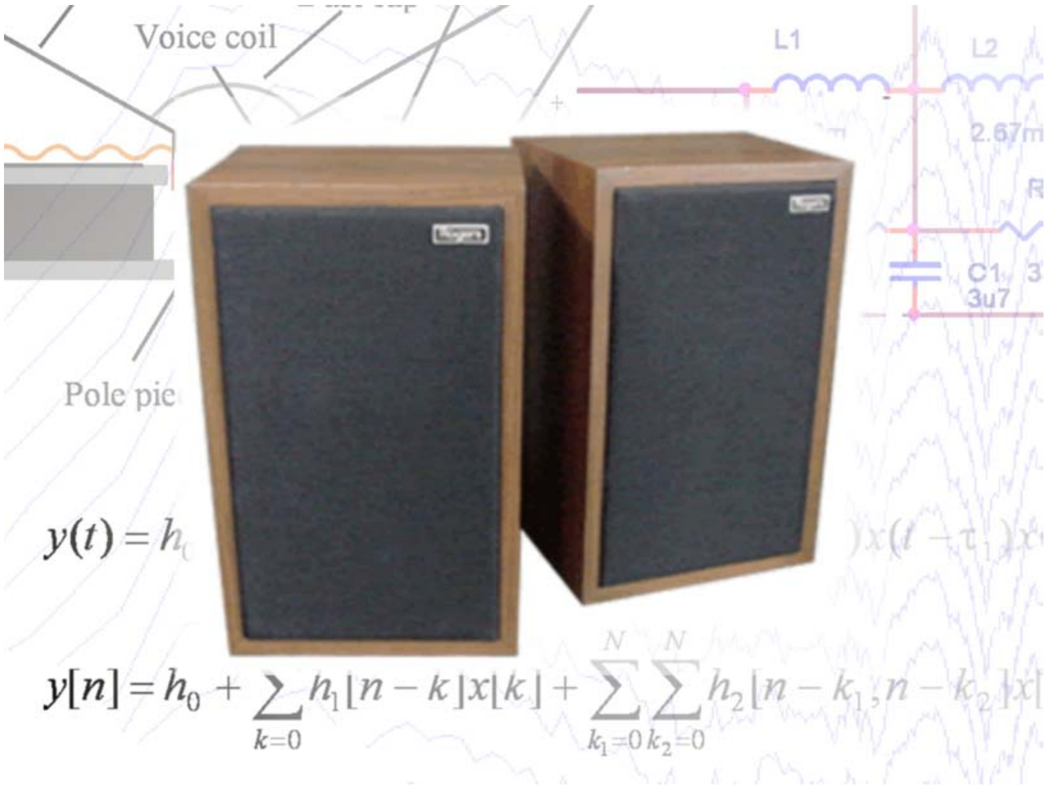
Imperial College London

Department of Electrical and Electronic Engineering

Final Year Project Report 2005



Project Title:	**A Novel Loudspeaker Equaliser**
Student:	**M. R. P. Thomas**
Course:	**4T**
Project Supervisor:	**Dr. Patrick Naylor**
Second Marker:	**Mike Brookes**

Final-Year Project Report

A Novel Loudspeaker Equaliser

M. R. P. Thomas

Abstract

Fundamentally, loudspeaker design has changed very little in the past 30 years. The late 70s saw the introduction of electromechanical modelling techniques which remain the basis for the design and analysis of loudspeakers today.

Aside from the use of lighter, stronger materials in the construction of drive units, the area which has undergone the most development is the design of equalisers. Classically, loudspeakers use a passive analogue filter network to compensate for a non-flat frequency response, but they are far from ideal and are prone to variation with temperature and age. Recent years have seen the use of linear DSP equalisers which conform to much tighter specifications.

However, there are two major flaws in the design of even the latest equalisers. Firstly, it has long been known that at extreme voice coil excursion a loudspeaker is a nonlinear device, though no documented attempt at producing a nonlinear equaliser has yet been proposed. Secondly is the use of swept sine tones to characterise frequency responses, which can sometimes yield different results to methods where excitation energy is spread over a wider bandwidth (which is more akin to music or speech). Swept sines generally provide no phase information, which is an area of increasing interest in loudspeaker equalisation.

This project investigates the analysis of loudspeaker frequency responses using a *Maximum-Length Sequence (MLS)* or *M-Sequence Decorrelation* technique, which uses pseudorandom noise to yield both amplitude and phase information. The measurements are used to model the frequency response over a wide range of amplitudes, fitting a Volterra-Series approximation and defining a level-dependent equaliser.

Acknowledgements

Dr. Patrick Naylor (*Comms & Signal Processing Dept.*, Imperial College), for his expertise and enthusiasm which have informed and inspired me throughout this project.

Danny Harvey (*Comms & Signal Processing Lab*, Imperial College), for lending me his personal audio equipment, without which my analysis software couldn't perform to its potential.

Roger Brownless (*Studio Group*, BBC R&D), for his assistance in the free-field room at BBC R&D.

Andrew Mason (*Transmission Systems Group*, BBC R&D), for the loan of equipment for early measurements.

Final-Year Project Report

A Novel Loudspeaker Equaliser

M. R. P. Thomas

Contents

Final-Year Project Report

A Novel Loudspeaker Equaliser

M. R. P. Thomas

1 Introduction

The assessment of the 'quality' of a loudspeaker is open to a myriad of objective and subjective measurements. Attempts to accurately and concisely model a loudspeaker as a linear circuit by R. H. Small in his brilliant set of papers [18] [19] [20] (published in the early 70s) remain the standard by which engineers characterise their drivers and enclosures.

It is generally accepted that the most important figure-of-merit is the loudspeaker's frequency response. This is rarely flat over the frequencies of interest (~50Hz – 20kHz) so equalisers are devised for Hi-Fi applications to flatten the response of the system. Early equalisers used passive analogue networks which were inaccurate as well as being prone to variations with temperature and age. More recent developments have seen the use of DSP to replace the analogue networks, providing tighter tolerances and no drift.

Existing techniques do, however, contain two major flaws. First is the assumption that the loudspeaker's frequency response is linear. It will be shown analytically and practically that at large voice-coil excursions the driving force and damping are no longer proportional to input current. Second is the almost universal use of swept-sine methods to characterise frequency responses. With swept sines, energy is concentrated over a very narrow bandwidth. It has been shown in [3] that frequency responses derived from spot tones can differ from those where the excitation energy is spread over a wider spectrum. Given that the energy in music and speech is rarely concentrated over a small bandwidth, analysis employing spot tones should be avoided. Swept sines also don't provide phase information, but it can be shown that the phase response (and therefore the group delay) is not necessarily linear and perhaps should be taken into consideration in the design of an equaliser.

The most popular techniques for determining frequency responses of *linear* systems will be described in 2.3 and it will be argued that the best method for characterising the nonlinear response of a loudspeaker is the *Maximum Length Sequence* (MLS) or *M-Sequence Decorrelation* technique. We will discuss the theory and implementation issues with MLS in detail as they are an important tool in this project.

Causes of nonlinearity in loudspeakers and appropriate modelling techniques will be discussed in some detail. Using software developed for MLS analysis, a best-fit linear equaliser and a level-dependent equaliser will be calculated from real measurements and the benefits of level-dependent equalisation will be demonstrated. Nonlinear modelling is often based upon Volterra Series approximations, which model nonlinearities as polynomials, and it is shown in [11] that a second-order Volterra 'Kernel' is sufficient for describing the nonlinear response of a loudspeaker. From empirical results, a Volterra Series approximation is calculated for a test loudspeaker and a method for creating an equaliser is discussed.

1.1 Project Objectives

To summarise, the project objectives are:

- Analyse where loudspeakers deviate from an ideal case and review existing measurement and equalisation techniques.
- Develop software which uses MLS sequences to determine the amplitude and phase response of a loudspeaker as a function of driving frequency and amplitude.
- Determine the root causes of loudspeaker nonlinearity and show that it is directly related to voice coil excursion.
- Analyse experimental results and attempt to fit nonlinearities to a Volterra-Series approximation.
- Design and test different types of equaliser.

1.2 Major Achievements

- At the time of the project's conception, it was assumed that the measurement of the test loudspeakers would be a straightforward swept sine and that most of the project would be dedicated to analysis. It became clear that MLS analysis was a much more elegant and accurate method (for reasons discussed later) but no open-source implementation existed and many of the available papers distinctly lacked in detail about implementation. I changed my schedule and spent a significant amount of time developing a set of command-line tools. Once the equaliser toolkit neared completion, a sensible progression was to bring the tools together in the form of a GUI. This offers the sound engineer a unique tool to analyse a loudspeaker (or any system), generate equalisers and test their validity with real measurements.
- Showing that loudspeaker nonlinearities are present at listening levels and not just at extreme clipping, and implementing a method to eliminate them almost entirely.

1.3 Report Structure

Theory

Chapter 2: Background

Putting the problem into context with an example loudspeaker frequency response. Describing common measurement techniques. Example of an analogue equaliser. Example of a digital equaliser.

Chapter 3: Maximum-Length Sequences

Describing the theory and practical implementation of MLS analysis software.

Chapter 4: Nonlinear Modelling

Reasons for nonlinearity in dynamic loudspeakers. Volterra-series approximations and their use in nonlinear modelling.

Chapter 5: Formal Equaliser Theory

Presents mathematical models of equalisers.

Practical

Chapter 6: Hardware

Measurement equipment, calibration.

Chapter 7: Results and Volterra Series Approximation

Results from a real loudspeaker measurements, implementation of linear- and level-dependent equalisers and Volterra Series approximation.

Chapter 8: Conclusions and Future Work

Reference

Chapter 9: References

Appendix 1 - User guide: Command-line tools

Appendix 2 - User guide: GUI

Appendix 3 - Command-line tool program listings

2 Background

In this chapter we shall discuss the problem of dynamic loudspeaker frequency responses, current approaches and how they may be improved. This is followed by a précis of common measurement techniques, associated practical difficulties, the information they provide the engineer and justification of which are the most useful for this project.

2.1 The Problem

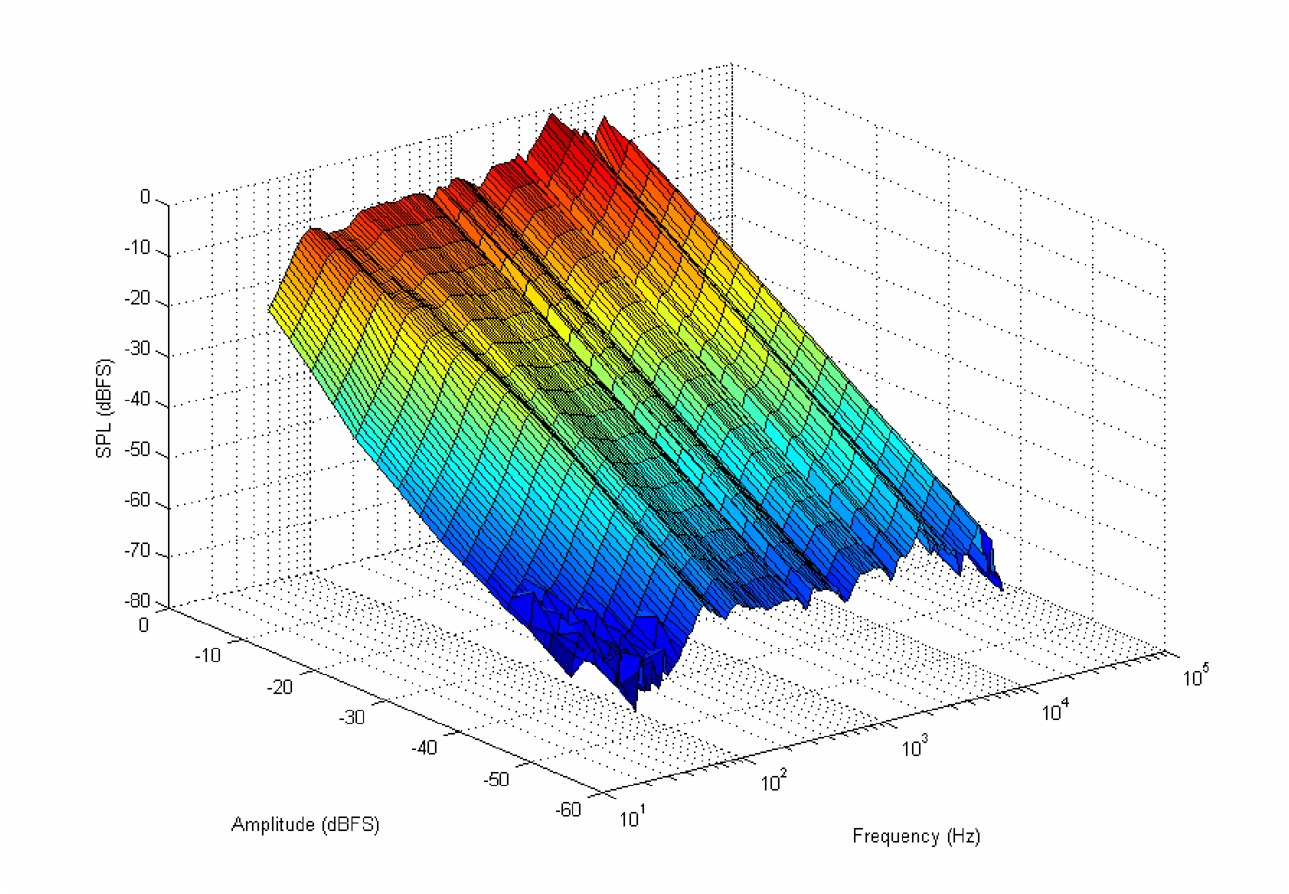


Figure 2.1: Sound pressure level as a function of both input amplitude and frequency for a loudspeaker. SPL is measured in dBFS, where 0dBFS corresponds to ~100dBA at 1m.

Figure 2.1 depicts the sound pressure level of a loudspeaker as a function of the amplitude and frequency of an input signal. 0dBFS is the 'full-scale' (normalised) amplitude of the input and output signals[1]. In this case, 0dBFS corresponds to about 100dBA at 1m – a moderately high drive level for this type of loudspeaker.

[1] dBFS is dB 'Full-Scale'. It is necessarily less than or equal to 0, where 0 corresponds to the maximum amplitude a system can reproduce before clipping occurs. 1 dBu is equal to 1mW across 600Ω, a common unit for measuring audio signal levels. dBA is the 'A-weighted' sound pressure level, which takes into account the

We can observe the following features from this graph:

1. At the extreme low-frequency end, the SPL decreases at a rate of 12dB / octave. [18] models a loudspeaker in a sealed enclosure as a high-pass filter with two zeros near the origin.
2. There is attenuation at the extreme high-frequency end. Loudspeaker drive units must be stiff in order to convert movement from the voice coil into sound pressure waves, but must also be light. Mass is equivalent to a series inductor: its impedance increases with frequency and so loudspeaker gain will decrease at very high frequencies.
3. Peaks and troughs between ~150Hz and 15kHz. These are caused by a number of different reasons, the most common being resonances in the wooden enclosure and cone break-up (standing-wave modes across the cone).
4. Frequency-selective compression at high drive levels. It will be shown in Chapter 4 that low frequencies are most affected by nonlinearity at high drive levels.

2.2 Existing Approaches

Existing equaliser networks often incorporate crossover networks, which ensure that the correct frequencies arrive at each drive unit. Their ideal function is depicted in *Figure 2.2*, where the blue line represents the frequency response of the loudspeaker at a typical listening level and the green is the frequency response of the equaliser; the result is a flattened frequency response.

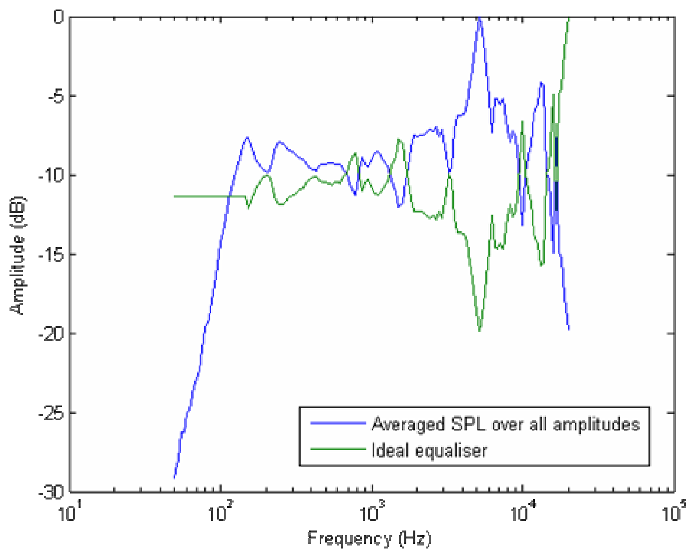


Figure 2.2: An ideal equaliser achieves a flat frequency response by inverting the response of the loudspeaker

A truly flat response cannot be achieved down to DC as loudspeaker enclosures are high-pass filters (shown in [18]) with zero DC gain. To flatten the response would require very high gain at low frequencies (or attenuation at high frequencies), so equaliser gain is usually constant below a certain threshold as shown.

The Control Engineer may see this as a straightforward proportional feedback control system, with a transducer mounted on the loudspeaker cone providing a feedback signal. This would avoid the need to measure frequency responses, providing a dynamic compensator and theoretically better results. Attempts have been made to implement such a control system but have had limited success due to poor transducer performance and are therefore very uncommon in Hi-Fi audio.

uneven and nonlinear frequency response of the human ear. 0 dBA is defined as the smallest sound pressure level audible to a human being. dBFS is used throughout this project.

2.2.1 Analogue Equalisers

Analogue equalisers are very common in Hi-Fi loudspeakers as they are simple and can be very cheap to manufacture. They are, however, very complicated to design and optimise; a particularly complex circuit given in *Figure 2.3* is from an LS3/5A-type loudspeaker and took hundreds of man-hours to design. It is far from ideal and C2 and L3 are both AOT (adjust-on-test) components to compensate for manufacturing variations. The network is also prone to drift with temperature and age.

A further drawback is that like most analogue crossover networks, it operates on high-level signals. A passive network can only attenuate, so they are continuously wasting power (in the given circuit diagram there is a large amount of HF attenuation and R1 has been known to cause the loudspeaker damping material to catch fire!).

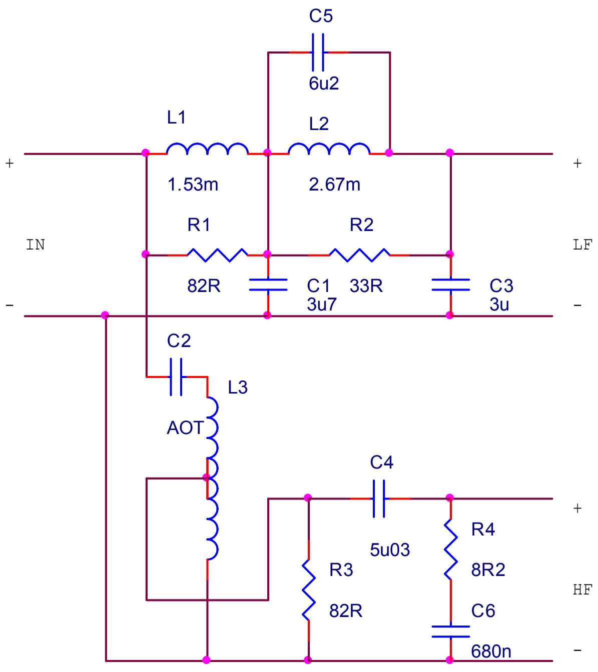
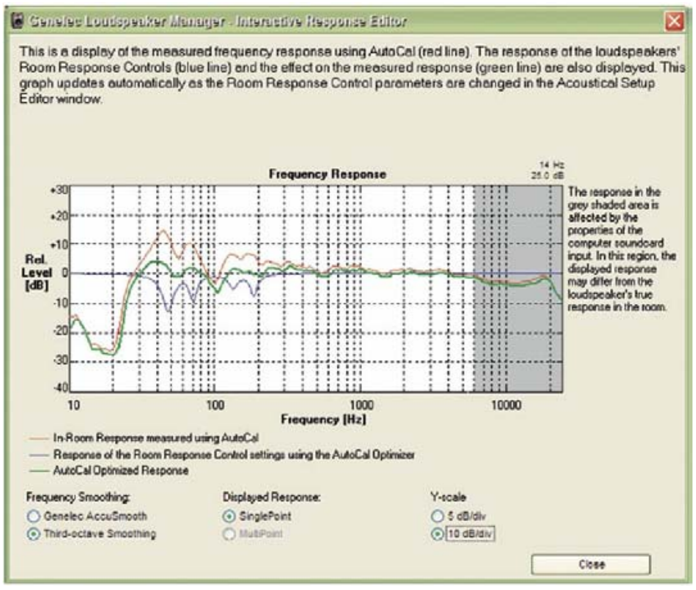


Figure 2.3: Crossover / equaliser network from an LS3/5A loudspeaker.

2.2.2 Digital Equaliser

A modern approach is to apply linear DSP to loudspeaker equalisation. This operates on low-level signals, can be specified to arbitrary precision (at the expense of systematic delay), is easy to recalibrate and does not drift with age. *Figure 2.4* is a screenshot from a state-of-the art Genelec system which represents the best in consumer loudspeaker technology. It uses pink noise bursts (explained in 2.3.4) to determine the amplitude response of the loudspeaker and implements up to 4 shelves and 4 notch filters to flatten the response. It takes into account on-and-off axis measurements and can correct some artefacts which are the result of poor room acoustics; this is outside the scope of this project.



Genelec Loudspeaker Manager - Interactive Response Editor

This is a display of the measured frequency response using AutoCal (red line). The response of the loudspeakers' Room Response Controls (blue line) and the effect on the measured response (green line) are also displayed. This graph updates automatically as the Room Response Control parameters are changed in the Acoustical Setup Editor window.

The response in the grey shaded area is affected by the properties of the computer soundcard input. In this region, the displayed response may differ from the loudspeaker's true response in the room.

Frequency Smoothing: Genelec AccuSmooth [✓ Third-octave Smoothing]

Displayed Response: [✓ SinglePoint] MultiPoint

Y-scale: 5 dB/div [✓ 10 dB/div]

Frequency Response
14 Hz
25.0 dB
Rel. Level [dB]
+30
+20
+10
0
-10
-20
-30
-40
10
100
1000
10000
Frequency [Hz]
In-Room Response measured using AutoCal
Response of the Room Response Control settings using the AutoCal Optimizer
AutoCal Optimized Response
Close

Figure 2.4: Screenshot from Genelec's Loudspeaker Editor

2.3 Improvements

Existing commercial techniques rely upon digital or analogue shelves, notches and band-pass filters to approximate an equaliser. Modern FFT techniques circumvent the need to think in these terms as each FFT 'bin' can be treated individually like in *Figure 2.2*, which is conceptually and computationally much simpler. However, no documented attempt in a commercial system has been made. Nonlinear modelling with conventional techniques is virtually impossible.

Dated measurement techniques are also commonly used, such as swept sines and pink noise (explained in 2.4) rather than impulse or MLS excitation. It is demonstrated in [3] that results may differ between techniques.

Figure 2.5: A loudspeaker is placed inside an anechoic chamber, which absorbs most of the radiated sound energy, allowing the frequency response of the loudspeaker to be accurately measured. Depicted is a particularly large chamber at NMIJ in Japan.

2.4 Measurement Techniques

There are five main approaches to measuring a frequency response and all impulse testing require that a loudspeaker be placed in an 'anechoic chamber' or 'free-field room' to ensure that the frequency response of the loudspeaker – and not the room it is placed in – is measured. An example chamber is depicted in *Figure 2.5*.

2.4.1 Swept sine excitation

The loudspeaker under test is fed with a swept sinusoid and the amplitude of the output is measured. Traditionally this would be plotted on a chart recorder but modern digital techniques have superseded this method. Digital techniques differ in that the sine is stepped in frequency, but the results are the same and in both domains slower sweep times result in improved SNR. Normally this method is used to obtain a logarithmic frequency axis and generally does not yield phase information, so is not suitable for obtaining an impulse response. Swept sines concentrate excitation energy to a very narrow spectrum, which can yield different results to signals occupying a wider bandwidth. Speech and music tend not to contain single sines but are more noise-like in nature, so swept sine excitation may not necessarily give accurate results.

Sine excitation can also be used to produce a figure-of-merit known as Total Harmonic Distortion (THD), which is a ratio of the energy in the harmonics to the total energy, expressed as a percentage. Harmonic energy is the total energy with a notch filter placed at the fundamental.

$$\%\text{THD} = \frac{\text{Energy in harmonics}}{\text{Total energy}} \times 100 \qquad (2.1)$$

2.4.2 Impulse excitation

The impulse response of a loudspeaker can be measured by simply sending an impulse through it and measuring the output [1] [4] . However, in order to have a flat frequency response in the range of interest this pulse has to be very short, but at the same time it must contain enough energy to produce a reasonable SNR, requiring very high-amplitude impulses which can sometimes overload the system. SNR can be improved by repeating the test and averaging, but the measurement time will increase correspondingly.

2.4.3 MLS excitation

Maximum Length Sequences (MLS) are pseudo-random signals containing only the values +1 and -1. They are spectrally flat but have the important property of an autocorrelation function which is a perfect impulse with a DC offset. By measuring the loudspeaker's output with an MLS signal, the impulse response may be obtained by de-convolving the output with the original signal by use of a modified Hadamard Transform. MLS processing is computationally simple though theoretically quite complex. It also gives good SNR and can be improved by averaging repeated MLS sequences.

2.4.4 White and Pink Noise

White noise is by definition spectrally flat. Excitation of a loudspeaker with white noise and analysis of a smoothed FFT of the response will therefore yield the amplitude response. However, this may give inaccurate results (particularly at high levels) as the nature of music and speech is decreasing energy with increasing frequency. White noise can be thought of as 'equal energy per Hz'; an alternative type of noise is 'pink' or '1/*f*' noise, which has been prefiltered at -3dB/octave to have equal energy per octave. Pink noise generators are difficult to realise because:

1. They require infinite gain at DC

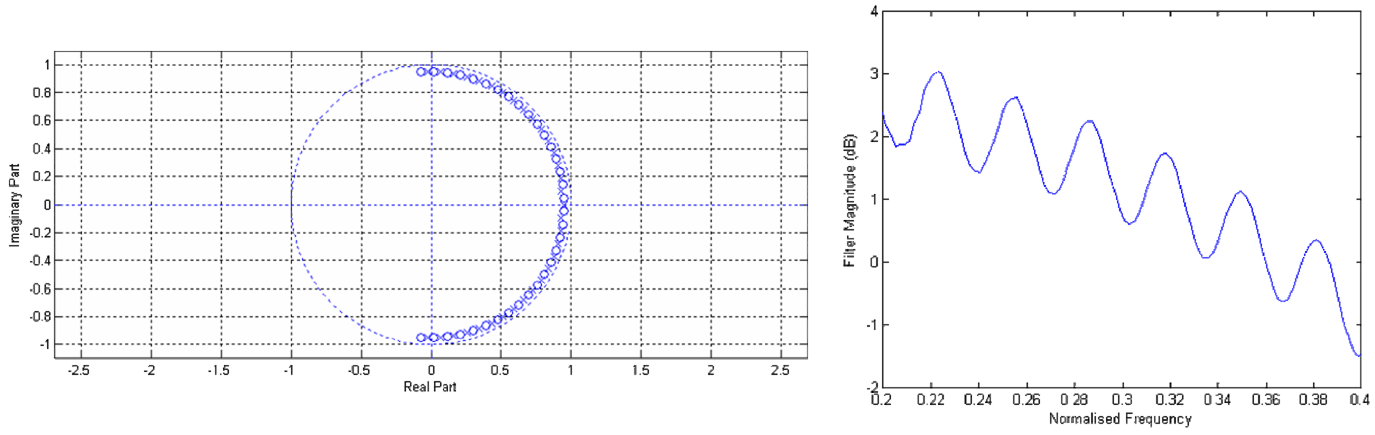


Figure 2.6: Pole-zero plot of a half-band pink noise filter and an octave of its corresponding magnitude response. The average slope is -3dB/octave, which is twice the slope achieved by a single reactive component.

2. The presence of a zero will cause the frequency response to drop at -6dB/octave, so alternating zeros and poles at narrow spacings near the unit circle are needed to approximate a -3dB/octave slope.

Figure 2.6 shows a pole-zero plot and one octave of its corresponding magnitude response. Closer approximations are achieved by placing poles and zeros closer together.

Other types of noise shaping are also applicable – for example when the spectrum of unwanted signals is known. Suggestions for shaping impulse excitation is proposed in [4] and MLS in [3], but it has not been implemented in this project to maintain simplicity.

2.4.5 Two-channel FFT

Noise, MLS, Impulse and swept sine methods all require that the system under test is linear. It is therefore reasonable to assume that the frequency response of a loudspeaker may be obtained by exciting it with any signal and comparing the FFTs of the input and output waveforms. This is more computationally complex than the MLS method but provides a greater degree of freedom when selecting the excitation signal.

2.5 Methods employed in this project

MLS excitation was concluded to be the best method for measuring the frequency response of a loudspeaker because of its mathematical elegance, broadband excitation and ability to extract phase information. Stepped sine sequences were also implemented to provide a simple comparison to MLS amplitude results and to provide THD information. Test signals at a variety of different amplitudes will be used to examine the frequency response at different amplitudes.

2.6 Summary

Most loudspeakers require a certain amount of equalisation in order to achieve a frequency response which is flat within acceptable limits. Existing analogue methods remain unchanged for in excess of thirty years and new digital techniques are often no more than digital realisations of analogue theory. Modern FFT techniques would allow much simpler design and greater flexibility than the existing methods and will be used in this project. No documented attempt at defining an equaliser to compensate for nonlinearities has been made so this will be investigated in some detail.

There are many methods available for characterising a loudspeaker's frequency response. Impulsive excitation, swept sines and pink noise have many drawbacks that are addressed by MLS analysis, which will be the primary measurement technique used in this project. Swept sines will also be used to determine THD values.

3 Maximum-Length Sequences

A Maximum-Length Sequence (MLS) is a periodic two-level signal of length $P = 2^N - 1$, where N is an integer and P is the periodicity, which yields the impulse response of a linear system under *circular* convolution. The impulse response is extracted by the deconvolution of the system's output when excited with an MLS signal. This chapter presents the underlying theory and how the Fast Hadamard Transform can provide a very efficient means of analysing an MLS sequence. A Matlab implementation of the algorithm presented in this chapter was written for the project (see Appendix 1). In doing so some difficulties arose from lack of emphasis of certain some important points in the available literature, so this description is aimed at the engineer interested in writing his own program.

Maximum-Length Sequences have a number of attractive properties. The most important is that with the exception of a DC error, the autocorrelation is a perfect impulse. They share many statistical properties with white noise despite being entirely deterministic, and may be generated very easily by use of shift-registers. Cross-correlation is computationally efficient as binary signals may be analysed using only additions and subtractions; no multiplications are necessary.

3.1 Basic MLS Theory

The MLS signal may be calculated recursively using the following relation:

$$n(k+3) = n(k) \oplus n(k+2) \tag{3.1}$$

Where $\oplus$ denotes an XOR (modulo-2 sum) operation. It can be implemented with a set of shift registers (usually initialised to all 1s) and an XOR gate; an example sequence with N=3 is shown in *Figure 3.1*. Note that if the shift registers were initialised to consist entirely of zeros then it would be impossible for a 1 to ever occur and the circuit would remain frozen. It is for this reason that MLS signals are 2^N -1 and not 2^N in length.

The binary MLS is converted to a signal by the mapping:

$$\begin{aligned} 0 &\rightarrow 1 \\ 1 &\rightarrow -1 \end{aligned} \tag{3.2}$$

Let the MLS signal be $x[n]$ and the impulse response of the system be $h[n]$. The output, $y[n]$, of the system stimulated with $x[n]$ will therefore be:

$$\begin{aligned} y[n] &= h[n] * x[n] \\ &= \sum_{k=-\infty}^{+\infty} x[k]h[n-k] \end{aligned} \tag{3.3}$$

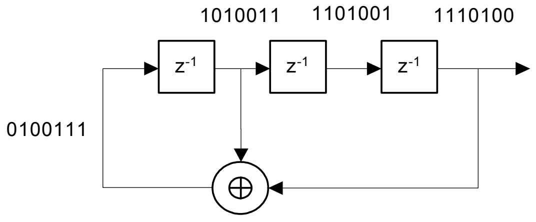


Figure 3.1: Binary feedback shift register for generating a Maximum-Length Sequence, n=3, P=7

Where $*$ denotes discrete linear convolution. The assumption of linear convolution is not mathematically sound as it requires the MLS to be aperiodic. By definition this cannot be true unless $N=\infty$. Circular convolution is therefore more applicable in this case. Let the periodic impulse response be $h'[n]$ and let the Dirac delta $\delta[n]$ be defined as:

$$\delta[n] = \begin{cases} 1, & n = 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.4}$$

And the periodic Dirac delta be $\delta'[n]$:

$$\delta'[n] = \begin{cases} 1, & n \bmod P = 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.5}$$

Then,

$$h'[n] = \sum_{k=-\infty}^{+\infty} \delta'[k] h[n-k]$$

$$= \sum_{k=-\infty}^{+\infty} h[n+kP] \tag{3.6}$$

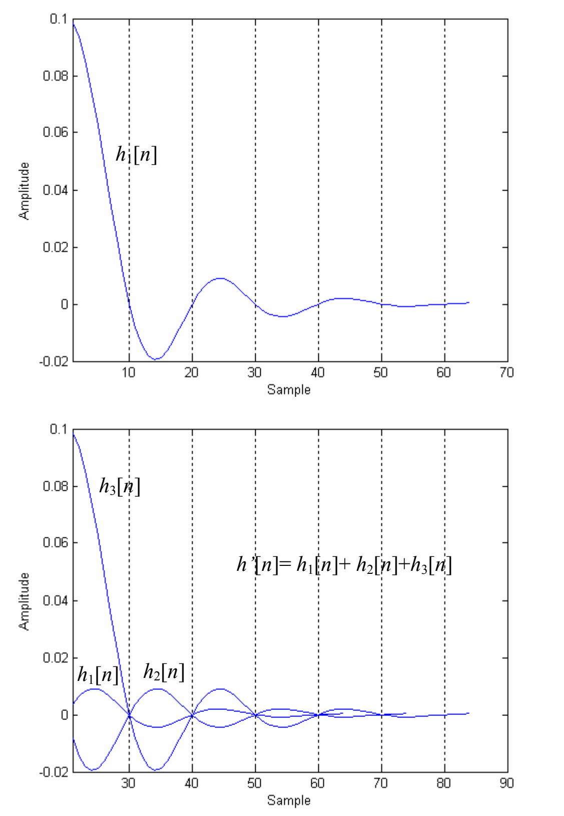


Figure 3.2: Time-aliasing in periodic impulse h'[n] caused by insufficiently long MLS. Upper figure is h[n] and lower, h'[n], is h[n] repeated every P samples.

Qualitatively, the periodic impulse response $h'[n]$ is the true impulse response $h[n]$ repeated at period P. If the periodic input is $x'[n]$ and the periodic output is $y'[n]$,

$$y'[n] = x'[n] \otimes h'[n]$$

$$= \sum_{k=0}^{P-1} x'[k] h'[n-k] \tag{3.7}$$

Where $\otimes$ denotes circular convolution. $[n-k]$ is calculated modulo P and $h'[n]$, $x'[n]$ and $y'[n]$ are periodic in P.

P must be chosen to be sufficiently large that the transients from previous impulse responses $h_k[n]$ have died down enough not to cause time-aliasing. *Figure 3.2* demonstrates time-aliasing graphically.

MLS necessarily measures the periodic impulse response and not the true impulse response. However, when applied to a real system circular convolution cannot be achieved as real systems apply linear convolution, with an unexcited initial state. Circular convolution may be approximated by stimulating the system with the MLS sequence *twice* and analysing the second sequence; again the sequence must be long enough to prevent time aliasing.

3.2 Deconvolution

Many texts on MLS present the deconvolution operation in terms of cross-correlation [2], [3]. I shall present my discussion in terms of convolution in a similar fashion to [7].

Recall that

$$y[n] = h[n] * x[n] \tag{3.8}$$

Then

$$y[n] * z[n] = h[n] * x[n] * z[n] = h'[n] \tag{3.9}$$

Iff $x[n] * z[n] = \delta'[n]$ and there is negligible time-aliasing.

If $z[n] = x[-n]$ then $x[n] * z[n]$ is the autocorrelation function of $x[n]$ and $x[n] * z[n] = \delta'[n]$ is fulfilled except for a DC offset.

By deconvolving, the signal $h'[n]$ is given by:

$$h'[n] = \frac{1}{P+1}\left(\sum_{k=1}^{P} y[k]x[k-n]\right) \tag{3.10}$$

(Recall $h'[n]$ is a periodic version of $h[n]$).

In the case when P=7:

$$\begin{bmatrix} h[0] \\ h[1] \\ h[2] \\ h[3] \\ h[4] \\ h[5] \\ h[6] \end{bmatrix} = \frac{1}{8} \begin{bmatrix} x[0] & x[1] & x[2] & x[3] & x[4] & x[5] & x[6] \\ x[1] & x[2] & x[3] & x[4] & x[5] & x[6] & x[0] \\ x[2] & x[3] & x[4] & x[5] & x[6] & x[0] & x[1] \\ x[3] & x[4] & x[5] & x[6] & x[0] & x[1] & x[2] \\ x[4] & x[5] & x[6] & x[0] & x[1] & x[2] & x[3] \\ x[5] & x[6] & x[0] & x[1] & x[2] & x[3] & x[4] \\ x[6] & x[0] & x[1] & x[2] & x[3] & x[4] & x[5] \end{bmatrix} \begin{bmatrix} y[0] \\ y[1] \\ y[2] \\ y[3] \\ y[4] \\ y[5] \\ y[6] \end{bmatrix}$$

In DSP applications, the matrix containing $x[n]$ is called the *M*-sequence matrix and shall be referred to as such from this point onwards.

Correlation detection with the *M*-sequence matrix can become prohibitively complex to compute as P becomes large. Applications in room acoustics can involve impulse responses of 5 seconds or more, which at a sample rate of 48 kHz requires P to be in the order of 2^{18} samples long. Correlation detection requires P^2 operations, which on a 100MHz DSP processor would take approximately 11 minutes to calculate.

Computational complexity can be dramatically reduced by exploiting similarities between the *M*-sequence matrix and the Walsh-Hadamard matrix, reducing the number of additions to $P\log_2(P)$ by use of the Fast Walsh-Hadamard Transform.

The *M*-sequence matrix may be described by:

$$m_{i,j} = n_{i+j-2(\text{modulo}-P)} \qquad i,j = 1,2,\ldots,P$$

Or successive circular left shifts of the top row. For *P*=7:

$$\mathbf{M} = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

From the construction of **M**, every row (column) of **M** can be expressed as a linear, modulo-two combination of the first *N* rows of **M**.

$$\mathbf{M} = \mathbf{LS} = \mathbf{S'L'} \qquad (3.11)$$

Where **L** is a binary matrix order $P \times N$ and **S** is order $N \times P$ and is formed by the first *N* rows of **M**.

Since all the rows of **M** are distinct, all the rows of **L** must be distinct, therefore every nonzero binary *N*-vector must appear as some row in **L**. Also, the first *N* rows of **L** form an identity matrix of *N*.

Let **σ** be the square matrix of order *N*, formed by the first *N* columns of **S**. Then,

$$\mathbf{L\sigma} = \mathbf{S'} \qquad (3.12)$$

Every nonzero binary *N*-vector appears as a row in both L and S', **σ** is necessarily non-singular, so,

$$\mathbf{L} = \mathbf{S'\sigma^{-1}} \qquad (3.13)$$

Using the above example,

$$\mathbf{S} = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 \end{bmatrix} \qquad (3.14)$$

$$\boldsymbol{\sigma} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \qquad (3.15)$$

$$\boldsymbol{\sigma}^{-1} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \qquad (3.16)$$

$$\mathbf{L} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix} \tag{3.17}$$

3.3 Walsh-Hadamard Transforms

A Hadamard Matrix may be constructed by the following recursive relation:

$$\mathbf{H}_{n+1} = \begin{bmatrix} \mathbf{H}_n & \mathbf{H}_n \\ \mathbf{H}_n & \overline{\mathbf{H}}_n \end{bmatrix} \tag{3.18}$$

Where $\overline{\mathbf{H}}$ denotes a binary inversion. An 8-th order Hadamard Matrix is exemplified in (3.19):

$$\begin{array}{ccc|cccccccc} & & & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ & & & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ & & & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{array} \tag{3.19}$$

Where the surrounding numbers are the binary indices of each row/column.

In a similar fashion to the factorisation of **M** into **S** and **L, H** may be factored into **B** and **B'**, where **B** is a binary representation of the numbers 0 to $2^{N\text{-}1}$.

$$\mathbf{B} \cdot \mathbf{B'} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} = \mathbf{H} \tag{3.20}$$

Denote $\hat{\mathbf{L}}$ and $\hat{\mathbf{S}}$ the matrices obtained by bordering the former on top with a row of all zeros and the latter on the left by a column of all zeros. The bordered matrix $\hat{\mathbf{M}}$ is then given by:

$$\hat{\mathbf{M}} = \hat{\mathbf{L}}\hat{\mathbf{S}} \tag{3.21}$$

The factors $\hat{\mathbf{L}}$ and **B**, $\hat{\mathbf{S}}$ and **B**' each exhaust all *N*-vectors in their rows and columns respectively. They differ only in the order in which the vectors occur, so we can write:

$$\hat{\mathbf{L}} = \mathbf{P_L B} \tag{3.22}$$

$$\hat{\mathbf{S}} = \mathbf{B' P_S} \tag{3.23}$$

Where $\mathbf{P_L}$ and $\mathbf{P_S}$ are $2^N \times 2^N$ permutation matrices describing the *N*-vector orderings in $\hat{\mathbf{L}}$ and $\hat{\mathbf{S}}$. This simple relation is due to the fact that the rows of **B** and columns of **B**' occur in natural numerical order. Using previous relations,

$$\hat{\mathbf{M}} = \hat{\mathbf{L}}\hat{\mathbf{S}} = \mathbf{P_L BB' P_S} = \mathbf{P_L H P_S} \tag{3.24}$$

Relation (3.24) holds regardless of whether values {0,1} or {1, -1} are used. An *M*-sequence transform can be summarised by:

1. Reorder the received data from the device under test after stimulation with an MLS signal according to $\mathbf{P_S}$.
2. Perform a Walsh-Hadamard transform.
3. Reorder the transformed data according to $\mathbf{P_L}$ and divide by P+1.

Note that the Walsh-Hadamard transform by matrix multiplication gains no computational advantage; instead the Fast Walsh-Hadamard transform should be used.

3.4 The Fast Walsh-Hadamard Transform

There are many different algorithms for calculating a 'fast' Walsh-Hadamard Transform. The following is one of the most elegant:

1. Construct *N*+1 columns, each with 2^N rows.
2. Place the input vector $x[n]$ in the 1st column.
3. Moving left to right, fill in the next column as follows:
   a. In the top half of this column, place the pairwise, mutually exclusive sums of the previous column.

x	**a**	**b**	**y**
x[0]	a[0]=x[0]+x[1]	b[0]=a[0]+a[1]	y[0]=b[0]+b[1]
x[1]	a[1]=x[2]+x[3]	b[1]=a[2]+a[3]	y[1]=b[2]+b[3]
x[2]	a[2]=x[4]+x[5]	b[2]=a[4]+a[5]	y[2]=b[4]+b[5]
x[3]	a[3]=x[6]+x[7]	b[3]=a[6]+a[7]	y[3]=b[6]+b[7]
x[4]	a[4]=x[0]-x[1]	b[4]=a[0]-a[1]	y[4]=b[0]-b[1]
x[5]	a[5]=x[2]-x[3]	b[5]=a[2]-a[3]	y[5]=b[2]-b[3]
x[6]	a[6]=x[4]-x[5]	b[6]=a[4]-a[5]	y[6]=b[4]-b[5]
x[7]	a[7]=x[6]-x[7]	b[7]=a[6]-a[7]	y[7]=b[6]-b[7]

Figure 3.3: A Fast Hadamard Transform

b. In the bottom half, place the pairwise differences.
c. Repeat (Step 3) for each of the remaining columns.

The last column contains the Hadamard transform, $y[n]$, of the input vector, $x[n]$.

3.5 Alternative Permutation

The matrices $\mathbf{P_S}$ and $\mathbf{P_L}$ are presented here merely to clarify the underlying theory. In practice the permutation operation may be calculated by finding the sum of the columns of **S** and the rows of **L** considered as a binary number, then reordering according to these values as follows:

$$\begin{array}{c} \times 4 \\ \times 2 \\ \times 1 \end{array}\begin{bmatrix} 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 \end{bmatrix}$$
$$\begin{pmatrix} 1 & 2 & 5 & 3 & 7 & 6 & 4 \end{pmatrix}$$

$$\begin{pmatrix} y[1] & y[2] & y[3] & y[4] & y[5] & y[6] & y[7] \end{pmatrix} \rightarrow$$
$$\begin{pmatrix} y[1] & y[2] & y[4] & y[7] & y[3] & y[6] & y[5] \end{pmatrix}$$

$$\begin{array}{ccc} \times 4 & \times 2 & \times 1 \end{array}$$
$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}\begin{pmatrix} 4 \\ 2 \\ 1 \\ 6 \\ 3 \\ 7 \\ 5 \end{pmatrix}$$

$$\begin{pmatrix} h[4] & h[2] & h[1] & h[6] & h[3] & h[7] & h[5] \end{pmatrix} \rightarrow$$
$$\begin{pmatrix} h[1] & h[2] & h[3] & h[4] & h[5] & h[6] & h[7] \end{pmatrix}$$

3.6 DC Coupling

Some papers [2], [6], have wrongly suggested methods for recovering DC components with MLS analysis. It is shown in [3] that if a system is AC coupled (as is the case in almost audio applications), sum of $h[n]$ over all n is zero:

$$\sum_n h[n] = 0 \tag{3.25}$$

It is suggested in [2] that in order to recover a DC component then equation (x) should be re-written with a DC offset term as follows:

$$h'[n] = \frac{1}{P+1}\left(\sum_{k=1}^{P} y[k]x[k-n] - \sum_{k=1}^{P} y(k) \right) \tag{3.26}$$

Which is equivalent to bordering **L** and **S** with $-\sum_n y[n]$ instead of zeros to form $\hat{\mathbf{L}}$ and $\hat{\mathbf{S}}$.

A simple argument against doing this is that in the case of a DC coupled system, the DC component may only by extracted either by infinitely long or assymetrical MLS stimuli, both of which are outside the scope of this project. I ignore the DC term entirely as it is inconsequential to loudspeaker equalisation.

3.7 SNR

Let the system's impulse response be *h*(*t*) and the noise *n*(*t*). Noise sources include quantisation (shot noise), thermal noise in the amplifiers and microphone and background noise. The signal received at the analyser is $g(t)=h(t)+n(t)$.

Averaging successive MLS bursts (again ignoring the first burst), we obtain the signal $\overline{g}(t)$. Assume that the noise process is ergodic, zero mean and white (it has an autocorrelation function of 0 everywhere but k=0):

$$E\{n(t)\}=0$$

$$R[k]=\frac{1}{P}\sum_{t=0}^{P-1-k} n(t)n(t+k)=\sigma_{n(t)}^2\delta(k) \qquad (3.27)$$

Assume also that the system's impulse response is time-invariant.

$\overline{g}(t)=\frac{1}{P}\sum_{i=1}^{P} g_i(t)=\frac{1}{P}\sum_{i=1}^{P} h(t)+n_i(t)=h(t)+\frac{1}{P}\sum_{i=1}^{P} n_i(t)$ The variance of the noise is given by:

$$\sigma_{n(t)}^2=E\left\{n^2(t)\right\}=E\left\{\left(\frac{1}{P}\sum_{i=1}^{P} n_i(t)\right)^2\right\}=\frac{1}{P^2}E\left\{\left(\sum_{i=1}^{P} n_i(t)\right)^2\right\}$$

$$=\frac{1}{P^2}E\left\{\left(\sum_{i=1}^{P} n_i^2(t)\right)\right\}+\frac{1}{P^2}\left\{\left(\sum_{i=1}^{P}\sum_{j=1}^{P}\left(n_i(t)n_j(t)\right)\right)\right\}$$

$$=\frac{1}{P^2}E\left\{\left(\sum_{i=1}^{P} n_i^2(t)\right)\right\}+\frac{1}{P^2}\sum_{i=1}^{P}\sum_{j=1}^{P}E\left\{n_i(t)n_j(t)\right\}$$

$$=\frac{1}{P^2}\sum_{i=1}^{P}\sigma^2+0$$

$$=\frac{1}{P}\sigma^2 \qquad (3.28)$$

Therefore as $P\to\infty, \sigma_{n(t)}^2\to 0$. As signal power is proportional to its variance, for every doubling in the number of averaged sequences, the SNR is improved by 3dB.

3.8 Practical Measurements

The frequency response at different amplitudes is obtained by analysing bursts of MLS at increasing amplitude. In order to correctly time-align the analysis software, an impulse of one sample is used as a sync pulse and the alignment of the MLS bursts is referenced to this point. Details of the issues of correct time-alignment and general information about the practical implementation of MLS code are given in Appendix 1.

3.9 Summary

MLS analysis is a computationally-efficient method for extracting the impulse response of linear systems. The steps for conducting an MLS analysis are as follows:

1. Determine the approximate length of a system's impulse response. Generate an MLS sequence at least as long as the impulse response using a binary feedback register and XOR gate as described in equation (1). Convert the sequence using the mapping $0 \rightarrow 1$, $1 \rightarrow -1$.
2. Stimulate the system with at least two successive MLS bursts and record the output. The SNR is improved by 3dB for a doubling in the number of repetitions.
3. Take the mean of all but the first burst.
4. Add a zero to the first element and reorder the averaged data according to $\mathbf{P_S}$.
5. Perform a Fast Walsh-Hadamard transform.
6. Reorder the transformed data according to $\mathbf{P_L}$ and divide by $P+1$.
7. The results is an estimation of a system's impulse response of length $P+1$. Perform an FFT to find the frequency-domain representation.

The importance of approximating circular convolution is often overlooked and can yield highly inaccurate results if it is ignored.

See the user guide in Appendix 1 and the Matlab code in Appendix 3 for further information about how this algorithm is implemented.

4 Nonlinear Modelling

This chapter presents an overview of the major causes of nonlinearity in loudspeakers and how they can be modelled. We use the assumption that loudspeaker nonlinearity can be characterised by a second-order Volterra approximation (as demonstrated in [11] and [12]) and describe two methods for extracting the Volterra Kernels from measurements. The latter method is used in Chapter 7 to determine the validity of a second-order model with real results.

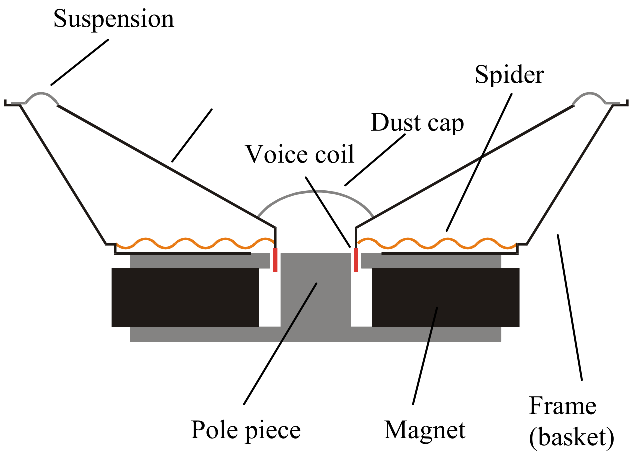


Figure 4.1: A typical dynamic loudspeaker under rest conditions

4.1 Loudspeaker Construction

An *electrodynamic* loudspeaker is a transducer which converts a voltage to a sound pressure wave using a current-passing coil in a static magnetic field. The coil carries the driving current and is surrounded by a permanent magnet with an air spacing. The current flow in the coil causes it to experience a force, moving the coil perpendicularly to the magnetic field. The diaphragm is glued to the coil and is connected to the frame by a suspension and spider, allowing it to move in and out but not laterally, keeping it centred between the magnets. See *Figure 4.1* for a simplified diagram.

4.2 Sources of nonlinearity

Nonlinearities can be grouped into three main categories [11]:

4.2.1 Motor Part

For low excursions, the coil experiences a linear field. As excursion increases, the coil moves towards the edge of the magnetic field (see *Figure 4.2*), reducing the force factor, $\int Bdl$, and therefore the force upon the coil.

The self-inductance of the voice coil is also position-dependent, because it protrudes from the central pole. This yields a reluctance force proportional to the square of the current.

$$F_x = \frac{1}{2} i^2 \frac{\mathrm{d}L(x)}{\mathrm{d}x} \tag{4.1}$$

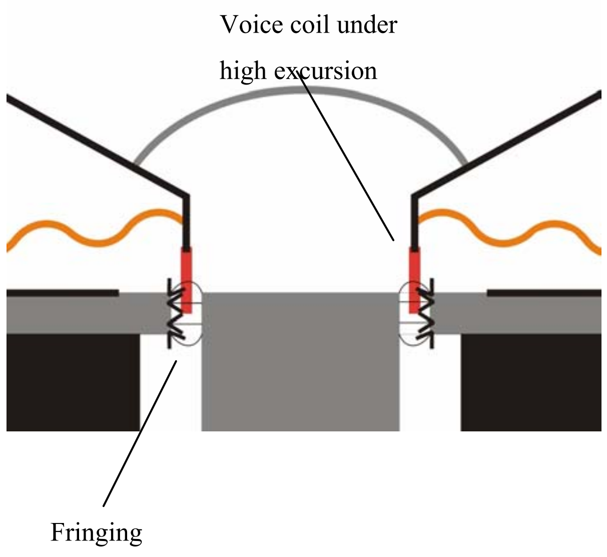


Figure 4.2: Voice coil under high excursion. The arrows depict the magnetic flux lines. The voice coil is mostly in the fringing field, causing a lower force to act upon the coil for a given current.

The voltage across the self-inductance is given by more than the time derivative, showing the following relation:

$$U = L(x)\frac{\mathrm{d}i}{\mathrm{d}t} + i\frac{\mathrm{d}L(x)}{\mathrm{d}x}\frac{\mathrm{d}x}{\mathrm{d}t} \qquad (4.2)$$

Finally, the operating point of the permanent magnet is influenced by the voice coil current.

4.2.2 Mechanical Part

The force-displacement curves of the spider and suspension show some hysteresis (*Figure 4.3*). The maximum excursion is limited by mechanical clipping, though this only occurs at extreme drive levels. Subharmonics are generated at the loudspeaker diaphragm due to nonlinearities in the cone material, again only occurring at extreme drive levels.

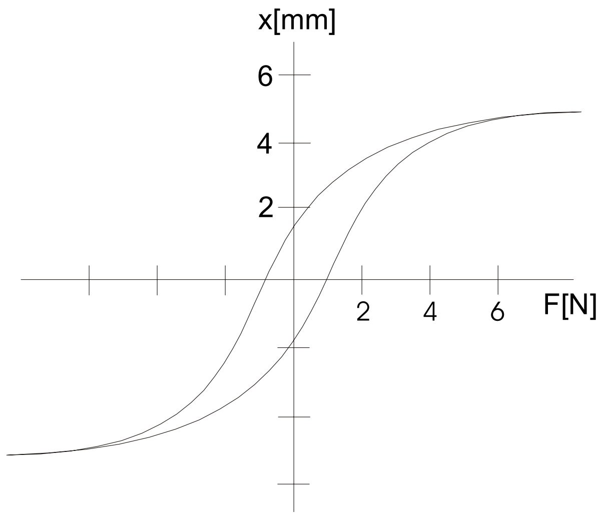


Figure 4.3: Typical force-displacement curve of suspension

4.2.3 Sound Radiation

Let p_0 be the static pressure inside the loudspeaker enclosure and V_0 be the static volume. Let (p_0+p) be the instantaneous pressure inside the loudspeaker enclosure and (V_0+V) be the instantaneous volume. Assuming that the box air compression is an adiabatic process, the air in the box obeys the following relation:

$$(p_0 + p)(V_0 + V)^{\gamma} = p_0 V_0^{\gamma} \qquad (4.3)$$

Which is clearly nonlinear.

The final cause of nonlinearity is the Doppler shift caused when the driving signal contains high-amplitude low-frequency signals and high-frequency signals.

4.2.4 Parameters

At low frequencies, where voice-coil excursion is greatest, the electrical, mechanical and acoustical components of a loudspeaker behave as concentrated elements, and may be modelled as lumped-elements in an equivalent electromechanical circuit. A loudspeaker may be modelled in terms of a damped mass-spring system, with nonlinear springs and damping mechanisms. The following is a simplified version of the model found in [11]:

$$\left.\begin{array}{cc} \int B\mathrm{d}l & \text{Force factor} \\ k(x) & \text{Stiffness of mass-spring system} \\ L_E(x) & \text{Voice coil inductance} \end{array}\right\}\text{Nonlinear parameters}$$

$$\left.\begin{array}{cc} E & \text{Generator voltage} \\ i & \text{Input current} \\ x & \text{Voice coil excursion} \\ \dot{x} & \text{Voice coil velocity} \\ m & \text{Moving mass} \\ R_m & \text{Mechanical damping} \\ R_E & \text{Electrical resistance of voice coil} \end{array}\right\}\text{Linear parameters}$$

The nonlinear parameters can be approximated with a 2nd order power series:

$$\begin{aligned} Bl &= Bl_0 + b_1 x + b_2 x^2 \\ k &= k_0 + k_1 x + k_2 x^2 \\ L_E &= L_{E_0} + l_1 x + l_2 x^2 \end{aligned} \qquad (4.4)$$

From these definitions we can form two differential equations, which can be solved to give a polynomial (Volterra) series, describing voice coil displacement as a function of current (or voltage).

$$\begin{aligned} E &= R_E i + \frac{d(L_E i)}{dt} + Bl\dot{x} \\ Bli &= m\ddot{x} + R_m \dot{x} + kx \end{aligned} \qquad (4.5)$$

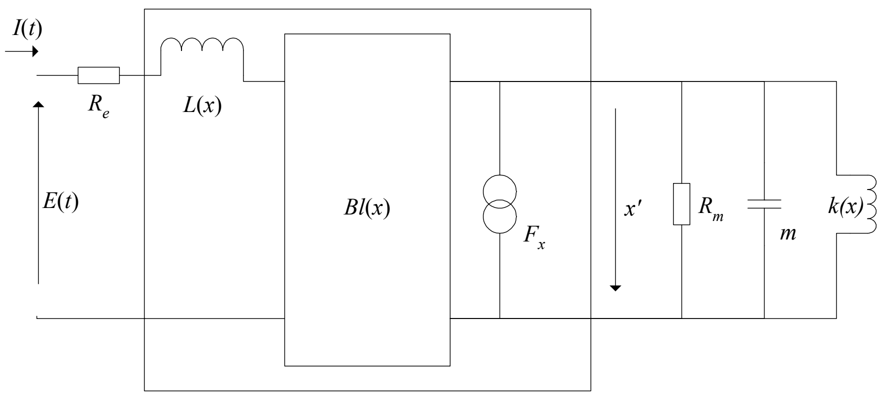


Figure 4.4: A simple electromechanical model of a loudspeaker in a closed-box enclosure. L(x), k(x) and Bl(x) are all nonlinear functions of voice coil excursion.

Higher order models are suggested in [12] and *k* can be split into a function of nonlinear air stiffness and mechanical stiffness, increasing the complexity of the nonlinear differential equation. Solving this is outside the scope of the project as it requires detailed knowledge of the loudspeaker parameters.
[11] and [12] agree that for frequencies of up to about 250Hz, a second-order Volterra series approximation is sufficient to describe the nonlinear behaviour of a loudspeaker, so it will be shown later that experimental results can be used to develop such a model.

4.3 Volterra Series

A Volterra Series is a concise and elegant method for characterising weakly nonlinear systems. It is assumed that the nonlinearity in the transfer function can be approximated by an n^th order polynomial:

$$y = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n \tag{4.6}$$

Often the DC (bias) term, a_0, is ignored. Nonlinear systems cannot be characterised by an impulse response alone as the output is not proportional to input amplitude; we therefore replace the notion of an impulse response with a *Volterra Kernel*. The Volterra Kernel describes each order of *x* as a matrix of increasing dimension:

$$\begin{aligned}
&\text{DC} : h_0 \\
&\text{1st order (linear)} : h_1(\tau) \text{ or } h_1[k] \\
&\text{2nd order} : h_2(\tau_1, \tau_2) \text{ or } h_2[k_1, k_2] \\
&\text{3rd order} : h_3(\tau_1, \tau_2, \tau_3) \text{ or } h_3[k_1, k_2, k_3]
\end{aligned}$$

And so on. h_1 is a vector, h_2 is a symmetric $n \times n$ matrix and h_3 is a symmetric $n \times n \times n$ matrix.
Convolution is described as:

$$\begin{aligned}
y(t) &= h_0 + \int_0^\infty h_1(\tau_1) x(t-\tau_1) \mathrm{d}\tau_1 + \int_0^\infty \int_0^\infty h_2(\tau_1, \tau_2) x(t-\tau_1) x(t-\tau_2) \mathrm{d}\tau_1 \mathrm{d}\tau_2 + \ldots \\
y[n] &= h_0 + \sum_{k=0}^{N} h_1[n-k] x[k] + \sum_{k_1=0}^{N} \sum_{k_2=0}^{N} h_2[n-k_1, n-k_2] x[k_1] x[k_2] + \ldots
\end{aligned} \tag{4.7}$$

Clearly the computational complexity of Volterra convolution increases exponentially with the order of the Kernel.
Many parametric and non-parametric methods exist for estimating Volterra Kernels. I shall present two popular forms which will be applied to experimental results later on.

4.3.1 2nd Order Volterra System Identification for arbitrary input signals: Powers's Method [22]

The frequency-domain input-output relationship of the second-order Volterra system under consideration is:

$$Y(f) = H_1(f) X(f) + \sum_{f_1 + f_2 = f} H_2(f_1, f_2) X(f_1) X(f_2) \tag{4.8}$$

Assuming $H_2(f_1, f_2) = H_2(f_2, f_1) = H_2 * (-f_1, -f_2)$. Transfer functions $H_1(f)$ and $H_2(f_1, f_2)$ are obtained as the least-squares solution to the set of equations, $Y(m) = A^T\mathbf{b}(m)$, where,

$$\mathbf{b}(m) = \left[H_1(m), H_2\left(\frac{m+1}{2}, \frac{m-1}{2}\right), H_2\left(\frac{m+3}{2}, \frac{m-3}{2}\right), ..., H_2\left(\frac{M}{4}, m - \frac{M}{4}\right) \right]^T \qquad (4.9)$$

and

$$A = \left[X(m), X\left(\frac{m+1}{2}\right)X\left(\frac{m-1}{2}\right), X\left(\frac{m+3}{2}\right)X\left(\frac{m-3}{2}\right), ..., X\left(\frac{M}{4}\right)X\left(m\frac{M}{4}\right) \right]^T \qquad (4.10)$$

Where *M* is the length of the FFT and *m*=0,…,*M*/2.

4.3.2 2nd Order Volterra System Identification for Gaussian input signals: Tick's Method [23]

A more common and computationally efficient method, Tick's algorithm uses Gaussian input signals and computes their cross-bispectra to obtain an approximation to the first- and second-order Volterra Kernels.

The output of a second-order Volterra system is given by:

$$y[n] = \sum_{k=0}^{N} h_1[n-k]x[k] + \sum_{k_1=0}^{N}\sum_{k_2=0}^{N} h_2[n-k_1, n-k_2]x[k_1]x[k_2]$$

The linear part $h_1[k]$, is estimated in the frequency domain from:

$$S_{yx} = H_1(\omega)S_{xx}(\omega) \qquad (4.11)$$

where $S_{xx}(\omega)$ is the power spectrum of process *x*[*n*] and $S_{xy}(\omega)$ is the cross-spectrum of the processes *x*[*n*] and *y*[*n*]. This relationship assumes that *x*[*n*] is symmetrically distributed.

The quadratic part $h_2(k_1,k_2)$ is estimated in the frequency domain from:

$$S_{yxx}(\omega_1, \omega_2) = 2H_2(\omega_1, \omega_2)S_{xx}(\omega_1)S_{xx}(\omega_2) + S_{xx}(\omega_2)\delta(\omega_1 + \omega_2)E(y[n]) \qquad (4.11)$$

which is obtained under the assumption that $h_2(k_1,k_2)=h_2(k_2,k_1)$ and *x*[*n*] is Gaussian. The cross-bispectrum, $S_{yxx}(\omega_1, \omega_2)$ is estimated via the direct (FFT) method in [23].

A simple test was derived to determine whether Tick's method could be used to recover the linear and quadratic terms for a virtual loudspeaker, with zero gain at DC and at the Nyquist frequency, and with arbitrary gain in between. Matlab's `FIR2` function was used to create an FIR filter of for the linear part ($h_1[\underline{n}]$):

```
h  = fir2(order-1, [0 0.2 0.4 0.6 0.8 1], [0 0.8 0.3 0.2  0.4  0]);
```

The quadratic part, $h_2[n]$, was made in a similar fashion and a symmetric matrix was defined:

```
g = fir2(order-1, [0 0.2 0.4 0.6 0.8 1], [0 0.8 0.3 0.2  0.4  0]);
h2 = g'*g;
```

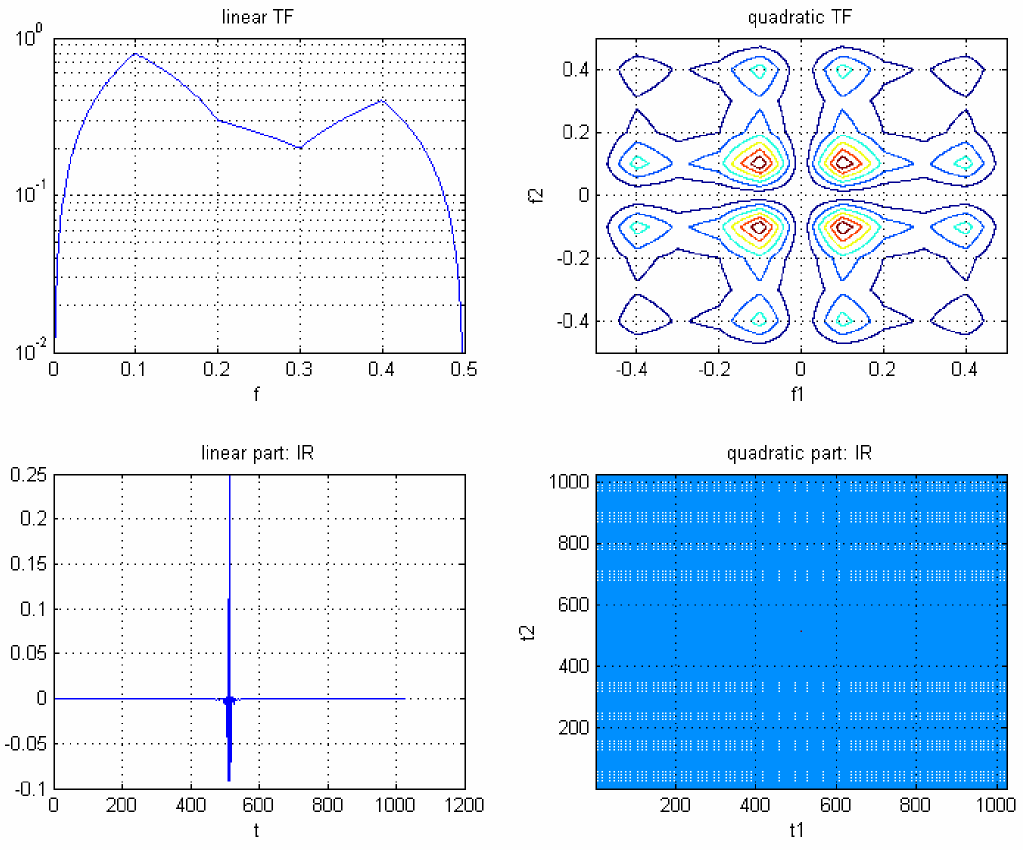


Figure 4.5: A virtual loudspeaker defined with linear and quadratic terms.

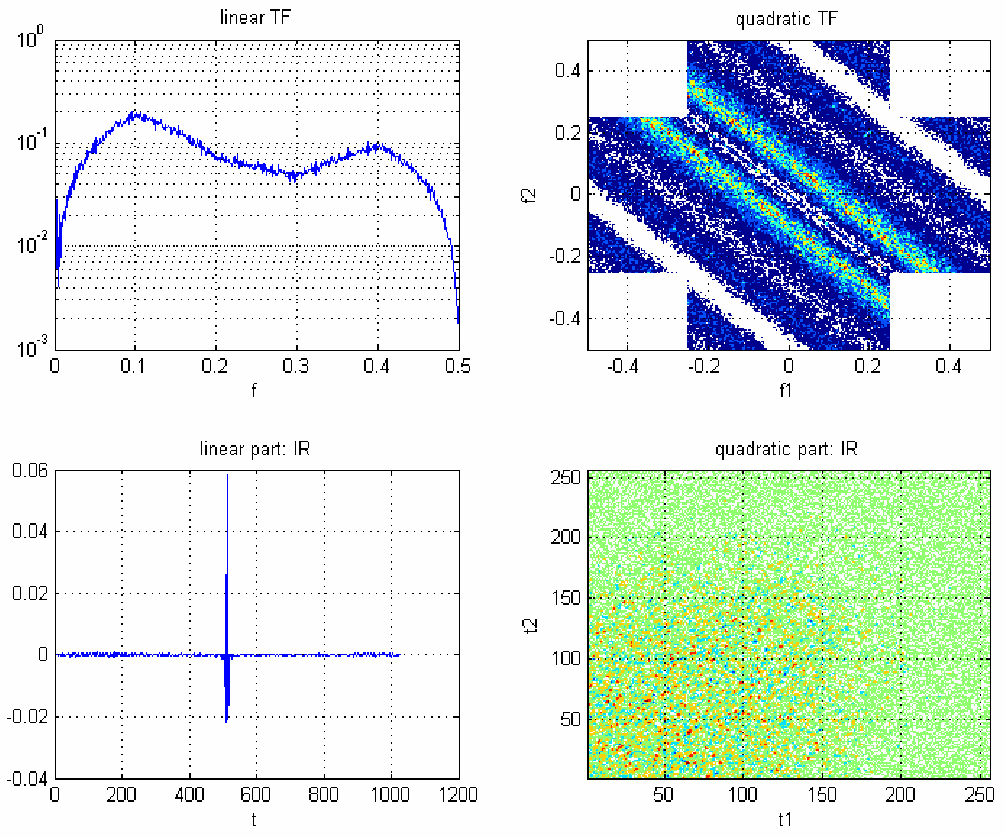


Figure 4.6: The recovered linear and quadratic transfer functions and impulse responses from filtered Gaussian noise using Tick's method.

Figure 4.5 shows the linear and quadratic transfer functions and impulse responses. A signal was generated consisting of 64 realisations of 1024-length Gaussian noise, which was then passed through a nonlinear convolution with the nonlinear system according to equation (4.6). A Matlab implementation of Tick's algorithm [26] was used to analyse the noise and extract the Volterra Kernels, the results of which are shown in *Figure 4.6.*

There is a clear resemblance between the original and recovered linear transfer functions, with the exception of reduced gain. The quadratic terms are markedly different but so too were the examples given in the reference material accompanying the Matlab code [26]. We see in Chapter 7 how this method can be applied to real measurements and discuss is validity.

4.4 Efficiency

Both methods are very inefficient compared with the MLS method in Chapter 3. A Gaussian signal of 1024 samples and 64 realisations takes about 30 minutes to analyse on a 2.7GHz Athlon processor under Matlab, compared with less than two seconds for a 4096-length MLS signal.

4.5 Summary

Nonlinearities in loudspeakers is caused primarily by voice-coil excursion and can be approximated with a second-order polynomial, described by Volterra Kernels. Some discrepancies exist in the analysis of a theoretical experiment, though whether they are a function of the algorithm or its implementation is unknown. Measurements presented in Chapter 7 are analysed with Tick's algorithm (4.3.2) with Matlab software obtained from [26] and demonstrate the validity of a second-order approximation.

5 Formal Equaliser Theory

This chapter presents equaliser theory in a formal mathematical way which describes the algorithms used in Chapter 7 and documented in Appendices 1 and 3. No known conventions exist for describing loudspeaker equalisers and the following is an attempt to develop a framework upon future work may be made.

Let us define the following symbols:

N_a	Number of driving amplitudes
N_f	Number of driving frequencies
$\mathbf{a} = [a_1, a_2, ..., a_{N_a}]$	Driving amplitudes
$a_{\min}, a_{\max}$	Lower and upper amplitude boundary
$a'_{\min}, a'_{\max}$	Indices of lower and upper amplitude boundaries
$\mathbf{f} = [f_1, f_2, ..., f_{N_f}]$	Driving frequencies
$f_{\min}, f_{\max}$	Lower and upper frequency boundary
$f'_{\min}, f'_{\max}$	Indices of lower and upper amplitude boundaries
$\mathbf{\Gamma}_{a,f}^{dB}$	SPL as a function of amplitude & frequency (range -∞ to 0 dBFS)
$\mathbf{\Phi}_{a,f}^{dB}$	Frequency-domain level-dependent equaliser (range -∞ to 0 dBFS)
$\mathbf{\Lambda}_{f}^{dB}$	Frequency-domain linear equaliser (range -∞ to 0 dBFS)

We shall use logarithmic units to represent the equalisers as they may be implemented by addition; this reduces computational complexity and simplifies their mathematical representation.

5.1 Level-Dependent Equaliser

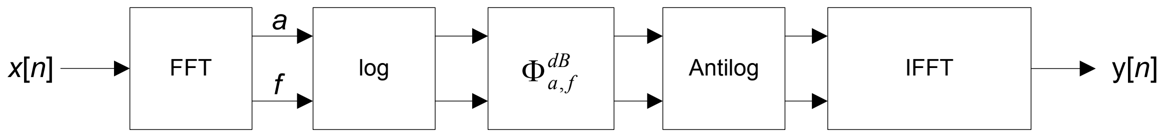


Figure 5.1: Flow diagram for a level-dependent equaliser. x[n] is the sequence to be analysed and y[n] is fed to the loudspeaker. The equaliser, $\Phi_{a,f}^{dB}$, is amplitude and frequency-dependent.

The ideal level-dependent equaliser gain (as a function of input amplitude and frequency) is the difference between the amplitude of the input signal and the SPL between two frequency boundaries. Outside these boundaries, the gain is set to the mean difference between the amplitudes of the input

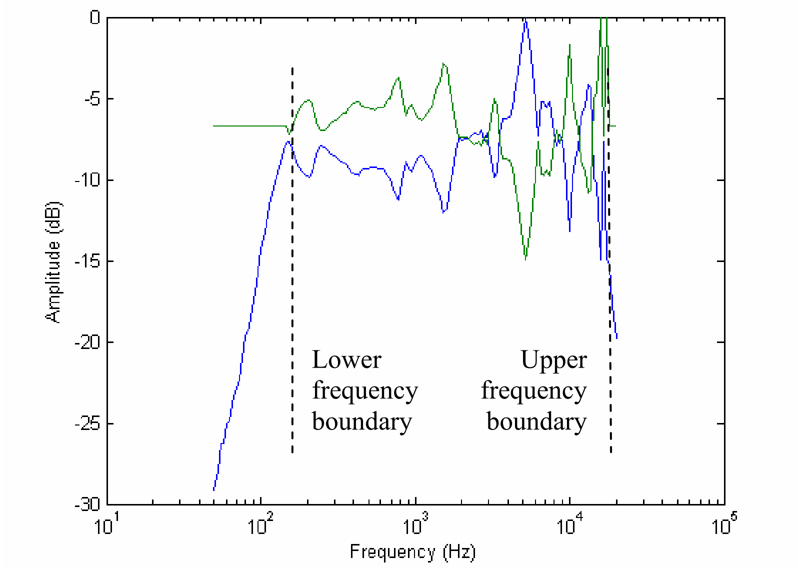


Figure 5.2: Equalisers are calculated between upper and lower frequency boundaries to prevent large gains at extreme low and high frequencies.

and the SPL (see *Figure 5.2*). The boundaries are set such that the equaliser gain doesn't become large at extreme LF and HF.

$$\mathbf{\Phi}_{a,f}^{dB} = \begin{cases} a - \mathbf{\Gamma}_{a,f}^{dB} & f = \{f_{\min}, f_{\min+1}, \ldots, f_{\max-1}, f_{\max}\} \\ \left\langle a - \mathbf{\Gamma}_{a,\mathbf{f}_{\{f_{\min}, f_{\max}\}}}^{dB} \right\rangle & otherwise \end{cases} \tag{5.1}$$

Where $\langle\ \rangle$ denotes a mean average and is calculated with f in the range $[f_{\min}, f_{\max}]$.

5.2 Linear Equaliser

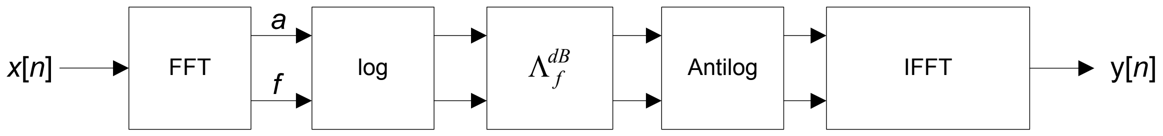


Figure 5.3: Flow diagram for a level-dependent equaliser. x[n] is the sequence to be analysed and y[n] is fed to the loudspeaker. The equaliser, Λ_f^{dB}, is frequency-dependent and amplitude-independent.

The ideal linear equaliser is the mean of the level-dependent equaliser across amplitudes, giving an equaliser which depends only upon frequency.

$$\mathbf{\Lambda}_f^{dB} = \frac{1}{a'_{\max} - a'_{\min}} \sum_{i=a'_{\min}}^{a'_{\max}} \mathbf{\Phi}_{\mathbf{a}_i,f}^{dB} \tag{5.2}$$

5.3 Volterra Equaliser

Due to time restrictions, a Volterra equaliser has not been implemented. Conceptually it may be implemented with a simple nonlinear feedback loop as shown in *Figure 5.4*:

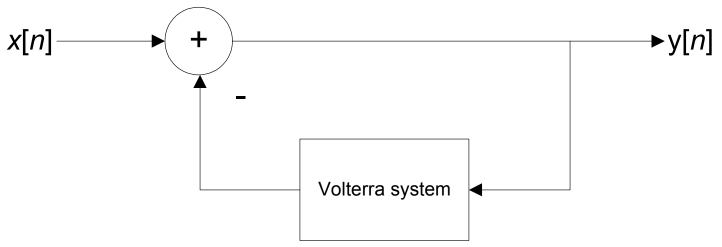


Figure 5.4: Volterra system used as a nonlinear feedback loop. The Volterra series is restricted to equalise inside lower and upper frequency boundaries only.

The Volterra system implements equation (4.7) using predetermined kernels. As with the linear and level-dependent equalisers, the equaliser should not affect frequencies outside the lower and upper frequency boundaries.

5.4 Equaliser Gain

In cases when a loudspeaker has been tested to extreme voice coil excursions, an equaliser should not introduce gain into the system. The gain of the equalisers derived in 5.1 and 5.2 should therefore be reduced so that the maximum gain is 0dBFS.

$$\mathbf{\Phi}_{a,f}^{dB} \to \mathbf{\Phi}_{a,f}^{dB} - \max\left(\mathbf{\Phi}_{a,f}^{dB}\right) \tag{5.3}$$

$$\mathbf{\Lambda}_{f}^{dB} \to \mathbf{\Lambda}_{f}^{dB} - \max\left(\mathbf{\Lambda}_{f}^{dB}\right) \tag{5.4}$$

The drawback of reducing the overall equaliser gain in this way is that it may cause the loudspeaker's operating region to change significantly, reducing the equaliser's accuracy. Omission of the gain terms in equations 5.3 and 5.4 minimises this effect but carries associated risk.

5.5 Phase Considerations

Under linear convolution, it is not possible to arbitrarily choose the frequency and phase of each FFT bin; this is only possible under circular convolution. The phase of the system is left unaffected, changing only the magnitude with no attempt to investigate phase equalisation due to time constraints.

5.6 Variance Analysis

A measurement of the deviation from a flat frequency response is to measure its mean variance across amplitudes:

$$\sigma_{Unequalised}^2 = \frac{1}{a'_{\min} - a'_{\max}} \sum_{i=a'_{\min}}^{a'_{\max}} \sum_{j=f'_{\min}}^{f'_{\max}} \left(\mathbf{\Gamma}_{\mathbf{a}_i,\mathbf{f}_j}^{dB} - \mu_i\right)^2 \quad \text{where } \mu_i = \frac{1}{f'_{\max} - f'_{\min}} \sum_{j=f'_{\min}}^{f'_{\max}} \mathbf{\Gamma}_{\mathbf{a}_i,\mathbf{f}_j}^{dB} \tag{5.5}$$

Then after equalisation,

$$\sigma_{Equalised}^2 = \frac{1}{a'_{\min} - a'_{\max}} \sum_{i=a'_{\min}}^{a'_{\max}} \sum_{j=f'_{\min}}^{f'_{\max}} \left(\mathbf{\Gamma}_{\mathbf{a}_i,\mathbf{f}_j}^{dB} + \mathbf{\Phi}_{\mathbf{a}_i,\mathbf{f}_j}^{dB} - \mu_i\right)^2 \quad \text{where } \mu_i = \frac{1}{f'_{\max} - f'_{\min}} \sum_{j=f'_{\min}}^{f'_{\max}} \left(\mathbf{\Gamma}_{\mathbf{a}_i,\mathbf{f}_j}^{dB} + \mathbf{\Phi}_{\mathbf{a}_i,\mathbf{f}_j}^{dB}\right)$$

(5.6)

5.7 Summary

This chapter has demonstrated a method for formally describing the function of linear and level-dependent equalisers in the context of loudspeaker equalisation. It will not be found in any literature and should not be thought of as definitive terminology.

6 Hardware

This chapter briefly discusses the hardware setup, calibration techniques and problems encountered.

6.1 Apparatus

- BBC R&D Free-Field Room
- Brüel & Kjær 4144 measurement microphone
- Brüel and Kjær 2010 heterodyne analyser (used as mic preamp)
- Professional audio power amplifier
- RME Hammerfall DSP PCMCIA adapter & Multiface II (24-bit, 96 kHz)
- Laptop
- Prism Sound dScope III PC-based audio oscilloscope
- Brüel and Kjær 2231 Precision Sound Level Meter

In order to accurately measure a loudspeaker's characteristics, it must be placed in an environment where background noise and room acoustics do not add significantly to the measured signal. Depicted in *Figure 6.1* is a Free-Field Room (FFR) or Anechoic Chamber. It is a room measuring approximately $10 \times 10 \times 10$m whose walls are lined with 1-2m long open-pore foam spikes which are highly absorbent at audio frequencies. The loudspeaker is mounted on a small metal platform in the centre of the room and suspended from the ceiling is a measurement microphone (assumed perfect).

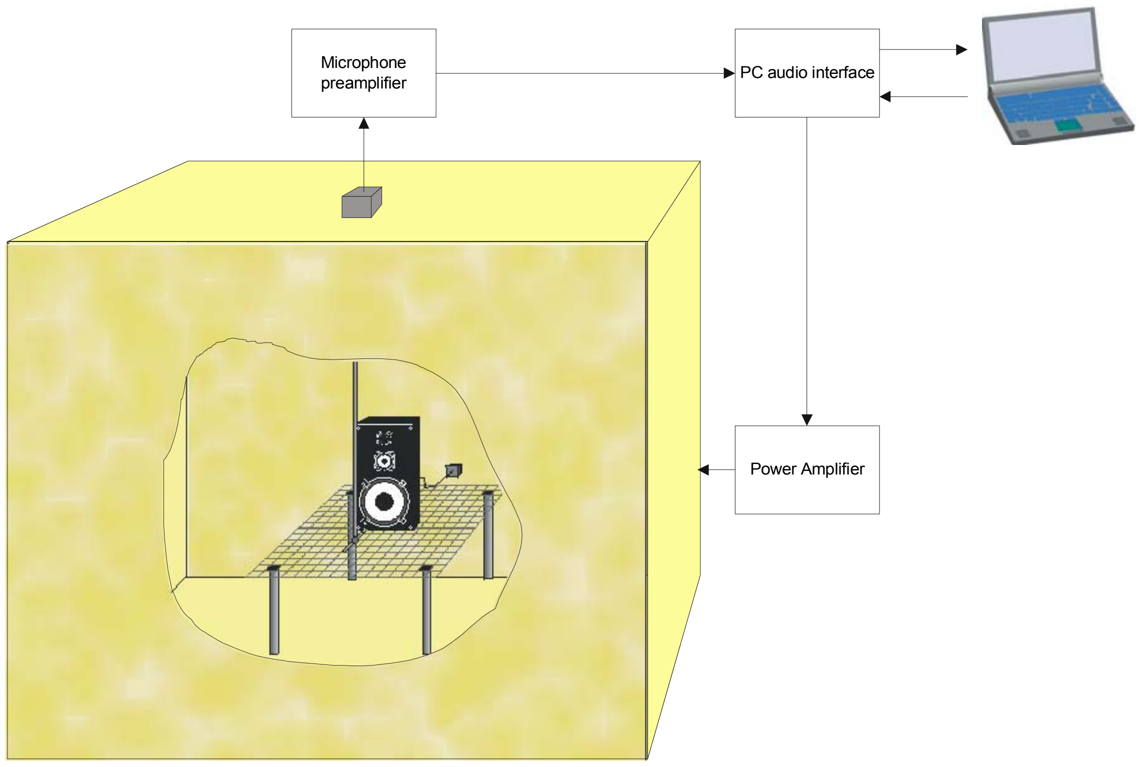


Figure 6.1: Equipment set-up. The loudspeaker is placed in the centre of an anechoic chamber and is measured with a high-quality microphone. The power amplifier, microphone preamplifier, audio interface and PC reside outside the chamber and connect via patchbays.

All other equipment is outside the chamber and is connected through patchbays.

6.2 Setup notes

The loudspeaker should be placed as centrally as possible in the chamber, with the microphone at exactly 1m on-axis from the front of the speaker.

It is important that a 0dBFS input signal gives a 0dBFS output signal for the generation of equalisers to work. It is also important to drive the loudspeaker hard enough to produce significant nonlinear effects (100-110dBA at 1m is probably enough for domestic loudspeakers). The calibration procedure is as follows:

1. Use the line-up generator to generate a noise burst at 0dBFS and adjust the power amplifier gain until an SPL of 100-110dBA is measured.
2. Adjust the preamplifier and PC mixer gain so that a -24dBFS line-up burst gives a -24dBFS measured peak amplitude. We use -24dBFS instead of 0dBFS to avoid the nonlinear region, which would cause over-estimated results if used as the line-up level.

In our treatment of loudspeaker modelling, we assume that it is a time-invariant system. This is valid if there is not significant temperature variation in the voice coils, so it is important to warm them to a steady state temperature before undertaking any measurements. White noise at ~85dBA was played through the loudspeaker under test for 20 minutes; this is only approximate unless temperature measurement is available but was adhered to upon each measurement session nonetheless.

6.3 Measurement Software

A Matlab toolbox was written to provide command-line utilities for the generation and analysis of MLS and stepped-sine signals and to create various types of equalisers. The functions were wrapped up into a GUI which allows a technician to calibrate a measurement setup, stimulate the system with various type of test signal, generate equalisers and test within a few minutes. A full user guide is found in Appendix 1.

Initial measurements were made with a Prism dScope III, which is an audio measurement oscilloscope that measures the amplitude response of a system with swept sine tones. The calibration procedure was identical to that used with the later Matlab software. Data was exported in the form of comma-separated variable files for each measurement amplitude which could be imported using a Matlab script. Details of this software are not presented in this report as they are now redundant, but are provided on the accompanying CD.

7 Results and Volterra Series Approximation

Here we shall examine the results of a Wharfedale Triton 3 loudspeaker. The test equipment was calibrated to produce ~100dBA at 1m, producing weak nonlinearity. The loudspeaker is capable of producing sounds at least 15dB louder but was driven moderately for fear of irreparable damage. MLS bursts of order 11 were repeated 32 times in 6dB increments from -96dBFS to 0dBFS. Lower and upper frequency thresholds were set at 90Hz and 16kHz respectively for specifying the equaliser.

7.1 MLS Analysis at Multiple Amplitudes

Figure 7.1 shows that this particular loudspeaker is far from ideal. It sounds particularly ‘bass-heavy’ and this can be seen in the large hump at around 110Hz. It also demonstrates nonlinearity; the peak at ~130Hz varies in shape, as does the peak at 5kHz.

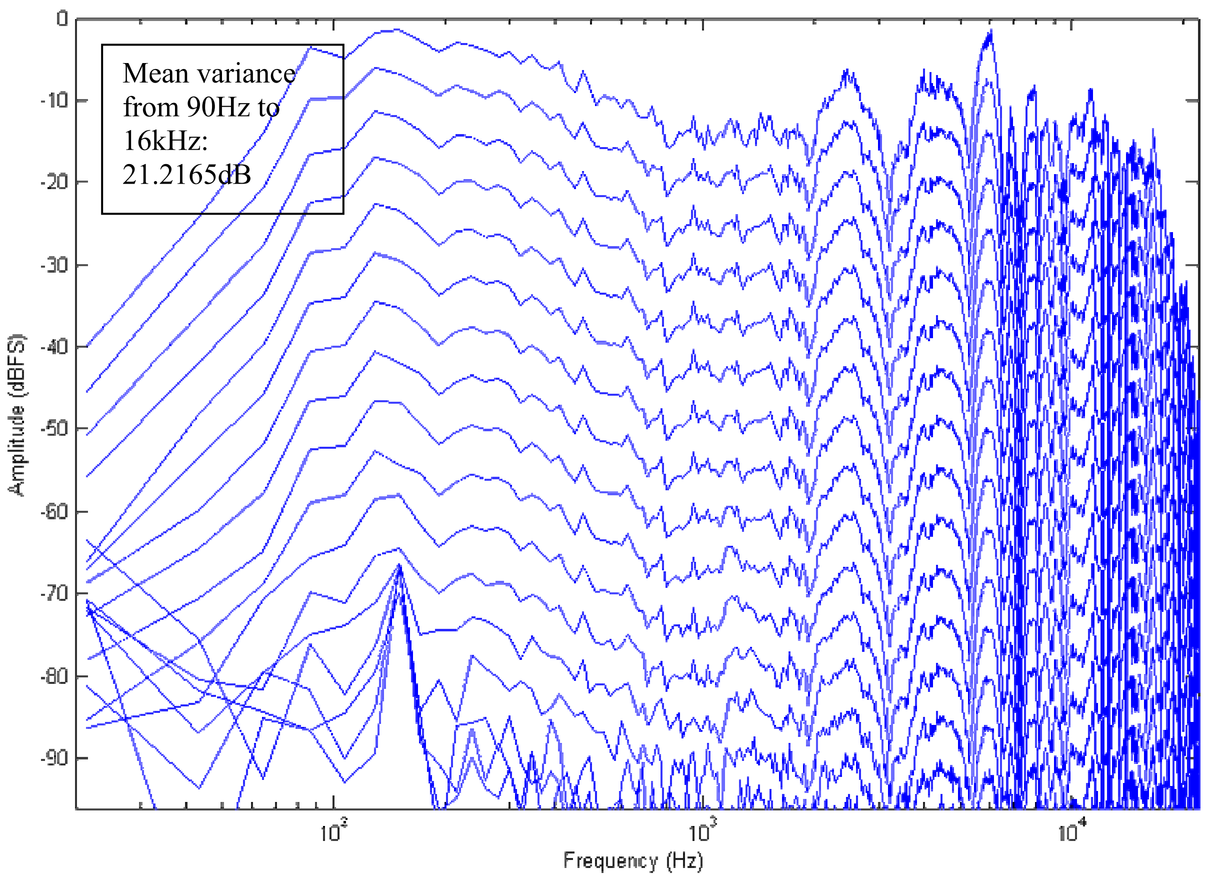


Figure 7.1: Frequency response of a Wharfedale Triton 3 at -96dBFS to 0dBFS where 0dBFS corresponds to 100dBA at 1m. The lower amplitudes are corrupted by noise but amplitudes from 38dBA upwards are clearly defined. Note the nonlinearity near the turnover point at ~90Hz.

7.2 Linear equaliser

A linear equaliser was defined for frequencies 90Hz-16kHz, amplitudes -78dBFS to 0dBFS and was used to pre-equalise an MLS sequence before playback through the loudspeaker. A non-attenuating equaliser was chosen as it was known that the loudspeaker would tolerate some additional gain; this is also more accurate as it produces a lower overall gain change and therefore reduces the effect of any nonlinearity as described in section 5.4.

The linear equaliser has significantly reduced the humps at 150Hz and 5kHz and the -18dB curve is almost completely flat. The 0dBFS curves was not equalised as well as the other curves as significant nonlinearity (and therefore distortion) was beginning to take effect. We will see in 7.4 that the THD at 0dBFS is significantly worse and that provides evidence for the preceding statement. The variance has reduced from 21dB to 0.22dB in the range of interest which is a significant improvement in the flatness of the frequency response overall.

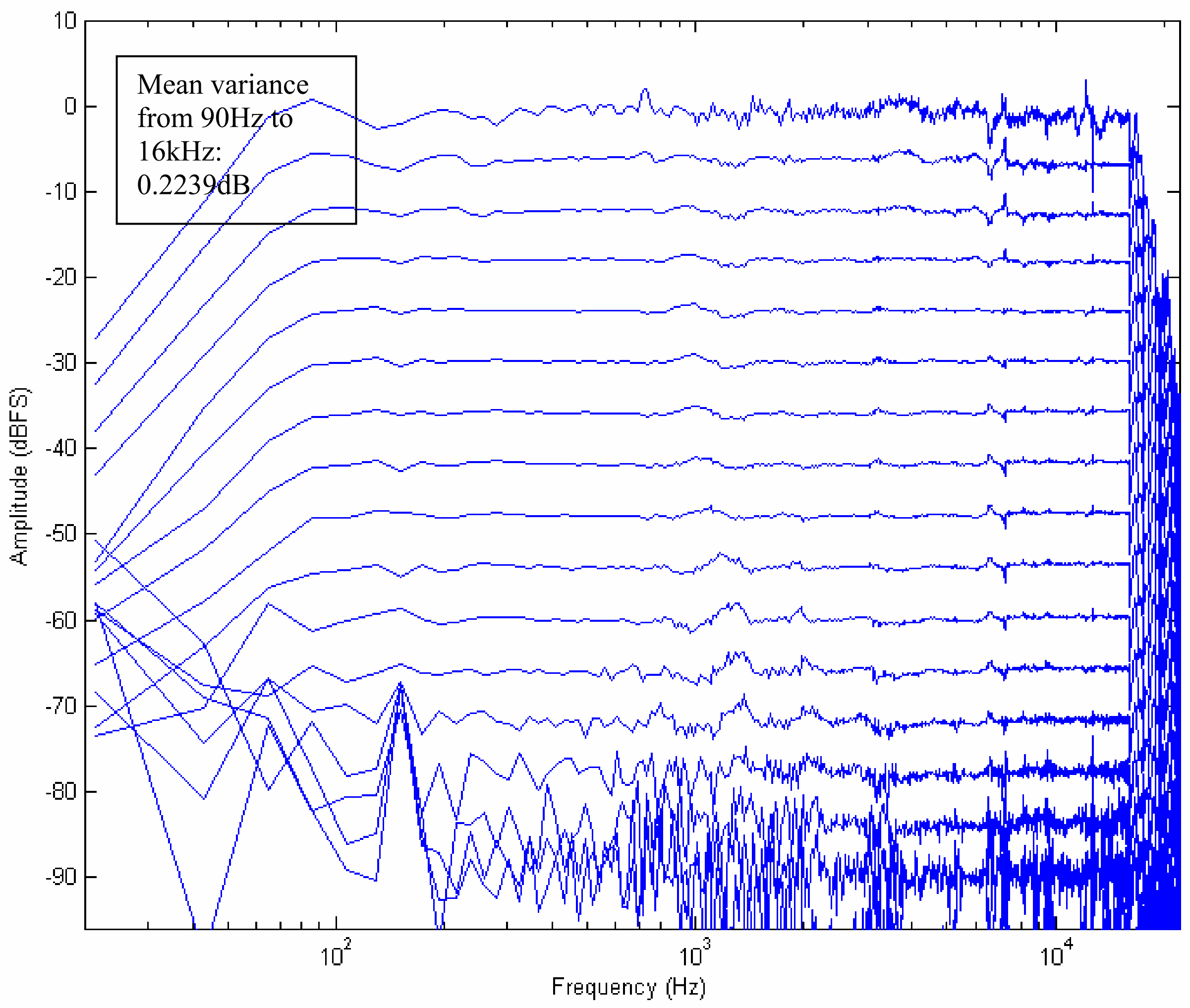


Figure 7.2: Frequency response after correction with linear equaliser. The equaliser was defined to work between frequencies 90Hz and 16kHz from -78dBFS to 0dBFS.

7.3 Level-dependent equaliser

The advantage of level-dependent equalisation over linear equalisation is apparent: using the same parameters as the linear equaliser (90Hz to 16kHz, -78dBFS to 0dBFS) the frequency response has been almost completely flattened in the range of interest, with a variance of 0.019dB.

Section 5.4 described how large equaliser gains at a particular frequency can cause the operating region of the loudspeaker to change and therefore reducing the effectiveness of the equaliser. Small perturbations in the level-dependent equalised response at around 100Hz (where nonlinearity is most prevalent) are likely to be due to this effect, though they are very small. Other variations are due to measurement errors, time-variance of the loudspeaker and system noise.

The results presented here may not necessarily be as good with speech or music. The measurement procedure defined a set of equalisers at specific levels and the test procedure used these same levels,

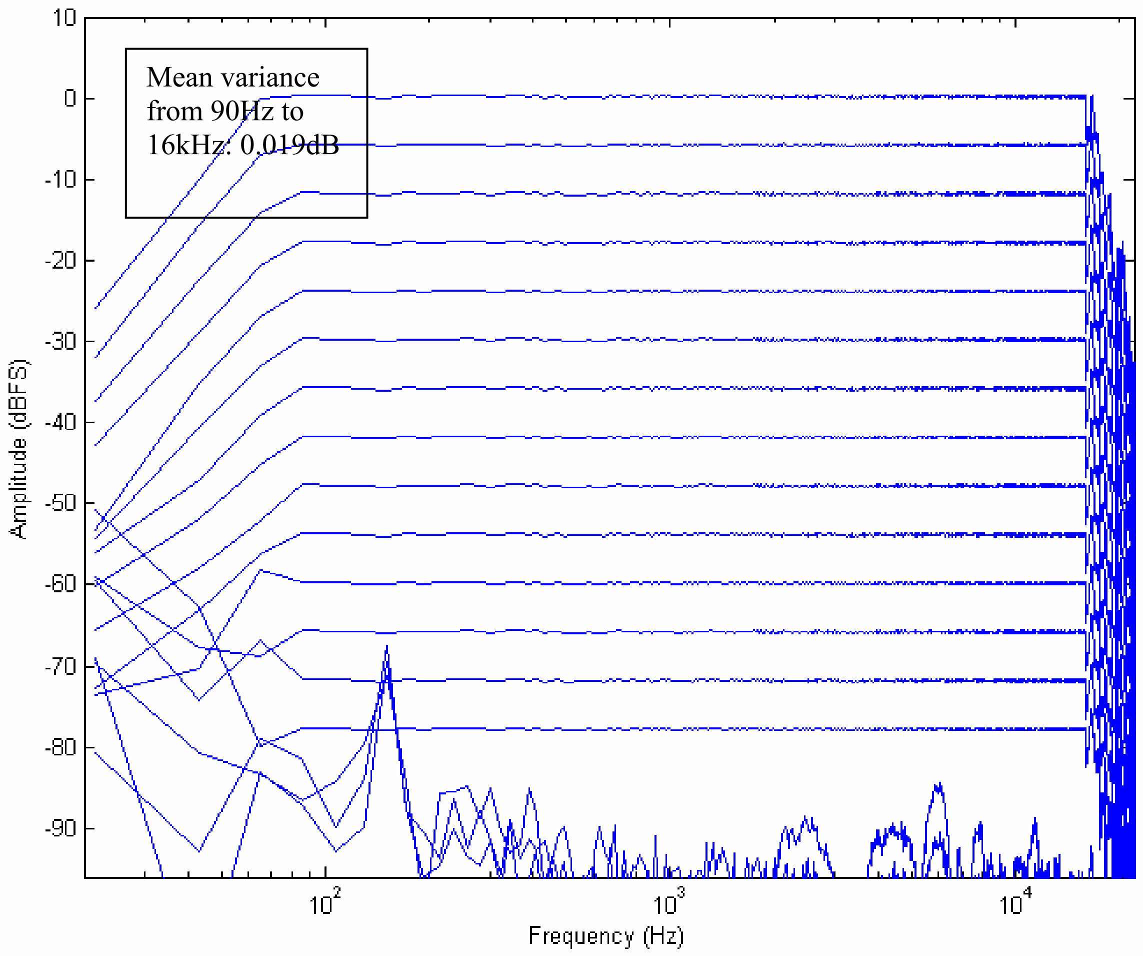


Figure 7.3: Frequency response after correction with a level-dependent equaliser. The equaliser was designed to be valid from 90Hz to 16kHz at amplitudes -78dBFS to 0dBFS.

giving the best possible results. But speech and music will occur at amplitudes in between those measured, giving a maximum error when the amplitude of a particular bin lies mid-way between two measured levels. This may be improved either by taking measurements at closer amplitude intervals, or interpolating measured values instead of switching between them as is done presently.

7.4 THD

THD measurements were made to provide further evidence to the loudspeaker's operating region. The inaccuracy of the linear equaliser in section 7.2 at 0dBFS agrees with the THD plot, which shows a significant increase in harmonic distortion at this amplitude caused by nonlinearity.

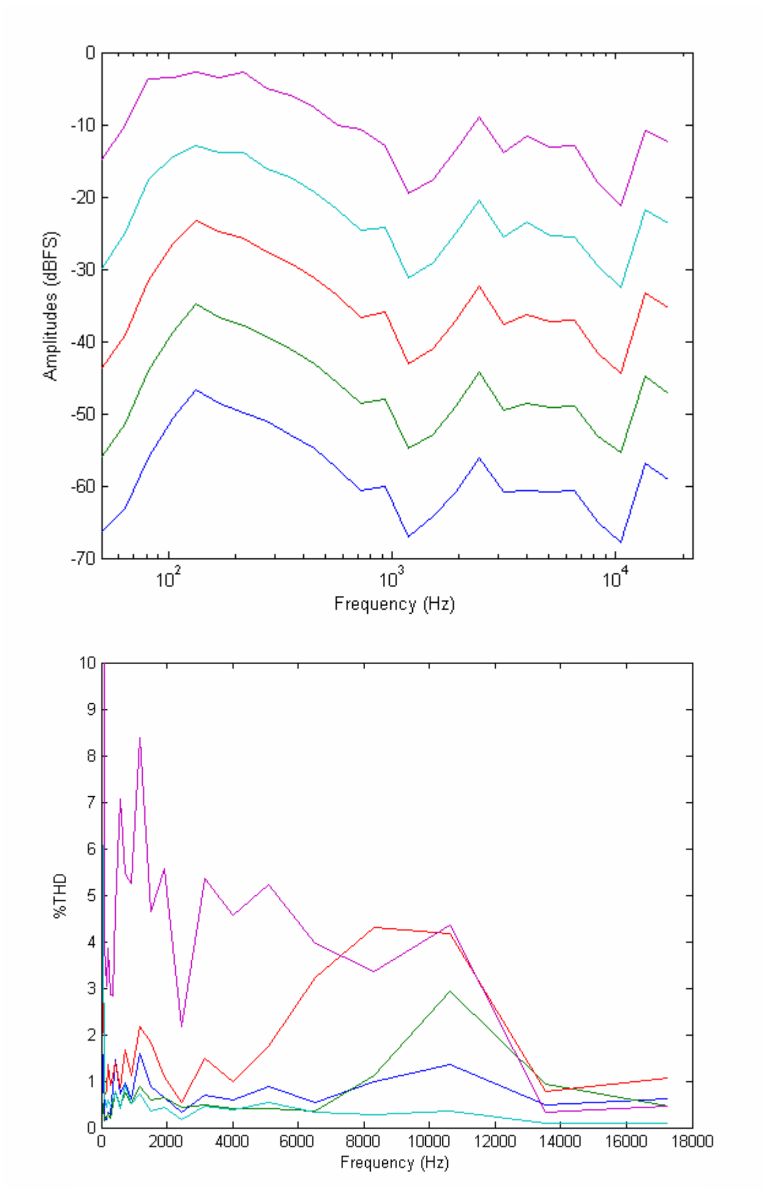


Figure 7.4: Sine excitation amplitude and the corresponding %THD measurements. Driving amplitudes were of 0dBFS to -48dBFS in 12dB steps. The THD is significantly worse at 0dBFS when the loudspeaker is driven into nonlinearity and harmonics are produced at greater levels.

7.5 Volterra Modelling

Can the results obtained with MLS analysis be modelled by a Volterra Series as described in section 4.3? The same loudspeaker with identical calibration was excited with 64 repetitions of a Gaussian noise burst consisting of 1024 samples (23ms) at 0dBFS (~100dBA at 1m). The recordings were run through Tick's Algorithm (section 4.3.2) and the results are presented in *Figure 7.5*. The Matlab implementation of the algorithm was obtained from [26] and was completely unmodified.

The linear transfer function resembles the transfer function from the MLS analysis shown in *Figure 7.1*. Without further analysis the quadratic transfer function is meaningless, so the derived nonlinear system was convolved with impulses according to the nonlinear convolution principle presented in equation (4.7). A Fourier transform was applied and the frequency-domain results are shown in *Figure 7.6*.

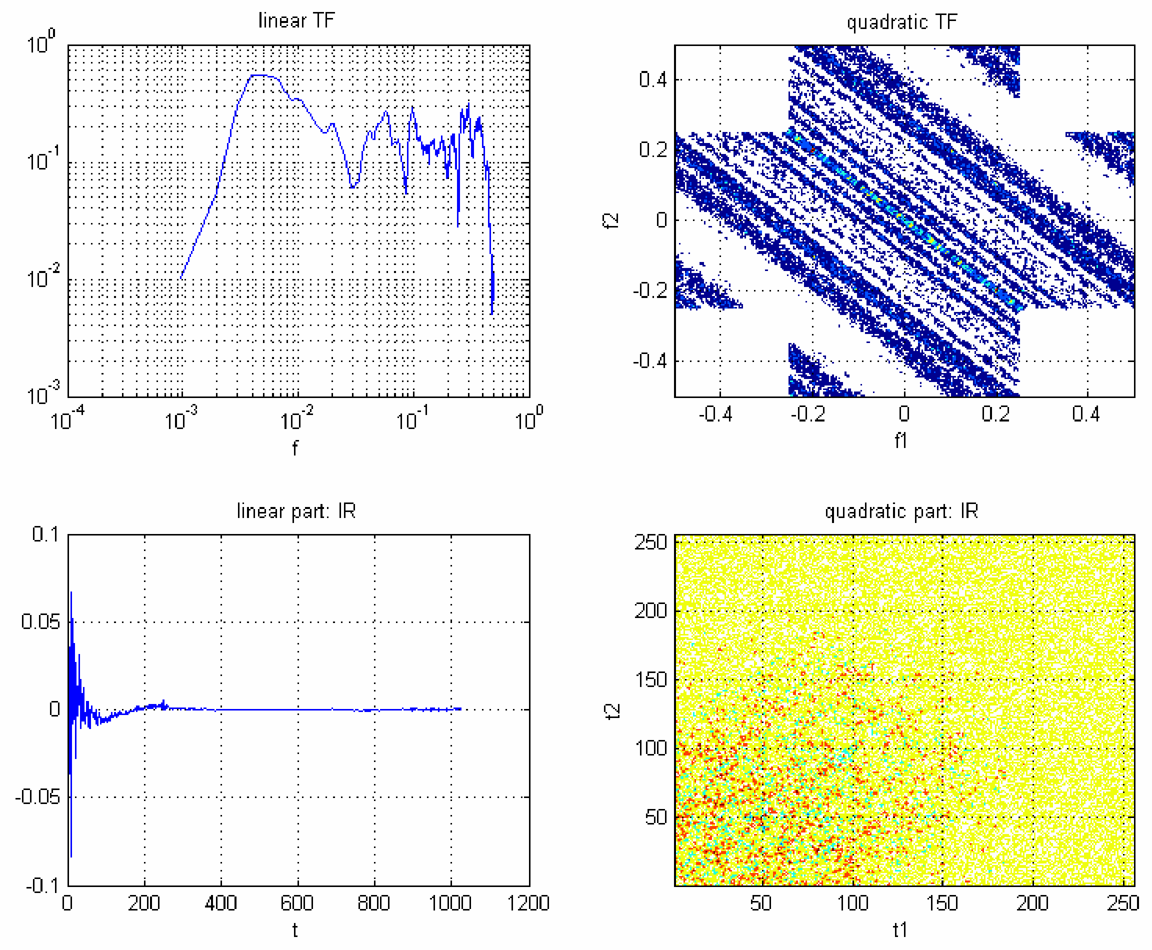


Figure 7.5: Linear and quadratic transfer functions derived with Tick's Algorithm. The loudspeaker was excited with 64 1024-sample bursts of Gaussian noise at 0dBFS (~100dBA at 1m). The linear part shows is very similar to the results of the MLS analysis as shown in Figure 7.1.

7.5.1 Frequency-domain description of the Volterra Model

The Volterra model was stimulated with impulses at levels -96dBFS to 0dBFS in 12dB increments and the results are presented in *Figure 7.6*. The overall linear shape is correct and some nonlinearities such as the shape of the peak at about 3kHz have been reproduced with some degree of accuracy, but the overall gain of the system is highly inaccurate. Levels at -60dBFS and below are too low in amplitude

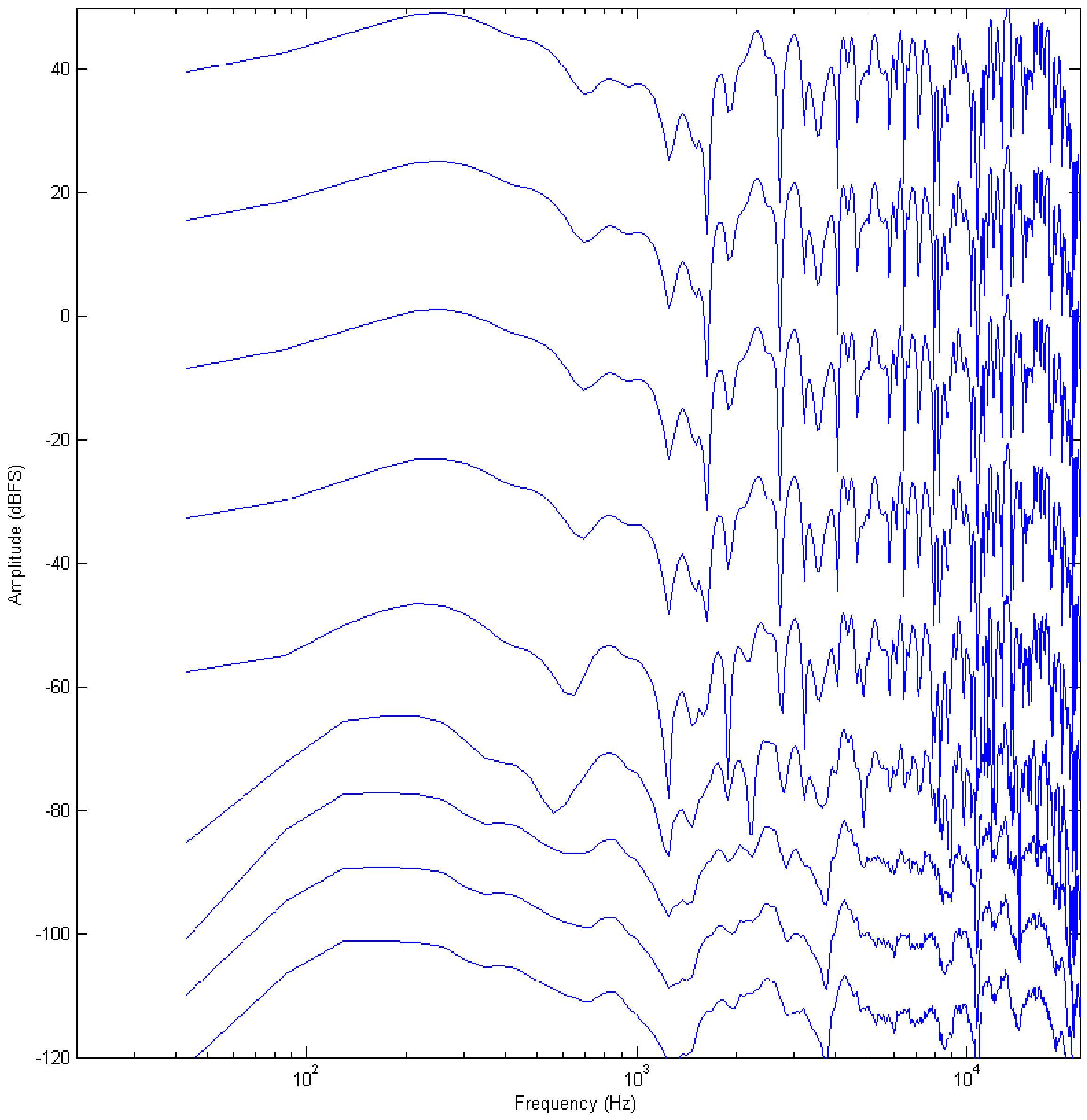


Figure 7.6: Second-order Volterra Model of the loudspeaker's transfer function in the frequency domain. The curves are the response to impulsive stimulation at levels -96dBFS to 0dBFS. Although the linear shape is roughly correct and some nonlinear characteristics have been faithfully reproduced, the overall gain of the system is completely wrong. This is likely to be due to an incorrect implementation of Tick's algorithm.

and those at -48dBFS and above are too high, suggesting that the quadratic part has been overestimated in the same way as in the test case in Section 4.3.2. It is unknown whether this is due to a flaw in the algorithm or the implementation as insufficient time was available for in-depth investigation.
The level-dependent equaliser was capable of equalising small nonlinearities caused by effects localised in frequency and driving amplitude, such as cone break-up. A 2nd order Volterra Kernel would be unable to correct these nonlinearities, but they may be small enough to be neglibible.

7.6 Summary

We have seen how an example loudspeaker can be equalised with linear and level-dependent equalisers. An attempt was also made to fit the results to a 2nd order Volterra Series.

A linear equaliser provides good equalisation at most listening levels and frequencies, but a level-dependent equaliser is required to achieve a truly flat response for all types of excitation.

Tick's Algorithm is convergent with the loudspeaker and has provided 2nd order Volterra Kernels, though implementing this model in the frequency domain yielded suspicious results. Whether this is a flaw in the algorithm or the implementation used is unknown.

A significant problem encountered with Tick's algorithm is processor loading. A 1024-length signal with 64 realisations takes about 30 minutes to analyse on a 2.7GHz Athlon processor under Matlab. For a comparison, a 4096-length MLS sequence with 64 repeats takes approximately two seconds to analyse, so faster algorithms must be devised if this is to be a viable analysis tool given present technology.

8 Conclusions

In this project we have reviewed the problem of loudspeaker equalisation, current approaches and their downfalls and measurement techniques for determining a loudspeaker's frequency response. A detailed discussion of MLS analysis and issues pertaining to its implementation was given; MLS was a particularly useful tool for measurement of a loudspeaker's impulse response and will hopefully see greater use in future commercial systems. Causes of nonlinearity and their mathematical modelling were presented, with particular reference to the use of a second-order Volterra Series. Finally we have seen results of practical experiments with MLS measurement, THD measurements, linear equalisers, level-dependent equalisers and Volterra approximation.

Theoretical modelling and practical measurements provide strong evidence that loudspeakers are nonlinear devices, and that nonlinearity is mainly due to voice-coil excursion. Low frequencies are more heavily affected as voice coil excursion is greater in low-frequency drive units. Through testing a system at a variety of amplitudes, an accurate nonlinear model of a loudspeaker can be determined in a relatively short period of time. An attempt to fit results to a 2nd order Volterra Series has shown promising results, though work needs to be done to improve the accuracy of the extracted kernels and the speed of execution.

Three types of equaliser were investigated: linear, level-dependent and Volterra. Linear equalisers are simple to implement and can be determined by averaging the frequency response of a loudspeaker at a variety of levels, then finding an equaliser to do the inverse. Level-dependent equalisers can equalise frequency responses at any number of different levels but are not a standard task for a DSP processor. Volterra equalisers model the impulse response of a system as an n^{th}-order polynomial and provide the most mathematically elegant solution. However, the determination of Volterra Kernels is highly computationally demanding, as is real-time equalisation.

State-of-the-art systems such as those by Genelec and JBL are not capable of achieving a frequency response as flat as the linear system presented in this project. Creating a stand-alone system capable of conducting an MLS analysis, determining a *nonlinear* equaliser and implementing it in real-time could be the next big turning point in loudspeaker design.

A useful tool developed for this project was a Matlab GUI, capable of conducting MLS and swept sine analysis, using the information to create linear and level-dependent equalisers. An equaliser can be designed and tested on real audio within a few minutes, and data can be easily imported and exported for processing and analysis in Matlab. This tool may be useful for future measurements on any audio system.

8.1 Future Work

Extend MLS theory to efficiently estimate Volterra kernel.

Determine a method for allowing the use of a level-dependent equaliser while maintaining linear phase.

Investigate frequency responses on- and off-axis and determine equalisers which give faithful equalisation for any listening position.

Extend the equaliser to correct each drive-unit independently.

Very little time was dedicated to the subjective testing of results. It is expected that although frequency response can be flattened with a nonlinear equaliser better than with a linear equaliser, the perceived improvement will be negligible.

Recent development in MEMS (Micro-Elecro-Mechanical Decvices) accelerometers could be applied to flattening a loudspeaker's frequency response without the need for detailed testing. Section 2.2 describes how a feedback mechanism based on transducer mounted on a loudspeaker cone can be used to develop a classic feedback control system. Early attempts did not yield good results because of the quality of the transducers, but recent MEMS developments may provide a feasible feedback mechanism.

Develop a level-dependent system which accounts for *perceived loudness*. *Figure 8.1* shows the ISO-standard and Fletcher-Munson contours, which are empirical models describing how the ear perceives sound at different levels. It is clear from the diagram why audio sounds bass-light when the volume is low. Take for example a classical concert: the composer wrote the music to be listened to live at a particular volume, but in a home listening environment, the music will almost certainly be played at a lower level. Knowing the sound pressure level at which the music was played and the sound pressure level in the listening environment, a correction may be applied so that the relative perceived loudness between each frequency is constant regardless of listening volume. There are a number of issues to be resolved: how often should the correction factor be updated? Are the contours accurate enough? How is the listening SPL determined in a room with multiple seats? This would make a good MEng project.

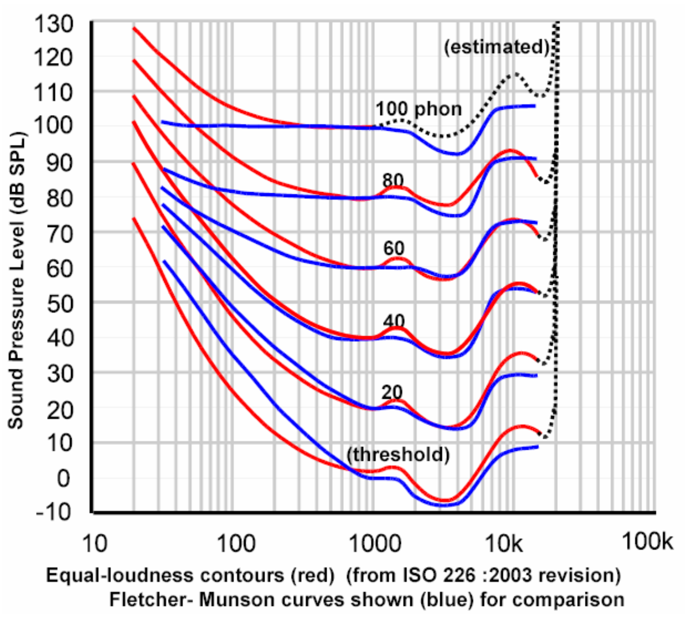


Figure 8.1: Equal loudness curves

9 Bibliography

[1] J. M. Berman & L. R. Fincham, “The Application of Digital Techniques to the Measurement of Loudspeakers,” *J. Audio Eng. Soc.*, vol. 25, pp. 370-384 (1977)

[2] J. Borish & J. B. Angell, “An Efficient Algorithm for Measuring the Impulse Response Using Pseudrandom Noise,” *J. Audio Eng. Soc.*, vol. 31, pp. 478-489 (1983)

[3] D. D. Rife & J. Vanderkooy, “Transfer-Function Measurement with Maximum-Length Sequences,” *J. Audio Eng. Soc.*, vol. 37, pp. 419-444 (1989)

[4] L. R. Fincham, “Refinements in the Impulse Testing of Loudspeakers,” *J. Audio Eng. Soc.*, vol. 33, pp. 133-140 (1985)

[5] C. Bleakley & R. Scaife, “New formulas for predicting the accuracy of acoustical measurements made in noisy environments using the averaged *m*-sequence correlation technique,” *J. Acoust. Soc. Am*. 97, pp. 1329-1332 (1995)

[6] W. T. Chu, “Impulse Response and Reverbration-Decay Measurements Made by Using a Periodic Pseudorandom Sequence,” *Applied Acoustics*, vol. 29, pp. 193-205 (1990)

[7] M. Cohn & A. Lempel, “On Fast M-Sequence Transforms,” *IEEE Trans. Inf. Theory*, IT-23 pp. 135-137 (1977)

[8] R. K. R. Yarlagadda & J. E. Hershey, “Hadamard Matrix Analysis and Synthesis With Applications to Communications and Signal/Image Processing,” *Kluwer Academic Publishers (Boston, MA)* (1997)

[9] J. Sylvester, "Thoughts on Inverse Orthogonal Matrices, simultaneous Sign-successions, and Tessellated Pavements in two or more colours, with applications to Newton's Rule, Ornamental Tile-work, and the Theory of Numbers," *Phil Mag,* Series 4, No 34, pp.461-471 (1967)

[10] W. K. Pratt, J. Kane & H. C. Andrews, "Hadamard Transform Image Coding," *Proc. IEEE*, Vol 57, No 1, p. 58-66 (1969)

[11] A. J. M. Kaizer, “Modeling of the Nonlinear Response of an Electrodynamic Loudspeaker by a Volterra Series Expansion”, *J. Audio Eng. Soc.*, vol 35, pp. 421-432 (1987)

[12] W. Klippel, “Dynamic Measurement and Interpretation of the Nonlinear Parameters of Electrodynamic Loudspeakers,” *J. Audio Eng. Soc.*, vol. 38, pp. 944-955 (1990)

[13] D. R. Birt, “Nonlinearities in Moving-Coil Loudspeakers with Overhung Voice Coils,” *J. Audio Eng. Soc.*, vol. 39, pp. 219-231 (1991)

[14] W. Klippel, “Nonlinear Large-Signal Behaviour of Electrodynamic Loudspeakers at Low Frequencies,” *J. Audio Eng. Soc.*, vol. 40, pp. 483-496 (1992)

[15] A. Dobrucki, “Nontypical Effects in an Electrodynamic Loudspeaker with a Non homogeneous Magnetic Field in the Air Gap and Nonlinear Suspensions,” *J. Audio Eng. Soc.*, vol. 42, pp. 565-576 (1994)

[16] A. Voishvillo, A. Terekhov, E. Czerwinski, S. Alexandrov, “Graphing, Interpretation, and Comparison of Results of Loudspeaker Nonlinear Distortion Measurements,” *J. Audio Eng. Soc.*, vol. 52, pp. 332-357 (2004)

[17] B. Duncan, “Exploring Equalisers,” *Studio Sound*, February 1992 pp. 32-84

[18] R. H. Small, “Closed-Box Loudspeaker Systems-Part 1: Analysis,” *J. Audio Eng. Soc.*, vol. 20, pp. 798-808 (1972)

[19] R. H. Small, “Closed-Box Loudspeaker Systems-Part 2: Synthesis,” *J. Audio Eng. Soc.*, vol. 21, pp. 11-18 (1973)

[20] R. H. Small, “Loudspeaker Standards and Loudspeaker Performance Limitations,” *J. Audio Eng. Soc.*, vol. 21, pp. 117-119 (1973)

[21] H. D. Harwood, M. E. Whatton, R. W. Mills, “The design of the miniature monitoring loudspeaker type LS3/5A,” *BBC Research Department Report 1976/29*

[22] Powers, E.J., Ch.P. Ritz, C.K. An, S.B. Kim, R.W. Miksad and S.W. Nam,
“Applications of digital polyspectral analysis to nonlinear systems modeling
and nonlinear wave phenomena,” *Proc. Workshop on Higher-Order Spectral
Analysis*, pp. 73-77, Vail, Colorado, June 1989.

[23] Tick, L.J., “The estimation of transfer functions of quadratic systems,” *Technometrics*, pp. 563-67, Nov 1961.

[24] A. J. Redfern & G. T. Zhon, “Performance Analysis of Volterra Kernel Estimators with Gaussian Inputs,” *Proc. IEEE Signal Processing Workshop*, pp. 162-165 (1997)

[25] U. Nuding, C. Zetzsche, K. Schill, G. Hauske, “Measurement of nonlinear 2nd-order kernels using Gaussian and natural inputs,” *IEEE Conf. Signal, Systems & Computers*, Vol. 1, pp. 760-764 (2004)

[26] A. Swami, “HOSA - Higher Order Spectral Analysis Toolbox,” *Matlab Central File Exchange, www.mathworks.com/matlabcentral/fileexchange*

Appendix 1 - User Guide: Command-line tools

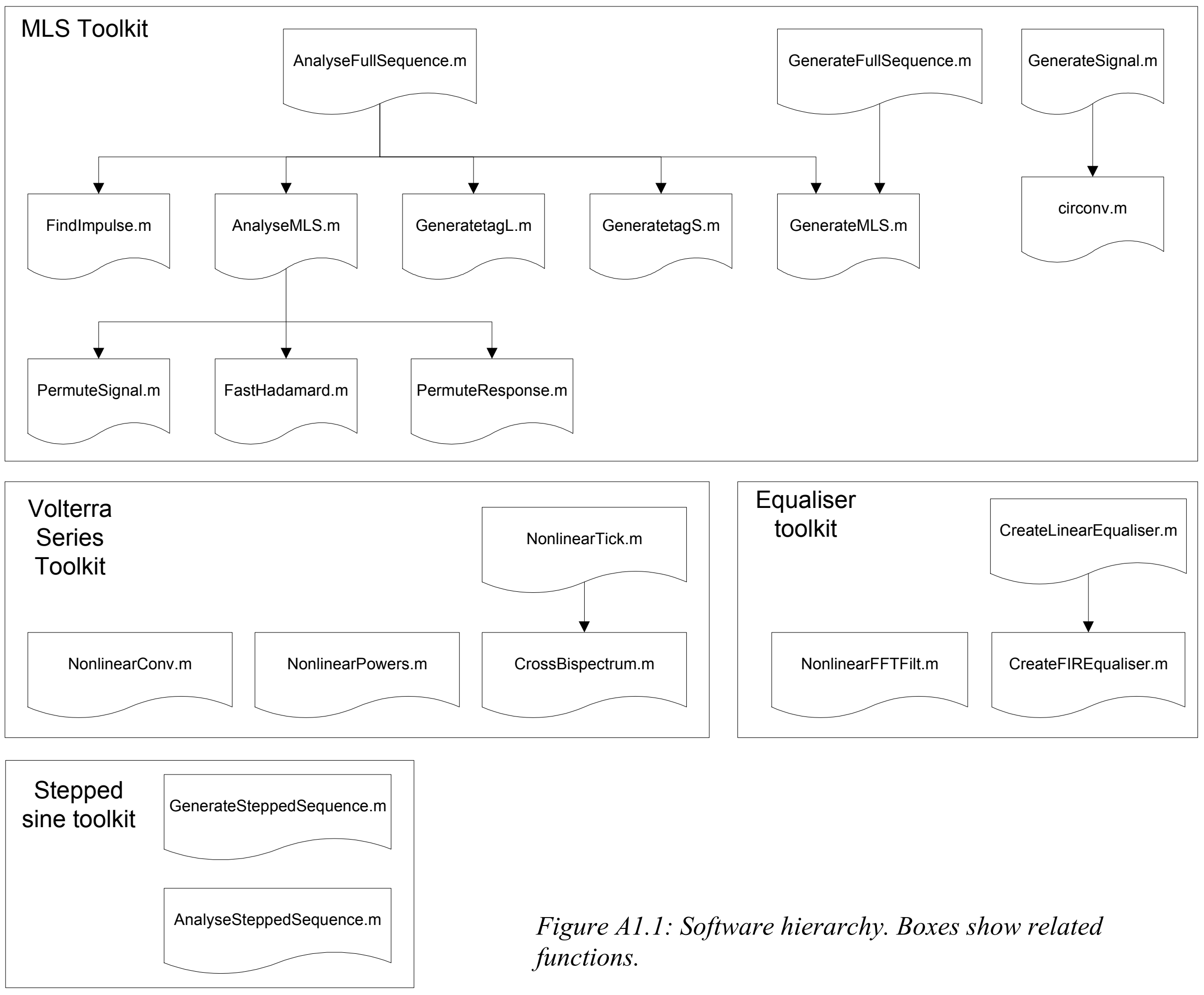


Figure A1.1: Software hierarchy. Boxes show related functions.

A1.1 MLS Toolkit

Name AnalyseFullSequence

Purpose Detects alignment impulse and analyses multi-level MLS bursts

Syntax [impulses] = AnalyseFullSequence(signal, offset, burstlevels, repetitions, N, DCCoupling)

Description

signal:	Recorded sequence from system under test
offset:	The impulse detection finds the point where the signal exceeds a certain rate of change. Pulse spreading may cause the detection to be erroneous by a few samples (rarely more than 10) and this may be corrected with the offset parameter. Positive offset causes negative time shift.
burstlevels:	1xM vector of burst amplitudes (can be linear or dBFS).
repetitions:	The number of repetitions of each amplitude (cannot be less than 2).
N:	Order of the MLS, where P=2^N-1

	DCCoupling:	Set to true for DC recovery method. Set to false for loudspeaker measurements.
	impulses:	An *M*x*N* vector of impulse responses, where *M* is the number of amplitudes and *N* is the order of the MLS.

Algorithm A signal is assumed to be of the form:

```
Positive impulse;
Pause $2^N$ samples;
for i=1 to length(burstlevels)
    for j=1 to repetitions
        Play MLS at level burstlevels(i);
    end
    Pause $2^N$ samples;
end
```

The second and subsequence repeats are averaged for each amplitude and analysed, yielding impulse responses for each amplitude.

Reference Chapter 3

Name GenerateFullSequence

Purpose Generates an impulse followed by bursts of MLS at different levels.

Syntax [sequence] = GenerateFullSequence(burstlevels, leveltype, repetitions, N)

Description	burstlevels:	A 1xM array of amplitudes in dBFS or normalised linear.
	leveltype:	Can be ‘dbfs’ or ‘lin’.
	repetitions:	Number of repetitions of each amplitude.
	N:	MLS order, where $P=2^N-1$.
	sequence:	The generated sequence.

Algorithm Sequence is of the form:

```
Positive impulse;
Pause $2^N$ samples;
for i=1 to length(burstlevels)
    for j=1 to repetitions
        Play MLS at level burstlevels(i);
    end
    Pause $2^N$ samples;
end
```

Reference Chapter 3

Name	FindImpulse	
Purpose	Detects a positive impulse	
Syntax	[impulseIndex] = FindImpulse(sample)	
Description	sample:	The sample to be analysed.
	impulseIndex:	The index of the detected impulse
Algorithm	Differentiate sample. If rate of change exceeds 0.02, index is returned.	
Reference	Chapter 3	

Name	AnalyseMLS	
Purpose	Returns impulse response from a given MLS signal	
Syntax	[impulse] = AnalyseMLS(signal,mls,tagS,tagL,N,DCCoupling)	
Description	signal:	Time-aligned received MLS signal (excluding 1st repetition).
	mls:	The original MLS signal.
	tagS:	Array for reordering of input samples (from GeneratetagS).
	tagL:	Array for reordering of output samples (from GeneratetagL).
	N:	Order of MLS, where P=2^N-1.
	DCCoupling:	Set to true for DC recovery method. Set to false for loudspeaker measurements.
	impulse:	Recovered impulse response
Algorithm	Rearrange according to tagS;	
	Apply Fast Hadamard Transform;	
	Rearrange according to tagL;	
Reference	Chapter 3	

Name	GeneratetagL	
Purpose	Generates array for the rearrangement of samples after a Hadamard Transform	
Syntax	tagL = GeneratetagL(mls, P, N);	
Description	mls:	The MLS signal for which tagL is valid.
	P:	Length of the MLS signal
	N:	Order of the MLS signal
	tagL:	1x(P+1) vector of indices.
Algorithm	Find which values of the tagS vector are a power of 2;	
	Generate **L** matrix by shifting MLS sequence right by amounts above (is Nx(P+1))	
	Sum columns as increasing powers of 2;	
Reference	Chapter 3	

Name	GeneratetagS	
Purpose	Generates array for the rearrangement of samples before a Hadamard Transform	
Syntax	tagS = GeneratetagS(mls, P, N);	
Description	mls:	The MLS signal for which tagS is valid.
	P:	Length of the MLS signal
	N:	Order of the MLS signal
	tagS:	1x(*P*+1) vector of indices.
Algorithm	Generate **S** matrix by shifting MLS right successively *N* times.	
Reference	Chapter 3	

Name	GenerateMLS	
Purpose	Generates an MLS sequence	
Syntax	y = mls(n, flag)	
Description	n:	order of MLS
	flag:	true for registers initialised to 1, false for random
	y:	*P*-length MLS sequence, where $P=2^N-1$
Algorithm	$n(k+3) = n(k) \oplus n(k+2)$	
Reference	Chapter x	

Name	PermuteSignal	
Purpose	Rearranges input signal according to tagS.	
Syntax	perm = PermuteSignal(signal, tagS, P, dcCoupled);	
Description	signal:	Signal to be arranged
	tagS:	1x*P* vector of indces from GeneratetagS
	P:	Length of MLS, where $P=2^N-1$
	DCCoupled:	Set to true for DC recovery method. Set to false for loudspeaker measurements.
	perm:	The rearranged signal
Algorithm	See below	
Reference	Chapter 3	

Algorithm

```
If DC coupled = true
        Set first element to sum of all other elements
else
        Set first element to zero
Rearrange signal according to tagS.
```

Name	PermuteResponse
Purpose	Rearranges output of Hadamard Transform according to tagL.
Syntax	resp = PermuteResponse(perm, tagL, P)
Description	perm: Output of Hadamard Transform
	tagL: 1xP vector of indices from GeneratetagL.
	P: Length of MLS, where P=2^N-1
	Resp: Rearranged signal
Algorithm	Rearrange signal according to tagL.
Reference	Chapter 3

Name	FastHadamard
Purpose	Applies a Fast Hadamard transform to a 1-D signal
Syntax	y = FastHadamard(x, P, N)
Description	x: Signal to be transformed
	P: Length of MLS, where P=2^N-1
	N: Order of MLS
	y: Transformed signal
Algorithm	Construct N+1 columns, each with 2^N rows.
	Place the input vector $x[n]$ in the 1st column.
	Moving left to right, fill in the next column as follows:
	In the top half of this column, place the pairwise, mutually exclusive sums of the previous column.
	In the bottom half, place the pairwise differences.
	Repeat (Step 3) for each of the remaining columns.
Reference	Chapter 3

Name	GenerateSignal
Purpose	Generates a signal by circularly convolving a filter with coefficients b with an MLS signal. Used for test purposes.
Syntax	signal = GenerateSignal(mls, b, P);
Description	mls: MLS signal
	b: FIR coefficients of filter
	P: Length of MLS, where P=2^N-1
Algorithm	$\text{signal} = \text{mls} \otimes \text{b}$
Reference	Chapter 3

Name circonv

Purpose Circular convolution

Syntax [C] = circonv(A,B,N)

Description

A:	First signal to circularly convolve
B:	Second signal to circularly convolve
N:	Length of convolution
C:	Circularly convolved output

Algorithm $C = A \otimes B$

Reference Chapter 3

A1.2 Equaliser Toolkit

Name CreateLinearEqualiser

Purpose Creates a linear equaliser from a level-dependent equaliser

Syntax [linearEqualiser] = CreateLinearEqualiser(equaliser)

Description

equaliser:	A level-dependent equaliser from CreateFIREqualiser
linearEqualiser:	A mean-average equaliser dependent on frequency only

Algorithm linearEqualiser = mean(equaliser);

Reference Chapter 5

Name CreateFIREqualiser

Purpose Creates a best-fit FIR level-dependent equaliser

Syntax equaliser = CreateFIREqualiser(burstlevels, impulses, leveltype, lowerfbound, upperfbound, attenuate, fs);

Description

burstlevels:	Array of burst amplitudes (dBFS or normalised linear)
impulses:	MLS-derived impulse responses (MxN)
leveltype:	'dbfs' or 'lin'
lowerfbound:	Lower frequency boundary
upperfbound:	Upper frequency boundary
attenuate:	True to create attenuate-only equalisers, false otherwise
fs:	Sampling frequency (usually 44100Hz)

Algorithm

```
freqresps = fft(impulses)
for i=1:1:M
        freqresps(i,:) = freqresps(i,:)-burstlevelsdBFS(i);
        equaliser(i,1:n) = -mean(freqresps(i,lowerboundind:upperboundind));
        equaliser(i,lowerboundind:upperboundind) = -
        freqresps(i,lowerboundind:upperboundind);
        Find FIR filter to best fit equaliser;
```

	end	
	Reduce gain to attenuate if necessary	
	Export FIR equalisers (MxN)	
Reference	Chapter 5	

Name	NonlinearFFTFilt	
Purpose	Like Matlab FFTFILT, but applies level-dependent FFT multiplication	
Syntax	y = NonlinearFFTFilt(burstlevels,leveltype,equaliser,x)	
Description	burstlevels:	Array of burst amplitudes (dBFS or normalised linear)
	leveltype:	‘dbfs’ or ‘lin’
	equaliser:	MxN array of equaliser impulse responses
	x:	Signal to be equalised
	y:	Equalised signal
Algorithm	Take FFT of input signal;	
	Take FFT of equalisers;	
	Overlap-add:	
	For each FFT bin, determine the level and frequency and apply correction factor from the closest equaliser;	
	end;	
Reference	Chapter 5	

A1.3 Volterra Series Toolkit

Name	NonlinearTick	
Purpose	Implements Tick’s algorithm to determine 2nd Order Volterra Kernels	
Syntax	[h,q] = nltick(x,y,nfft,wind,segsamp,overlap)	
Description	x:	Input signal to system
	y:	Output signal from system
	nfft:	Length of FFT to use in analysis of spectra / bispectra
	wind:	Window specification for frequency-domain smoothing If 'wind' is a scalar, it specifies the length of the side of the square for the Rao-Gabr optimal window [default=5] If 'wind' is a vector, a 2D window will be calculated via w2(i,j) = wind(i) * wind(j) * wind(i+j) If 'wind' is a matrix, it specifies the 2-D filter direct
	segsamp:	Samples per segment (default: so as to have 8 records)
	overlap:	Percentage overlap, allowed range [0,99]. (Default = 5)
	h:	Linear Volterra Kernel

	q:	2nd order Volterra Kernel
Algorithm	See chapter x	
Reference	Chapter 4	

Name	NonlinearPowers	
Purpose	Implements Powers's algorithm for determining 2nd order Volterra Kernels	
Syntax	[h,q] = nlpow(x,y,nfft)	
Description	x:	Input signal to system
	y:	Output signal from system
	nfft:	Length of FFT to use in analysis
	h:	Linear Volterra Kernel
	q:	2nd order Volterra Kernel
Algorithm	See chapter 4	
Reference	Chapter 4	

Name	NonlinearConv2ndOrder	
Purpose	Implements 2nd order Volterra Convolution	
Syntax	y = NonlinearConv2ndOrder(x, h, q)	
Description	x:	Input signal to system
	h:	Linear Volterra Kernel
	q:	2nd order Volterra Kernel
	y:	Convolved sequence
Algorithm	See chapter 4	
Reference	Chapter 4	

A1.4 Stepped Sine Toolkit

Name	GenerateSteppedSequence
Purpose	Generates a sequence of stepped sines at multiple amplitudes, impulse-aligned
Syntax	sequence = GenerateSteppedSequence(burstlevels, leveltype, lowerFreq, n, steptime,fs)

Description

burstlevels:	Array of burst amplitudes (dBFS or normalised linear)
leveltype:	'dbfs' or 'lin'
lowerFreq:	Starting frequency (Hz)
n:	Number of levels
steptime:	Time for each step (ms)
fs:	Sampling frequency (usually 44100Hz)
sequence:	The generated sequence

Algorithm

```
r = 10^(log10((fs/2)/lowerFreq)/n);
Generate impulse;
Pause of steptime;
for i=1:1:n
        Generate frequency at (lowerFreq*r^(i-1))
end
```

Reference Chapter 2

Name	AnalyseSteppedSequence
Purpose	Analyses stepped sequence from GenerateSteppedSequence
Syntax	[freqs levels thd] = AnalyseSteppedSequence(signal, offset, burstlevels, lowerFreq, n, steptime,fs)

Description

signal:	Signal from system under test
offset:	The impulse detection finds the point where the signal exceeds a certain rate of change. Pulse spreading may cause the detection to be erroneous by a few samples (rarely more than 10) and this may be corrected with the offset parameter. Positive offset causes negative time shift.
burstlevels:	Array of burst amplitudes (dBFS or normalised linear)
lowerFreq:	Starting frequency
n:	Number of steps
steptime:	Time for each step (ms)

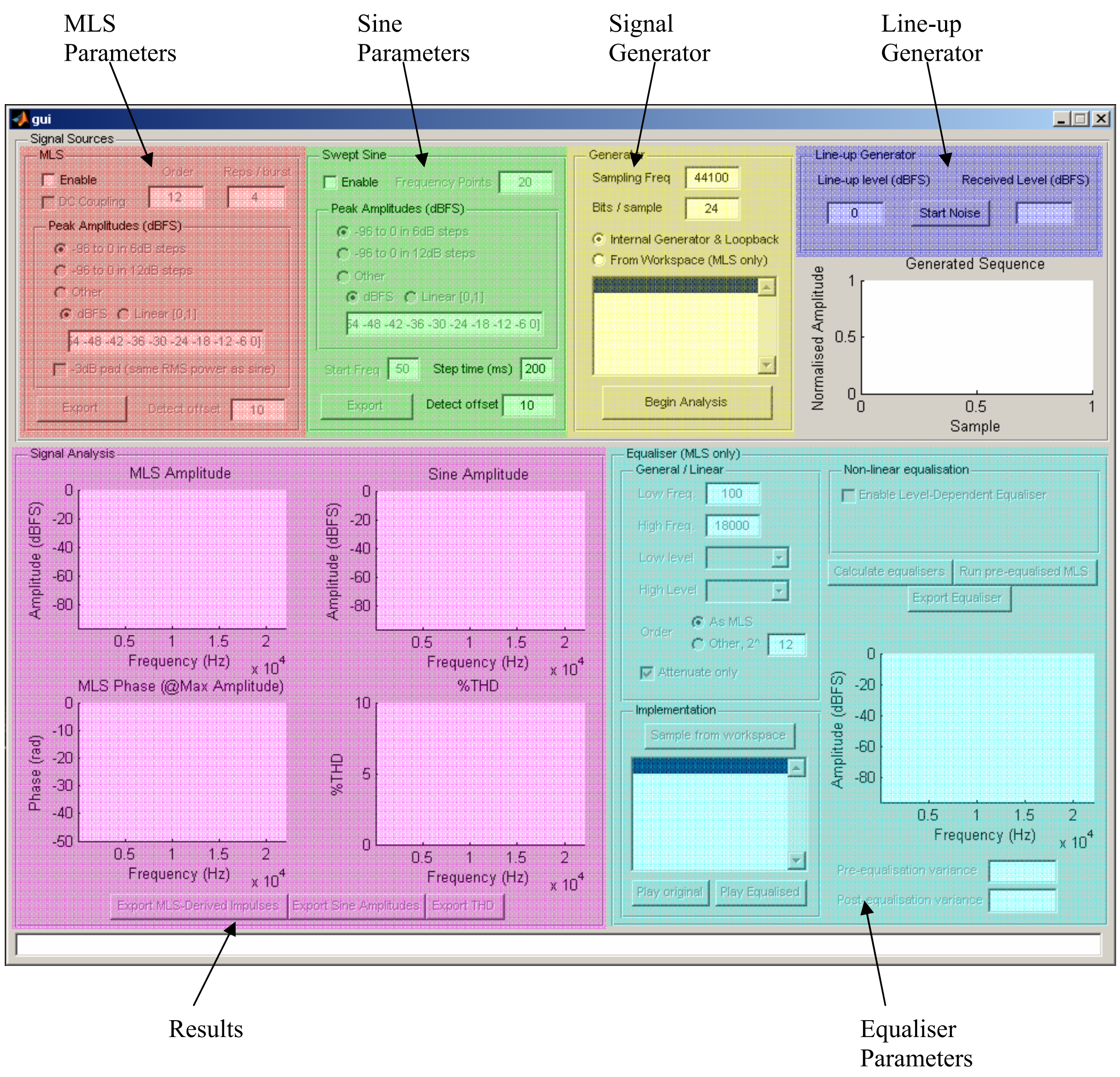
	fs:	Sampling frequency (usually 44100Hz)
	freqs:	Array of frequencies for each step
	levels:	Normalised received levels for each step
	thd:	Total harmonic distortion (normalised)

Algorithm

```
Detect impulse;
for i=1:1:n
        Take ¼ to ¾ of the step extent and determine RMS level;
        Take Hamming window and FFT;
        Find RMS energy;
        Notch out fundamental and find remaining energy;
        Find ratio of harmonic and fundamental energy for THD
end
```

Reference Chapter 2

Appendix 2 - User Guide: GUI



MLS: Order 12, Reps / burst 4; Peak Amplitudes (dBFS): [✓ -96 to 0 in 6dB steps] -96 to 0 in 12dB steps, Other, [✓ dBFS] Linear [0,1], [4 -48 -42 -36 -30 -24 -18 -12 -6 0]; Detect offset 10

Swept Sine: Frequency Points 20; Peak Amplitudes (dBFS): [✓ -96 to 0 in 6dB steps] -96 to 0 in 12dB steps, Other, [✓ dBFS] Linear [0,1], 54 -48 -42 -36 -30 -24 -18 -12 -6 0]; Start Freq 50; Step time (ms) 200; Detect offset 10

General: Sampling Freq 44100; Bits / sample 24; [✓ Internal Generator & Loopback] From Workspace (MLS only)

Line-up Generator: Line-up level (dBFS) 0; Received Level (dBFS)

Equaliser (MLS only): Low Freq. 100; High Freq. 18000; Order [✓ As MLS] Other, 2^ 12; [✓ Attenuate only]

Figure A2.1: GUI

The GUI is designed to incorporate all the functionality of the command-line toolkits into a convenient graphical interface. Within minutes it is possible to generate MLS sequences, analyse them and generate and test equalisers. Although stepped sine sequences are not used in the generation of equalisers, they can provide information about the system’s THD.

A2.1 MLS Parameters

Enable box:	Tick this to enable MLS anaylsis.
DC Coupling:	Enable if DC recovery is to be used in analysis.

Order:	The order *N* of the MLS, where length $P=2^N-1$
Reps/Burst:	The number of repetitions of the MLS signal for each amplitude. A doubling in the number of repetitions increases the SNR by 3dB.
Peak Amplitudes:	The amplitude levels for which the system will be tested. -96dBFS to 0dBFS in 12dB and 6dB steps are provided for convenience, but the user may give their own levels by hitting the 'other' radio button and writing an array of the form [x1 x2 x3] (Matlab syntax). The values can be in dBFS or normalised units (in the range [0,1]) by clicking the appropriate radio button.
-3dB Pad:	If excitation energy is of importance and comparisons are to be made between MLS and sine sequences, a 3dB pad may be added to reduce the RMS power of the excitation to that of a sine of equal peak amplitude.
Export:	The generated sequence may be exported to the Matlab workspace to implement external filters. A dialogue box appears when clicked for the user to specify a variable name.
Impulse Offset:	The impulse detection finds the point where the signal exceeds a certain rate of change. Pulse spreading may cause the detection to be erroneous by a few samples (rarely more than 10) and this may be corrected with the offset parameter. Positive offset causes negative time shift.

A2.2 Sine Parameters

Enable box:	Tick this to enable stepped sine anaylsis.
Frequency points:	The number of different frequencies with which to stimulate the system.
Peak Amplitudes:	The amplitude levels for which the system will be tested. -96dBFS to 0dBFS in 12dB and 6dB steps are provided for convenience, but the user may give their own levels by hitting the 'other' radio button and writing an array of the form [x1 x2 x3] (Matlab syntax). The values can be in dBFS or normalised units (in the range [0,1]) by clicking the appropriate radio button.
Start Freq:	The lowest frequency to generate. The program increases the frequency at logarithmic spacing up to the half the sampling frequency, the number of which is defined by 'Frequency Points'.
Step Time:	The number of milliseconds for each step in frequency.
Export:	The generated sequence may be exported to the Matlab workspace to implement external filters. A dialogue box appears when clicked for the user to specify a variable name.
Impulse Offset:	The impulse detection finds the point where the signal exceeds a certain rate of change. Pulse spreading may cause the detection to be erroneous by a few samples (rarely more than 10) and this may be corrected with the offset

parameter. Positive offset causes negative time shift.

A2.3 Signal Generator

Sampling Freq:	The system sampling frequency (default 44100Hz).
Bits/sample:	Sample resolution (default 24 bits).
Internal Generator & Loopback:	Test sequences are generated real-time and the output of the system under test is simultaneously recorded.
From Workspace (MLS ONLY):	Test sequences which have been exported using the export button in the MLS parameter box can be imported using this radio button. Select the Matlab workspace variable to be analysed from the list.
Begin analysis:	Starts analysing the system under test according to the setup parameters defined above.

A2.4 Line-up Generator

It is important that a 0dBFS input gives a 0dBFS output in order for the equalisers to be calculated correctly. Use the line-up generator to play a burst of noise and to display the level of received audio. Gains elsewhere in the system, such as the PC mixer output gain, power amplifier gain, microphone preamp gain and PC mixer input gain may need adjusting to get correspondence between these two values.

A2.5 Signal Analysis

A2.5.1 MLS Analysis

MLS Amplitude:	Displays a graph of the system amplitude as determined by the MLS analysis. Multiple levels are superimposed onto the same graph.
MLS Phase:	The phase of the highest-amplitude test sequence is displayed.
Export MLS-Derived Impulses:	Exports an MxN matrix of impulse responses (where M is the number of burst levels and N is the length of the MLS). A dialogue box is given for the user to specify the variable name to export to the Matlab workspace.

A2.5.2 Sine Analysis

Sine Amplitude:	Displays a graph of the system amplitude as determined by the MLS analysis. Multiple levels are superimposed onto the same graph.
THD:	Displays a graph of the system %THD determined by the sine analysis. Multiple levels are superimposed onto the same graph.

Export Sine Amplitudes: Exports an MxN matrix of amplitude responses (where M is the number of burst levels and N is the number of frequency points). A dialogue box is given for the user to specify the variable name to export to the Matlab workspace. A variable 'freqs' is automatically exported and contains an array of measurement frequencies.

Export THD: Exports an MxN matrix of normalised THD values (where M is the number of burst levels and N is the number of frequency points). A dialogue box is given for the user to specify the variable name to export to the Matlab workspace. A variable 'freqs' is automatically exported and contains an array of measurement frequencies.

A2.6 Equaliser Parameters

Low Freq: The lower frequency boundary for which the equaliser will be defined.

High Freq: The upper frequency boundary for which the equaliser will be defined.

Low Level: The lower amplitude for which the equaliser will be defined. This is useful when high noise floors have corrupted the analysis.

Upper Level: The lower amplitude for which the equaliser will be defined.

Order: By default the order of the equaliser is the same as the order of the MLS, but this may be reduced if desired by clicking the 'Other' radio button and specifying an order.

Attenuate only: In cases when a loudspeaker has been tested to extreme voice coil excursions, an equaliser should not introduce gain into the system. The gain of the equalisers derived in x.x should therefore be reduced so that the maximum gain is 0dBFS.

Enable Level-Dependent Equaliser: Enables level-dependent equaliser.

Calculate equalisers: Determines an equaliser according to given parameters and displays the predicted response after equalisation.

Run pre-equalised MLS: As above, but re-runs the pre-equalised MLS analysis to give definitive results. Displays response and calculate pre- and post- equalisation variance.

Export Equaliser: Exports an MxN matrix of equaliser impulse responses, where M is the amplitude levels for which the equaliser is defined and N is the length of the equaliser.

Sample from workspace: Displays a list of variables in the Matlab workspace.

Play Original: Plays variable selected in the variable list at sampling frequency defined in the generator box.

Play Equalised: Plays variable selected in the variable list after equalisation at sampling frequency defined in the generator box.

A2.7 A Typical Session

Objectives: Measure the response of a loudspeaker at levels [-96 -84 -72 -60 -48 -36 -24 -12 0] dBFS, where 0dBFS corresponds to 110dBA at 1m. Use an 11-th order MLS burst with 32 repetitions. Use a sampling frequency of 44100 and resolution of 24 bits. Determine the THD at the same amplitudes and 20 frequency points beginning at 50Hz, with 250ms step times.

Create and test an attenuate-only linear equaliser with a lower frequency boundary of 100Hz and an upper frequency boundary of 18kHz, using levels -60dBFS to 0dBFS.

Method:

1. Set the PC mixer so that the record channel is routed to the input to which the microphone preamplifier is connected (see Chapter 6 for hardware setup guidelines).
2. Place the loudspeaker in the centre of an anechoic environment with a microphone on-axis at 1m. Place a SPL meter next to the microphone. Use the line-up generator to generate a noise burst at 0dBFS and adjust the power amplifier gain until an SPL of 110dBA is measured.
3. Adjust the preamplifier and PC mixer gain so that a -24dBFS line-up burst gives a -24dBFS measured peak amplitude. We use -24dBFS instead of 0dBFS to avoid the nonlinear region, which would cause over-estimated results if used as the line-up level.
4. Tick the Enable MLS box. Type 11 in the MLS order box and 32 in the repetitions box. Click the radio button corresponding to the required amplitudes. Leave the -3dB pad unchecked and the detect offset at the default value.
5. Tick the Enable swept sine box. Type 20 in the frequency points box, 100 in the start freq box and 250 in the step time box. Click the radio button corresponding to the required amplitudes and leave the detect offset at the default value.
6. Set the sampling freq box in the generator section to 44100 and the bits/sample to 24. Press the Begin Analysis button.
7. Listen to the test tones. The Generated sequence box will depict the sequence you hear. Wait for the test tones to finish. The analysis should take a few seconds and will display the results in the signal analysis section.
8. In the equaliser section, set low freq to 100 and high freq to 18000. Use the drop down lists to set the low level to -60 and high level to 0. Set order to 'As MLS' and click the attenuate only box. Click the 'Run pre-equalised MLS'. This will calculate the best-fit linear equaliser and run a pre-equalised MLS sequence. The results figure should be relatively flat in the figure. The variance of the frequency response in the range of interest is displayed in the pre- and post- equalisation variance boxes.

9. To test with real audio sequences, use the Matlab function `wavread` to load a mono, 44100Hz wav file into the workspace. Hit the ‘Sample from workspace’ button to display a list of variables. Click the correct variable and press the ‘Play original’ button to hear the unequalised version. Click ‘Play equalised’ for an equalised version.

Appendix 3 - Command-line tool program listings

```
function impulses =
AnalyseFullSequence(signal,offset,burstlevels,repetitions,N,DCCoupling)

% AnalyseFullSequence
%
% Detects alignment impulse and analyses multi-level MLS bursts
%
% [impulses] = AnalyseFullSequence(signal, offset, burstlevels,
% repetitions, N, DCCoupling)
%
% signal:       Recorded sequence from system under test
% offset:       The impulse detection finds the point where the signal
%               exceeds a certain rate of change. Pulse spreading may cause
%               the detection to be erroneous by a few samples (rarely more
%               than 10) and this may be corrected with the offset
%               parameter. Positive offset causes negative time shift.
% burstlevels:  1xM vector of burst amplitudes (can be linear or dBFS).
% repetitions:  The number of repetitions of each amplitude (cannot be less
%               than 2).
% N:            Order of the MLS, where P=2N-1
% DCCoupling:   Set to true for DC recovery method. Set to false for
%               loudspeaker measurements.
% impulses:     An MxN vector of impulse responses, where M is the number
%               of amplitudes and N is the order of the MLS.

impulsetoburst = 2^N;
bursttoburst = 2^N;
P=2^N-1;
mls = GenerateMLS(N,1);
tagS = GeneratetagS(mls,P,N);
tagL = GeneratetagL(mls,P,N);

impulseindex = FindImpulse(signal)
startindex = impulseindex + impulsetoburst - offset + length(mls);

% Go through all levels
for i=1:1:length(burstlevels)
    % Average all repetitions, excluding first
    acc = 0;
    for j=1:1:repetitions-1;
        startindex;
        acc = acc + signal(startindex:startindex+length(mls)-1);
        startindex = startindex + length(mls);
    end
    mean(i,:) = acc/(repetitions-1); %-1 because we are ignoring the first
burst

    impulses(i,:) = AnalyseMLS(mean(i,:),mls,tagS,tagL,N,DCCoupling);

    startindex = startindex + length(mls) + bursttoburst; % Extra
length(mls) because we want to skip first burst
end
```

```
function impulseresp = AnalyseMLS(signal,mls,tagS,tagL,N,DCCoupling);

% Analyses an MLS sequence.
%
% signal: Must be the same length as the MLS sequence and be taken from the
% beginning of a SECOND MLS sequence to approximate circular convolution.
%
% mls: a P-length MLS sequence (Where P=2^N-1)
%
% fs: sampling frequency in Hz
%
% DCCoupling: True if device under test is DC coupled, false otherwise.

P = 2^N-1;
perm = PermuteSignal(signal, tagS, P, DCCoupling);
had = FastHadamard(perm, P+1, N);
resp = PermuteResponse(had, tagL, P);
impulseresp = resp;

function [freqs levels thd] = AnalyseSteppedSequence(signal, offset,
burstlevels, lowerFreq, n, steptime,fs)

% AnalyseSteppedSequence
%
% Analyses stepped sequence from GenerateSteppedSequence
%
% [freqs levels thd] = AnalyseSteppedSequence(signal, offset, burstlevels,
lowerFreq, n, steptime,fs)
% signal:       Signal from system under test
% offset:       The impulse detection finds the point where the signal
%               exceeds a certain rate of change. Pulse spreading may cause
%               the detection to be erroneous by a few samples (rarely more
%               than 10) and this may be corrected with the offset
%               parameter. Positive offset causes negative time shift.
% burstlevels:  Array of burst amplitudes (dBFS or normalised linear)
% lowerFreq:    Starting frequency
% n:            Number of steps
% steptime:     Time for each step (ms)
% fs:           Sampling frequency (usually 44100Hz)
% freqs:        Array of frequencies for each step
% levels:       Normalised received levels for each step
% thd:          Total harmonic distortion (normalised)


s = size(signal);
if (s(1) > 1)
    signal = signal';
end

upperFreq = fs/2;
r = 10^(log10(upperFreq/lowerFreq)/n);

starttoimpulse = 10000;
frametime = steptime/1000*fs;

impulseindex = FindImpulse(signal)
startindex = impulseindex + frametime - offset;

freqs = lowerFreq*r.^(0:1:n-1);
```

```
% Go through all levels
for i=1:1:length(burstlevels)
    for j=1:1:n;
        frame = signal( startindex:startindex+frametime-1 ); % Grab the
entire frame

        % Calculate levels
        levels(i,j) = sqrt(2)*sqrt(mean(frame.^2));          % Is this the
right thing to do? Turns RMS into peak

        % Calculate THD
        frame = frame.*hamming(frametime)';                  % Window frame
with Hamming window

        framefft = fft(frame);                               % Take FFT of
windowed frame

        framefft = framefft(1:floor(length(framefft)/2));    % Get rid of
aliasing components

        totalWindowedEnergy = sqrt(mean(abs(framefft).^2)); % Calculate
total energy
        fundamentalBin = round(freqs(j)/fs*length(frame))+1; % Find fft bin
of fundamental

        lowNotch = round(0.7*freqs(j)/fs*length(frame))+1;
        highNotch = round(1.5*freqs(j)/fs*length(frame))+1;
        if (highNotch > length(framefft))
            highNotch = floor(length(framefft));
        end

        framefft(lowNotch:highNotch) = 0;                    % Implement
notch

        harmonicEnergy = sqrt(mean(abs(framefft).^2));       % Find the
remaining energy
        thd(i,j) = harmonicEnergy/totalWindowedEnergy;       % Calculate
normalized THD.

        startindex = startindex + frametime;                 % Move onto
next frame
    end

    startindex = startindex + frametime;
end
```

```
%CIRCONVN-point circular convolution
%
%C = CIRCONV(A,B,N) performs the N-point circular convolution
%of vectors A and B.  C is returned as a row vector.  A and B
% must be vectors, but may be of different lengths.  N must be
% a positive, non-zero integer.  The results of CIRCONV will
%match that of CONV if N>=( length(A) + length(B) - 1).  This
%is also the alias-free criterion for the circular convolution.
%
%See also CONV

function[C] = circonv(A,B,N)

% TEST NUMBER OF ARGUMENTS
if nargin~=3,
  error('CIRCONV uses exactly 3 input arguments');
end

% TEST N
if N<=0,
  error('N must be great than zero.');
end

% TEST TO SEE IF EITHER A OR B ARE MATRICES
if ndims(A)>2 | ndims(B)>2 | min( size(A) )>1 | min( size(B) )>1,
  error('circonv works only on vectors');
end

% MAKE SURE VECTORS ARE COLUMN VECTORS SO
% THAT MATRIX OPERATIONS WORK
if size(A,2)>1,
  A=A';
end

if size(B,2)>1,
  B=B';
end

% APPEND ZEROS IF NECESSARY
if N>length(A),
  A=[A ; zeros(N-length(A),1)];
end

if N>length(B),
  B=[B ; zeros(N-length(B),1)];
end

% TAKE ONLY THE FIRST N POINTS
A = A(1:N);
B = B(1:N);

% PRODUCE FOLD ADD TABLE.  IT IS AN NxN SQUARE MATRIX
% AS IS IT IS IN THE FORM NORMALLY USED, BUT THE DIAG
% COMMAND SUMS DIAGONALS TOP LEFT TO BOTTOM RIGHT
% SO WE MUST FLIP IT LEFT-RIGHT
FoldAddTable = A*B';
FoldAddTable = fliplr(FoldAddTable);

% SUM DIAGONALS OF FOLDADDTABLE TO FIND COMPONENTS OF C
```

```
C=zeros(1,N);

% MAIN DIAGONAL ELEMENT
C(N) = sum( diag(FoldAddTable,0) );

% OTHER ELEMENTS ARE THE SUM OF TWO DIAGONALS ONE ABOVE
% THE MAIN AND THE OTHER IN THE COMPLEMENTARY POSITION
% BELOW THE DIAGONAL.  HERE COMPLEMENTARY MEANS THAT
% THE DIFFERENCE IN DIAGONAL LOCATION IS N: (N-x)-(-x)=N
% THE DIAGONALS ARE NUMBERED SUCH THAT 0 IS THE MAIN
% DIAGONAL, +1 IS THE DIAGONAL IMMEDIATELY ABOVE THE MAIN
% DIAGONAL AND -1 IS THE DIAGONAL IMMEDIATELY BELOW
% THE MAIN DIAGONAL.  THIS IS THE CONVENTION OF THE
% DIAG() FUNCTION
for x=1:(N-1),
  C(x)= sum( diag(FoldAddTable, N-x) ) + sum( diag(FoldAddTable, -x) );
end
function equaliser =
CreateFIREqualiser(burstlevels,impulses,leveltype,lowerfbound,upperfbound,a
ttenuate,fs);

% CreateFIREqualiser
%
% Creates a best-fit FIR level-dependent equaliser
%
% equaliser = CreateFIREqualiser(burstlevels, impulses, leveltype,
% lowerfbound, upperfbound, attenuate, fs);
%
% burstlevels:  Array of burst amplitudes (dBFS or normalised linear)
% impulses:     MLS-derived impulse responses (MxN)
% leveltype:    'dbfs' or 'lin'
% lowerfbound:  Lower frequency boundary
% upperfbound:  Upper frequency boundary
% attenuate:    True to create attenuate-only equalisers, false otherwise
% fs:           Sampling frequency (usually 44100Hz)

freqresps = [];
[m,n] = size(impulses);
for i=1:1:m
    freqresps(i,:) = 20*log10(abs(fft(impulses(i,:))));
end

% Convert burst levels to dBFS
if (strcmp(leveltype,'lin'))
    burstlevelsdBFS = 20*log10(burstlevels);
elseif (strcmp(leveltype,'dbfs'))
    burstlevelsdBFS = burstlevels;
else
   error('Unrecognised leveltype')
end


% Find indices of frequency boundaries
lowerboundind = floor(n*lowerfbound/fs+1);
if (upperfbound > fs/2)
   upperboundind = n/2+1;
else
   upperboundind = ceil(n*upperfbound/fs+1);
end
```

```
% Calculate equaliser
for i=1:1:m
   % Bring all up to 0dBFS
   freqresps(i,:) = freqresps(i,:)-burstlevelsdBFS(i);
   % Equalise...
   equaliser(i,1:n) = -mean(freqresps(i,lowerboundind:upperboundind)); %
Fill in unequalised parts with average equalised level for now
   equaliser(i,lowerboundind:upperboundind) = -
freqresps(i,lowerboundind:upperboundind); % Equalise below Nyquist rate
   equaliser(i,:) = abs(10.^(equaliser(i,:)/20)); %Make linear again

   x = fir2(n, 0:2/n:1, equaliser(i,1:n/2+1)); % Calculate linear-phase FIR
   equaliser(i,:) = x(1:n);
end

if (attenuate == true)
  %Normalise to have max gain of 0dBFS
  equaliser = equaliser/abs(max(max(equaliser)));
end

 function y = FastHadamard(x, P1, N)

% FastHadamard
%
% Applies a Fast Hadamard transform to a 1-D signal
%
% y = FastHadamard(x, P, N)
%
% x:     Signal to be transformed
% P:     Length of MLS, where P=2N-1
% N:     Order of MLS
% y:     Transformed signal

k1 = P1;
for k=1:1:N
    k2 = k1/2;
    for j=1:1:k2
        for i=j:k1:P1
            i1 = i + k2;
            temp = x(i) + x(i1);
            x(i1) = x(i) - x(i1);
            x(i) = temp;
        end
    end
    k1 = k1/2;
end

y = x;

% -- This is a Hadamard transform, but it is very slow --
% Is x a column vector?
%if (size(x,1) == 1)
%    x = x';
%end

%y = Hadamard(P1)*x;
```

```
function peakindex = FindImpulse(sample)

% FindImpulse
%
% Detects a positive impulse
%
% [impulseIndex] = FindImpulse(sample)
%
% sample:   The sample to be analysed.
% impulseIndex:     The index of the detected impulse


diffsample = diff(sample); % Differentiate input samples
for i=1:1:length(sample)
    if (diffsample(i) > 0.02)
        peakindex = i+1;    %+1 because Matlab's diff function compares
with previous sample
        break;
    end
end

function sequence = GenerateFullSequence(burstlevels, leveltype,
repetitions, N)

% GenerateFullSequence
%
% Generates an impulse followed by bursts of MLS at different levels.
%
% [sequence] = GenerateFullSequence(burstlevels, leveltype, repetitions, N)
%
% burstlevels:  A 1xM array of amplitudes in dBFS or normalised linear.
% leveltype:    Can be ‘dbfs’ or ‘lin’.
% repetitions:  Number of repetitions of each amplitude.
% N:            MLS order, where P=2N-1.
% sequence:     The generated sequence.


starttoimpulse = 10000;
impulsetoburst = 2^N;
bursttoburst = 2^N;

impulse = [1 zeros(1,impulsetoburst-1)];
sequence = [zeros(1,starttoimpulse) impulse];
mlsfull = GenerateMLS(N,1);

for i=1:1:length(burstlevels)-1
   if (burstlevels(i+1) - burstlevels(i) < 0)
       % Burst levels must be monotonic increasing for later analysis in
       % generation of equalisers.
       error('burstlevels must be monotonic increasing')
   end
end

if (strcmp(leveltype,'dbfs'))
    burstlevels = 10.^(burstlevels./20);
end

for i=1:1:length(burstlevels)
    mls = burstlevels(i).*mlsfull;
    for j=1:1:repetitions
        sequence = [sequence mls];
    end
    sequence = [sequence zeros(1,bursttoburst)];
end

function  y = mls(n,flag)

% GenerateMLS
%
% Generates an MLS sequence
%
% y = mls(n, flag)
%
% n:        order of MLS
% flag:     true for registers initialised to 1, false for random
% y:        P-length MLS sequence, where P=2N-1


switch n                                 %assign taps which will yeild a
maximum
case 2                               %length sequence for a given bit length
   taps=2;                           %I forget the reference I used, but
theres
   tap1=1;                           %a list of appropriate tap values in
   tap2=2;                           %Vanderkooy, JAES, 42(4), 1994.
case 3
   taps=2;
   tap1=1;
   tap2=3;
case 4
   taps=2;
   tap1=1;
   tap2=4;
case 5
   taps=2;
   tap1=2;
   tap2=5;
case 6
   taps=2;
   tap1=1;
   tap2=6;
case 7
   taps=2;
   tap1=1;
   tap2=7;
case 8
   taps=4;
   tap1=2;
   tap2=3;
   tap3=4;
   tap4=8;
case 9
   taps=2;
   tap1=4;
   tap2=9;
case 10
   taps=2;
   tap1=3;
   tap2=10;
case 11
```

```
    taps=2;
    tap1=2;
    tap2=11;
case 12
    taps=4;
    tap1=1;
    tap2=4;
    tap3=6;
    tap4=12;
case 13
    taps=4;
    tap1=1;
    tap2=3;
    tap3=4;
    tap4=13;
case 14
    taps=4;
    tap1=1;
    tap2=3;
    tap3=5;
    tap4=14;
case 15
    taps=2;
    tap1=1;
    tap2=15;
case 16
    taps=4;
    tap1=2;
    tap2=3;
    tap3=5;
    tap4=16;
case 17
    taps=2;
    tap1=3;
    tap2=17;
case 18
    taps=2;
    tap1=7;
    tap2=18;
case 19
    taps=4;
    tap1=1;
    tap2=2;
    tap3=5;
    tap4=19;
case 20
    taps=2;
    tap1=3;
    tap2=20;
case 21
    taps=2;
    tap1=2;
    tap2=21;
case 22
    taps=2;
    tap1=1;
    tap2=22;
case 23
    taps=2;
    tap1=5;
    tap2=23;
```

```
case 24
   taps=4;
   tap1=1;
   tap2=3;
   tap3=4;
   tap4=24;
%case 25
%    taps=2;
%    tap1=3;
%    tap2=25;
%case 26
%    taps=4;
%    tap1=1;
%    tap2=7;
%    tap3=8;
%    tap4=26;
%case 27
%    taps=4;
%    tap1=1;
%    tap2=7;
%    tap3=8;
%    tap4=27;
%case 28
%    taps=2;
%    tap1=3;
%    tap2=28;
%case 29
%    taps=2;
%    tap1=2;
%    tap2=29;
%case 30
%    taps=4;
%    tap1=1;
%    tap2=15;
%    tap3=16;
%    tap4=30;
%case 31
%    taps=2;
%    tap1=3;
%    tap2=31;
%case 32
%    taps=4;
%    tap1=1;
%    tap2=27;
%    tap3=28;
%    tap4=32;
otherwise
   disp(' ');
   disp('input bits must be between 2 and 24');
   return
end

if (nargin == 1)
    flag = 0;
end

if flag == 1
    abuff = ones(1,n);
else
    rand('state',sum(100*clock))
    while 1
        abuff = round(rand(1,n));
        %make sure not all bits are zero
        if find(abuff==1)
            break
        end
    end
end

for i = (2^n)-1:-1:1

   xorbit = xor(abuff(tap1),abuff(tap2));       %feedback bit

   if taps==4
      xorbit2 = xor(abuff(tap3),abuff(tap4));%4 taps = 3 xor gates & 2
levels of logic
      xorbit = xor(xorbit,xorbit2);             %second logic level
   end

    abuff = [xorbit abuff(1:n-1)];
    y(i) = (-2 .* xorbit) + 1;      %yields one's and negative one's (0 ->
1; 1 -> -1)
end

function signal = GenerateSignal(mls, b, P);

% GenerateSignal
%
% Generates a signal by circularly convolving a filter with coefficients b
% with an MLS signal. Used for test purposes.
%
% signal = GenerateSignal(mls, b, P);
% mls:  MLS signal
% b:    FIR coefficients of filter
% P:    Length of MLS, where P=2N-1


% Convolve input with MLS
plot(abs(fft(b)));
title('Actual filter');

%signal = circonv(b,mls,P);         %Circular convolution
signal = filter(b,1,[mls mls]);         %Discrete-time convolution
signal = signal(end-P+1:end);

function sequence = GenerateSteppedSequence(burstlevels, leveltype,
lowerFreq, n, steptime,fs)

% GenerateSteppedSequence
%
% Generates a sequence of stepped sines at multiple amplitudes, impulse-
aligned
%
% sequence = GenerateSteppedSequence(burstlevels, leveltype, lowerFreq, n,
steptime,fs)
%
% burstlevels:  Array of burst amplitudes (dBFS or normalised linear)
% leveltype:    ‘dbfs’ or ‘lin’
% lowerFreq:    Starting frequency (Hz)
% n:             Number of levels
% steptime:      Time for each step (ms)
% fs:            Sampling frequency (usually 44100Hz)
% sequence:      The generated sequence


upperFreq = fs/2;

r = 10^(log10(upperFreq/lowerFreq)/n);

% Make frequencies

starttoimpulse = 10000;
frametime = steptime/1000*fs;

impulse = [1 zeros(1,frametime-1)];
sequence = [zeros(1,starttoimpulse) impulse];

t = 0:1:frametime-1;

sines = [];
for j=1:1:n
    f = lowerFreq*r^(j-1);
    sines = [sines sin(2*pi/fs*t*f)];
end
sines = [sines zeros(1,frametime)];

for i=1:1:length(burstlevels)-1
   if (burstlevels(i+1) - burstlevels(i) < 0)
        % Burst levels must be monotonic increasing for later analysis in
        % generation of equalisers.
        error('burstlevels must be monotonic increasing')
   end
end

if (strcmp(leveltype,'dbfs'))
    burstlevels = 10.^(burstlevels./20);
end

for i=1:1:length(burstlevels)
    sequence = [sequence sines.*burstlevels(i)];
end

function tagL = GeneratetagL(mls, P, N);

% GeneratetagL
%
% Generates array for the rearrangement of samples after a Hadamard
Transform
%
% tagL = GeneratetagL(mls, P, N);
%
% mls:      The MLS signal for which tagL is valid.
% P:        Length of the MLS signal
% N:        Order of the MLS signal
% tagL:     1x(P+1) vector of indices.
% Convert {-1,1} to binary
binmls = (mls-1)./-2;

S = GeneratetagS(mls,P,N);

% Find which values of the tagS vector are powers of 2
for i=1:1:P
    for j=1:1:N
        if (S(i) == 2^(j-1))
            index(j) = i;
        end
    end
end

index = index;

powerindices = 0:1:N-1;
powers = 2.^powerindices;

for i=1:1:N
  L(i,1:mod(index(i),P)) = binmls(mod(index(i),P):-1:1);
  L(i,mod(index(i),P)+1:P) = binmls(P:-1:mod(index(i),P)+1);
end

tagL = powers*L;

function tagS = GeneratetagS(mls, P, N)

% GeneratetagS
%
% Generates array for the rearrangement of samples before a Hadamard
Transform
%
% tagS = GeneratetagS(mls, P, N);
%
% mls:      The MLS signal for which tagS is valid.
% P:        Length of the MLS signal
% N:        Order of the MLS signal
% tagS:     1x(P+1) vector of indices.


% Convert {-1,1} to binary
binmls = (mls-1)./-2;

% Make S matrix by making first line mls and shifting every subsequent row
% RIGHT up to N then multiply each row by the correct power of 2.
powerindices = N-1:-1:0;
powers = 2.^powerindices;

for i=1:1:N
   S(i, 1:i-1) = binmls(P-i+2:P);
   S(i, i:P) = binmls(1:P-i+1);
end

tagS = powers*S;
```

```
function [y] = NonlinearFFTFilt(burstLevels, levelType, equaliser, x)

% NonlinearFFTFilt
%
% Like Matlab FFTFILT, but applies level-dependent FFT multiplication
%
% y = NonlinearFFTFilt(burstlevels,leveltype,equaliser,x)
%
% burstlevels:  Array of burst amplitudes (dBFS or normalised linear)
% leveltype:    'dbfs' or 'lin'
% equaliser:    MxN array of equaliser impulse responses
% x:            Signal to be equalised
% y:            Equalised signal


s = size(equaliser);

% Convert to linear bursts
if (strcmp(leveltype,'dbfs'))
    burstlevels = 10.^(burstlevels./20);
end

% Get equaliser into frequency domain of length(x)
for i=1:1:s(1)
   equaliser(i,:) = fft(equaliser(i,:),length(x));
end

% Get input signal into frequency domain
fftX = fft(x);

y = zeros(1,length(x));
for i=1:1:length(fftX)
   level = abs(fftX(i));
   %Go through each bin
   for j=1:1:length(burstlevels) % Find correct level
      if( burstlevels(j) > level)
         y(i) = fftX(i).*equaliser(j-1,i);
         break;
      end
   end
end

y = ifft(y);

function resp = PermuteResponse(perm, tagL, P)

% PermuteResponse
%
% Rearranges output of Hadamard Transform according to tagL.
%
% resp = PermuteResponse(perm, tagL, P)
%
% perm:     Output of Hadamard Transform
% tagL:     1xP vector of indices from GeneratetagL.
% P:        Length of MLS, where P=2N-1
% Resp:     Rearranged signal


fact = 1/(P+1);
```

```
%Ignore first element
perm = perm(2:end);

%for i=1:1:P
%    resp(i) = perm(tagL(i))*fact;
%end

resp = perm(tagL).*fact;

resp(P+1) = 0;

function perm = PermuteSignal(signal, tagS, P, dcCoupled);

% PermuteSignal
%
% Rearranges input signal according to tagS.
%
% perm = PermuteSignal(signal, tagS, P, dcCoupled);
%
% signal:       Signal to be arranged
% tagS:         1xP vector of indces from GeneratetagS
% P:            Length of MLS, where P=2N-1
% DCCoupled:    Set to true for DC recovery method. Set to false for
%               loudspeaker measurements.
% perm:         The rearranged signal


%DC coupling:
if (dcCoupled == 1)
    dc = 0;
    for i=1:1:P
        dc = dc + signal(i);
    end
    perm(1) = -dc;
else
    perm(1) = 0;    % Not sure if this is the right thing to do yet...
end

for i=1:1:P
    perm(tagS(i)+1) = signal(i);
end
```