UNIVERSITY OF OSLO
Department of Informatics

Estimation of attenuation in tissue as a function of depth from RF ultrasound data

Master's thesis

Kjetil N. Limkjær

4th May 2009

Contents

List of Figures

1 Introduction

When an ultrasound beam from a transducer reaches tissue, the signal will be attenuated. This attenuation is partly caused by absorption, and partly by scattering. For most soft tissues absorption accounts for most of the attenuation [Ophir et al., 1984].

Research has shown that different types of tissues will display different attenuation characteristics. For instance, cirrhotic liver tissue will absorb more of the ultrasonic signal than healthy tissue [Wells et al., 1976]. Accurate non-intrusive measurement of attenuation could thus be an important diagnostic aid.

In this thesis we will review various methods for estimating this attenuation, and compare their efficency under various conditions. To facilitate this comparison we will be using the ultrasound simulation program Field II [Jensen, 1996, Jensen and Svendsen, 1992].

1.1 Our work in summary

We wil start off by quickly reviewing the research that has been done so far on the subject of ultrasonic attenuation, with a particular focus on attenuation estimation. Because of the extensive research that has been done on this subject, a thorough review of the literature published would warrant a thesis in its own right. We will, however, provide an overview of some of the methods that can be used to estimate the ultrasonic attenuation in soft tissues.

The main focus of this thesis will be on a simulation framework for applying different attenuation estimators. Our main goal has been to make an easy-to-use, generic framework where it is easy to add new estimators, rather than providing a large set of estimators from the outset. We have also aimed for making it simple to compare the performance on various estimator functions on similar datasets, optionally complicating simulations by adding noise or modifying other simulation parameters. Ideally these simulations should be done using soft tissue-like reflector clusters and their backscattered data, to maximise the similarities to real-world conditions. The fact is unfortunately that it has proven difficult to get any sort of consistent results in this manner, so instead we resorted to using more narrowly defined “phantom reflectors”. This can be seen as being a middle ground between the shadowed reflector method and the backscatter approach.

We will conclude the thesis with a summary of our results and suggestions for

further research.

1.2 A birds-eye view of research into ultrasonic attenuation

Research into the ultrasonic attenuation properties of soft tissues has been going on since the late 1930s, at the very least [Pohlman, 1939]. The use of these properties for tissue characterisation, however, was only suggested much later. In the early 1970s several articles and presentations were presented concerning the likely use of attenuation estimation as a medical diagnostics tool. This generated a lot of interest in the subject—Melvin Linzer even went so far as to call it "perhaps the last major frontier in medical ultrasound" in his introduction to the proceedings of the first international Seminar on Ultrasonic Tissue Characterization [Linzer, 1976].

The attenuation in soft tissues was at first thought to increase linearly with frequency, with initial studies confirming this [Colombati and Petralia, 1950, Esche, 1952]. It was later discovered that an exponential increase was more generally applicable, with the attenuation taking the form of Equation (1).

$$|H(f)| = e^{-2\alpha_0 f^n Z} \tag{1}$$

In this equation, $|H(f)|$ represents the attenuation, α_0 is the attenuation coefficient and n represents the exponent of frequency dependence [Narayana and Ophir, 1983]. However, one can assume linearity for most types of soft tissue in the low megahertz range [Linzer and Norton, 1982], and in our discussion this is what we will concentrate on. "Low" is also a relative term, of course—for instance, kidney tissue has been shown to be be linear up to around 100 Mhz [Dunn, 1976].

Research into soft tissue attenuation has as we see it been focused mainly on two things:

- Estimating the attenuation coefficient
- Cataloguing the coefficients for different tissues

The cataloguing element is important in that it provides the driving force for the estimation techniques, for instance by detailing differences in attenuation between healthy and sick organs [Miller et al., 1983]. A significant overlap between the two types of articles also exists. However, in this thesis our focus will be primarily on the estimation techniques.

There has been no shortage of research on the subject since it was first introduced, with several researchers contributing different estimation methods. The attenuation is most easily measured in the frequency domain, but several time-domain methods for estimation have been suggested, using various types of centre frequency estimators. The subject is still being actively researched, and new techniques were still being proposed at the time of writing. *In vivo* application of the methods still prove problematic, however, because of the many different parameters involved.

1.3 Thesis outline

Section 2 acts as an overview of some of the methods for attenuation estimation.

Section 3 presents a selection of methods for centre frequency estimation.

Section 4 introduces the simulation tools we have used and created.

Section 5 discusses the simulations we've performed and their results.

Section 6 concludes the thesis with a review of what we've done and suggestions for future work.

At the very end an Appendix is presented, containing the Matlab source files for the simulation framework.

2 Methods for attenuation estimation

All methods used for attenuation estimation are similar in that they all require the same information to function - two signals, $\boldsymbol{s_1}$ and $\boldsymbol{s_2}$, and a depth difference, $\boldsymbol{\Delta z}$. The methods for obtaining the two signal measurements can differ, but the essence of the estimate is the same.

2.1 The transmission method

This is the most basic way of measuring attenuation, and also the most accurate. An ultrasonic transmitter and receiver are set up as in Figure 1, with the tissue sample to be analyzed submerged in fluid in between. Using this setup, the signal s_1 is first recorded with the tissue sample removed, travelling through the fluid only. The s_2 signal is then recorded with the tissue in place. The Δz parameter is then the width of the tissue.

2.2 The shadowed reflector method

This is similar to the transmission method, but requires only a single transducer. Refer to Figure 2 for the setup. As in the transmission method, the signals s_1 and s_2 are recorded with and without the tissue sample in place, but the Δz parameter is now double the width of the tissue, as the signal has to travel through it twice [Lele et al., 1976].

2.3 Backscattered signal

What the previous two methods both have in common is that they are mostly suitable for *in vitro* application, seeing as they require the tissue sample to be isolated for measuring. While there are some *in vivo* applications for the transmission and shadowed reflector method—for instance in ultrasonic mammography [Green and Taenzer, 1984]—the backscatter method is generally the most applicable for this. Here, no reflector is used, instead the backscattered signal from the tissue itself is recorded. The s_1 and s_2 measurements are simply scatter measurements from different parts of the tissue, a shallow part and a deep part. Often, they are just two different windowed segments of the same measurement (see Figure 3). As for Δz, it is now double the depth difference between the shallow and the deep sample.

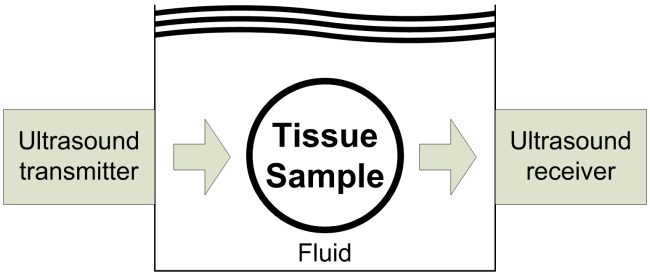


Figure 1: Setup for measuring attenuation using transmitted ultrasound

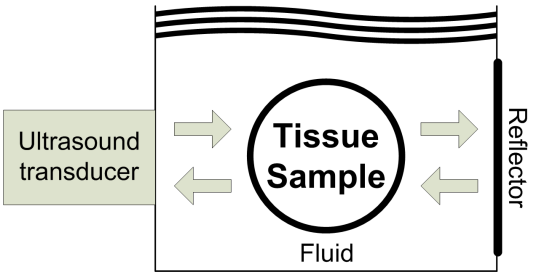


Figure 2: Setup for measuring attenuation using reflected ultrasound

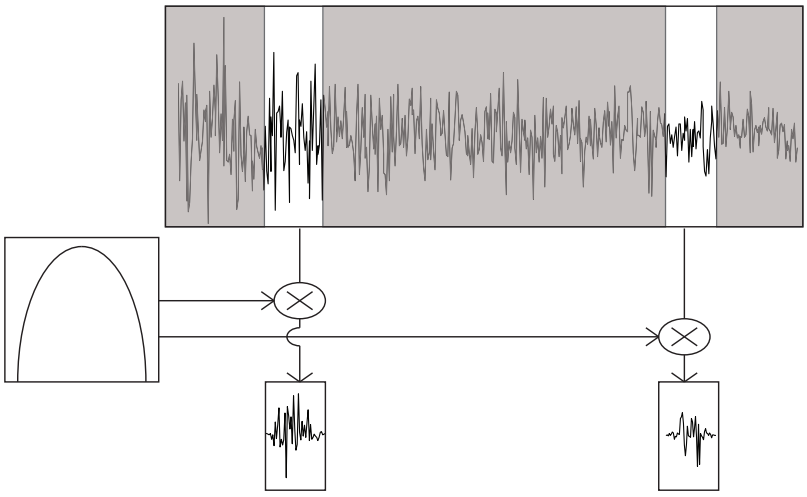


Figure 3: Windowed signal segments

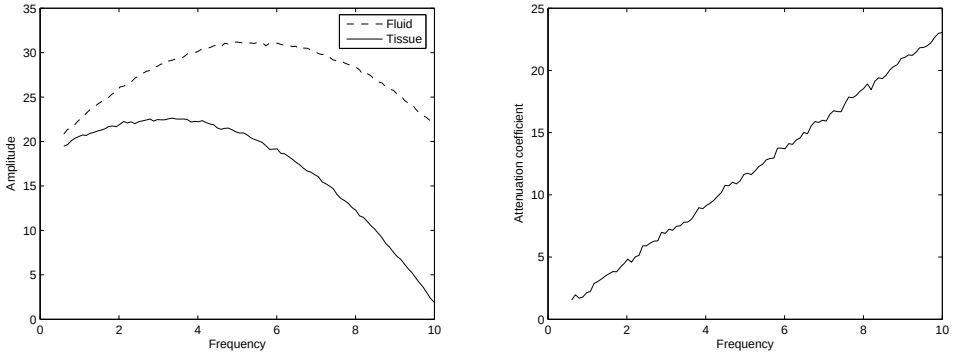
We will be concentrating on the backscatter method, as this is the most interesting method for diagnostic use; if we would like to use attenuation measurement for internal organ diagnosis—for instance to investigate the possibility of liver disease—it is preferable for the patient if the organ in question does not first need to be removed.

2.4 Log spectral difference

The log spectral difference is a conceptually simple way of estimating the attenuation coefficient. First, we calculate the log power spectra of s_1 and s_2, using the Fourier transform. Then, we take the difference between these spectra, and divide it by Δz, giving the formula

$$\alpha(f) = \frac{P_1(f) - P_2(f)}{\Delta z} \tag{2}$$

where $\boldsymbol{f}$ is frequency and $\boldsymbol{P_1}$ and $\boldsymbol{P_2}$ are the power spectra of s_1 and s_2. This gives us the attenuation for a given frequency. As we assume linear attenuation, a line can be fitted to $\alpha(f)$. The slope of this line is the attenuation coefficient α [Miller et al., 1976]. See Figure 4 for an illustration. In most cases, especially when using backscattered signals, a least squares line will need to be fitted to the difference to estimate the attenuation slope.



(a) Spectra (b) Attenuation

Figure 4: Log spectral difference. The amplitude is given in dB and the attenuation is measured in dB/(cm Mhz).

2.5 Spectral shift

This method makes use of the frequency-dependence of attenuation. In essence, the tissue behaves like a lowpass filter, as higher frequencies are attenuated more than lower ones. The average and peak frequencies of the transmitted pulse will decrease the further the signal travels through the medium. A Gaussian pulse shape is assumed, which ensures that the spectrum shape and variance is preserved. The frequency shift Δf will be proportional to the slope of attenuation β, the distance travelled x and the pulse variance. The average centre frequency has been shown to be a better basis for the estimate than the peak frequency. To calculate the average centre frequency or *centroid*, the formula

$$c_f = \frac{\int_{f_1}^{f_2} fP(f)df}{\int_{f_1}^{f_2} P(f)df} \tag{3}$$

is used [Ophir et al., 1984]. The discrete equivalent to this is

$$c_f = \frac{\sum_0^{F_s/2} fP(f)}{\sum_0^{F_s/2} P(f)} \tag{4}$$

If linearly frequency dependent attenuation and constant bandwith is assumed, the characterization reduces to estimating and tracking the mean frequency. This can be done in several ways, one of the simplest is the zero crossings method [Ophir et al., 1984]. Other methods include various types of autoregressive models, and the adapted estimator presented by Baldeweck et al. [1994]. The centre-frequency estimation part of the spectral shift method can in fact be done in such diverse ways that it warrants it own section, and an overview of some estimators can be found in section 3.

Once the centre frequencies $\boldsymbol{f_{c1}}$ and $\boldsymbol{f_{c2}}$ have been established for our two signals s_1 and s_2, the attenuation coefficient can be calculated by the following formula [Ophir et al., 1984]

$$\alpha = \frac{f_{c1} - f_{c2}}{\sigma^2 \Delta z} \tag{5}$$

2.6 Matched filter pulse compression

As a work-around for the trade-offs between depth resolution and spectral resolution, this method was proposed. This method transmits bursts of chirps in place of the usual Gaussian pulse. The received signals are then processed

with two amplitude weighted matched filters to estimate the attenuation. Adjustments are made iteratively to "whiten" the received spectrum over the useful system bandwith. This way the attenuation coefficients can be calculated through several tissue segments iteratively, in real time [Meyer, 1979, Ophir et al., 1984].

2.7 Amplitude methods

These methods attempt to estimate the attenuation from only the amplitude of the received signals, not knowing anything about the frequency content. In essence a large number amplitude measurements are taken for a series of depths. Then the average $ln(A)$ of the amplitudes is calculated for each depth, and a least squares model is used to fit this data to a linear model [Mountford and Wells, 1972]. This will give you the attenuation in dB/cm, but tells you nothing of the frequency-dependent attenuation.

The amplitude methods are separated into wideband and narrowband methods. What separates them is the pulse width transmitted. Assuming a Gaussian spectrum, the backscattered energy can be described by the equation

$$E(\alpha_0, \ell) \propto A_0 e^{-(\alpha_0 \ell f_0 - \alpha_0^2 \ell^2 \sigma^2/2)}$$

where ℓ is the measurement depth. This implies that the spectral energy decay is not simply linear in α and ℓ, but has an additional quadratic term $(\alpha_0 \ell \sigma)^2$. This means that this has to be accounted for in estimations, or that the measurements are taken over a narrow enough region where the quadratic term can be neglected [Ophir et al., 1984]. Narrowband amplitude methods get around this problem by instead transmitting a pulse with a narrow enough bandwith that the quadratic term can be ignored, because σ^2 is close to zero.

2.8 Echo Envelope Peak method

He and Greenleaf introduced a method based on finding the peaks of the echo envelope [He and Greenleaf, 1986]. This is an iterative method with relatively low overhead. It takes advantage of indications that the variance of the mean power of the backscattered echoes can be found from the noise-to-signal ratio of the echo envelope.

2.9 Entropy difference method

This method was introduce by Jang et al. in 1988. It uses the entropy of sample values from the envelope to estimate the attenuation, by "finding the minimum difference of the entropies for two adjacent regions as the attenuation is continuously compensated for" [Jang et al., 1988].

3 Centre frequency estimation

3.1 The zero crossings method

The centre frequency can be estimated by measuring the change in the amount of zero crossings in the received signals, if we assume a Gaussian spectrum. It can be shown that the square root of the second moment in the power spectrum of a waveform is related to the density of zero crossings. This gives us the center frequency estimator [Ophir et al., 1984]

$$\lambda \simeq 2\sqrt{f_c^2 + \sigma^2} \simeq 2f_c$$

where λ denotes the zero crossings density. We originally made efforts to include this estimator as part of our framework, but unfortunately it requires long transmitted pulses or at least a very large amount of pulses to give accurate results, leaving it cumbersome to work with in Field II. One study mentioned counting about 3.5×10^6 zero crossings at each studied depth [Ophir et al., 1985].

3.2 Autoregressive estimators

An autoregressive estimator, or *AR estimator* essentially works by fitting the signal to an *AR model* and calculating the power spectrum of this model. The centre frequency can then be found by calculating the average frequency of the spectrum.

When considering an AR estimator, it is important to first understand the idea of an AR model. The difference equation describing an AR model is

$$x(n) = -\sum_{k=1}^{p} a_p(k)x(n-k) \tag{6}$$

where p is the model order and $a_p(k)$ are the called the *prediction coefficients* or *AR coefficients* [Kay and Marple, 1981, Proakis and Manolakis, 1996, Hayes, 1996, Wear et al., 1995]. Using filter design terminology, this can also be described as a filter having the system function

$$H(z) = \frac{b(0)}{1 + \sum_{k=1}^{p} a_p(k)z^{-k}}$$

Autoregressive in this context means that we are using previous samples of $x(n)$ to calculate the new output, the process is thus *regressing* upon itself [Wear et al., 1995].

When applying an AR model such as this as a predictor for a sampled signal $x(n)$—which will most likely not perfectly match the model output—there will be an error involved. This error is given by

$$e_p^+(n) = x(n) - \hat{x}(n) \tag{7}$$

which is called the *forward prediction error*, and can be extended to the *backward prediction error* involving predicting previous values $x(n-j)$ of the signal, given by [Hayes, 1996]

$$e_j^-(n) = x(n-j) - \hat{x}(n-j) \tag{8}$$

All of the various AR methods are based on the same underlying model given in Equation (6). Where they differ is in how the prediction coefficients are determined. The idea behind determining the coefficients is to minimize the errors $e(n)$, but as we shall see there are several ways to do this.

To calculate the power spectrum from an AR model, the following equation is used [Wear et al., 1995]

$$P(F) = \frac{\rho \Delta t}{1 + \sum_{k=1}^{p} a_p(k) e^{-j2\pi f k \Delta t}} \tag{9}$$

where ρ is the variance of the error, f is frequency and Δt is the sampling interval.

There is a potentially large speed advantage to be gained if we instead of calculating the entire spectrum focus on what we are interested in, which is the centre frequency. One way of detecting the centre frequency is by finding the *maximum energy frequency*, which is the frequency that has the highest energy in the spectrum - the top peak in the frequency plot. It is possible to evaluate this frequency directly by differentiating Equation (9) with respect to f and setting it to zero, which gives us [Girault et al., 1998]

$$f_{max}(n) = \frac{f_s}{2\pi} \cos^{-1}\left(\frac{-a_1(n)}{4}\left(1 + \frac{1}{a_2(n)}\right)\right)$$

We will now discuss three possible ways of calculating the AR coefficients.

3.2.1 Burg's algorithm

The essence of Burg's algorithm is that it minimizes the sum of the forward and backward prediction errors. This sum is defined by [Hayes, 1996]

$$\varepsilon_j^B = \varepsilon_j^+ + \varepsilon_j^- = \sum_{n=j}^{N} \left|e_j^+(n)\right|^2 + \sum_{n=j}^{N} \left|e_j^-(n)\right|^2 \tag{10}$$

To calculate the AR coefficients keeping this restriction in mind, the Levinson-Durbin recursion is used [Proakis and Manolakis, 1996]. In this way, the AR coefficients are calcuated one by one, i.e. the lower-order coefficients are kept the same while higher-order ones are calculated. For instance, in a third-order and fifth-order AR model based on Burg's algorithm, the AR coefficients a_1, a_2 and a_3 will be the same.

3.2.2 The modified covariance method

The modified covariance method, like Burg's algorithm, seeks to minimize the sum of the forward and backward prediction errors [Hayes, 1996],

$$\varepsilon_p^M = \varepsilon_p^+ + \varepsilon_p^-$$

However, in contrast to Burg's algorithm, it doesn't use a sequential approach, so for a given model order the whole set of AR coefficients are determined. There is thus a chance that higher-order models created by this method can perform better than Burg's algorithm.

3.2.3 Yule-Walker

The Yule-Walker algorithm uses the autocorrelation $r_x(k)$ of the data to calculate the AR coefficients. In most cases the statistical autocorrelation is not known, and we need to estimate it from the sampled data. It is then used to solve the Yule-Walker equations

$$r_x(k) + \sum_{l=1}^{p} a(p)(l) r_x(k-l) = |b(0)|^2 \delta(k); k \geq 0$$

for the AR coefficients $a_p(k)$[Hayes, 1996].

4 Simulation tools

4.1 Field II and simulated attenuation accuracy

Field II is a ultrasound simulation toolkit for Matlab that is frequently used in the academic ultrasound community. Although it does support simulations of attenuation, not much has been published on the accuracy of these simulations.

When we started our initial work on this thesis, we had a simulation framework in place for a while that was quite different from the one presented in this article. Not to put too fine a point upon it, it was severely broken. Unfortunately, like a broken clock happens to be right twice a day, those simulations did indeed perform very well for one set of data - our initial test case. Needless to say, when we started trying to extend those simulations it did not work out too well. This prompted us to adopt a more rigorous approach.

4.1.1 Applicability of Field II for attenuation estimation

To investigate whether or not Field II is suitable for attenuation estimation, we thus decided to run some initial tests. What we were interested in ascertaining was the accuracy of the attenuation simulation in the program, and in which ranges we could expect realistic results.

As a starting point, we used program settings taken from one of the examples in the Field II manual, namely Example 6.1 - Phased Array Imaging[Jensen, 2001, p. 57]. This entails a 100 MHz sampling rate, a 3 MHz centre frequency, and a 128-element linear array. The array elements are 5 mm high by half a wavelength wide, with a kerf of 0.1 mm. The speed of sound is set to 1540 m/s. To minimize the potential for error, we also used direct measurement of the spatial response instead of relying on scatterers. This can be seen as an analogue of the transmission method. To minimize any source of error and to avoid introducing any dependence on the various estimators at this point, we used the simplest method that came to mind to measure the attenuation. This entailed calculating the power spectra using FFT and directly comparing the power levels for various attenuations, depths and frequency.

Initially, we tested attenuation for the centre frequency, depending on depth, with a fixed α of 0.5 dB/cm/Mhz. We set up Monte Carlo simulations of a random set of 1 000 depths in the range [0, 1000] mm. We then calculated the power spectrum of the attenuated and unattenuated spatial responses at the given depths. The difference between these responses were compared with

the expected attenuation. These differences were then plotted against depth, the results can be seen in Figure 5. The sweet spot for experiments with these settings seem to lie in the 0-400 mm region, with a rapid decrease in accuracy after about 450 mm.

We then set out to see whether these results were consistent over the rest of the frequency range. As a quick check, we fixed the depth at 100 mm and compared simulations of a random set of 1 000 frequencies in the 2 - 4 MHz range to the expected attenuation. We used the attenuation at 3 MHz as a baseline for comparisons. The results are shown in Figure 6.

Seeing as the errors in these results are small enough to suggest that Field II's attenuation simulations are working, at least within a certain range, we ran a test varying the α parameter as well. Soft tissue attenuation coefficients usually lie in the range 0.5-1.0 dB/(cm Mhz)[Papadakis, 1999, p. 58], so this is the range we will be examining. Looking at the results of simulations for depths 0-450 mm, we further narrowed the depth range for best results down to 50-100 mm. However, within this range we still had a mean average error (MAE) of around 0.19 dB/(cm Mhz) compared to the real α. Suspecting that this could be improved, we started tweaking the simulation parameters. Thinking that interactions between array elements could lead to inaccuracies, we reduced the number of elements to 16. This brought the MAE down to 0.15 dB/(cm Mhz), still more than we would like. The next step was to increase the length of the emitted signal from 2 wavelengths to 5, to increase the energy around the center frequency. This had a huge impact on accuracy, bringing the MAE down to 0.016 dB/(cm Mhz) with a standard deviation (σ) of 0.002. As interesting differences in attenuation often are more pronounced than that, 0.1 dB/(cm Mhz) or above[Wilson et al., 1987], we acknowledge that the attenuation simulation in Field II is more than accurate enough for our purposes. Of course, it is also a very clear possibility that the accuracy could be further improved if the transceiver setup or the methods used for estimation were different.

During the tweaking of parameters, we also discovered that the sampling rate could be lowered to around 80 Mhz without affecting the accuracy, giving us a speed boost of about 10 %. Further lowering of the sample rate did however have an adverse effect on the accuracy. Adjustments were also done to the number of elements, which were brought down to 4. Adding more elements after that did not help accuracy in our simulations, and removing more elements decreased it. The fact that we can get by with so few elements is an artifact of our simple test setup. We only have a few reflectors, so we only get a reflected signal from a narrow, well defined direction. Under normal conditions more elements would be needed to get accurate data. However, as our test setup al-

lows it, we will be using a reduced number of elements to reap the speed and accuracy benefits this provides.

4.2 Simulation implementation details

We wanted to use simulated bakscattered signals for the estimations, as this is the most interesting method when one takes real-world applications into account. We used examples from the manual as a starting point, which described a 40 x 10 x 50 mm phantom, but soon discovered that we needed more narrow phantoms to ensure the accuracy we wanted. In fact, in our setup estimators broke down at phantom thicknesses well below a micrometer. We thus needed a set of very narrow reflectors to get any useful data out of the estimators when using the Field II backscatter simulation functions. Because of this, we set up two sharply defined "phantom reflectors" instead of using one bigger phantom. We eventually arrived at a working solution using two very small, effectively two-dimensional phantoms of 0.3 x 0.3 x 0 mm each. We realise that this is more an analogue of the shadowed reflector method than it is similar to the varied ways soft tissues would interact with the signal inside the human body. However, implementing the simulations in this manner meant that we could compare the estimators in a best-case scenario, so we know that we have a functioning framework before starting to introduce complications.

On implementing the centre frequency detection-based estimators, we had to make further adjustments to the simulation model. The problem was that the number of FFT bins we were using (initially 4096) was insufficient to distinguish the drops in centre frequency between measurements. The number was adjusted up to 2^{15}, where diminishing returns started to come into play with regard to accuracy/speed concerns. The reason such a high number of bins is needed is as follows. Recall the formula from Equation (5). In our system, we have estimated the bandwith σ^2 at about 12 kHz. Δz is set to 1.5 cm. Let's say that we would like our estimator to have a resolution of 0.01α. Solving for $d = f_{c1} - f_{c2}$, we get

$$d = 0.01 \cdot (12000 \cdot 1.5) = 180$$

In other words, a change in attenuation of 0.01 dB/(cm Mhz) corresponds to a centre frequency shift of around 180 Hz. Considering that we have a 40Mhz range of useful values with an 80 Mhz sampling frequency, 4096 bins gives us a resolution of

$$40\text{Mhz}/4096 \approx 10\text{kHz}$$

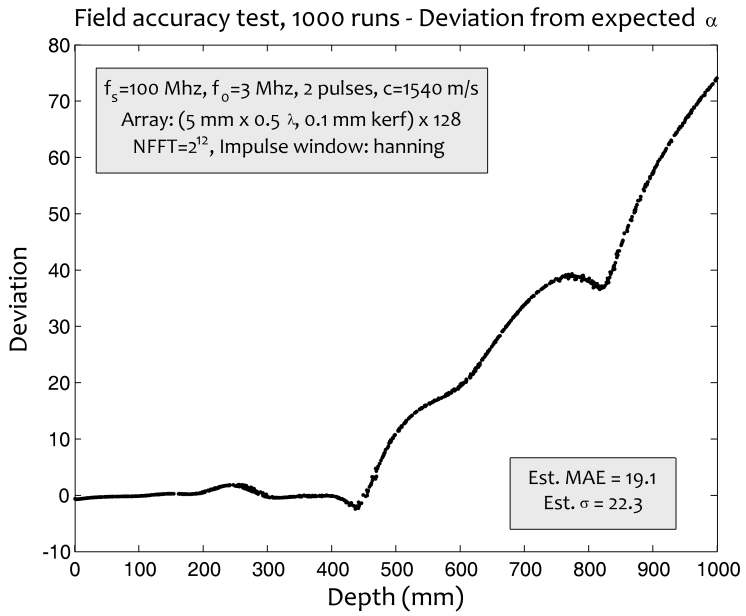


Figure 5: Deviation from expected attenuation in simulations, according to depth. Deviation unit is dB/(cm Mhz).

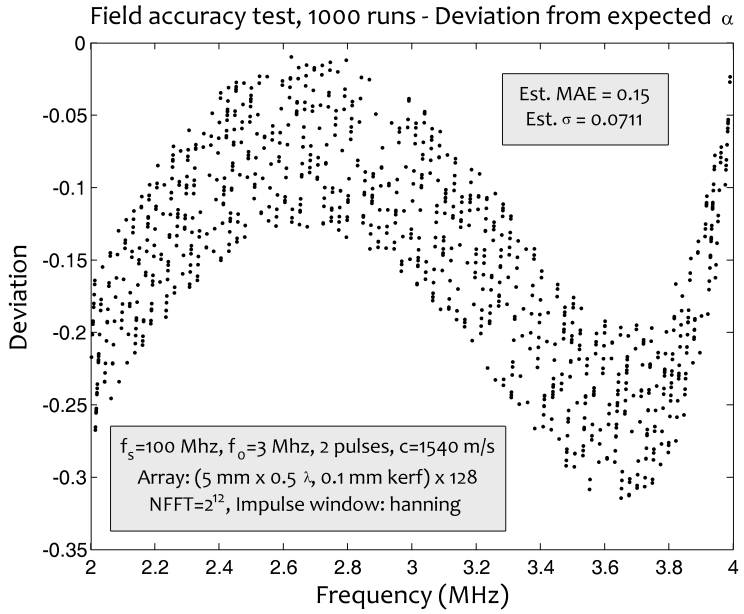


Figure 6: Deviation from expected attenuation in simulations, according to frequency. Deviation unit is dB/(cm Mhz).

To get the wanted resolution, we would need

$$40\text{Mhz}/180\text{Hz} \approx 2^{18}$$

bins. However, experiments have shown us that 2^{15} gives us good enough results with a significant speed boost compared to higher bin numbers.

4.3 The simulation framework

Because we were interested in running several types of comparisons between the various estimators, we thought it prudent to create a small framework around Field II in Matlab to facilitate the comparisons. The criteria envisioned for the framework was that it should be easily configurable, with little work involved in creating new estimators and changing simulation settings. In addition, it should be easy to identify the settings used to run the simulation from a plotted graph. We ended up with three parts:

- A configuration file for program settings, **sim_config.m**
- An estimator testing function, **test_estimator.m**
- Comparison tests comparing the MAE of various estimators

The estimator testing function runs a given estimator a number of times within a certain depth range, for a certain range of α. On completing these runs, a 3D scatterplot of errors are created, annotated with the mean absolute error and standard deviation. Vectors of errors, depths and alphas are returned in case further analysis should be required. The estimator to use is passed to the function as a parameter. The other settings are specified in the configuration file.

The estimator test function and the comparison tests all accept function handles as parameters, making it easy to extend the framework with new estimators. For instance, if you would like to compare the logpsd and burg estimators for a range of noise levels, you would run

```
noise_comparison({@log_estimator, @burg_estimator})
```

Adding a new estimator is relatively simple. You need to create a Matlab function taking two signals as parameters, the shallow reflection and the deep reflection. The first thing the estimator needs to do is declare which configuration settings from **sim_config.m** it requires, using the Matlab **global** statement. For instance, the depth difference between these two is stored in the global variable

SIM_DELTAZ. This value will most likely need doubling in the estimator, as we are dealing with backscattered signals.

As an example, consider a simple estimator **guess_estimator.m**:

```
function estimate = guess_estimator(s1, s2)

global SIM_MINALPHA SIM_MAXALPHA
estimate = SIM_MINALPHA + rand(1) * (SIM_MAXALPHA-SIM_MINALPHA);
```

This particular estimator has the advantage of being very fast, conceptually simple, and more accurate than other estimators under a number of circumstances.

4.4 Simulation framework construction

The internal process of running an estimator test is as follows:

1. The emit and receive apertures are initialized using the **create_aperture** helper function.
2. The reflector phantoms are created using the **create_phantom** helper function.
3. The helper function **simulate_backscatter** is called to set up the Field II parameters and run the backscatter simulation.
4. Finally, **find_bs_pulse** is called to "cut out" the two signals from the backscatter data and the estimator function is called.

The **create_aperture** function is very simple, just a wrapper around the Field II aperture creation function that takes the configuration settings into account.

create_phantom sets up the "phantom reflectors". They are created as evenly weighted point scatterers. To better emulate the scattering characteristics of soft tissues this should ideally be made into a single, larger phantom containing a set of randomly weighted scatterers. However, determining a good size for this phantom and choosing the right number of point scatterers to include in it is not necessarily a simple task.

simulate_backscatter is again a simple function. One minor detail is that it sets up the frequency dependent and frequency independent attenuation parameters in the form that Field II requires [Jensen, 1996]. The frequency dependent term is linearized through the center frequency of our transmitted pulse, as was done

in the examples in the Field II Users' Guide, however as α isn't fixed in our simulations we need to recalculate these parameters for every backscatter call.

find_bs_pulse naïvely cuts out a bit of the backscattered signal from the depth we're interested in. Let's call this received pulse *bp*. The length of *bp*—that is, the length of what we cut out from the backscattered signal—is currently hardcoded to four transmitted pulse lengths, to compensate for the increased length that comes from the convolution with the transmitter's impulse response. Ideally this should be made into a configurable parameter at a later stage, as some estimators work better with different lengths. The same goes for the windowing function, which is not applied at the moment - i.e. a rectangular window is used. Most likely applying a Hanning window and slightly shorter window length here would improve the performance of the AR estimators—initial experiments confirm this.

5 Simulations

5.1 The log spectral difference estimator

The first estimator we implemented was the log spectral difference estimator. As stated earlier, it is based on fitting a line to the difference between two signals at different depths. For an illustration, see Figure 7. For our purposes, we have hardcoded the starting and ending points of the line fitting to $\pm$ 0.3 MHz of the emitted signal's centre frequency. This simple method gives remarkably good results in a simulation environment.

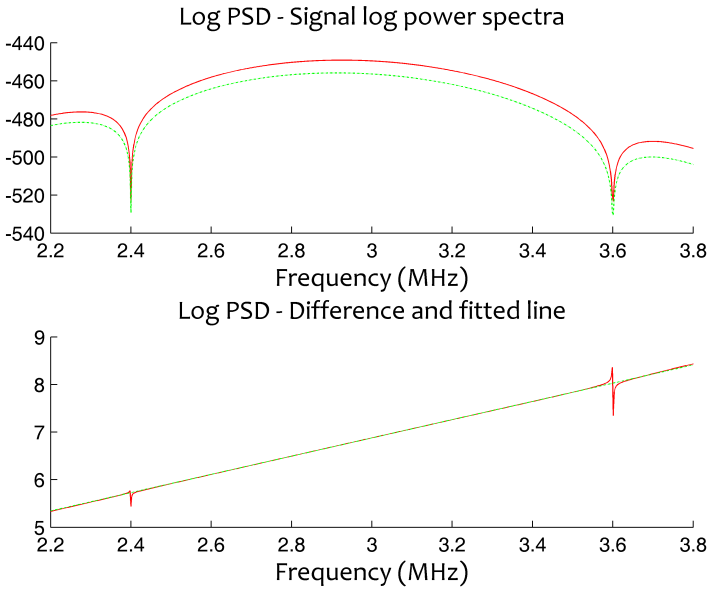


Figure 7: Calculating the log spectral difference. y-axis is in dB.

Running this estimator through our test framework, we found a mean absolute error of only 0.01 dB/(cm Mhz), with a standard deviation of about 0.009 dB/(cm Mhz). The error plot (Figure 8a) looks evenly distributed over the whole tested range.

5.2 Center frequency estimation

Secondly, we implemented estimators based on center frequency shift measurement. For the centre frequency detection, we settled on a trifecta of AR based

methods - Burg's algorithm, the modified covariance method and Yule-Walker. The reason for choosing these three algorithms in particular is that they have been used earlier in attenuation estimation, and have been shown to perform similarly [Wear et al., 1995]. The results of testing can bee seen in Figure 8b, 8c and 8d.

When implementing the AR based estimators, we needed to know the variance—or bandwidth—of the emitted pulse. We decided to estimate this from simulations, running the AR simulations "in reverse", estimating the variance from a known α. The code to do this is in **estimate_variance.m**.

5.3 Noise tolerance

After having run an initial set of simulations to confirm that our estimators were working well, we set out to measure their tolerance to noise. We added white noise to the received signal, and set out to find the level at which the estimators would start to break down.

We considered two types of noise level references - an absolute reference, and a relative reference. For the absolute reference, we used the received signal level in the centre of our simulation field (depth 70 mm, α 0.7 dB/(cm Mhz)). The relative noise reference is the received signal level at the site of simulation. The absolute noise reference is intended as a simulation of systemic noise - for instance electrical noise influencing the ultrasonic transducer. The relative noise reference is meant to represent noise introduced at the site of reflection.

The AR based estimators all performed similarly, with degradation setting in around -70 dB using the relative reference. A comparative plot can be seen in Figure 9a. A similar plot for the log PSD estimator is in Figure 9b—we separated the plots to give some idea of the degradation in the log PSD method, otherwise it would disappear as a thin line in the bottom of the AR methods plot. Looking at the plot of the estimator based on Burg's algorithm for a relative noise level of -58 dB in Figure 9c, one can see that it starts to underestimate the coefficients for larger actual values of α. This is in stark contrast to the log spectral difference method, which only degrades to similar results at much higher noise levels, around -26 dB. A plot of this can be seen in Figure 9d. This is perhaps to be expected, as white noise should be spectrally flat, introducing no major artifacts into the spectral difference between shallow and deep measurements. One might assume that the AR based estimators would also be more reistant against these types of error, as they are modeling a process driven by white gaussian noise. However, if we look at Equation 9, which tells us how the

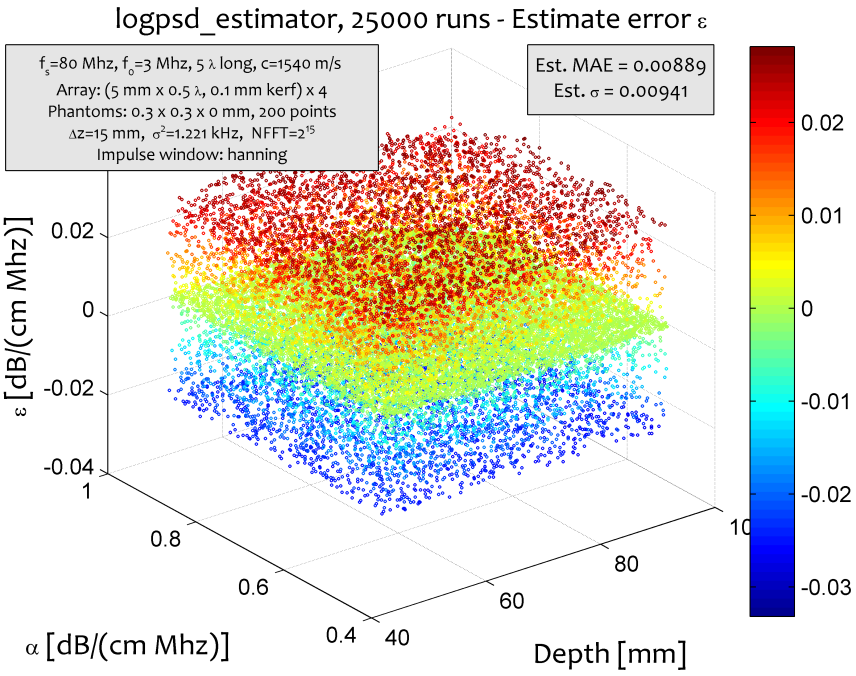
power spectrum for an AR model is calculated, we notice that the magnitude of the power spectrum increases with the variance of the noise [Clecom, 2009]. This might be a factor in the low error tolerance we are observing.

When it comes to absolute error performance, the results underline what we saw in the relative performance test. The AR methods all perform equally badly, while the log PSD method shines. The mean absolute error for the AR methods is much too high for comfort already at -80 dB, where the log PSD method still produces usable results. The MAE rates at different noise levels can be seen in Figure 10a and 10b. To illustrate just how badly the methods start to fail when errors start to creep in, plots of the Burg's algorithm-based estimator and the log PSD-based estimator at an absolute noise level of -54 dB can be found in Figure 10c and 10d.

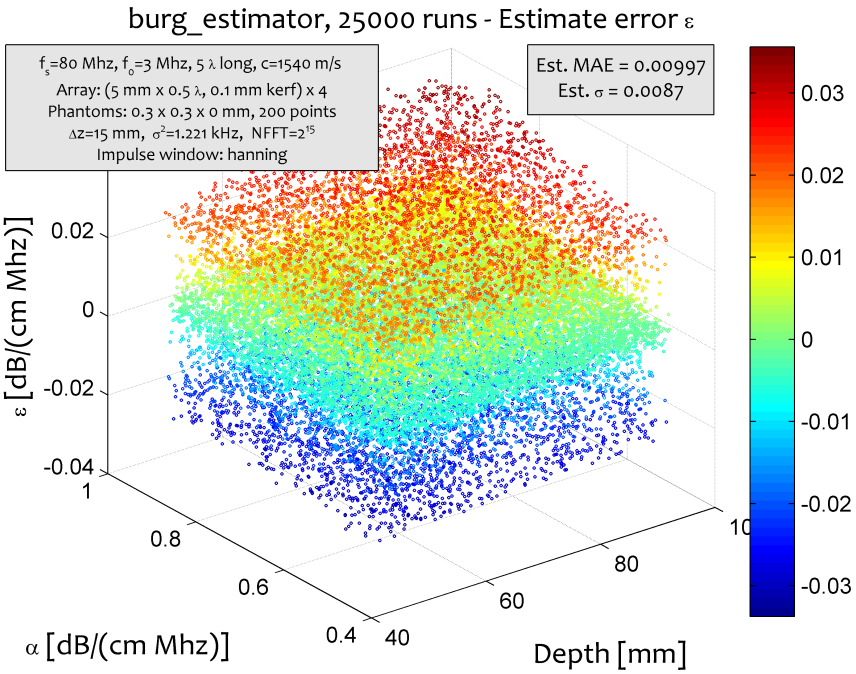
One point to be made here, however, is that the error here is added after summing the individual transducer elements, as the signal we get from Field II is already added together. It is possible get the individual element data from Field II, however this incurs a significant performance penalty. Had we applied the noise at each individual element before summing, the error performance would probably have been a lot better, as the white noise would have been reduced by a factor proportionate to the number of elements.

5.4 Irregularity tolerance

As stated earlier, we had a lot of problems with getting Field II to work consistently with backscattered signals. Eventually we figured out that the only way to get adequate results was to work with very thin phantoms, well below one micrometer thick. Because of this we thought it could be interesting to see how the different estimators would cope with different phantom widths, to introduce irregularities in the way signals are reflected. We ran a series of tests on the different estimators with phantom depths varying from 0 to 200 nanometers, or just below half a wavelength. What this showed was that the AR based estimators performed similarly, but the log spectral difference method was less tolerant, giving more errors as the phantom thickness increased. This is because the randomly placed point scatterers that make up the phantoms introduce irregularities in the received spectra that can be quite severe, disturbing the difference line enough to throw off the estimator significantly in some cases. However, the disparities between the estimators only started increasing significantly at MAEs more than 0.1, which can already be considered broken for our purposes. The comparison chart can be seen in Figure 11.

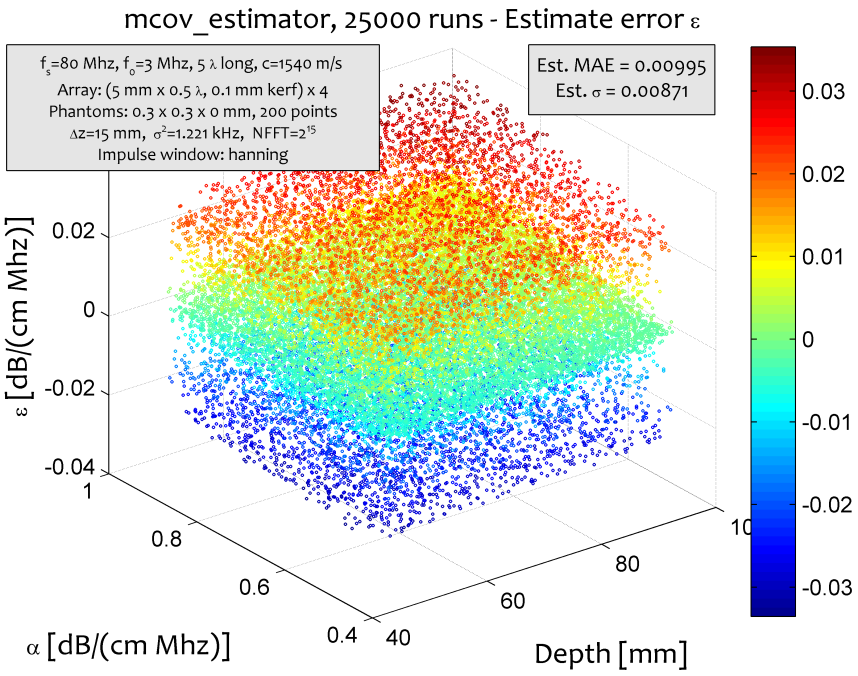


(a) Log spectral difference estimator results

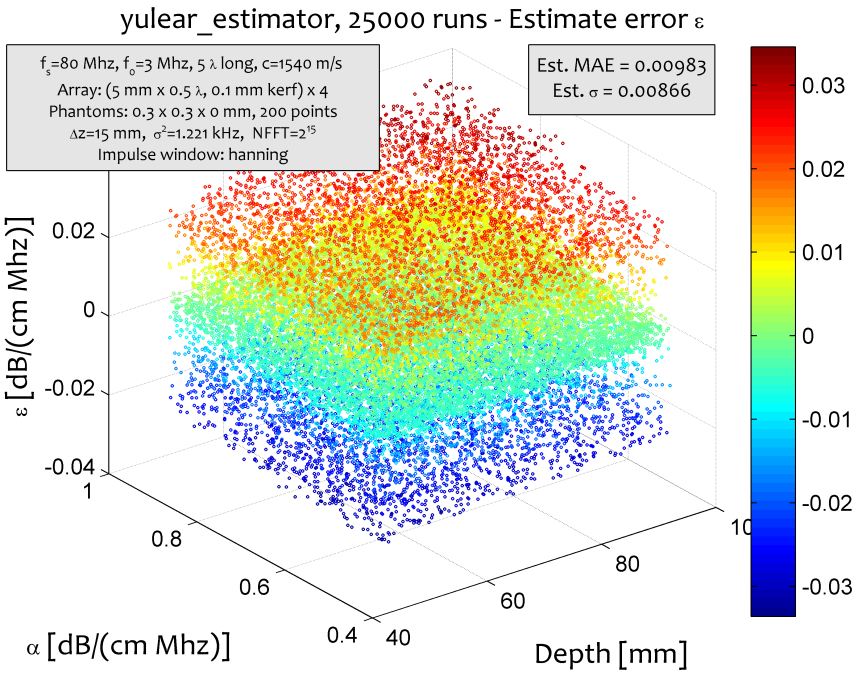


(b) Burg AR estimator results

Figure 8: Estimator test results

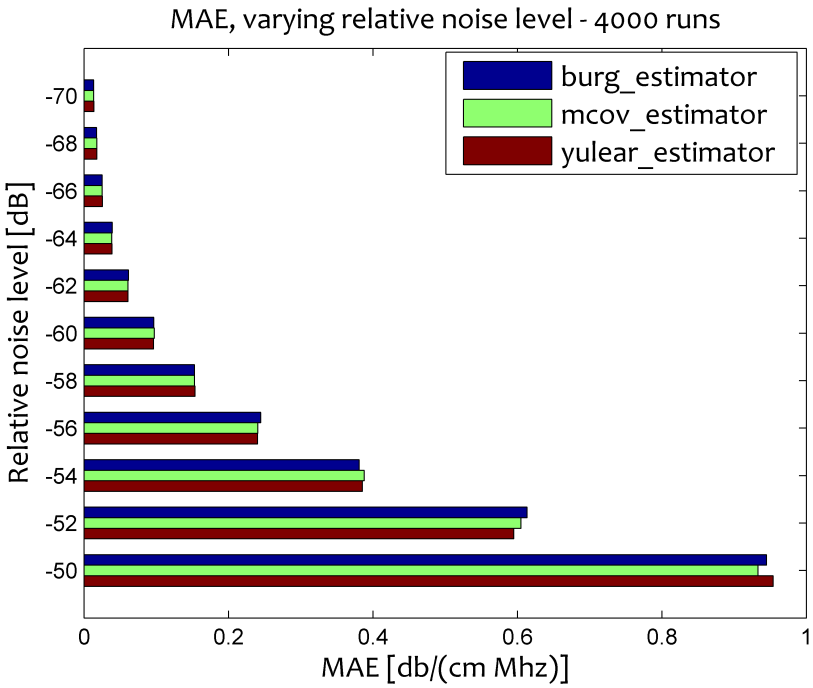


(c) Modified covariance AR estimator results

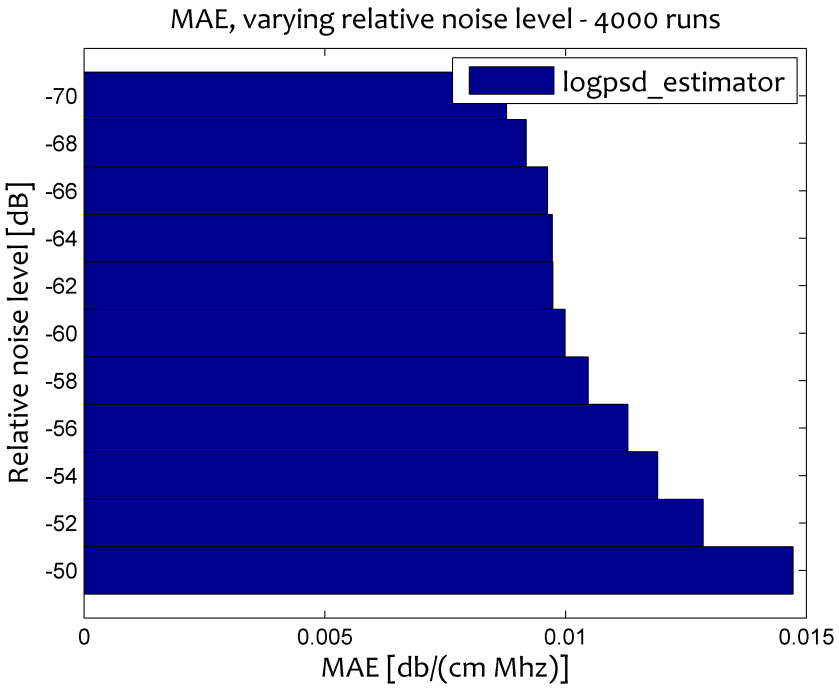


(d) Yule-Walker AR estimator results

Figure 8: Estimator test results, cont.

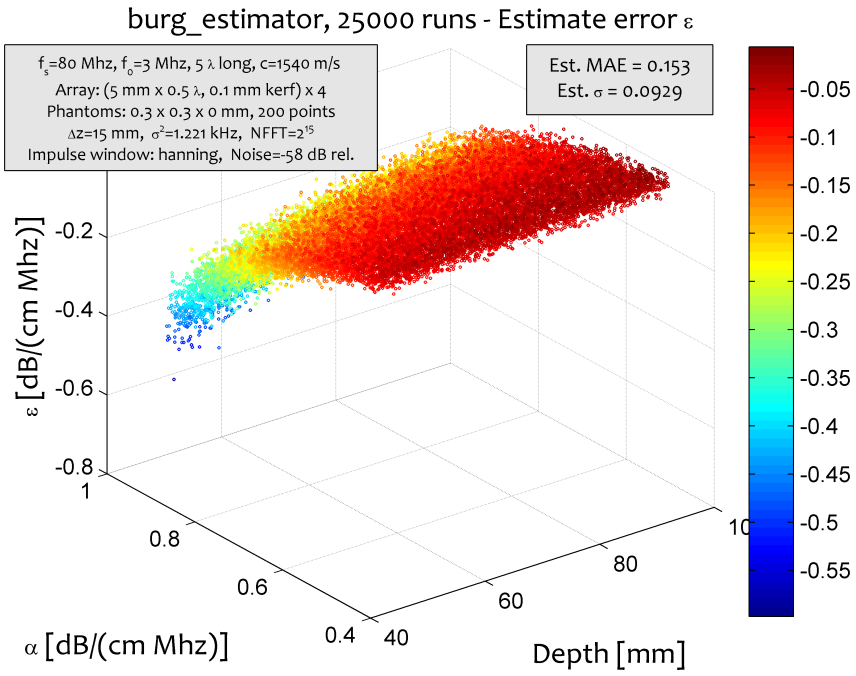


(a) Relative noise performance of AR based estimators

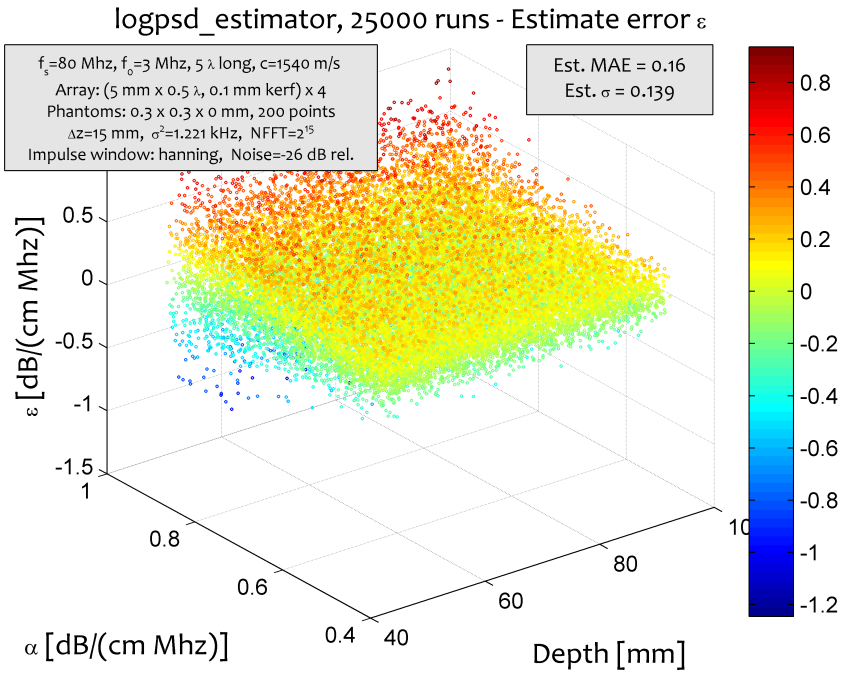


(b) Relative noise performance of log PSD estimator

Figure 9: Relative noise performance

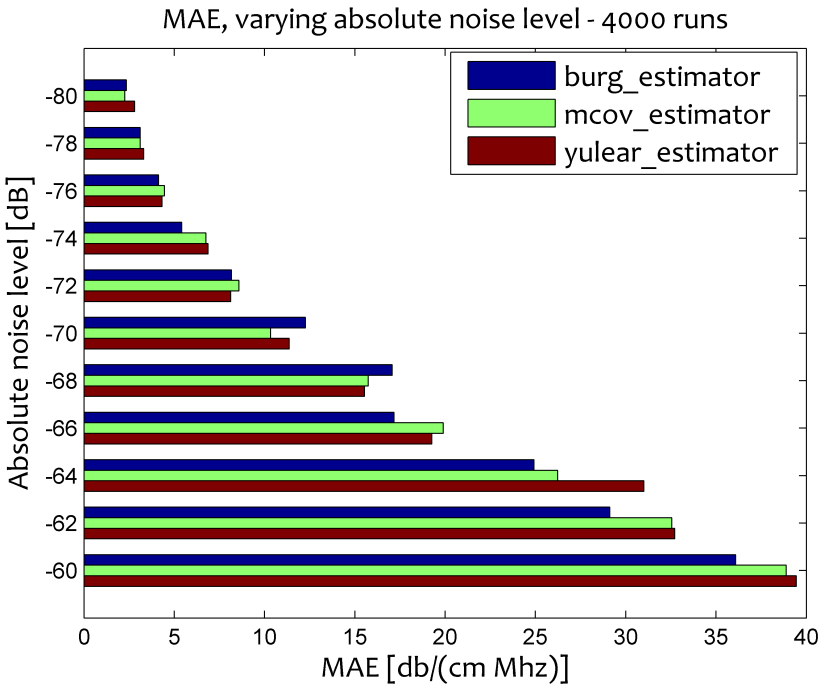


(c) Relative noise plot of Burg's algorithm

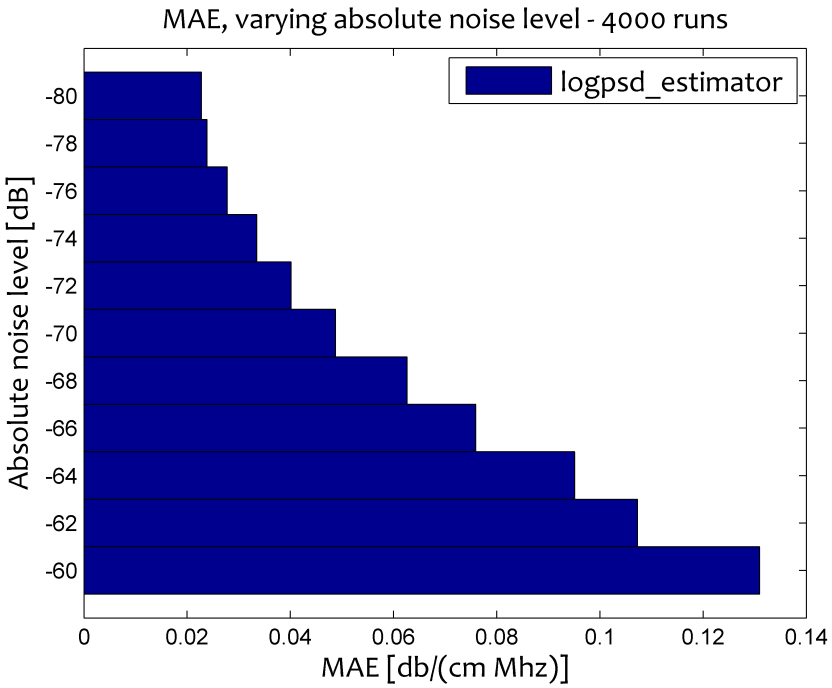


(d) Relative noise plot of log PSD estimator

Figure 9: Relative noise performance, cont.

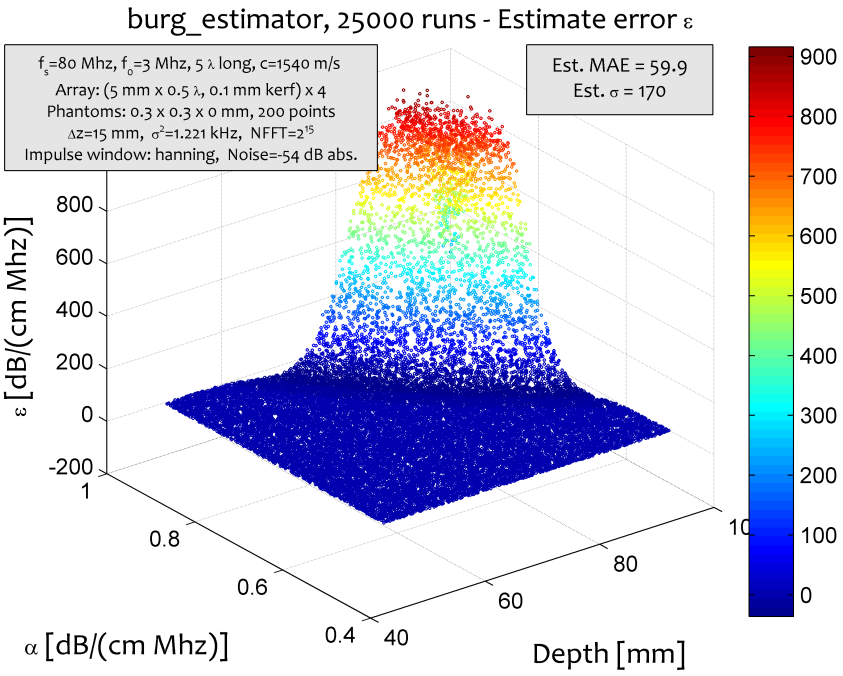


(a) Absolute noise performance of AR based estimators

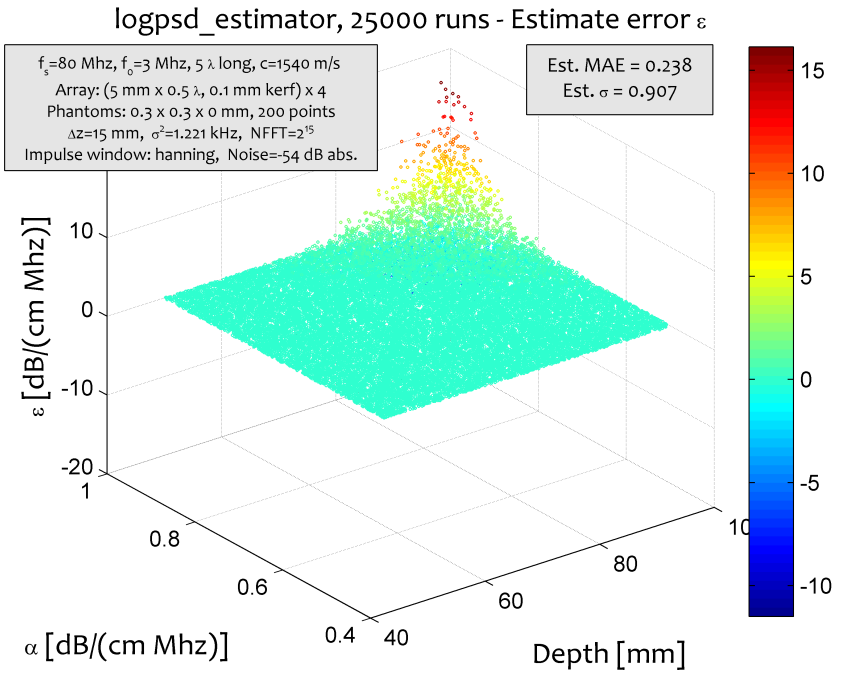


(b) Relative noise performance of log PSD estimator

Figure 10: Relative noise performance



(c) Absolute noise plot of Burg's algorithm



(d) Absolute noise plot of log PSD estimator

Figure 10: Absolute noise performance, cont.

5.5 Gate length tolerance

In the article by Wear et al. [1995], it was found that AR methods performed better at short gate lengths than the traditional DFT method. We thought it would be interesting to try to reproduce these results in a simulation environment. After running some experiments with progressively shorter gate lengths—that is, reducing Δz—we could initally not find any such discrepancies. However, we discovered that the AR methods could be made usable at shorter gate lengths through windowing the received pulses with a Hanning window, whereas this did not help as much for the spectral difference method. Windowing also decreased the overall accuracy of the log PSD method, but didn't affect the AR methods performance noticably. Actually applying the Hanning window to the received pulses hurt the accuracy of the log PSD method overall, even though it improved it slightly for short gate lengths. For the AR the windowing had a noticably positive effect, especially after changing the variance parameter to compensate for the smaller bandwith of the received pulses. Positioning the phantoms only 3 mm apart, where the two pulses are noticably overlapping on a plot before windowing, the Burg AR method had an MAE of 0.03 dB/(cm Mhz), while the log PSD method measured around 0.25 dB/(cm Mhz).

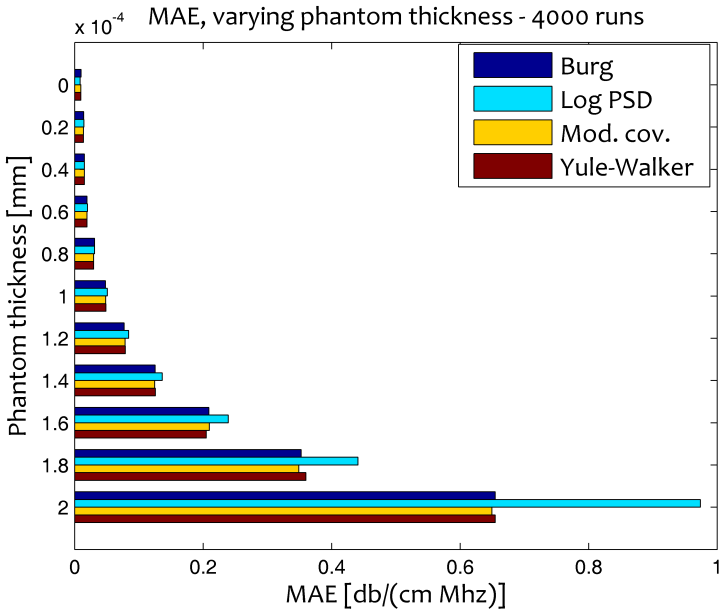


Figure 11: Estimator MAE for various thicknesses. The unit of the deviation is dB/(cm Mhz).

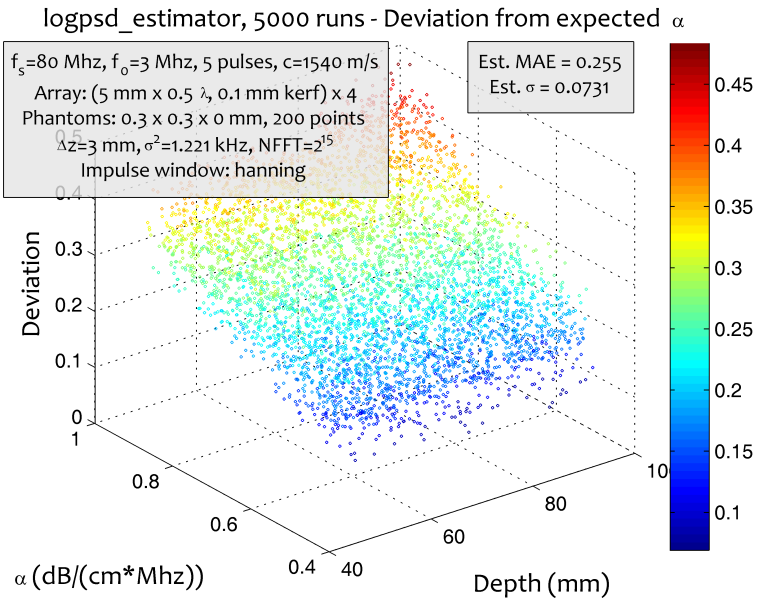


Figure 12: Log PSD estimator performance with $\Delta z = 3mm$. Deviation unit is dB/(cm Mhz).

6 Conclusion

It is not by chance that many of the papers referenced in this article have chosen to include, either in their introduction or in their closing remarks, a sighing note about the difficulties they have encountered. I will proceed to join in their chorus. Attenuation estimation is indeed a difficult subject. and all of the estimation methods have some tradeoffs. First of all, attenuation can only be accurately measured with the transmission method. The backscatter methods that are most useful for *in vivo* measurements are only estimates, and will never be completely accurate. This is especially true seeing as most of the attenuation in soft tissues is caused by absorption, while all of the information used for the estimate comes from scattering. A change in scattering could thus lead to a large change in the estimates while overall attenuation could stay near constant.

Our attempts at implementing a proper comparative simulation framework using backscatter have mirrored the difficulties that arise in the real world with these estimations. It is obvious that more measures need to be taken to alleviate the spectrum changes that occur when larger areas of randomly spread scatterers exist. There is also the question of whether the groups of distributed point scatterers that Field II uses as analogues for soft tissue phantoms accurately reproduce the scattering that would occur in real soft tissues. Clearly more research is needed.

As far as we have been able to test our selection of estimation methods, performance seem to be similar. In fact, you can say that they have performed similarly badly in backscatter simulations, all breaking down when the thickness of our scatterer collections increase beyond a few hundred nanometers. From our experiments it would seem that the log spectral difference method has an advantage over the AR methods in that it is much more resistant to white noise. If our theory about the role of the noise variance in the bad error performance of the AR algorithms is correct, however, this might be improved by somehow normalizing the spectrum depending on the noise. Despite the AR methods being time domain methods, they also perform slower than our log spectral difference implementation as it now stands. This also has potential for improvement, however, for instance by implementing the center frequency calculation optimization mentioned in Girault's paper cited earlier [Girault et al., 1998].

In the simulation framework that we have created, there are also several possibilities for improvement. A good start would be to rework the way configuration files are stored and parsed. The current way of parsing configuration files while running estimators makes it cumbersome to run simulations with differ-

ent parameters alongside each other, for instance to take advantage of multiple processors or processor cores on a workstation. A good way to proceed forward could be to use a structure for the options instead of global variables, and passing this structure to the simulation methods as a parameter. The existing configuration files could be used as default settings in case no configuration parameter was provided.

There are also other settings that could be included, which the program as it stands lacks. As commented on in Section 5, a choice of windowing function for the received signals would be a welcome addition. Preferably this should be a parameter of the estimation function, to facilitate easy comparisons of methods where the ideal windowing function for each method might differ.

What is perhaps the most important part to improve, however, are the simulated phantoms. As it is now, the "phantom reflectors" we have created are not a good analogue for the way the signal would be scattered and dispersed by the inhomogenous media—our vital organs—inside the human body. It would be preferable to use larger phantoms, preferably one phantom that is large enough to include both measurement points. A place to start might be to try to improve the existing methods, for instance by adding filtering or averaging, to the point where they can be used for a thicker, more irregular phantom than what is possible now.

7 Errata

Unfortunately, much like the estimates we have studied this thesis is not error-free. A bug has found its way into the variance estimator and the AR based estimators, so the unit of the pulse bandwidth SIM_SIGMA2 is off by one decimal digit. Plot annotations which read $\sigma^2 = 1.221$ kHz are thus actually supposed to read $\sigma^2 = 12.21$ kHz. We apologize for the mistake.

Bibliography

T. Baldeweck, A. Herment, P. Laugier, and G. Berger. Attenuation estimation in highly attenuating media using high frequencies: a comparison study between different mean frequency estimators. In *Ultrasonics Symposium, 1994. Proceedings., 1994 IEEE*, volume 3, pages 1783–1786, 1-4 Nov. 1994.

Clecom. Ar (autoregressive) spectrum. `http://www.clecom.co.uk/science/autosignal/help/AR_(AutoRegressive)_Spectru.htm`, 05 2009.

Spartaco Colombati and Stefano Petralia. Assorbimento di ultrasuoni in tessuti animali. *La Ricerca Scientifica*, 20(1-2):71–78, 1950.

F. Dunn. Ultrasonic attenuation, absorption, and velocity in tissues and organs. In Melvin Linzer, editor, *Ultrasonic Tissue Characterization*, pages 21–29, Washington, D. C., 1976. National Bureau of Standards.

R. Esche. Untersuchungen zur Ultraschallabsorption in tierischen Geweben und Kunststoffen. *Akustische Beihefte*, 2:71–74, 1952.

J.-M. Girault, F. Ossant, A. Ouahabi, D. Kouame, and F. Patat. Time-varying autoregressive spectral estimation for ultrasound attenuation in tissue characterization. *Ultrasonics, Ferroelectrics and Frequency Control, IEEE Transactions on*, 45(3):650–659, May 1998. ISSN 0885-3010.

Philip S. Green and Jon C. Taenzer. Compact ultrasound apparatus for medical examination. US Patent, 1984.

Monson H. Hayes. *Statistical Digital SIgnal Processing and Modeling*. John Wiley & Sons, Inc., 1996.

P. He and J. F. Greenleaf. Application of stochastic analysis to ultrasonic echoes–estimation of attenuation and tissue heterogeneity from peaks of echo envelope. *J Acoust Soc Am*, 79(2):526–534, Feb 1986.

H. S. Jang, T. K. Song, and S. B. Park. Ultrasound attenuation estimation in soft tissue using the entropy difference of pulsed echoes between two adjacent envelope segments. *Ultrason Imaging*, 10(4):248–264, Oct 1988.

J.A. Jensen and N. B. Svendsen. Calculation of pressure fields from arbitrarily shaped, apodized, and excited ultrasound transducers. *IEEE Trans. Ultrason., Ferroelec., Freq. Contr.*, 39:262–267, 1992.

Jørgen Arendt Jensen. Field: A program for simulating ultrasound systems. *Paper presented at the 10th Nordic-Baltic Conference on Biomedical Imaging, Published in Medical & Biological Engineering & Computing*, 34, Supplement 1, Part 1:351–353, 1996.

Jørgen Arendt Jensen. *User's Guide for the Field II program*. Technical University of Denmark, 2001. URL `http://server.oersted.dtu.dk/personal/jaj/field/documents/users_guide.pdf`. Release 2.86, August 17, 2001.

S.M. Kay and Jr. Marple, S.L. Spectrum analysis—a modern perspective. *Proceedings of the IEEE*, 69(11):1380–1419, Nov. 1981. ISSN 0018-9219.

P. P. Lele, A. B. Mansfield, A. I. Murphy, J. Namery, and N. Senapati. Tissue characterization by ultrasonic frequency-dependent attenuation and scattering. In Melvin Linzer, editor, *Ultrasonic Tissue Characterization*, pages 167–196, Washington, D. C., 1976. National Bureau of Standards.

M. Linzer and S. J. Norton. Ultrasonic tissue characterization. *Annu Rev Biophys Bioeng*, 11:303–329, 1982.

Melvin Linzer, editor. *Ultrasonic Tissue Characterization*, Washington, D. C., 1976. National Bureau of Standards.

Charles R. Meyer. An iterative, parametric spectral estimation technique for high-resolution pulse-echo ultrasound. *Biomedical Engineering, IEEE Transactions on*, BME-26(4):207–212, April 1979. ISSN 0018-9294.

J.G. Miller, D.E. Yuhas, J.W. Mimbs, S.B. Dierker, L.J. Busse, J.J. Laterra, A.N. Weiss, and B.E. Sobel. Ultrasonic tissue characterization: Correlation between biochemical and ultrasonic indices of myocardial injury. In *Ultrasonics Symposium, 1976*, pages 33–43, 1976.

J.G. Miller, J.E. Perez, J.G. Mottley, E.I. Madaras, P.H. Johnston, E.D. Blodgett, III Thomas, L.J., and B.E. Sobel. Myocardial tissue characterization: An approach based on quantitative backscatter and attenuation. In *Ultrasonics Symposium, 1983*, pages 782–793, 1983.

R A Mountford and P N T Wells. Ultrasonic liver scanning: the quantitative analysis of the normal a-scan. *Phys. Med. Biol.*, 17(1):14–25, January 1972.

P. A. Narayana and J. Ophir. Spectral shifts of ultrasonic propagation: A study of theoretical and experimental models. *Ultrason Imaging*, 5(1):22–29, Jan 1983.

J. Ophir, T. H. Shawker, N. F. Maklad, J. G. Miller, S. W. Flax, P. A. Narayana, and J. P. Jones. Attenuation estimation in reflection: progress and prospects. *Ultrason Imaging*, 6(4):349–395, Oct 1984.

J. Ophir, M. A. Ghouse, and L. A. Ferrari. Attenuation estimation with the zero-crossing technique: phantom studies. *Ultrason Imaging*, 7(2):122–132, Apr 1985.

Emmanuel P. Papadakis. *Ultrasonic instruments and devices*. Academic Press, 1999.

R. Pohlman. Über die Absorption des Ultraschalls im menschlichen Gewebe und ihre Abhängigkeit von der Frequenz. *Physikalische Zeitschrift*, 40: 159–161, 1939.

John G. Proakis and Dimitris G. Manolakis. *Digital Signal Processing Principles, Algorithms, and Applications*. Prentice-Hall International, Inc., Third edition, 1996.

K.A. Wear, R.F. Wagner, and B.S. Garra. A comparison of autoregressive spectral estimation algorithms and order determination methods in ultrasonic tissue characterization. *Ultrasonics, Ferroelectrics and Frequency Control, IEEE Transactions on*, 42(4):709–716, July 1995.

P. N. T. Wells, R. A. Mountford, M. Halliwell, and P. Atkinson. Quantitative a-scan analysis of normal and cirrhotic liver. In Melvin Linzer, editor, *Ultrasonic Tissue Characterization*, pages 61–70, Washington, D. C., 1976. National Bureau of Standards.

L. S. Wilson, D. E. Robinson, K. A. Griffiths, A. Manoharan, and B. D. Doust. Evaluation of ultrasonic attenuation in diffuse diseases of spleen and liver. *Ultrason Imaging*, 9(4):236–247, Oct 1987.

A Source code

A.1 Configuration and helper functions

A.1.1 calculate_freqdep_att.m

```
function freq_att = calculate_freqdep_att(alpha)

freq_att = alpha*100/1e6;
```

A.1.2 calculate_indep_att.m

```
function indep_att = calculate_indep_att(alpha)

sim_config;
global SIM_F0;

indep_att = alpha*(SIM_F0/1e6)*100;
```

A.1.3 create_aperture.m

```
function aperture = create_aperture()

sim_config;
global SIM_C SIM_F0 SIM_ELWIDTH SIM_FS SIM_PULSES SIM_ELEMENTS ...
    SIM_ELHEIGHT SIM_ELKERF SIM_WINDOW;

lambda = SIM_C/SIM_F0; % Wavelength [m]
width = SIM_ELWIDTH * lambda;

% Set the sampling frequency
set_sampling(SIM_FS);
% Set the impulse response and excitation of the emit aperture
impulse_response = sin(2*pi*SIM_F0*(0:1/SIM_FS:SIM_PULSES/SIM_F0));
impulse_response = impulse_response .* ...
    feval(SIM_WINDOW, max(size(impulse_response)))';

excitation = sin(2*pi*SIM_F0*(0:1/SIM_FS:SIM_PULSES/SIM_F0));
% The focus will be adjusted during simulations, but we need an initial
% dummy value as it is a required parameter.
focus = [0 0 50];
aperture = xdc_linear_array ...
    (SIM_ELEMENTS, width, SIM_ELHEIGHT, SIM_ELKERF, 1, 1, focus);
xdc_impulse (aperture, impulse_response);
xdc_excitation (aperture, excitation);
```

A.1.4 create_phantom.m

```
function [points, amplitudes] = create_phantom(depth)

sim_config;
global SIM_PHANTOM_POINTS SIM_PHANTOM_X SIM_PHANTOM_Y SIM_PHANTOM_Z;

N=SIM_PHANTOM_POINTS; % Number of scatterers per phantom
z_start = depth/1000; % Start of phantom 1 surface [mm];

% Create the general scatterers
x = (rand(N,1)-0.5) * SIM_PHANTOM_X;
y = (rand(N,1)-0.5) * SIM_PHANTOM_Y;
z = z_start + (rand(N,1)-0.5) * SIM_PHANTOM_Z;

points = [x y z];
amplitudes = ones(N,1);
```

A.1.5 estimate_variance.m

```
function [variance, stddev] = estimate_variance(estimator)

sim_config;
global SIM_MINALPHA SIM_MAXALPHA SIM_MINDEPTH SIM_MAXDEPTH;
global SIM_FS SIM_DELTAZ SIM_EXPERIMENTS SIM_NFFT;

emit_aperture = create_aperture();
receive_aperture = create_aperture();

depths = SIM_MINDEPTH + (SIM_MAXDEPTH-SIM_MINDEPTH) ...
    .* rand(SIM_EXPERIMENTS, 1);
alphas = SIM_MINALPHA + (SIM_MAXALPHA-SIM_MINALPHA) ...
    .*rand(SIM_EXPERIMENTS, 1);
variances = zeros(SIM_EXPERIMENTS, 1);

tic
for idx=1:SIM_EXPERIMENTS
    % Define two small phantoms with scatterers
    [p1, a1] = create_phantom(depths(idx));
    [p2, a2] = create_phantom(depths(idx) + SIM_DELTAZ);
    points = [p1; p2];
    amplitudes = [a1; a2];

    [s, t] = simulate_backscatter(emit_aperture, ...
        receive_aperture, alphas(idx), depths(idx), points, amplitudes);

    s1 = find_bs_pulse(s, t, depths(idx));
    s2 = find_bs_pulse(s, t, depths(idx) + SIM_DELTAZ);
```

```
    max1 = max(length(s1), length(s2));
    nfft = max(SIM_NFFT, 2^(nextpow2(max1)));

    [psd1, f1] = pburg(s1, 2, nfft, SIM_FS);
    [psd2, f2] = pburg(s2, 2, nfft, SIM_FS);
    cf1 = sum(f1 .* psd1) / sum(psd1);
    cf2 = sum(f2 .* psd2) / sum(psd2);

    variances(idx) = (cf1 - cf2) / (alphas(idx) * SIM_DELTAZ * 1000);

    if (mod(idx, floor(SIM_EXPERIMENTS/100)) == 0)
        disp([num2str((idx/SIM_EXPERIMENTS)*100) '% done']);
    end
end
toc

figure;
scatter3(depths, alphas, variances, 2, variances);
caxis('auto');
h = xlabel('Depth (mm)');
set(h, 'FontName', 'Candara');
h = ylabel('\alpha (dB/(cm*Mhz))');
set(h, 'FontName', 'Candara');
h = zlabel('Deviation');
set(h, 'FontName', 'Candara');
h = title(['Variance estimation, ' ...
    num2str(SIM_EXPERIMENTS, '%i') ' runs -Estimated variance']);
set(h, 'FontName', 'Candara');
annotate_graph(variances(:));
colorbar;

variance = median(variances(:));
stddev = std(variances(:));
```

A.1.6 find_bs_pulse.m

```
function p = find_bs_pulse(s, t, depth)

sim_config;
global SIM_C SIM_FS SIM_F0 SIM_PULSES
pulselength = floor(4*SIM_PULSES*SIM_FS/SIM_F0);

pos = floor((((depth*2)/(1000*SIM_C)) -t) * SIM_FS)+1;
p = s(max(pos,1):min(length(s),pos+pulselength));
% wdow = hanning(length(p))';
% p = p .* wdow;
```

A.1.7 logpsd.m

```
function [mx, f] = logpsd(x, fs, nfft)
% Based on code in Matlab Technote 1702, "Using FFT to Obtain Simple
% Spectral Analysis Plots"
% http://www.mathworks.com/support/tech-notes/1700/1702.html

Fs = fs / 1e6; % Sampling freq in mHz
% Take fft, padding with zeros so that length(fftx) is equal to nfft
fftx = fft(x,nfft);

% Calculate the numberof unique points
NumUniquePts = ceil((nfft+1)/2);
% FFT is symmetric, throw away second half
fftx = fftx(1:NumUniquePts);
% Take the magnitude of fft of x and scale the fft so that it is not a
% function of the length of x
mx = abs(fftx)/length(x);
% Take the square of the magnitude of fft of x.
mx = mx.^2;
% Since we dropped half the FFT, we multiply mx by 2 to keep the same
% energy. The DC component and Nyquist component, if it exists, are
% unique and should not be mulitplied by 2.
if rem(nfft, 2) % odd nfft excludes Nyquist point
  mx(2:end) = mx(2:end)*2;
else
  mx(2:end -1) = mx(2:end -1)*2;
end
% Take the logarithm to obtain dB scale.
mx = 10 * log10(mx + (mx==0)*eps);
% This is an evenly spaced frequency vector with NumUniquePts points.
f = (0:NumUniquePts-1)*Fs/nfft;
```

A.1.8 sim_config.m

```
function sim_config()
%SIM_CONFIG Set some global variables used in simulations

global SIM_FS SIM_F0 SIM_C;
SIM_FS = 80e6;              % Sampling rate [Hz]
SIM_F0 = 3e6;               % Centre frequency for emitted wave [Hz]
SIM_C = 1540;               % Speed of sound [m/s]

global SIM_ELEMENTS SIM_ELHEIGHT SIM_ELWIDTH SIM_ELKERF SIM_PULSES;
SIM_ELEMENTS = 4;           % Number of elements in transducer
SIM_ELHEIGHT = 5 / 1000;    % Element height [m]
SIM_ELWIDTH = 0.5;          % Element width [wavelengths]
SIM_ELKERF = 0.1 / 1000;    % Element kerf [m]
SIM_PULSES = 5;             % Number of pulses to transmit

global SIM_WINDOW;
SIM_WINDOW = @hanning;    % Window used for transmitted pulse

global SIM_SIGMA2;
SIM_SIGMA2 = 1.2212;        % Bandwidth, used for cf \alpha estimation [kHz]
%SIM_SIGMA2 = 1.1457;        % Bandwidth, used for cf \alpha estimation [kHz]

global SIM_MINALPHA SIM_MAXALPHA SIM_MINDEPTH SIM_MAXDEPTH;
SIM_MINALPHA = 0.5;         % Minimum \alpha for tests [dB/(cm*Mhz)]
SIM_MAXALPHA = 1.0;         % Maximum \alpha for tests [dB/(cm*Mhz)]
SIM_MINDEPTH = 50;          % Minimum depth for tests [mm]
SIM_MAXDEPTH = 100;         % Maximum depth for tests [mm]

global SIM_DELTAZ SIM_EXPERIMENTS SIM_NFFT;
SIM_DELTAZ = 15;            % Distance between phantoms [mm]
SIM_EXPERIMENTS = 100;   % Number of experiments to run in tests
SIM_NFFT = 2^15;            % FFT points to use for AR estimations

global SIM_PHANTOM_X SIM_PHANTOM_Y SIM_PHANTOM_Z SIM_PHANTOM_POINTS;
SIM_PHANTOM_X = 0.3 / 1000; % Width of phantoms [m]
SIM_PHANTOM_Y = 0.3 / 1000; % Height of phantoms [m]
SIM_PHANTOM_Z = 0;          % Depth of phantoms [m]
SIM_PHANTOM_POINTS = 200; % Number of scatterers per phantom

global SIM_NOISE SIM_NOISE_LEVEL SIM_PLOT SIM_SILENT SIM_NOISE_REF;
SIM_NOISE = 0;                  % Simulate noise?
SIM_NOISE_LEVEL = -58;          % Noise level [dB]
SIM_PLOT = 1;                   % Plot graphs for tests
SIM_SILENT = 0;                 % Do not output progress info
%SIM_NOISE_REF = 'absolute';   % Noise level relative to simulation centre
SIM_NOISE_REF = 'relative';    % Noise level relative to received signal
```

A.1.9 simulate_backscatter.m

```
function [signal, time] = simulate_backscatter( ...
    emit_aperture, receive_aperture, alpha, depth, points, amplitudes)

freq_att = calculate_freqdep_att(alpha);
indep_att = calculate_indep_att(alpha);

% Set attenuation according to alpha
if alpha == 0
    set_field('use_att', 0);
else
    set_field('use_att', 1);
    set_field('att', indep_att);
    set_field('Freq_att', freq_att);
end

% Generate aperture for emission
focus = [0 0 depth] / 1000;
xdc_focus(emit_aperture, 0, focus);
xdc_focus(receive_aperture, 0, focus);

% Calculate spatial response
[signal, time] = ...
    calc_scat(emit_aperture, receive_aperture, points, amplitudes);
signal = signal.';
```

A.1.10 simulate_point_measurement.m

```
function [signal, time] = ...
    simulate_point_measurement(emit_aperture, alpha, depth)

freq_att = calculate_freqdep_att(alpha);
indep_att = calculate_indep_att(alpha);

% Set attenuation according to alpha
if alpha == 0
    set_field('use_att', 0);
else
    set_field('use_att', 1);
    set_field('att', indep_att);
    set_field('Freq_att', freq_att);
end

% Generate aperture for emission
focus = [0 0 depth] / 1000;
xdc_focus(emit_aperture, 0, focus);

% Calculate spatial response
```

```
[signal, time] = calc_hp(emit_aperture, focus);
signal = signal.';
```

A.1.11 annotate_graph.m

```
function h = annotate_graph(deviations)

sim_config;
global SIM_ELEMENTS SIM_ELHEIGHT SIM_ELWIDTH SIM_ELKERF SIM_WINDOW;
global SIM_PHANTOM_X SIM_PHANTOM_Y SIM_PHANTOM_Z SIM_PHANTOM_POINTS;
global SIM_DELTAZ SIM_SIGMA2 SIM_NFFT SIM_FS SIM_F0 SIM_C SIM_PULSES;
global SIM_NOISE SIM_NOISE_LEVEL SIM_NOISE_REF;

h = annotation('textbox', [.02 .75 .40 .18]);
set(h, 'BackgroundColor', [0.9 0.9 0.9]);
%set(h, 'FaceAlpha', 0.8);
set(h, 'FontName', 'Candara');
set(h, 'HorizontalAlignment', 'Center');
set(h, 'FontSize', 6);
lastline = ['Impulse window: ' func2str(SIM_WINDOW)];
if SIM_NOISE ~= 0
    lastline = [lastline ', ' ...
    ' Noise=' num2str(SIM_NOISE_LEVEL, '%i') ' dB'];
    if strcmpi(SIM_NOISE_REF, 'relative')
        lastline = [lastline ' rel.'];
    else
        lastline = [lastline ' abs.'];
    end
end
set(h, 'String', { ...
    ['f_s=' num2str(SIM_FS/1e6, '%i') ' Mhz, ' ...
    'f_0=' num2str(SIM_F0/1e6, '%i') ' Mhz, ' ...
    num2str(SIM_PULSES, '%i') ' \lambda long, ' ...
    'c=' num2str(SIM_C, '%i') ' m/s '] ...
    ['Array: ' ...
    '(' num2str(SIM_ELHEIGHT*1000,0) ...
    ' mm x ' num2str(SIM_ELWIDTH, 0) ' \lambda, ' ...
    num2str(1000*SIM_ELKERF, 0) ' mm kerf) x ' ...
    num2str(SIM_ELEMENTS,'%i')], ...
    ['Phantoms: ' ...
    num2str(SIM_PHANTOM_X * 1000, 1) ' x ' ...
    num2str(SIM_PHANTOM_Y * 1000, 1), ' x ' ...
    num2str(SIM_PHANTOM_Z * 1000, 1), ' mm, ' ...
    num2str(SIM_PHANTOM_POINTS, '%i'), ' points'], ...
    ['\Deltaz=' num2str(SIM_DELTAZ, '%i') ' mm, '...
    ' \sigma^2=' num2str(SIM_SIGMA2, 4) ' kHz, ' ...
    ' NFFT=2^{' num2str(log2(SIM_NFFT), '%i') '}'], ...
    lastline ...
    });
```

```
h = annotation('textbox', [.58 .83 .20 .1]);
set(h, 'BackgroundColor', [0.9 0.9 0.9]);
%set(h, 'FaceAlpha', 0.8);
set(h, 'FontName', 'Candara');
set(h, 'HorizontalAlignment', 'Center');
set(h, 'FontSize', 7);
set(h, 'String', { ...
    ['Est. MAE = ' num2str(mean(abs(deviations)), 3)], ...
    ['Est. \sigma = ' num2str(std(abs(deviations)), 3)]...
    });
```

A.2 Testing functions

A.2.1 test_estimator.m

```
function [deviation, depths, alphas] = test_estimator(estimator)

sim_config;
global SIM_MINALPHA SIM_MAXALPHA SIM_MINDEPTH SIM_MAXDEPTH SIM_SILENT;
global SIM_DELTAZ SIM_EXPERIMENTS SIM_NOISE SIM_NOISE_LEVEL SIM_PLOT;
global SIM_NOISE_REF;

emit_aperture = create_aperture();
receive_aperture = create_aperture();

depths = SIM_MINDEPTH + (SIM_MAXDEPTH-SIM_MINDEPTH) ...
    .* rand(SIM_EXPERIMENTS, 1);
alphas = SIM_MINALPHA + (SIM_MAXALPHA-SIM_MINALPHA) ...
    .*rand(SIM_EXPERIMENTS, 1);
deviation = zeros(SIM_EXPERIMENTS, 1);

if SIM_NOISE && ~strcmpi(SIM_NOISE_REF, 'relative')
    halfway = SIM_MINDEPTH + ((SIM_MAXDEPTH -SIM_MINDEPTH)/2);
    midalpha = SIM_MINALPHA + ((SIM_MAXALPHA -SIM_MINALPHA)/2);
    % Use most shallow, least attenuated signal as reference level
    [p1, a1] = create_phantom(halfway);
    [p2, a2] = create_phantom(halfway + SIM_DELTAZ);
    [s, t] = simulate_backscatter(emit_aperture, ...
            receive_aperture, midalpha, halfway, p1, a1);
    sigpow = mean(s.^2);
    noiselvl = sqrt(10^(SIM_NOISE_LEVEL/10) * sigpow);
end

tic
for idx=1:SIM_EXPERIMENTS
    % Define two small phantoms with scatterers
    [p1, a1] = create_phantom(depths(idx));
    [p2, a2] = create_phantom(depths(idx) + SIM_DELTAZ);
    points = [p1; p2];
    amplitudes = [a1; a2];

    [s, t] = simulate_backscatter(emit_aperture, ...
        receive_aperture, alphas(idx), depths(idx), points, amplitudes);
    if (SIM_NOISE ~= 0)
        if strcmpi(SIM_NOISE_REF, 'relative')
            % Use received signal as reference level
            sigpow = mean(s.^2);
            noiselvl = sqrt(10^(SIM_NOISE_LEVEL/10) * sigpow);
        end
```

```
        noise = randn(size(s)) * noiselvl;
        s = s + noise;
    end

%    plot(s)
%    error('Halting for debug plot');

    s1 = find_bs_pulse(s, t, depths(idx));
    s2 = find_bs_pulse(s, t, depths(idx) + SIM_DELTAZ);

%    figure;
%    plot(s1)
%    figure;
%    plot(s2)
%    error('Halting for debug plot');

    deviation(idx) = alphas(idx) -feval(estimator, s1, s2);
    if ~SIM_SILENT
        if (mod(idx, floor(SIM_EXPERIMENTS/10)) == 0)
            disp([num2str((idx/SIM_EXPERIMENTS)*100) '% done']);
        end
    end
end
toc

if SIM_PLOT
    figure;
    scatter3(depths, alphas, deviation, 2, deviation);
    caxis('auto');
    h = xlabel('Depth [mm]');
    set(h, 'FontName', 'Candara');
    h = ylabel('\alpha [dB/(cm Mhz)]');
    set(h, 'FontName', 'Candara');
    h = zlabel('\epsilon [dB/(cm Mhz)]');
    set(h, 'FontName', 'Candara');
    h = title([strrep(func2str(estimator), '_', '\_') ', ' ...
        num2str(SIM_EXPERIMENTS, '%i') ...
        ' runs - Estimate error \epsilon']);
    set(h, 'FontName', 'Candara');
    annotate_graph(deviation(:));
    colorbar;
end
```

A.2.2 test_field_accuracy.m

```
function [deviation, depths, alphas] = test_field_accuracy()

tic;
sim_config();
global SIM_MINALPHA SIM_MAXALPHA SIM_MINDEPTH SIM_MAXDEPTH;
global SIM_F0 SIM_FS SIM_EXPERIMENTS SIM_NFFT;

depths = SIM_MINDEPTH + (SIM_MAXDEPTH-SIM_MINDEPTH) ...
    .* rand(SIM_EXPERIMENTS, 1);
alphas = SIM_MINALPHA + (SIM_MAXALPHA-SIM_MINALPHA) ...
    .*rand(SIM_EXPERIMENTS, 1);
deviation = zeros(SIM_EXPERIMENTS, 1);

emit_aperture = create_aperture();

f0 = SIM_F0 / 1e6;

for idx=1:SIM_EXPERIMENTS
    [s1, t1] = simulate_point_measurement(emit_aperture, 0, depths(idx));
    [s2, t2] = simulate_point_measurement(emit_aperture, ...
        alphas(idx), depths(idx));

    % Calculate power spectrum
    [freqs, findex] = logpsd(s1, SIM_FS, SIM_NFFT);
    [freqs2, findex2] = logpsd(s2, SIM_FS, SIM_NFFT);

    [val, ind] = min(abs(findex-f0));
    deviation(idx) = (alphas(idx) * f0 * depths(idx)/10) ...
        - (freqs(ind) - freqs2(ind));
    if (mod(idx, floor(SIM_EXPERIMENTS/10)) == 0)
        disp([num2str((idx/SIM_EXPERIMENTS)*100) '% done']);
    end
end
toc

figure;
scatter3(depths, alphas, deviation, 2, deviation);
caxis('auto');
h = xlabel('Depth (mm)');
set(h, 'FontName', 'Candara');
h = ylabel('\alpha (dB/(cm*Mhz))');
set(h, 'FontName', 'Candara');
h = zlabel('\epsilon [dB/(cm Mhz)]');
set(h, 'FontName', 'Candara');
h = title(['Field accuracy test, ' ...
    num2str(SIM_EXPERIMENTS, '%i') ...
    ' runs - Deviation from expected \alpha']);
```

```
set(h, 'FontName', 'Candara');
annotate_graph(deviation(:));
colorbar;

figure;
plot(depths, deviation, 'k.');
h = xlabel('Depth (mm)');
set(h, 'FontName', 'Candara');
h = ylabel('\epsilon [dB/(cm Mhz)]');
set(h, 'FontName', 'Candara');
h = title(['Field accuracy test, ' ...
    num2str(SIM_EXPERIMENTS, '%i') ...
    ' runs - Deviation from expected \alpha']);
set(h, 'FontName', 'Candara');
annotate_graph(deviation(:));

figure;
plot(alphas, deviation, 'k.');
h = xlabel('\alpha [dB/(cm*Mhz)]');
set(h, 'FontName', 'Candara');
h = ylabel('\epsilon [dB/(cm Mhz)]');
set(h, 'FontName', 'Candara');
h = title(['Field accuracy test, ' ...
    num2str(SIM_EXPERIMENTS, '%i') ...
    ' runs - Deviation from expected \alpha']);
set(h, 'FontName', 'Candara');
annotate_graph(deviation(:));
```

A.2.3 test_field_accuracy_fixed_alpha.m

```
function [deviation, depths] = test_field_accuracy_fixed_alpha()

tic
sim_config();
global SIM_MINDEPTH SIM_MAXDEPTH;
global SIM_F0 SIM_FS SIM_EXPERIMENTS SIM_NFFT;

alpha = 0.5;
depths = SIM_MINDEPTH + (SIM_MAXDEPTH-SIM_MINDEPTH) ...
    .* rand(SIM_EXPERIMENTS, 1);
deviation = zeros(SIM_EXPERIMENTS, 1);

emit_aperture = create_aperture();

for idx=1:SIM_EXPERIMENTS
    [s1, t1] = simulate_point_measurement(emit_aperture, 0, depths(idx));
    [s2, t2] = simulate_point_measurement(emit_aperture, ...
        alpha, depths(idx));

    % Calculate power spectrum
    nfft = max(SIM_NFFT, 2^(nextpow2(s1)));
    [freqs, findex] = logpsd(s1, SIM_FS, nfft);
    [freqs2, findex2] = logpsd(s2, SIM_FS, nfft);

    [val, ind] = min(abs(findex-(SIM_F0/1e6)));
    deviation(idx) = (alpha * (SIM_F0/1e6) * depths(idx)/10) ...
        - (freqs(ind) - freqs2(ind));
    if (mod(idx, floor(SIM_EXPERIMENTS/100)) == 0)
        disp([num2str((idx/SIM_EXPERIMENTS)*100) '% done']);
    end
end
toc

figure;
plot(depths, deviation, 'k.');
h = xlabel('Depth (mm)');
set(h, 'FontName', 'Candara');
h = ylabel('Deviation');
set(h, 'FontName', 'Candara');
h = title(['Field accuracy test, ' ...
    num2str(SIM_EXPERIMENTS, '%i') ...
    ' runs - Deviation from expected \alpha']);
set(h, 'FontName', 'Candara');
annotate_graph(deviation(:));
```

A.2.4 test_field_accuracy_fixed_freq.m

```
function [deviation, freqs] = test_field_accuracy_fixed_freq()

tic
sim_config();
global SIM_FS SIM_EXPERIMENTS SIM_NFFT;

minfreq = 2;
maxfreq = 4;
depth = 100;

alpha = 0.5;
exfreqs = minfreq + (maxfreq -minfreq) .* rand(SIM_EXPERIMENTS, 1);
deviation = zeros(SIM_EXPERIMENTS, 1);

emit_aperture = create_aperture();

for idx=1:SIM_EXPERIMENTS
    [s1, t1] = simulate_point_measurement(emit_aperture, 0, depth);
    [s2, t2] = simulate_point_measurement(emit_aperture, alpha, depth);

    % Calculate power spectrum
    nfft = max(SIM_NFFT, 2^(nextpow2(s1)));
    [freqs, findex] = logpsd(s1, SIM_FS, nfft);
    [freqs2, findex2] = logpsd(s2, SIM_FS, nfft);

    [val, ind] = min(abs(findex-exfreqs(idx)));
    deviation(idx) = (alpha * (depth/10) * exfreqs(idx)) ...
        - (freqs(ind) - freqs2(ind));

    if (mod(idx, floor(SIM_EXPERIMENTS/100)) == 0)
        disp([num2str([idx/SIM_EXPERIMENTS)*100) '% done']);
    end
end
toc

figure;
plot(exfreqs, deviation, 'k.');
h = xlabel('Frequency (MHz)');
set(h, 'FontName', 'Candara');
h = ylabel('Deviation');
set(h, 'FontName', 'Candara');
h = title(['Field accuracy test, ' ...
    num2str(SIM_EXPERIMENTS, '%i') ...
    ' runs - Deviation from expected \alpha']);
set(h, 'FontName', 'Candara');
annotate_graph(deviation(:));
```

A.2.5 delta_comparison.m

```
function [maes, deltas] = delta_comparison(estimators)
% Please comment out the SIM_DELTAZ line in sim_config.m before
% running this function.

min_delta = 2.5; % minimum thickness [mm]
max_delta = 4;  % maximum thickness [mm]
steps = 11;
stepsize = (max_delta -min_delta) / (steps -1);
margin = (max_delta -min_delta) * 0.1;

global SIM_DELTAZ SIM_PLOT SIM_SILENT SIM_EXPERIMENTS;
SIM_PLOT = 0;
SIM_SILENT = 1;

deltas = min_delta:stepsize:max_delta;
maes = zeros(steps, length(estimators));

for idx = 1 : steps
    SIM_DELTAZ = deltas(idx);
    for estidx = 1 : length(estimators)
        estfunc = estimators{idx};
        errs = test_estimator(estfunc);
        maes(idx, estidx) = mean(abs(errs));
    end
end

SIM_PLOT = 1;
SIM_SILENT = 0;

figure;
barh(deltas, maes, 1);
ylim([[min_delta -margin) (max_delta + margin)]];
set(gca, 'ytick', deltas);
set(gca, 'YDir', 'reverse');
h = legend('Burg', 'Log PSD', 'Mod. cov.', 'Yule-Walker', ...
    'Location', 'SouthEast');
set(h, 'FontName', 'Candara');
h = title(['MAE, varying \Deltaz- ' ...
    num2str(SIM_EXPERIMENTS, '%i') ' runs']);
set(h, 'FontName', 'Candara');
h = ylabel('\Deltaz [mm]');
set(h, 'FontName', 'Candara');
h = xlabel('MAE [db/(cm Mhz)]');
set(h, 'FontName', 'Candara');
```

A.2.6 noise_comparison.m

```
function noise_comparison()
% Please comment out the SIM_NOISE_LEVEL line in sim_config.m before
% running this function.

min_nl = -60; % minimum noise level [dB]
max_nl = -46;  % maximum noise level [dB]
steps = 8;
stepsize = (max_nl-min_nl)/(steps-1);

global SIM_NOISE SIM_NOISE_LEVEL SIM_PLOT SIM_SILENT SIM_EXPERIMENTS;
SIM_PLOT = 0;
SIM_SILENT = 1;

levels = min_nl:stepsize:max_nl;
maes = zeros(steps, 4);

for idx = 1 : steps
    SIM_NOISE_LEVEL = levels(idx);
    dburg = test_estimator(@burg_estimator);
    dlogpsd = test_estimator(@logpsd_estimator);
    dmcov = test_estimator(@mcov_estimator);
    dyulear = test_estimator(@yulear_estimator);

    maes(idx, 1) = mean(abs(dburg));
    maes(idx, 2) = mean(abs(dlogpsd));
    maes(idx, 3) = mean(abs(dmcov));
    maes(idx, 4) = mean(abs(dyulear));
end

SIM_PLOT = 1;
SIM_SILENT = 0;

figure;
barh(levels, maes, 1);
set(gca, 'YDir', 'reverse');
h = legend('Burg', 'Log PSD', 'Mod. cov.', 'Yule-Walker', ...
    'Location', 'NorthEast');
set(h, 'FontName', 'Candara');
h = title(['MAE after adding noise -' ...
    num2str(SIM_EXPERIMENTS, '%i') ' runs']);
set(h, 'FontName', 'Candara');
h = ylabel('Noise level [dB]');
set(h, 'FontName', 'Candara');
h = xlabel('MAE [db/(cm Mhz)]');
set(h, 'FontName', 'Candara');
```

A.2.7 thickness_comparison.m

```
function [maes, thicknesses] = thickness_comparison()
% Please comment out the SIM_PHANTOM_Z line in sim_config.m before
% running this function.

min_th = 0; % minimum thickness [mm]
max_th = 0.0002; % maximum thickness [mm]
steps = 11;
stepsize = (max_th-min_th)/(steps-1);
margin = (max_th -min_th) * 0.1;

global SIM_PLOT SIM_SILENT SIM_EXPERIMENTS SIM_PHANTOM_Z;
SIM_PLOT = 0;
SIM_SILENT = 1;

thicknesses = min_th:stepsize:max_th;
maes = zeros(steps, 4);

for idx = 1 : steps
%    SIM_NOISE_LEVEL = levels(idx);
    SIM_PHANTOM_Z = thicknesses(idx);
    dburg = test_estimator(@burg_estimator);
    dlogpsd = test_estimator(@logpsd_estimator);
    dmcov = test_estimator(@mcov_estimator);
    dyulear = test_estimator(@yulear_estimator);

    maes(idx, 1) = mean(abs(dburg));
    maes(idx, 2) = mean(abs(dlogpsd));
    maes(idx, 3) = mean(abs(dmcov));
    maes(idx, 4) = mean(abs(dyulear));
end

SIM_PLOT = 1;
SIM_SILENT = 0;

figure;
barh(thicknesses, maes, 1);
ylim([[min_th - margin] [max_th + margin]]);
set(gca, 'ytick', thicknesses);
set(gca, 'YDir', 'reverse');
h = legend('Burg', 'Log PSD', 'Mod. cov.', 'Yule-Walker', ...
    'Location', 'NorthEast');
set(h, 'FontName', 'Candara');
h = title(['MAE, varying phantom thickness -' ...
    num2str(SIM_EXPERIMENTS, '%i') ' runs']);
set(h, 'FontName', 'Candara');
h = ylabel('Phantom thickness [mm]');
set(h, 'FontName', 'Candara');
```

```
h = xlabel('MAE [db/[cm Mhz]]');
set(h, 'FontName', 'Candara');
```

A.3 Estimator functions

A.3.1 logpsd_estimator.m

```
function estimate = logpsd_estimator(s1, s2)

sim_config();

global SIM_F0 SIM_FS SIM_NFFT SIM_DELTAZ
% Calculate power spectrum
maxl = max(length(s1), length(s2));
nfft = max(SIM_NFFT, 2^(nextpow2(maxl)));

bottom = (SIM_F0/1e6) -0.3;
top = (SIM_F0/1e6) + 0.3;

[freqs, findex] = logpsd(s1, SIM_FS, nfft);
[freqs2, findex2] = logpsd(s2, SIM_FS, nfft);

% Calculate log power spectral difference
psdiff = freqs - freqs2;
[val, linebegidx] = min(abs(findex -bottom ));
[val, lineendidx] = min(abs(findex -top));

p = polyfit(findex(linebegidx:lineendidx), psdiff(linebegidx:lineendidx), 1);
estimate = (p(1) * 10) / (2*SIM_DELTAZ);

% figure
% subplot(2,1,1);
% h = title('Log PSD - Signal log power spectra');
% set(h, 'FontName', 'Candara');
% hold on
% plot(findex, freqs, 'r');
% plot(findex2, freqs2, 'g-.');
% xlim([2.2 3.8]);
% h = xlabel('Frequency (MHz)');
% set(h, 'FontName', 'Candara');
% subplot(2,1,2);
% h = title('Log PSD - Difference and fitted line');
% set(h, 'FontName', 'Candara');
% hold on
% plot(findex, psdiff, 'r');
% plot(findex, p(1).*findex + p(2), 'g- .');
% xlim([2.2 3.8]);
% h = xlabel('Frequency (MHz)');
% set(h, 'FontName', 'Candara');
% error('LogPSD debug plot');
```

A.3.2 burg_estimator.m

```
function estimate = burg_estimator(s1, s2)

sim_config;
global SIM_FS SIM_NFFT SIM_SIGMA2 SIM_DELTAZ

fs = SIM_FS;
sigma2 = SIM_SIGMA2;
deltaz = SIM_DELTAZ;
maxl = max(length(s1), length(s2));
nfft = max(SIM_NFFT, 2^(nextpow2(maxl)));

[psd1, f1] = pburg(s1, 2, nfft, fs);
[psd2, f2] = pburg(s2, 2, nfft, fs);
cf1 = sum(f1 .* psd1) / sum(psd1);
cf2 = sum(f2 .* psd2) / sum(psd2);
plot(f1, psd1)
error('asdf')
estimate = ((cf1 -cf2) / (sigma2 * deltaz * 1000));
```

A.3.3 mcov_estimator.m

```
function estimate = mcov_estimator(s1, s2)

sim_config;
global SIM_FS SIM_NFFT SIM_SIGMA2 SIM_DELTAZ

fs = SIM_FS;
nfft = SIM_NFFT;
sigma2 = SIM_SIGMA2;
deltaz = SIM_DELTAZ;

[psd1, f1] = pmcov(s1, 2, nfft, fs);
[psd2, f2] = pmcov(s2, 2, nfft, fs);
cf1 = sum(f1 .* psd1) / sum(psd1);
cf2 = sum(f2 .* psd2) / sum(psd2);

estimate = ((cf1 -cf2) / (sigma2 * deltaz * 1000));
```

A.3.4 yulear_estimator.m

```
function estimate = yulear_estimator(s1, s2)

sim_config;
global SIM_FS SIM_NFFT SIM_SIGMA2 SIM_DELTAZ

fs = SIM_FS;
nfft = SIM_NFFT;
sigma2 = SIM_SIGMA2;
deltaz = SIM_DELTAZ;

[psd1, f1] = pyulear(s1, 2, nfft, fs);
[psd2, f2] = pyulear(s2, 2, nfft, fs);
cf1 = sum(f1 .* psd1) / sum(psd1);
cf2 = sum(f2 .* psd2) / sum(psd2);

estimate = ((cf1 -cf2) / (sigma2 * deltaz * 1000));
```