear and time invariant [25, Chapter 2, p. 33], but an approximation that is non the less quite accurate as most systems have vanishing non-linear components at small deflections from the equilibrium of the system [26, Chapter 4, p. 266].

The system at hand propagates a measurement signal in order through an oscilloscope, a cable, the transmitting transducer, the medium, the receiving transducer, another cable, an amplifier, a third cable, a digital filter, and through a fourth cable back to the oscilloscope. The signal originates from an arbitrary waveform generator, but this element will be kept out of the system model as the signal is being measured at the oscilloscope before propagating through the rest of the system. For a visual of the signal path refer to fig. 2.3.

For conformity with previous works at the University of Bergen [27] (and to a lesser extent [13; 14; 15; 16; 17]) the system parts, called blocks, will be numbered as:

1. Electronics connected between the first measurement of the signal and the transmitting transducer, meaning the port on the oscilloscope an the cable connecting the oscilloscope to the transmitting transducer. As the incoming signal from the arbitrary waveform generator is connected with a parallel t-junction to a high impedance port on the oscilloscope and the cable connecting to the transmitter, it is thought that the oscilloscope port is of minimal importance and as such only the cable is accounted for in the current work.

2. The transmitting transducer, including any electronics welded onto it, which in the current work happens to be a set of cables to move the electronic connection point away from the radiation surfaces of the transducer.

3. The medium, including any obstructions placed inside or in the vicinity of the system in such a manner that they influence the signal propagated through the medium.

4. The receiving transducer, including any electronics welded onto it. As similar transducers are used on both ends this also includes a set of cables in the current work.

5. Electronics connected between the receiving transducer and the last measurement of the signal. In the current work this includes the cable connecting the receiving transducer to the amplifier, the amplifier, the cable connecting the amplifier to the filter, the filter, and the cable connecting the filter to the oscilloscope.

6. Termination, which in the current work is just the last connection the oscilloscope.

As will be discussed in section 2.3.1 transfer functions for individual or groups of system blocks can be constructed, and will be labeled by the system block where the signal is inputted, and the system block where the output signal is inputted next, i.e. the transfer function for block number 1 is named H_{12}.

A special case is introduced in order to be able to calculate the open-circuit voltage of the receiving transducer, where the transfer function for block number 5 is split into $H_{5\text{open}5}$ and H_{56}, representing the cable connecting the receiver compensated with the internal impedance of the receiving transducer (see section 2.3.3 for more details), and the rest of block 5 respectively.

2.3.1 Transfer functions

The defining properties of linear time invariant systems are that the output of the system depends only linearly on the input to the system and this dependance dosen't change over time.

To see this property it is easier to start with the example of a causal linear time invariant system, where the current output value only depend on past input values. In such a system one can calculate the current output value as the weighted sum

$$\begin{aligned} s_2(t) &= a_0 s_1(t - 0\Delta t) + a_1 s_1(t - 1\Delta t) + a_2 s_1(t - 2\Delta t) + \dots \\ &= \sum_{n=0}^{\infty} a_n s_1(t - n\Delta t) \end{aligned} \tag{2.33}$$

for sufficiently small Δt. Letting Δt go to zero one can construct a Riemann integral from the above expression which converges to

$$s_2(t) = \int_0^{\infty} h_{12}(\tau) s_1(t - \tau)\, d\tau\,. \tag{2.34}$$

If the requirement of causality is dropped above the math works out the same way but the output signal may then also depend on future values of the input signal as well. This results in the final result instead being the convolution integral

$$s_2(t) = \int_{-\infty}^{\infty} h_{12}(\tau) s_1(t - \tau)\, d\tau = h_{12}(t) * s_1(t)\,. \tag{2.35}$$

Selecting $s_1(t)$ as a Dirac delta pulse it is evident from the sifting property of the Dirac delta function that $h_{12}(t)$ is the impulse response of the system.

If sequential system blocks are numerated such that $s_1(t)$ is the input to block number 1, $s_2(t)$ is the output from block number 1 and the input to block number 2, and $s_3(t)$ is the output of block number 2, a chain of convolutions can be established to calculate the final output;

$$s_3(t) = h_{23}(t) * (h_{12}(t) * s_1(t)) \tag{2.36}$$

There is of course no limit to how many blocks can be linked like this, and the number of blocks picked here was just picked for illustrational purposes.

Using the convolution theorem (equation (2.25)) the above relation can be shown to be equivalent to

$$\begin{aligned}\mathcal{F}\{s_3(t)\}(f) &= \mathcal{F}\{h_{23}(t)\}(f)\mathcal{F}\{h_{12}(t)\}(f)\mathcal{F}\{s_1(t)\}(f) \\ &= H_{23}(f)H_{12}(f)\mathcal{F}\{s_1(t)\}(f)\end{aligned}, \tag{2.37}$$

where the $H_{nm}(\cdot)$ functions are the transfer functions discussed earlier.

Combining the above transfer functions $H_{12}(f)$ and $H_{23}(f)$ to $H_{13}(f)$ one can right away see that transfer functions are simply the ratio between the Fourier transform of the output signal and the input signal, generally

$$H_{nm}(f) = \frac{\mathcal{F}\{s_m(t)\}(f)}{\mathcal{F}\{s_n(t)\}(f)}\,. \tag{2.38}$$

It can be easily proved by using the Fourier shift theorem combined with the identity $\cos(x) = {}^1\!/_2\left(e^{ix} + e^{-ix}\right)$ that sinusoidals have the Fourier transform

$$\mathcal{F}\{A\cos(2\pi f_0 t + \theta)\}(f) = \frac{A}{2}e^{i\theta\frac{f}{f_0}}\left(\delta(f + f_0) + \delta(f - f_0)\right). \tag{2.39}$$

Using this fact together with equation (2.38) it is evident that the magnitude and phase of the transfer function represents relative scaling and phase shift of a signal measured at different points in the system;

$$H_{nm}(f_0) = \frac{A_n}{A_m}e^{i(\theta_n - \theta_m)}\,. \tag{2.40}$$

2.3.2 Transmission line model

As several cables are used in the system a way to calculate their transfer function is necessary.

The telegrapher's equations is a set of differential equations describing cables for which lumped models are inadequate, either because high precision is required or because the cables are long compared to the wavelength of the signal. Such cables are called transmission lines.

It is assumed by the telegraphers equations that each differential element of length of a two conductor cable can be modelled by four components, as shown in fig. 2.4:

- A series resistor of value $R\Delta x$, representing simple losses in the cable.
- A series inductor of value $L\Delta x$, representing self inductance of the cable.
- A shunt resistor of value $(G\Delta x)^{-1}$, representing conductance in the dielectric separating the two conductors.
- A shunt capacitance of value $C\Delta x$, representing the capacitance between the two conductors.

Using Kirchhoff's circuit laws, letting Δx go to zero, one can from the circuit described above deduce the relations

$$\begin{aligned}\frac{\partial V}{\partial x} &= \lim_{x\to 0}\frac{\Delta V}{\Delta x} = -RI - L\frac{\partial I}{\partial t} \\ \frac{\partial I}{\partial x} &= \lim_{x\to 0}\frac{\Delta I}{\Delta x} = -GV - C\frac{\partial V}{\partial t}\end{aligned}, \tag{2.41}$$

which when substituted one into the other can be reduced to a single linear partial differential equation;

$$\frac{\partial^2 V}{\partial x^2} = LC\frac{\partial^2 V}{\partial t^2} + (CR + LG)\frac{\partial V}{\partial t} + GRV\,. \tag{2.42}$$

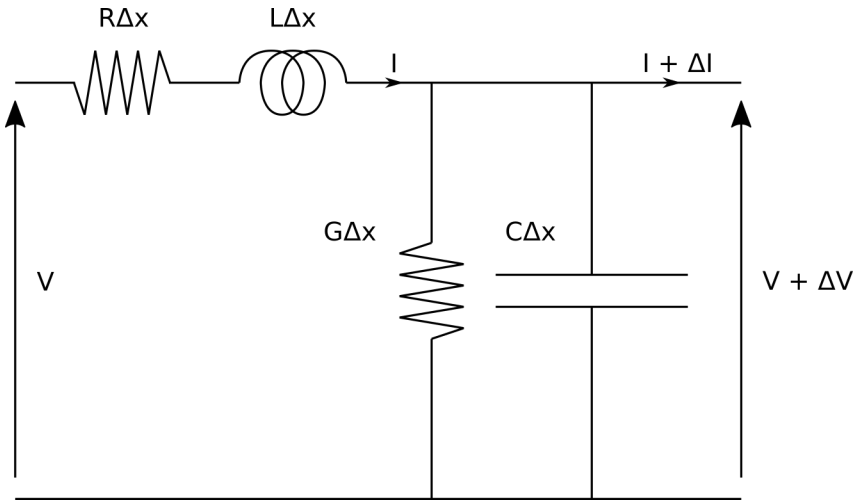


Figure 2.4: Schematic of transmission line segment of size Δx.

Assuming solutions of the form $V = V_0 e^{i\omega t}$ the above equation reduces to the one dimensional Helmholtz equation,

$$\left(\frac{\partial^2}{\partial x^2} - (R + i\omega L)(G + i\omega C)\right) V = \left(\frac{\partial^2}{\partial x^2} - ZY\right) V = 0 \tag{2.43}$$

whose solution is commonly known to be

$$V = Ae^{\sqrt{ZY}x} + Be^{-\sqrt{ZY}x}\,. \tag{2.44}$$

For shorter cables at lower frequencies the cumulative influence of resistance in the cables are negligible, so assuming $R = 0$ and $G = 0$ is reasonable. This yields $\sqrt{ZY} = i\omega\sqrt{LC} = ik$, which means that the wave propagation speed is given as $c = 1/\sqrt{LC}$.

A derivation for I can be done in similar fashion to the derivation for V above, and used in conjunction with Ohm's law, $V = IZ$, with some algebraic manipulation it can be derived that

$$\begin{aligned} V_S &= -V_R \cos(kl) + iI_R R_C \sin(kl) \\ I_S &= I_R \cos(kl) - i\frac{V_R}{R_C} \sin(kl) \end{aligned} \tag{2.45}$$

where subscript S and R represents values on the "source" and "receiver" sides of the transmission line respectively, and l and $R_C = \sqrt{L/C}$ is its length and characteristic impedance [28, Chapter 1, pp. 5-19].

Using the trigonometric identities $\sin^2(x) + \cos^2(x) = 1$ and $\cos(x) = 1 - \sin(x)\tan(x/2)$ the above equations can be transformed to the set

$$\begin{aligned} V_S &= \left(iR_C \tan\left(\frac{kl}{2}\right) + \frac{R_C}{i\sin(kl)}\right) I_S - \frac{R_C}{i\sin(kl)} I_R \\ V_R &= -\frac{R_C}{i\sin(kl)} I_S + \left(iR_C \tan\left(\frac{kl}{2}\right) + \frac{R_C}{i\sin(kl)}\right) I_R \end{aligned}, \tag{2.46}$$

which for impedances $Z_a = iR_C \tan(kl/2)$ and $Z_b = -iR_C(\sin(kl))^{-1}$ are equivalent to the equations describing the circuit in fig. 2.5 (See [29, Chapter 5] for a more in depth treatment of this set of equations).

For a known source voltage V_S and receiver load Z_R solving the linear system of equations

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ -1 & Z_a + Z_b & 0 & -Z_b \\ 0 & -Z_b & -1 & Z_a + Z_b \\ 0 & 0 & -1 & Z_R \end{pmatrix} \begin{pmatrix} V_S \\ I_S \\ V_R \\ I_R \end{pmatrix} = \begin{pmatrix} V_S \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{2.47}$$

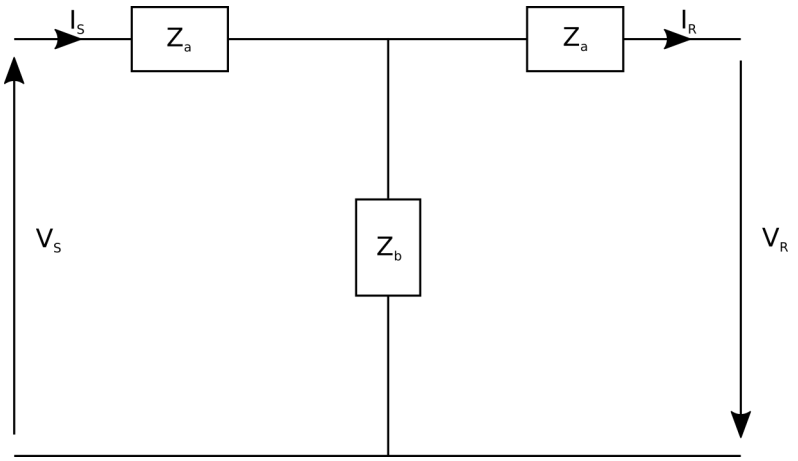


Figure 2.5: Equivalent lumped circuit for a transmission line.

yields currents and voltages on both sides of the transmission line, so that constructing transfer functions is just a matter of dividing corresponding quantities by each other, i.e

$$H^{VV}(f) = \frac{V_R(f)}{V_S(f)}\,. \tag{2.48}$$

2.3.3 Correcting for internal impedance

As mentioned earlier a special case is introduced for the transfer function of the cable connected to the receiving transducer, $H_{5\text{open}5}$, so that it's transfer function also includes the effects of the internal impedance of the receiving transducer. This means that $H_{n5\text{open}} = {}^{H_{n5}}\!/_{H_{5\text{open}5}}$ gives results as if the receiving transducer was unloaded.

By Thévenin's theorem [30, Chapter 5, pp. 77-78] a real voltage source can be represented as an ideal voltage source, V_I, in series with an internal impedance, Z_I, which means that when used in conjunction with the transmission line model

$$V_I = I_S Z_I + V_S = V_S \frac{Z_I + Z_S}{Z_S}\,, \tag{2.49}$$

where $Z_S = {}^{V_S}\!/_{I_S}$. Thus when calculating $H_{5\text{open}5}$ with the transmission line model described in the previous section an extra term must be included, e.g.

$$H^{VV}_{5\text{open}5}(f) = \frac{V_R(f)}{V_I(f)} = \frac{V_R(f)}{V_S(f)} \frac{Z_S}{Z_I + Z_S}\,. \tag{2.50}$$

2.3.4 Reciprocity based propagation model

In general electroacoustic transducers can be governed by an arbitrary set of equations. For a limited subset of these transducers, called reciprocal transducers, a useful relation can be deduced relating their transmitting current sensitivity

$$S_I = \frac{p(\vec{r}_0)}{I}\,, \tag{2.51}$$

to their receiving voltage sensitivity

$$M_V = \frac{V}{p(\vec{r}_0)}\,, \tag{2.52}$$

for some arbitrary definition of $\vec{r}_0$, and V being the open-circuit voltage generated by the transducer [31]. A common definition for $\vec{r}_0$ is 1 m from the transducer along the acoustic axis, or backpropagated by means of spherical waves to this point if it fails to be in the transducers far-field [29].

Linear passive electroacoustic transducers are those whose transduced "output" quantities are related only as a linear superposition of it's "input" quantities, i.e. it's governing equations are of the form

$$\begin{aligned} p(\vec{r}) &= \int_S z_0(\vec{r},\, \vec{s})(\vec{v}(\vec{s}) \cdot \vec{n}(\vec{s}))\, d\vec{s} + h(\vec{r})I \\ E &= \int_S h'(\vec{s})(\vec{v}(\vec{s}) \cdot \vec{n}(\vec{s}))\, d\vec{s} + Z_T I \end{aligned} \quad , \tag{2.53}$$

where S is the surface of the transducer in contact with the medium. Of these, reciprocal transducers additionally satisfy

$$\begin{aligned} \frac{h(\vec{r})}{h'(\vec{r})} &= e^{i\alpha} \\ z_0(\vec{r},\, \vec{s}) &= z_0(\vec{s},\, \vec{r}) \end{aligned} \quad , \tag{2.54}$$

which essentially means that the normal velocity at point $\vec{r}$ contributes equally as much to the pressure at point $\vec{s}$ as the normal velocity at point $\vec{s}$ contributes to the pressure at point $\vec{r}$, and that the transduction effect is reversible in a one-to-one fashion, i.e. applying pressure $p(\vec{r})$ to the transducer generating voltage V results in the same transducer state as applying the resulting voltage V to the transducer, with a possible phase shift introduced.

Foldy and Primakoff proved that for a reciprocal transducer the ratio of it's two sensitivities mentioned above when incoming waves are restricted to spherical ones are given as [32]

$$\frac{M_V}{S_I} = J_{\text{sph}} = \frac{2d\lambda}{i\rho c} e^{i(kd-\alpha)} \,. \tag{2.55}$$

Two identical transducers, Tx and Rx, located in each others far-fields, so that waves transmitted from one is approximately spherical when received by the other, on the same acoustical axis such that the transmitter's transmitting current sensitivity is measured at a distance $r_0 = |\vec{r}_0|$, and the receiver's receiving voltage sensitivity is measured in the point located distance d away from the transmitter, is thus related as

$$\frac{V^{\text{Rx}}}{V^{\text{Tx}}} = \frac{S_I M_V}{Z_T} \frac{r_0}{d} e^{-ik(d-r_0)} \,. \tag{2.56}$$

Using the reciprocity parameter J_{sph} together with the relation $S_I/Z_T = S_V$, the transmitting voltage sensitivity, the above equation becomes (as shown by [13])

$$H^{VV}_{25\text{open}}(f) C_{\text{dif}}(f) = \frac{V^{\text{Rx}}}{V^{\text{Tx}}} = S_V^2 Z_T J_{\text{sph}} \frac{r_0}{d} e^{-ik(d-r_0)} \,. \tag{2.57}$$

In the current work the voltages in the above equation corresponds with the input to block 2 and the output from block 5open when no diffraction is present except its far-field limit value (i.e. the far-field condition discussed above is met), which is indicated by the equality with the transfer function $H^{VV}_{25\text{open}}(\cdot)$ corrected with the diffraction correction $C_{\text{dif}}(\cdot)$ (to be discussed in section 2.4.1). Dividing through by $C_{\text{dif}}(\cdot)$ (as shown in [33]) yields an uncorrected transfer function.

Constant	Value	
a_0	331.5024	$\mathrm{m\,s^{-1}}$
a_1	0.603 055	$\mathrm{m\,s^{-1}\,°C^{-1}}$
a_2	-5.28×10^{-4}	$\mathrm{m\,s^{-1}\,°C^{-2}}$
a_3	51.471 935	$\mathrm{m\,s^{-1}}$
a_4	0.149 587 4	$\mathrm{m\,s^{-1}\,°C^{-1}}$
a_5	-7.82×10^{-4}	$\mathrm{m\,s^{-1}\,°C^{-2}}$
a_6	-1.82×10^{-7}	$\mathrm{m\,s^{-1}\,Pa^{-1}}$
a_7	3.73×10^{-8}	$\mathrm{m\,s^{-1}\,°C^{-1}\,Pa^{-1}}$
a_8	-2.93×10^{-10}	$\mathrm{m\,s^{-1}\,°C^{-2}\,Pa^{-1}}$
a_9	−85.209 31	$\mathrm{m\,s^{-1}}$
a_{10}	−0.228 525	$\mathrm{m\,s^{-1}\,°C^{-1}}$
a_{11}	5.91×10^{-5}	$\mathrm{m\,s^{-1}\,°C^{-2}}$
a_{12}	−2.835 149	$\mathrm{m\,s^{-1}}$
a_{13}	-2.15×10^{-13}	$\mathrm{m\,s^{-1}\,Pa^{-2}}$
a_{14}	29.179 762	$\mathrm{m\,s^{-1}}$
a_{15}	0.000 486	$\mathrm{m\,s^{-1}\,Pa^{-1}}$

Table 2.1: Fitted values for coefficients in equation (2.58).

2.3.5 Sound speed model

In the model from the previous section an unknown parameter difficult to determine with high precision is the speed of sound (introduced through the parameter k). As especially signal phase is highly dependant on the speed of sound it is important to have a good model for this, and as such several attempts have been made to construct such a model, notably by Wong [34], and Cramer [35].

For a zero-frequency signal Cramer proposed the fitted model

$$c_0 = \begin{pmatrix} 1 & x_w & p & x_c \end{pmatrix} \begin{pmatrix} a_0 & a_1 & a_2 \\ a_3 & a_4 & a_5 \\ a_6 & a_7 & a_8 \\ a_9 & a_{10} & a_{11} \end{pmatrix} \begin{pmatrix} 1 \\ t \\ t^2 \end{pmatrix} + \begin{pmatrix} x_w^2 & p^2 & x_c^2 & x_w p x_c \end{pmatrix} \begin{pmatrix} a_{12} \\ a_{13} \\ a_{14} \\ a_{15} \end{pmatrix}, \tag{2.58}$$

with accompanying coefficients given in table 2.1 for the temperature t given in degrees Celsius, the ambient pressure p given in pascals, and the concentration of water vapour and carbon dioxide x_w and x_c given as mole fractions.

As pointed out by O'Donnell et al. [36; 37] linear attenuative (absorbing or scattering) systems have to be dissipative to preserve causality (this can easily be seen by subtracting a single frequency sinusoid from a signal with finite support in time, as the result will have infinite support in either direction), which is an effect described by the Kramers-Kronig relations.

A commonly used approximation to this relation for acoustical systems (e.g. [27, Chapter 4]) was given by Morfey and Howell [38] as

$$\frac{1}{c_0} - \frac{1}{c_p} = \sum_i \frac{\alpha_i}{\omega_{ri}}, \tag{2.59}$$

where c_p is the phase speed of a given frequency, and α_i is the attenuation coefficient for the i-th relaxation mechanism resonating at ω_{ri}.

In air the most notable relaxation mechanisms are due to internal vibrations of nitrogen and oxygen molecules [27, Chapter 4] for which fitted models of the form

$$\alpha_i = C e^{-\theta_i/T} \frac{f_{ri} f^2}{f_{ri}^2 + f^2}, \tag{2.60}$$

where θ_i is the characteristic vibrational temperature of the molecule, are available in the ANSI standard on atmospheric absorption [39] with accompanying fitted models for f_{ri}.

2.3.6 Incorporating attenuation in lossless models

As explored in section 2.2 the complex exponentials $\mathbb{C} \to \mathbb{C}$ forms a basis for sufficiently "nice" functions $f\colon \mathbb{C} \to \mathbb{C}$. Without proof it will be stated that this also applies to $f\colon \mathbb{C}^n \to \mathbb{C}$, which is spanned by the complex exponentials

$$Ae^{-i(\vec{k}\cdot\vec{x})}\,. \tag{2.61}$$

This can easily be seen by taking the previously defined Fourier transform in each dimension separately as each dimension is linearly independent.

Interpreting $\vec{x} \in \mathbb{R}^n$ as a spatial position and defining

$$\vec{k} = \vec{k}_0 - i\vec{\alpha} \tag{2.62}$$

one can see that the above form will have the kind of exponential decay in its spatial dimension x_i as an attenuation of α_i neper per unit distance exhibits. Thus retrofitting a lossless model with a complex $\vec{k}$ introduces a parameter with which to tune the exponential decay produced by attenuation mechanisms.

This observation is substantiated by the form of the wave equation given in section 2.1.1 where $\vec{k}$ is directionless and reduces to a scalar.

In the current work this technique is used with the propagation model from section 2.3.4 along with the attenuation coefficients for classical (thermoviscous) and rotational absorption [40; 27, Chapter 4]

$$\alpha_{cl} = \frac{\omega^2}{2\rho_0 c^3}\left(\frac{4}{3}\eta + \frac{(\gamma-1)\kappa}{c_{\mathcal{P}}}\right) \tag{2.63a}$$

$$\alpha_{rot} = \frac{\omega^2}{2\rho_0 c^3}\left(\frac{3}{5}\eta\right) \tag{2.63b}$$

as approximated by the ANSI standard on atmospheric absoption [39], and the relaxation mechanisms due to nitrogen and oxygen as discussed in the previous section.

2.3.7 Combined system model

Combining the models discussed in previous sections one can construct a model for the full system

$$H_{16}^{VV}(f) = \overbrace{H_{12}^{VV}(f)}^{\substack{\text{Transmission line}\\(\text{section } 2.3.2)}} \underbrace{H_{23}^{Vp}(f)H_{34}^{pp}(f)H_{45\text{open}}^{pV}(f)}_{\substack{\text{Reciprocity based propagation model}\\(\text{sections } 2.3.4 \text{ to } 2.3.6)}} \overbrace{H_{5\text{open}5}^{VV}(f)}^{\substack{\text{Transmission line with correction for internal impedance}\\(\text{sections } 2.3.2 \text{ and } 2.3.3)}} H_{56}^{VV}(f)\,, \tag{2.64}$$

where transfer function $H_{ij}^{XY} = {}^{X_i}\!/_{Y_j}$ is defined by the quantities described in table 2.2.

Of the transfer functions in equation (2.64) only $H_{56}^{VV}(f)$ is yet to be discussed. As this subsystem have electrical terminals on either side where arbitrarily short cables can be connected to equipment to measure its transfer function, and the behaviour of the subsystem is not really of interest in the current work, a model to calculate its transfer function has been foregone and instead the transfer function is simply measured separately.

Isolating out all electrical transfer functions gives and expression from which the transfer function of only the acoustic part of the measurement system can be calculated from measured data;

$$H_{25\text{open}}^{VV}(f) = \frac{H_{16}^{VV}(f)}{H_{12}^{VV}(f)H_{5\text{open}5}^{VV}(f)H_{56}^{VV}(f)}\,. \tag{2.65}$$

Quantity	Definition
V_1	Voltage over the cable connecting the transmitting transducer and the oscilloscope (i.e. voltage between signal and ground in t-junction, assumed to be equal to voltage over oscilloscope terminals).
V_2	Voltage over the terminals of the transmitting transducer.
p_3	Pressure free- far-field (possibly back-propagated) on-axis pressure at the reference distance for transmitting transducer.
p_4	Free-field pressure at the reference distance for the receiving transducer.
$V_{5\text{open}}$	Compensated open-circuit voltage over the terminals of the receiving transducer.
V_5	Voltage over the terminals of the first component in the post-processing electronics.
V_6	Voltage over the oscilloscope terminals.

Table 2.2: Definition of quantities used in transfer functions.

Comparing results of this calculation with calculations of $H^{VV}_{25\text{open}}(f)$ based on finite-element modelling (see chapter 3), either with the reciprocity method from section 2.3.4 or with a full finite-element solution (beyond the scope of the current work), verification of the diffraction model from section 2.4.4 can be done.

2.4 Diffraction correction

2.4.1 Definition

Diffraction is the result of receivers having an extended receiving surface, which leads to multiple pick-ups of the transmitted signal. As the distance from the origin of the transmitted signal varies with each pickup point the travel time of the picked up signal will also have a variable delay, resulting in multiple phase shifted copies of the same signal being picked up.

Assuming each pickup point on the receiver contributes equally to the received signal, the average of the field variable carrying the signal at the pickup points will be proportional to the received signal, i.e if the received signal is given as a voltage and the signal propagated as a pressure wave then

$$V \propto \langle p \rangle \,. \tag{2.66}$$

In the above relation the coefficient of proportionality is often identified as the receivers sensitivity.

To account for this effect, diffraction correction amounts to creating a relation between the average of the field variable carrying the signal over the surface of the receiver and an equivalent plane wave being picked up by a point receiver [9; 6];

$$H_{\text{dif}} = \frac{\langle p \rangle}{p_{\text{plane}}} \,. \tag{2.67}$$

Dividing through by this factor in equation (2.66) cancels the factor $\langle p \rangle$ yielding an equation that can be used to relate received signals to their equivalent diffraction-less plane wave representations. This correction can also be introduced into the system model as a "transfer function" to correct for relative diffraction

$$C_{\text{dif}}(z) = \frac{p_{\text{corr}}(z)}{p_{\text{meas}}(z)} = \frac{\frac{z_{\text{ff}}}{z} H_{\text{dif}}(z_{\text{ff}})}{H_{\text{dif}}(z)} \,, \tag{2.68}$$

as shown by Mosland [14, Chapter 2, p. 11]. This essentially accounts for all diffraction effects that exists at distance z but is gone by distance z_{ff}, thus if z_{ff} is placed at a distance sufficiently far away one can correct for all but the far-field limit of the diffraction.

2.4.2 Far-field diffraction correction

One of the simpler approximations for a piezoelectric disc is the model for a uniformly vibrating circular piston mounted flush in a rigid baffle of infinite extent [27, Chapter 5]. This model is not an accurate model for a piezoelectric transducer in all regimes, but due to the way the diffraction corrections in the subsequent section are normalized it will yield the same diffraction correction in the far-field limit of both the baffled piston diffraction correction and the simplified finite element diffraction correction. The reason for choosing this approximation is that the corresponding Rayleigh integral [41, Chapter 15, p. 162; 20, Chapter 7, p. 179],

$$p(\vec{x}) = i\frac{\rho_0 c v_0}{\lambda} \int\limits_S \frac{e^{-ik|\vec{x}'|}}{|\vec{x}'|} dS\,, \tag{2.69}$$

where $|\vec{x}'|$ is the distance between the surface element dS and $\vec{x}$, has a closed form solution in certain regions, most notably on the acoustic axis, and under far-field assumptions.

As C_{dif} involves the diffraction correction in the far-field it is beneficial to have a closed form expression for this. Assuming a transmitter and an acoustically transparent receiver with radii a and b respectively, aligned on each other's axes, and located in each other's far-field, the above expression can be simplified, since $\vec{x} \approx (0, 0, z_{\text{ff}})$, which makes the integrand in the Rayleigh integral a constant. Averaging as prescribed in the diffraction correction formula yields

$$\langle p \rangle (z_{\text{ff}}) = \int\limits_{S_b} \left(i\frac{\rho_0 c v_0}{\lambda} \int\limits_{S_a} \frac{e^{-ikz_{\text{ff}}}}{z_{\text{ff}}}\, dS_a \right) \frac{dS_b}{\pi b^2} = \frac{i\rho_0 c v_0 k a^2}{2 z_{\text{ff}}} e^{-ikz_{\text{ff}}}\,. \tag{2.70}$$

Using the above expression the numerator of (2.68) can be expressed as

$$\frac{z_{\text{ff}}}{z} H_{\text{dif}}(z_{\text{ff}}) = \frac{ika^2}{2z}\,. \tag{2.71}$$

2.4.3 Baffled piston diffraction correction

Outside the regions mentioned in the previous section where the baffled piston model has a closed form solution it still has integral solutions, i.e. the Rayleigh integral. Another such integral solution, due to King [8], can be constructed by solving the wave equation in cylindrical coordinates in terms of Bessel functions of the first kind;

$$p(r, z) = i\rho_0 c v_0 k a \int\limits_0^\infty e^{-\mu z} J_0(\alpha r) J_1(\alpha a)\, \frac{d\alpha}{\mu}\,, \tag{2.72}$$

for $\mu = \sqrt{\alpha^2 - k^2}$.

This expression lends itself better to construction of a near-field diffraction correction along the lines of the far-field diffraction correction from the last section, since the integrand simplifies when averaged over a receiver of the same dimensions as the transmitter, as shown by Williams [7];

$$\langle p \rangle (z) = \rho_0 c v_0 \left(e^{-ikz} - \frac{4}{\pi} \int\limits_0^{\pi/2} e^{-ik\sqrt{z^2 + (2a\cos(\theta))^2}} \sin^2(\theta)\, d\theta \right)\,. \tag{2.73}$$

The corresponding diffraction correction was tabulated by Khimunin [9; 10];

$$H_{\text{dif}}^{\text{BPDC}}(z) = 1 - \frac{4}{\pi} \int\limits_0^{\pi/2} e^{-ik\left(\sqrt{z^2 + (2a\cos(\theta))^2} - z\right)} \sin^2(\theta)\, d\theta\,. \tag{2.74}$$

2.4.4 Simplified finite element diffraction correction

To account for diffraction effects that occur for piezoelectric discs but not for baffled pistons Lunde, Frøysa, Kippersund, and Vestrheim proposed to use finite element analysis to calculate the average pressure [12] used in the diffraction correction, and relate this to an equivalent baffled piston model.

Solving equation (2.70) for v_0 gives a way to back-propagate the average far-field pressure at z_{ff} to an equivalent front velocity of a baffled piston, which multiplied with $\rho_0 c$ gives an equivalent plane wave pressure to use in diffraction correction.

$$p_0^{\text{eq,plane}} = \rho_0 c v_0^{\text{eq}} = \frac{2z_{\text{ff}}\langle p\rangle(z_{\text{ff}})}{ika^2} e^{ikz_{\text{ff}}}\,. \tag{2.75}$$

The final diffraction correction can then be calculated by taking the average pressure over the receiver area using finite element analysis and dividing by the above equivalent plane wave;

$$H_{\text{dif}}^{\text{SFDC}}(z) = \frac{\langle p\rangle(z)}{\langle p\rangle(z_{\text{ff}})}\frac{ika^2}{2z_{\text{ff}}} e^{ik(z-z_{\text{ff}})} = \frac{\langle p\rangle(z)e^{ikz}}{\langle p\rangle(z_{\text{ff}})e^{ikz_{\text{ff}}}} H_{\text{dif}}(z_{\text{ff}})\,. \tag{2.76}$$

2.5 Finite element analysis

As most processes in nature are described by partial differential equations, solving these equations is vital to understand it. While many algebraic equations and ordinary differential equations can be solved in closed form, most partial differential equations can not.

It is however often possible to approximate the solution of partial differential equations numerically given proper constraints. Therefore it is of particular interest to have methods that allow this kind of approximation.

In the following section a general deduction of the framework needed for the finite elements simulation in chapter 3 are presented. The deductions made have partly been based on similar deductions from [42; 19; 18; 43] and should largely agree with the stated resulting finite element matrices in [13; 14; 15; 16; 17] except for the method of enforcing Dirichlet boundary conditions, which instead follows COMSOL MULTIPHYSICS [18; 43].

2.5.1 Problem

The field of partial differential equations is a large field, and as such defining a canonical partial differential equation accounting for every possible variation is not easy. A large family of equations can however be written on the form

$$D[u(\vec{x})] = g(\vec{x}) \text{ in } \Omega\,. \tag{2.77}$$

Here the mapping g and the differential operator $D[\cdot]$ is generally known while the mapping u is sought. This form can be used for a number of different types of mappings, and thus type notation for them will here be suppressed, but in many cases u and g will simply be functions relating a scalar or vector to another scalar or vector. This simple mapping happens in the case where only one governing principle is taken into account in the problem.

For more complicated phenomena where multiple interlinked governing principles act together multiple interlinked differential equations would need to be formed. To accommodate this in the concise notation above it would have to be extended to a mixed formulation where the mappings u and g map to tuples of scalars or vectors, however for simplicity an indexed set

of equations

$$\begin{cases} D_1[u_1(\vec{x}), u_2(\vec{x}), \ldots, u_j(\vec{x}), \ldots, u_n(\vec{x})] = g_1(\vec{x}) \text{ in } \Omega_1 \\ D_2[u_1(\vec{x}), u_2(\vec{x}), \ldots, u_j(\vec{x}), \ldots, u_n(\vec{x})] = g_2(\vec{x}) \text{ in } \Omega_2 \\ \vdots \\ D_i[u_1(\vec{x}), u_2(\vec{x}), \ldots, u_j(\vec{x}), \ldots, u_n(\vec{x})] = g_i(\vec{x}) \text{ in } \Omega_i \\ \vdots \\ D_n[u_1(\vec{x}), u_2(\vec{x}), \ldots, u_j(\vec{x}), \ldots, u_n(\vec{x})] = g_n(\vec{x}) \text{ in } \Omega_n \end{cases} \tag{2.78}$$

will instead be used with the tuple

$$u(\vec{x}) = (u_1(\vec{x}), u_2(\vec{x}), \ldots, u_j(\vec{x}), \ldots, u_n(\vec{x})) \tag{2.79}$$

substituted with the arguments to the differential operator $D[\cdot]$ for economy of notation. This results in an almost as concise notation as (2.77), yielding

$$D_i[u(\vec{x})] = g_i(\vec{x}) \text{ in } \Omega_i \,. \tag{2.80}$$

As constants of integration will have to be eliminated from solutions of differential equations boundary conditions will also have to be specified before a partial differential equation problem can be solved. There are many ways of including such conditions, but some of the more usual ones are to include extra equations of the Dirichlet type

$$u_i = u_i^0(u(\vec{x})) \text{ on } \Gamma_i^D \,, \tag{2.81}$$

or of the Neumann type

$$\frac{\partial u_i}{\partial n} = t_i^0(u(\vec{x})) \text{ on } \Gamma_i^N \,. \tag{2.82}$$

For many problems these will simply be constants, but for interlinked problems they are also often used to tie together adjacent sub-domains with different governing principles. This can be done for example by require continuity in a corresponding function in each domain.

The current discussion will however be restricted to partial differential problems on the form

$$\begin{cases} D_i[u(\vec{x})] = 0 = \nabla \cdot T_i[u(\vec{x})] - U_i[u(\vec{x})] \\ \qquad = \nabla \cdot \sum_{j=1}^{n} T_{ij}[u_j(\vec{x})] - \sum_{j=1}^{n} U_{ij}[u_j(\vec{x})] & \text{in } \Omega_i \\ u_i = u_i^0(u(\vec{x})) \\ \qquad = \sum_{j=1}^{n} u_{ij}^0(u_j(\vec{x})) + u_{i0}^0 & \text{on } \Gamma_i^D \\ T_i[u(\vec{x})] \cdot n = t_i^0(u(\vec{x})) \\ \qquad = \sum_{j=1}^{n} t_{ij}^0(u_j(\vec{x})) + t_{i0}^0 & \text{on } \Gamma_i^N \end{cases} , \tag{2.83}$$

which might seem daunting, but the essential parts here is that none of the operators in the problem contain cross-terms between different $u_j(\vec{x})$ and all $D_i[\cdot]$ contain terms with a divergence operator in front of it. Both of these properties of the problem will be exploited in the following derivation of the finite element method, but this does not mean that the finite element method is restricted to problems with these properties.

Note that the last equation above will for $T_i[\cdot]$ being a gradient like operator, which is the case for all applications in the current work, essentially be a generalized Neumann boundary condition, and thus is labelled as such here.

The reason for the above choice of equations is because this form covers both problems similar to Poisson's equation, which describes many aspects of mechanical deformations due to stress and strains, problems similar to Helmholtz' equation, which describes wave propagation, and interlinking of these. Thus the above set of equations can be used to describe waves generated by harmonic deformation of solids, vice versa.

2.5.2 Weak formulation

The problem posed in the previous section may in some circumstances not have a solution in the conventional sense, as in there might not exist a function that when substituted into the set of equations given perfectly cancels both sides due to discontinuities. This problem can be avoided by generalising the concept of a solution to also include so-called "weak solution".

To derive the weak form of equation (2.80) first transform it into the equivalent integral form

$$\int_{\Omega_i} w_{ik}(\vec{x}) \,\widetilde{\otimes}\, D_i[u(\vec{x})]\, d\Omega_i = \int_{\Omega_i} w_{ik}(\vec{x}) \,\widetilde{\otimes}\, g_i(\vec{x})\, d\Omega_i \,. \tag{2.84}$$

by multiplying by the arbitrary test function $w_{ik}(\vec{x})$ in the same function space as $u(\vec{x})$. $w_{ik}(\vec{x})$ is indexed by k here to signify that multiple different possible functions could be picked, even though the space of functions from which it originates might not actually be indexable. The $\widetilde{\otimes}$ operator introduced is defined as the scalar tensor product, i.e. if the factors are vectors it would be the dot product while if the factors are matrices it would be the double dot product, etc.

The above integral equation is essentially an equation of two inner products, and for economy of notation will be annotated as such;

$$\langle w_{ik},\, D_i[u] \rangle_{\Omega_i} = \langle w_{ik}, g_i \rangle_{\Omega_i} \,. \tag{2.85}$$

Note that in this notation function arguments are suppressed as is customary.

Performing integration by parts on the above equation now introduces the weak solutions by "lifting some of the differential burden" on the sought functions $u_j(\vec{x})$ because some terms involving them loose some differential operators in favour of newly introduced differential operators on the test function.

Reformulating the problem at hand yields

$$\langle w_{ik},\, U_i[u] \rangle_{\Omega_i} + \langle \nabla w_{ik},\, T_i[u] \rangle_{\Omega_i} = \langle w_{ik},\, T_i[u] \cdot n \rangle_{\Gamma_i} \tag{2.86}$$

where Γ_i is the boundary around the domain Ω_i. It is immediately obvious that for the part of the boundary where Neumann boundary conditions are imposed these can simply be substituted straight into the above equation, however Dirichlet boundary conditions does not fit as well.

There are several ways of incorporating Dirichlet boundary conditions into the weak formulation, a quite versatile of these being the method of Lagrange multipliers. Defining

$$T_i[u(\vec{x})] \cdot n = \lambda_i(\vec{x}) \text{ on } \Gamma_i^D \tag{2.87}$$

introduces an extra unknown function $\lambda_i(\vec{x})$, which allows incorporation of the Dirichlet boundary conditions directly as a new equation without making the system overdetermined;

$$\langle v_{ik},\, u_i \rangle_{\Gamma_i^D} = \langle v_{ik},\, u_i^0(u) \rangle_{\Gamma_i^D} \,. \tag{2.88}$$

In the above equation a new test function $v_{ik}(\vec{x})$ has to be introduced as $u_i(\vec{x})$ is not in the same function space as $w_{ik}(\vec{x})$.

Since Dirichlet and Neumann boundary conditions act on different parts of the boundary introducing them now that both of them "look like" Neumann boundary conditions is done by splitting the boundary into a Dirichlet part Γ_i^D and a Neumann part Γ_i^N:

$$\langle \cdot, \cdot \rangle_{\Gamma_i} = \langle \cdot, \cdot \rangle_{\Gamma_i^D} + \langle \cdot, \cdot \rangle_{\Gamma_i^N} \,. \tag{2.89}$$

Putting together the above formulation for the differential problem and its boundary conditions one can rearrange the weak formulation into its generic bilinear-linear form,

$$\begin{cases} \sum_{j=1}^{n} a_{ij}^{u}(u_j(\vec{x}),\, w_{ik}(\vec{x})) + \sum_{j=1}^{m} a_{ij}^{u\lambda}(\lambda_j(\vec{x}),\, w_{ik}(\vec{x})) &= L_i^u(w_{ik}) \\ \sum_{j=1}^{n} a_{ij}^{\lambda u}(u_j(\vec{x}),\, v_{ik}(\vec{x})) &= L_i^{\lambda}(v_{ik}) \end{cases}, \tag{2.90}$$

by organizing terms by unknown function. This can be done because there is no cross-terms between any of the unknown functions. The newly introduced functionals $a_{ij}^{\square}(\cdot,\,\cdot)$ and $L_i^{\square}(\cdot)$ are given in the equation list below.

$$a_{ij}^{u}(u_j(\vec{x}),\, w_{ik}(\vec{x})) = \langle w_{ik},\, U_{ij}[u_j]\rangle_{\Omega_i} + \langle \nabla w_{ik},\, T_{ij}[u_j]\rangle_{\Omega_i} - \langle w_{ik},\, t_{ij}^{0}(u_j)\rangle_{\Gamma_i^N} \tag{2.91a}$$

$$a_{ij}^{u\lambda}(\lambda_j(\vec{x}),\, w_{ik}(\vec{x})) = -\langle w_{ik},\, I_{ij}[\lambda_j]\rangle_{\Gamma_i^D} \tag{2.91b}$$

$$a_{ij}^{\lambda u}(u_j(\vec{x}),\, v_{ik}(\vec{x})) = \langle v_{ik},\, I_{ij}[u_j]\rangle_{\Gamma_i^D} - \langle v_{ik},\, u_{ij}^{0}(u_j)\rangle_{\Gamma_i^D} \tag{2.91c}$$

$$L_i^{u}(w_{ik}) = \langle w_{ik},\, t_{i0}^{0}\rangle_{\Gamma_i^N} \tag{2.91d}$$

$$L_i^{\lambda}(v_{ik}) = \langle v_{ik},\, u_{i0}^{0}\rangle_{\Gamma_i^D} \tag{2.91e}$$

Note that many of the functionals will be zero when $i \neq j$. Examples of this is $a_{ij}^{u\lambda}(\cdot,\,\cdot)$ because only one λ_j is defined per governing equation, and $a_{ij}^{\lambda u}(\cdot,\,\cdot)$ when Dirichlet boundary conditions are constant. To annotate the factors which are always zero when $i \neq j$ the operator

$$I_{ij}[f(\cdot)] = \begin{cases} f(\cdot) & \text{if i = j} \\ 0 & \text{else} \end{cases} \tag{2.92}$$

has been introduced.

2.5.3 Weighted residual methods

For linear differential equations solutions are generally a weighted sum of a fundamental set of solutions, but for partial differential equations there is no known method to solve for this fundamental set generally, and therefore solutions must be approximated.

One method of approximating solutions to differential equations is to assume a set of "fundamental solutions" $\psi_{jl}(\vec{x})$ for each solution $u_j(\vec{x})$ and do a weighted sum over the set. More rigorously, this amounts to projecting the solution into the function space spanned by the assumed functions, $\Psi_j = span(\psi_{j1}(\vec{x}),\, \psi_{j2}(\vec{x}),\, \ldots,\, \psi_{jl}(\vec{x}),\, \ldots,\, \psi_{jp_j}(\vec{x}))$, called the approximation space for $u_j(\vec{x})$.

$$\widetilde{u}_j = proj_{\Psi_j}(u_j) = \sum_{l=1}^{p_j} \alpha_{jl}\psi_{jl}(\vec{x}) \tag{2.93}$$

Since the Lagrange multiplier functions are also unknowns they also need to be projected into an approximation space $\Phi_j = span(\phi_{j1}(\vec{x}),\, \phi_{j2}(\vec{x}),\, \ldots,\, \phi_{jl}(\vec{x}),\, \ldots,\, \phi_{jq_j}(\vec{x}))$.

$$\widetilde{\lambda}_j = proj_{\Phi_j}(\lambda_j) = \sum_{l=1}^{q_j} \beta_{jl}\phi_{jl}(\vec{x}) \tag{2.94}$$

Substituting $\widetilde{u}_j$ and $\widetilde{\lambda}_j$ for u_j and λ_j, and exploiting the linear properties of the bilinear-linear

formulation from the previous section yields a set of linear equations:

$$\begin{cases} \sum_{j=1}^{n}\sum_{l=1}^{p_j} \alpha_{jl} a_{ij}^{u}(\psi_{jl}(\vec{x}),\, w_{ik}(\vec{x})) + \sum_{j=1}^{m}\sum_{l=1}^{q_j} \beta_{jl} a_{ij}^{u\lambda}(\phi_{jl}(\vec{x}),\, w_{ik}(\vec{x})) &= L_i^u(w_{ik}(\vec{x})) \\ \sum_{j=1}^{n}\sum_{l=1}^{p_j} \alpha_{il} a_{ij}^{\lambda u}(\psi_{jl}(\vec{x}),\, v_{ik}(\vec{x})) &= L_i^\lambda(v_{ik}(\vec{x})) \end{cases} . \tag{2.95}$$

Picking as many test functions, $w_{ik}(\vec{x})$ and $v_{ik}(\vec{x})$, as base functions in the corresponding projection spaces Ψ_i and Φ_i one can replicate the above equations for each test function to form a matrix equation

$$\begin{cases} \sum_{j=1}^{n} K_{ij} U_j + \sum_{j=1}^{m} N_{ij}^F \Lambda_j &= L_i \\ \sum_{j=1}^{n} N_{ij} U_j &= M_i \end{cases} , \tag{2.96}$$

where the newly introduced matrices and vectors are given as:

$$K_{ij}U_j = \begin{pmatrix} a_{ij}^u(\psi_{j1}, w_{i1}) & \cdots & a_{ij}^u(\psi_{jp_j}, w_{i1}) \\ \vdots & \ddots & \vdots \\ a_{ij}^u(\psi_{j1}, w_{ip_i}) & \cdots & a_{ij}^u(\psi_{jp_j}, w_{ip_i}) \end{pmatrix} \begin{pmatrix} \alpha_{j1} \\ \vdots \\ \alpha_{jp_j} \end{pmatrix} \tag{2.97a}$$

$$N_{ij}^F\Lambda_j = \begin{pmatrix} a_{ij}^{u\lambda}(\phi_{j1}, w_{i1}) & \cdots & a_{ij}^{u\lambda}(\phi_{jq_j}, w_{i1}) \\ \vdots & \ddots & \vdots \\ a_{ij}^{u\lambda}(\phi_{j1}, w_{ip_i}) & \cdots & a_{ij}^{u\lambda}(\phi_{jq_j}, w_{ip_i}) \end{pmatrix} \begin{pmatrix} \beta_{j1} \\ \vdots \\ \beta_{jq_j} \end{pmatrix} \tag{2.97b}$$

$$N_{ij}U_j = \begin{pmatrix} a_{ij}^{\lambda u}(\psi_{j1}, v_{i1}) & \cdots & a_{ij}^{\lambda u}(\psi_{jp_j}, v_{i1}) \\ \vdots & \ddots & \vdots \\ a_{ij}^{\lambda u}(\psi_{j1}, v_{iq_i}) & \cdots & a_{ij}^{\lambda u}(\psi_{jp_j}, v_{iq_i}) \end{pmatrix} \begin{pmatrix} \alpha_{j1} \\ \vdots \\ \alpha_{jp_j} \end{pmatrix} \tag{2.97c}$$

$$L_i = \begin{pmatrix} L_i^u(w_{i1}) \\ \vdots \\ L_i^u(w_{ip_i}) \end{pmatrix} \tag{2.97d}$$

$$M_i = \begin{pmatrix} L_i^\lambda(v_{i1}) \\ \vdots \\ L_i^\lambda(v_{iq_i}) \end{pmatrix} \tag{2.97e}$$

Combining the above matrices for every equation i and unknown j into one big partitioned matrix according to the summation gives

$$\begin{pmatrix} K_{11} & \cdots & K_{1n} & N_{11}^F & \cdots & N_{1m}^F \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ K_{n1} & \cdots & K_{nn} & N_{n1}^F & \cdots & N_{nm}^F \\ N_{11} & \cdots & N_{1n} & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ N_{m1} & \cdots & N_{mn} & 0 & \cdots & 0 \end{pmatrix} \begin{pmatrix} U_1 \\ \vdots \\ U_n \\ \Lambda_1 \\ \vdots \\ \Lambda_n \end{pmatrix} = \begin{pmatrix} L_1 \\ \vdots \\ L_n \\ M_1 \\ \vdots \\ M_n \end{pmatrix} , \tag{2.98}$$

which for economy of notation can be further be simplified to

$$\begin{pmatrix} K & N^F \\ N & 0 \end{pmatrix} \begin{pmatrix} U \\ \Lambda \end{pmatrix} = \begin{pmatrix} L \\ M \end{pmatrix} , \tag{2.99}$$

which is essentially a matrix equation on the form $Ax = b$.

By selecting the test function to be the same as the base functions for the corresponding projection spaces, $w_{ik} = \psi_{ik}$ and $v_{ik} = \phi_{ik}$, the above method turns into the Galerkin method which for symmetric bilinear terms $a_{ij}^{\square}(\cdot,\cdot)$ and $a_{ij}^{u\lambda}(\cdot,\cdot) = a_{ij}^{\lambda u}(\cdot,\cdot)$ will result in a symmetric matrix problem.

2.5.4 Numerical integration

To compute the matrix A and the vector b from the previous section a numerical integration technique is needed since most of the non-zero elements are defined in terms of integrals.

One approach to to evaluating these integrals is using numerical quadrature

$$\int_D f(x)\, dD = \sum_i c_i f(x_i)\,, \tag{2.100}$$

which reduces the integration to a simple sum.

Since the quadrature approach is in general not exact it is important to pick weightings a_i and sample points b_i carefully. If non-representative sample points or poor weightings are picked quadratures can yield wildly incorrect results, but for some choices of projection spaces Ψ_k and Φ_i it is possible to pick them such that the quadrature yields exact values for the integrals. An example of this is the Gaussian quadrature of n terms which is exact for polynomials of degree $2n - 1$ or below.

2.5.5 Discretization

In the finite element method the base functions in the function spaces Ψ_i and Φ_i are usually selected as a set of bump functions filling the domain Ω_i such that only a few number of bump functions have support on any given subsection of the domain. For brevity construction of these base functions will be done in terms of ψ_i here as construction is identical for all function spaces.

To avoid encoding redundant information each bump function ψ_i is "centred" on a unique point in the domain so that

$$\psi_i(\vec{x}_j) = \delta_{ij}\,, \tag{2.101}$$

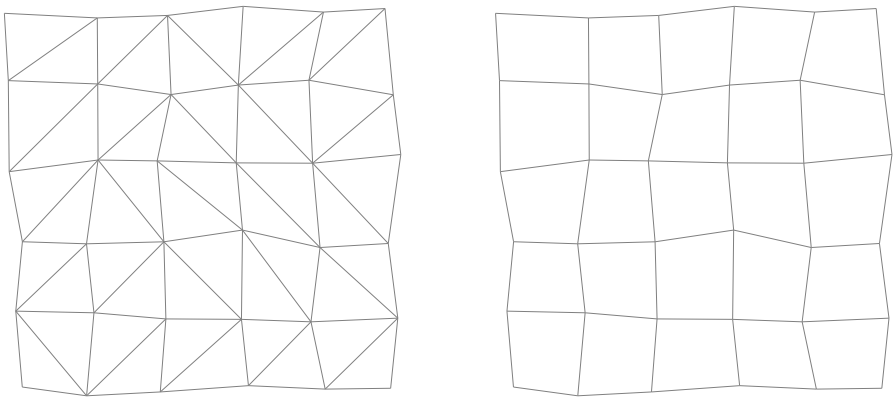
where $\vec{x}_j$ is the "center" of $\psi_j(\cdot)$ and δ_{ij} is the Kronecker delta. The "center" in this context is taken to be the highest point inside the support of the associated function.

The above selection of base functions naturally leads to a grid-like structure, which can easily be seen by scattering the "center" points of all base functions, called nodes, in the domain and drawing up grid lines between adjacent ones. Such a discretisation might look like one of the meshes in fig. 2.6.

As seen in fig. 2.6 multiple different meshes can be constructed for a single domain. Choice of mesh like this is highly problem dependant, but common meshes consist of triangle or quadrilateral mesh elements.

The discretisation done thus far has yielded a mesh with nodes in the corners of each mesh element, which for first order elements are usually fine, as these elements tend to have their nodes in the corners anyway. For higher order elements additional nodes are usually placed on edges of the element, or in its interior. The interpolation functions associated with interior nodes are usually defined with support that does not extend outside the element. This is the same as adding restrictions when scattering the nodes in the first place, and when drawing up a mesh such that all the elements end up with additional nodes where prescribed.

To simplify working with the base function each element can be considered separately if one splits base functions associated with nodes located on edges between elements into parts witch only have support in each of the elements. Thus the original base function for a node can be



(a) Triangles (b) Quadrilaterals

Figure 2.6: Two possible discretizations of a domain with different element types.

reconstructed by summing over the parts

$$\psi_p(\vec{x}) = \sum_{q \in \text{node } p} \varphi_q(\vec{x})\,. \tag{2.102}$$

To simplify further each element can be distorted to look like a given reference element by introducing a shift of coordinates. Thus all integrations done can be done over a pre-defined element instead of on a highly irregular mesh.

$$\psi_i(\vec{x})\,d\Omega = \psi_i(\vec{\chi}(\vec{x}))|J(\vec{x}))|\,d\Omega_e \tag{2.103}$$

This shift of coordinates inevitably introduces a Jacobian of the form

$$J = \begin{pmatrix} \frac{\partial \chi_1}{\partial x_1} & \cdots & \frac{\partial \chi_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial \chi_n}{\partial x_1} & \cdots & \frac{\partial \chi_n}{\partial x_n} \end{pmatrix}, \tag{2.104}$$

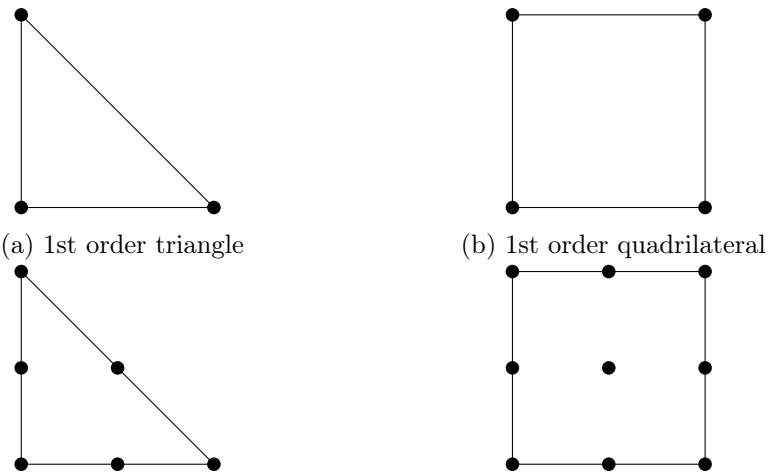
which will need to be computed for a given distortion. To approximate this calculation the whole element's shape is approximated with a set of mapping functions between local coordinates $\vec{\chi}$ and global coordinates $\vec{x}$. A common choice is to do this approximate mapping with the base functions associated with the element in the same way the sought function is approximated, which yields iso-parametric elements.

Depending on what shape of elements in the mesh selected different families and order of elements are available. Figure 2.7 shows the first and second order elements from the Lagrange family of elements, which is a common choice. To derive the base functions for a Lagrange type element one can utilize the orthogonality property from equation (2.101). By setting the characteristic polynomial, here shown for a triangle with vertices at (0, 0), (1, 0), and (0, 1),

$$\varphi_i(\xi,\eta) = a_{i1} + a_{i2}\xi + a_{i3}\eta + a_{i4}\xi^2 + a_{i5}\xi\eta + a_{i6}\eta^2\,, \tag{2.105}$$

equal to zero at every node but one a Vandermonde matrix problem can be formulated

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & .5 & 0 & .25 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & .5 & .5 & .25 & .25 & .25 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & .5 & 0 & 0 & .25 \end{pmatrix} \begin{pmatrix} a_{i1} \\ a_{i2} \\ a_{i3} \\ a_{i4} \\ a_{i5} \\ a_{i6} \end{pmatrix} = \begin{pmatrix} \delta_{i1} \\ \delta_{i2} \\ \delta_{i3} \\ \delta_{i4} \\ \delta_{i5} \\ \delta_{i6} \end{pmatrix}. \tag{2.106}$$



(a) 1st order triangle (b) 1st order quadrilateral

(c) 2nd order triangle (d) 2nd order quadrilateral

Figure 2.7: 1st and 2nd order triangle and quadrilateral reference elements.

The solution to this problem are the Lagrange polynomials shown below and plotted in fig. 2.8.

$$\varphi_1 = 1 - 3\xi - 3\eta + 2\xi^2 + 4\xi\eta + 2\eta^2 \tag{2.107a}$$
$$\varphi_2 = 4\xi - 4\xi^2 - 4\xi\eta \tag{2.107b}$$
$$\varphi_3 = -\xi + 2\xi^2 \tag{2.107c}$$
$$\varphi_4 = 4\xi\eta \tag{2.107d}$$
$$\varphi_5 = -\eta + 2\eta^2 \tag{2.107e}$$
$$\varphi_6 = 4\eta - 4\xi\eta - 4\eta^2 \tag{2.107f}$$

2.6 Helmholtz-Kirchhoff equation

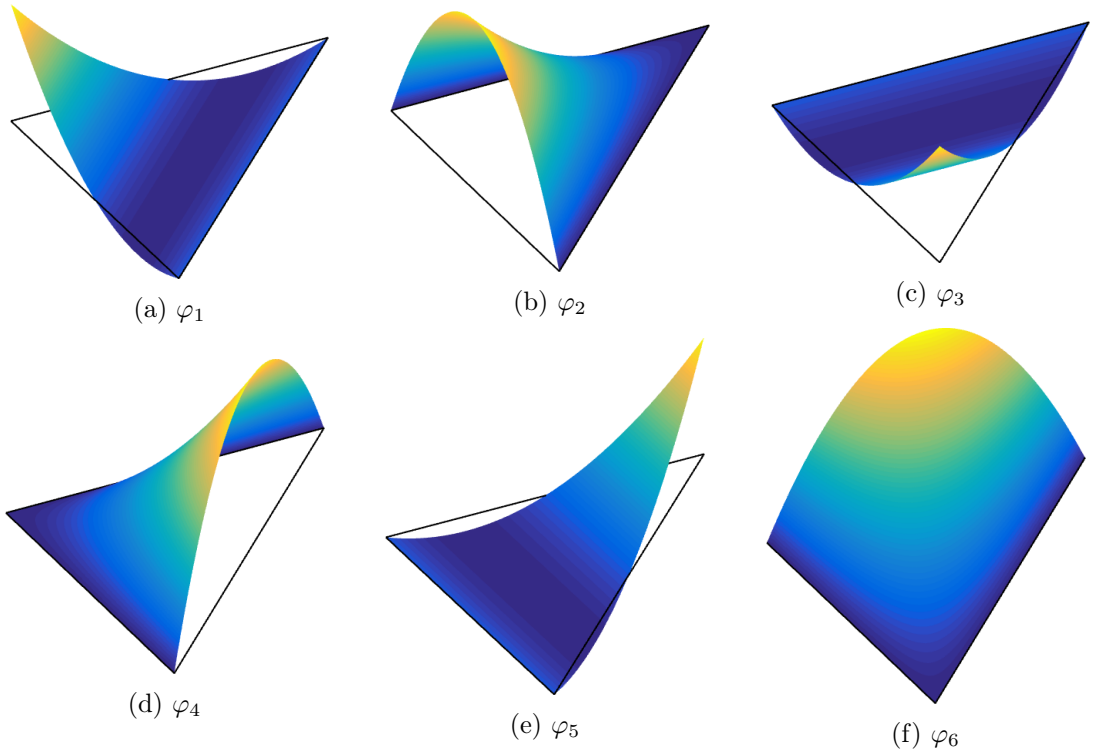
The normal formulation of Helmholtz-Kirchhoff equation gives a way to calculate an acoustic field at an arbitrary point enclosed inside a surface given that the field and its derivative is know on the enclosing surface [21, Chapter 1], and all acoustical sources and obstructions are located outside the surface. In this thesis it will be necessary to calculate the acoustic field in this way, but since this will be used to extend the effective domain of finite element simulations the relation will need to be restated in terms of the field outside the surface where all acoustical sources and obstructions are inside the surface. A similar derivation has been done in the user manual for Comsol [18, Evaluating the Acoustic Field in the Far-Field Region], but the derivation itself is not given. For completeness sake it will be given here.

2.6.1 Green's theorem

Given a vector field $\vec{\varphi}(\vec{x})$ the divergence theorem states that the sum of all divergence in a volume enclosed by a surface is equal to the flux of the vector field passing through that surface. Thus we can form the equation

$$\iiint_A \nabla \cdot \vec{\varphi}\, dV = \oiint_{S_I} \vec{\varphi} \cdot \hat{n}_I\, dS \tag{2.108}$$

to evaluate the divergence of volume A in figure (2.9).



(a) φ_1 (b) φ_2 (c) φ_3

(d) φ_4 (e) φ_5 (f) φ_6

Figure 2.8: Interpolation functions for a 2nd order Lagrangian triangle reference element.

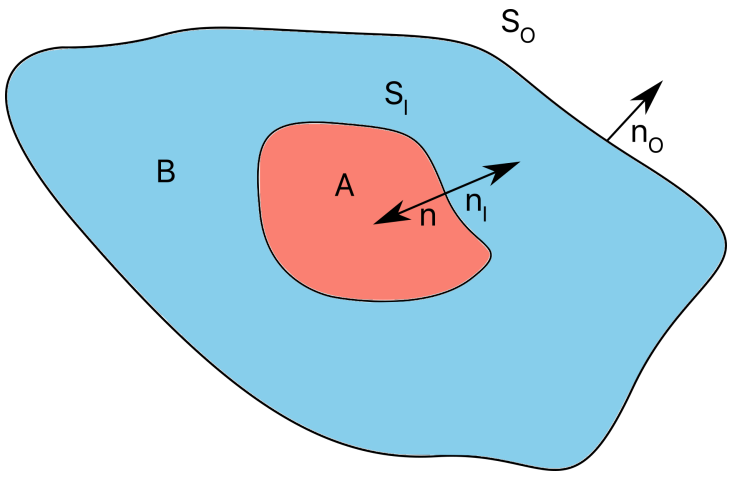


Figure 2.9: Domains

The divergence theorem also holds for the union of domain A and B together with the surface around domain B since B in figure (2.9) fully encapsulates domain A, yielding

$$\iiint\limits_{A\cup B} \nabla\cdot\vec{\varphi}\,dV = \iiint\limits_{A} \nabla\cdot\vec{\varphi}\,dV + \iiint\limits_{B} \nabla\cdot\vec{\varphi}\,dV = \oiint\limits_{S_O} \vec{\varphi}\cdot\hat{n}_O\,dS\,. \tag{2.109}$$

Combining equation (2.108) and equation (2.109) we can relate the volume integral over domain B to the surface integral around domain A (on the surface S_I) together with the surface integral around domain B (on the surface S_O). Thus we have

$$\iiint\limits_{B} \nabla\cdot\vec{\varphi}\,dV = \oiint\limits_{S_O} \vec{\varphi}\cdot\hat{n}_O\,dS - \oiint\limits_{S_I} \vec{\varphi}\cdot\hat{n}_I\,dS\,. \tag{2.110}$$

Defining $\vec{\varphi}$ as $G\nabla p - p\nabla G$, where both $p(\vec{x})$ and $G(\vec{x})$ are scalar fields, and applying the identity $\nabla\cdot(U\nabla V) = U\nabla^2 V + \nabla U\cdot\nabla V$ twice yields

$$\iiint\limits_{B} (G\nabla^2 p - p\nabla^2 G)\,dV = \oiint\limits_{S_O} (G\nabla p - p\nabla G)\cdot\hat{n}_O\,dS - \oiint\limits_{S_I} (G\nabla p - p\nabla G)\cdot\hat{n}_I\,dS \tag{2.111}$$

which is Green's theorem (similar to [20, Appendix A8, p. 521]) for a domain squeezed in between two separate surfaces.

2.6.2 Outer boundary

To deal with the outer boundary we introduce Sommerfeld's radiation condition as given in [44, Chapter 4, p. 178].

$$\lim_{r\to\infty} r\left(\frac{\partial\phi}{\partial r} + \frac{1}{c}\frac{\partial\phi}{\partial t}\right) = 0 \tag{2.112}$$

By looking for solutions of the form $\phi = \phi_0 e^{i\omega t}$ to this equation, and dividing through by r (doing this effectively ignores the decay rate constraint given in Sommerfeld's equation by multiplying by r, and thus is a looser boundary condition) this equation can be simplified to (see [19, Chapter 3, p. 46])

$$\nabla\phi\cdot\hat{n}_O = -ik\phi\,. \tag{2.113}$$

Moving the boundary S_O to infinity while imposing Sommerfeld's radiation condition for both p and G lets us cancel everything in the integral over the surface S_O from equation (2.111), and yields

$$\oiint\limits_{S_O} (G\nabla p - p\nabla G)\cdot\hat{n}_O\,dS = \oiint\limits_{S_O} (-Gikp + pikG)\,dS = 0\,. \tag{2.114}$$

2.6.3 Pressure in the extended domain

To calculate the pressure in domain B we have to solve the wave equation,

$$\left(1 + \tau_S\frac{\partial}{\partial t}\right)\nabla^2 p = \frac{1}{c^2}\frac{\partial^2 p}{\partial t^2}\,, \tag{2.115}$$

in this domain.

If we look for solutions of the form $p = p_0 e^{i\omega t}$ the wave equation reduces to Helmholtz equation

$$(\nabla^2 + k^2)p = 0 \tag{2.116}$$

where $k = \frac{\omega}{c}/\sqrt{1+i\omega\tau_S}$ (see [20, Chapter 8, p. 212] for a more detailed discussion of this).

A Green's function can be designed to solve Helmholtz equation (2.116) by taking advantage of the sifting property of the Dirac delta function. By using $\vec{r} - \vec{r}_q$ as a the parameter to the delta function this property lets us use the delta function to evaluate some function under an integral at a query point $\vec{r}_q$.

$$(\nabla^2 + k^2)G = -\delta(\vec{r} - \vec{r}_q) \tag{2.117}$$

Solving equation (2.116) and (2.117) for $\nabla^2 p$ and $\nabla^2 G$ respectively and inserting the resulting expression into the volume integral from equation (2.111) yields the expression

$$\iiint_B (G\nabla^2 p - p\nabla^2 G)\, dV = \iiint_B p\delta(\vec{r} - \vec{r}_q)\, dV = p(\vec{r}_q)\,, \tag{2.118}$$

which cancels everywhere except at the query point $\vec{r}_q$.

2.6.4 Formula

Inserting the results from equation (2.118) and (2.114) into equation (2.111), and interchanging $\hat{n}_I$ with $-\hat{n}$, yields the Helmholtz-Kirchhoff equation.

$$p(\vec{r}_q) = \oiint_{S_I} (G\nabla p - p\nabla G) \cdot \hat{n}\, dS \tag{2.119}$$

To evaluate this equation we need a closed form solution for G. The literature [21; 44; 20] gives the solution to equation (2.117) as

$$G = \frac{e^{-ik|\vec{r} - \vec{r}_q|}}{4\pi|\vec{r} - \vec{r}_q|}\,. \tag{2.120}$$

The gradient of G can easily be found since G is spherically symmetric, and thus the gradient only has a non zero derivative in the radial component.

$$\nabla G = \frac{dG}{dr} = \frac{(\vec{r} - \vec{r}_q)e^{-ik|\vec{r} - \vec{r}_q|}(-1 - ik|\vec{r} - \vec{r}_q|)}{4\pi|\vec{r} - \vec{r}_q|^3} = -G\frac{1 + ik|\vec{r} - \vec{r}_q|}{|\vec{r} - \vec{r}_q|^2}(\vec{r} - \vec{r}_q) \tag{2.121}$$

Inserting equation (2.120) and (2.121) into (2.119) yields the expression for the far field pressure as given in [18, Evaluating the Acoustic Field in the Far-Field Region].

$$p(\vec{r}_q) = \frac{1}{4\pi} \oiint_{S_I} \frac{e^{-ik|\vec{r} - \vec{r}_q|}}{|\vec{r} - \vec{r}_q|} \left(\nabla p + p\frac{1 + ik|\vec{r} - \vec{r}_q|}{|\vec{r} - \vec{r}_q|^2}(\vec{r} - \vec{r}_q) \right) \cdot \hat{n}\, dS \tag{2.122}$$

2.6.5 Verification of the solution

To verify that equation (2.120) is actually a solution to equation (2.117) integrate the equation over a sphere centred at $\vec{r}_q$, and apply the divergence theorem (as suggested by [20, Chapter 5, p. 142]) to get

$$\iiint_V (\nabla^2 + k^2) G \, dV = \oiint_S \nabla G \cdot \hat{n} \, dS + k^2 \iiint_V G \, dV \,. \tag{2.123}$$

To ease evaluation define $s = |\vec{r} - \vec{r}_q|$. This together with spherical coordinates eliminates the need for absolute values in the expressions and equation (2.120) and (2.121) reduces to $G = {}^{e^{-iks}}/_{4\pi s}$ and $\nabla G = -G(ik + {}^1/_s)$ respectively.

The surface integral in the first term of equation (2.123) only contains constant terms as G is spherically symmetric. Thus to integrate it simply multiply by a factor of $4\pi r^2$. This, together with defining the radius of the sphere as R, yields

$$\oiint_S \nabla G \cdot \hat{n} \, dS = -e^{-ikR}(ikR + 1) \,. \tag{2.124}$$

The volume integral in the second term of equation (2.123) can be divided up into a spherical integral integrated over a radial integral. Again the spherical integral can be evaluated by multiplying by a factor of $4\pi r^2$ due to G being spherically symmetric, yielding

$$\oiint_S G \, dS = s e^{-iks} \,. \tag{2.125}$$

and taking this result as the sphere to be integrated in the radial direction gives the equation

$$k^2 \iiint_V G \, dV = k^2 \int_0^R s e^{-iks} \, ds = -1 + e^{-ikR}(1 + ikR) \,. \tag{2.126}$$

Summing equation (2.124) and (2.126) yields the result -1. By definition integrating the delta function yields a value of 1 and thus the left hand side and the right hand side of equation (2.123) is equal.

Since we did not specify the sphere integrated over more than its center we can make the sphere arbitrarily small and equation (2.123) through (2.126) will still hold. Thus we can conclude that G is translated correctly in relation to the delta function in equation (2.116), and that G actually is a solution.

Chapter 3

Simulation

3.1 Geometry

The size and complexity of the geometry used in finite element simulations is one of the major factors in computation time (this is immediately obvious as each mesh element contributes a number of degrees of freedom to the problem at hand). Therefore it is of great interest to keep the geometry as simple as possible. The two transducer transmitter and receiver problem is in general a three dimensional problem spanning at least the separation distance between the two transducers, but modelling such a problem has proven to be a challenge with the hardware at hand.

Simplifications is therefore made by assuming the two transducers are reciprocal (see section 2.3.4), thus reducing the needed geometry to only the transmitting transducer, and assuming that the problem is rotationally symmetric (i.e. the transducers don't have any rotational modes), thus reducing the problem to a half space two dimensional problem.

This simplified geometry is shown in fig. 3.1,

3.2 Materials

For each domain in the geometry a corresponding material needs to be selected. In the current thesis the default air material properties in COMSOL Multiphysics are used for the medium and PML domain while the material properties for Pz27 given by Aanes [45, Chapter 5, p. 78] is used for the piezo.

In the medium domain the relevant material property is density, which is given by

$$\rho = \frac{0.02897p}{8.314T}\,, \tag{3.1}$$

where p is ambient pressure in pascals and T is the thermodynamic temperature in kelvin.

For the piezo the relevant material properties are density, which is specified as $7700\,{}^{\mathrm{kg}}/_{\mathrm{m}^3}$, elasticity matrix which is given in table 3.1, coupling matrix which is given in table 3.2, and relative permittivity matrix which is given in table 3.3.

3.3 Equations

The equations solved in each domain is dependant on the purpose of the domain. In the geometry used there exists three distinct domains, one which models a piezoelectric disc and two which acts as fluid loading for said disc.

In the piezoelectric medium the acoustic wave equation and associated electric potential equilibrium equation presented in section 2.1.3 is used, while the two medium domains are modelled by Helmholtz equation from section 2.1.1.

Figure 3.1: The geometry used

$$\begin{pmatrix} c_{11}^E & c_{12}^E & c_{13}^E & 0 & 0 & 0 \\ c_{12}^E & c_{11}^E & c_{13}^E & 0 & 0 & 0 \\ c_{13}^E & c_{13}^E & c_{33}^E & 0 & 0 & 0 \\ 0 & 0 & 0 & c_{44}^E & 0 & 0 \\ 0 & 0 & 0 & 0 & c_{44}^E & 0 \\ 0 & 0 & 0 & 0 & 0 & c_{66}^E \end{pmatrix}$$

Constant	Value ($\times 10^{10}\,\mathrm{N\,m^{-2}}$)
c_{11}^E	$12.025 \quad (1 + \mathrm{i}/96)$
c_{12}^E	$7.62 \quad (1 + \mathrm{i}/70)$
c_{13}^E	$7.42 \quad (1 + \mathrm{i}/120)$
c_{33}^E	$11.005 \quad (1 + \mathrm{i}/190)$
c_{44}^E	$2.11 \quad (1 + \mathrm{i}/75)$
c_{66}^E	$2.16 \quad (1 + \mathrm{i}/315)$

Table 3.1: Pz27 elasticity matrix

$$\begin{pmatrix} 0 & 0 & e_{31} \\ 0 & 0 & e_{31} \\ 0 & 0 & e_{33} \\ 0 & e_{15} & 0 \\ e_{15} & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Constant	Value ($\mathrm{C\,m^{-2}}$)
e_{31}	$-5.4 \quad (1 - \mathrm{i}/166)$
e_{33}	$17.0 \quad (1 - \mathrm{i}/324)$
e_{15}	$11.2 \quad (1 - \mathrm{i}/200)$

Table 3.2: Pz27 coupling matrix

$$\begin{pmatrix} \epsilon_{11}^S/\epsilon_0 & 0 & 0 \\ 0 & \epsilon_{11}^S/\epsilon_0 & 0 \\ 0 & 0 & \epsilon_{33}^S/\epsilon_0 \end{pmatrix}$$

Constant	Value ($\times 10^{-9}\,\mathrm{F\,m^{-1}}$)
ϵ_{11}^S	$8.110\,44 \quad (1 - \mathrm{i}/50)$
ϵ_{33}^S	$8.145\,85 \quad (1 - \mathrm{i}/130)$

Table 3.3: Pz27 relative permittivity matrix

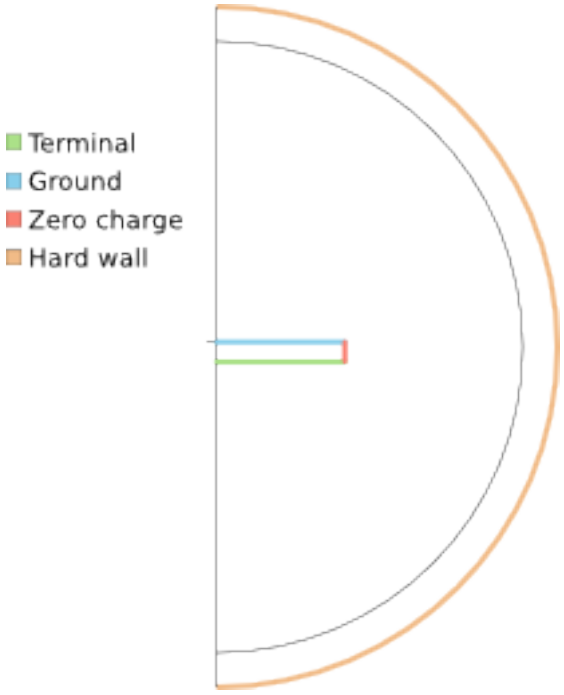


Figure 3.2: The boundary conditions used

There also exists two boundaries in the geometry that need to be constrained in order to solve the problem at hand, namely the outer boundary of the problem domain and the boundary between the piezoelectric domain and the medium domain.

To deal with the end of the simulation domain at the outer edge of the problem domain a hard wall boundary condition is applied, as

$$\frac{\partial p}{\partial n} = 0\,. \tag{3.2}$$

The exact condition applied here does not matter too much as the PML region between this boundary and the inner medium domain will dampen down any wave travelling through it, and this boundary condition will not have a chance to interact with waves of any appreciable amplitude.

At the boundary between the piezoelectric domain and the medium domain two boundary conditions are applied to connect the two domains since they are formulated with different variables.

The boundary condition

$$\frac{\partial p}{\partial n} = -\rho \frac{d^2\vec{u}}{dt^2} \cdot \vec{n} \tag{3.3}$$

expresses the pressure in the medium domain generated by displacements on the surface of the piezoelectric domain, and the boundary condition

$$T \cdot \vec{n} = p\vec{n} \tag{3.4}$$

expresses the stress in the piezoelectric domain generated by pressure in the medium domain. These are both essentially just Newtons second law stated in different ways.

For the specific case of a piezoelectric disc the radial edge of the disc does not have an electric connection to the two terminals on either side of the disk. Therefore there is no way to build up charge on this boundary. To model this constraint the boundary condition

$$\vec{D} \cdot \vec{n} = 0 \tag{3.5}$$

is applied.

On each terminal the voltage at that terminal is imposed as a boundary condition as

$$V = V_0 \,. \tag{3.6}$$

For the ground terminal this value is zero, while on the voltage terminal the a value of one is chosen. As long as the results of interest are just relative quantities like the sensitivity or the impedance of the disc the actual value chosen for the voltage terminal doesn't matter much outside of problems relating to floating point round-off errors.

3.4 Perfectly matched layer

As mentioned earlier in the outermost simulation domain a perfectly matched layer (PML) is used to absorb outgoing waves so that the hard wall boundary condition applied to the outermost boundary doesn't produce reflections traveling back into the medium domain, interfering with the simulated fields.

This can be done since smoothly introducing an imaginary part $-i\vec{\sigma}(\vec{x})/\omega$ to the coordinates inside the PML domain (analytical continuation of the solution), yielding exponentially decaying solutions in it, doesn't influence solutions computed outside the domain [46, Section 3, pp. 6-8].

$$e^{-i\vec{k}\cdot\vec{x}} \to e^{-i\vec{k}\cdot\vec{x}'} = e^{-i\vec{k}\cdot\vec{x}} e^{-\left(\frac{\vec{k}}{\omega}\right)\cdot\vec{\sigma}(\vec{x})} \tag{3.7}$$

In the current work the default PML in COMSOL Multiphysics is used, which is given by the coordinate transform

$$\vec{x}' = \vec{x} - \vec{n}_\xi \xi L + \vec{n}_\xi f(\xi) \tag{3.8}$$

for a PML domain of length L in the propagation direction $\vec{n}_\xi$, which is the normal to the inner boundary of the domain [43, Chapter 5, pp. 298-299]. ξ in the above expression linearly maps the position inside the PML from 0 to 1 so that $\xi = 0$ and $\xi = 1$ corresponds to the inner and outer boundary of the PML domain respectively. This results in the coordinates inside the PML domain in the propagation direction $\vec{n}_\xi$ to be exclusively controlled by the coordinate stretching function $f(\cdot)$; e.g. in three dimensional spherical coordinates if a PML domain is placed in the region $r_0 \leq r \leq r_0 + L$ the transformed coordinates in it will be given by

$$\vec{x}' = \begin{pmatrix} r_0 \\ \theta \\ \phi \end{pmatrix} + \begin{pmatrix} f(\xi) \\ 0 \\ 0 \end{pmatrix} \,. \tag{3.9}$$

COMSOL Multiphysics' default coordinate stretching function is also chosen for $f(\cdot)$;

$$f(\xi) = s\lambda\xi \left(\frac{1}{3p(1-\xi)+4} - \frac{i}{3p(1-\xi)} \right) \,, \tag{3.10}$$

where s and p are tuneable parameters to compensate for obliquely incident waves and a curved PML boundary, left at their default value of 1 in the current work as these are optimisation parameters to allow for coarser meshes to preform better. This coordinate stretching function resizes the PML domain so that is is always a quarter of a wavelength long in the transformed coordinates, with $\vec{\sigma}(\vec{x})$ going to infinity at the outer boundary of the PML domain.

3.5 Mesh

As the finite element method relies on discretisation of space in order to solve the problem at hand, a mesh is needed. It is important that such a mesh is fine-grained enough to capture the characteristics of the problem.

The PML region in the othermost layer is topologically identical with a rectangle and as such a quadratic element mesh is easily fitted in this region. The inner regions are harder to mesh with quadratic elements, and due to the utilised software not supporting any practical way of meshing circular regions with these elements triangular elements are used instead. This choice of elements can be assumed to raise the lower limit of mesh granularity to achieve convergence as triangular elements generally converges slower than their quadratic counterpart [42, Chapter 2, pp. 38-39, Chapter 4, pp. 110-119].

Since the current work is dealing with single frequency bursts the Nyquist-Shannon sampling theorem sets an absolute minimum mesh granularity of two elements per wavelength, however such coarse meshes have been found to be inadequate to properly capture the characteristics of the time harmonic wave problem. Instead a rule of thumb of five to six elements for accurate magnitude values is found in the literature [47; 19, Chapter 4, pp. 95–96]. In the current work phase information, along with the magnitude information, is of interest and thus a finer mesh granularity is assumed to be needed, and a granularity of about 10 elements per wavelength have been found to yield satisfactory results (which is corroborated by [48] who claims 8 to be sufficient).

3.6 Solver

To actually solve the posed finite element problem a numerical solver is used. In the current work the direct solver PARDISO [49; 50] have been used without much regard for solution parameters, as these generally only affect the computation speed of the solver. Other solvers, like the iterative Krylov subspace solvers, have been considered, but the matrix assembly stage of the solution process have proved to be the most time consuming step, and thus the gain of using iterative solvers is most likely minimal. Direct solvers also have the advantage of not giving an answer when the problem at hand is ill-posed, and thus erroneous answers from unconstrained problems won't be misinterpreted as correct solutions.

The PARDISO solver uses LU-decomposition to decompose the problem

$$Ax = \begin{pmatrix} a_{11} & \cdots & a_{n1} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix} x = b \tag{3.11}$$

into a lower and upper triangular matrix problem

$$LUx = \begin{pmatrix} l_{11} & 0 & \cdots & 0 \\ l_{21} & l_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{pmatrix} \begin{pmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & u_{22} & \cdots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & u_{nn} \end{pmatrix} x = b\,, \tag{3.12}$$

which in turn can be solved quite quickly with a forward substitution step $Ld = b$ followed by a backward substitution step $Ux = d$.

PARDISO also avoids the problem of singular L or U matrices in the case where the matrix A has zeros on its diagonal by permuting the matrix and the corresponding vector b with one or more permutation matrices.

3.7 Post-processing

As the quantities computed by the finite element analysis described in the previous sections only yields values for the pressure, displacement and voltage fields additional post-processing of the data is required to obtain the desired transfer functions and diffraction corrections.

Since the medium domain used in the simulations is only slightly bigger than the transmitting piezoelectric disc it is unlikely that the far-field pressure used in the calculation of transfer functions and diffraction corrections exists inside it, and as such the extrapolation method described in section 2.6.4 is used to obtain it, integrating over the boundary between the medium and the PML domains. According to the COMSOL Multiphysics manual [18, Evaluating the Acoustic Field in the Far-Field Region] this works for distances in the vicinity of the simulation domain, but the technique is numerically unstable when extrapolations are carried out too far away from it. A distance of $z_{\text{ff}} = 10\,\text{m}$ is used in the current work, which in all scenarios studied is over 30 times the Rayleigh distance.

While calculating the diffraction correction $H_{\text{dif}}^{\text{SFDC}}(f)$, and its associated transfer function $C_{\text{dif}}^{\text{SFDC}}(f)$, can be calculated with only the simulated field variables and the above extrapolation the transfer function $H_{25\text{open}}^{VV}(f)C_{\text{dif}}(f)$ additionally needs the value of $Z_T = {}^{V_0}/_{I}$.

The current I can be calculated by integrating the normal component in the current direction of the current density over one of the terminals of the piezoelectric disc [23]

$$I = \int_S \vec{J} \cdot \vec{n}\, dS = \int_S \frac{\partial \vec{D}}{\partial t} \cdot \vec{n}\, dS\,, \tag{3.13}$$

and said current density can be calculated as the time derivative of the displacement field from the constitutive equations given in section 2.1.3.

Chapter 4

Measurement

4.1 Experimental setup

The experimental setup used is inherited from Andersen, and is largely kept as described in his master thesis [17, Chapter 3, pp. 32-42].

A Tektronix DPO3012 digital oscilloscope is used to digitise the signal as it enters the cable leading to the transmitter, and then again as the signal leaves the receiver electronics. For more accurate waveform readings the oscilloscope is configured to average 128 consecutive bursts for each reading.

To generate the signals used for measurements an Agilent 33220A arbitrary waveform generator is used. To get the best readings possible the generator is configured to output at its highest possible voltage, 10 V, except when measuring close to the radial modes (80 kHz to 120 kHz and 240 kHz to 260 kHz); for which it instead is configured to output at 1 V (as suggested by Andersen) and 100 mV for the first and second mode respectively to avoid the transducers behaving non-linearly [17, Chapter 6, pp. 96-96].

The receiver electronics consists of a Brüel & Kjær 2636 measurement amplifier configured to amplify the signal by 60 dB and a Krohn-Hite 3940A digital filter configured as a high-pass Butterworth filter with a cutoff frequency of 4 kHz. To connect the receiver electronics internally and to the oscilloscope RG-58 coaxial cables with BNC connectors on either end are used.

The acoustic subsystem under measurement is connected to the rest of the measurement setup by RG-178 B/U coaxial cables with BNC connectors on one end and banana jacks on the other. The jacks are present so that transducers can be easily switched, although this feature is not used much in the current work.

Alignment of the system is done using linear and rotational stages from Physik Instrumente (PI): Two linear stages, PI M-535.22 and PI M-531.DG, is used to move the receiver in its lateral plane, and another linear stage, PI miCos LS270, is used to move the transmitter along its longitudinal axis. A PI M-037.PD worm-gear rotational stage is additionally mounted on the transmitter to rotate it about its vertical axis.

To measure the electric characteristics of the transducers for use in the transmission line model from section 2.3.2 a HP 4192A impedance analyser is used.

Environmental parameters are recorded with a ASL F250 MKII thermometer, a Vaisala HMT313 hygrometer, and a Brüel & Kjær UZ0004 correction barometer.

The transducers being measured is made by previous master students at the University of Bergen [14–17], and is constructed by soldering leads with banana plugs at one end to 20×2 mm PZ-27 discs and taping on a rods to the leads for suspending the transducers. Some of these transducers have additionally been shielded by the current author by wrapping 2-3 layers of aluminum foil around the leads, held in place by electrical tape, as seen in fig. 4.1. The reason for the simplistic transducer construction is to ease comparison with simulations [14, Chapter 9, p. 125; 15, Chapter 8, p. 121].

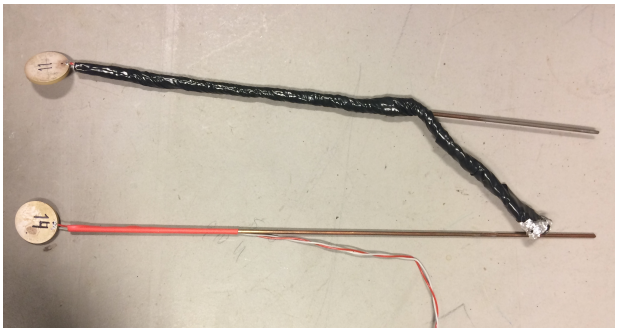


Figure 4.1: The two types of transducers used: shielded (above) and unshielded (below).

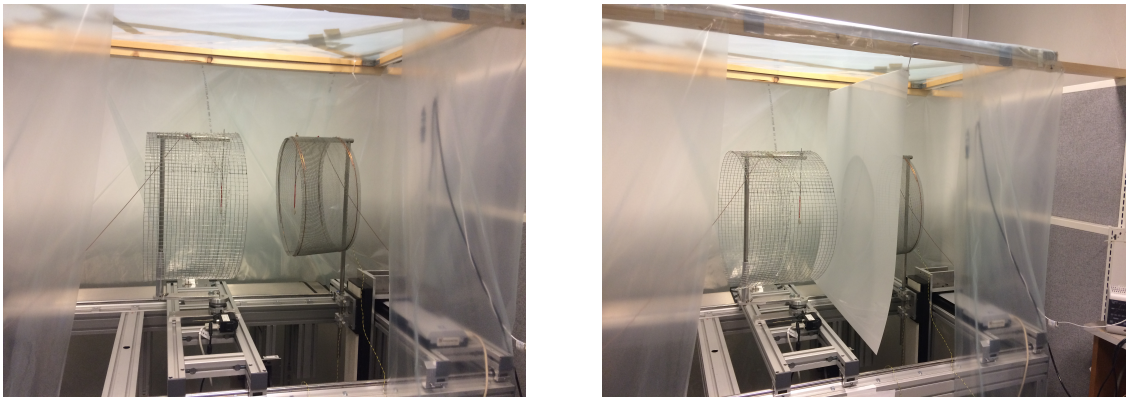


Figure 4.2: The acoustic measurement setup used with and without the plastic screen blocking the acoustic path inserted.

To reduce coherent electrical noise both the transmitting and receiving transducer is encircled by a Faraday cage; whose effect have been explored by Andersen [17, Chapter 6, p. 88]. To further reduce this noise the current author has additionally introduced a measurement step involving measurement of the coherent noise without the acoustical signal (see section 4.3.5) and as such a plastic screen to block the acoustic signal path has also been introduced. The measurement setup with the Faraday cages and with and without the plastic screen can be seen in fig. 4.2.

4.2 Measuring transfer functions

The transfer function of a system was previously defined and discussed in section 2.3.1, and was shown to be the ratio of the Fourier transform of an output signal and its corresponding input signal. The method used in the current work to obtain this ratio is to measure input and output waveforms of tone-bursts of sufficiently concentrated frequencies in the frequency range of interest, calculating the Fourier components from the measured waveforms.

For the measurement setup at hand the transfer functions measured this way will contain spurious contributions from the non-acoustic components in the system; notably the cables connecting the acoustic subsystem to the rest of the system, and the receiver electronics (i.e. the amplifier and the filter, including cables connecting them to each other and to the point of measurement). Additionally there is also spurious contributions from coherent electrical noise generated by the transmitting transducer, due to full or partial lack of shielding. The removal of these contributions are done by taking their associated transfer functions into account after calculating the transfer function of the whole system, as described in the following subsections.

4.2.1 Burst length and measurement window

When measuring transfer functions with the waveform approach, coarsely outlined above, the first thing to take into account is for how long to excite the system measured with the tone-burst, and when to sample the systems input and output signals. Optimally one would want to have a long run of sampled data in steady-state for each frequency in the measurement range, so that a discrete Fourier transform can yield an accurate result for each of them.

In section 2.3.1 it is shown that a system's impulse response weights previous system inputs contribution to current output; and as such the length of a system's transient period when subjected to new input can be as long as it's impulse response (the length here measured as the time taken until all subsequent values are below a negligible threshold). To satisfy the the above outlined optimal measurement condition one have to introduce a delay after subjecting the system under measurement to a new signal before sampling (or discard data sampled during this delay) to avoid sampling the system's transient, and another delay between two subsequent signals to allow for the former to die down.

For some systems there might be other constraints, other than its transient length, that constrains when measurements can be taken. In the current work this arrises when measuring the full system; when spurious reflections from the environment and the system itself should not be part of the measured waveform. This effectively sets an upper limit to the time between the start of the tone-burst and the end of the measurement equal to the time taken for the first such spurious reflection to reach the receiver.

The above mentioned constraints effectively yield lower and upper bounds for the measurement window, and an upper bound for the burst length; and the measurement window and burst length should be chosen accordingly, maximising the length of the sampled steady-state signal.

4.2.2 Compensating for propagation time

As the propagation speed in the acoustic part of the measured system is not negligible, sampling of the output signal of the system must be delayed accordingly to ensure comparison between equal parts of the input and output signal.

In the current work the delay introduced when measuring the full system is computed using the model presented in section 2.3.5 applied to a plane wave; yielding a delay $t = {}^{d}/_{c}$. The propagation distance d here is the separation distance calibrated during alignment, as described in section 4.3.2. Since the model relies on temperature, pressure, and humidity these quantities are also measured as described in section 4.3.3 and fed into the model.

Adding this correction also ensures that any subsequent phase measurements are now relative to the phase of a plane wave, i.e. one is measuring the slowly varying phase.

4.2.3 Avoiding spectral leakage

As shown in section 2.2.3, for a discrete Fourier transform to be accurate the sampled window needs to contain a whole number of periods of a periodic signal, otherwise the transients "seen" by the discrete Fourier transform at either end of the signal (due to the assumption of the sampled window being a whole number of periods) will cause energy to be redistributed to adjacent bins outside the frequency bin allocated to the actual frequency of the signal [51, Spectral leakage and windowing].

This effect is often referred to as spectral leakage, and several signal processing techniques exists to counter it (e.g. applying additional windowing with a tuned frequency response). In the current work spectral leakage is countered by clipping the signal to an integer number of periods.

The first step in ensuring an integer number of periods in the sampled signal is to transmit an integer number of periods. This further restricts the length of the tone-burst discussed in

section 4.2.1; although this extra constraint is insubstantial for all but the lowest frequencies as the original measurement window is much larger than the signal period.

Subsequently the first and last part of the sampled signal is discarded so that the signal starts and ends in points where the signal's value crosses zero, while maximising the number of integer number of periods retained (for implementation details refer to appendices C.1.9 and C.1.11). This sacrifices some frequency resolution in favour of avoiding spectral leakage.

4.2.4 Calculating transfer functions

When two waveforms have been measured and selected as prescribed in the previous sections the last step to computing a transfer function is to compute the waveforms' discrete Fourier transforms and divide one by the other as described in section 2.3.1. Any algorithm that computes the discrete Fourier transform would suffice for this purpose, and in the current work the Goertzel algorithm is used (as opposed to the plain discrete Fourier transform described in section 2.2.4 or the more standard Cooley-Tukey fast Fourier transform algorithm) as it efficiently computes single frequency discrete Fourier transforms.

Since the implementation of the Goertzel algorithm used (the default implementation shipped with MATLAB 2017a) can only compute the discrete Fourier transform of the default Fourier "bins" extra zero-padding needs to be introduced to shift the center frequency of one of these bins to the frequency of the signal. The center frequency of a Fourier bin is given by $f = k\frac{f_s}{N}$ (see eq. (2.31)) when k is the bin number, and f_s and N is the signals sampling frequency and sample count respectively. As introducing extra zeros at the end of a signal doesn't affect the signal's discrete Fourier transform, but increases N, the Fourier bins can be tuned by this kind of padding. In the current work a selected space of possible paddings are simply searched for a padding that gives the closest match between the signal's frequency and a Fourier bin (for implementation details refer to appendix C.1.10).

4.3 Measurement procedure

4.3.1 Measuring transducer admittances

The transmission line model from sections 2.3.2 and 2.3.3 takes the load of the transmission line and electrical characteristics of the voltage source connected to it, and as such the admittance or impedance of the transducers have to be measured. This is done using the previously mentioned impedance analyser with a frequency sweep over the frequencies to be measured later on when measuring the full system's transfer function while the instrument is connected to the cables leading to the transducers and configured to excite them with 1 V to measure their admittance.

After measurements are done for all desired frequencies the admittance of the transducers are found by applying the transmission line model constrained with the measured admittance on the source side of the cable, and calculating the transducers admittance on the receiver side; given by

$$Y_T = \frac{I_R}{-V_R}\,. \tag{4.1}$$

4.3.2 Transducer alignment

As the transducers are directional their relative rotation must be kept minimal in order to match the head-on setup simulated (chapter 3). In the current work the transducers are aligned by means of laser measurements at close distance before the transmitting transducer is translated along its longitudinal axis into its final position.

Figure 4.3: Closeup of a transducer inscribed with a reticle.

To aid alignment, each transducer is inscribed with a ruler assisted hand-drawn reticle. The axes of the reticle is placed by marking visually perpendicular lines at the cords of the disc face measured to be the longest with the ruler.

When brought close (the distance is limited by the size and range of the laser put in-between the discs, and is in the current measurement setup ≈240 mm [17, Chapter 3, pp. 41-43]), the transducers are aligned first visually so that the alignment process starts in a roughly aligned state. The aforementioned reticles are then centred by placing the laser in the center of one and then the other disc is realigned so that the laser also hits its reticle's center. In the current measurement setup the receiver is actuated in this alignment plane, and as such it is selected for convenience to be moved in this alignment step.

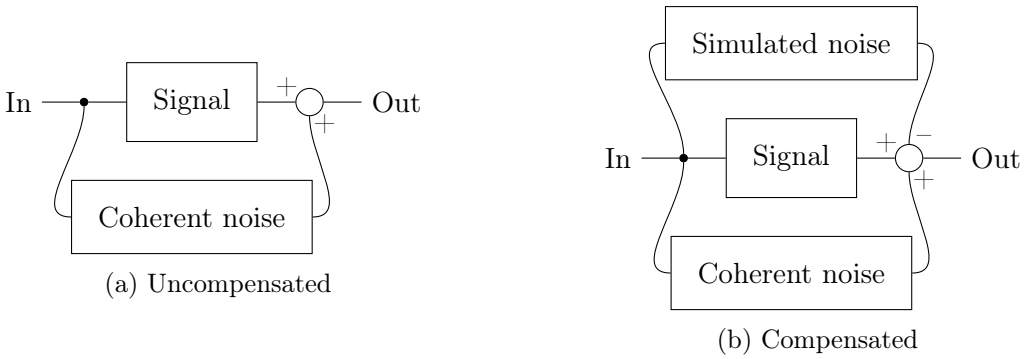
Next separation distances between the transducers are measured and compared. For two perpendicular lateral axes the separation distance is measured at the disc's rim in either end and compared. If the difference in separation distance along an axis is deemed to be too large (in the current work a maximum difference of ≈100 µm is used) the corresponding axis is realigned by rotating the disc accordingly. In the current measurement setup only the vertical rotational axis of the transmitter is actuated, and as such all other axes are hand aligned. The vertical rotational axis of the receiver is aligned by unfastening the transducer from the rest of the measurement setup and rotating; while the horizontal rotational axis is aligned by taking advantage of the stiffness of the cables holding the disc of the transducer up. After alignment of the last axis, all axes are checked again (in case an axis was disturbed while aligning another) and any unaligned axes are realigned.

When all axes have been aligned and verified the transducer is translated along its longitudinal axis to the desired position. The travel distance of the transmitter is calculated by measuring the separation distance between the disc centres and subtracting from the desired separation distance for the measurements to be taken [17, Chapter 3, p. 43].

4.3.3 Measuring full system's transfer function

After aligning the transducers as prescribed in the previous section the full systems's transfer function is measured as described in section 4.2.

As mentioned in section 4.2.2 the model used to compensate for propagation time relies on pressure, temperature, and humidity. While the thermometer and hygrometer used in the current measurement setup can interface with a computer, the barometer used is analogue and thus doesn't have this capability. This means that temperature and humidity readings can be done quickly and precisely along with each waveform measurement, while pressure readings are



(a) Uncompensated

(b) Compensated

Figure 4.4: A system with coherent noise with and with and without additional compensation.

only done once manually at the start of a measurement series and the recorded value is used for every subsequent waveform measurement in the series.

During transit through the system the reduction in signal strength is significant; at ∼100 dB reduction, the signal is sometimes close to the noise floor when recorded at the receiving end. To alleviate this a high output voltage of 10 V peak-to-peak is used at the signal generator throughout most of the frequency spectrum, but as pointed out by Andersen this will cause non-linear behaviour close to the resonant modes of the transducers [17, Chapter 6, p. 96]. To avoid this problem an output voltage of 1 V and 100 mV is instead used in the problematic regions 80 kHz to 120 kHz and 240 kHz to 260 kHz respectively.

4.3.4 Compensating for cables and receiver electronics

As shown in section 2.3.7 the effects of the cables and receiver electronics can be removed from the measured transfer function by dividing through by their respective transfer functions; which for the cables are modelled by the models from sections 2.3.2 and 2.3.3, and for the receiver electronics are simply measured.

Measuring the transfer function for the receiver electronics, H_{56}, is done as described in section 4.2, with no compensation for propagation time or reflections.

With the signal generator at hand the lowest achievable voltage is 10 mV peak-to-peak, which if applied directly to the receiver electronics in the configuration used would be amplified by ≈60 dB to ≈10 V. This is well outside the specified max voltage of the filter used and as such an additional attenuator is inserted after the signal generator to attenuate the generated signal. As this attenuator is inserted before the point where the input signal is measured its effects are accounted for without extra effort.

4.3.5 Compensating for coherent noise

In the current measurement system coherent electrical noise is generated as the transducers act like antennas due to the previously mentioned lack of shielding. As the coherent noise is predictable it can be removed by introducing a negated version of the signal [30, Chapter 3, pp. 41-42], as described by the block diagrams in fig. 4.4.

In the current work the coherent noise is predicted by applying the system's transfer function when the coherent noise is considered it's output, $H_{16\text{noise}}$, to the measured system input waveform. For this to be accurate $H_{16\text{noise}}$ should be measured at all the Fourier bins calculated for the signal it is applied to. These Fourier bins also include negative frequencies which is simply computed as the conjugate of their positive equivalent due to the conjugate symmetry property of Fourier transforms of real signals [52, Properties of the Fourier Transform].

For implementation details see appendix C.1.8, which additionally adds functionality for interpolating the transfer function used and resampling the signal it is applied to if the above

matching criterion is not satisfied. As this more advanced functionality has yielded poorer results than just measuring the transfer function at the needed frequencies it is not taken advantage of in the current work; except for extending the transfer function at lower frequencies by means of nearest neighbour extrapolation when the lowest frequencies have proved hard to measure.

Measuring $H_{16\text{noise}}$ is done as described in section 4.2 with a plastic screen inserted in-between the transducers to block the acoustic signal path. As the coherent electric noise signal propagates virtually instantaneous (as compared to period of the signal) no compensation for propagation time is applied. Reflections are also not considered as the present reflections are simply a part of the coherent noise signal to be compensated for.

Chapter 5

Results

5.1 Noise reduction

To improve measurement precision a simple noise analysis with accompanying reduction techniques have been carried out in the current work. Previous works have tried to reduce noise by introducing averaging [14; 15; 16; 17] (for white noise) and Faraday cages (for coherent electrical noise) and grounding these [16; 17]. The effectiveness of these are also evaluated here.

5.1.1 Low frequency background noise

As electrical equipment tend to create 50 Hz to 60 Hz (depending on operating frequency) background noise all such equipment deemed unnecessary for the measurements at hand was unplugged in the lab before the measurements presented in this chapter was taken.

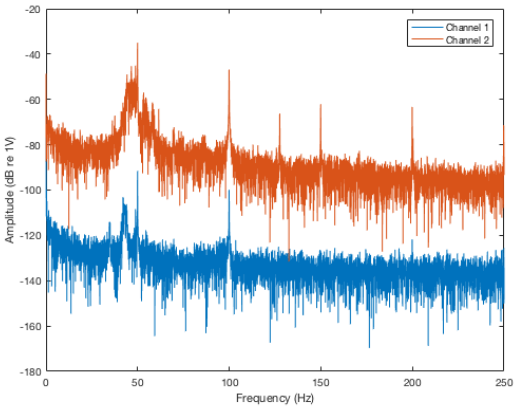
To get a baseline reading of low frequency noise a high resolution (Tektronix patented process which essentially runs the oscilloscope digitiser at the highest possible frequency and downsamples to the desired sample frequency by averaging all samples within a sample period [53, How the Analog Acquisition Modes Work, p. 89]) 20 s voltage trace at a sample rate of 500 Hz was measured of the input and output of the measurement setup. The discrete Fourier transform of this baseline is shown in fig. 5.1a and shows significant spikes at 50 Hz and its overtones. As this noise could be either electrical noise from transformers and the like, or acoustical noise from cooling fans being picked up by the transducers a second measurement with the loudest fans shut off was also taken, yielding the spectrum shown in fig. 5.1b. As no appreciable difference can be spotted between the two the noise is assumed to be electrical and as such must be removed by filtering or better grounding.

Applying the high-pass Butterworth filter used in the measurement procedure at its lowest used cutoff frequency (4 kHz) brings the white noise level of the receiving side down by ≈40 dB as shown in fig. 5.2a, but does not seem to do much to reduce the relative amplitude of the 50 Hz noise.

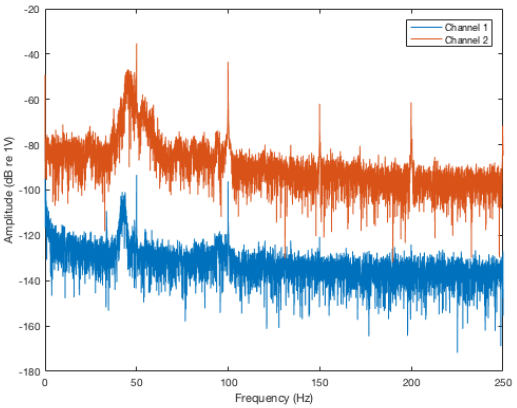
The transfer functions measured in the current work are not measured using the high resolution mode as its effectiveness is diminished by the higher sample rate used during their measurement. When additionally disabling this mode used in this section while measuring the background noise with the filter applied most of the 50 Hz noise does however mostly drown in the added white noise, as shown in fig. 5.2b.

5.1.2 White noise

To combat white noise all measurements taken with the measurement procedure for measuring transfer functions are averaged over 128 subsequent bursts. The effectiveness of this, a reduction in the noise level of ≈20 dB, can be seen from fig. 5.3 in which the frequency spectrums of

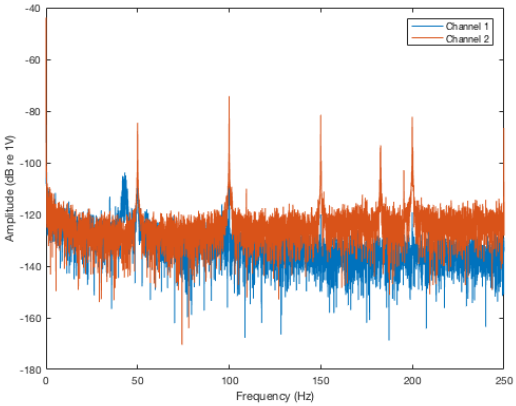


(a) Baseline

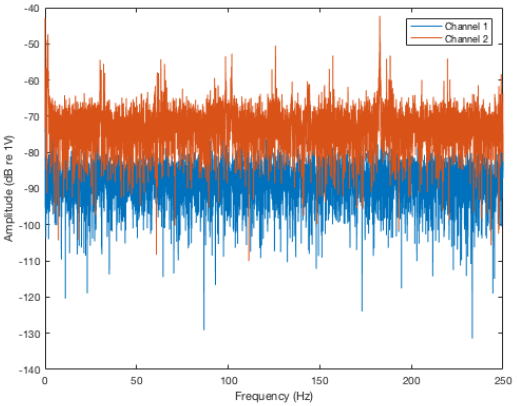


(b) Baseline with no audible noise

Figure 5.1: Frequency spectrums of baseline measurements.

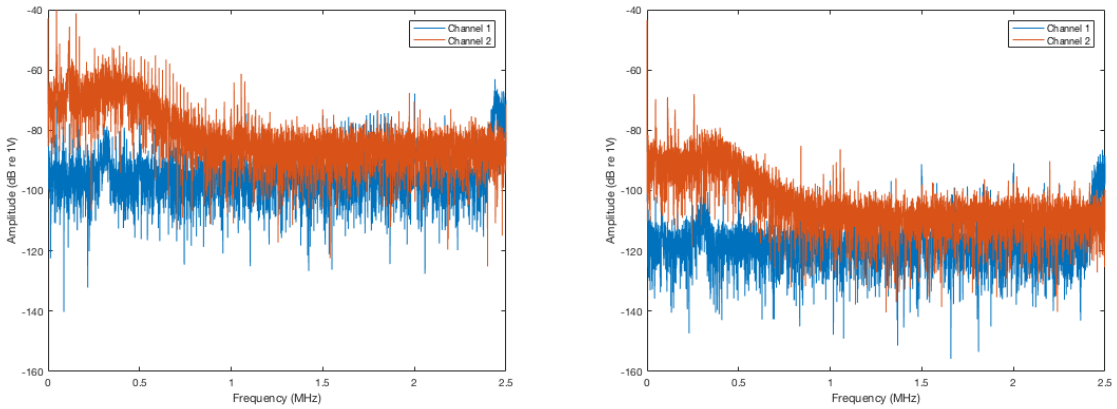


(a) High resolution



(b) Single sampled

Figure 5.2: Frequency spectrum of low frequency background noise.



(a) No averaging

(b) Averaged 128 times

Figure 5.3: Frequency spectrum of background noise.

measurements of the system without a transmitted signal present is taken with and without averaging enabled.

This dropping of the white noise level will additionally reveal some of the 50 Hz noise that was drowned when not averaging as explained in the previous section.

5.1.3 Coherent noise

As discussed previously in chapter 4 an appreciable amount of coherent noise is generated as the transducers are not fully shielded. To sample this coherent noise a signal at the second radial mode (≈260 kHz) of the transducers was propagated through the system as if measuring the system's transfer function. The second radial mode was used as it seemingly generates the most coherent noise of the frequencies investigated in the current work. For comparison the coherent noise was sampled for several different shielding configurations.

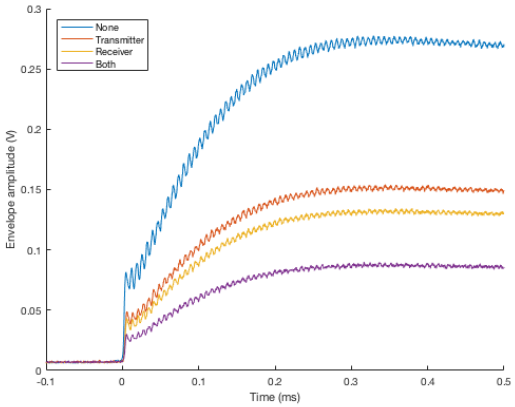
The first evaluated shielding is the Faraday cages used by previous master students [16; 17]. These cages are cylindrical metal meshes with openings in the longitudinal direction to allow for unimpeded sound propagation; as seen in fig. 4.2.

Figure 5.4a compares the envelopes (as calculated by the function envelope in MATLAB 2017a; which calculates $|h + iH(h)|$ for a signal h where H is the Hilbert transform) of the sampled coherent noise and it can be seen that using one Faraday cage drops the coherent noise level by 6 dB to 7 dB depending on which side the cage is placed on, while using both drops it by about 10 dB.

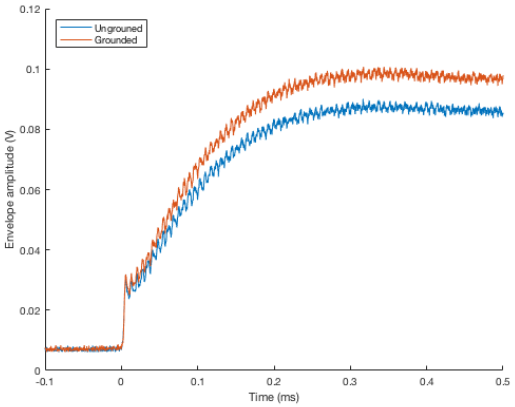
Previous master students additionally grounded the Faraday cage on the receiving side by wiring it to ground at the input of the amplifier with an extra cable [17, Chapter 3, p. 53]. Figure 5.4b shows measurements with and without this extra grounding; and shows an increase in coherent noise by about 1 dB when the extra grounding is used. This extra grounding is thus dropped in all subsequent measurements in the current work.

In the current work new shielding is also introduced in the form of aluminium foil wrapped around the leads of the transducers; as shown in fig. 4.1. The effect of this shielding can be seen in fig. 5.4c; which shows a drop in the coherent noise level of about 2 dB to 3 dB when just one side is shielded, and about 9 dB when both are shielded.

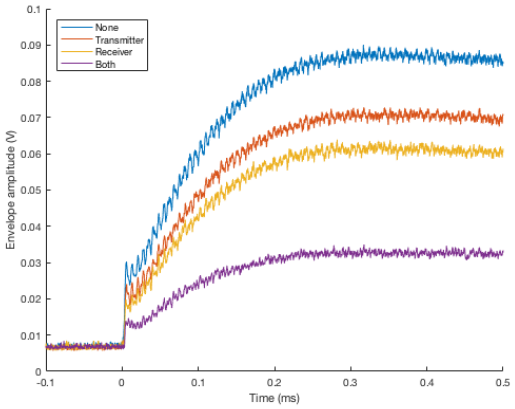
As described in section 4.3.5 a post-processing technique new in the current work is additionally used to remove coherent noise. As measuring the transfer function of the noise used in this post-processing is by far the most time consuming step in the measurement process, due to the number of required measurement frequencies, it is of interest to reuse the measured transfer function for different distances. Figure 5.5 shows the transfer function for two distances



(a) Faraday cages



(b) Grounded Faraday cages



(c) Faraday cages and aluminium foil wrapping

Figure 5.4: Envelope of coherent noise with different shielding.

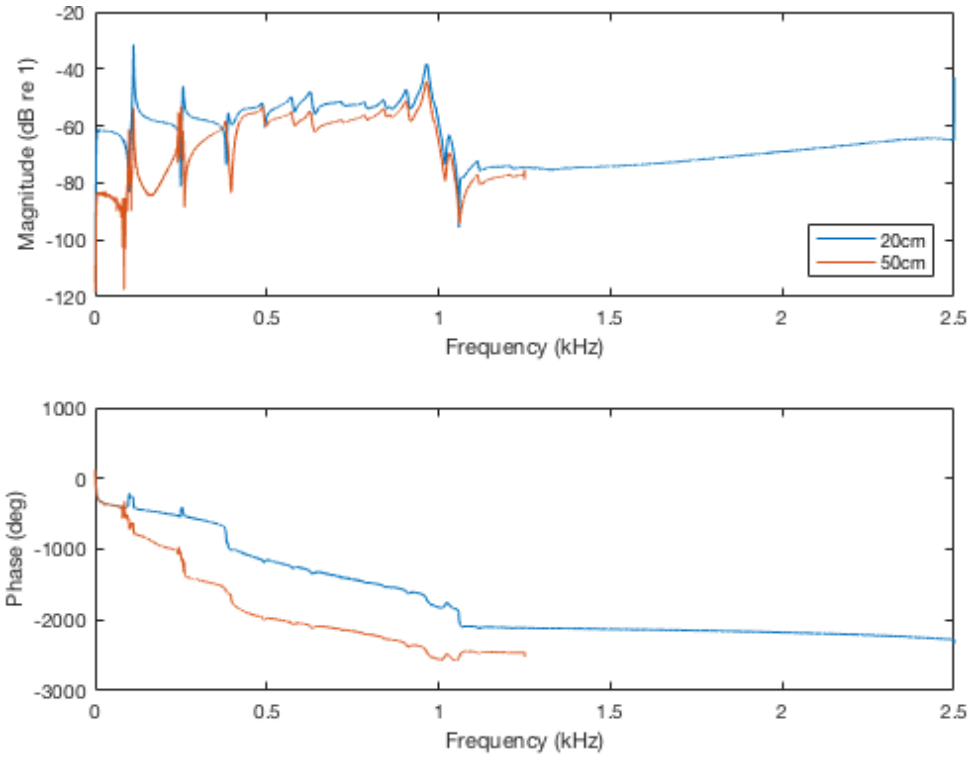


Figure 5.5: The measured transfer function of the noise compensated for ($H_{16\text{noise}}$) compared at 20 cm and 50 cm.

Constant	Value	Note
Characteristic impedance (R_C)	$50\,\Omega$	
Capacitance (C)	$93\,\text{nF km}^{-1}$	
Self inductance (L)	$232.5\,\mu\text{H km}^{-1}$	Calculated as $L = R_C^2 C$
Transmitter cable length (l_{tx})	304 cm	Measured
Receiver cable length (l_{rx})	304.5 cm	Measured

Table 5.1: Parameters for cables used.

($kd \approx 0.01$ and $kd \approx 0.026$ for the highest measured frequency of 2.5 MHz); and shows prominent near field effects, which means the data does not lend itself very well to reuse at different separation distances. In the current work the noise transfer function is therefore measured in sequence with the full system transfer function for which it is used for post-processing.

Figure 5.6 compares a signal with and without this post-processing along with the other techniques described above; and it can be seen that the technique lowers the coherent noise by about 11 dB.

Taking into account all the presented noise reduction methods a total of about 30 dB reduction of the coherent noise level was achieved.

5.2 Cable transfer functions (H_{12} and $H_{5\text{open}5}$)

Combining the measurements from section 4.3.1 with the transmission line model from sections 2.3.2 and 2.3.3 along with length measurements of the cables (done by hand with a ruler) and data-sheet values for the self inductance and capacitance of the cables given in table 5.1 [54], the transfer function for the cables used to connect the transducers to the rest of the measurement system can be calculated. These transfer functions can be seen in fig. 5.7.

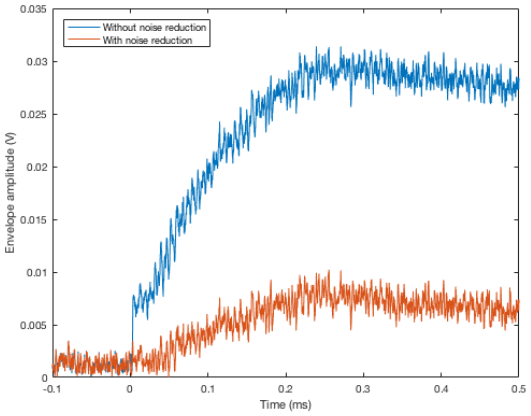


Figure 5.6: Envelope of coherent noise with and without post-processing applied.

It is apparent that the cable on the transitter side, characterised by H_{12}, does not contribute significantly to the transfer function of the full system, and could therefore be ignored without making the results inaccurate to any appreciable degree. It is however taken into account in the current work as there is no downsides to doing so.

$H_{5\text{open}5}$ does however have a significant contribution to the full system transfer in the vicinity of resonances of the receiving transducer; as can be seen from the figure. By inspecting the calculations it can be seen that this is mostly due to the correction for the voltage drop over the internal impedance of the receiving transducer from section 2.3.3, and as such a lumped model with the load in parallel with the cable's capacitance would suffice; given by

$$\begin{aligned} Z_S &= \left(\frac{1}{Z_L} + i\omega C l_{\text{rx}} \right)^{-1} \\ H_{5\text{open}5} &= \frac{Z_S}{Z_S + Z_I} \end{aligned} \quad . \tag{5.1}$$

Figure 5.8 shows the difference between the two models. The full transmission line model is again however used in the current work as there is no downsides to doing so.

5.3 Receiver transfer function (H_{56})

The transfer function used to correct for the effects of the receiver electronics is shown in fig. 5.9.

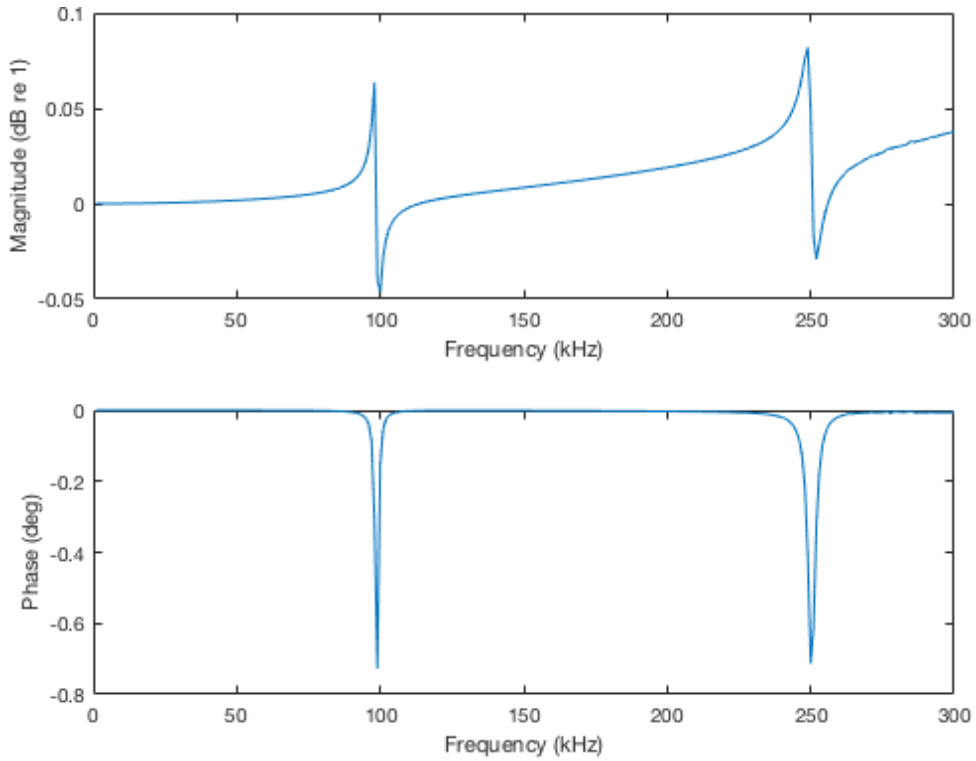
Since the receiver electronics only consist of an amplifier and a Butterworth filter their transfer function quite expectedly is only a gain with a mostly linear phase response, except for in the vicinity of the cutoff point, as seen in the figure.

The shown transfer function is used as is in the current work and its properties are not investigated further.

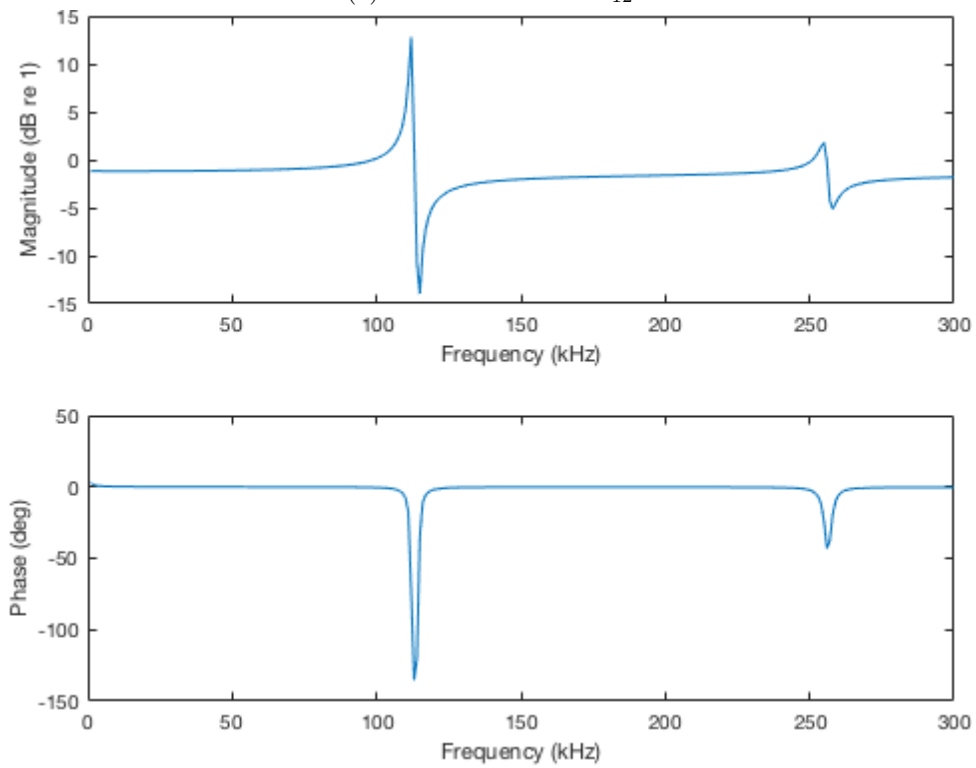
5.4 Acoustic transfer function ($H_{25\text{open}}$)

5.4.1 Nonlinear behaviour

Andersen previously investigated the nonlinearity around the first radial mode of the measured system [17, Chapter 6, pp. 96-97]; and while this seems to be the most dominant nonlinearity at longer separation distances, when measuring at shorter distances the second mode becomes significant.



(a) Transmitter side H_{12}



(b) Receiver side $H_{5\text{open}5}$

Figure 5.7: Transfer functions for the cables connecting the transducers to the rest of the measurement system.

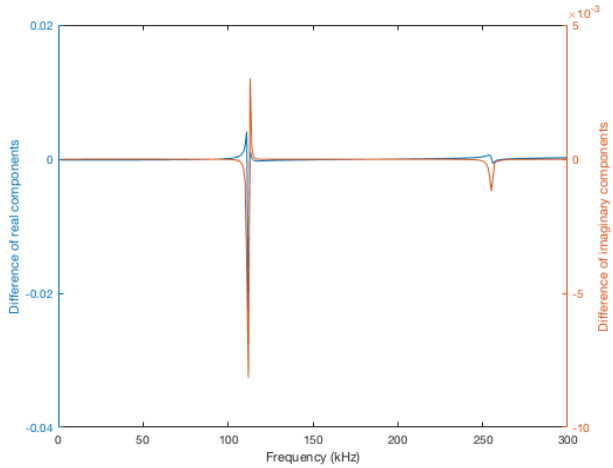


Figure 5.8: Difference of $H_{5\text{open}5}$ computed with transmission line and lumped model.

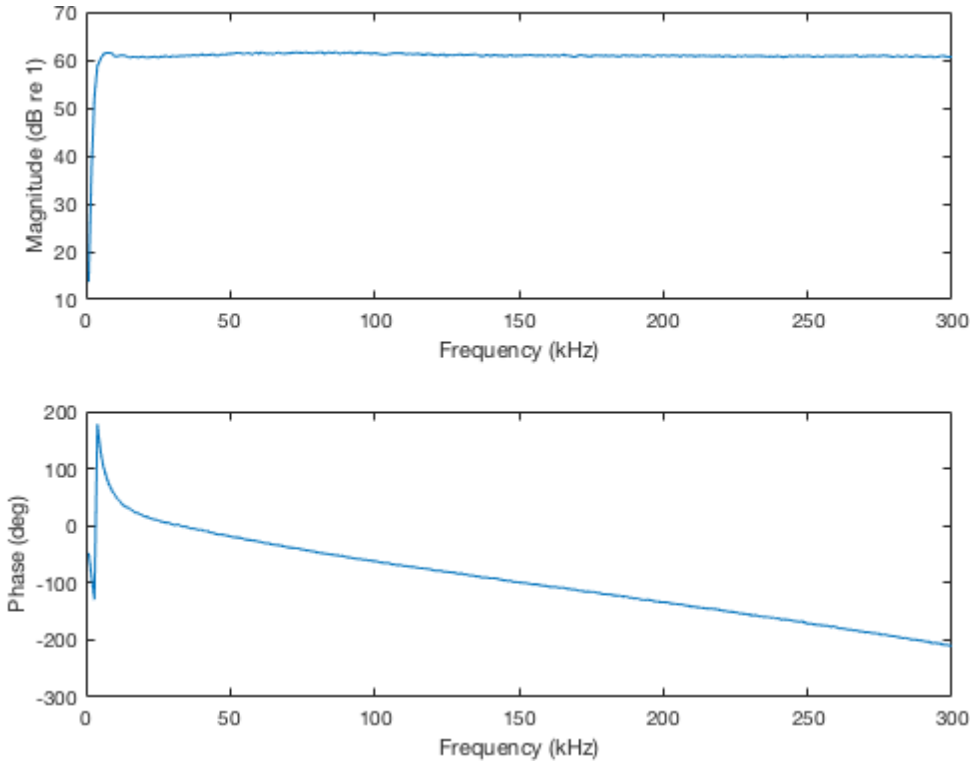


Figure 5.9: Transfer function for the receiver electronics.

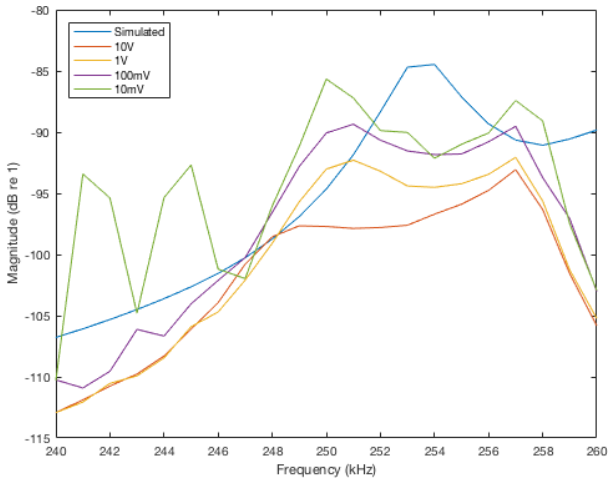


Figure 5.10

Figure 5.10 shows the magnitude of the simulated and measured transfer function close to the second radial mode, and it can be seen that when progressively lowering the excitation voltage of the transmitting transducer the transfer function converges towards the simulated transfer function. While the correspondence is quite good when the excitation voltage is 10 mV, measuring at this voltage is not practical as the signal barely breaks through the white noise level; which can be seen from the unstable transfer function in the interval leading up to the peak. Due to this high noise level the mentioned measurement at 10 mV might also be due to noise.

5.4.2 20cm

The simulated and measured transfer functions for the acoustic subsystem with a separation distance of 20 cm is shown in fig. 5.11. While there is not a perfect fit between the measured and simulated data most intervals have a relatively close correspondence.

As can be seen in the magnitude plot the simulated resonances generally occur at slightly higher frequencies than the measured ones, which might be due to inaccurate material parameters yielding a too high speed of sound or too large values for the transducer dimensions; both of which makes the value ka grow slower.

In addition to this the measured magnitude seems to deviate down from the simulated magnitude as frequency increases, possibly indicating that there is some attenuation not taken into account or interference with other signals (e.g. reflections) partially cancelling the measured waveforms. To exclude the interference hypothesis the period between subsequent bursts were slowly increased well beyond the point where no previous reflections could be seen on the oscilloscope before the trigger for emitted bursts. While this excludes reflections from old bursts interfering with newer bursts it does not exclude reflections from objects close to the transducers reflecting the newer bursts in such a way that the reflection reaches the receiver before the burst ends. Of special concern here are the mount for the transducers which are located close enough for this same burst reflection to happen. Previous master students have shaped these poles to be triangular to divert the reflections away from the receiving transducers, but it is not known if this is sufficient. This issue is not investigated further in the current work.

Looking at the phase plot in the figure one can see that the measured phase seems to fall slightly faster than the simulated phase, which probably is due to imprecise Fourier window placements when calculating the phase. Figure 5.12 shows the measured phase with an additional linear contribution calibrated in the interval 150 kHz to 225 kHz to make it match the simulated phase. The found coefficients of adjustment are on the order of $0.2\,^\circ\,\mathrm{kHz}^{-1}$ to $0.6\,^\circ\,\mathrm{kHz}^{-1}$, which

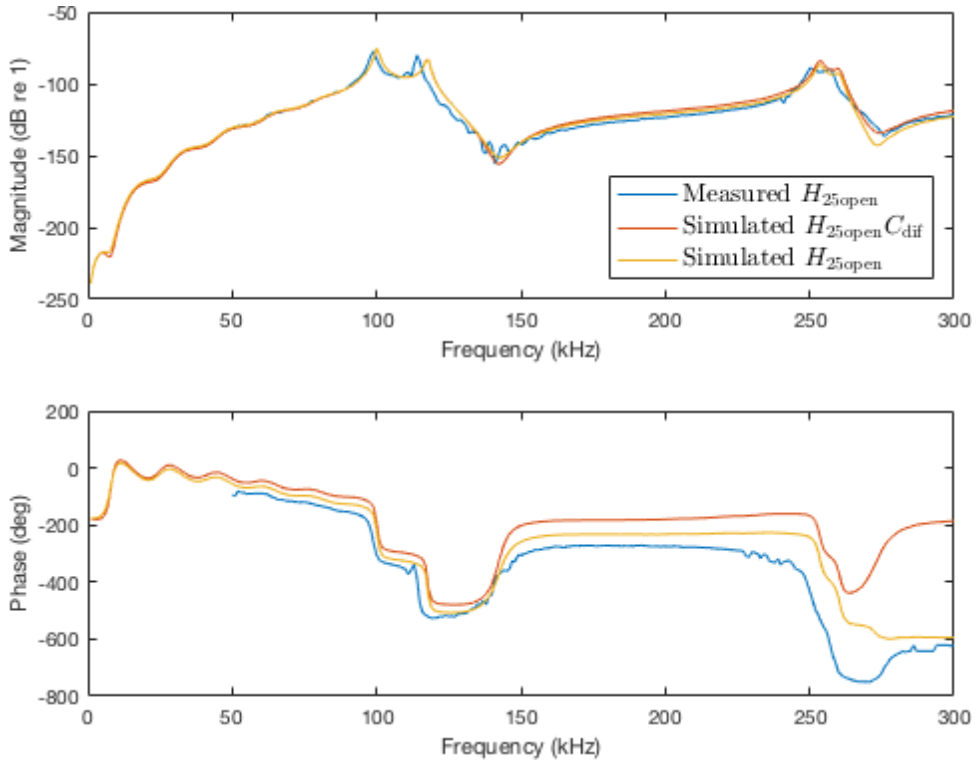


Figure 5.11: Measured and simulated (with and without diffraction correction) transfer function for the acoustic subsystem at a separation distance of 20 cm.

can be explained by an error in the propagation time (d/c) on the order of 1 µs. This error is well within the margin of the measurement of d ($\sim \pm 10$ µm) and the margin of error of the model used for c (>7000 ppm, almost exclusively due to imprecise temperature measurements; ± 0.01 °C [17, Chapter 3, p. 38]).

An extra dip in the phase not predicted by the simulation model can be seen in measured data in the range 250 kHz to 275 kHz (right after the second radial mode). It is not known wether this is a real effect or an artefact of the measurement procedure. If it is an artefact it seems to be local to the range as the phase converges back to the simulated one above 275 kHz, barring the falling phase discrepancy discussed above. In this higher region of 275 kHz to 300 kHz the simulation model with diffraction predicts a rather noticeable deviation of more than 360° from the diffractionless model which seemingly also shows up on the measured data.

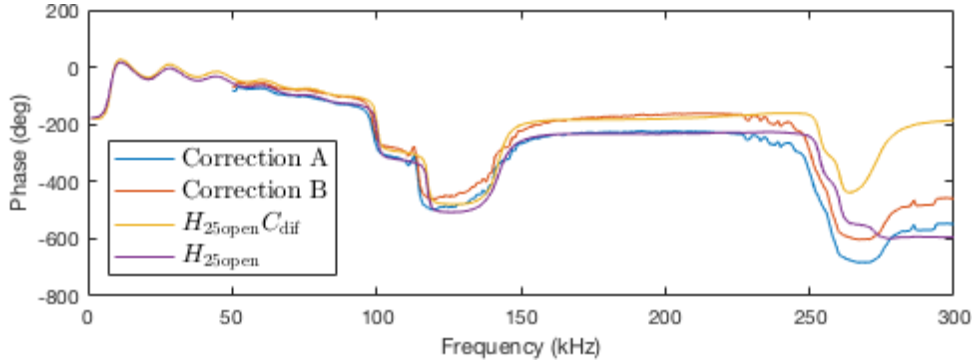


Figure 5.12: Phase of $H_{25\text{open}}$ at a separation distance of 20 cm corrected with $0.25\,^\circ\,\text{kHz}^{-1}$ (correction A) and $0.55\,^\circ\,\text{kHz}^{-1}$ (correction B) less phase lag along with simulated phases with and without diffraction correction.

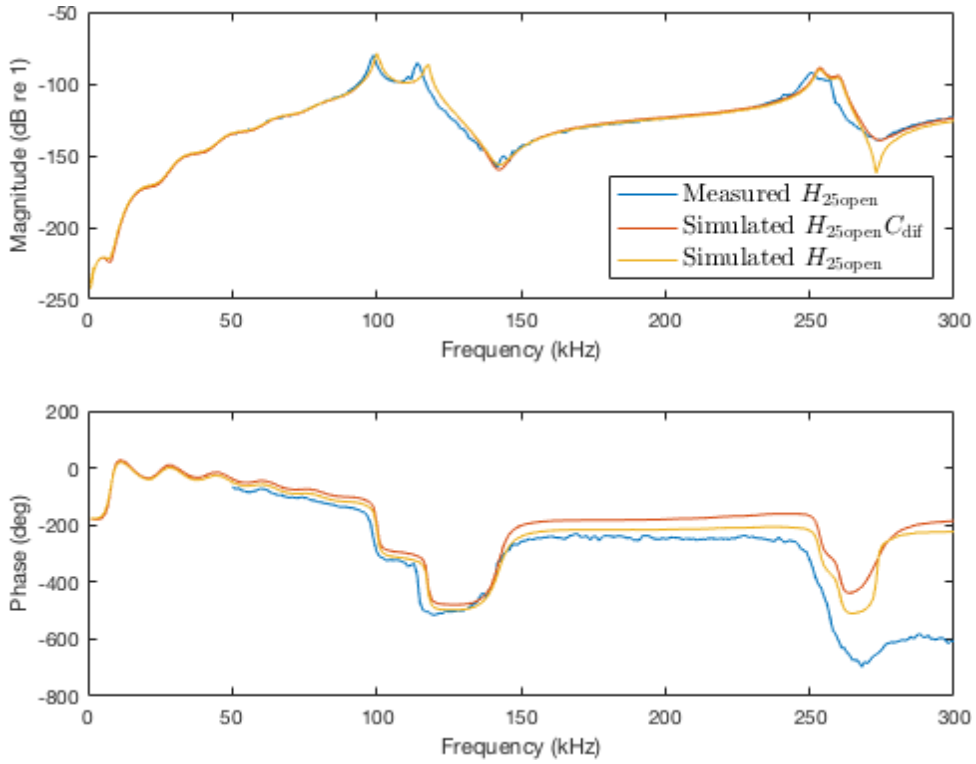


Figure 5.13: Measured and simulated (with and without diffraction correction) transfer function for the acoustic subsystem at a separation distance of 30 cm.

5.4.3 30cm

Figure 5.13 shows the simulated and measured transfer functions for the acoustic subsystem at a separation distance of 30 cm; and one of the notable features is again the discrepancies in the range 250 kHz to 275 kHz. At this separation distance there is also a discrepancy between the simulated magnitude with diffraction effects included and the measured data; which in the range instead seemingly follows the diffractionless curve.

Also of note is that the ∼360° phase shift seen in both the measured and simulated phase with diffraction effects at 20 cm separation distance is here only present in the measured curve.

It is of interest that 30 cm is approximately the Rayleigh distance for the transducers used when operating at a frequency of 300 kHz, and this transition from near-field to far-field might be implicated in the disparity between the simulated and measured curves; but this is not investigated in the current work.

5.4.4 50cm

A separation distance 50 cm was also used in the current work for comparison with Andersen [17]; who measured at this distance and longer.

Figure 5.14 shows the transer function of the acoustical subsystem simulated and measured at this distance; which shows good correspondence with each other, barring the problems of inaccurate material parameters and phase drift discussed before.

There is seemingly also good correspondence with the data presented by Andersen [17, Chapter 6, pp. 97-102] except for a few features seen in some of his data as the phase rises after each resonant mode; but as the transducers measured are not the same it is not known what causes these discrepancies. This is not investigated any further in the current work.

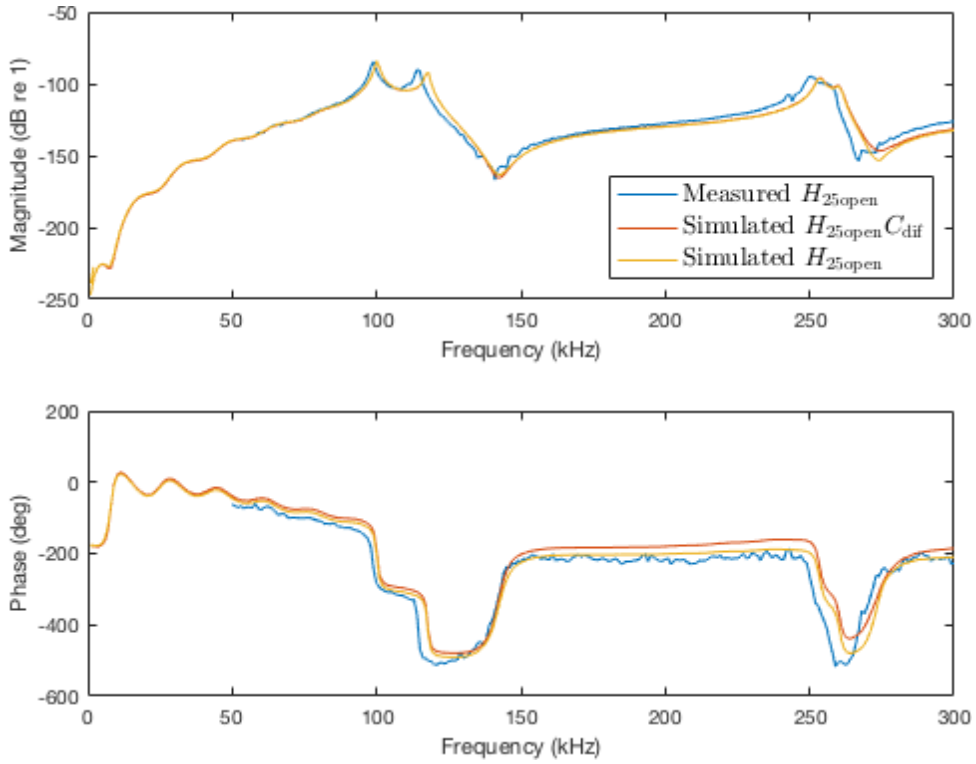


Figure 5.14: Measured and simulated (with and without diffraction correction) transfer function for the acoustic subsystem at a separation distance of 50 cm.

5.5 COMSOL/FEMP comparison

Since previous works at the University of Bergen on the topic have used an internally developed finite element analysis software, FEMP, it is of interest to compare the simulations used in the current work with simulations from this software. Simulations presented in this section were computed with FEMP version 6.0.0, as described in appendix A.

5.5.1 Transducer properties

As shown in fig. 5.15 both the COMSOL simulations used in the current work and simulations done with FEMP yield the same admittance to the point where the curves are seemingly overlapping and the COMSOL curve can't be properly seen.

Figure 5.16 compares the sensitivity of the transducer as computed by COMSOL and FEMP, and discrepancies can be seen in both the magnitude and phase. As will be discussed in the next section the magnitude disparity might be caused by problems with FEMP.

The observed phase difference can be explained by a difference in propagation distance of about $\sim$10 µm; and since the mesh elements used are of size $\sim$100 µm for this frequency region in FEMP, and slightly finer in COMSOL (by a factor of ~ 3, since in COMSOL the same mesh is used for all frequencies) this difference in phase is probably explainable in terms of mesh resolution.

5.5.2 Transfer function

As seen in fig. 5.17 the same discrepancies seen in the source sensitivity of the transducer unsurprisingly also show up in the transfer function of the system. Figure 5.18 shows the difference in magnitude of the transfer function as computed by COMSOL and FEMP which resembles the

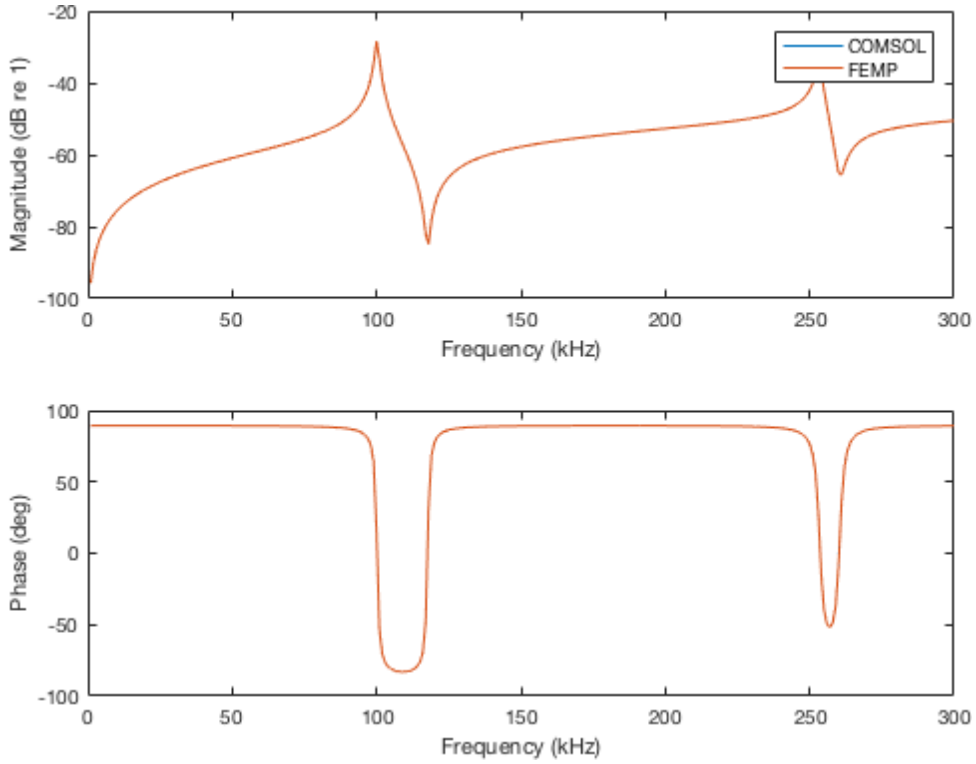


Figure 5.15: Comparison of admittance as computed by COMSOL and FEMP.

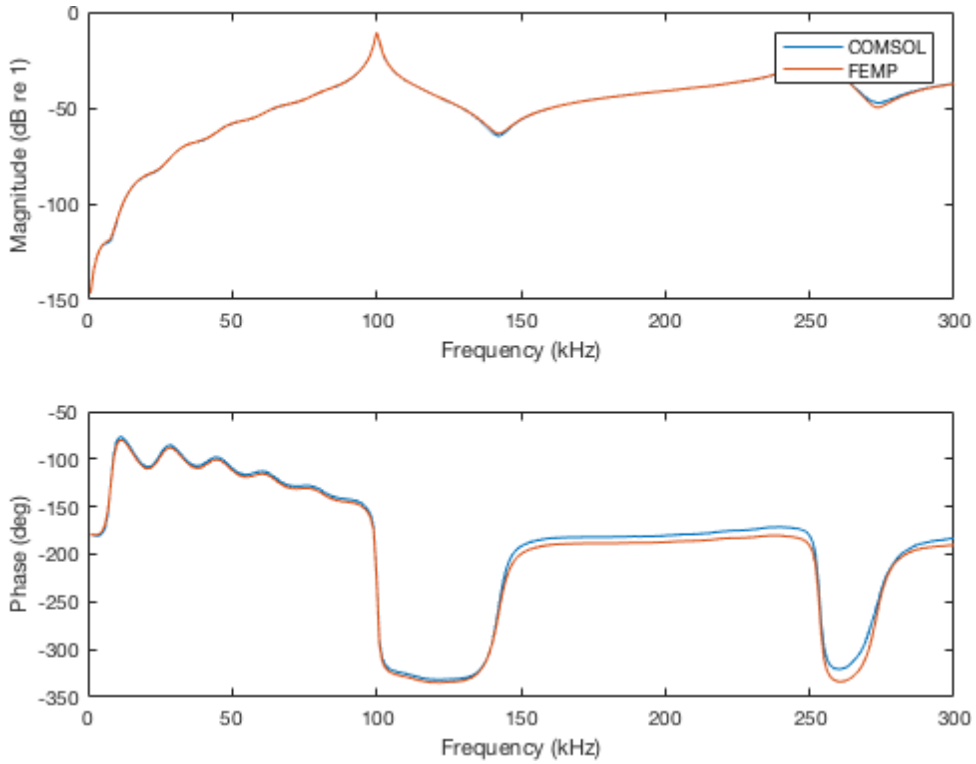


Figure 5.16: Comparison of source sensitivity as computed by COMSOL and FEMP.

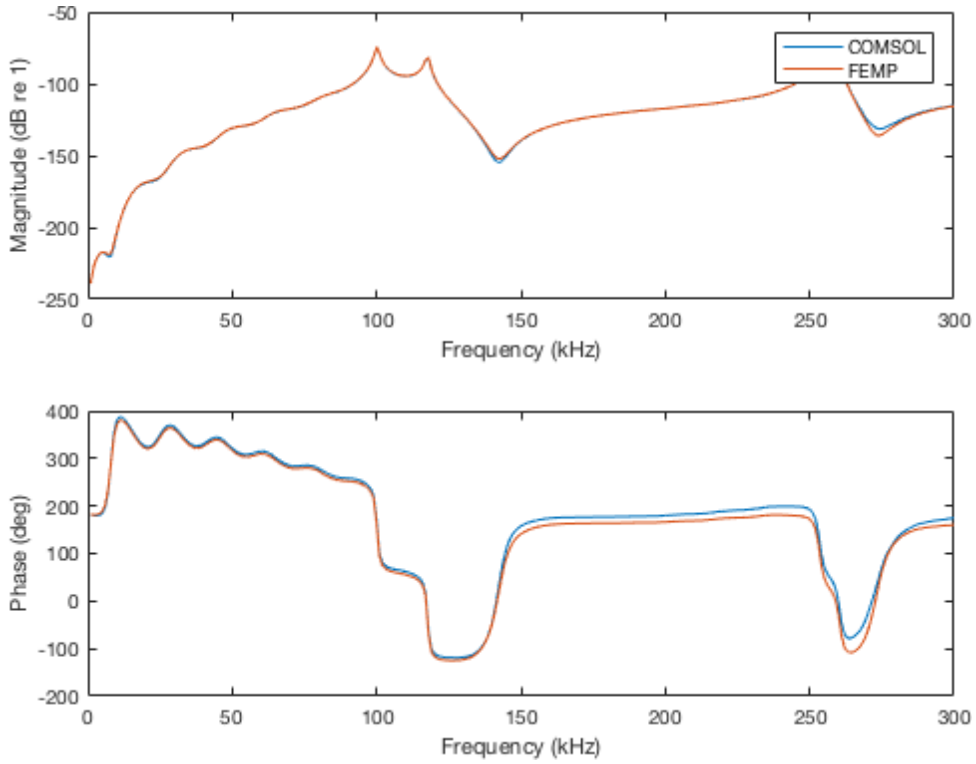


Figure 5.17: Comparison of the transfer function $H_{25\text{open}}C_{\text{dif}}$ as computed by COMSOL and FEMP.

shape of the correction term C_{dif} for a simplified finite element diffraction correction; which can be seen in Figure 5.19.

Assuming diffraction was present in $H_{25\text{open}}C_{\text{dif}}$ as computed by FEMP but not as computed by COMSOL, one would expect the difference between the two to be C_{dif} as

$$\overbrace{20\log_{10}(H_{25\text{open}}C_{\text{dif}})}^{\text{COMSOL}} - \overbrace{20\log_{10}(H_{25\text{open}}C_{\text{dif}}/C_{\text{dif}})}^{\text{FEMP}} = 20\log_{10}(C_{\text{dif}})\,. \tag{5.2}$$

Assuming the diffraction is present in COMSOL but not FEMP, both, or none yields a difference of $20\log_{10}(1/C_{\text{dif}})$, 0, and 0 respectively. Comparing with Andersen [17, Chapter 6, p. 95, fig. 6.20] the observed difference between the two transfer functions corresponds with C_{dif} for a separation distance of 50 cm.

The current author hypothesises that de observed discrepancy is due to to the extrapola-

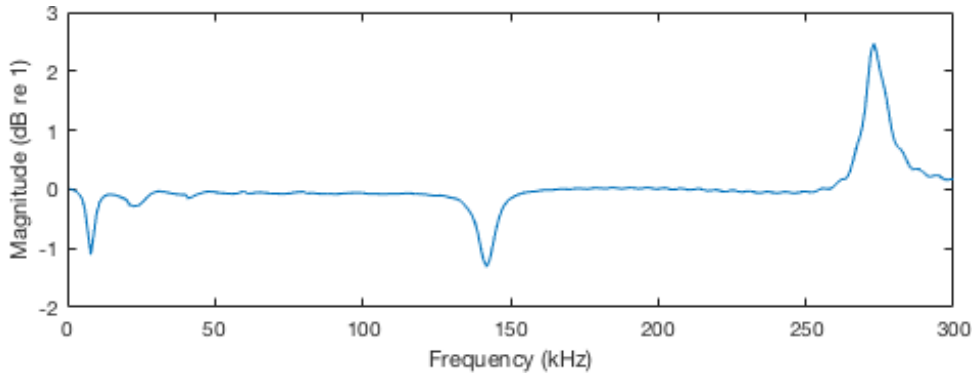


Figure 5.18: Difference between transfer function $H_{25\text{open}}C_{\text{dif}}$ as computed by COMSOL and FEMP (COMSOL minus FEMP).

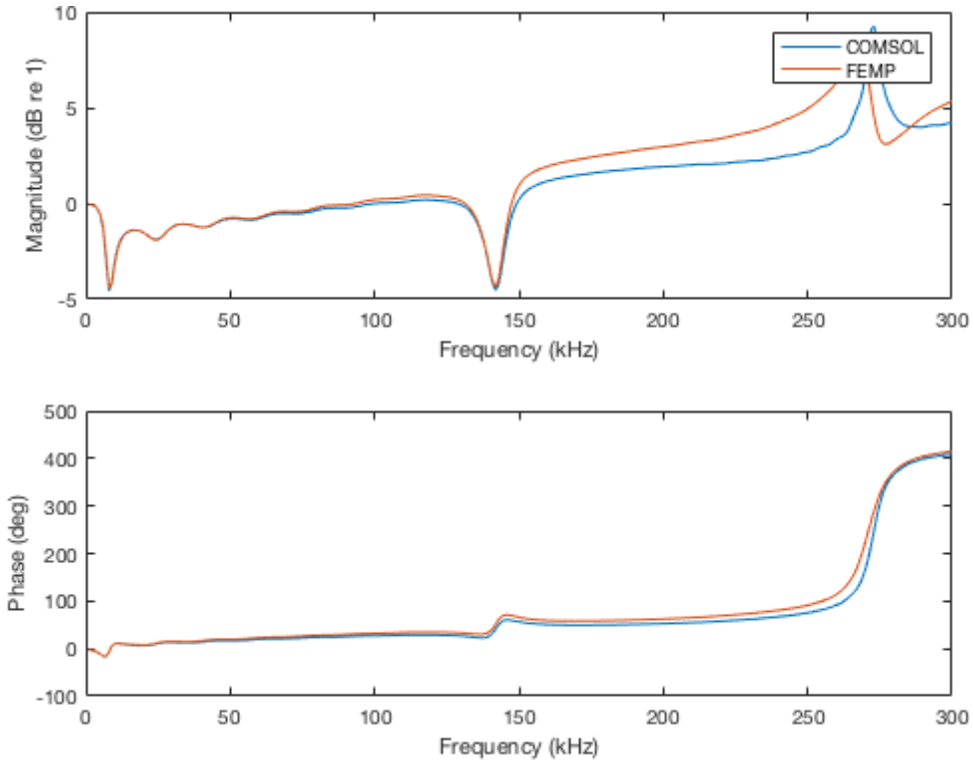


Figure 5.19: Comparison of diffraction correction as computed by COMSOL and FEMP.

tion method used in FEMP for calculating the far-field pressure for the sensitivity as FEMP propagates the solution on the outer rim of the simulation domain towards infinity by means of interpolating inside Asltey-Leis infinite elements. The current author is concerned that this might propagate near-field effects present on this rim to the far-field points where the pressure for the source sensitivity is sampled, since Asltey-Leis elements essentially interpolate several different plane waves (12 as implemented in FEMP) where the innermost 3 are fitted to the pressure values present on the outer rim of the simulation domain [19, Chapter 3, pp. 53-55], and this might not be enough to properly resolve the transition from near- to far-field. This hypothesis is not verified in the current work due to time constraints.

To test the hypothesis and possibly work around the problems described above one could try working with a larger simulation domain so that the outer rim of the simulation domain contains less near-field effects, but as this takes a lot of computational time with the simulation parameters used in the current work this is not investigated further. A better tradeoff between resolution, domain shape, and computational time could remedy these problems and bring the computational time down to plausible time scales, but this is not investigated in the current work as FEMP is not used as a primary tool for simulations.

5.5.3 Diffraction correction

Figure 5.19 shows the correction factor for simplified finite element diffraction correction as computed by COMSOL and FEMP, and again a discrepancy can be seen in the magnitude of the two. Here the discrepancy stems from the computed average pressure over the would be receiver, which can be seen in fig. 5.20.

Similar to the problem described in the previous section; this problem might be related to interpolation in the Astley-Leis infinite elements as this is how the average pressure over the would be receiver is computed in FEMP, however it is not certain it is related to diffraction as the effect spans a wider frequency range than the effects explored in the previous section. This problem is not investigated further in the current work due to the previously stated constraints.

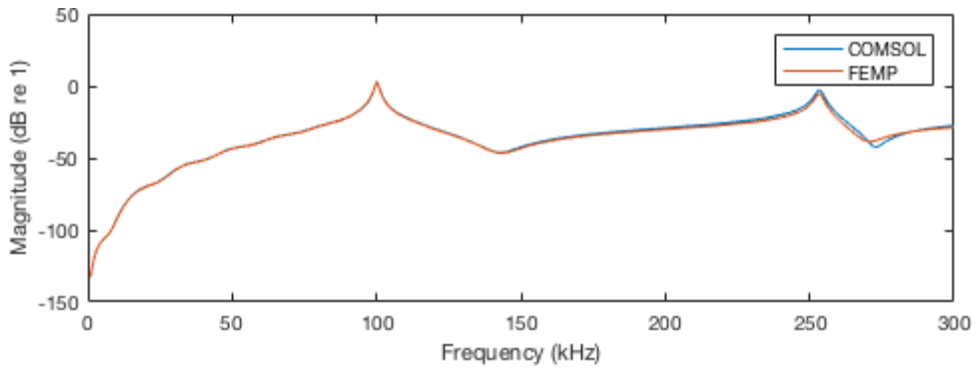


Figure 5.20: Comparison of average pressure at receiver as computed by COMSOL and FEMP.

Chapter 6

Discussion

6.1 Validity of the results

As the discrepancies pointed out in the previous chapter might call into question the validity of the presented data a discussion of this topic is warranted.

The most glaring discrepancies are probably the ones between the simulations done with COMSOL and FEMP, calling into question the extrapolation techniques used calculate pressure outside the finite element domain in one, or both, of the tools.

As the presented difference between the two from section 5.5.2 hints at the possibility of diffraction effects being included in the source sensitivity calculations from FEMP it seems more likely than not that FEMP is at least partially to blame for the observed discrepancies; or at least is prone to produce erroneous simulations given non-optimal, but seemingly valid, user input. Since some of the observed discrepancies seemingly relate to diffraction it is possible that a larger finite element domain would mitigate the problems; but if the problems are caused by the limited resolution in the Astley-Leis infinite elements as proposed by the current author the concern would possibly need be addressed in all simulation done with FEMP as the computed solution would only converge to the true solution in the limit where the domain size is infinite (although the effects might be virtually invisible with quite small domains as the solution might converge quickly).

The second discrepancy between the FEMP and COMSOL simulations presented in section 5.5.3, although not investigated as much by the current author, also seemingly stems from problems with FEMP as the diffraction correction (C_{dif}) computed with COMSOL seems to agree with the diffraction correction presented by Andersen [17, Chapter 6, p. 95, fig. 6.20] while the diffraction correction computed with FEMP does not. To the author's best knowledge Andersen's data was obtained with FEMP, meaning FEMP is probably capable of producing the correct results, however maybe in a way that eluded the current author's brief experience with it. After a brief correspondence with Kocbach [19] (the original developer of FEMP) with additional input from Storheim [13] it seems to the current author to be some confusion to which version(s) of FEMP in which this capability is fully functional, and it is possible that the version compared in the current work has a faulty or partial implementation.

Since the rest of the simulations presented corresponded well with previous results by Andersen [17], and the above discrepancies seems to be due to problems with FEMP, it seems plausible that the simulations done with COMSOL, which are used in the current work, are accurate; however as the COMSOL documentation [18] doesn't provide implementation details on all the features used feature tests should be implemented to verify each component of the simulations carried out in the current work. Of these, the COMSOL manual specifically raises the concern of the stability of the Helmholtz-Kirchhoff method of extrapolating the pressure field outside the finite element domain for long extrapolation distances [18, Evaluating the Acoustic Field in the Far-Field Region]. While this has not specifically been investigated in the current

work it seemingly hasn't been a problem due to the correspondence with Andersen's results, and as such the distances used in the current work probably don't fall under the category of "long"; but for feature testing this is probably the most obvious candidate.

For the separation distance of 50 cm the presented measurement data seems to be in good agreement with the presented simulation data, and to a slightly lesser extent in agreement with Andersen's data for the separation distance [17, Chapter 6, pp. 97-102]. This lends credibility to the measurement procedure, including the newly introduced noise reduction technique; and as such it is plausible that the data measured at shorter distances are also accurate; however as measurement uncertainties are not calculated it is not possible to draw any definitive conclusions based on the data. This should be improved in subsequent works, so that conclusions hinted by the current work can be confirmed.

The discrepancies seen between the measured and simulated transfer function $H_{25\text{open}}$ around the second radial mode at a separation distance of 30 cm, and to a lesser extent at 20 cm, is an unresolved issue in the current work; which could hint at problems with the diffraction correction used in the simulations, or problems with the measurement procedure at these short separation distances.

As discussed previously the remaining discrepancies between the measurements and simulations are probably due to inaccurate material constants used in the simulations and the inherent difficulty of measuring the slowly varying phase accurately. Although the discrepancies are present in the data they are easily explained and corrected for, and as such does not pose a limitation when testing the used diffraction correction.

6.2 Interpretation of the unexplained discrepancies

While no hard conclusions can be drawn as the data lacks associated error estimates, the aforementioned discrepancies around the second radial mode might hint at problems with the tested diffraction correction.

As the simplified finite element diffraction correction is a free-field diffraction correction, it does not take into account effects related to the interaction between the pressure field and the receiving transducer; and as such the discrepancies seen in the magnitude of $H_{25\text{open}}$ at a separation distance of 30 cm might be due to local dips in pressure at the receiver which would be present in the equivalent free-field, but is suppressed when the field interacts with the receiving transducer. These suppressions might occur due to the wast difference in specific acoustic impedance between the medium ($\sim$400 Rayl) and the transducer ($\sim$30 MRayl); making the transducer appear like a hard wall boundary to incoming waves. Such a hard wall boundary is described approximately by the boundary condition

$$\frac{\partial p}{\partial n} = 0\,; \tag{6.1}$$

which eliminates sudden dips along the normal axis of the receiving transducer in the transducers vicinity as it requires the pressure to converge to a stable value as the wave approaches the transducer.

The observed discrepancies in phase are not explained as easily as the magnitude discrepancies. It is however possible that the same mechanism might be partly or wholly responsible as the largest discrepancy, the missing 360° phase shift after the second radial mode at a separation distance of 30 cm, coincides with the magnitude discrepancy described above. It is also possible that this is related to the phase dip seen after the second radial mode at a separation distance of 20 cm and 30 cm which is missing from the simulated transfer functions, however the author has no clear cut hypothesis for this proposed relationship.

6.3 Conclusions

As mentioned before the lack of error estimates prohibits drawing any definite conclusions from the presented data in the current work; yet the provided data seems to hint at problems with the simplified finite element diffraction correction as mentioned in the previous section.

Besides conclusions about the stated subject matter, the current work effectively demonstrates the effectiveness of the techniques used.

As the previous works have manly dealt with the FEMP simulation software, while the current work instead utilises COMSOL MULTIPHYSICS as the main simulation software, the current work effectively demonstrates the fitness of COMSOL for the problem at hand.

Since the two softwares features different approaches to dealing with unbounded domains, which causes the method of extrapolating pressure values outside the finite element domain to differ, the current work also demonstrates that the Hemholtz-Kirchhoff expression for field values outside the finite element domain is apt for the task.

While the presented comparison between COMSOL and FEMP seems to show some issues with FEMP, particularly when extrapolating outside the simulated finite element domain, it does not conclusively demonstrate anything as the current author can not exclude user error as a possible reason for the discrepancies. However as the simulated transfer function based on the reciprocity model seems to include diffraction effects when simulated with FEMP the current work does hint at possibly fundamental problems with the extrapolation method used in the software as discussed earlier.

The current work does however demonstrate the effectiveness of the newly introduced method of dealing with coherent noise, which in section 5.1.3 was shown to significantly decrease this type of noise. As the analysis in the current work was limited to just a single frequency it doesn't conclusively demonstrate the effectiveness of the technique for all frequencies, but the current author sees no reason why the results would not generalise to a wider frequency range. The current work also demonstrates the effectiveness of aluminium foil to reduce the coherent noise, although as the coherent noise is electromagnetic in nature this should not come as a surprise.

6.4 Suggestions for future work

As one of the biggest weaknesses with the current work is the lack of error estimations future work should include this in order to verify the conclusions hinted to in this work. To optimise these error estimates to the smallest possible intervals better material data for the transducers and a more precise method of estimating the propagation time of the signal is also needed if the future works approach the diffraction correction verification problem in the same manner as the current work.

Future works could also explore the possibility of improving and extending the coherent noise reduction method introduced in the current work to also compensate for reflections; which would allow for measurements at much shorter separation distances along with longer excitation bursts, which in turn might eliminate the need to estimate propagation time to compensate for the plane wave component, and yield more precise measurements due to the longer bursts. This could be done by treating each reflection as a coherent noise source that turns on after a specific delay. The spectra of these sources could possibly be measured by measuring the system with long bursts, which when a full set of discrete Fourier frequencies have been measured could be fitted and gated in the time domain to yield spectra of different sources.

An alternate route to further explore the simplified finite element diffraction correction is to simulate the whole acoustic system (transmitter, medium, and receiver) by means of finite element analysis or equivalent techniques, comparing the results, since these full system simulations would include effects like the suppression of local minima proposed in section 6.2. Since the full system simulation should be equivalent to the physical system in the low excitation (linear)

limit this could possibly verify and explain the discrepancies observed in the current work.

When mentioning improvements to simulations, it would also be interesting to know what causes the discrepancies between COMSOL and FEMP, particularly the discrepancies shown in the average pressure over the receiver in the current work. However as COMSOL seems to correctly simulate the problem at hand this might not be the best allocation of the limited research time associated with such works. Instead time should probably be invested in the feature testing of COMSOL discussed earlier to verify that the results produced are accurate.

Bibliography

[1] T. C. Hebb, "The velocity of sound", Physical Review (Series I) **20**, 89–99 (1905) DOI: 10.1103/PhysRevSeriesI.20.89.

[2] C. D. Reid, "Some investigations into the velocity of sound at ultra-sonic frequencies using quartz oscillators", Physical Review **35**, 814–831 (1930) DOI: 10.1103/PhysRev.35.814.

[3] H. B. Huntington, A. G. Emslie, and V. W. Hughes, "Ultrasonic delay lines. I.", Journal of the Franklin Institute **245**, 1–23 (1948) DOI: 10.1016/0016-0032(48)90826-6.

[4] M. Born and E. Wolf, *Principles of Optics*, 6th edition (Pergamon Press, 1980).

[5] E. Lommel, "Beugungserscheinungen einer kreisrunden Öffnung und geradlinig begrenzter Schirme", Abhandlungen der Bayerischen Akademie der Wissenchaften **15**, 229–328 (1886).

[6] P. H. Rogers and A. L. V. Buren, "An exact expression for the Lommel-diffraction correction integral", The Journal of the Acoustical Society of America **55**, 724–728 (1974) DOI: 10.1121/1.1914589.

[7] A. O. Williams Jr, "The piston source at high frequencies", The Journal of the Acoustical Society of America **23**, 1–6 (1951) DOI: 10.1121/1.1906722.

[8] L. V. King, "On the acoustic radiation field of the piezo-electric oscillator and the effect of viscosity on transmission", Canadian Journal of Research **11**, 135–155 (1934) DOI: 10.1139/cjr34-080.

[9] A. S. Khimunin, "Numerical calculation of the diffraction corrections for the precise measurement of ultrasound absorption", Acta Acustica united with Acustica **27**, 173–181 (1972).

[10] A. S. Khimunin, "Numerical calculation of the diffraction corrections for the precise measurement of ultrasound phase velocity", Acta Acustica united with Acustica **32**, 192–200 (1975).

[11] G. A. Sabin, "Calibration of piston transducers at marginal test distances", The Journal of the Acoustical Society of America **36**, 168–173 (1964) DOI: 10.1121/1.1918929.

[12] P. Lunde, K. E. Frøysa, R. Kippersund, and M Vestrheim, "Transient diffraction effects in ultrasonic meters for volumetric, mass and energy flow measurement of natural gas", in Proceedings of the 21st North Sea Flow Measurement Workshop (2003).

[13] E. Storheim, "Diffraction effects in the ultrasonic field of transmitting and receiving circular piezoceramic disks in radial mode vibration. FE modelling and comparison with measurements in air", PhD thesis (The University of Bergen, 2015).

[14] E. Mosland, "Reciprocity calibration method for ultrasonic piezoelectric transducers in air", MA thesis (University of Bergen, 2013), HDL: 1956/7164.

[15] R. Hauge, "Finite element modeling of ultrasound measurement systems for gas. Comparison with experiments in air.", MA thesis (University of Bergen, 2013), HDL: 1956/7182.

[16] A. A. Søvik, "Ultrasonic measurement systems for gas. Finite element modelling compared with measurements in air", MA thesis (University of Bergen, 2015).

[17] K. K. Andersen, "Reciprocity calibration of ultrasonic piezoelectric disks in air ", MA thesis (University of Bergen, 2015), HDL: 1956/11006.

[18] *COMSOL Multiphysics version 4.2a Manual*, COMSOL AB (COMSOL AB).

[19] J. Kocback, "Finite Element Modeling of Ultrasonic Piezoelectric Transducers", PhD thesis (University of Bergen, 2000).

[20] L. E. Kinsler, A. R. Frey, A. B. Coppens, and J. V. Sanders, *Fundamentals of Acoustics*, 4th edition (John Wiley & Sons, 2000).

[21] P. Lunde, "Lecture notes, PHYS374 Theoretical Acoustics".

[22] C. Canuto and A. Tabacco, *Mathematical Analysis II*, 2nd edition (Springer International Publishing, 2015).

[23] "IEEE Standard on Piezoelectricity", ANSI/IEEE Std 176-1987 (1988) DOI: 10.1109/IEEESTD.1988.79638.

[24] A. Boggess and F. J. Narcowich, *A First Course in Wavelets with Fourier Analysis* (John Wiley & Sons, 2009).

[25] B. Douglas, *The Fundamentals of Control Theory*, Revision 1.5 (2017).

[26] R. A. Adams and C. Essex, *Calculus: A Complete Course*, 7th edition (Pearson Education, 2009).

[27] M. Vestrheim, "Lecture notes, PHYS373 Acoustic measurement systems".

[28] R. W. P. King, H. R. Mimno, and A. H. Wing, *Transmission Lines, Antennas and Waveguides*, 2nd edition (Dover Publications, 1965).

[29] M. Vestrheim, "Lecture notes, PHYS272 Acoustic transducers".

[30] J. P. Bentley, *Principles of Measurement Systems*, 4th edition (Pearson Education, 2005).

[31] R. J. Bobber, "General reciprocity parameter", The Journal of the Acoustical Society of America **39**, 680–687 (1966) DOI: 10.1121/1.1909941.

[32] L. L. Foldy and H. Primakoff, "A general theory of passive linear electroacoustic transducers and the electroacoustic reciprocity theorem. I", The Journal of the Acoustical Society of America **17**, 109–120 (1945) DOI: 10.1121/1.1916305.

[33] P. Lunde, "Modeller for beskrivelse av elektroakustisk sender-mottaker mÃělesystem, inkl. diffraksjonseffekter / diffraksjonkorreksjon", Jan. 2013.

[34] G. S. K. Wong, "The variation of the specific heat ratio and the speed of sound in air with temperature, pressure, humidity, and CO2 concentration", Journal of the Acoustical Society of Japan (E) **11**, 145–155 (1990) DOI: 10.1250/ast.11.145.

[35] O. Cramer, "The variation of the specific heat ratio and the speed of sound in air with temperature, pressure, humidity, and CO2 concentration", The Journal of the Acoustical Society of America **93**, 2510–2516 (1993) DOI: 10.1121/1.405827.

[36] M. O'Donnell, E. T. Jaynes, and J. G. Miller, "General relationships between ultrasonic attenuation and dispersion", The Journal of the Acoustical Society of America **63**, 1935–1937 (1978) DOI: 10.1121/1.381902.

[37] M. O'Donnell, E. T. Jaynes, and J. G. Miller, "Kramers–Kronig relationship between ultrasonic attenuation and phase velocity", The Journal of the Acoustical Society of America **69**, 696–701 (1981) DOI: 10.1121/1.385566.

[38] C. L. Morfey and G. P. Howell, "Speed of sound in air as a function of frequency and humidity", The Journal of the Acoustical Society of America **68**, 1525–1527 (1980) DOI: 10.1121/1.385080.

[39] "Method for calculation of the absorption of sound by the atmosphere", ANSI/ASA S1.26-1995 (1995).

[40] H. E. Bass, H.-J. Bauer, and L. B. Evans, "Atmospheric absorption of sound: Analytical expressions", The Journal of the Acoustical Society of America **52**, 821–825 (1972) DOI: 10.1121/1.1913183.

[41] J. W. Strutt, *The Theory of Sound*, 2nd edition, Vol. 2 (Macmillan and Co, 1896).

[42] O. C. Zienkiewicz, R. L. Taylor, and J. Z. Zhu, *The Finite Element Method*, 6th edition, Vol. 1 (Butterworth-Heinemann, 2005).

[43] *COMSOL Multiphysics version 4.4 Reference Manual*, COMSOL AB (COMSOL AB).

[44] A. D. Pierce, *Acoustics: An Introduction to Its Physical Principles and Applications* (Acoustical Society of America, 1989).

[45] M. Aanes, "Interaction of piezoelectric transducer excited ultrasonic pulsed beams with a fluid-embedded viscoelastic plate. Finite element modeling, angular spectrum modeling and measurements", PhD thesis (The University of Bergen, 2014), HDL: 1956/7891.

[46] S. G. Johnson, "Notes on Perfectly Matched Layers (PMLs)", Mar. 2010.

[47] S. Marburg, "Six boundary elements per wavelength: is that enough?", Journal of Computational Acoustics **10**, 25–51 (2002) DOI: 10.1142/S0218396X02001401.

[48] B. Ghose, K. Balasubramaniam, C. V. Krishnamurthy, and A. S. Rao, "Two dimensional FEM simulation of ultrasonic wave propagation in isotropic Solid Media using COMSOL", in COMSOL Conference (2010).

[49] C. G. Petra, O. Schenk, M. Lubin, and K. Gärtner, "An augmented incomplete factorization approach for computing the Schur complement in stochastic optimization", SIAM Journal on Scientific Computing **36**, C139–C162 (2014) DOI: 10.1137/130908737.

[50] C. G. Petra, O. Schenk, and M. Anitescu, "Real-time stochastic optimization of complex energy systems on high-performance computers", IEEE Computing in Science & Engineering **16**, 32–42 (2014) DOI: 10.1109/MCSE.2014.53.

[51] M. C. Nechyba, "Lecture notes, EEL3135 Discrete-Time Signals and Systems".

[52] D. Kundur, "Lecture notes, ECE316 Communication Systems".

[53] *DPO3000 Series Digital Phosphor Oscilloscope User Manual*, Tektronix (Tektronix).

[54] *DATA SHEET RG 178 B/U*, 0019/0894, Lapp Group (Lapp Group, June 2008).

Appendix A

FEMP simulations

A.1 Proposed FEMP Python API

While trying to run simulations in FEMP the current author ran into problems where FEMP would get terminated by the system for exhausting its memory pool. To remedy this situation a way of controlling FEMP from Python is proposed by the current autor below; which alleviates the problem by allowing for segmented runs with cleaned up MATLAB workspace in-between. This new way of controlling FEMP also yields an easier and cleaner scriptable interface as a side-effect.

Adding in the file "__init__.py" (whose contents are given in the code listing below) into the FEMP root directory, and adding FEMP to the Python path, lets one import the specified `run_solver` function to control FEMP.

The proposed implementation relies on either the MATLAB Python API, or the `pexpect` package, one of which needs to be on the Python path in order for the current implementation to work. If the MATLAB Python API is present the current implementation will simply issue commands to an embedded MATLAB process via this API to control FEMP, whereas if the MATLAB Python API is not present the current implementation will use the `pexpect` package to spawn a MATLAB REPL in which it issues commands to FEMP as if a human were typing them at the MATLAB prompt. This dual implementation is done as the MATLAB Python API is not kept current, leading to newer distributions of Python not being supported by it.

For a full Python conversion of FEMP the API is made to return the results of simulations as NumPy arrays. This is supported through the h5py package as version 7.3 of the MATLAB file format used is really just HDF5 files, and as such can be read by a standard HDF5 implementation.

A few places the f-string notation introduced in Python 3.6 is used in the current implementation (all of which could easily be converted to the `.format()` syntax for backward compatibility), rendering the current implementation to require Python 3.6 or newer. As Python 3.6 is not currently supported by the MATLAB Python API (but probably will be in MATLAB R2017b, which releases shortly after publication of the current work), only the `pexpect` based fallback implementation is currently usable.

```
1  """Python wrapping layer for FEMP.
2
3  This module contains functionality to control the FEMP solver through Python.
4  It relies upon the MATLAB Python API, but provides its own polyfill
5  based on pseudo-terminals (pexpect) if the API is unavailable.
6  """
7
8  __author__ = 'Andreas Hagen'
```

```
__version__ = '1.1'

import os
import sys
from contextlib import suppress
from pathlib import Path
from shutil import move
from tempfile import TemporaryDirectory, NamedTemporaryFile

from h5py import File

# Import MATLAB API or polyfill the necessary functions if unavailable
try:
    from matlab.engine import start_matlab
except ImportError:
    from collections import namedtuple

    from pexpect import spawn, ExceptionPexpect
    from pexpect.replwrap import REPLWrapper

    def start_matlab():
        """Spawns a MATLAB instance controlled with the returned object's eval function through a pty."""
        try:
            child = spawn(os.getenv('MATLAB', 'matlab'), ['-nodesktop'], encoding='utf-8', timeout=None)
        except ExceptionPexpect:
            raise EnvironmentError('"$MATLAB" did not contain a valid MATLAB launch command')

        child.logfile_read = sys.stdout
        repl = REPLWrapper(child, '>> ', None)

        MATLABEnginePolyfill = namedtuple('MATLABEnginePolyfill', ['eval'])
        return MATLABEnginePolyfill(lambda command, **_: repl.run_command(command))


def run_solver(config, save=None, env_setup=lambda pwd: None, lib=Path(__file__).resolve().parent, engine=None):
    """Runs the FEMP solver with specified configuration.

    :param dict config: Dictionary of FEMP .inn directives.
    :param str save: Path to results file to save
    :param callable env_setup: Callback function to set up environment for FEMP, i.e. copy in material files.
    :param str lib: Path to the FEMP library. Defaults to the directory the current
    module is located in as this module is usually placed alongside FEMP.
    :param engine: MATLAB engine in which to run FEMP.

```

```
    :return FEMPResults: Sanitized version of the results generated by FEMP.
    """
    # Start up or retrieve cached MATLAB engine if one is not provided
    if engine is None:
        try:
            engine = run_solver.engine
        except AttributeError:
            engine = run_solver.engine = start_matlab()

    with TemporaryDirectory() as pwd:
        env_setup(str(pwd))  # Run environment setup
        engine.eval(cd_command(str(pwd)))  # Switch MATLAB to correct
↪   directory

        # Generate config file
        with NamedTemporaryFile(mode='w', dir=pwd, suffix='.inn',
↪   delete=False) as file:
            conf_file = Path(file.name)
            print(inn_file(config), file=file)

        # Setup results file
        if save is None:
            save, move_save_file = Path(pwd, f'{conf_file.stem}_results.mat'),
↪   True
        else:
            save, move_save_file = Path(save), False

        # Run FEMP simulation
        engine.eval(femp_command(lib, conf_file.name, save.name), nargout=0)
        return FEMPResults(save, move_save_file)


def cd_command(directory):
    """Generates MATLAB code to change current working directory.

    :param str directory: Path of the directory to change to.
    :return str: MATLAB command to run to change the current working
↪   directory.
    """
    directory_escaped = directory.replace("'", "''")
    return f"cd('{directory_escaped}');"


def femp_command(prefix, infile, outfile):
    """Generates MATLAB code to load and execute FEMP.

    :param str prefix: Path to the FEMP library. Should generally be the same
        directory as the current file is placed in.
    :param str infile: Name of .inn file to pass along to FEMP. Must be on
        the MATLAB path or given as an absolute path.
```

```
    :param str outfile: Name of the .mat file FEMP should store its results in.
    :return str: MATLAB command to run to start FEMP.
    """
    return f"restoredefaultpath; clear; " \
           f"addpath('{prefix}', genpath('{prefix}/solver'), '{prefix}/postprocess'); " \
           f"donotsave = true; fnavn = '{infile}'; fempsolver; save('{outfile}', 'result', '-v7.3');"


def inn_file(config):
    """Generates .inn file contents for use with the FEMP solver.

    :param dict config: Dictionary (possibly ordered) of FEMP .inn directives.
        Values might be strings or numbers, in which case they are used as
        directive values as is; lists, in which case they are joined by commas;
        or dictionaries, in which case entries are put on separate lines with
        keys and values separated by a comma.
    :return str: Generated .inn file contents.
    """
    def entry(values):
        if isinstance(values, str):
            # Strings will be interpreted as lists a couple of lines down if not handled separately
            return values

        with suppress(AttributeError):
            # Yields key/value pairs for use in the 'set' directive
            return '\n'.join(f'{variable},{value}' for variable, value in values.items())

        with suppress(TypeError):
            # Joins argument lists separated by comma
            return ','.join(str(value) for value in values)

        # Defaults to pass arguments as is, assuming it's a single value
        return str(values)

    # Joins directives separated by two line breaks
    return '\n\n'.join(f'{directive}\n{entry(values)}\nend' for directive, values in config.items())


class FEMPResults:
    """Contains a FEMP result dataset.

    Data is accessed as attributes on the returned object, e.g. the admittance
    data would be accessed as obj.admittance.

```

```
    :param Path results_file: Path to the results file to access.
    :param bool manage_file: Specifies whether the object should manage the
    ↪ results file.
    """
    def __init__(self, results_file, manage_file=False, recover_complex=True):
        if manage_file:
            self._pwd = TemporaryDirectory()  # Directory is kept as long as a
    ↪ reference to it is kept
            move(str(results_file), self._pwd.name)
            results_file = Path(self._pwd.name, results_file.name)

        self.recover_complex = recover_complex
        self._results_file = File(str(results_file))

    def __getattr__(self, item):
        try:
            # MATLAB stores references in the main datasets, so the real
    ↪ datasets have to be looked up
            data = self._results_file[self._results_file[f'/result/{item}'][0,
    ↪ 0]][...]
        except KeyError:
            raise AttributeError(f"'{self.__class__.__name__}' object has no
    ↪ attribute {item}")

        with suppress(TypeError):
            if self.recover_complex and {'real',
    ↪ 'imag'}.issubset(data.dtype.fields):
                data = data['real'] + data['imag']*1j

        return data
```

A.2 Running FEMP simulations

In the current work the proposed Python API for FEMP discussed in the previous section is used to run FEMP simulations for comparison with the COMSOL Multiphysics simulations actually used. To run these simulations, avoiding the previously mentioned memory problems, FEMP is simply executed repeatedly with a dictionary of configuration directives as shown in the below code listing.

```
from collections import ChainMap
from contextlib import closing
from functools import partial
from itertools import chain
from shutil import copy2

from numpy import newaxis, dtype, exp, pi, absolute, linspace, array_split,
    ↪ tile, fromiter, float64, complex128
from tables import open_file

from FEMP import run_solver as femp

```

```
# Calculate frequency ranges
f_start, f_stop, f_step = 1e3, 3e5, 1e3
frequencies = linspace(f_start, f_stop, round((f_stop - f_start) / f_step) + 1)
ranges = map(lambda lst: (lst[0], lst[-1]), array_split(frequencies, 15))

# Calculate separation distances
d_start, d_stop, d_step = .2, .5, .1
distances = linspace(d_start, d_stop, round((d_stop - d_start) / d_step) + 1)

# Constants
domain_size = 30.01e-3
elements_per_wavelength = 10
disc_radius = 10e-3
disc_depth = 2e-3
disc_material = 1
fluid_material = 2
reference_distance = 1
far_field_distance = 1e3

# Set up common config
config_defaults = {
    'meshingtype': ('elementsperwavelength', 'f_stop'),
    'order': 2,
    'infiniteorder': 12,
    'materialfile': 1,
    'piezodiskfluid': (disc_radius, disc_depth, elements_per_wavelength, elements_per_wavelength, disc_material,
                       domain_size, elements_per_wavelength, fluid_material, 1.3),
    'directharmonicanalysis': ('f_start', 'f_step', 'f_stop', 'complex_loss'),
    'admittance': (0, 0, 0),
    'sensitivity': (0, 0, 0, reference_distance),
    'pressureatreceiverdistance': (0, 0, 0, disc_radius, 0, d_start, d_stop, d_step),
    'onaxispressure': (0, 0, 0, far_field_distance / domain_size, 1),
    'save': (
        'admittance_f',
        'admittance',
        'sensitivity',
        'pressureatreceiverdistance_p',
        'onaxispressure',
        'onaxispressure_r'
    )
}


def tabulate(results):
    # Transfer data and define nice variable names
    f = results.admittance_f
    r0 = tile(reference_distance, f.shape)
```

```
    a = tile(disc_radius, f.shape)
    rho = tile(results.rho_medium, f.shape)
    c = tile(results.c_medium, f.shape)
    Y = results.admittance
    SV = results.sensitivity
    p = results.pressureatreceiverdistance_p.T

    # Pick the on-axis node closest to the far field distance as the far field pressure
    r = results.onaxispressure_r.squeeze()
    idx = absolute(r - far_field_distance).argmin()
    dff = tile(r[idx], f.shape) - disc_depth/2
    pff = results.onaxispressure[:, idx, newaxis]

    # Build data table
    rows = (tuple(chain.from_iterable(x)) for x in zip(r0, a, rho, f, c, Y, SV, p, dff, pff))
    columns = [(var, float64) for var in ('r0', 'a', 'rho', 'f', 'c')] + \
              [(var, complex128) for var in ('Y', 'SV')] + \
              [(f'p{int(distance * 100)}cm', complex128) for distance in distances] + \
              [('dff', float64), ('pff', complex128)]
    return fromiter(rows, dtype=columns)


def post_process(file, start=None, stop=None):
    # Helper to lazy initialize tables
    def table(distance):
        name = f'd{int(distance * 100)}cm'
        return getattr(file.root, name) if hasattr(file.root, name) else file.create_table(
            file.root, name,
            dtype([(var, float64 if var is 'f' else complex128)
                   for var in ('f', 'Y', 'SV', 'SVslow', 'H25open', 'H25openslow', 'Hdif', 'Cdif')])
        )

    # Calculate and save results
    for r0, a, rho, f, c, Y, SV, *p, dff, pff in file.root.data.read(start, stop):
        k = 2*pi*f/c
        SVslow = SV * exp(1j*k*r0)

        for d, p in zip(distances, p):
            Jsph = -2j*r0/f/rho * exp(1j*k*r0)
            H25open = SV**2 / Y * Jsph * r0/d * exp(-1j*k*(d - r0))
            H25openslow = H25open * exp(1j*k*d)

            Hdif = p/pff * .5j*k*a**2/dff * exp(1j*k*(d - dff))
            Cdif = .5j*k*a**2/d / Hdif

```

```
            table(d).append([(f, Y, SV, SVslow, H25open, H25openslow, Hdif,
↪   Cdif)])


with closing(open_file('results.h5', mode='w')) as file:
    for lower, upper in ranges:
        # Run FEMP simulation
        config = ChainMap({'set': {
            'f_start': lower,
            'f_stop': upper,
            'f_step': f_step
        }}, config_defaults)
        data = tabulate(femp(config, env_setup=partial(copy2,
↪   'material1.dat')))

        # Post-process and save data
        try:
            file.root.data.append(data)
        except AttributeError:
            file.create_table(file.root, 'data', data)
        post_process(file, len(file.root.data) - len(data),
↪   len(file.root.data))
        file.flush()
```

Also used is the below material file. It is simply given in the standard FEMP material format. For more information on this see the FEMP documentation.

```
         1 piezo pz27
# mechanical terms
1.20250e+11 7.62000e+10 7.42000e+10 0.00000e+00 0.00000e+00 0.00000e+00
7.62000e+10 1.20250e+11 7.42000e+10 0.00000e+00 0.00000e+00 0.00000e+00
7.42000e+10 7.42000e+10 1.10050e+11 0.00000e+00 0.00000e+00 0.00000e+00
0.00000e+00 0.00000e+00 0.00000e+00 2.11000e+10 0.00000e+00 0.00000e+00
0.00000e+00 0.00000e+00 0.00000e+00 0.00000e+00 2.11000e+10 0.00000e+00
0.00000e+00 0.00000e+00 0.00000e+00 0.00000e+00 0.00000e+00 2.16000e+10
# coupling terms
0.00000e+00 0.00000e+00 0.00000e+00 0.00000e+00 1.12000e+01 0.00000e+00
0.00000e+00 0.00000e+00 0.00000e+00 1.12000e+01 0.00000e+00 0.00000e+00
-5.40000e+00 -5.40000e+00 1.70000e+01 0.00000e+00 0.00000e+00 0.00000e+00
# dielectric terms
8.11044e-09 0.00000e+00 0.00000e+00
0.00000e+00 8.11044e-09 0.00000e+00
0.00000e+00 0.00000e+00 8.14585e-09
# density and damping coefficients
7.70000e+03 9.99000e+02 9.99000e+02
# mechanical Q-factors
9.60000e+01 7.00000e+01 1.20000e+02 0.00000e+00 0.00000e+00 0.00000e+00
7.00000e+01 9.60000e+01 1.20000e+02 0.00000e+00 0.00000e+00 0.00000e+00
1.20000e+02 1.20000e+02 1.90000e+02 0.00000e+00 0.00000e+00 0.00000e+00
0.00000e+00 0.00000e+00 0.00000e+00 7.50000e+01 0.00000e+00 0.00000e+00
0.00000e+00 0.00000e+00 0.00000e+00 0.00000e+00 7.50000e+01 0.00000e+00
```

```
0.00000e+00 0.00000e+00 0.00000e+00 0.00000e+00 0.00000e+00 3.15000e+02
# piezoelectric Q-factors
0.00000e+00 0.00000e+00 0.00000e+00 0.00000e+00 -2.00000e+02 0.00000e+00
0.00000e+00 0.00000e+00 0.00000e+00 -2.00000e+02 0.00000e+00 0.00000e+00
-1.66000e+02 -1.66000e+02 -3.24000e+02 0.00000e+00 0.00000e+00 0.00000e+00
# dielectric Q-factors
5.00000e+01 0.00000e+00 0.00000e+00
0.00000e+00 5.00000e+01 0.00000e+00
0.00000e+00 0.00000e+00 1.30000e+02
# end of material data
         2 fluid air t=24degC pA=1atm h=.25 f=150kHz
1.18817e+00 1.42441e+05 0.00000e+00 0.00000e+00
```

Appendix B

labctrl

B.1 Source code

B.1.1 acoustic.py

```
from contextlib import closing

from numpy import linspace, newaxis
from tables import open_file

from instruments import Oscilloscope, WaveformGenerator, Thermometer,
↪  Hygrometer
from spec.data import setup_variables, frequency_variable,
↪  environment_variables, waveform_variables
from spec.measurements import measure_waveforms

# Connect to instruments
oscilloscope = Oscilloscope('USB0::0x699::0x0410::C010246::INSTR')
waveform_generator = WaveformGenerator('GPIB0::10::INSTR')
thermometer = Thermometer('GPIB0::3::INSTR')
hygrometer = Hygrometer('ASRL4::INSTR')

# Measurement parameters
distance = .2
propagation_time = distance / 347  # Underestimate
burst_length = .8 * propagation_time
measurement_count = 300 - 49
sample_count = 10000
frequencies = linspace(start=50e3, stop=3e5, num=measurement_count)

with closing(open_file('D:\\andreas\\datanew\\acoustic.h5', mode='w')) as
↪  file:
    # Setup output
    setup_variables(file, distance=distance, burst_length=burst_length)
    f = frequency_variable(file, expected_rows=measurement_count)
    t, p, h, = environment_variables(file, expected_rows=measurement_count)
    ttx, htx, trx, hrx = waveform_variables(file, sample_count=sample_count,
↪  expected_rows=measurement_count)

```

```
    for frequency in frequencies:
        try:
            timestamp, amplitude = measure_waveforms(
                frequency=frequency,
                burst_length=burst_length,
                propagation_time=propagation_time,
                voltage=.1 if 240e3 < frequency < 260e3 else 1 if 80e3 <
↪   frequency < 120e3 else 10,
                oscilloscope=oscilloscope,
                waveform_generator=waveform_generator
            )
        except ValueError:
            continue

        # Frequency
        f.append((frequency,))

        # Waveforms
        ttx.append(timestamp['tx'][newaxis])
        htx.append(amplitude['tx'][newaxis])
        trx.append(timestamp['rx'][newaxis])
        hrx.append(amplitude['rx'][newaxis])

        # Environmental parameters
        t.append((thermometer.temperature,))
        p.append((101325.,))
        h.append((hygrometer.relative_humidity / 100,))
```

B.1.2 admittance.py

```
from contextlib import closing

from numpy import linspace
from tables import open_file

from instruments.impedanceanalyzer import ImpedanceAnalyzer
from spec.data import frequency_variable, admittance_variable
from spec.measurements import measure_admittance

# Connect to instruments
impedance_analyzer = ImpedanceAnalyzer('GPIB0::17::INSTR')

# Measurement parameters
measurement_count = 300
frequencies = linspace(start=1e3, stop=3e5, num=measurement_count)


with closing(open_file('D:\\andreas\\datanew\\admittance.h5', mode='w')) as
↪   file:
    # Setup output
    f = frequency_variable(file, expected_rows=measurement_count)
```

```
    Y = admittance_variable(file, expected_rows=measurement_count)

    for frequency in frequencies:
        f.append((frequency,))
        Y.append((measure_admittance(frequency, impedance_analyzer),))
```

B.1.3 noise.py

```
from contextlib import closing

from numpy import linspace, newaxis
from tables import open_file

from instruments import Oscilloscope, WaveformGenerator
from spec.data import setup_variables, frequency_variable, waveform_variables
from spec.measurements import measure_waveforms

# Connect to instruments
oscilloscope = Oscilloscope('USB0::0x699::0x0410::C010246::INSTR')
waveform_generator = WaveformGenerator('GPIB0::10::INSTR')

# Measurement parameters
distance = .2
burst_length = 1.6e-3
measurement_count = 5001
sample_count = 10000
frequencies = linspace(start=0, stop=2.5e6, num=measurement_count)

with closing(open_file('D:\\andreas\\datanew\\noise.h5', mode='w')) as file:
    # Setup output
    setup_variables(file, distance=distance, burst_length=burst_length)
    f = frequency_variable(file, expected_rows=measurement_count)
    ttx, htx, trx, hrx = waveform_variables(file, sample_count=sample_count,
    ↪  expected_rows=measurement_count)

    for frequency in frequencies:
        try:
            timestamp, amplitude = measure_waveforms(
                frequency=frequency,
                burst_length=burst_length,
                propagation_time=0,
                voltage=1 if 80e3 < frequency < 120e3 else 10,
                oscilloscope=oscilloscope,
                waveform_generator=waveform_generator
            )
        except ValueError:
            continue

        # Frequency
        f.append((frequency,))

```

```
        # Waveforms
        ttx.append(timestamp['tx'][newaxis])
        htx.append(amplitude['tx'][newaxis])
        trx.append(timestamp['rx'][newaxis])
        hrx.append(amplitude['rx'][newaxis])
```

B.1.4 receiver.py

```
from contextlib import closing

from numpy import linspace, newaxis
from tables import open_file

from instruments import Oscilloscope, WaveformGenerator
from spec.data import setup_variables, frequency_variable, waveform_variables
from spec.measurements import measure_waveforms

# Connect to instruments
oscilloscope = Oscilloscope('USB0::0x699::0x0410::C010246::INSTR')
waveform_generator = WaveformGenerator('GPIB0::10::INSTR')

# Measurement parameters
burst_length = 1.6e-3
measurement_count = 300
sample_count = 10000
frequencies = linspace(start=1e3, stop=3e5, num=measurement_count)

with closing(open_file('D:\\andreas\\datanew\\receiver.h5', mode='w')) as
↪   file:
    # Setup output
    setup_variables(file, burst_length=burst_length)
    f = frequency_variable(file, expected_rows=measurement_count)
    ttx, htx, trx, hrx = waveform_variables(file, sample_count=sample_count,
↪   expected_rows=measurement_count)

    for frequency in frequencies:
        try:
            timestamp, amplitude = measure_waveforms(
                frequency=frequency,
                burst_length=burst_length,
                propagation_time=0,
                voltage=10e-3,
                oscilloscope=oscilloscope,
                waveform_generator=waveform_generator
            )
        except ValueError:
            continue

        # Frequency
        f.append((frequency,))

```

```
        # Waveforms
        ttx.append(timestamp['tx'][newaxis])
        htx.append(amplitude['tx'][newaxis])
        trx.append(timestamp['rx'][newaxis])
        hrx.append(amplitude['rx'][newaxis])
```

B.1.5 instruments/__init__.py

```
from .base import Instrument, Property, SCPIInstrument

from .oscilloscope import Oscilloscope
from .waveformgenerator import WaveformGenerator
from .thermometer import Thermometer
from .hygrometer import Hygrometer
from .filter import Filter
```

B.1.6 instruments/base.py

```
from functools import lru_cache, partialmethod, partial

from pyvisa import ResourceManager


@lru_cache()  # Lazy load
def _rm():
    return ResourceManager()


class Instrument:
    def __init__(self, resource, set_message='{} {}', query_message='{}?'):
        if isinstance(resource, str):
            resource = _rm().open_resource(resource)

        self.resource = resource
        self.set_message = set_message
        self.query_message = query_message

    def __repr__(self):
        return f'{self.__class__.__name__}({self.resource!r})'

    def _io(self, func, *args, **kwargs):
        return getattr(self.resource, func)(*args, **kwargs)

    read = partialmethod(_io, 'read')
    read_raw = partialmethod(_io, 'read_raw')
    write = partialmethod(_io, 'write')
    query = partialmethod(_io, 'query')
    query_values = partialmethod(_io, 'query_values')
    query_ascii_values = partialmethod(_io, 'query_ascii_values')
    query_binary_values = partialmethod(_io, 'query_binary_values')
```

```
33
34      def __getattribute__(self, item):
35          attr = super().__getattribute__(item)
36
37          if isinstance(attr, Property):
38              message =
    ↪   self.query_message.format(self.attribute_to_property(item))
39              query_function = getattr(self, attr.query_function)
40              return attr.parse(query_function(message))
41          else:
42              return attr
43
44      def __setattr__(self, item, value):
45          try:
46              attr = super().__getattribute__(item)
47
48              if isinstance(attr, Property):
49                  message =
    ↪   self.set_message.format(self.attribute_to_property(item),
    ↪   attr.format(value))
50                  self.write(message)
51                  return
52          except AttributeError:
53              pass
54
55          return super().__setattr__(item, value)
56
57      def attribute_to_property(self, attr):
58          return attr
59
60
61  class Property:
62      _formaters = {}
63
64      def __init__(self, type_=str, query_function='query', formater=str,
    ↪   **kwargs):
65          self.type = type_
66          self.formater = formater
67          self.query_function = query_function
68
69          if (1 if formater is not str else 0) + len(kwargs) > 1:
70              raise ValueError('Incorrect number of formaters specified.')
71
72          for formater, param in kwargs.items():
73              if formater in self._formaters:
74                  setattr(self, formater, param)
75                  self.formater = partial(self._formaters[formater], param)
76              else:
77                  raise ValueError(f'Unknown formater {formater}.')
78
79      def format(self, value):
```

```
        return self.formater(self.type(value))

    def parse(self, value):
        return self.type(value)

    @classmethod
    def add_formater(cls, caster, method):
        cls._formaters[caster] = method


def propertyformater(caster):
    return lambda method: Property.add_formater(caster, method)


@propertyformater('choices')
def choice_formater(choices, choice):
    if choice not in choices:
        raise ValueError(f'Invalid option {choice}..')

    return choice


@propertyformater('quotes')
def quoted_formater(quotes, string):
    return f'{quotes}{string}{quotes}'


@propertyformater('prefix')
def prefix_formater(prefix, string):
    return f'{prefix}{string}'


@propertyformater('postfix')
def postfix_formater(postfix, string):
    return f'{string}{postfix}'


class SCPIInstrument(Instrument):
    namespace_separator = ':'

    def attribute_to_property(self, attr):
        return f':{attr.replace("_", self.namespace_separator)}'
```

B.1.7 instruments/filter.py

```
from instruments import Instrument, Property


class KrohnHite3940(Instrument):
    CH = Property(int, prefix='1.')
    T = Property(int)
```

```
    M = Property(int)
    F = Property(float)

    def __init__(self, *args, **kwargs):
        super().__init__(*args, set_message='{}{}', **kwargs)
        self.write('CE')


class Filter(KrohnHite3940):
    def config(self, lowpass=None, highpass=None, kind='butterworth'):
        if kind == 'butterworth':
            kind = 1
        elif kind == 'bessel':
            kind = 2
        else:
            raise ValueError(f'Unknown kind passed to config: {kind}.')

        if lowpass and highpass:
            self.CH = 1
            self.M, self.T = 3, kind
            self.F = highpass

            self.CH = 2
            self.F = lowpass

        elif lowpass:
            self.CH = 2
            self.M, self.T = 1, kind
            self.F = lowpass

        elif highpass:
            self.CH = 1
            self.M, self.T = 2, kind
            self.F = highpass

        else:  # Bypass
            self.CH, self.M = 1, 5
            self.CH, self.M = 2, 5
```

B.1.8 instruments/hygrometer.py

```
from instruments import Instrument, Property


class VaisalaHMT313(Instrument):
    FORM = Property()
    SEND = Property()

    def __init__(self, *args, **kwargs):
        super().__init__(*args, query_message='{}', **kwargs)

```

```
        self.resource.read_termination = None
        self.resource.write_termination = '\r'


class Hygrometer(VaisalaHMT313):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)

        self.FORM = 'RH #n'
        self.resource.read_termination = '\n'

    @property
    def relative_humidity(self):
        return float(self.SEND)
```

B.1.9 instruments/impedanceanalyzer.py

```
import re

from instruments import Instrument, Property


class HP4192A(Instrument):
    A = Property(int)
    C = Property(int)
    F = Property(int)
    V = Property(int)
    OL = Property(float, postfix='EN')
    FR = Property(float, postfix='EN')

    def __init__(self, resource):
        super().__init__(resource, set_message='{}{}')
        self.resource.read_termination = '\r\n'


class ImpedanceAnalyzer(HP4192A):
    def measure(self, frequency, voltage):
        self.V = 1  # Enable averaging
        self.A = 2  # Set a + ib output
        self.C = 3  # Select admittance (parallel circuit)
        self.F = 0  # Output from display A and B
        self.OL = voltage  # Set voltage
        self.FR = frequency / 1e3  # Set frequency
        value = self.query('EX')

        number = '[+\-]?[\d.]+E[+\-]?[\d]+'
        match = re.match(f'NGFN({number}),NBFN({number})', value)

        if match:
            return float(match.group(1)) + float(match.group(2))*1j
        else:
```

```
            raise ValueError(f'Impedance analyzer returned unreadable value {value!r}.')
```

B.1.10 instruments/oscilloscope.py

```
from collections import namedtuple, OrderedDict
from time import strftime

from numpy import array, arange, logspace, newaxis, absolute, int16

from . import SCPIInstrument, Property


_channels = ('CH1', 'CH2', 'CH3', 'CH4',
             'MATH', 'REF1', 'REF2', 'REF3', 'REF4',
             'D0', 'D1', 'D2', 'D3', 'D4', 'D5', 'D6', 'D7', 'D8', 'D9',
 'D10', 'D11', 'D12', 'D13', 'D14', 'D15',
             'DIGITAL', 'AUX')
_scales = namedtuple('Scales', ('horizontal', 'ch1', 'ch2'))


class TektronixDPO3000(SCPIInstrument):
    ACQUIRE_MODE = Property(choices=('SAMPLE', 'PEAKDETECT', 'HIRES',
 'AVERAGE', 'ENVELOPE'))
    ACQUIRE_NUMAVG = Property(int)
    ACQUIRE_STATE = Property(choices=('OFF', 'ON', 'RUN', 'STOP'))
    ACQUIRE_STOPAFTER = Property(choices=('RUNSTOP', 'SEQUENCE'))

    CH1_COUPLING = Property(choices=('AC', 'DC', 'GND'))
    CH1_LABEL = Property(quotes='"')
    CH1_POSITION = Property(int)
    CH1_SCALE = Property(float)
    CH2_COUPLING = Property(choices=('AC', 'DC', 'GND'))
    CH2_LABEL = Property(quotes='"')
    CH2_POSITION = Property(int)
    CH2_SCALE = Property(float)

    CURVE = Property(array, query_function='query_values')

    DATA_SOURCE = Property(choices=_channels)
    DATA_ENCDG = Property(choices=('ASCII', 'FASTEST', 'RIBINARY', 'RPBINARY',
 'SRIBINARY', 'SRPBINARY'))
    DATA_WIDTH = Property(int)

    CURSOR_FUNCTION = Property(choices=('OFF', 'SCREEN', 'WAVEFORM'))
    CURSOR_VBARS_POSITION1 = Property(float)
    CURSOR_VBARS_POSITION2 = Property(float)
    CURSOR_VBARS_UNITS = Property(choices=('SECONDS', 'HERTZ', 'DEGREES',
 'PERCENT'))

    HORIZONTAL_DELAY_MODE = Property(choices=('OFF', 'ON'))
```

```
    HORIZONTAL_POSITION = Property(float)
    HORIZONTAL_RECORDLENGTH = Property(int)
    HORIZONTAL_SCALE = Property(float)

    HEADER = Property(choices=('OFF', 'ON'))
    DATE = Property(quotes='"')
    TIME = Property(quotes='"')

    SELECT_CH1 = Property(choices=('OFF', 'ON'))
    SELECT_CH2 = Property(choices=('OFF', 'ON'))
    SELECT_CONTROL = Property(int)

    TRIGGER_A = Property(choices=('SETLEVEL',))
    TRIGGER_A_EDGE_SOURCE = Property(choices=_channels)
    TRIGGER_A_MODE = Property(choices=('AUTO', 'NORMAL'))

    WFMOUTPRE_XINCR = Property(float)
    WFMOUTPRE_XZERO = Property(float)
    WFMOUTPRE_YMULT = Property(float)
    WFMOUTPRE_YOFF = Property(float)
    WFMOUTPRE_YZERO = Property(float)

    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)

        self.resource.read_termination = '\n'
        self.resource.write_termination = '\n'


Channel = namedtuple('Channel', ['number', 'label', 'propagates'])


class Oscilloscope(TektronixDPO3000):
    hdivs = 10
    _hscales = logspace(-9, 3, 13)[newaxis].T * array((1, 2, 4))
    _hscales[3, 0] = 8e-7
    _hscales = _hscales.flatten()[:-2]

    vdivs = 10
    _vscales = logspace(-3, 1, 5)[newaxis].T * array((1, 2, 5))
    _vscales =_vscales.flatten()[:-2]

    def __init__(self, *args, noise_percentage=.1, timeout=300000, **kwargs):
        super().__init__(*args, **kwargs)

        self.resource.timeout = timeout
        self.resource.values_format.is_binary = True
        self.resource.values_format.datatype = 'h'
        self.resource.values_format.is_big_endian = True
        self.resource.values_format.container = int16

```

```
94          self.write('*RST;*WAI')
95
96          self.TRIGGER_A_EDGE_SOURCE = 'AUX'
97          self.TRIGGER_A_MODE = 'NORMAL'
98          self.TRIGGER_A = 'SETLEVEL'
99
100         self.SELECT_CH1 = self.SELECT_CH2 = 'ON'
101         self.CH1_COUPLING = self.CH2_COUPLING = 'AC'
102
103         self.HORIZONTAL_DELAY_MODE = 'OFF'
104         self.HORIZONTAL_RECORDLENGTH = 1e4
105         self.HORIZONTAL_POSITION = noise_percentage * 100
106
107         self.DATE = strftime('%Y-%m-%d')
108         self.TIME = strftime('%H:%M:%S')
109
110         self.HEADER = 'OFF'
111         self.DATA_WIDTH = 2
112         self.DATA_ENCDG = 'RIBINARY'
113
114     def acquire(self, time_scale=None, ch1_scale=None, ch2_scale=None,
    ↪ length=None, average=None, position=None):
115         if average:
116             self.ACQUIRE_MODE = 'AVERAGE'
117             self.ACQUIRE_NUMAVG = average
118
119         if time_scale:
120             self.HORIZONTAL_SCALE = time_scale
121
122         if length:
123             self.HORIZONTAL_RECORDLENGTH = length
124
125         if position:
126             self.HORIZONTAL_POSITION = position
127
128         self.CH1_POSITION = 0
129         if ch1_scale:
130             self.CH1_SCALE = ch1_scale
131
132         self.CH2_POSITION = 0
133         if ch2_scale:
134             self.CH2_SCALE = ch2_scale
135
136         self.ACQUIRE_STOPAFTER = 'SEQUENCE'
137         self.ACQUIRE_STATE = 'RUN'
138         self.write('*WAI')
139
140     def record(self, channel):
141         self.DATA_SOURCE = f'CH{channel}'
142         data = self.CURVE
143
```

```
        x_zero = self.WFMOUTPRE_XZERO
        x_increment = self.WFMOUTPRE_XINCR
        y_zero = self.WFMOUTPRE_YZERO
        y_multiplier = self.WFMOUTPRE_YMULT
        y_offset = self.WFMOUTPRE_YOFF

        timestamps = x_zero + arange(0, len(data)) * x_increment
        amplitudes = (data - y_offset) * y_multiplier + y_zero

        return timestamps, amplitudes

    def cursors(self, cursor1=None, cursor2=None):
        self.CURSOR_FUNCTION = 'WAVEFORM'
        self.CURSOR_VBARS_UNITS = 'SECONDS'

        if cursor1 is not None:
            self.CURSOR_VBARS_POSITION1 = cursor1

        if cursor2 is not None:
            self.CURSOR_VBARS_POSITION2 = cursor2

    @property
    def channel(self):
        return self.SELECT_CONTROL

    def scale(self, channel=None):
        return getattr(self, (f'CH{channel}' if channel else 'HORIZONTAL') +
↪   '_SCALE')

    def get_scales(self, direction, minimum_scale=0,
↪   maximum_scale=float('inf')):
        if direction == 'vertical':
            return self._vscales[(minimum_scale <= self._vscales) &
↪   (self._vscales <= maximum_scale)]
        elif direction == 'horizontal':
            return self._hscales[(minimum_scale <= self._hscales) &
↪   (self._hscales <= maximum_scale)]
        else:
            raise ValueError(f'Unknown direction {direction}.')

    def step_scale(self, current_scale, delta=0, direction='vertical',
↪   **kwargs):
        scales = self.get_scales(direction, **kwargs)
        current_idx = absolute(scales - current_scale).argmin()
        next_idx = current_idx + delta

        if next_idx < 0:
            next_idx = 0
        elif next_idx >= len(scales):
            next_idx = len(scales) - 1

```

```
        return scales[next_idx]

    def scale_delta(self, scale1, scale2, direction='vertical'):
        scales = self.get_scales(direction)
        idx1 = absolute(scales - scale1).argmin()
        idx2 = absolute(scales - scale2).argmin()
        return idx2 - idx1
```

B.1.11 instruments/thermometer.py

```
import re
from datetime import datetime, timedelta

from instruments import Instrument, Property


class ASLF250MKII(Instrument):
    D = Property()

    def __init__(self, *args, **kwargs):
        super().__init__(*args, query_message='{}', **kwargs)

        self.resource.read_termination = '\r\n'
        self.resource.write_termination = '\n'


class Thermometer(ASLF250MKII):
    def __init__(self, *args, skip_time=timedelta(seconds=5), **kwargs):
        super().__init__(*args, **kwargs)

        self.last_query_time = datetime.min
        self.skip_time = skip_time

    @property
    def temperature(self):
        if datetime.now() - self.last_query_time >= self.skip_time:
            value = self.D
            self.last_query_time = datetime.now()
            match = re.match('(\w) ([\d.]+)(\w)', value)

            if match:
                return float(match.group(2))
            else:
                raise ValueError(f'Thermometer returned unreadable value
↪   {value!r}.')
        else:
            return float('nan')
```

B.1.12 instruments/waveformgenerator.py

```
from instruments import SCPIInstrument, Property


class Agilent33220A(SCPIInstrument):
    BURST_INTERNAL_PERIOD = Property(float)
    BURST_MODE = Property(choices=('TRIGGERED', 'GATED'))
    BURST_NCYCLES = Property(int)
    BURST_STATE = Property(choices=('OFF', 'ON'))

    FREQUENCY = Property(int)
    FUNCTION = Property(choices=('SINUSOID', 'SQUARE', 'RAMP', 'PULSE',
↪   'NOISE', 'DC', 'USER'))
    OUTPUT = Property(choices=('OFF', 'ON'))
    VOLTAGE = Property(float)


class WaveformGenerator(Agilent33220A):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.write('*RST')

    def signal(self, frequency, cycles, voltage, period):
        self.FUNCTION = 'SINUSOID'
        self.FREQUENCY = frequency
        self.VOLTAGE = voltage

        self.BURST_STATE = 'ON'
        self.BURST_MODE = 'TRIGGERED'
        self.BURST_NCYCLES = cycles
        self.BURST_INTERNAL_PERIOD = period

        self.OUTPUT = 'ON'
```

B.1.13 spec/data.py

```
from tables import FloatAtom, ComplexAtom


def setup_variables(file, distance=None, burst_length=None):
    if distance is not None:
        file.create_array(file.root, 'd', distance, title='Separation
↪   distance')
    if burst_length is not None:
        file.create_array(file.root, 'l', burst_length, title='Burst length')


def frequency_variable(file, expected_rows=None):
    return file.create_earray(file.root, 'f',
                              atom=FloatAtom(8),
```

```
                                shape=(0,),
                                title='Frequency',
                                expectedrows=expected_rows)


def environment_variables(file, expected_rows=None):
    return (file.create_earray(file.root, label,
                                atom=FloatAtom(8),
                                shape=(0,),
                                title=description,
                                expectedrows=expected_rows)
            for label, description in (('t', 'Temperature'),
                                       ('p', 'Pressure'),
                                       ('h', 'Relative humidity')))


def waveform_variables(file, sample_count, expected_rows=None):
    return (file.create_earray(file.root, label,
                                atom=FloatAtom(8),
                                shape=(0, sample_count),
                                title=description,
                                expectedrows=expected_rows)
            for label, description in (('ttx', 'Transmitter timestamps'),
                                       ('htx', 'Transmitter amplitudes'),
                                       ('trx', 'Receiver timestamps'),
                                       ('hrx', 'Receiver amplitudes')))


def admittance_variable(file, expected_rows=None):
    return file.create_earray(file.root, 'Y',
                              atom=ComplexAtom(16),
                              shape=(0,),
                              title='Admittance',
                              expectedrows=expected_rows)
```

B.1.14 spec/measurements.py

```
from collections import OrderedDict

from numpy import absolute, floor

from instruments.oscilloscope import Channel


def measure_waveforms(frequency, burst_length, propagation_time, voltage,
                      oscilloscope, waveform_generator):
    # Measurement parameters
    noise_percentage = .1
    window = (propagation_time + burst_length) / (1 - noise_percentage)
    cycles = floor(burst_length * frequency)  # Number of cycles per burst
    duration = cycles / frequency
```

```
15
16      if cycles < 1:
17          raise ValueError('Burst length too short to include any cycles.')
18
19      # Set up burst
20      waveform_generator.signal(frequency, cycles, voltage=voltage, period=3 *
    ↪  window)
21
22      # Initialize channels and scales
23      channels = (Channel(1, label='tx', propagates=False), Channel(2,
    ↪  label='rx', propagates=True))
24      time_scale = oscilloscope.get_scales('horizontal', minimum_scale=window /
    ↪  oscilloscope.hdivs)[0]
25      amplitude_scales = OrderedDict(((channel, oscilloscope.scale(channel)) for
    ↪  channel, _, _ in channels))
26
27      timestamp, amplitude = {}, {}
28      while not all(label in timestamp and label in amplitude for _, label, _ in
    ↪  channels):
29          # Prepare capture
30          oscilloscope.acquire(time_scale, *amplitude_scales.values(),
    ↪  average=128, position=100 * noise_percentage)
31          oscilloscope.cursors(propagation_time, propagation_time + duration)
32
33          for channel, label, propagates in channels:
34              # Record trace
35              timestamp[label], amplitude[label] = oscilloscope.record(channel)
36
37              # Check for clipping or suboptimal scaling
38              start = propagation_time if propagates else 0
39              end = start + duration
40              burst = (start < timestamp[label]) & (timestamp[label] < end)
41              peak = absolute(amplitude[label][burst]).max()
42              fill = peak / (amplitude_scales[channel] * oscilloscope.vdivs / 2)
43              next_scale = oscilloscope.step_scale(amplitude_scales[channel],
44                                                   delta=-1 if fill < .25 else 1
    ↪  if fill > .75 else 0,
45                                                   minimum_scale=10e-3)
46
47              # Delete trace and rescale if the channel needs to be rescaled
48              if oscilloscope.scale_delta(next_scale, amplitude_scales[channel])
    ↪  != 0:
49                  amplitude_scales[channel] = next_scale
50                  del timestamp[label], amplitude[label]
51
52      return timestamp, amplitude
53
54
55  def measure_admittance(frequency, impedance_analyzer):
56      return impedance_analyzer.measure(frequency, voltage=1)
```

Appendix C

tftools

The tftools library contains a set of MATLAB utility functions written by the current author to facilitate post-processing of transfer functions.

The physical models in the library and its tools are written as general as feasible, and as such may lend itself reuse elsewhere. Due to time constraints the library is poorly documented, and users are encouraged to read the source code for hints on how to use it.

C.1 Source code

C.1.1 attenuationNitrogen

Path: models/attenuationNitrogen.m

```
function [alphaN, frN] = attenuationNitrogen(f, t, p, h)
T = 273.15 + t; % Thermodynamic temperature

% Approximate equations after ANSI/ASA S.126-1995
pa = p/1e3; % Kilopascals
pr = 101.325; % NIST STP pressure in kilopascals
Tr = 293.15; % NIST STP temperature
xw = waterVapor(t, p, h);
xwp = xw * 100; % Mole fraction of water vapor in percent

frN = pa/pr .* (T/Tr).^(-1/2) .* (9 + 280*xwp.*exp(-4.170*((T/Tr).^(-1/3) - 1)));
alphaN = f.^2 .* (T/Tr).^(-5/2) .* 0.1068.*exp(-3352.0./T).*(frN./(frN.^2 + f.^2));
end
```

C.1.2 attenuationOxygen

Path: models/attenuationOxygen.m

```
function [alphaO, frO] = attenuationOxygen(f, t, p, h)
T = 273.15 + t; % Thermodynamic temperature

% Approximate equations after ANSI/ASA S.126-1995
pa = p/1e3; % Kilopascals
pr = 101.325; % NIST STP pressure in kilopascals
Tr = 293.15; % NIST STP temperature
```

```
xw = waterVapor(t, p, h);
xwp = xw * 100; % Mole fraction of water vapor in percent

frO = pa/pr .* (24 + (4.04e4*xwp).*(0.02 + xwp)./(0.391 + xwp));
alphaO = f.^2 .* (T/Tr).^(-5/2) .* 0.01275.*exp(-2239.1./T).*(frO./(frO.^2 +
↪  f.^2));
end
```

C.1.3 BPDC

Path: models/BPDC.m

```
function H = BPDC(k, a, z)
H = 1 - (4/pi) * ...
    integral(@(theta) exp(-1i * k .* (sqrt(z.^2 + (2*a*cos(theta)).^2) - z))
↪  .* ...
                        sin(theta).^2, ...
             0, pi/2, 'ArrayValued', true);
end
```

C.1.4 carbonDioxide

Path: models/carbonDioxide.m

```
function xc = carbonDioxide
xc = 4e-4; % Mole fraction of carbondioxide, assumed 400ppm (CITATION NEEDED)
end
```

C.1.5 speedOfSound

Path: models/speedOfSound.m

```
function c = speedOfSound(f, t, p, h)
%C Calculates the speed of sound
%   c = c(f, t, p, h) Calculates the speed of sound based on the values
%   given for frequency (), temperature (Celsius), pressure (Pascal),
%   and relative humidity (percentage in range 0-1).
f = f(:)';
t = t(:)';
p = p(:)';
h = h(:)';

xc = carbonDioxide; % Mole fraction of carbondioxide
xw = waterVapor(t, p, h); % Mole fraction of water vapor

[alphaO, frO] = attenuationOxygen(f, t, p, h);
[alphaN, frN] = attenuationNitrogen(f, t, p, h);

% Approximate equation after Cramer/1993
a = [331.5024 0.603055 -0.000528 51.471935 0.1495874 -0.000782 -1.82e-7 ...
     3.73e-8 -2.93e-10 -85.20931 -0.228525 5.91e-5 -2.835149 -2.15e-13 ...
```

```
     29.179762 0.000486];
tp = [ones(size(t)); t; t.^2];
c0 = a*[tp; repmat(xw, 3, 1).*tp; repmat(p, 3, 1).*tp; xc*tp; ...
        xw.^2; p.^2; repmat(xc^2, 1, length(t)); xw.*p.*xc];

% Dispersion correction after Howell and Morfey/1980
c = (c0.^-1 - (alpha0./fr0 + alphaN./frN)/(2*pi)).^-1;
end
```

C.1.6 transmissionLine

Path: models/transmissionLine.m

```
function E = transmissionLine(f, l, L, C, Z, varargin)
p = inputParser;
addOptional(p, 'impedance', 'load', @(x) any(strcmp(x, {'input', 'load'})))
parse(p, varargin{:})
impedance = p.Results.impedance;

c = sqrt(L*C)^-1;
RC = sqrt(L/C);

k = 2*pi*f / c;
Za = 1i * RC * tan(k*l/2);
Zb = -1i * RC ./ sin(k*l);
V = 1;

E = zeros(length(f), 4);
for idx = 1:length(f)
    if strcmp(impedance, 'load')
        constraints = [zeros(3, 4); 0 0 -1 Z(idx)];
    elseif strcmp(impedance, 'input')
        constraints = [zeros(3, 4); -1 Z(idx) 0 0];
    end

    %    Vs Is               Vr Ir
    A = [ 1  0                0  0;
         -1  Za(idx)+Zb(idx)  0 -Zb(idx);
          0 -Zb(idx)          1  Za(idx)+Zb(idx);
          0  0                0  0] + constraints;
    b = [ V  0                0  0]';

    x = A\b;
    E(idx,:) = x;
end

E = array2table(E, 'VariableNames', {'Vsource', 'Isource', ...
                                     'Vreceiver', 'Ireceiver'});
end
```

C.1.7 waterVapor

Path: models/waterVapor.m

```
function xw = waterVapor(t, p, h)
T = 273.15 + t; % Thermodynamic temperature

% fW, psv and xw given in Cramer/1993 appendix
f = 1.00062 + 3.14e-8*p + 5.6e-7*t.^2; % Enhancement factor
psv = exp(1.2811805e-5*T.^2 - 1.9509874e-2*T + 34.04926034 - 6.3536311e3./T);
 ↪  % Saturation vapor pressure
xw = h .* f .* psv./p; % Mole fraction of water vapor
end
```

C.1.8 applytf

Path: transferfunctions/applytf.m

```
function h = applytf(h, f, H)
H = griddedInterpolant(f, H, 'nearest');
H = H(0:f(2)-f(1):f(end)) / 2;
H = [H fliplr(conj(H(2:end-1)))].';

for idx = 1:size(h, 2)
    h(:,idx) = real(ifft(fft(h(:,idx)) .* H));
end
end
```

C.1.9 crossings

Path: transferfunctions/private/crossings.m

```
function crossings = crossings(h)
%CROSSINGS Detects zero crossings in the signal h
%   crossings = crossings(h) Finds all the points at which the signal
%   crosses the zero line. When a zero crossing is found the point before
%   or after the crossing is returned depending on which is closer to zero.
h = h(:);

% Find all crossings
crossings = find(h(1:end-1) .* h(2:end) <= 0 & ...
                 (h(1:end-1) ~= 0 | h(2:end) ~= 0)); % First index of
 ↪  crossings
[~, idx] = min(abs(h([crossings crossings + 1])), [], 2); % Is first or second
 ↪  index smaller?
crossings = crossings + idx - 1; % Correct if second is smaller
crossings = unique(crossings); % Remove duplicates
end
```

C.1.10 fftpad

Path: transferfunctions/private/fftpad.m

```
function [N, n] = fftpad(f, L, fs, range)
%FFTPAD Optimizes fft padding to center a frequency bin close to the
%frequency f.
%   fftpad(f, L, fs) finds the optimal signal length in the range [L 2L].
%   fftpad(___, range) extends the range to the specified range parameter.

if nargin < 4
    range = [1 2];
end

% Possible values
n = ceil(range(1)*L*f/fs):floor(range(2)*L*f/fs);
N = n*(fs/f);

% Find optimal
[~, idx] = min(N - round(N));

% Extract
n = n(idx) + 1; % Off by one due to MatLab indexing
N = round(N(idx));
end
```

C.1.11 waveformtf

Path: transferfunctions/waveformtf.m

```
function H = waveformtf(f, ttx, htx, trx, hrx, bt, pt)
if length(pt) == 1
    pt = repmat(pt, size(f));
end

if length(bt) == 1
    bt = repmat(bt, size(f));
end

H = cellfun(@transferfunctionvalue, ...
    num2cell(f, 1), ...
    num2cell(ttx, 1), num2cell(htx, 1), ...
    num2cell(trx, 1), num2cell(hrx, 1), ...
    num2cell(bt, 1), num2cell(pt, 1));
end

function H = transferfunctionvalue(f, ttx, htx, trx, hrx, bt, pt)
for data = [struct('ch', 'tx', 'time', ttx, 'sig', htx, 'propagates', false), ...
            struct('ch', 'rx', 'time', trx, 'sig', hrx, 'propagates', true)]
    % Reassign loop data
        time = data.time;
    sig = data.sig;
    ch = data.ch;
    propagates = data.propagates;
```

```
25
26      % Remove bias
27      sig = sig - mean(sig);
28
29      % Calculate sample rate
30      dt = time(2) - time(1);
31      fs = 1/dt;
32
33      for data = [struct('idx', 1, 'discard', .4, 'indexer', crossings(sig)),
    ↪   ...
34                  struct('idx', 2, 'discard', 0, 'indexer', 1:length(time))]
    ↪   %#ok<FXSET>
35          % Reassing loop data
36          idx = data.idx;
37          discard = data.discard;
38          indexer = data.indexer;
39
40          % Estimate start and stop based on characteristic data
41          start = pt * propagates;
42          stop = start + bt;
43
44          % Discard part of the signal according to the discard variable and
45          % readjust to closest estimated period based on characteristic data
46          discard = discard * bt;
47          cyclestarts = (0:floor(f * bt)-1) / f;
48          [~, ndx] = min(abs(cyclestarts - discard));
49          start = start + cyclestarts(ndx);
50
51          % Select closest sample to calculated start and stop according to
52          % the indexer variable
53          [~, start] = min(abs(time(indexer) - start));
54          [~, stop] = min(abs(time(indexer) - stop));
55          ndx = indexer(start):indexer(stop);
56
57          % Calculate DFT bin of cut signal
58          h = sig(ndx);
59          L = length(h);
60          [N, n] = fftpad(f, L, fs);
61          dft.(ch)(idx) = goertzel([h; zeros(N-L, 1)], n);
62      end
63  end
64
65  % Calculate transfer function value
66  Habs = abs(dft.rx(1)/dft.tx(1));
67  Hangle = angle(dft.rx(2)/dft.tx(2));
68  H = Habs * exp(1i * Hangle);
69  end
```