2017

DEVELOPMENT OF AN IMPROVED TIME VARYING LOUDNESS MODEL WITH THE INCLUSION OF BINAURAL LOUDNESS SUMMATION

Jeremy Edward Charbonneau
University of Windsor

DEVELOPMENT OF AN IMPROVED TIME VARYING LOUDNESS MODEL WITH THE INCLUSION OF BINAURAL LOUDNESS SUMMATION

By

Jeremy Charbonneau

A Dissertation
Submitted to the
Faculty of Graduate Studies
through the Department of
Mechanical, Automotive & Materials Engineering
in Partial Fulfillment of the Requirements for the
Degree of Doctor of Philosophy
at the University of Windsor

Windsor, Ontario, Canada
2017

Development of an Improved Time Varying Loudness Model with the Inclusion of Binaural Loudness Summation

by

Jeremy Charbonneau

APPROVED BY:

_____

L. Wilber, External Examiner
Northwestern University

_____

R. Barron
Department of Mathematics & Statistics

_____

R. Gaspar
Department of Mechanical, Automotive & Materials Engineering

_____

E. Tam
Department of Mechanical, Automotive & Materials Engineering

_____

C. Novak, Advisor
Department of Mechanical, Automotive & Materials Engineering

May 8, 2017

DECLARATION OF ORIGINALITY

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication.

I certify that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis and have included copies of such copyright clearances to my appendix.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.

ABSTRACT

As the perceived quality of a product is becoming more important in the manufacturing industry, more emphasis is being placed on accurately predicting the sound quality of everyday objects. This study was undertaken to improve upon current prediction techniques with regard to the psychoacoustic descriptor of loudness and an improved binaural summation technique.

The feasibility of this project was first investigated through a loudness matching experiment involving thirty-one subjects and pure tones of constant sound pressure level. A dependence of binaural summation on frequency was observed which had previously not been a subject of investigation in the reviewed literature.

A follow-up investigation was carried out with forty-eight volunteers and pure tones of constant sensation level. Contrary to existing theories in literature the resulting loudness matches revealed an amplitude versus frequency relationship which confirmed the perceived increase in loudness when a signal was presented to both ears simultaneously as opposed to one ear alone. The resulting trend strongly indicated that the higher the frequency of the presented signal, the greater the increase in observed binaural summation.

The results from each investigation were summarized into a single binaural summation algorithm and inserted into an improved time-varying loudness model. Using experimental techniques, it was demonstrated that the updated binaural summation algorithm was a considerable improvement over the state of the art approach for predicting the perceived binaural loudness. The improved function

retained the ease of use from the original model while additionally providing accurate estimates of diotic listening conditions from monaural WAV files. It was clearly demonstrated using a validation jury test that the revised time-varying loudness model was a significant improvement over the previously standardized approach.

DEDICATION

This dissertation is dedicated to my family and friends. To my parents Patricia and Alfred Charbonneau who inspired those inquisitive traits which led me to academics in the first place. They have always supported all my decisions and career choices. My siblings Aaron and Rebecca who while not always nearby could always be relied upon whenever I needed them.

A dedication to my various groups of friends and colleagues who were present in both my academic and personal life; as they made the doctorate experience an unforgettable one.

Lastly this work is dedicated to my wonderful wife Stephanie. Without her encouragement, this project would have been a missed opportunity. Her continuous support and patience through many hours of proof reading have been invaluable in bringing this project to fruition.

ACKNOWLEDGEMENTS

The author would like to thank the advisory committee for their continuous support and constructive feedback. The document went through several revisions and their insight was invaluable for keeping the focus of the project on the right track.

A special thank you to Brian C.J. Moore and Brian R. Glasberg for providing a working model of their loudness metric for modification. Without this, there would have been no study.

Lastly, it is prudent to thank the participants who volunteered for the respective studies. Without their time, dedication and feedback none of this research would have been possible.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

NOMENCLATURE

α_a	short term loudness estimate
α_{al}	long term loudness estimate
A1	amendment 1
AGC	automatic gain control
atm	atmospheres (pressure unit)
ANSI	American National Standards Institute
Bark	unit of critical bandwidth, in memory of Bark-Hausen
BDEL	binaural difference required for equal loudness
B_f	frequency of binaural tone
B_{Lp}	amplitude of binaural tone
°C	degrees Celsius
dB	unit decibels
dBA	A-weighted decibels
DC	direct current
DIN	Deutsches Institut für Normung (Institute for Standardization)
E	excitation level
ERB	equivalent rectangular bandwidth
f	frequency (Hz or kHz)
fc	centre frequency (Hz or kHz)
FFT	Fast Fourier Transform
FIR	finite impulse response filter
FRF	frequency response function
HATS	head and torso simulator device
Hz	Hertz (1 cycle per second)
ISO	International Organisation for Standardization
kHz	kilohertz (1000 cycles per second)
kPa	kilopascal
L_N	loudness level (phon)
L_p	sound pressure level (dB)
ΔL_p	change in pressure level (dB)
L_{pi}	sound pressure level in a noise band (dB)
MAF	minimum audible field
MAP	minimum audible pressure
M_f	frequency of monaural tone
M_{Lp}	amplitude of monaural tone
ms	milliseconds
n	index notation
N	overall loudness (sone)
N’	specific loudness (sone/Bark or sone/number of ERBs)
N_5	percentile loudness scale exceeded 5% of the duration
NL	non-linear temporal decay block
NVH	noise, vibration, and harshness
P	sound pressure (unit Pa)
Pa	Pascal

PC	personal computer
phon	unit of loudness level
P_{ref}	reference sound pressure, 20 µPa
RMS	root-mean-squared
s	second
S_n	instantaneous loudness at time instance ‘n’
S'_n	short term estimate of loudness
S'_{n-1}	the previous estimate of short term loudness
$S''n$	long term estimate of loudness
S''_{n-1}	the previous estimate of long term loudness
SLM	sound level meter
sone	unit of loudness
SPL	sound pressure level
TP	label for low-pass filters in DIN45631/A1 loudness model
TVL	time-varying loudness model
µPa	micropascals
WAV	audio file format or waveform file

I. INTRODUCTION

Over the past several decades urban communities experienced increasing noise levels as populations rose and residential sectors encroached on louder industrial regions. To alleviate this problem, municipal and provincial regulators set acoustical limits for commercial businesses as a means of reducing the overall noise levels near sensitive areas. While such guidelines were based on the most up-to-date information at the time, the technology required to enforce them was relatively expensive and not readily available. Moreover, the technical competence of the users in both measurement and interpretation of the data required training beyond that to be found among most local enforcement agencies.

Since the 1930's, it was clear that linear sound pressure measurements did not correlate well with the actual perception of a sound signal. The reason for this being that the sensitivity of the human auditory system tends to vary with respect to frequency. To correct for this variance, a filtering function known as the A-weighting filter was developed to approximate the human perception of sounds. The resulting sound pressure level is presented as an A-weighted decibel (dBA) that can be compared to the applicable criteria and summarized as a pass/fail rating for regulatory bodies.

This method for conformance continued for years although it was only meant to be a temporary solution to a complex problem. Once these criteria were set the municipalities grew accustomed to the limits and the regulatory agencies became resistant to change. The problem escalated further as the approach was applied to the product

development and automotive industries; becoming a common measure for quality control. Technology improved and sound level meters with built-in A-weighting functions became more affordable. All the above established a comfort in the A-weighted decibel system regardless of the numerous studies indicating a fundamental problem with using the A-weighting system; the A-weighting approximation was only intended to correct the sound spectra specifically in the range of the 40-phon equal loudness contour in both shape and amplitude. As this specific pattern was not commonly encountered in environmental measurements, the A-weighting function was an inappropriate means of quantifying environmental noise measurements by the acoustical consulting community. The predicament in this case was that consulting groups were obligated to follow the measurement guidelines imposed by the municipal and provincial regulating authorities, regardless of a desire to present better representative information.

Unlike early noise measuring instrumentation, the current generation of handheld sound analyzers have become portable, accurate and capable of recording prolonged measurements with high sampling rates. Real-time signal processing and reporting has become possible for the simultaneous operation of multiple complex algorithms. This has allowed for the inclusion of increasingly complex prediction models for sound perception which could be used in place of the simplified A-weighting filter. An example of this is the *loudness algorithm*; a powerful function which fully accounts for the perceived intensity of a signal including the hearing sensitivity to both amplitude and frequency, as well as additional hearing mechanisms including masking, summation and temporal averaging.

Loudness is defined as the subjective quality of a sound relating to how an individual perceives the acoustical energy. Varying person-to-person, the true perception of a sound can depend on both the psychological and physiological conditions of the individual's current state including fatigue and hearing sensitivity (or acuity), [1]. Predictions of loudness can be achieved using a series of electronic auditory filters to mimic the sensitivity of the human ear that varies with the amplitude and frequency. Once calculated, loudness predictions can then be used as an input into more complex psychoacoustic descriptors such as annoyance, tonality and sharpness; all of which contribute useful information regarding the overall quality of sounds.

The first loudness models were developed during the 1960's and were only applicable for steady-state signals which did not vary with respect to time. The focus with these early models was on improving the loudness functions to better reflect how the perceived loudness varies with amplitude. As these stationary models improved, subsequent research became focused on the development of non-stationary or time-varying loudness models which are substantially more complex. This requires a running analysis of the transient sound to account for any temporal changes in both the amplitude and frequency. With recent advances to the Glasberg and Moore Time-Varying Loudness (TVL) model it is possible to better predict the human perception of loudness which includes consideration of temporal, spectral and overall intensity, [2, 3, 4].

In the pursuit to continually improve noise measurement techniques, the objective of this study is to improve on the capabilities of the TVL metric through an investigation of the binaural summation mechanism. Specifically, it is proposed that the correction

applied for a diotic (two-channel) signal presentation may be influenced by the frequency content of that signal and the summation mechanism should be adjusted to reflect this.

Binaural measurements are often measured by a Head And Torso Simulator (HATS). These specifically designed acoustic acquisition tools are uniquely suited to provide reproducible measurements and simulating how an acoustic signal reaches the tympanic membrane of the human ear. When using a HATS system to acquire an acoustic signal, the information at the left and right ear channels are encoded with environment specific information, including directional cues and frequency shifts caused by reflections and absorption because of the profile of the torso and pinna contours. The signals measured at each in-ear microphone can be combined to obtain a binaural loudness estimation, however there has been a great deal of debate regarding how the two signals should be combined so that the resultant adequately represents the human perception of binaural sound. For example, consider the case of a frontal incidence sound source. When listening to this source, a diotic case is often observed where equal sound pressure levels are presented to each ear with nearly identical spectra. Several theories exist on how to best combine the two spectra including:

(i) a simple sum of the two loudness values (the loudness of the two signals was a factor of 2 greater than the loudness of one signal alone);
(ii) a constant increase in loudness by a factor greater than 1 but less than 2;
(iii) an increase in loudness which varies with respect to amplitude; and,
(iv) an average of the two loudness levels.

Each of these approaches has been supported for decades and continue to receive support in recent literature. The intention of this study is to show that binaural summation

does exist (and therefore disproves theory (iv) above), while providing insight as to how the binaural loudness summation varies with respect to frequency.

Initial studies focused on the determination of the summation amount or ratio to account for binaural loudness. From the many available methods (described in more detail in later chapters), a loudness matching paradigm was chosen for this research as it gives a direct response to the binaural summation mechanism for each individual. For this, a large sample of normal hearing listeners were asked to match the perceived loudness of presented monaural tones compared against diotic samples at the same frequency. The design and results of the listening test produced a measure of the binaural difference required for equal loudness (BDEL) at each frequency. The result is an estimation of the derived binaural summation mechanism.

The above discussion outlines the existing problem in acoustics where the new technology and theories developed by academia meet resistance from the governmental bodies. This resistance is due to a lack of understanding or confusion resulting from multiple theories presented on the same topic. Given the approach and objectives outlined above, the goals of this study are outlined as follows:

- Confirm the actual binaural summation mechanism utilizing a large sampling of normal hearing individuals to guarantee an accurate representation of the subjective quality.
- If necessary, propose a new calculation technique unique in utilizing the most up-to-date measurements of true human perception with regard to binaural summation.
- Use the findings to provide a major contribution to the measurement and reporting of environmental noise for conformance investigations.

Further, it was anticipated that this technology may perhaps open new avenues of research into the field of environmental noise perception including elaboration on this important algorithm for sound localization models and long term monitoring techniques. Findings of this investigation were implemented into an updated TVL loudness model capable of refined loudness estimates based on current information. This is followed by a discussion of the validity of the results along with a review of the performance of the model based on additional jury test results.

II. NUMERICAL AND PRACTICAL TECHNIQUES

The development of the psychoacoustic-quantity loudness required extensive study and collaborative research on a global scale. With such a rich history, the current chapter provides an introduction to loudness including a brief review of its development and an overview to the intricate mechanisms involved. This knowledge provided the foundation for the present study as an understanding of these concepts was essential for the discussion which followed. At the end of this chapter the focus of this investigation will be identified prior to a review of the literature and conflicting theories.

2.1 Sound Pressure Versus Sound Pressure Level

For the purposes of this work, the measured amplitude of a signal may be expressed as a Sound Pressure Level (SPL for short, L_p) in units of decibels (dB). This condition was used in accordance with the standard North American practices and conventions. The decibel measures a logarithmic unit of pressure calculated via reference value where sound pressure fluctuations were measured in units of Pascals (Pa) and readily converted to decibels using **Equation 1** below. The reference sound pressure of 20 micropascals (μPa) used here was the approximate sound pressure required for an average young adult to detect a pure tone presented between the frequencies of 1 kilohertz (kHz) and 3 kHz, [5].

$$L_p(dB) = 20\log_{10}\left(\frac{x\ (Pa)}{20\ \mu Pa}\right) \quad \textbf{... (1)}$$

The term sound pressure level was often associated with the volume or magnitude of a noise source. When reported to a regulating body this value indicated the amplitude

of a sound as it was measured at the monitoring location. However, as more detail was often required to sufficiently describe the noise source (such as the spectral content and temporal characteristics), other descriptors such as loudness were developed with strong support in the acoustics industry.

2.2 Mechanisms of a Loudness Model

The following key developments and theories form a basic understanding of loudness and provide important insight as to how each model was developed and functions. Such concepts include: the equal loudness contours, the filtering mechanism of the human ear and the concept of spectral masking applicable to loudness.

2.2.1 History of Loudness and the Equal Loudness Contours

Contrary to sound pressure level, the loudness of a signal included subjective qualities of the source which were specific to the trends in human perception. A measured sound pressure level confirmed only the amplitude of a signal at the measurement location. Loudness, on the other hand, was uniquely based on both the frequency content and amplitude of the signal; giving more meaning to the results. In other words, while the sound pressure level was a physical quantity of the signal, loudness was a psychoacoustic descriptor and very much a subjective perception of intensity which may vary between listeners.

In many ways, Kingsbury was considered one of the forefathers of quantifying loudness; his study from the Bell Laboratories in 1927 probed the relative perception of 22 subjects in order to determine a relationship between an individual's response to pure tones and their associated perceived intensity, [6]. For the remainder of this document a

pure tone may be defined as any acoustical signal which can be described as a single sinusoidal wave function. While investigating the limitations of the audible spectrum Kingsbury was able to derive contours of equal loudness by using pure tones which spanned the frequency range from 60 Hertz (Hz) up to and including 4000 Hz or 4 kilohertz (kHz), [6]. This exploration was one of the first to suggest that hearing sensitivity may vary across the frequency spectrum. Of additional note, his findings also confirmed that no definite trend existed between the left and right ears and that the sex of an individual did not have any influence on the sensitivity of hearing.

The work of Kingsbury was followed by several updates to the equal loudness contours with the common goal of quantifying the audible range for the typical, normal hearing observer, [1]. This becomes apparent as any discussion relating to this important data-set must include reference to the work of Fletcher and Munson; who conducted their own investigation in 1933 while working at the Bell Telephone Laboratories, [1]. Their results compiled the first complete set of equal loudness contours for free-field listening, (see hashed lines in **Figure 2.1** below). 'Free-field' in this instance was defined as a listening environment where the signal reached the observer in a relatively pure state, free of any obstructions or reflections. The lowermost contour was particularly important as it represented the threshold of hearing where pure tones become just audible when perceived in quiet listening environments (shown here as a loudness level of 0 dB).

To ensure compatibility with all future work and to remove any confusion between studies, Fletcher and Munson set out specific definitions for the acoustical terms used for their study, [1]. Until their summary, these terms were often used interchangeably or with slightly different meanings which increased potential for confusion or misinterpretations

between studies. Of the definitions identified none were more relied upon in future investigations than the definition of the 1 kHz reference tone. While any signal could have been chosen for this purpose, the authors selected this reference for several logical reasons: it was relatively simple to define, had historically been used as a reference in previous studies, the signal was located near the centre of the audible hearing spectrum and it was mathematically convenient for simplifying formulas. Consequently, the 1 kHz tone became a benchmark reference for nearly all future studies involving loudness and, by extension, the basis for identifying the equal loudness contours as discussed below, [1].

Fletcher and Munson's equal loudness contours were derived using a loudness matching procedure. Subjects were asked to adjust the amplitude of a target tone until the perceived amplitude was equally as loud as the reference 1 kHz sinusoid. Once the two tones were perceived to be equally intense the sound pressure level was recorded for that frequency and the observer proceeded to match the next pair of signals. Contours soon began to develop, each representing pure tones of varying frequency which were perceived to be equally as loud as the reference for that contour. The equal loudness contours were derived in this manner from the threshold of hearing up to the threshold of feeling at 10 dB increments.

The threshold of feeling is another subjective term used to describe the point at which an audible sound begins to invoke an additional sensation which the participant 'feels.' The sensation may very person to person as an increased pressure sensation, internal vibration or discomfort resulting from the high amplitude signals. This is most evident in **Figure 2.1** below where the upper and lower audible extremes of various

studies have been summarized from the minimal audible field (MAF), or minimal audible perception (MAP) up to the threshold of feeling (Feeling), or threshold of discomfort (Discomfort). The area between these two extremes summarizes the range of 'normal' human perception.

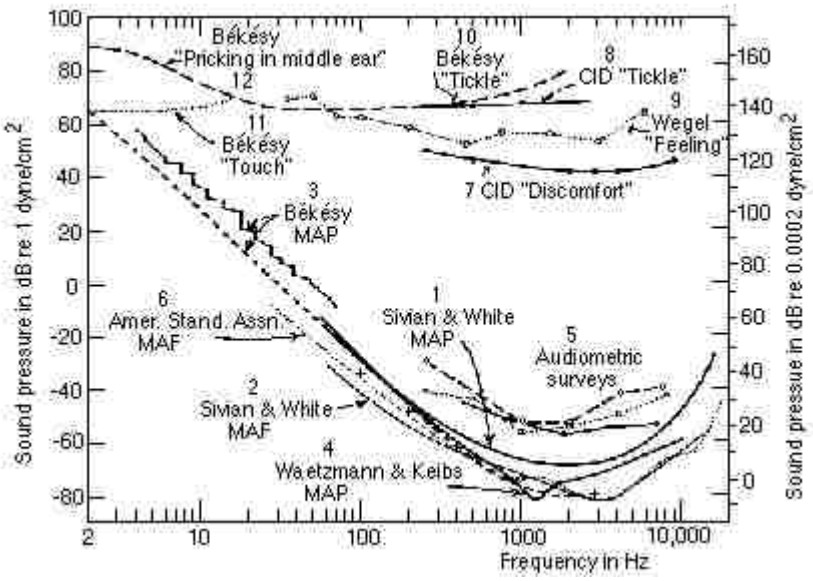


Figure 2.1 – Threshold of hearing compared to the threshold of feeling/discomfort, [7].

To quantify loudness within the normal hearing range, two numerical scales were developed as the research progressed. The most popular scale in recent literature was the unit phon. As each equal loudness contour was generated with respect to a 1 kHz reference tone, each contour was identified by the value at this location as having a loudness level equal to the amplitude. For example, a contour with an amplitude of 40 dB at 1 kHz is referred to hereafter as the 40-phon contour. By extension successive points along each equal loudness contour represent pure tones of equal loudness level (i.e. pure tones of varying frequency, each with an amplitude of 40 phons). The less common unit of loudness, the sone, was the first measure adopted.

A unit sone provided a measure more closely related to a doubling or halving of a signal. Taking the 40-phon contour as an example, a level of 40-phons corresponded to a loudness of 1 sone. As the loudness intensity appeared to double at 50-phons the associated loudness was 2 sones. Similarly, a halving of intensity was perceived from 40- to 30-phons which resulted in a loudness value of 0.5 sones. This trend continued for each respective equal loudness contour, always with reference to the 40 phon loudness of 1 sone.

Fletcher and Munson's results have been regarded as the most influential loudness investigation to date in producing the first complete picture of human perception from the threshold of hearing to the threshold of pain and spanning the entire frequency range of perception. Around the same time Sivian and White, as well as Churcher and King, were all investigating the same phenomenon; the results of each of these studies were consolidated into one plot as presented in **Figure 2.2** below, (for reference the Churcher and King results were identified in the figure legend as 'x Present Paper'), [8, 9]. It was clear there were significant differences between the studies particularly in the lower frequency regions.

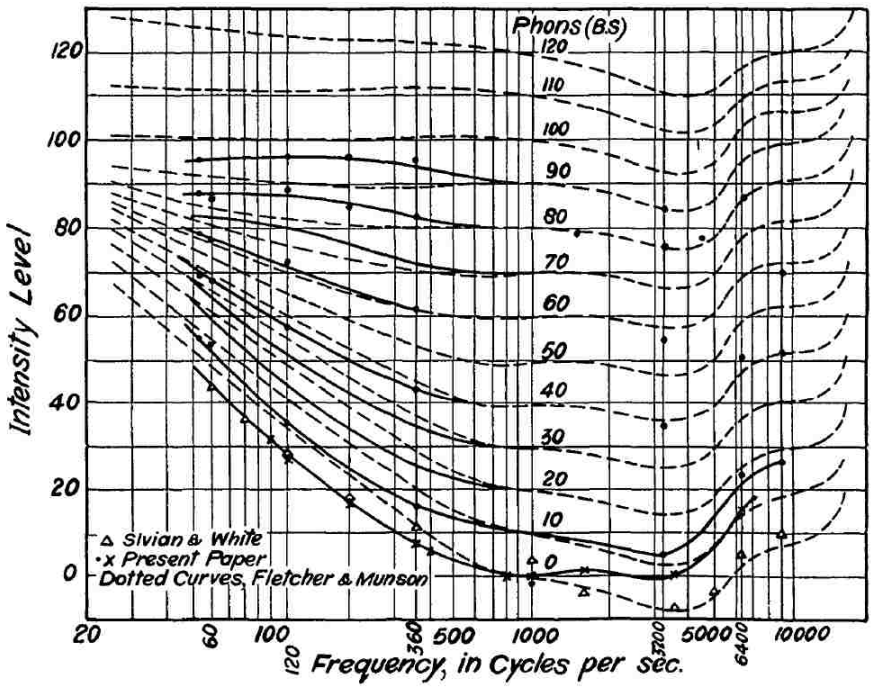
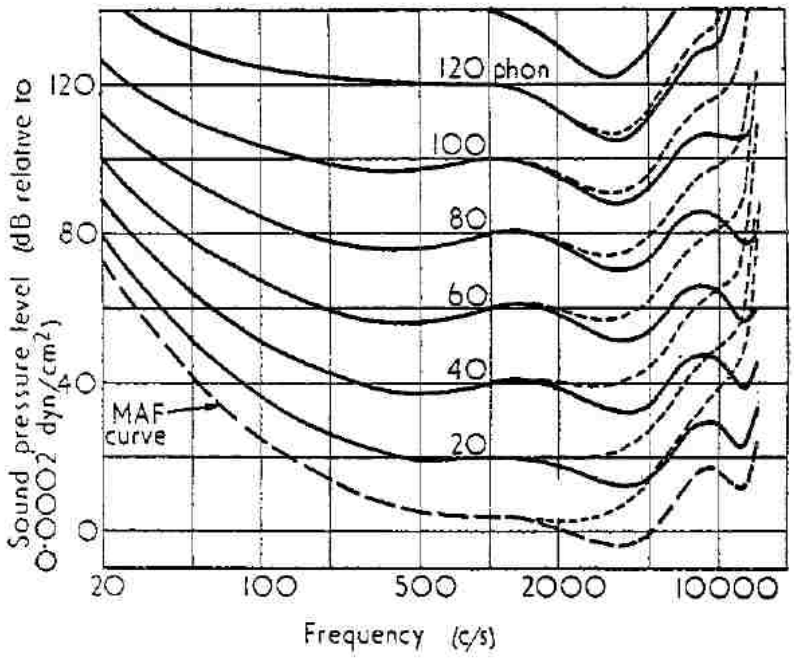


Figure 2.2 – Comparison plot for early versions of the contours [9].

In 1956 Robinson and Dadson were tasked with settling these differences with the assistance of the National Physical Laboratory, [10]. Their research incorporated a four-year study to further the work of Fletcher and Munson and build on their investigation. To improve on the results, the authors greatly increased the number of subjects tested (from 10 to 120), and improved on the investigation techniques. The resulting contours demonstrated smooth trends in hearing which ranged from 20 Hz up to and including 10 kHz for ontologically normal listeners between 16 and 63 years of age. Their work as shown in **Figure 2.3** would soon be accepted as the first standardized set of equal loudness contours by the International Organization for Standardization (ISO) as ISO 226:1961.



Equal-loudness contours

——— age twenty years; – – – – age sixty years.

Figure 2.3 – Equal loudness contours as derived by Robinson and Dadson [10].

Of note in the above figure was the influence of age on hearing sensitivity. Clear deviations have been observed between the different age groups as the frequency increased above 1 kHz. Two such groups are indicated above while similar trends to a lesser degree were observed by the authors for the other age groups as well.

As more information became available and technology improved, the ISO 226 equal loudness contours underwent several revisions with the first occurring in 1987 and the most recent in 2003. Unlike previous versions, the current loudness pattern has significantly larger slopes as was particularly evident below 1 kHz. For comparison, the two latest versions of the standard have been included in **Figure 2.4** below.

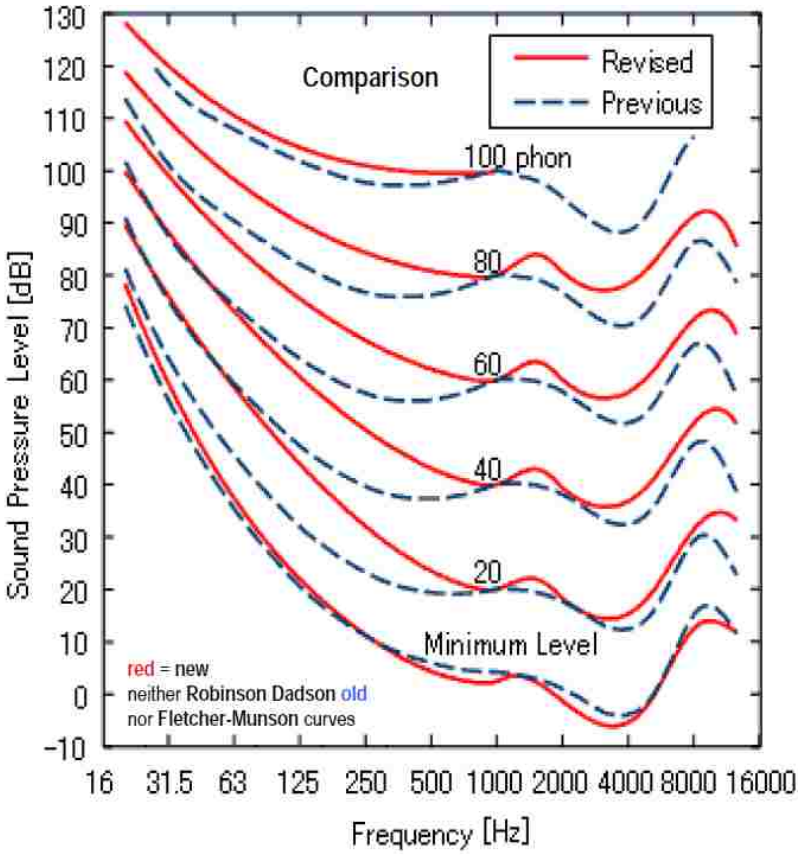


Figure 2.4 – Comparison of the old ISO 226:1987 equal loudness contours and the revised ISO 226:2003 curves [11].

The equal loudness contours provided an important foundation for the development of all known loudness metrics. Standardization of the contours ensured that all such models would be based on a common set of reference levels on which the complex algorithms could be built. With the foundation set the next step was a matter of converting a recorded signal from a physical spectrum into an approximation of the human ear's response. While there were several theories as to how this should be done, the two most prominent approximations became the Bark Band model and the Equivalent Rectangular Bandwidth (ERB) model, each identified below.

2.2.2 Critical Bandwidths and Function Basilar Membrane

The primary function of the ear (and, in particular, the basilar membrane), is to convert sound pressure waves into nerve impulses complete with spectral cues. Through many years of research, it was determined that the perceived intensity of a signal was directly related to the total number of cochlear impulses transmitted to the brain, [1]. Thus, it became well known as early as the 1930s that the loudness associated with a complex tone (or a series of pure tones) was the sum of the individual intensities of each of the tonal components, [1]. When Fletcher and Munson's model was initially published, the authors had a theory relating to this frequency summation phenomenon. Simply put, when two pure tones were close enough in frequency the resulting audible impression was that of only a single component, [1]. A trial and error approach revealed that as the gap between these frequency components increased there was a point where the two pure tones began to sound distinctly different from one another. Once this occurred the tones were considered to be within separate bands of sensitivity which appeared to correspond to specific regions along the basilar membrane. The bandwidth of these regions appeared to grow with frequency and were relatively constant between observers. Based on these findings it was generally accepted that the function of the cochlea could be approximated as a series of band-pass filters of increasing bandwidth with frequency.

Eventually a system was adopted which divided the audible spectrum mathematically into Constant Percentage Bandwidths (CPBs) dependent on the frequency. The standardization organizations accepted the division based on preferred frequency values where approximations were mathematically convenient for both calculations and reporting. As such, the octaves and 1/3-octave bandwidths became the

preferred system for representing acoustic measurements; the exact values of the mid-band frequencies for each system are available as identified in ANSI S1.11, [12].

Zwicker (Critical-Band Function, Bark Bands)

Unfortunately, the convenience of the octave scale was accepted at the cost of accuracy within the audible spectrum. In reality, the bandwidths and band-edge values for frequency sensitivity have been found to vary considerably from those convenient values identified above. In 1961 Zwicker produced a letter to the editor summarizing an improved German based standard for subdividing the audible frequency range. In this system, the Frequenzgruppen or Critical Bands identified 24 numbered pass-bands with more accurate representations of the hearing sensitivities [13]. By design these bands were hereafter referred to as Bark Bands in memory of Bark-Hausen – the creator of the unit phon loudness level. This new system of bandwidths extended from 50 Hz up to and including a centre frequency of 13.5 kHz where the purpose of this improved system was a higher resolution for the future development of loudness-calculation procedures based on the Bark Band scale.

Glasberg and Moore (Equivalent Rectangular Bandwidths, ERBs)

Not all loudness theories accepted the Bark Bands as the most accurate representation. When approximating the critical bandwidths Moore and Glasberg adopted what has become known as the Equivalent Rectangular Bandwidth (ERB) model, a scale of bandwidths with a finer resolution than the Bark bands. In a manner similar to the Bark Bands numbers, ERB values designated frequency bands along the audible spectrum. The primary difference between the ERB model and Bark Bands was prominent in the lower frequency region; as shown in **Figure 2.5** below. Note: for frequencies below 1 kHz, the

classical critical-bandwidth function used for the Bark Bands appeared to have constant bandwidths of approximately 100 Hz while the ERBs continued to decrease in size. Moore and Glasberg attributed this difference to sufficient evidence that the bandwidth of auditory filters continued to decrease below 500 Hz unlike the traditional critical-band model suggested, see [14].

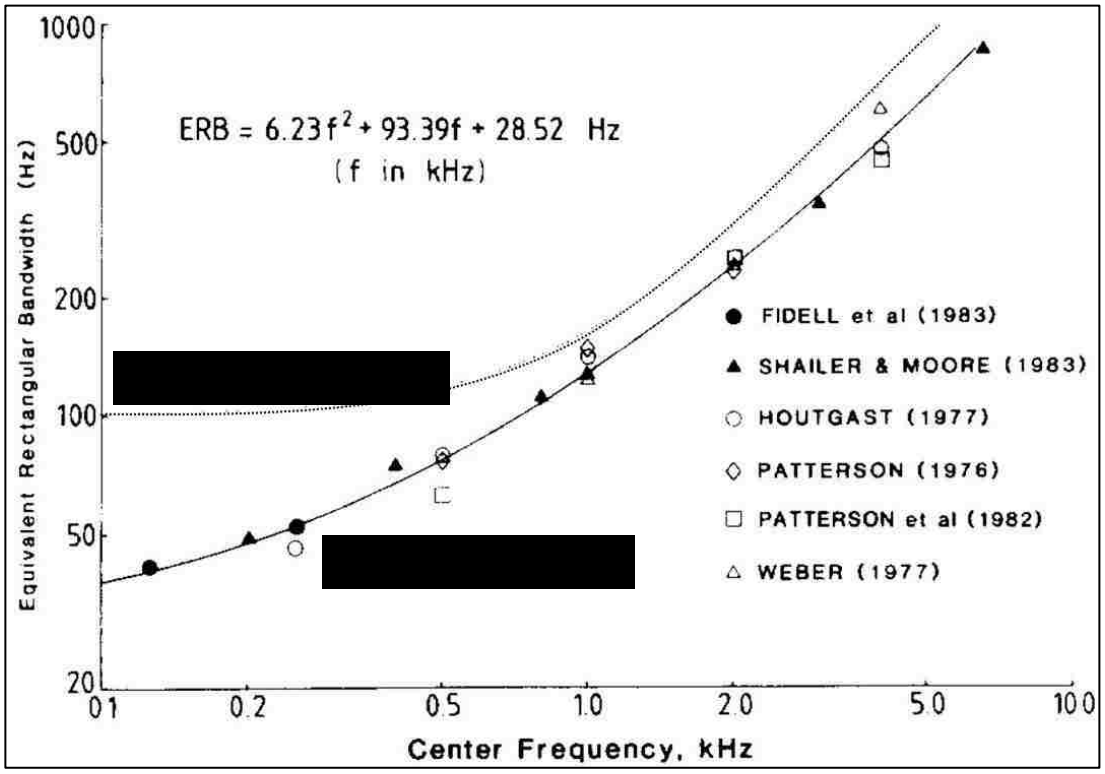


Figure 2.5– Comparison of auditory bandwidths as a function of frequency: solid line – equivalent rectangular bandwidth (ERB) function; dotted line – classical critical-band function (Bark bands), reproduced from [15].

Regardless of which band-pass model was used, it should be mentioned that the actual filter-shape was not 'ideal' as compared to a theoretical filter. In the ERB approximation the auditory filter-shape was assumed to have a rounded-exponential shape as suggested by Patterson et al. in 1982, [16]. A representation of this shape has been reproduced in **Figure 2.6** below where it was clearly shown that the slope of each skirt

varied with amplitude. As the level of the signal increased the lower frequency slope was observed to decrease more steadily. When viewed as a complete filter bank (not shown), this asymmetrical sloping suggested that frequency bands located several bandwidths above a specific pure tone would continue to perceive acoustical energy regardless of the separation. For frequency bands located at frequencies below the target tone, this off-band listening would only occur for adjacent bandwidths in close proximity as the upper auditory filter slope becomes steeper at higher amplitudes; the resulting impact of this concept has been summarized in **Section 2.2.3** that follows.

It was Moore and Glasberg's opinion that the ERB method of calculating the excitation pattern was more accurate based on their derivation approach of the notch-noise masker technique. While a complete comparison between Bark-Bands and the ERB was beyond the scope of this investigation, the finer resolution of the ERB scale and the supporting information indicated in **Figure 2.5** adds merit to this opinion.

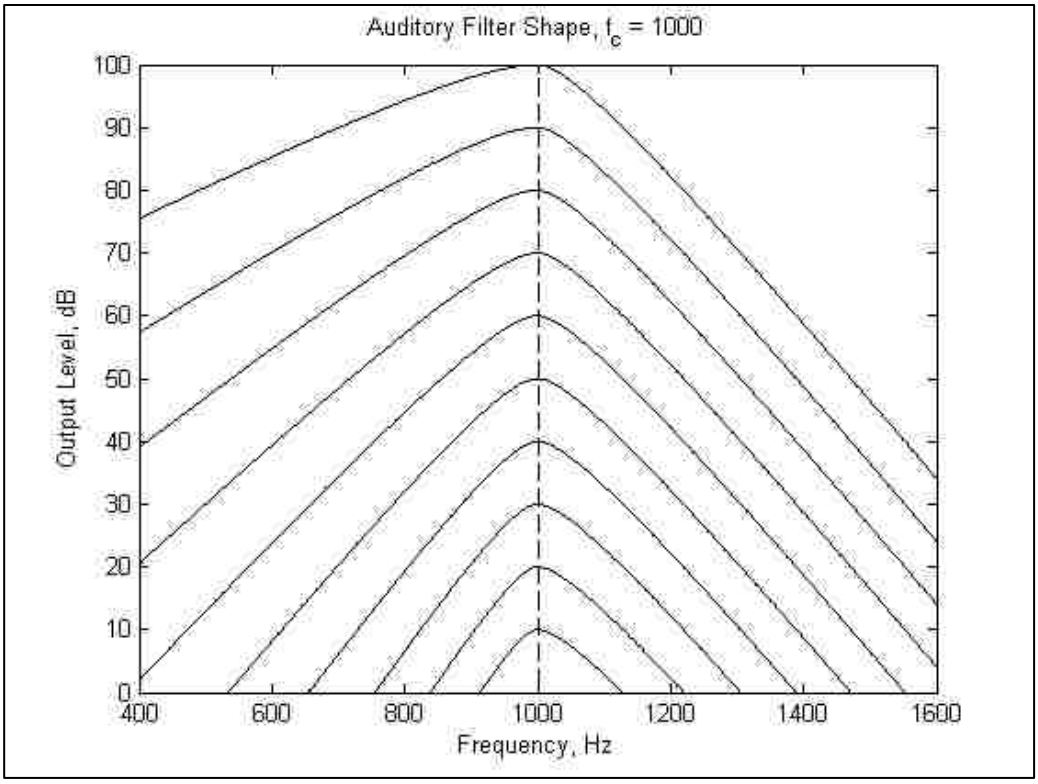


Figure 2.6 – Auditory filter shapes of the 1 kHz filter at various amplitudes.

It should be mentioned that while the shape of the auditory filters varied only markedly between normal hearing individuals, there was a considerable difference between filter shapes for individuals with varying levels of hearing impairment, [17]. Such variance was large even between subjects who exhibited similar absolute threshold trends. An investigation of hearing impaired individuals was beyond the scope of the present investigation, therefore, unless otherwise indicated all results presented hereafter are representative of normal hearing individuals. Even with such a restriction, the variability of auditory filter shape continued to increase with frequency and further with the increasing age of the listener, [16]. This information is intended only as an

introduction to these important concepts with regard to loudness. For more information, please refer to the supporting documentation indicated above.

2.2.3 The Role of Frequency Masking in Perception

The resulting output response from the each of the above auditory filter-banks created an excitation pattern specific to each signal. Due to the actual shape of the pass-band filters and the close proximity of adjacent bands there was significant overlap from each filter into adjacent bands. This phenomenon resulted in two important outcomes for the acoustician: (i) the physical measurement of any signal including pure tones would include energy traces in adjacent bands (as mentioned above); and (ii) the actual perception of this tone could mask or cause adjacent tones to become inaudible if they fell below a sufficient level of excitation. To demonstrate this latter point, consider **Figure 2.7** below reproduced from [18].

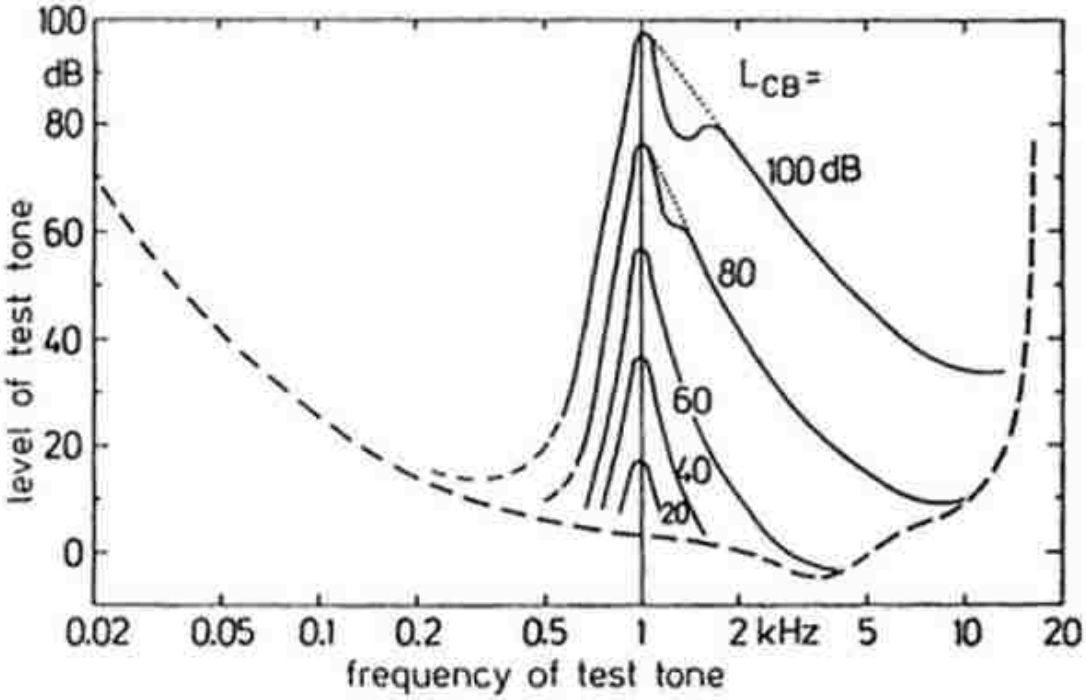


Figure 2.7 - Simultaneous masking of pure tones by a critical-band-wide noise with centre frequency of 1 kHz, reproduced from [18].

Figure 2.7 demonstrates the concept of simultaneous masking; when a 'masker' (in this case a band of white noise, one-critical-band in width and centred at 1 kHz), was introduced at a known amplitude and a masking or silencing effect was observed in the adjacent bands. The contours in the figure were derived using this phenomenon as each line represented pure tones of varying frequency which were 'just masked' as a result of this noise band. For each contour, it could be concluded that any tone presented below the recorded amplitude would be sufficiently masked by the 'masker' signal. Likewise any tone presented above this point would be audible to the observer. The resulting shape revealed a masking pattern which was more heavily applied towards higher frequencies than lower ones. This suggested that the auditory filter shape in the human ear was not

symmetrical, a concept which will be heavily relied upon later in even the simplest of loudness models, [1]. For more information regarding simultaneous masking, please refer to [18].

Only through a thorough understanding of each of the above concepts was it possible for researchers to produce a calculation model capable of predicting the perceived loudness of a signal. Each model was required to answer several questions of perception including:

1) Does the perceived loudness vary with frequency/amplitude?
    - ✓ Equal loudness contours confirmed this.
2) How does the human ear divide up this information as it is perceived?
    - ✓ Critical bandwidth functions provide this approximation.
3) How does one signal influence the perception of another?
    - ✓ Simultaneous masking confirms adjacent bands impact one another.

Using this information several loudness algorithms have been developed over the years which will be outlined below; for the interest of this document the following chapters solely focused on those models which have been standardized by international organizations. Emphasis was placed on the Glasberg and Moore loudness models for reasons indicated in **Section 2.6** which follows.

2.3 Stationary Loudness Metrics

Stationary or steady-state signals are those samples with characteristics which did not vary with respect to time. As they could often be expressed mathematically such signals were classified as deterministic and therefore easier to implement into most

computational models. It was no wonder that these signals were the focus of the first models for predicting loudness. Using the mechanics of hearing as outlined above, each of the following loudness metrics were applicable to steady-state sounds using averaged spectra from long-term measurements.

The first documented measure of loudness was developed by Steinberg in 1925, [19]. Without going into detail, it has been stated that the model agreed well with the known characteristics of hearing at the time. However, as more data became available this initial model of loudness was determined to be inadequate or in need of significant improvement, [1]. Fletcher and Munson revisited the concept in their 1933 paper where they suggested an improved model. Again, due to the complexities of loudness estimation even the authors recognized at that time that more work was necessary while referring to their results as not 'fully developed' in [1]. As additional models became available, standardization organizations began to consider which of the available theories should be used exclusively. The following subsections identify those models currently accepted along with a brief review of their associated histories.

2.3.0 A-Weighting (Informative Description)

While not strictly a loudness metric the A-weighting filter has become the most popular signal presentation method since its inception, (for reasons outlined in the Introduction). It was the first estimation of loudness and relied upon as the default measurement system for multiple municipal and provincial noise regulations. Because of its popularity, a summary of loudness metrics would have been incomplete without an honourable mention of this algorithm.

Originally derived from Fletcher and Munson's equal loudness contours, the A-weighting filter is an inverted approximation of the 40-phon contour (normalized to zero at 1 kHz). Additional weighting filters were created around the same time which mimicked alternative loudness contours including the 80-phon contour (B-weighting, signified by dB(B)), and the 100-phon contour (C-weighting, signified by dB(C)). Each function could be applied to any recorded signal in order to account for the mid-range sensitivity of the human hearing system. The A-weighted response is heavily attenuated in the lower frequency range with smaller corrections above 6 kHz. The intended application of the A-weighting filter was to represent environmental noise measurements to simulate how humans actually perceived the noise sample. The caveat being that each filter-bank was created as a means of adjusting only those signals recorded approximately around the same amplitude and spectral content as the respective contour. Critics of the approach were therefore quick to point out that the A-weighing filter has been incorrectly applied for years. For example, the A-weighting filter was only intended for mid-range SPLs between 20-55 dB, the B-weighting for SPLs between 55-85 dB and the C-weighting filter was intended for SPLs between 85-140 dB, [20]. Despite this Schomer et al. observed the A-weighting filter has been applied regardless of the signal amplitude, [21]. Given the fact that it was unlikely for an environmental noise source to have a spectral pattern similar to the 40-phon contour, most signals were being inappropriately attenuated because they poorly represented the low frequency noise contributions. It is clear therefore known errors are associated with using the A-weighting metric as a means of presenting loudness information. While an in-depth review of the A-weighting metric

was not included as part of this investigation, it was necessary to include it as a discussion point since A-weighting is currently a reoccurring concept in the study of loudness.

2.3.1 ISO 532

The first complete loudness model to be accepted as a standardized approach was adopted by the International Organization for Standardization (ISO) as ISO 532 "Acoustics – Method for Calculating Loudness Level." The approach, developed by S. Stevens in 1936, was on the forefront of loudness development including the proposal of a new measurement scale based on the unit sones; a set of loudness metrics used by all the standardized loudness models thereafter, [22, 23]. Stevens was further attributed with developing the power law for loudness which would later be included as part of the Mark VI loudness model. To implement the standard 1/1-octave band information was entered via equations in conjunction with coefficient lookup charts. The Mark VI was limited as it was only applicable for the diffuse-field listening environment; a reflective environment where acoustical energy is equal in all directions. Such a condition was not often found in environmental acoustics.

By 1975 there were several loudness models in development with no clear indication of one that was better than the rest. This lack of clarity resulted in the ISO 532 document being amended to include a second algorithm which had more flexibility than the first. To differentiate between two, Stevens' method, the lesser used of the two became and known as ISO 532A while the new popular method based on Zwicker's approach and became ISO 532B.

While Stevens' approach was limited to 1/1-octave results in a diffuse field, the Zwicker loudness model allowed for a greater resolution with 1/3-octave spectra and the option for two separate listening conditions; as either a frontal-incident sound source or as an observer within a diffuse field, [24]. One important difference between Zwicker's approach and the previous models was the inclusion of the Bark-band approximation outlined in **Section 2.2.2A** above. As the critical-bandwidths were wider than 1/3-octaves in the lower frequency region (i.e. below 280 Hz), multiple 1/3-octaves were logarithmically summed to form approximate critical-band levels into the following three separate groups: (i) all frequencies below 90 Hz; (ii) the three third-octave bands from 90 Hz up to 180 Hz; and (iii) the final two remaining 1/3-octave bands 180 Hz and 280 Hz. In this manner, the resulting data closely approximated the critical-band pattern as specified in [13].

In its original form the ISO 532B method was a graphical approach requiring the user to input critical-band data onto the included figures for either the free-field or diffuse-field listening conditions. Once all the band-levels were recorded on the appropriate figure, the influence of spectral masking was introduced via downward sloping curves whenever the level in the next highest Bark-band was lower than the previous. Otherwise vertical lines were used to connect data points as the levels increased between adjacent bands. After all the data was entered, the area enclosed beneath the resulting step-plot corresponded to the total loudness value in sones. Understandably, this form of the ISO 532B model was time-consuming to use as it required each data point to be drawn individually along with the masking pattern. With improvements in computational technology, this method became easier to use and the Zwicker loudness model would

eventually be accepted by the German standardization organization: Deutsches Institut für Normung (DIN) e.V. (German Institute for Standardization).

2.3.2 DIN 45631

Unfortunately, an English version of the DIN 45631 standard was unavailable at the time of this dissertation. In lieu of the actual standard, the following discussion was made possible from an article released by Zwicker et al. in 1991 outlining an updated procedure and the associated BASIC code for implementation, [25, 26].

It was confirmed in the updated draft that the ISO 532B document and the DIN 45631 standard were "largely identical" and produced nearly the same results, [26]. The update removed the tedious nature of the graphical method and allowed users to enter 1/3-octave spectral data directly into the computer program. The improvement sped up calculation times considerably while removing the potential for user error when calculating the overall loudness. Recall that the previous approach required the user to determine the area under the specific loudness contour; given the complex shape of contours involved, the area calculation often included minor approximations. It was important to note however that the computerized version still relied on tabulated values for calculating excitation patterns rather than mathematical functions, [14]. Such a limitation forced the program to interpolate between values which could possibly lead to some irregularities in the predicted results.

2.3.3 ANSI S3.4

The American National Standards Institute (ANSI) chose to adopt a different model for loudness called the "American National Standard Procedure for the

Computation of Loudness of Steady Sound" under the current Standard number ANSI S3.4:2007, [27]. Originally developed by Moore and Glasberg this loudness model has undergone several revisions as time has passed, [14]. The first iteration, based loosely on Zwicker's loudness model, included several modifications and extensions such as: (i) mimicking the amplitude trends found in the equal loudness contours; (ii) clearly defining the concepts of critical bandwidths; and (iii) including considerations for partial masking on loudness without correction factors, [14]. As the focus of the current investigation was an extension of their 2007 work, the workings of the ANSI standard were investigated in greater detail than the other standardized methods.

Implementation of the ANSI model allowed sound spectra to be specified using one of three options including pure tones (either single or multiple creating a complex signal), noise bands, or the more common 1/3-octave spectra. While the first two options provide reliable results, their application was limited as these signal types are not often found in practice. The final method allowed the user more flexibility through 1/3-octave spectral data to be taken from either fabricated or physical measurements. Such information was specified using 26 adjacent 1/3-octave bands from 50 Hz up to and including 16 kHz. The use of this option more closely mimicked the previous two standards as 1/3-octave information was readily available using current sound acquisition equipment. The remainder of this overview has been based on the 1/3-octave input option.

As spectral information was fed into the program a transfer function corrected the data based on the assumed listening conditions. This unique stage in the application allowed the user to convert any signal from the procedure by which it was recorded into an approximate spectrum as it would have arrived at the eardrum, regardless of the

listening condition. While the ISO 532B and DIN 45631 models were both capable of free-field and diffuse field conditions, the ANSI S3.4:2007 model incorporated a third listening condition for perceived loudness via headphones (or any user-specified correction). This addition allowed the user to account for the specific headphone Frequency-Response-Function (FRF) of the circum-aural or supra-aural headphones used in the signal presentation. It was not uncommon for psychoacoustic evaluations to use such headphones where various makes and models for each type are available. It was worth noting that even within one specific headphone design, the manufacturing process may produce a variety of FRF functions that must be measured in order to achieve the most accurate results for each pair used.

Once the listening condition was applied the spectra underwent another transfer-function imitating the transmission of the signal from the eardrum through the middle ear. A signal perceived through a human ear requires acoustical energy to be transferred from the eardrum to the cochlea where the waveform was transformed from an airborne signal to a fluid borne vibration with associated losses and gains. Therefore, this secondary transfer function accounted for the spectral changes and the resulting waveform replicated the signal as it would have entered the cochlea; the signal acquisition center of the human ear. At this point the frequency sensitivity of the basilar membrane assessed the signal as thousands of stereocilia were excited to convert the fluid borne vibrations into electrical signals in a manner similar to the auditory filter banks outlined above.

In the case of ANSI S3.4 the conversion was accomplished using the auditory filter shapes and ERB filter bank designations outlined in **Section 2.1.2**. Recall that the array of band-pass filters specified in the ERB network consisted of overlapping

passbands whose overall output was measured as excitation level per ERB (measured in decibels per ERB). The shape of each band-pass filter was the rounded-exponential function as indicated in **Figure 2.6** above. To maintain a high degree of spectral resolution the excitation level was calculated in 0.1 ERB-steps from ERB values of 1.8 through 38.9. Note that spectral information was provided by the user in 26 third-octave bands therefore interpolation was incorporated into the model based on a second-order polynomial fit and sets of three adjacent data points, [28].

Like the previous models, the excitation pattern would then be converted to a specific loudness contour before the overall loudness could be determined. For the ANSI standard, this was done at a much finer resolution than previous models as the loudness density was determined for each component in unit sones/ERB; the resolution at this point remained at 0.1 ERB intervals. The calculation of this value depended on the level of excitation at each ERB whether it was below the threshold of hearing, above the threshold of feeling (i.e. $E > 100$ dB), or at an intermediate level between the two. At this point, spectral masking was automatically incorporated due to the spectral shape of adjacent filters. The overall loudness was calculated by summing all the spectral components across the whole ERB-number scale. As greater accuracy was achieved by calculating the specific loudness at 0.1-ERB intervals, the resulting summation at this point would be divided by 10 to produce the final result. An additional step allowed the overall loudness (in sones) to be converted to an overall loudness level in unit phons as done in the previous models; the phon scale was often easier to comprehend by most users than was the sone scale.

The entire ANSI loudness model was available as a standalone executable file included with the standard. An approximation of loudness levels using this standard was easily achieved so long as the user implemented the correct listening conditions for the observer and provided the measured 1/3-octave spectrum; a procedure similar to the DIN 45631 approach. Unfortunately, each of the above stationary loudness models were extremely limited in their application as a truly steady-state signal is rarely observed in most product evaluation set-ups and even less-so in outdoor monitoring situations. To extend beyond these limitations each of the above models had to be modified to include a model reflective of the more complex auditory mechanisms involved in the measure of the time-varying loudness of real-world acoustics.

2.4 Temporal Loudness Mechanisms

Per ISO 2204:1973, three main categories of non-stationary signals have been identified: continuous fluctuating noise (such as music and speech), intermittent noise, and impulsive noise, [4]. A signal that varied with respect to time was considerably more difficult to analyze given the sensitivity characteristics and short-term memory of the human ear. During assessments, careful attention would therefore be given to the length of the signal and the trends present in the temporal variations.

A general assumption was made that at any particular instant the perceived loudness could be approximated by assuming the signal was made up of a series of short time segments, each with a steady-state characteristic. The length of such segments can be approximated by the time required for a sufficiently short signal to reach a steady state loudness level. This time may vary between loudness theories where the Glasberg and Moore model reaches the peak instantaneous loudness after 100 ms., [2]. This was a

necessary assumption as the processed data must be sufficiently long enough to represent the spectral components of audible wavelengths while retaining the required information. Overlapping time segments were often used to achieve this where loudness could be calculated at high temporal resolutions, resulting in an instantaneous value for each moment in time. In order to represent the loudness of a longer signal, an averaging mechanism must often be employed to smooth out the instantaneous fluctuations; one such method has been summarized in the following sub-section.

2.4.1 Temporal Summation/Averaging

It was observed in literature that as temporal signals entered the ear, the auditory system would continuously integrate the response while it extracted the important information, [29]. The initial sensation appeared to overshoot the actual amplitude before the perceived loudness stabilized. This resulted in a 'popping' sensation when the audible signal was initially turned on. Once the signal was turned off again, another phenomenon occurred where an 'after-effect' caused the loudness to linger as a decaying function. This theory was summarized in the following **Figure 2.8** by Namba et al, [29].

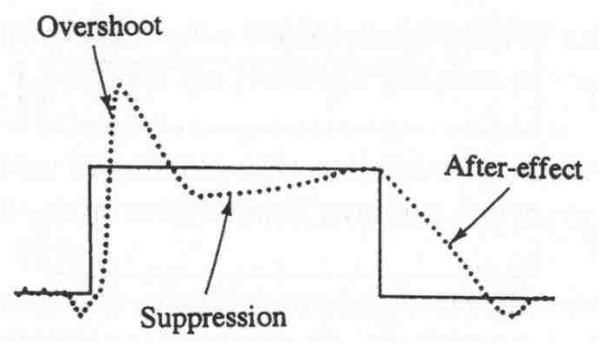


Figure 2.8 – Temporal response for a single steady-state signal.
Originally from [30] and reproduced in [29].

It became clear that the measured response of the signal was markedly different than the one actually perceived. The time required for the initial perception was considerably faster than the decay time or the time required for the perception to stop. The initial response was not instantaneous since the ear took some time to 'build-up' to the actual amplitude of the signal. To demonstrate this, consider a temporally short signal such as an 80 ms noise burst. Assuming the popping sensation was mitigated via the ramp-up function the perceived loudness of the signal would increase with increased signal duration until the length of the burst reached a "critical-duration," [31]. Once the critical length had been reached the perceived loudness level appeared to stabilize. To demonstrate this further, consider a noise burst greater than 100-200 ms. in length where it could be demonstrated that the perceived loudness of the burst remained constant regardless of the signal duration. Below this value the loudness appeared to decrease with duration by approximately 10 phons per ten-fold decrease signal in length, see [2], [31], [32] and [33]. Therefore, two rules have been identified for short signal presentation: (i) to avoid a perceived overshooting of the loudness a ramp-up function must be used to remove unwanted popping; and (ii) to achieve the full perception of a signal, the sample must be present for a sufficient length of time.

In a manner similar to how the ear needed time to adjust to the new signal, the auditory system required even longer to relax after it had been excited. This phenomenon was demonstrated previously by the 'after-effect' present in **Figure 2.8** above. Loudness models were designed to account for this decay time using temporal averaging which steadily decreased once the signal had stopped.

2.4.2 Temporal Pre-Masking

Aside from the prolonged sensation, an additional by-product of the 'after-effect' was a masking of any signal whose amplitude was less than the decaying function at that time. This phenomenon became known as post-masking with regard to temporal loudness and could be demonstrated via **Figure 2.9** below. Note there were three phases of temporal masking which could readily be identified as (i) pre-masking; (ii) simultaneous masking, like those identified in **Section 2.1.3**; and lastly (iii) post-masking because of the temporal decay function or time to relax as identified in **Section 2.4.2**.

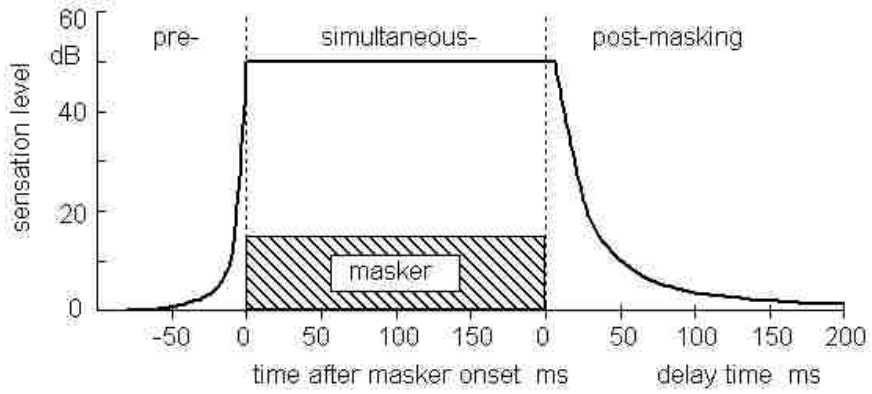


Figure 2.9 – Temporal masking phases in the presence of a masker signal, [18].

Pre-masking was a difficult concept to measure, model, and understand. The initial loudness of a signal could be of such an amplitude that it would overshadow or reduce the loudness of a signal already present. The duration of this phenomenon, as indicated by **Figure 2.8,** was relatively short and would last only 20 ms. before the masking signal began. The concept of pre-masking becomes easier to understand by recalling that the perceived level of a signal is not instantaneous and that it takes time for the auditory

system to adjust to the amplitude of the presentation. With this understanding one could assume there would be a faster build-up time for louder signals than those which are quieter in comparison. It is clear how pre-masking would occur provided the second signal in the series was louder than the first and the start-times were within milliseconds of one another, [18]. As pre-masking is difficult to observe even when targeting it (the perception of the phenomenon requires specialized training, [34]), it has often been neglected from temporal loudness models.

2.4.3 Summarizing Overall Loudness

Each of the above mechanisms were implemented into the numerous time-varying loudness metrics in order to expand on their stationary predecessors. Using these techniques, the perceived loudness at each instance in time was approximated regardless of how the signal varied; this often resulted in large fluctuations in predicted loudness levels. When reporting this information, it was useful to summarize an entire signal's worth of loudness values using an overall loudness level. As with most subjective quantities there was still significant debate as to how this summarization should occur.

It was well understood that instantaneous loudness was more of a placeholder than an actual, perceived value. Often calculated at a resolution of 1 ms. the instantaneous loudness changed faster than the auditory system could react to it. In order to account for this the various loudness models summarized the overall loudness via statistical results such as percentiles or mean values which were useful for reporting when signals were quasi-stationary or did not change drastically with respect to time. The accuracy of the final value would then be confirmed by either standard deviations or interquartile ranges

where larger deviations would yield higher levels of uncertainty in the resultant overall loudness value, [29].

Recent approaches applied more weight to the temporal summation mechanisms and instead, relied more on the time-averaging of the results to account for the temporal loudness decay, to visualize this effect refer to **Figure 2.10** below. Regardless of which approximation was used, it was clear that quantifying non-steady state signals with a single number was a difficult task and was handled differently by each non-stationary loudness metric.

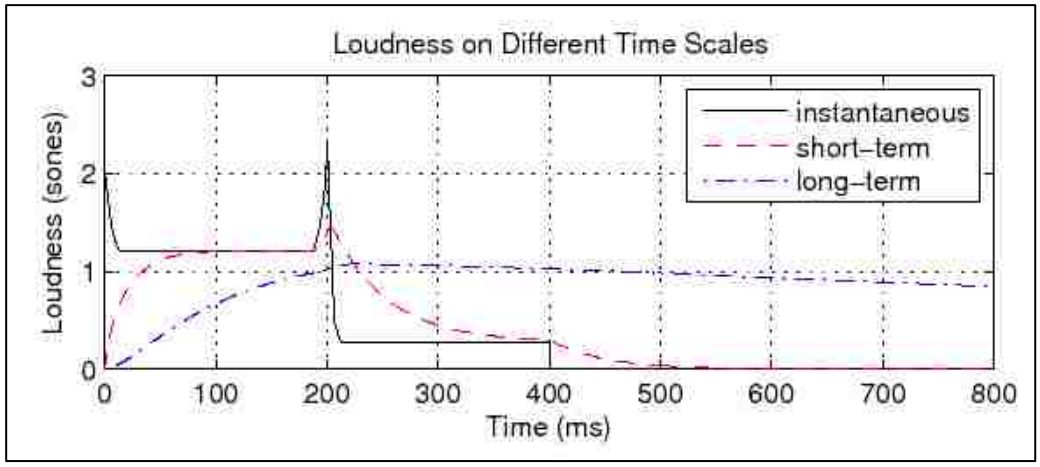


Figure 2.10 – Influence of temporal loudness averaging, [35].

2.5 Non-Stationary Loudness Metrics

The calculation models developed for time-varying loudness predictions were predominantly based off the existing stationary loudness metrics. For the standardized metrics outlined above two such models were established: the German standard DIN

45631/A1 – an Amendment to the stationary model; and the Glasberg and Moore Time-Varying Loudness (TVL) model - currently being considered for standardization.

2.5.1 DIN 45631/A1

At the time of this investigation, a working version of the DIN 45631/A1 algorithm was available as part of an acoustical analysis suite (PULSE Reflex Ver. 19.0). Unfortunately, a translated hardcopy of the standard was not available, only a draft version as referenced in [36]. The following information was extracted from the draft document including a block diagram of the model reproduced as **Figure 2.11** below. However, much like the DIN 45631 stationary model, the language barrier forced several assumptions to be made regarding the specific details and fundamental concepts which remained unclear.

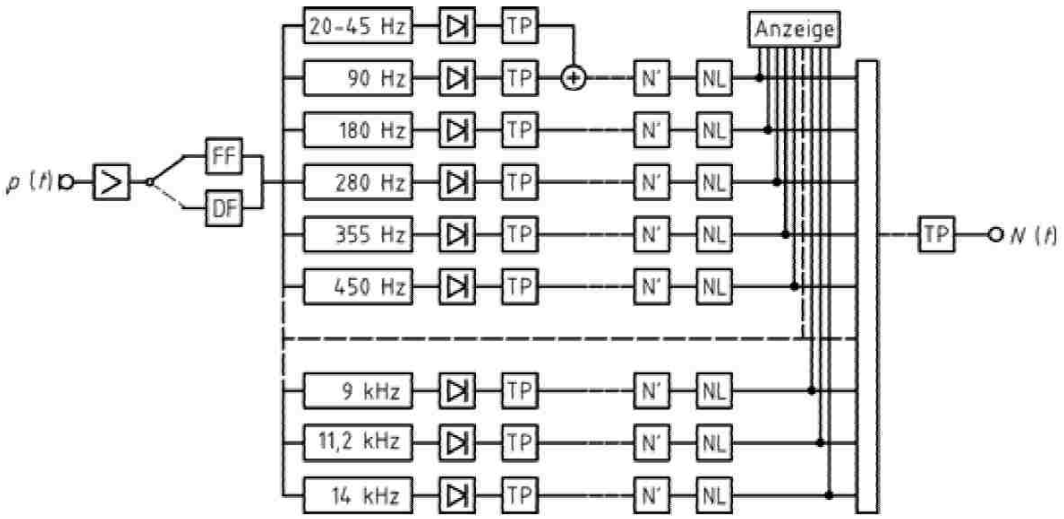


Figure 2.11 – Block diagram of the standardized DIN 45631/A1 time-varying loudness model, [36].

Much like the stationary predecessor, the model required the user to specify either a free-field or diffuse-field environment. The signal was then transferred into the

frequency domain as the bank of Bark-band filter bank was applied from 20 Hz up to and including 14 kHz. To ensure accurate approximations, the Bark-bands were accompanied by rectifiers and low-pass filters (labelled 'TP' in **Figure 2.11**), which filtered the data to remove unwanted noise. The amplitude in each Bark-band was next converted into a specific loudness value (N') where the resulting data-set was a specific loudness pattern measured in units sones per Bark. To this point, the model had closely followed the stationary loudness metric where a summation of the area under the specific loudness contour would provide an estimate of instantaneous loudness. To include a temporal influence, the authors applied a non-linear temporal decay (NL) which varied in direct relation to the duration of the signal; in this model the slope of the loudness decay was steeper for short signals than longer ones as would be expected from the discussion above. Lastly, the overall loudness of the sample (N) was approximated as the area formation under the specific loudness pattern.

In order to summarize the overall loudness level, the DIN 45631/A1 standard accounted for temporal variability using the percentile loudness scale where the N_5 percentile (a 'peak' loudness level which was exceeded during at least 5% of the recorded duration), was to be reported for the overall loudness perceived, [36]. Other percentiles could be recorded at the user's discretion. A low-pass filter applied at the end of the signal accounted for the temporal summation of shorter signals (i.e. durations less than 100 ms.), being perceived quieter than longer signals.

2.5.2 Time-Varying Loudness Model

As the Time-Varying Loudness (TVL) model was the primary focus of this investigation, the inner workings have been explained in more detail than the previous

methods. Glasberg and Moore introduced this approach in 2002 as an expansion of their stationary loudness model (described previously in **Section 2.3.3** above), [2, 14]. The authors sought to develop a model which removed the limitations of their previous metric while targeting known temporal phenomena such as amplitude modulation and the influence of temporal length.

The literature suggested that when a signal was perceived to fluctuate with respect to time (as was often the case for complex tones and modulating signals), the perceived loudness corresponded to some level between the Root-Mean-Squared (RMS) of the resultant loudness and the peak loudness level observed, see [2], [18], [37], [38], [39], [40], [41], [42] and [43]. Additionally, as demonstrated in **Section 2.4.1** the duration of the noise source had a strong influence on how loud the sample was perceived regardless of whether or not the intensity of the signal would be held constant, see [2], [31], [32], and [33]. Glasberg and Moore focused their improved approach on these features to develop a model capable of predicting the perceived loudness of brief sounds as a function of duration as well as an overall loudness estimate of sounds with strong temporal characteristics and modulation.

In order to understand the models operation, a brief summary of the individual steps has been introduced here with a visual flow chart on the following page. Specific details of the more complicated aspects have been described following the flow chart to completely describe this powerful algorithm. The standard calculation procedure as summarized from [2] is as follows:

(i) The user was required to upload a 16-bit recorded *.WAV file into the program with a calibration factor ensuring the desired amplitude;

(ii) Like the DIN model a ‘Finite Impulse Response (FIR) filter’ was applied to mimic the transfer function from the recorded WAV file to the middle ear, (described in further detail below). The resultant filtered response presented the signal as it would appear had the measured data been free to propagate from the recorded conditions to the middle ear of an average listener;

(iii) To derive the spectral components at each frequency of interest the filtered signal was fed into a series of six FFT functions. This and the following two steps were completed at one millisecond intervals for a high temporal resolution;

(iv) The ‘Physical Spectrum’ generated from the FFTs was then converted to an ‘Excitation Pattern’ using the ERB-bandwidth filters as mentioned in **Section 2.2.2 B**;

(v) With the addition of the frequency masking characteristics the excitation pattern was then converted into an instantaneous ‘Specific Loudness’ contour.

(vi) The area under the resultant contour provided an estimate of the ‘Instantaneous Loudness’ at that point in time and a resolution of 1 ms time segments;

(vii) ‘Short Term Loudness’ was achieved via the averaging mechanism. Such a function mimicked an automatic gain control system with attack and release times to calculate a short-term loudness value;

(viii) Finally, a second averaging mechanism with longer attack and release times was employed to produce the final overall loudness level of long-term or ‘Overall Loudness,’ [44].

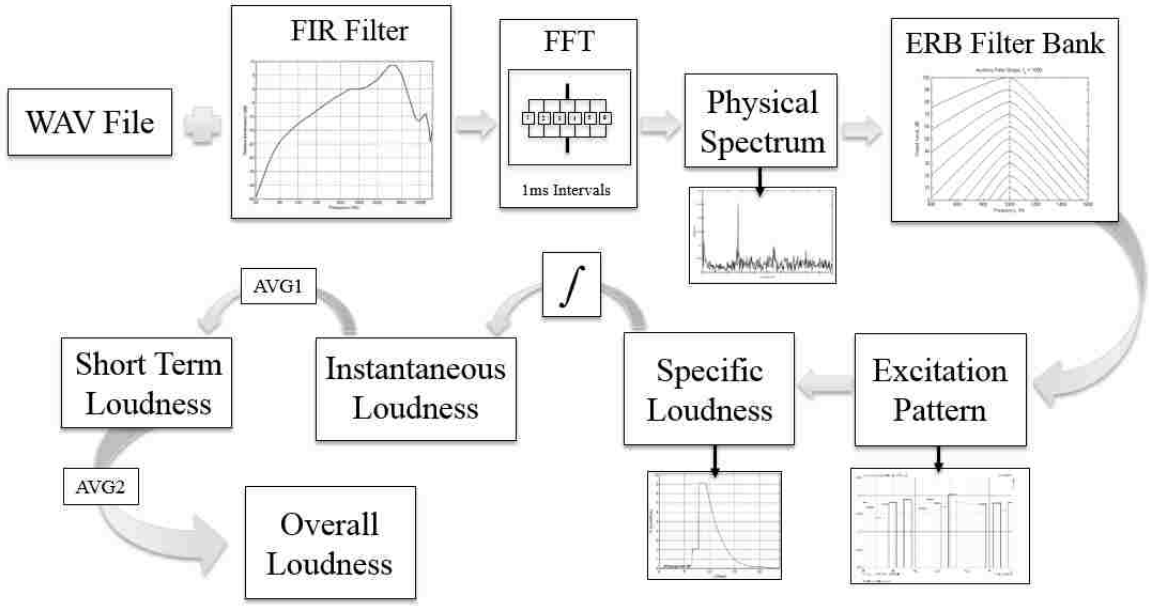


Figure 2.12 – Time-varying loudness model flowchart.

Like the stationary predecessor ANSI S3.4, the appropriate FIR Filter was required to be carefully chosen by the user to account for the transfer from the measurement location to the listening conditions of the observer. As the signal was often recorded via microphone in the absence of a listener, the FIR filter approximated the influence of the head and torso on the signal. Three standard options have been provided with the model including a free-field condition, a diffuse-field condition, and a flat response which may be appropriate for listening via laboratory grade headphones (such as the Etymōtic Research ER2 insert headphones which were specifically designed to have a flat response at the eardrum). With the flat response, the model would not apply any head related torso function as it would assume that the signal would already account for such influences. This scenario was applicable where the signal had been recorded by the HATS acoustical listening device which simulated an otological normal hearing

individual. In most cases, it was best to measure the true response of the equipment at the eardrum. Regardless of which method was applied, the output of the FIR filter could be considered as the best approximation of the signal as it reached the inner ear, [2].

To produce the most accurate results, the authors employed six FFTs operating simultaneously in parallel while maintaining identical temporal centres and overlapping extents. The individual FFT functions varied in both frequency bandwidth and segment duration as visualized in **Figure 2.12** below. Doing so maintained a high degree of signal resolution in both the upper and lower frequencies (as lower frequencies required longer wavelengths), while reducing unwanted errors. To achieve this resolution, a limitation was placed on the model as acoustic signals were required to have a minimum length of 64 ms. to account for low frequency wavelengths. The resulting output from this process was the physical spectrum of the signal calculated at 1 ms. intervals.

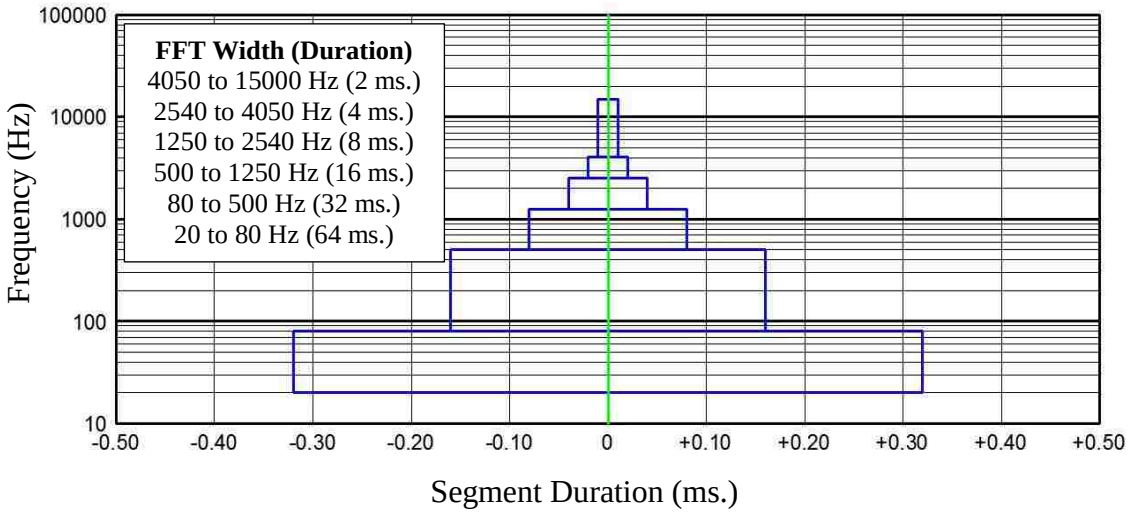


Figure 2.13 – Temporal widths and spectral ranges of the six FFT algorithms used in the time varying loudness model. Legend indicates the frequency bandwidth and segment duration of each FFT where the durations are centred about a common temporal center.

At this point, the model operated similar to their stationary metric where the physical spectrum was converted into an excitation pattern and again into a specific loudness contour at 0.1 ERB intervals. Instead of the overall loudness however, the value calculated as the area under the curve was what the authors called "instantaneous loudness." Unfortunately, the human auditory system is not able to consciously perceive fluctuations in 1 ms. intervals and this value became more of a placeholder or intervening variable for future calculations as outlined in **Section 2.4.3**. The actual perception of loudness may be thought of as a running average or temporal integration of this instantaneous response; in the TVL model the authors presented such averages as either a short term or a long-term loudness value.

Short term loudness and long term loudness are best defined through the understanding of speech. A listener may interpret a sentence either as a whole or broken down into individual components. In this case the, short term-loudness referred to the loudness of the individual syllables while the long-term loudness would be an overall estimate of the intensity for the sentence. While calculating these two values the authors assumed a temporal averaging behaving in a like an Automatic Gain Control (AGC) circuit as indicated by **Equation (2)** and **Equation (3)** below. Such circuits included an attack or release time depending on how the signal fluctuates.

A short-term estimate of loudness (S'_n), at each instant (n), was calculated via **Equation 2** as the temporal average of the instantaneous loudness (S_n) and compared against the previous instantaneous estimate (S'_{n-1}). The constant (α_a) was dependent on whether the gain control was in a state of attack (where $S_n > S'_{n-1}$) or a state of release ($S_n \leq S'_{n-1}$). In other words, the running estimate of short term loudness depends on

how the instantaneous loudness of each frame compares to the short-term loudness of the previous frame, (each successive frame is an increase of 1 ms. in this case). Constants for attack and release times were set to 0.045 and 0.02 respectively as a means of accounting for the variations of loudness with duration while maintaining reasonable predictions for amplitude modulating signals. Short term loudness has been found to increase relatively quickly with a somewhat longer decay time which is indicative of forward masking (as seen by the constants used in the equation below), [2].

$$S'_n = \alpha_a S_n + (1 - \alpha_a) S'_{n-1} \qquad \textbf{... (2)}$$

Long term loudness (S''_n) followed a similar calculation procedure as demonstrated by **Equation 3.** Again, the constant (α_{al}) was dependent on whether the gain control was in a state of attack (where $S'_n > S''_{n-1}$) or in a state of release ($S'_n \leq S''_{n-1}$) with constants of 0.01 and 0.0005 respectively. It was clear that the attack constant is greater than the release constant indicating that the long-term loudness may increase quickly and decay more slowly.

$$S''_n = \alpha_{al} S'_n + (1 - \alpha_{al}) S''_{n-1} \qquad \textbf{... (3)}$$

Depending on the type of signal being analyzed it is up to the user to interpret the short term and long term loudness values as necessary. For convenience, the minimum and maximum values are output by the program along with a running calculation of both. An overall estimate for the signal is provided via an average of the results.

2.6 The Best Performing Model and Intentions of Work

While conducting work on a Masters in Engineering, the author of this investigation compared all of the standardized loudness metrics above with respect to the updated equal loudness contours ISO 226:2003, [45]. The hypothesis stated that as the equal loudness contours represent the current map of sensitivity for the average human listener, it would seem sensible that each loudness model would mimic the updated contours as the loudness is varied across the frequency spectrum.

Results of the study confirmed that the ANSI S3.4:2007 (and by extension the TVL model), proved to be the best performing metric with respect to the updated equal loudness contours; for more information, as to the details of the experiment please refer to [45]. Using the previous investigation as a base the current study focused on improving the Time-Varying Loudness model, specifically improving how the model approximates binaural perception.

The following chapter summarizes the exhaustive literature survey conducted regarding the existing binaural summation theories. It will be shown that several theories exist between both the standardized loudness metrics and the associated literature which they are based on regarding how binaural summation is applied in industry. The results from the review and intentions of the author are outlined in the following chapter.

III. LITERATURE SURVEY

Loudness by its very definition is a subjective psychoacoustic quality and there are several supporting theories for how the internal mechanisms operate and should be accounted for. The concept of binaural summation is no different as multiple theories exist for how it should be applied and specifically the amount of summation which is necessary. A review of the available literature has been included below to provide a comparison between the current results and those found previously. The intention of this comparison was to summarize all preceding investigations and to confirm that the previous investigations support the current experimental procedures.

3.1 Binaural Loudness

To this point, each of the standardized loudness models has been described with respect to a monaural loudness level. Monaural in this case refers to listening with only one ear as opposed to 'binaural' meaning both ears. In the current standardized loudness models, the incorporation of binaural loudness appears almost as an afterthought. With a quick paragraph or single line, most standards include a simple binaural approximation using a correction function applied to the monaural estimate. Other metrics have made no mention of binaural loudness at all and therefore assume that a monaural loudness value is an adequate approximation. This standardized practice still exists even though it is a well-known fact that listening with two ears as opposed to one stimulates a greater perceived loudness level for nearly all signals types. A concept which can quickly be demonstrated by inhibiting one ear and comparing the perceived intensity of any source against the uninhibited listening condition.

Most loudness metrics were designed to utilize a single-channel sound spectrum which closely mimicked monaural listening. It was reasonable therefore to approximate a binaural listening condition by applying a correction factor based on the single-channel results. Over time, four separate theories have been adopted to achieve this approximation: (i) assume that there was no binaural summation and the monaurally calculated value was an ideal representation regardless of listening conditions; (ii) assume the binaural summation was simply the average of the two ears, for a full-frontal incident signal this equated to a value similar to (i) where there was no summation taking place; (iii) represent the summation mechanism as a constant increase over the monaural condition, for instance a common approach increased the monaural levels by 3 dB as indicated by a standard acoustical energy doubling; and (iv) assume a binaural signal would be perceived as a doubling of loudness compared to a monaural signal, in this case the calculated values would simply be doubled for binaural reporting. Each of these theories had their own supporters and thus considerable variability became apparent across studies and in the standardized metrics.

3.1.1 Standardized Binaural Methods

Individual authors cannot agree on one binaural summation mechanism and this dispute carried into the various standardized loudness metrics, adding further to the confusion on this topic. A quick review of the metrics has been compiled here to demonstrate:

ISO 532A (Stevens' Method) – The initial ISO model was intended for 1/1-octave band information to be recorded from a single channel. As the approach was applicable only for a diffuse-field it was assumed that the signal reached the listener's ears equally

from all directions and that a single recording was representative of both ears, [24]. Calculations were then completed via formulae or charts with no mention of whether the resultant data predicts monaural or binaural loudness. While it was difficult to identify which summation method this model uses or the underlying assumptions, it was reasonable to assume that theory (i) was adopted in this instance given the diffuse-field restrictions.

ISO 532B (Zwicker's Method) – The second component of the ISO 532 document is suitable for 1/3-octave information in either diffuse-field or free-field listening conditions. Like Method A, this approach only accepted single channel data representative of both ears. Calculations were available via spectral charts with no mention of whether the predicted loudness was a monaural or binaural output. Again, the summation method was unclear.

DIN 45631 (Modified/Improved Zwicker Method) – As the DIN 45631 document was unavailable, the workings of this model are unknown. It has been assumed that this model is patterned from the ISO 532B model on which it was based (see above) or from the DIN 45631/A1 extension summarized below.

ANSI S3.4:2007 (Glasberg and Moore Approach) – The ANSI model allowed users to enter 1/3-octave information from a single channel in a variety of listening conditions. The model clearly recognized that predictions were based on monaural loudness as a default and predicted the overall loudness (in sones) from one ear at a time. In order to approximate binaural loudness, the model instructs the user to simply add the overall loudness (in sones) from both the left and right channels. For a diotic presentation

this results in a doubling of loudness from each ear separately as in theory (iv) above, (i.e. twice as loud).

DIN 45631/A1 (Improved Zwicker Method Addendum) – The DIN non-stationary model was capable of post processing temporal signals recorded as WAV files in third party post processing packages. Within the draft version of the standard the case was identified where a binaural head was to be used for measurements. In that instance the results from both ears were to be reported and a representative single-digit value taken as the maximum of both channels. By deduction the DIN model assumed that there was no binaural summation regardless of how many channels were included in the prediction. Therefore, the monaural loudness will equal the binaural loudness in all cases as in theory (i) above.

TVL Model (Glasberg and Moore TVL Model) – As part of the TVL model 16-bit WAV files may be inserted directly for loudness predictions as the user specifies the listening conditions. Literature confirms that; while the model initially predicted the monaural loudness of a signal, a diotic listening condition had been included. Like the ANSI procedure the binaural assumption instructs the instantaneous loudness to be summed across both ears to give an overall instantaneous loudness value. Short term and long term binaural loudness may then be calculated via the temporal averages identified above. The loudness summation only took place at the instantaneous loudness phase and was based on an overall summation from a doubling of loudness as in theory (iv) above. This is a conservative approach which may overestimate the loudness level perceived.

From comparison of the standardized approaches it was clear that the various parties did not agree on one binaural summation mechanism. This confusion was expanded upon in published literature where the number of theories and supporting institutions grew significantly. Presented below are a few of the techniques employed that derived this complex mechanism of loudness; providing requisite background knowledge in order to review these theories.

3.1.2 Listening Conditions and Jury Test Methods

The only true method of quantifying the human response to a signal is through a complete jury test experiment involving a large test subjects population which exhibits a great deal of diversity. In terms of measuring the binaural summation phenomenon, it was apparent from literature that a variety of test procedures have been used which all targeted the same phenomena. With several methods available and conflicting results, it became difficult to determine which test procedure was more appropriate and why. The most popular test methods utilized in the past include: (i) loudness scaling; (ii) magnitude estimation; (iii) magnitude production; (iv) psychological magnitude balance (a combination of 'ii', and 'iii'); (v) forced choice paired-comparison techniques; (vi) cross-modality matching; (vii) one-vs-two-ear ratio production; and (viii) monaural-binaural loudness matching. The first four methods were focused on determining the loudness function of either monaural listening or binaural listening individually while the remaining four compared two separate presentations. Each of these methods exhibited their own strengths and weaknesses however the results from each may vary markedly. An understanding of each method is crucial before comparisons can be made as the way

signals were presented to the listener may significantly impact the perception and by extension the associated results.

i. Loudness Scaling

While this form of loudness estimation is not a matching technique directly, the results obtained from loudness scaling may loosely relate back to binaural summation when results are compared. During this task subjects were asked to assign signals into various categories, one example of groupings included labels of intensity which ranged from "inaudible" to "very loud." Results provided insight for the experimenter as to how the loudness of a sample varied as a function of the perceived intensity, [46]. As only one signal is 'scaled' at a time the loudness function related only to monaural or binaural loudness; not binaural summation directly. Critics of this approach have observed that results between studies have often been considerably different and large variance has been found between listeners, see [47]. To limit the amount of variance between studies, several attempts have been made to reduce bias of results and even standardize the procedures, [46, 48]. While this method has proved useful for diagnostic purposes and hearing aid fitting, extracting a binaural summation function from these results has been a difficult task, often resulting in only an approximate trend.

ii. Magnitude Estimation

Magnitude estimation is an open-ended rating scale where subjects are instructed to assign an arbitrary loudness level to each presented signal. Using this value as a reference, each successive signal is then assigned an appropriate value which is proportional to the first; often subjects are instructed to assign an unperceivable signal a ranking of zero with subsequent ratings increasing accordingly.

A major set-back of this form of matching is the tendency of subjects to constrict or limit the range of their estimates based on previous selections. One method of avoiding this is to instruct the participants on the number of test signals used in the experiment and give examples of the extreme options. This method has been used by several authors in the past including Marks in [49] while developing binaural summation techniques. However, Hellman and Zwislocki identified several counter-points to this practice: (i) a reference standard may be considered inappropriate by some subjects who in turn will substitute their own more convenient number; or (ii) in agreement with previous authors concluding that the designated reference tends to introduce an unnecessary bias toward a certain response, [50]. Instead, the authors explored a magnitude estimation procedure without a reference value where the comparison between the two results produced a strong correlation.

Further, Stevens and Greenbaum identified a particular bias with regard to magnitude estimation. During their experiments, the authors identified a form of "regression" where subjects appeared to limit the range of magnitude estimates based on a personal numerical response bias, [51].

iii. Magnitude Production

Like magnitude estimation, magnitude production required the subject to determine loudness based on an arbitrary scaling system. During magnitude production, the participant is given a series of numbers in random order which represent a target loudness intensity. He or she must then adjust the amplitude of the given signal to match this value. In many ways, magnitude production is an inverse procedure to that of magnitude estimation. It has been identified that the method of magnitude production

often produced steeper loudness functions than the method of magnitude estimation, [52]. Care must be taken while using this approach; this method of deriving the loudness function is subject to a known bias from the properties of the dial for amplitude adjustments. Results presented by Joshi and Jesteadt confirm that the resulting functions were strongly dependant on the sequential settings used for the decibel attenuator or "sone potentiometer" when stepped increments are used, [53] (see also [54]).

iv. Psychological Magnitude Balance

After recognizing the bias associated with the magnitude estimation and magnitude production approaches, Hellman and Zwislocki adjusted their procedure to account for both procedures at once, [50]. Their new approach required subjects to investigate each loudness match at least two times which ensured the subject completed both the magnitude estimation and magnitude production scenarios. To further remove bias associated with the magnitude production technique, it was recommended that the magnitude estimation approach should always occur prior to the magnitude production phase. Once both sets of data were collected, the average of the results was concluded with the opinion that an average of the two procedures is less likely to contain bias than the results of either procedure alone.

v. Paired-Comparison Techniques

The paired-comparison test is one of the simpler methods of matching loudness between either monaural-binaural pairs or similar signals of different amplitudes. Subjects are presented an alternating pair of signals and are asked to judge which of the two combinations sound louder, see Levelt et al. [55]. Amplitudes are then adjusted and the test is repeated in order to 'bracket-in' on a loudness match. This method may prove time

consuming as each match only revealed if a reference is lower or higher than the target tone; where the procedure must be carefully defined to avoid unnecessary testing. Aside from the time commitment, the paired-comparison technique has proved useful to isolate specific trends in hearing.

vi. Cross-Modality Matching

In this form of comparison, the perception of two different senses are compared in order to develop a relationship between the two. One example of this was performed by Reynolds and Stevens who required participants to match the apparent intensity of a band of noise presented binaurally with that the apparent intensity of a 60-Hz vibration signal applied to the finger tip of a participant, [52]. While this method did not directly lead to a binaural summation function it was worth identifying as a manner of comparing loudness to a reference level. Additional forms of this approach included adjusting the brightness of a light or length of a string/line to relate to the loudness intensity; a method useful for obtaining the response from young children, [56, 57].

vii. One-Versus-Two-Ear Ratio Production

In this form of comparison, experimenters were taking advantage of the fact that listening with two ears has often been associated with a signal being perceived as twice as loud. With this idea in mind, the ratio production tasked subjects to either (i) adjust a binaural signal until it appeared twice as loud as a fixed monaural signal; or (ii) adjust a monaural signal until it appeared as one-half as loud as a fixed binaural signal. This method took advantage of the fact that when compared to a reference signal there were two ratios which were considered reasonably easy for a participant to judge: a signal which is half as loud, or twice as loud (in terms of the ratios this resulted in targeting the

2:1 and 1:2 comparison values). It is interesting to note that this method was used to confirm the starting values of the equal loudness contours. Through extensive experimentation, researchers were able to confirm that a 1 kHz signal which was 10 dB above a reference level was perceived to be twice as loud, [18]. This observation lead acousticians to adopt the sone system where each doubling of sone value was a doubling of loudness; hence relative to 40 dB (40 phon and 1 sone), a 50 dB tone was perceived as twice as loud (50 phon and 2 sones), and a 60 dB tone (60 phon and 4 sones) was perceived to be four times as loud. In terms of binaural summation this method of loudness comparison is less common in literature but has been used in the past by experimenters such as Reynolds and Stevens, [52].

viii. Monaural-Binaural Loudness Matching

The last method to be identified here is an equal loudness estimation between a signal presented monaurally and the same signal presented binaurally with a period of silence in-between. Loudness matching using amplitude adjustment gave the participants the most freedom in determining a loudness match where the participant was given control over the signal's amplitude. By the nature of a paired match, there were then two methods of amplitude adjustment for loudness: (i) match a fixed reference signal to an adjustable target tone, and (ii) matching a variable reference signal to a fixed target tone. Bracketing techniques are often included in this form of experiment where participants were limited to upper and lower adjustments; always increasing or decreasing the loudness to 'zero-in' on the loudness match. As mentioned previously, the equal loudness contours were initially derived using this experimental form of loudness matching by Fletcher and Munson, [1].

When used to explore the concepts of binaural summation the two methods of amplitude adjustment were altered slightly: (i) match a fixed reference monaural signal to an adjustable binaural tone, and (ii) matching an adjustable monaural signal to a fixed reference binaural tone. By targeting the function using both approaches any unwanted bias may be removed as in Hellman and Zwislocki's balanced approach.

- - - - - - - - - - -

After a review of the signal presentation techniques, it was apparent that there were multiple approaches which differed for deriving the trends present in loudness. Each method targeted the subjectivity of loudness sensation which required a large sampling of participants for accurate results. Literature relating to loudness functions and the associated binaural summation mechanisms have been included in the following section. Where applicable, the listening conditions used for the experiments have also been included. Given the differences outlined above, comparisons between the values are to only be done with caution.

3.1.3 Literature Review of Loudness Summation

A vast amount of research has been dedicated to determining the appropriate loudness function and the resulting binaural summation model. While the bulk of this information was reported several decades ago, reoccurring concepts or theories are prevalent in recent investigations.

The basic theory states that loudness will increase with amplitude regardless of the listening condition and signal type. The rate of this increase was the primary target of the experimental methods outlined above. It has been generally accepted that for levels above one sone (40 dB at 1 kHz) the relationship between loudness and sound pressure

level has been found to be in the form of a power function as indicated in **Equation 4**. Below one sone the loudness function no longer follows this form and becomes progressively steeper as the amplitude approaches zero. This change in shape has been associated with the sensitivity increase near the audible threshold where the loudness of the signal becomes directly proportional to the sound pressure level, [50].

$$L = kP^n \qquad \textbf{... (4)}$$

Where **Equation 4** is applicable the loudness of a stimulus (L) is equated to a constant of proportionality (k) multiplied by the sound pressure (P) which has been raised to some exponent (n). Under this relationship, the proportionality constant depends strongly on the units of measurement and the exponent defines the slope of the resulting function. The exponent is the target of nearly all loudness investigations and is influenced by the experimental test method used and which signal the subject varies during the test. A sampling of exponents from the literature has been included in **Table 1** where values often differ depending on the listening condition investigated; either a monaural or binaural presentation. Where both exponents are available from the same author a comparison of the results are indicative of the ratio for binaural loudness summation (i.e. the relationship for the monaural to binaural conversion and the influence of signal amplitude).

Table 1- Summary of loudness functions, applicable for signals above 40 dB.

Experimental signal	Loudness Function Exp.	Reference, Date Published
Magnitude estimation (pure tones over entire hearing spectrum 20 Hz-20 kHz)	0.60 (B) (0.3 for intensity)*	Stevens, 1956 [58]
Magnitude estimation (white noise band 250 Hz – 2 kHz)	0.56 (B) 0.50 (M)	Reynolds and Stevens, 1960 [52]
Magnitude production (white noise band 250 Hz – 2 kHz)	0.67 (B) 0.63 (M)	Reynolds and Stevens, 1960 [52]
Cross-modality matching (white noise band 100 Hz – 500 Hz + 60 Hz vib.)	0.64-0.78 (B) 0.51-0.66 (M)	Reynolds and Stevens, 1960 [52]
One-vs-two-ear ratio production (white noise band 100 Hz – 500 Hz)	0.58 (B) 0.55 (M)	Reynolds and Stevens, 1960 [52]
Monaural-binaural loudness matching (white noise band 100 Hz – 500 Hz)	0.60 (B) 0.54 (M)	Reynolds and Stevens, 1960 [52]
Magnitude estimation	0.54 (B)	Hellman and Zwislocki, 1961 [50]
Psychological magnitude balance (1 kHz)	0.54 (M)	Hellman and Zwislocki, 1963 [50]
Magnitude estimation, magnitude production (1 kHz pure tone)	0.43, 0.53 (B) 0.48, 0.52 (M)	Scharf and Fishken, 1970 [59]
Combined data of 1 kHz tone and noise	0.48 (B) 0.50 (M)	Scharf and Fishken, 1970 [59]
Magnitude estimation	0.6 (B)	Stevens, 1975 [60]
Magnitude estimation, production, and loudness matches (1, 3 kHz + noise)	0.6 (B)	Hellman, 1976 [61]
Magnitude estimation of low frequency pure tones (100 Hz), high frequency pure tones (400 Hz, 1 kHz).	Low - 0.75 (B) High - 0.60 (B)	Marks, 1978 [49]
Paired comparison tests using broadband noise low-passed to 5 kHz.	0.34 – Children (B) 0.55 – Adults (B)	Schneider and Cohen, 1997 [62]
Noise bursts, magnitude estimation	0.54-0.90 (B) (0.27-0.45 for I)*	Zahorik and Wightman, 2001 [63]
Magnitude production (1 kHz tone with varying knob settings)	0.62-0.78 (M) (0.31-0.39 for I)*	Joshi and Jesteadt, 2013 [53]

(B) – Binaural Signal Presentation, (M) – Monaural Signal Presentation

* Original data approximated for Steven's Power Law with exponents for sound intensity (e.g. $L = kI^{0.3}$ where I is to the energy flux density and is proportional to the square of sound pressure), [22]

From **Table 1** it is clear how the various experimental methods resulted in significantly different exponents, even for investigations by the same author. It should be noted that as loudness is a subjective quality, the response of participants does not always follow a specific trend such as the power function. In the case of Scharf and Fishken's investigation for wideband white noise the authors quickly discovered that they could not fit the data into an exponential trend. Results alternatively suggested that the contours appeared bowed or concave downwards on a log-log plot of loudness versus sound pressure. Therefore, the loudness appeared to increase rapidly at low amplitudes with decreasing slope as the sound pressure level was increased, [59]. This is another example of the range of results available in the literature which demonstrates a source of confusion among theories. For the purposes of this investigation it will be assumed that the relationship is based on the power function as the majority of other researchers have suggested.

Once the trends became clearer for how loudness varied with respect to amplitude, attempts were made to relate the monaural and binaural loudness functions to potentially derive a method of converting between the two mathematically. A binaural signal presents the participant with two channels of data to be assessed simultaneously as a single signal. This internal summation process became known as 'binaural summation' and proved to be a topic of dispute in the loudness community. The first documented loudness function including binaural summation was Fletcher and Munson's 1933 loudness model, [1]. At the time research suggested that binaural hearing was driven by the better of the two ears and under diotic conditions the loudness of a monaural tone could be assumed to be half that of a binaural tone, [1]. This conversion simplified loudness calculations and lead to

the first theory of binaural summation, suggesting that listening with two ears would always appear twice as loud as listening with one (a simple energy summation).

After the theory of simple energy summation was largely abandoned in the 1930s, experimenters began summarizing the difference between the two monaural and binaural loudness functions as either a constant ratio of loudness or a constant increase in SPL, [55]. When the two loudness functions are distinct but parallel (i.e. of the same exponent), the difference between the two can clearly be seen on a common plot as in **Figure 3.1** on the following page.

In the provided example, the binaural and monaural loudness functions may be considered to be power functions with the same exponent. Consider the 40 dB example as identified by the highlighted portion (**A**). From the figure, it appears that this 40 dB signal would be perceived as having a monaural loudness of approximately 0.69 sones and an associated binaural loudness level of 1.0 sone. Therefore, a ratio may be derived for converting from one function to the next, in this case the ratio is equal to 1.45 (i.e. 1.00/0.69 = 1.45). This relationship has become known as the Binaural-to-Monaural Loudness ratio (the **BML ratio** for short). As these two line are parallel, it necessarily follows that this relationship will hold for any signal above 30 dB in this instance.

Similarly, consider the binaural and monaural signals indicated at highlighted portion (**B**) which both exhibited a perceived loudness value of 2.0 sones. In this instance the binaural signal has an amplitude of 50 dB while the monaural signal has an amplitude of approximately 57 dB. As each signal has been demonstrated to have an equal loudness value, it can be stated that for two parallel loudness functions the binaural difference

required for equal loudness **(BDEL)** is 7 dB (57 dB – 50 dB = 7 dB). Therefore, two separate loudness functions may be related to one another using either the BML ratio or the BDEL value provided that both functions are parallel to one another. As most researchers chose to present only the BML ratio, the following comparisons have been converted to this format where applicable.

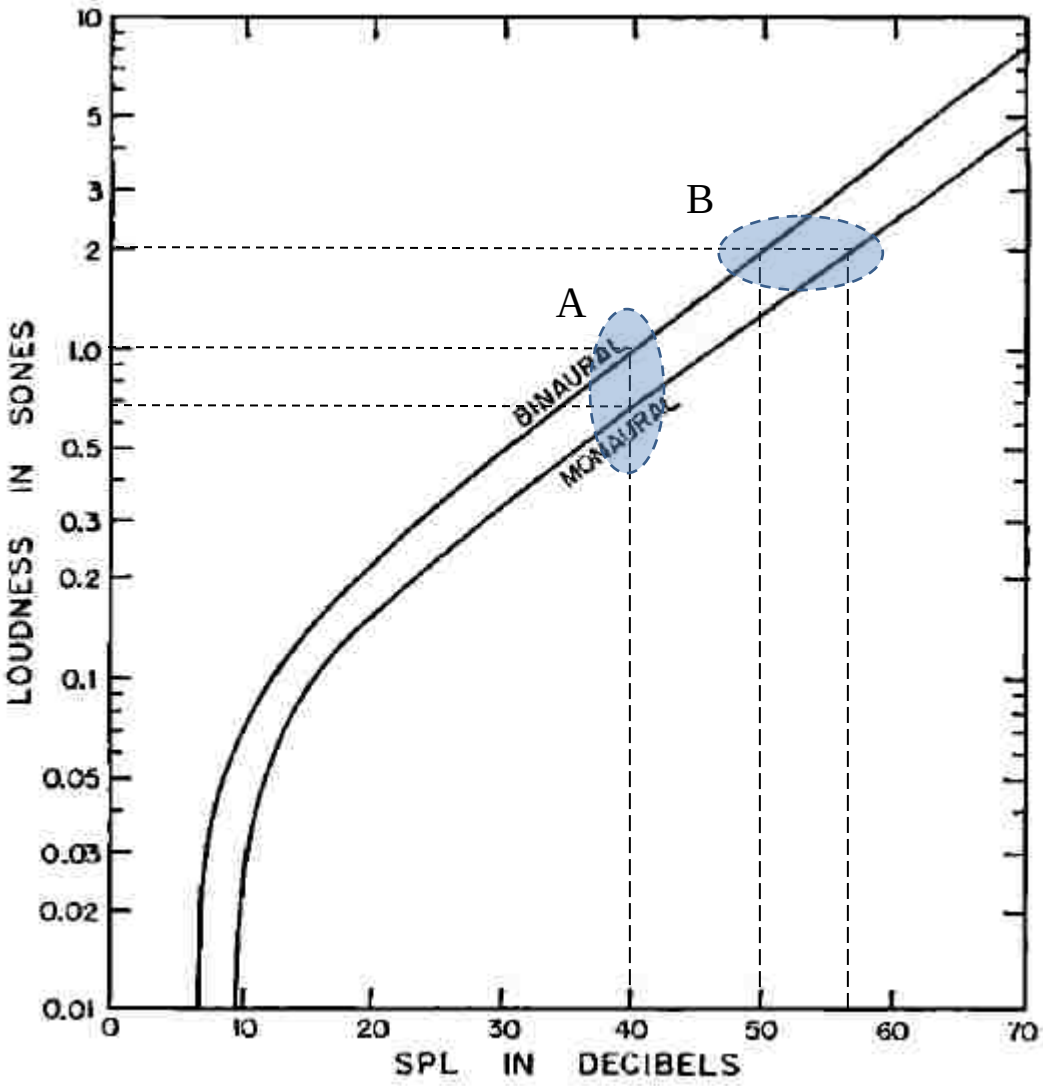


Figure 3.1 - Comparison of (A) the binaural to monaural loudness ratio and (B) the binaural difference required for equal loudness, (figure adopted from [52]).

Much like the loudness functions themselves, values for the BML ratio have been found to vary over a relatively large range from 1.05 to 2.50 where experimental methods and results differed considerably (for a few examples see [26], [52], [59], [64] and [65]). As technology improved and more complex methods became available, several other theories were proposed. For the purposes of this investigation, a review of the results will be grouped into the following theory-sets: (A) perfect binaural summation (ratio of 2.0); (B) a constant ratio, often less than 2.0; (C) binaural summation which varies with respect to amplitude; and (D) the theory that binaural summation does not exist. Each of these theories will be identified below along with supporting literature for comparison.

A criticism of the BML ratio is that, in reality, it only applies for the diotic listening condition where identical signals were present at each ear. In any other case, such as when the listeners head is even slightly turned, the mechanisms of binaural summation are influenced by what has become known as binaural inhibition; the influence of perceived loudness from one ear reducing the perception of the other. For the purposes of this investigation binaural presentations will be limited to the frontal incidence or diffuse field scenario of diotic listening (an identical presentation to both ears). Dichotic listening conditions where the SPL differs from one ear to the other are beyond the scope of this investigation.

A. Perfect Binaural Summation

The theory outlined by Fletcher and Munson's model assumed that the loudness of a signal presented to both ears will always be perceived as twice as loud when compared against the same signal being presented to one ear alone. Hellman and Zwislocki dubbed

this theory "a hypothesis of perfect summation" where the BML ratio was found as 2.0 regardless of the signal's amplitude, [50]. That is, the monaural loudness function and the binaural loudness function appear parallel and offset vertically by a factor of two on a common plot. In terms of sound pressure levels this theory required that a binaural signal be perceived to be equally as loud as a monaural signal only when the two differ in amplitude by a constant amount. Most literature suggests this difference to be 10 dB, as seen in [66]. Several others have observed this trend in summation where supporting literature has been summarized in **Table 2** below. Recall that this form of binaural summation is included as the standardized approach in the ANSI S3.4 stationary loudness model.

Table 2 - Experimental results supporting the theory of perfect summation (i.e. BML ratio 2.0).

EXPERIMENTAL METHOD	RESEARCHERS, DATE PUBLISHED
Simply assumed, (see **[55]**)	Fletcher and Munson, 1933, 1937 [1], [67]
Unknown, (confirmed by **[50]**)	Stevens and Davis, 1938 [67]
Psychological magnitude balance of 1 kHz signal	Hellman and Zwislocki, 1963 [50]
Magnitude estimation and loudness matching, 1 kHz tones	Levelt et al., 1972 [55]
Magnitude estimation (100 Hz, 400 Hz, and 1 kHz)	Marks, 1978 [49]
Magnitude estimation of two-tone complexes of 1 kHz and 1.1 kHz paired at varying amplitudes	Algom et al. 1989 [68]

With regard to the equal loudness contours of ISO 226:2003, the theory of perfect summation corresponds to a 10 dB increase in SPL for mid-range frequencies. This can be explained as follows where an attempt has been made to extend the discussion across the entire audible spectrum. According to the equal loudness contours and their standard definition, a doubling of loudness can be measured as the difference in amplitude between

two adjacent contours. In other words, a 1 kHz pure tone at 40 dB (1 sone), will sound twice as loud when the tone reaches 50 dB (2 sones). From this example, a signal must increase by 10 decibels in order to be perceived as twice as loud. Therefore, under the theory of perfect summation, it would necessarily follow that the monaural to binaural conversion is simply an addition of 10 dB. It would be easy to apply this addition at all frequencies of interest; however, it has been demonstrated via the equal loudness contours that the hearing sensitivity of the ear varies with frequency. This effect can be clearly shown as the calculated difference between adjacent contours. For a visual aid, this difference has been presented in **Figure 3.2** below; please refer to **Figure 2.3** for the equal loudness contours used in this derivation.

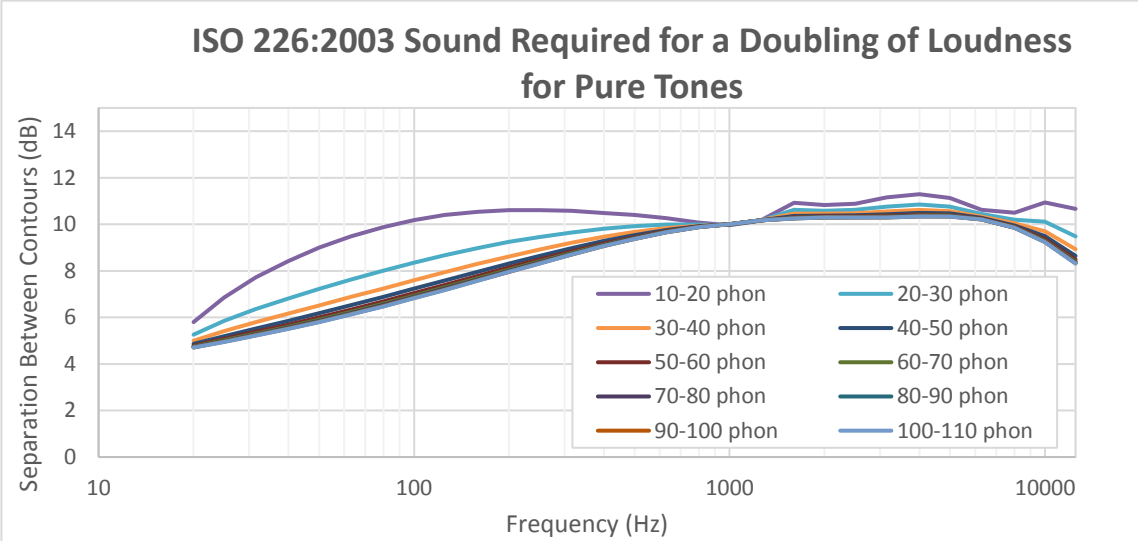


Figure 3.2 - Increase required for a doubling of loudness as a function of pure tones and frequency.

While this figure applies only to pure tones, it is interesting to note that the increase required for a doubling of loudness is not constant across the spectrum. For loudness levels above 30 phons it appears that the perceived increase varies as a logarithmic function from 5 dB at 20 Hz up to 10 dB at 630 Hz. Beyond this frequency

the contours follow the adage that a doubling of loudness is represented as a 10 dB increase in amplitude, up to an including 8 kHz where the difference begins to drop once again.

It is important to note that this derivation was based solely on the pure tones used for the equal loudness contours. The approximation of binaural summation typically occurred at the end of each loudness model, once the overall monaural loudness had been calculated. At this point, there is no spectral content associated with the signal, as monaural loudness is calculated as the area under the specific loudness plot. A doubling of loudness from the area cannot readily be expressed as a simple increase in SPL based on existing information.

B. Constant Ratio, less than 2.0

While perfect binaural summation was a convenient theory for calculations, recent investigations have suggested that a BML ratio of 2.0 may be overestimating the perceived loudness. For example, Scharf was able to conclude that the binaural summation of pure tones (500, 1000, and 2000 Hz) occurred at a rate which was less than twice as loud as the monaural counterpart; an indication of less than perfect binaural summation, [69]. These concepts were revisited by Scharf and Fishken confirming that the exponent of the power law was constant for pure tones beyond 10 dB; that is, the binaural to monaural ratio will not change with increasing SPL, [59]. Similar to those results presented above, this theory suggested the monaural loudness function and the binaural function would appear parallel on a common plot; in the case of [59] the separation between the signals would be a factor less than two. Unfortunately, the exact

BML ratio had been found to vary considerably between reports as a result of the various testing methods applied and technology used for each study. A summary of the available literature has been included in **Table 3** below.

C. Binaural summation which varies with respect to amplitude

When the power law exponents differ between the monaural and binaural loudness functions the BML ratio would appear to increase (or occasionally decrease), with amplitude. For exponents greater than one, the BML ratio would appear to grow; particularly at high intensity levels. Hellman and Zwislocki theorized that the annoyance of high intensity levels may be to blame for this growth as a tone presented to both ears would only serve to emphasize this effect, [50]. Regardless of the cause, several researchers have adopted the theory that binaural summation does exist and appears to vary with respect to amplitude.

In most cases, it was easiest for the authors to summarize the binaural difference required for equal loudness (BDEL) as a single value. For example, a 3 dB difference suggested that the results of jury testing confirmed a monaural tone presented 3 dB greater than the binaural tone would be perceived equally as loud. Similar findings have been found at various amplitudes from a variety of signal types. While most took this form, others resembled a multi-stage BML ratio function as demonstrated in **Table 4** below. In the case of Reynolds and Stevens, the monaural to binaural summation ratio varied as a power function which would appear as a straight line on a log-log coordinate system; with the monaural amplitude on the abscissa and the resulting binaural level on the ordinate. Like the loudness functions, the amplitude of the BML ratio was dependent on the exponent of the power function and amplitude of the signal.

Table 3 - Literature summary of constant ratio/rate, less than 2.0.

Experimental Method	BML Ratio	Researchers, Date Published
Loudness matching using narrowband stimuli of Bark-band-width.	1.5 (6 dB)	Robinson and Whittle, 1960 [70], [71]
Thermal noise (gas tube type). Literature included a plot with no exponent.	- -	Irwin, 1965 [72]
Magnitude estimation and magnitude production using a 1 kHz tone produces a constant BML ratio value in this range.	1.6-1.85	Scharf and Fishken, 1970 [59]
Unknown, referenced from **[73]**	1.65	Jankovic and Cross, 1977 [74]
Loudness matching via paired-comparisons of 200 Hz and 2 kHz tones. No value.	- -	Gigerenzer and Strube, 1983 [75]
Magnitude Estimation of varying durations of a 2 kHz signal, (not true for white noise)	1.84 (7 dB)	Algom et al, 1989 [73]
Magnitude estimation of broadband noise (red, pink and blue) and narrow-band white-noise centered on 250, 710 Hz and 2 kHz.	1.4-1.8	Zwicker and Zwicker, 1991 [76]
Magnitude estimation of 500 Hz and 1 kHz pulsed tones.	1.4 (5-6 dB)*	Mulligan, 1992 [77]
Magnitude estimation of 500 Hz pure tones presented via induced loudness exponents.	<2.0 (8-9 dB)	Marks et al., 1995 [78]
Cross-modality matches using string length and 1 kHz tones.	< 2.0 (1.3 in [79])	Marozeau et al., 2006 [65]
Loudness matching of 1/3-octave noise bands centered on 400 Hz, 1 kHz and 5 kHz.	1.22 (3 dB)* 'power-sum'	Sivonen and Ellermeier, 2006 [80]
Loudness matching using loudspeaker presentations of pink noise and narrow-band noise samples centered at 1 kHz and 5 kHz.	1.22 (3 dB)* 'power-sum'	Sivonen, 2007 [81]
Literature review (see **[59]**, **[76]**, **[82]**, and **[83]**)	1.5	Moore and Glasberg, 2007 [66]
Loudness matching using noise bands of various widths.	1.4 (5-6 dB)	Culling and Edmonds, 2007 [82]
Narrow band noise was presented as a binaurally synthesized sample	1.22 (3 dB)* 'power-sum'	Sivonen and Ellermeier, 2008 [71]
Paired-comparison matching using 1-ERB noise-bands (250 Hz, 500 Hz, 1 kHz, 2 kHz, and 4 kHz) and three wide-band sources.	1.4 (6 dB)*	Edmonds and Culling, 2009 [84]
Magnitude estimation of 1 kHz tones and recorded spondees via earphones, loudspeaker	1.23-1.56	Epstein and Florentine, 2009 [64]
Magnitude estimation of spondees via monitored live-voice presentation	1.05	Epstein and Florentine, 2009 [64]
Review of literature in **[79]**	1.2-1.5 (3-6 dB)	Sivonen and Ellermeier, 2011 [85]

For the table above results of various investigations were converted to a binaural to monaural ratio using approximations found in the literature. These approximations were as follows:
* Gains of 3 dB and 6 dB correspond to binaural to monaural ratios of 1.2 and 1.5 respectively, [71].

Table 4 - Literature survey of binaural loudness which varies with amplitude.

Experimental Method / Function	Researchers, Date Published
Methodology was unknown as a translation was unavailable. The BDEL was found to increase gradually with amplitude up to a maximum at 35 dB. Signal | 20 dB | 35 dB | >35 dB 1000 Hz | 3 dB | 6 dB | 6 dB	Causse and Chavasse, 1942 [86]
Wide band noise above 40 dB. Magnitude estimation, magnitude production, one-vs-two ear ratio, loudness matching, and cross-modality matching. BML ratio increases as a power function with an exponent of 0.066. The BML ratio therefore varies from 1.5 at low amplitudes to a ratio of 2.0 at 90 dB and increases.	Reynolds and Stevens, 1960 [52]
Loudness matching of 500 Hz and 2 kHz pure tones. Binaural summation appeared to decrease with amplitude, the actual function of this decrease were undefined. The decrease was more prominent for the 2 kHz tone.	Porsolt and Irwin, 1967 [87]
Loudness matching using 1 kHz pure tones and bands of white noise of one-critical bandwidth, one octave and a broadband with a low-pass filter at 7 kHz. The BDEL was found to increase gradually with amplitude up to a maximum at 40 dB. Signal | 20 dB | 40 dB | >40 dB 1000 Hz | 3 dB | 6 dB | 6 dB	Scharf, 1968 [88]
Loudness matching of 500 Hz, 1 kHz, and 2 kHz. The BML ratio was found to vary with both amplitude and frequency. Signal | 20 dB SL | 30 dB SL | 80 dB SL 1000 Hz | 1.8 | 1.5 | 1.3 2000 Hz | 2.0 | 1.7 | 1.4	Scharf, 1969 [69]
Magnitude estimation and magnitude production where the BML ratio appears to vary with for broad-band noise from 30 dB to 110 dB as follows, (not a power law function). Signal | 30 dB SL | 110 dB White Nosie | 1.3 | 1.7	Scharf and Fishken, 1970 [59]
Paired-comparison for adults and children using broadband noise samples which were low-pass filtered to 5 kHz. The BDEL was found to increase with amplitude. Signal | 30 dB | 90 dB Noise-band | 5 dB | 16 dB	Schneider and Cohen, 1997 [62]
Loudness matching procedure using a 1 kHz tone. The BDEL was found to increase with amplitude. Signal | 20 dB | 60 dB | 100 dB 1000 Hz | 4 dB | 10 dB | 5-6 dB	Whilby et al., 2006 [83]

D. Binaural Listening shows no superiority

The final theory has very little support in the literature, that is, a notion where binaural listening shows no superiority over monaural listening. There are two forms of this theory which have been encountered. The first was observed by Chouard while investigating the dichotic perception of automotive cabin noise for overall loudness, (from [89]). In this instance the perceived binaural loudness was found to be the arithmetic mean of the perceived loudness in sones from the left and right ears individually. While this method demonstrated a concept of binaural inhibition (where the response of one ear inhibited the sensation of the other), the vast amount of literature for the remaining theories suggested that more research may be necessary. In 2007 Moore and Glasberg updated their stationary model to include the influence of binaural inhibition for dichotic signals; a manner of accounting for perfect summation with a penalty for off angle listening, [66]. This updated model had good agreement with past literature but still did not account for changes in binaural summation with frequency.

The second form of this theory is present in the DIN 45631/A1 draft where the binaural loudness was taken as the louder of the two ears. The intended application here was for measurements conducted via 2-channel HATS listening device. Again, this is an antiquated approach given the processing power of the current data acquisition systems. In the case of a diotic listening condition this method would simply be the loudness perceived by either ear alone.

3.2 What is Missing? Identification of Problem Statement

It has been demonstrated through a substantial amount of evidence that binaural summation does exist and can in fact be measured. Where diotic listening is concerned,

various theories have been devised for this summation where the perceived binaural loudness may be: (i) twice as loud as the monaural stimulus; (ii) a multiple above the monaural signal which is less than double; (iii) dependant on a function of amplitude; or (iv) represented by the average of both ears or simply the loudness of the better ear. While the results of such measurements and the corresponding theories are still under debate, it was clear to the author that one important trend has been overlooked in this literature – How does the binaural summation vary with respect to frequency?

The question arose while examining the equal loudness contours of ISO 226:2003. It appeared as though the perceived loudness of a signal increased at a constant rate as the amplitude was increased from the threshold of hearing to the threshold of feeling. Markedly apparent was the fact that this perception was heavily dependent on the frequency of the tone. The rate of increasing loudness was not constant across the frequency spectrum, specifically at low frequencies where a small increase in sound pressure level equated to a relatively large increase in perceived loudness. Few sources in literature actually focused on the influence of frequency on binaural summation. Those sources which were found have been summarized in the following paragraphs.

In 1948 Hirsh was one of the first to recognize that the inhibition of binaural signals was clearly related to the frequency of the stimulus and alluded that this may extend to a binaural mechanism dependant on frequency, [90]. To the knowledge of the current author, this statement was not investigated any further and the topic was largely forgotten. No further mention was made until 1978 when Marks concluded the loudness summation at various frequencies appeared to fall along a single function, [49]. The author concluded that the summation was ‘perfect’ at any given frequency, contrary to the

hypothesis of the present study. This was again supported by Zwicker and Zwicker 1991 during their investigation of various spectral patterns from noise samples (red, pink and blue) as they determined that the BML ratio was in no way influenced by frequency, amplitude or spectral shape, [76].

More recently, Zhang and Mao revisited the concept of binaural summation in 2010 with particular attention given to the interaural difference between the ears, [89]. Results of their research appeared to demonstrate a visible fluctuation of the perceived loudness for higher frequency spectra. Above 500 Hz, there were indications that higher frequencies had a greater influence on how loud a particular tone was perceived. Even though their results limited the fluctuation to +/- 1 phon (+/- 1 dB at higher frequencies), their conclusions confirmed that frequency may play a minor role in the amount of binaural summation. More research was required in order to determine the cause for these fluctuations, [89].

Based on these observations it was clear that more information was necessary about this topic with regard to how the binaural summation mechanism was influenced by frequency. The current state of the art procedure as summarized by the globally accepted and standardized loudness metrics (i.e. perfect summation), has been shown to deviate significantly as presented in recent literature. The following research has therefore been designed to challenge the state of the art procedure and produce an improved loudness model based on new information. The anticipated outcome of this research was therefore a loudness metric capable of accurate binaural loudness predictions with consideration for frequency content.

IV. EXPERIMENTAL DETAILS

The derivation of the relationship between frequency and binaural summation required the employment of several jury test investigations to ensure accuracy in the data and confidence in the conclusions. The first investigation involved pure tones of constant amplitude which were presented using a loudness matching paradigm. The resulting information raised more questions than conclusions and the experimental methods were completely redesigned for a follow-up experiment. Ultimately, the findings confirmed the initial hypothesis and lead to an improved binaural summation algorithm based on frequency that could readily be applied to any loudness model.

4.1 Initial Investigation – Constant Sound Pressure Level

The purpose of the initial investigation was simply to determine whether the hypothesis that 'binaural summation does exist and in-fact varies as a function of frequency.' To test this theory, an experiment was devised wherein human volunteers were required to match the loudness of alternating monaural and binaural signals.

From the literature review it was apparent that the primary method for deriving the binaural summation mechanism was based on the difference between the individual monaural and binaural loudness functions, (i.e. the functions relating perceived loudness to amplitude under each listening condition). Unfortunately, this approach proved to be quite time consuming for both the experimenter and the volunteers who were particularly susceptible to listening fatigue. Therefore, an alternative approach was employed for more direct results. Recall that the original loudness functions depended on magnitude

estimation/production techniques which focused on a single presentation method at a time. In 1970 a study by Scharf and Fishken confirmed there is good agreement between the results obtained through the traditional magnitude estimation/production approach and that of the more direct loudness matching task, [59]. Based on this information it was decided that the latter method would be employed for the current investigation as it would provide consistent results with literature while maintaining greater flexibility for the experimenter and a shorter testing period for the subjects. The selected approach came with the added benefit of using the existing resources at the University of Windsor.

During the experimental design, it was noted that matching type experiments were subject to a known bias. Reoccurring trends associated with the presentation order of the signals appeared to strongly influence the matched results. An effective manner of reducing this bias was subsequently developed by Hellman and Zwislocki who simply repeated their investigations using an inverted experimental method and averaging the results, [91]. Recall from **Section 2.6.1** that the authors dubbed this approach the balanced loudness matching experiment which permitted the bias associated with each of the two methods to counterbalance each other. As will be seen from the results in the following chapter, the balanced approach was equally beneficial in the current investigation where it provided additional insight for preliminary conclusions.

4.1.1 Experimental Set-up

To produce the binaural and monaural signals for comparison, the experimental set-up shown in **Figure 4.1** was devised which satisfied the requirement for the simultaneous presentation and monitoring. Test tones were pre-recorded and presented via Adobe Premiere which output **Signal A** to the left channel and **Signal B** to the right

channel. As the personal computer available for the experiment exhibited a considerable amount of cross-talk between the left and right channels, an external MBOX sound card was employed providing a noise-free two-channel output. Loudness matching required one signal to be held constant while the amplitude of the other was controlled by the participant. Such control was provided via the 10-turn potentiometer which directly attenuated one of the two channels (e.g. **Signal B** in **Figure 4.1**), prior to reaching the Yamaha MX12/6 Mixing Console. The monaural and binaural presentations were then readily achieved using the built-in PAN controls of the mixing board; for a monaural signal the PAN was set to either 'L' or 'R' while the binaural setting was located equally between the two. In this manner, the binaural setting ensured that the two ears would receive equal sensation levels with the left and right ear. The in-phase binaural signal used in this instance would be perceived as though it was originating from within the listener's head; an unavoidable source of error at this point which will be discussed in **Section 6.2.1**. The set-up successfully provided an alternating binaural-monaural pair over two sets of headphones (Sennheiser HD 600's), where the mixing console output was split for simultaneous presentation and monitoring as per **Figure 4.1**.

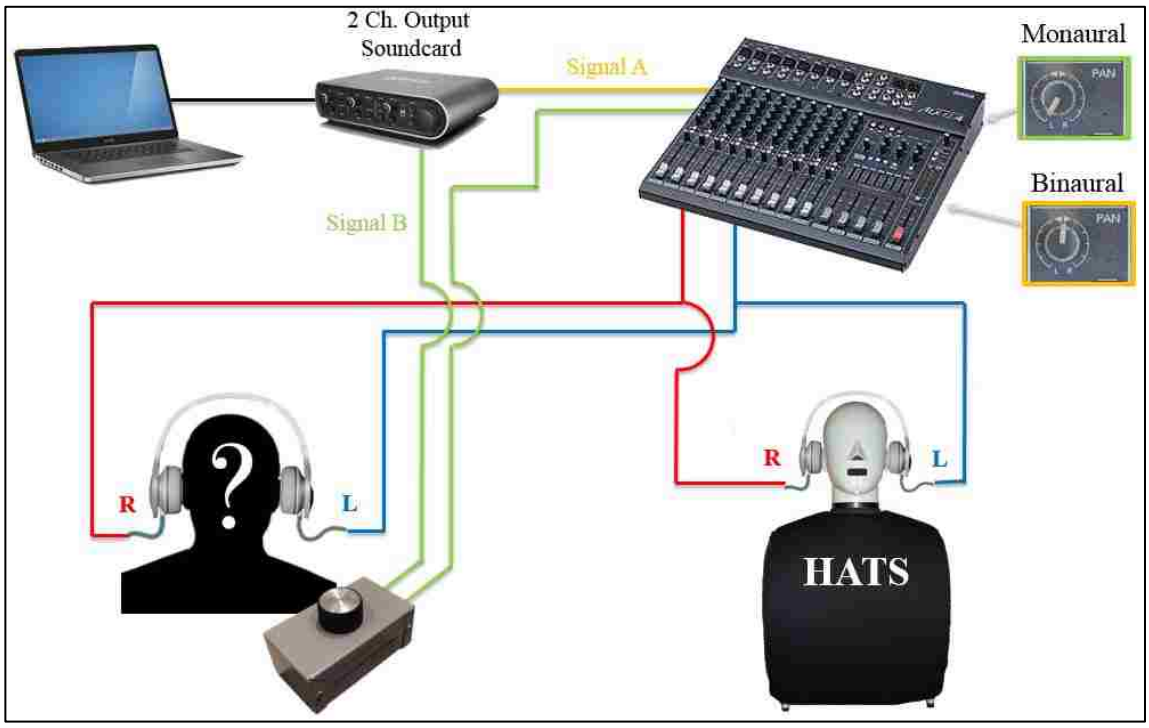


Figure 4.1 - Experimental set-up for the initial investigation where identical signals are presented simultaneously to the observer (?) and the head and torso simulator device.

4.1.2 Procedure

The experiment was carried out in a controlled test environment at the University of Windsor. To the limit external influences during the test, a semi-anechoic chamber was used wherein both the participant and the binaural head were isolated from unwanted noise sources. Within this quiet listening environment, the ambient sound pressure levels were reduced to 45 dB (or 17 dBA), with an 85 Hz lower cut-off frequency.

As participants arrived for the study, the purpose of the investigation was explained to each volunteer and a demonstration of the matching procedure was given. Throughout the procedure volunteers were asked to remain seated in the centre of the room to ensure their field of view was kept clear of any distractions. The experimenter

then monitored and controlled the experiment from a control area outside the chamber while the volunteer performed the matching task inside it. The initial investigation included a series of twenty-seven (27) pure tones with centre frequencies ranging from 40 Hz up to and including 10 kHz in 1/3-octave increments. The spectral range was limited by the headphones as each frequency was to be presented at a constant amplitude of 60 dB SPL. At this amplitude, the pure tones would not overly stress the headphones while accounting for the low frequency limitations of the anechoic chamber.

The loudness matching task required a comparison of the signals. For illustration purposes, these signals have been identified in **Figure 4.1** as **Signal A** and **Signal B**. The two signals were pure tones of the same frequency which differed only in their presentation method as one was a monaural tone (presented to either the left or the right ear) and the other was a binaural diotic tone. The pair was chosen in this manner consistent with the findings of Zhang and Mao. These authors demonstrated that a binaural/monaural pair of the same frequency had been shown to lead to more consistent and reliable results between jury test subjects, [89]. In the present experiment, each stimulus was presented as an alternating loop where one signal was audible for 2 seconds followed by a 0.3 second period of silence before the alternate tone. To reduce the 'popping' sensation, each tone was corrected with an electronic cosine rise and decay ramp of 0.2 seconds. During the comparison, one presentation method (i.e. **Signal A** in **Figure 4.1**) would remain fixed at the reference sound pressure level while the volume control of **Signal B** was adjusted via the 10-turn linear-potentiometer. The potentiometer provided the participant with complete gain control over the signal from zero to 100 percent volume.

To ensure that participants had sufficient time for the matching procedure, each pair of signals could loop indefinitely while the amplitude of the variable signal was being adjusted. Subjects were given as much time as they desired to match the test signals at their discretion. Once the participant was satisfied that the two signals were perceived to be equally as loud as one another, the subjects were asked to remove their headphones and place them on the table in front of them. Once this action was observed on the monitor, the experimenter then made a recording of the matched pair for post processing before initiating the next pure-tone pair for the observer. The experiment continued in this manner until all pure-tone pairs had been matched by the observer.

During the experiment care was taken to ensure that participants with long hair or glasses were mindful to maintain an unobstructed sound path. As the above procedure required participants to remove and reposition the headphones several times throughout the test, additional care was taken to ensure that this practice would not impact the subsequent measurement results. It was stressed to the volunteers that they should strive for consistent positioning of the circum-aural headphones. This was particularly important as the transducer-to-ear sound path may differ significantly between placements. Several researchers have investigated this influence through repeated headphone placements using both HATS devices and human heads with a wide range of results, see [92], [93], [94], [95], and [96]. While not all subjects are sensitive to headphone positioning the authors observed that several of their participants experienced large variability between threshold measurements. Over-ear-cushions are to be completely sealed against the head to prevent leakage resulting from a poor fit. The previous research determined that as the placement was intentionally varied to stress the response, the largest fluctuations were

observed in the higher frequencies (i.e. above 5 kHz) as SPL varied between 2.5 dB and 9.0 dB for pure tones. In terms of audiometric measurements, poor headphone placement has been found to produce spreads of up to 15 dB in repeated trials, [97] as seen in [98]. Note however that these results were from fabricated conditions designed to demonstrate the worst-case scenarios. When participants were left to position their own headphones, Paquier et al. observed that the spread was significantly reduced to acceptable levels, [99]. Following this information, participants in the current investigation were free to position their own headphones and were instructed to position them as comfortably as possible.

The amount of time required to complete the experiment was an important aspect of the testing. By allowing each participant as much time as necessary to match each pair, this ensured that they were satisfied with the results. The consequence of this freedom was the increased time required to complete the experiment. The testing time varied considerably from 45 minutes to 90 minutes depending on the participant. To avoid listener fatigue, breaks were encouraged at 30-minute intervals and the participant could stop at any time without penalty.

In accordance with the balanced methodology of Hellman and Zwislocki, two presentation methods were used for the experiment. The reference was randomly selected for each participant as either the monaural or binaural signal and a method of adjustment was used for the alternate signal, [50]. For consistency, each trial was presented over the same pair of Sennheiser HD 600 headphones. The high-quality transducers provided constant results due to their relatively flat response at each ear; as was verified prior to the start of the experiment. Loudness matches were then recorded via the second set of Sennheiser HD 600 headphones which were placed over the calibrated Brüel & Kjær Type

4100D Head and Torso Simulator (HATS) device. Monitoring of signal amplitude was done in real-time using Brüel & Kjær Pulse LabShop software ensuring that diotic presentations remained equal at each ear and that the reference level remained fixed at the desired amplitude.

4.1.3 Subject Control

As a university based project, recruitment was targeted towards the University of Windsor undergraduate and graduate students. The age range within the population was limited to those students between 18 and 30 years of age to target the average hearing listeners who were free of age related hearing loss. A total of thirty-one (31) volunteers took part in the initial experiment including twenty-five (25) males and six (6) females. While it was understood by the experimenter that cultural and ethnic backgrounds may impact the definition of a 'normal' hearing individual, (language/accent differences may result in differing sensitivities in perception), the volunteer recruitment process could not guarantee an accurate representation from multiple groups. The project continued with this limitation in mind.

Prior to participating, subjects were required to complete a hearing threshold screening test. This followed the procedure set forth by Marks in which the author's criteria required participants to be able to perceive a specific threshold level before including their responses in the results [49]. While the current investigation did not exclude any data, subjects exhibiting any sign of hearing loss were flagged as such.

The screening test was implemented via the free online resource from the Digital Recordings Canada website and a pair of the Sennheiser HD 600 headphones, [100]. This

web based applet automatically stepped through 24 test frequencies at each 1/3-octave centre frequency from 20 Hz up to and including 20 kHz for both the left and right ears. The simplicity of the program permitted signal presentations to be carried out directly from a personal computer where the amplitude was calibrated to 1 kHz using an external coupler. Tones were introduced to participants with starting amplitudes below the threshold of hearing which slowly increase until the perception was indicated via the spacebar. After each response, the program would automatically move onto the next signal until the entire spectrum had been presented. The resulting plot provided a reasonable estimation of the hearing threshold for the individual; an example of the author's response is shown in **Figure 4.2** below.

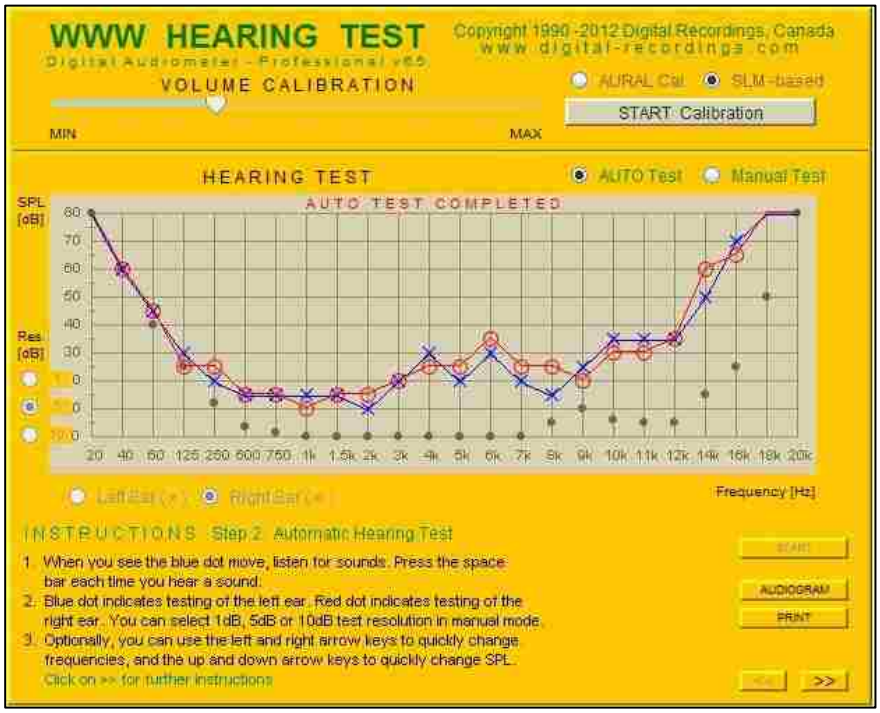


Figure 4.2 - Threshold hearing test of the author as implemented by the Digital Recordings web-based digital audiometer, [100]. The red line indicates the threshold of the right ear while the blue indicates the left. The dotted line demonstrates the average normal hearing listener as per the ISO guidelines.

While conducting the screening procedure, it was observed that the lowest threshold values in the mid frequency region were consistently above 10 dB; indicative of a potential error in the presentation. Upon further investigation, it was clear to the experimenter that, as a result of an error in the program or an unknown hardware limitation, the digital audiometer was not accurately predicting the threshold contours for the individual. In light of this, the screening paradigm was abandoned halfway through the investigation with the following justification. Aside from approximate threshold trends, the hearing test information was unreliable and unnecessarily added to the amount of time required for each participant. In place of the threshold test individuals who had personal knowledge of their own deficient or impaired hearing were asked to indicate so prior to the experiment. This information was used as the flagging mechanism for the remainder of the listeners. From the data that was collected, only a few subjects exhibited mild hearing loss in the mid-to-high frequency range; these subjects were allowed to continue the experiment while their data was isolated to be compared separately. Since their results were similar to the 'normal hearing' individuals, all results were pooled together for the final comparison.

4.1.4 Shortcomings of Initial Experiment

The results of the initial experiment (as presented in **Section 5.1** below), left more questions than answers regarding the true mechanisms for binaural summation. To target these discrepancies directly, a follow-up experiment was devised which could be tailored for each participant of the study with greater control over the accuracy for each signal presentation. Such a task was carried out using the sensation level investigation as outlined below.

4.2 Sensation Level Investigation

Similar to the above experiment, the follow-up investigation involved a loudness matching task using pure tones. In this instance, the primary goal was a derivation of the binaural summation mechanism with specific focus on the influence of signal amplitude and sensation level. Before investigations began however, the entire experimental method was re-examined in an attempt to account for as many external factors and potential errors as possible and ensure additional confidence in the results.

4.2.1 Improved Audiometer

Initial improvements targeted the audiometer and focused on how the threshold of hearing was measured and used in the study. It could be demonstrated that the left and right ears of a listener typically did not have an equal sensitivity to the same sound pressure level and frequency, [72]. In the current investigation, this proved critical as the sensation level of each ear would differ as a result. To account for such differences in previous studies loudness matching experiments have followed one of two correction techniques: (i) the recorded sound pressure level from each channel could be adjusted to account for the threshold of each ear accordingly; or (ii) using the method which Fletcher and Munson adopted, interaural differences could be considered irrelevant once a sufficiently large number of listeners had taken part in the study, [72]. As the initial experiment relied on an audiometer only as a screening tool, the reliability of results for that particular investigation were under the assumption of the latter case. For the current study, it was desirable to present pure tones at an exact level above the threshold for each individual. To achieve this, a new audiometer was created by the author which could accurately account for the interaural differences from each ear as signals were created.

The audiometer was designed with respect to pure-tone audiometry; the current standard for assessing the hearing sensitivity of an individual, [101]. Through air-conduction presentation the program created pure tones from 20 Hz to 20 kHz at 1/3-octave increments as per the procedure described in ISO 8253-1 for pure-tone audiometry without masking, [102]. While it was clear that several internationally accepted standards existed for such a task, the ISO model was chosen as the author had access to the 1989 version of the document in which the procedure is outlined; for a comparative listing of available standards, refer to [103]. It was recognised that the ISO document was updated in 2010; however, as the update version was unavailable it was assumed that the overall procedure had remained the same between versions with only minor revisions (as is most often the case).

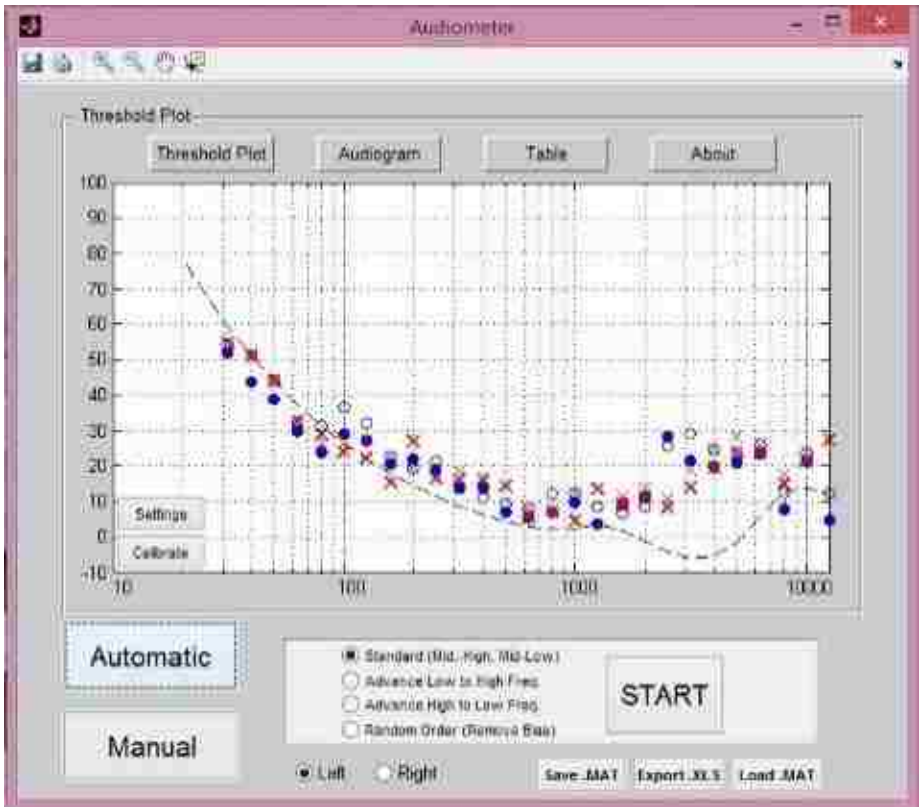


Figure 4.3 - Audiometer GUI created for the purpose of this experiment. Example shows the audiogram for the author (compared against Figure 3.2 in Section 3.1.3).

The resulting graphical user interface (GUI, as shown in **Figure 4.3**) used for the audiometer was written entirely in MATLAB to allow for minute calibrations at each 1/3-octave. Signal generation and conditioning was provided via MATLAB code written by Kamil Wojcicki and accessed through MATLAB Central's File Exchange network, for access to these files refer to [104] and [105] (both files have been included unaltered as part of the Audiometer code in Appendix A). Using this approach pure tones could readily be produced with repeatable accuracy down to three decimal places for upper frequencies and one decimal place at lower frequencies. The calibration of each frequency was a considerable improvement over the previously mentioned web-based applet which could

only be calibrated at the 1 kHz value. The author therefore ensured that any unwanted frequency response from the headphones would not interfere with the desired threshold measurements as each frequency was properly accounted for. Additionally, the new audiometer provided a direct transfer of the threshold measurement into the presentation program outlined in the following section. This addition significantly reduced the amount of time required for the experiment, and in turn reduced listener fatigue. For source code regarding the operation of the author's audiometer refer to **Appendix A.1**.

4.2.2 Presentation Software and Experimental Set-up

During the calibration for each pair of headphones, it became abundantly clear that, to accurately present pure tones of a constant sensation level, the presentation software would need to simultaneously account for the differences between the participant's headphones and the headphones positioned on the HATS device. To completely understand the need for the corrections, it must first be understood that the response of separate headphones may differ despite being of the same make and model. Using an approach similar the audiometer calibration, the FRF characteristics for each pair of headphones used in the current experiment were derived and presented in **Figure 4.4** below. During calibration, pure tones of constant amplitude were introduced at a target amplitude and frequency. The resulting deviation of the measured response from the intended target was then recorded as an indication of the gain adjustment required at each frequency. Once this process was repeated at all frequencies of interest, the complete FRF curve was obtained for each channel and a series of correction values was recorded. The resulting FRF functions for each pair of headphones are presented on a common plot in **Figure 4.4** below. From this comparison, it is clear how the response may vary even for

headphones of identical make and model. Note that these corrections were consistent between multiple measurements taken over a series of months with only minor fluctuations.

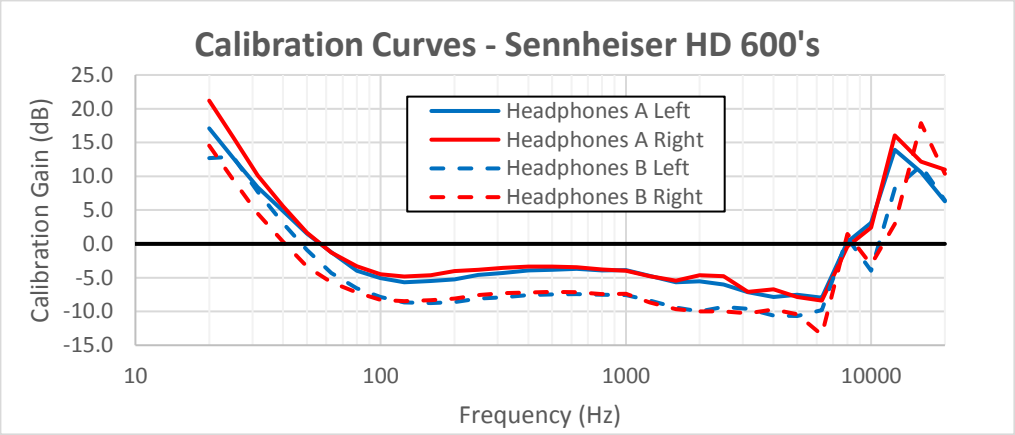


Figure 4.4 - Calibration contours for both headphones used in the experiments.

Once the FRF functions were derived, the next step was to create a new presentation program which could account for the individual headphone correction factors. The resulting interface, shown in **Figure 4.5** provided complete control over the presentation of both monaural and binaural tones; with respect to either linear sound pressure levels or the sensation levels desired for the follow-up investigation.

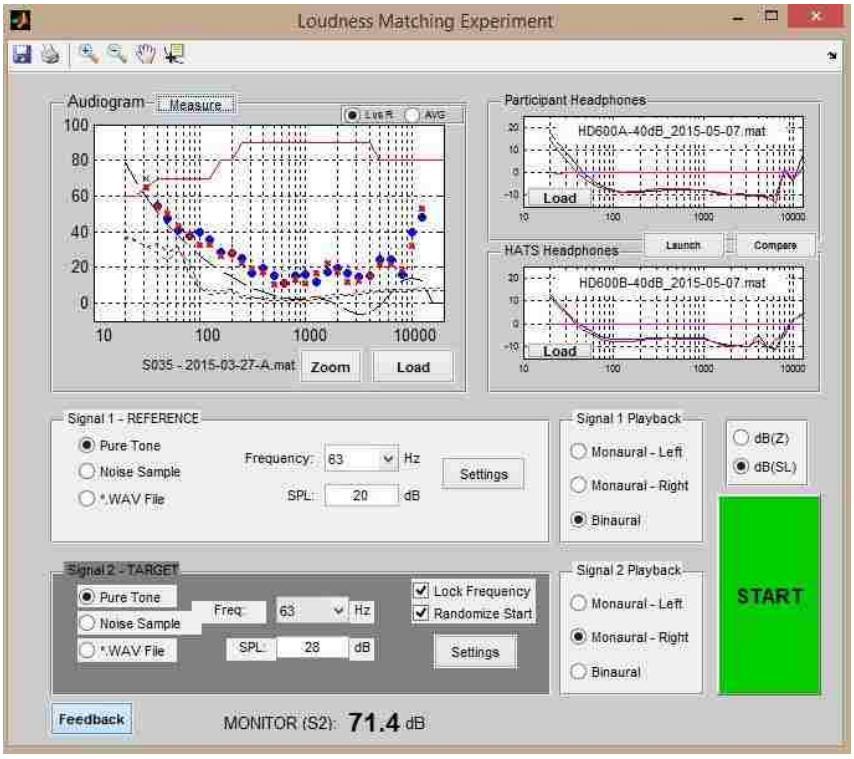


Figure 4.5 - Test GUI written for the purpose of this experiment.

To demonstrate how the program operates, it is convenient to start with the linear tone presentation method first. To achieve a sound pressure level of 30 dB at the observer's eardrum, the generated signal must first be corrected to adjust for the FRF characteristics of the observer's headphones. For this example, refer back to the gain values required for each pair of headphones as presented in **Figure 4.4** at 2 kHz. To produce a pure tone of 30 dB at 2 kHz, the software must produce a signal of 35.5 dB to compensate for headphone Pair A applying an FRF adjustment of -5.5 dB. As the HATS device monitors this signal simultaneously using the second pair of headphones, the measured value is anticipated to be 25.5 dB (at this frequency headphone Pair B applies the greater adjustment of -10.0 dB, a difference which would otherwise alarm the experimenter if there was no account for the difference). As a result, the observer is

presented with a 30 dB tone while the experimenter records a 25.5 dB tone of the same signal as presented over the monitor. Without this feedback, the experimenter would not know if the intended level was reaching the observer since the measured response appears 4.5 dB lower than the desired amplitude.

To convert the linear sound pressure level into a measure of sensation the amplitude of the signal was adjusted with reference to the threshold of hearing at each frequency. A new unit is introduced here, the 'decibel sensation level, dB(SL)' which references the threshold of hearing using a correction factor at each frequency; this adjustment is applied in a similar manner to the A-weighting sensitivity correction as a subtraction between the two. Continuing from the previous example assume the threshold of hearing for an individual was measured as 11.7 dB at 2 kHz. To achieve a sensation level of 20 dB(SL) for the observer the software produces a pure tone 20 dB above the threshold of hearing at that frequency. In the case of the 2 kHz tone, the software therefore would need to produce a 37.2 dB signal which still accounted for the headphone FRF of the observer (5.5 dB) in addition to the threshold correction (i.e. 20.0 + 11.7 + 5.5 = 37.2 dB). Likewise, the experimenter would now observe a pure tone of 27.2 dB on the monitor. Using this approach pure tones of constant sensation level were generated at each frequency. Specific adjustments were tailored to each participant's threshold of hearing as it varied between ears. As a visualization aid, the contours of constant sensation level targeted for this follow-up investigation have been included as **Figure 4.6** below with reference to the ISO 226 threshold of hearing contour. Two amplitudes of constant sensation level were chosen for the experiment including 20 dB(SL) and 30 dB(SL).

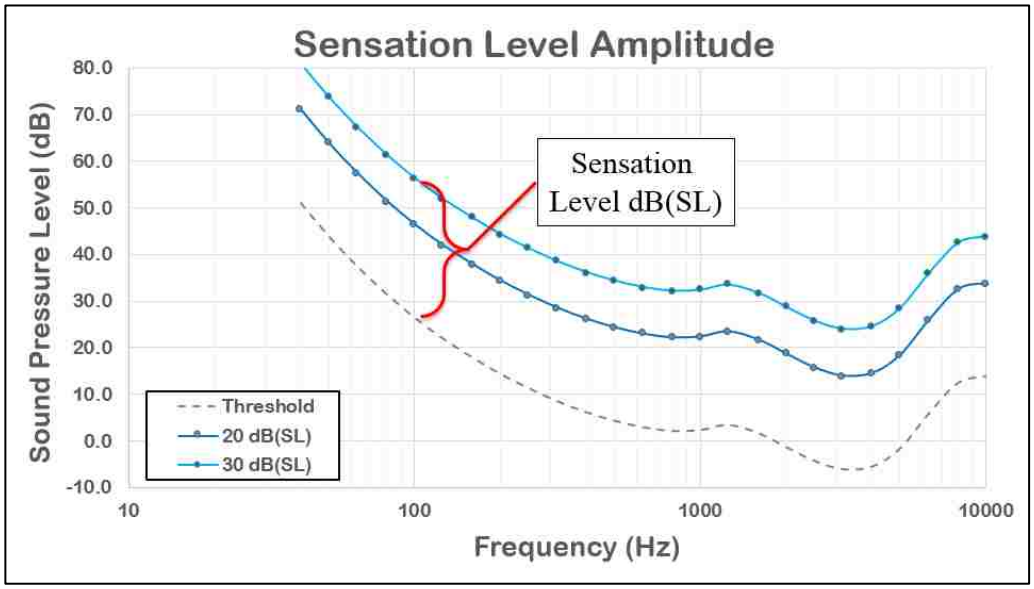


Figure 4.6 - Contours of constant sensation level,
(i.e. 20 dB(SL) and 30 dB(SL)).

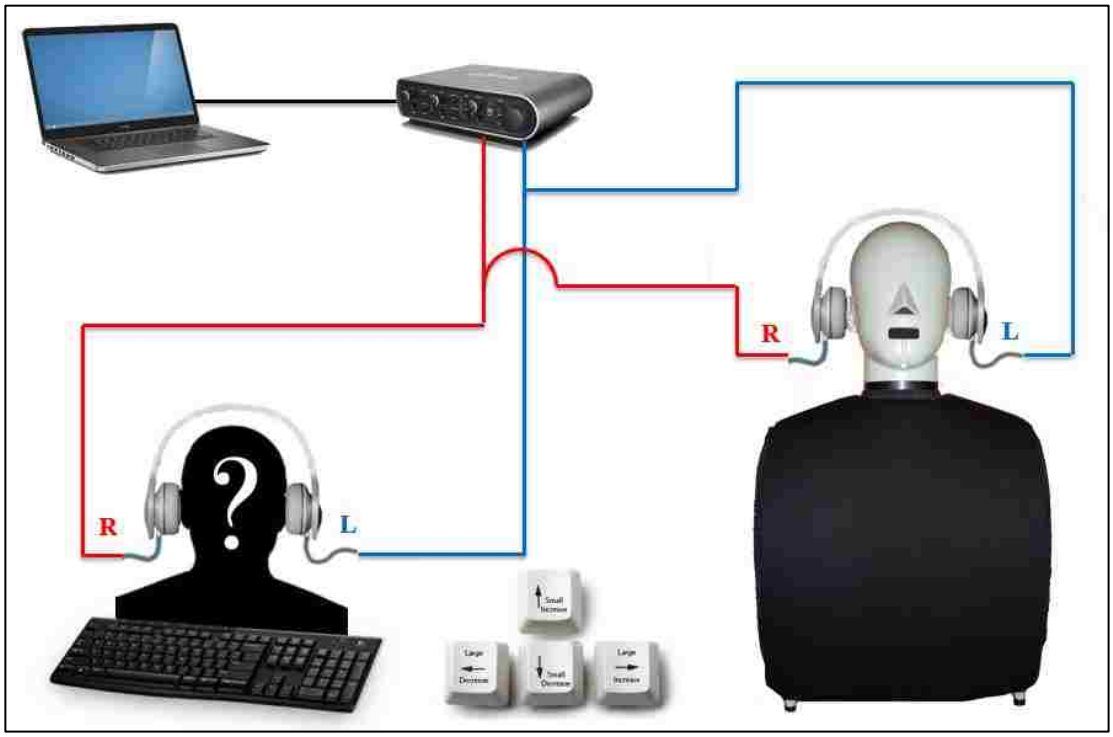


Figure 4.7 - Experimental set-up for the second investigation where identical signals are directly from the MBOX external sound card.

4.2.3 Procedure

The physical experimental set-up was considerably simplified using this approach as shown in **Figure 4.7** above. The function of the mixing board was accomplished electronically and the potentiometer was replaced via a USB keyboard for incremental adjustments in loudness. While new limits could be placed on the monitoring program to prevent excessive noise levels, each loudness match was constantly monitored via the HATS listening device as both a safety precaution and a data acquisition method. Additional information regarding the source code of the listening test program is available in **Appendix A.2**.

The task required for the observer remained unchanged from the previous experiment. Each participant was asked to complete an audiogram before the experiment began which was immediately followed by the loudness matching of various tones for equal loudness. The experimental method was divided into two groups of listeners: those who were presented with a reference monaural tone and were asked to adjust the binaural target; and those participants who used the binaural pair as a reference. The mechanism for binaural summation was then derived in the same manner through the use of the binaural difference required for equal loudness and an average of all the results.

4.2.4 Subject Control

In an attempt to increase the amount of potential data points, the population restrictions were relaxed. The age range was expanded to include volunteers between 18 and 43 years of age and still targeted the normal hearing listeners. A total of forty-nine (49) volunteers took part in this experiment including thirty-nine (39) males and ten (10) females.

As threshold data was used directly for the signal presentations, the interaural differences were closely monitored to ensure a true diotic presentation. Of those who volunteered for the study only one participant exhibited abnormal hearing characteristics and had to be removed from the remainder of the data-pool. The observed threshold of that participant was distinctively unilateral and negatively impacted the results. As the target of the current investigation was a loudness metric for normal hearing individuals, any indication of abnormal hearing trends are required to be removed from the dataset.

4.3 Upgraded Model Verification

Once the results from the sensation level experiment were summarized, potential improvements to the TVL model were identified and applied. The next reasonable step was to verify the performance of the improved model using a variety of test signals including direct feed pure tones, recorded pure tones, and more complex time-varying signals. Additional jury testing was used for this purpose where loudness matches could be compared against the predicted loudness levels from the updated model.

4.3.1 Pure Tone Verification Check

To verify the operation of the model in the most direct manner possible pure tones were fed into the program at specific amplitudes which were selected to create a desired response. Amplitudes were chosen to test the derived summation algorithm at each frequency of interest based on the algorithm. For example, the loudness level of a binaural tone at 'B_f' Hz and an amplitude of 'B_{Lp}' dB should be equal to the loudness of a monaural tone presented at 'M_f' Hz and 'M_{Lp}' dB only if the statements of **Equation 5** below prove

true, (note the derivation of the values used in **Equation 5** are included in **Section 5.2.1** below).

$$B_f = M_f \quad \& \quad M_{Lp} = B_{Lp} + (3.2*\log10(M_f)-2.9); \quad \textbf{... (5)}$$

From this relationship, a 1 kHz binaural signal at 40 dB should be perceived to be equally as loud as a monaural signal presented at 46.7 dB and the same frequency; this satisfies both portions of **Equation 5** above. As part of the verification check the loudness level will be calculated for both monaural and binaural signals where the difference between the two will represent a measure of the performance. As the difference between the predicted values decreases, the accuracy of the model will increase with regard to the intended results.

4.3.2 Verification Signals

The performance of the model was investigated using **two** independent sets of signals. The first set verified the ideal case where the response of the model was examined using computer generated signals fed directly into the program. In this 'Direct Feed' test WAV files were fabricated via MATLAB and fed directly into the model in a controlled manner. The added benefit of this presentation allowed each signal to be free from any external influences such as ambient noise or the inherent sensitivities from the headphones or microphones. The second set of signals were akin to those presented to the jury test participants. Pure tones were generated via the loudness matching program and presented to the HATS device in the semi-anechoic chamber over the calibrated headphones. The resulting response to each tone was then recorded using the HATS microphones in a similar manner to that used during the jury test trials. This data set was unique as it

included the uncontrollable characteristics of ambient noise and the headphone response functions which would further tax the sensitivity of the model.

For each presentation type the binaural summation mechanism was investigated at four separate amplitudes to cover the entire hearing spectrum. The amplitudes chosen included pure tones of 20 dB, 40 dB, 60 dB, and 80 dB generated at each 1/3-octave spanning the entire frequency range of perception. Several limitations existed which governed the range of pure tones available. For convenience, each limitation has been summarized on a common plot as **Figure 4.8** below. The first boundary was that of the TVL model itself. All sound pressure levels below a reference threshold of hearing would be ignored by the program. Thus, pure tones would be limited to amplitudes above the ISO 226 threshold for this investigation. The second limitation stems from the ambient sound pressure level of the semi-anechoic chamber which barred the recording of mid-frequency pure tones near the threshold. The final restriction ensured that the pure tones remained within the limits of the headphones. High-amplitude, low frequency pure tones were unobtainable with the current headphones due to the excessive energy level requirements. With these restrictions in place the resulting reference pure tones used for this investigation have been summarized in **Figure 4.8** as the solid black horizontal lines between those limits outlined above. As a means of verifying the model's performance the pure tones were recorded in pairs where the loudness level of a given binaural tone should theoretically match the loudness level of a monaural tone whose amplitude is fixed via **Equation 5** above.

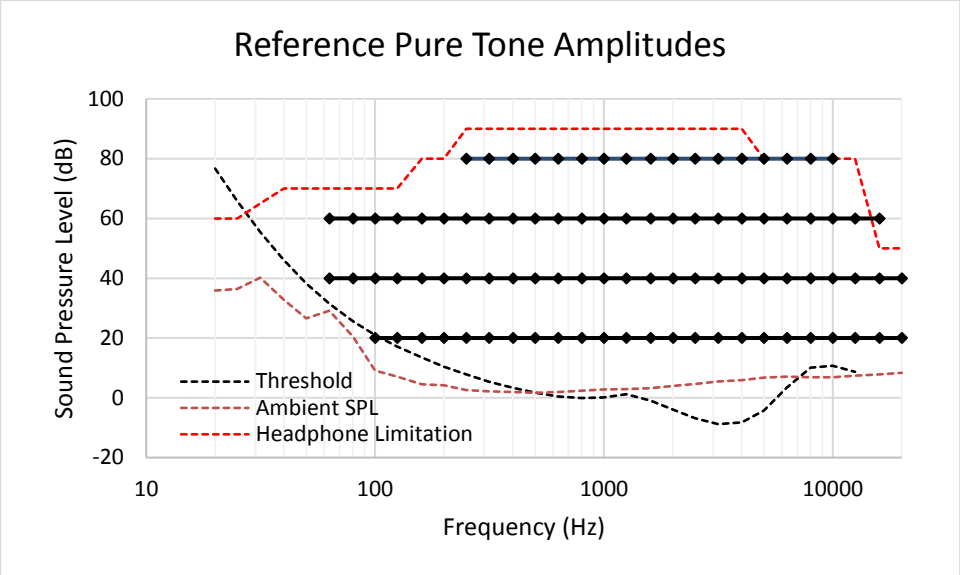


Figure 4.8 - Pure tone selections with reference to the combined limitations of the TVL model (black hashed line), the ambient noise levels (brown hashed line) and the maximum headphone limitations (red hashed line).

4.3.3 Stimuli and Procedure for Model Validation

The final test of the updated model investigated how well the resulting summation mechanism would match actual jury test results. To accomplish this task, the binaural to monaural matching procedure was repeated once again using WAV files which were deemed complex in nature. Signals were selected based on a wide variety of stationary, quasi-stationary, and time-varying signatures which ensured a range of spectral and temporal characteristics chosen to stress various aspects of the model. The following list summarizes the test signals selected:

1) Pink Noise Low (80 Hz – 1 kHz)
2) Pink Noise Wide (80 Hz – 16 kHz)
3) Pink Noise High (1 kHz – 16 kHz)
4) White Noise Low (80 Hz – 1 kHz)
5) White Noise Wide (80 Hz – 16 kHz)
6) White Noise High (1 kHz – 16 kHz)
7) Low 1/1-Octaves (63 Hz – 1 kHz)
8) 1/1-Octaves (63 Hz – 8 kHz)
9) High 1/1-Octaves (1 kHz – 8 kHz)
10) Complex Tones 1 (63 Hz + 250 Hz)
11) Complex Tones 2 (500 Hz + 2 kHz)
12) Complex Tones 3 (250 Hz + 4 kHz)
13) Complex Tones 4 (250 Hz + 1 & 4 kHz)
14) Complex Tones 5 (4 kHz + 16 kHz)
15) Drill Noise (Constant)
16) Drill Noise (Pulsed)
17) Bagpipe Recording
18) Didgeridoo Recording
19) Movie Introduction Music
20) Vehicle Cabin Noise (Steady Speed)
21) Electronic Siren
22) Flute Music Sample
23) Rumbler Siren Noise
24) Clarinet Music Sampler
25) Tuba Musical Sample
26) Violin Musical Sample
27) Speech Segment (“Blade of Grass”)
28) Phone Notification 1
29) Phone Notification 2
30) Phone Notification 3

The first half of the signals (1-14) were identified as deterministic signals with complex spectral patterns. These included six noise bands of various spectral widths and eight complex tone sets. Noise bands were selected as they are commonly found throughout the literature for loudness investigations (as seen in **Table 3** of **Section 3.1.3B**). Complex tones, while less common, serve the purpose of clearly testing the improved binaural summation algorithm using multiple pure tones simultaneously; a direct extension of the studies identified above.

When used for sound quality investigations, complex tones have been known to produce an audible phenomenon referred as a beat sensation. Moore and Glasberg identified this phenomenon as the "regular envelope fluctuation which occur(s) when two sinusoids are added together; the fluctuations occur at a rate equal to the frequency difference between the two sinusoids," [15]. The effect on perception depends on the frequency separation where a small delta may be perceived as periodic variations in loudness. As the frequency separation increases the two-tone complex may invoke a sensation of roughness. Further separation removes any perception of 'beat' and the two tones would sound more distinct to the observer. For the purpose of this investigation, two-tone complexes were selected to be sufficiently far enough apart that the 'beat' sensation would be minimized during the jury test trials; as with any subjective quantity, this response was listener dependant.

The remaining 15 noise samples were selected as short duration (~2 seconds), samples which could be found in everyday life of the participants. The samples targeted a variety of sources such as electric power tools, automotive signals, music samples and cell phone notifications. For more information regarding the signals selected please refer to **Appendix B.1** where each signal was post-processed for spectral content and 1/3-octave information.

The procedure of the validation test closely followed the two initial jury test investigations. Volunteers were asked to match the loudness level of two samples using the same presentation method outlined in the second experiment. Once they were satisfied that monaural and binaural signals appeared equally as loud, the experimenter recorded the matched data-set for post-processing. Each participant completed the series of

loudness matches within 45 minutes of the start; therefore, listener fatigue was unlikely to be a problem during this final investigation.

The performance of the updated model was assessed using the results from each volunteer. Monaural and binaural signals were extracted from participant matches through post processing and converted into the required 16-bit WAV files. Once inserted into the TVL model using the appropriate settings (either monaural or binaural), the measured accuracy of the response was taken as the difference between the monaural and binaural loudness estimates from each pair. Similar to the results of the pure tones verification test, an ideal match would result in a zero phon difference between each monaural-binaural pair. This was of course the definition given by each volunteer, as the two signals were adjusted and identified to be equally as loud.

V. DISCUSSION OF RESULTS

5.1 Constant Sound Pressure Level Investigation

The initial investigation relied upon the method of adjustment wherein the subject had complete control over the amplitude of the stimulus while comparing it to the amplitude of a reference. By adjusting the level of the controlled signal, the subject could force the two signals be perceived as equally loud while the physical difference between them would correspond to the level of summation produced under those conditions. The role of the experimenter was then to monitor the response and record the resulting information.

As each participant matched the amplitude of a monaural stimulus with that of the diotic binaural tone, the resultant physical separation between the two revealed the binaural summation necessary for equal loudness. For instance, when a 1 kHz pure tone of 61 dB is perceived to be equally as loud as a diotic 1 kHz pure tone presented at 53 dB, the difference of 8 dB represented how much louder a binaural signal appears than a monaural tone at the same amplitude. Therefore, at 1 kHz the binaural difference required for equal loudness (BDEL) is 8 dB; the spectral shape for this example has been included in **Figure 5.1** below from Subject 005's results.

Of the thirty-one (31) volunteers, nineteen (19) were randomly selected to match the loudness levels where the monaural tone was the 60 dB reference; the remaining twelve (12) were alternatively presented with the binaural pair as the reference. For comparison, results of all thirty-one (31) matches have been presented on a common plot in **Figure 5.2** below.

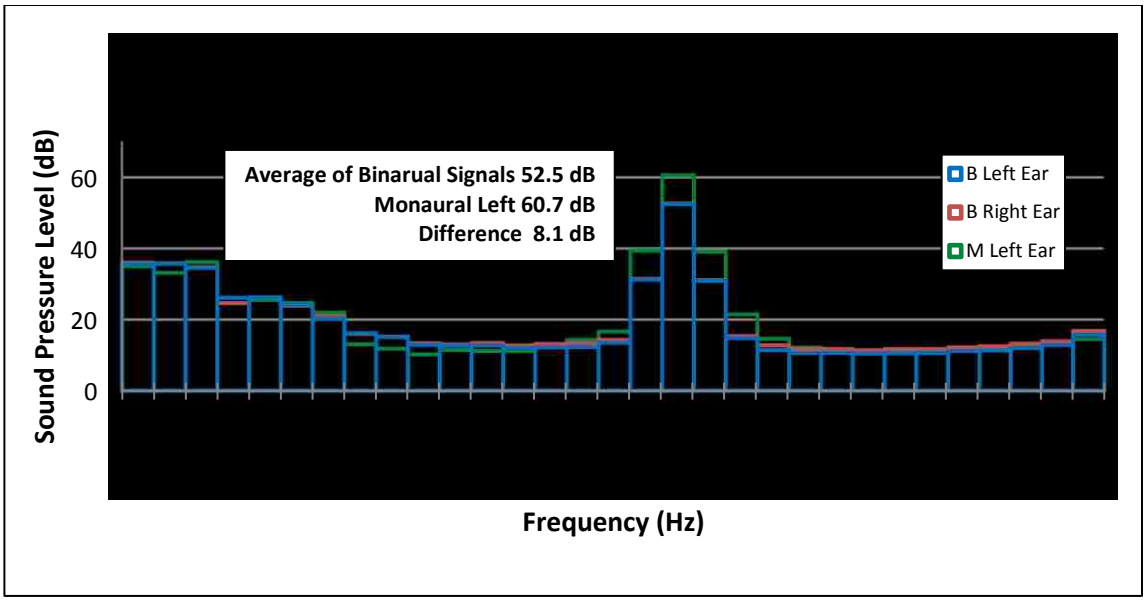


Figure 5.1- Subject 005's response for level difference required for equal loudness (LDEL) at 1000 Hz. Presentation method was constant monaural tone at 60 dB and variable binaural diotic tone (demonstrated by identical left and right signals).

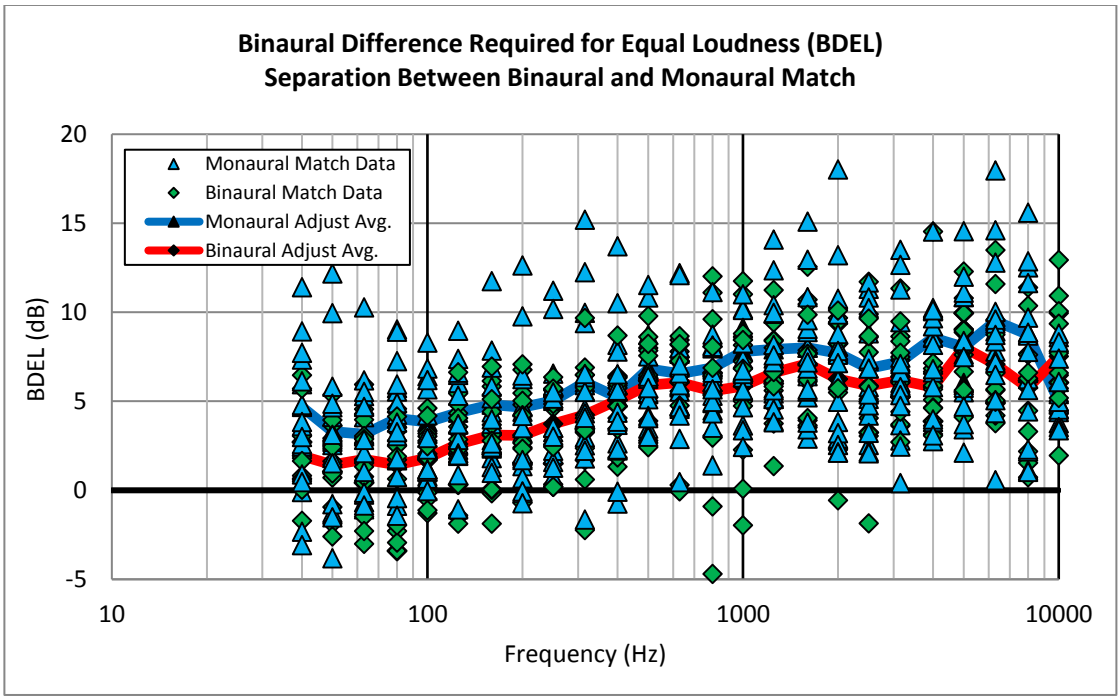


Figure 5.2 - Summary of BDEL results from thirty-one participants of the initial investigation.

Review of **Figure 5.2** indicates a significant spread of data was observed at all frequencies from the loudness matches. When comparing the two presentation methods separately, the average functions appear to yield largely similar results; the **RED** contour indicates an average of the variable binaural matches while the **BLUE** contour indicates the average of the variable monaural matches. The similar trends present between the two contour sets are analogous to the investigation results of Hellman and Zwislocki where the biases associated with each of the two matching methods appear to counterbalance each other; particularly in the mid frequency range, [50]. The difference between the two contours may be explained by a form of "regression" as explained by Reynolds and Stevens, [52]. According to their results, any given matching procedure targeting a function of perception would be strongly influenced by which stimulus the observer is allowed to vary; one presentation method will always be measured at a higher amplitude than the other.

To demonstrate how the data was concentrated around the averages **Figure 5.3** and **Figure 5.4** below summarize the interquartile ranges (IQR) for each data point. A comparison of these figures reveals that more consistent results were obtained for the variable binaural presentation suggesting a better agreement between subjects with this approach. This trend was present at nearly all frequency points as the variable monaural IQR was consistently greater than the variable binaural IQR. The range of values demonstrates this; the variable monaural IQR varied from -4.0 dB to +3.5 dB, while the variable binaural IQR varied from -3.1 dB to +2.3 dB.

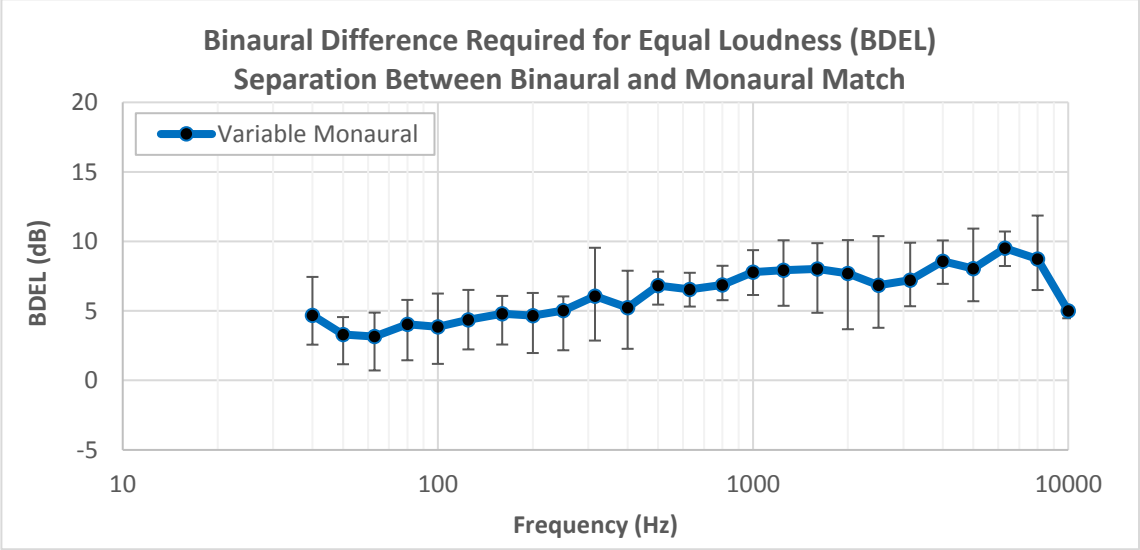


Figure 5.3 – Interquartile range summary of the variable monaural loudness matching results.

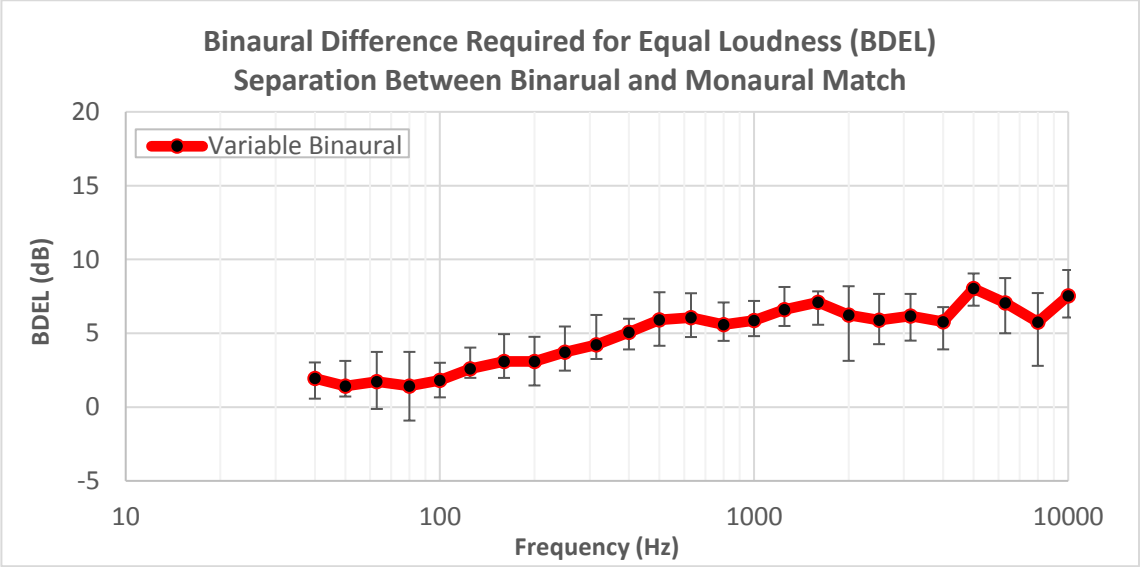


Figure 5.4 – Interquartile range summary of the variable binaural loudness matching results.

The contours present in **Figure 5.2** suggest two important conclusions: (i) the two comparison methods produce similar trends; and (ii) the binaural summation mechanism does exist and appears as an increasing function with frequency, (as opposed to a single constant value). In review of these results the question arose whether this trend may be influenced by the signal amplitude as well, (as **Section 2.6.3C** would suggest). The current investigation compared the binaural difference required for equal loudness against changes in frequency while the amplitude of the signal remained constant as a linear sound pressure level of 60 dB. Consider the fact that the human perception of a 60 dB tone changes drastically across the frequency spectrum. This can be demonstrated by investigating the difference between the threshold of hearing at each frequency and the amplitude of the associated 60 dB reference in each case. According to ISO 226:2003 the threshold of hearing at 1 kHz is approximately 2.4 dB; a 60 dB pure tone at 1 kHz would therefore be perceived as having a 57.6 decibel sensation level, dB(SL) (a marginal difference between the linear amplitude and perceived sensation level at this frequency). Next consider a 40 Hz tone where the threshold of hearing is 51.1 dB. In this instance a 60 dB reference tone would only be perceived as 8.9 dB(SL) (60 dB – 51.1 dB = 8.9 dB(SL)), which is a significant difference between the physical amplitude and the perceived sensation level. **Figure 5.5** below provides a visual comparison for the remainder of the test frequencies where the sensation level of each reference tone (shown as the 'Deviation from Threshold Contour'), has been shown on a common plot with the results of **Figure 5.2** scaled to a separate axis.

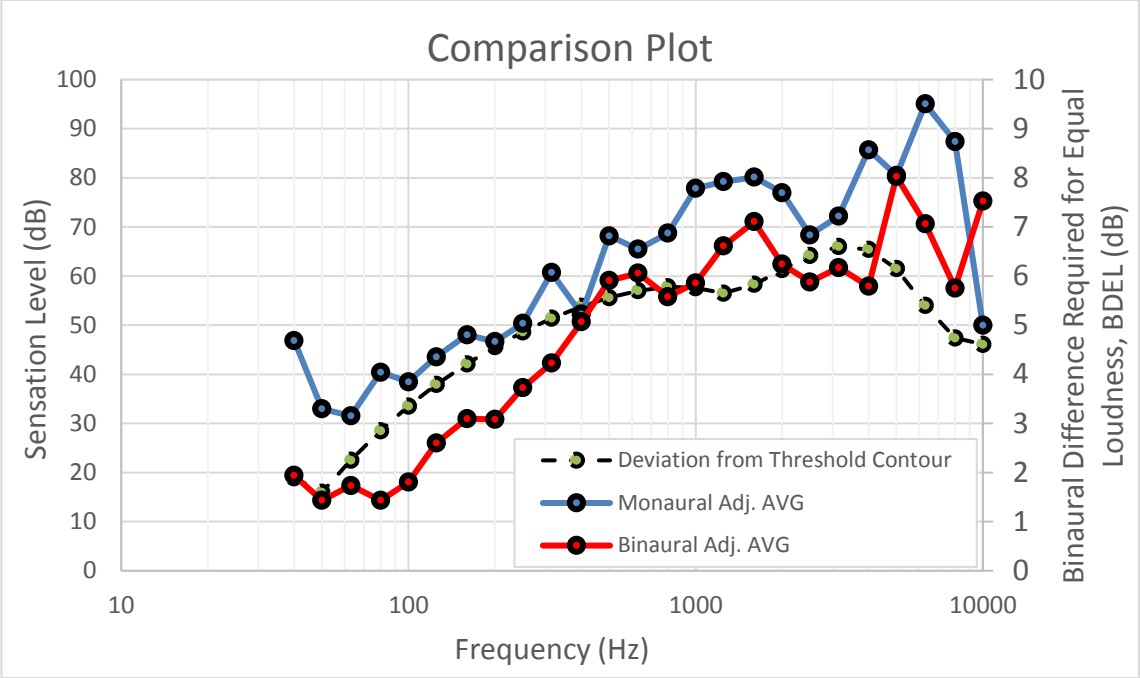


Figure 5.5 - Summation contours with reference to the ISO 226 contour from Figure 3.2 for perfect summation (right axis) normalized and compared against the deviation of the reference signal from the threshold contour (left axis).

If the spectral content of each pure tone was ignored, a quick comparison suggests that the actual trend in binaural summation mechanism closely follows the shape of the deviation from threshold contour. This fact could lead to the conclusion that binaural summation may vary with respect to amplitude as opposed to frequency, depending on the sensation level of a pure tone. Similar to theories in **Section 2.6.3C**, the actual BDEL may therefore vary from approximately 2 dB near the threshold of hearing, up to 8 dB as the sensation level approaches 65 dB(SL). While the similarities between the sensation level contour and the binaural summation mechanism may prove to be coincidental, the relationship warranted further investigation. Based on these findings a second experiment was devised to repeat the above investigation with pure tones of constant sensation level as opposed to constant sound pressure level. The hypothesis in this instance would be to

confirm that the amplitude of a test signal would have no influence on their respective mechanism for binaural summation.

5.2 Constant Sensation Level Investigation

Using two amplitudes of constant sensation level (i.e. 20 dB(SL) and 30 dB(SL)), and the procedure outline in **Section 4.2** above, forty-eight (48) participants matched the loudness of monaural and binaural pure tones through a loudness matching paradigm. The results of this investigation have been summarized into **Figure 5.6** below where, similar to **Section 5.1**, the average of variable binaural matches have been indicated by the **BLUE** contour while the average of the variable monaural matches have been indicated by the **RED** contour. With a larger participant pool the spread of data appears significant while densely populated around the respective averages. Several outliers can be observed; however, the two presentation methods appear closely grouped in the results.

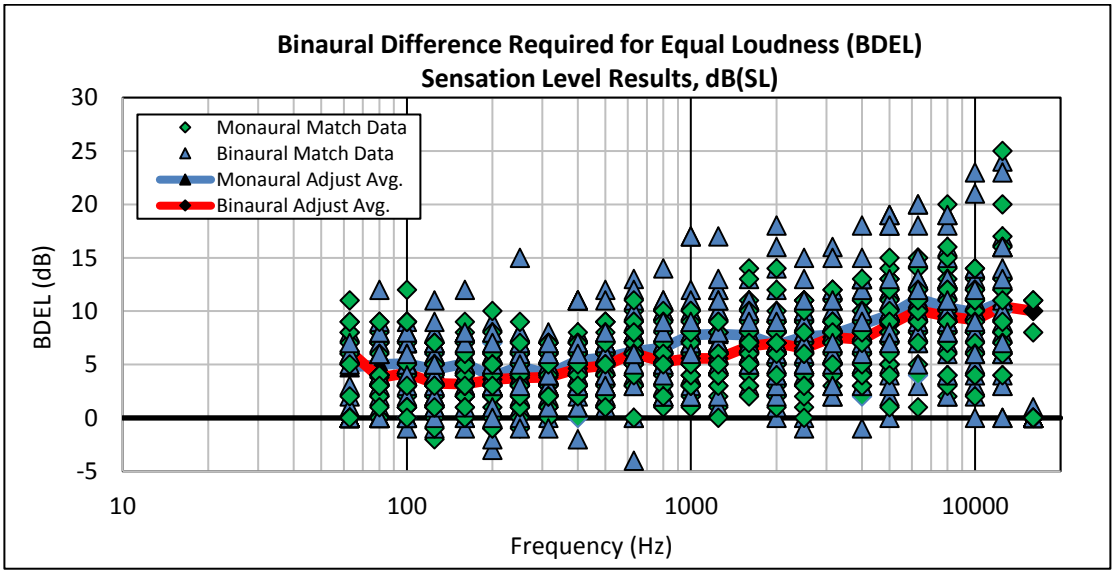


Figure 5.6 - Results of sensation level experiment from forty-eight participants.

Once again the binaural summation mechanism clearly indicated an increasing summation function with frequency. While considerable variation was observed in the amount of binaural summation across listeners, it was recognised that this was common even in the most recent investigations, see [71]. As the participant pool was expanded, the resulting averages suggest increased levels of confidence in the data where similarities between the two presentation methods were much closer than in the previous investigation. To measure the amount of variation, the interquartile ranges were calculated as presented in **Figure 5.7** and **Figure 5.8** below. Once again the variation surrounding the binaural adjust method appeared more condensed (with the IQR varying from -3.4 dB below the average to 2.7 dB), when compared to the monaural adjust method (with an IQR from -4.9 dB to 5.1 dB).

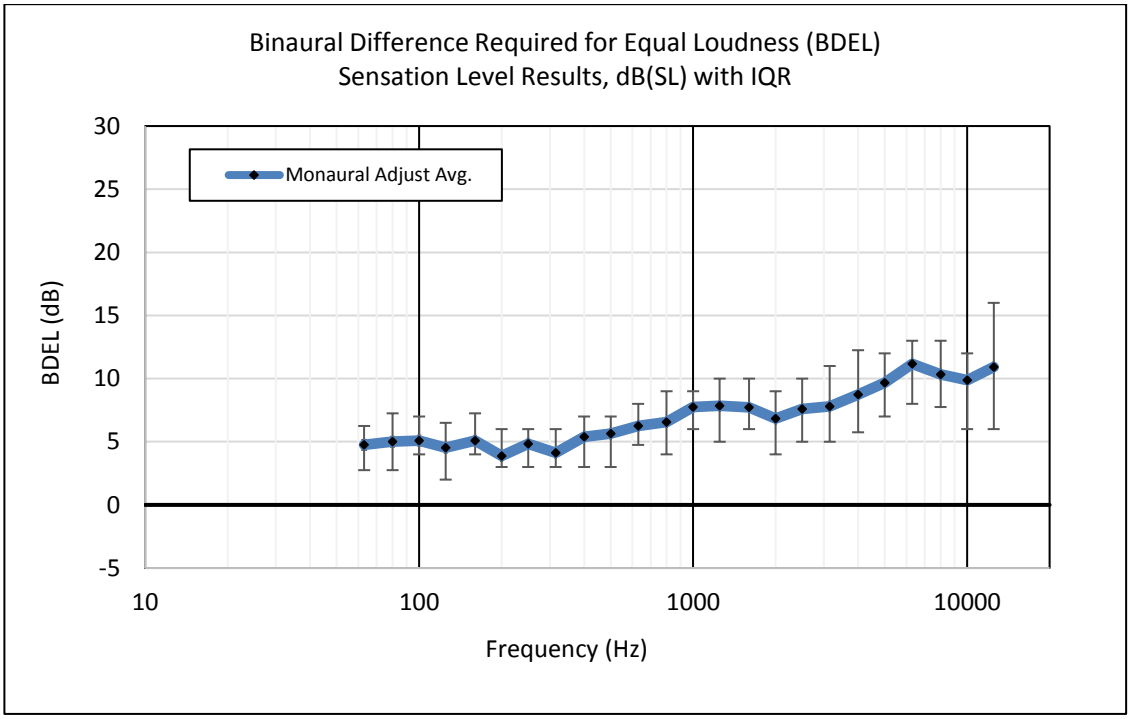


Figure 5.7 – Interquartile range from varying monaural signal.

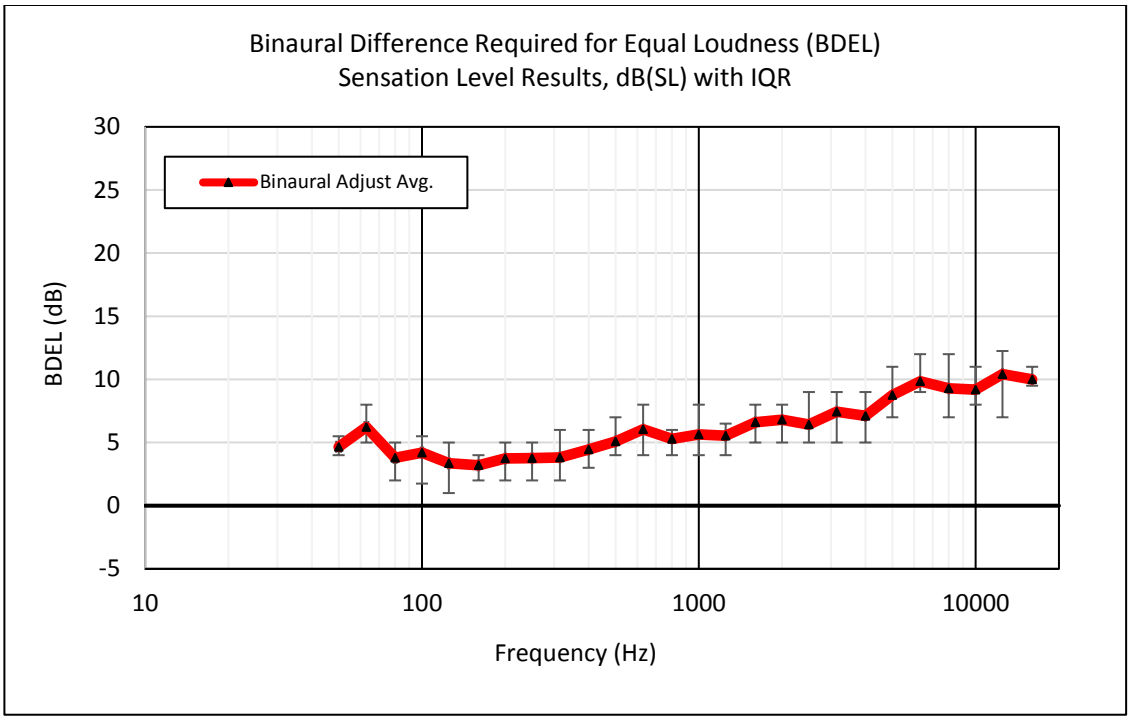


Figure 5.8 - Interquartile range from varying binaural signal results.

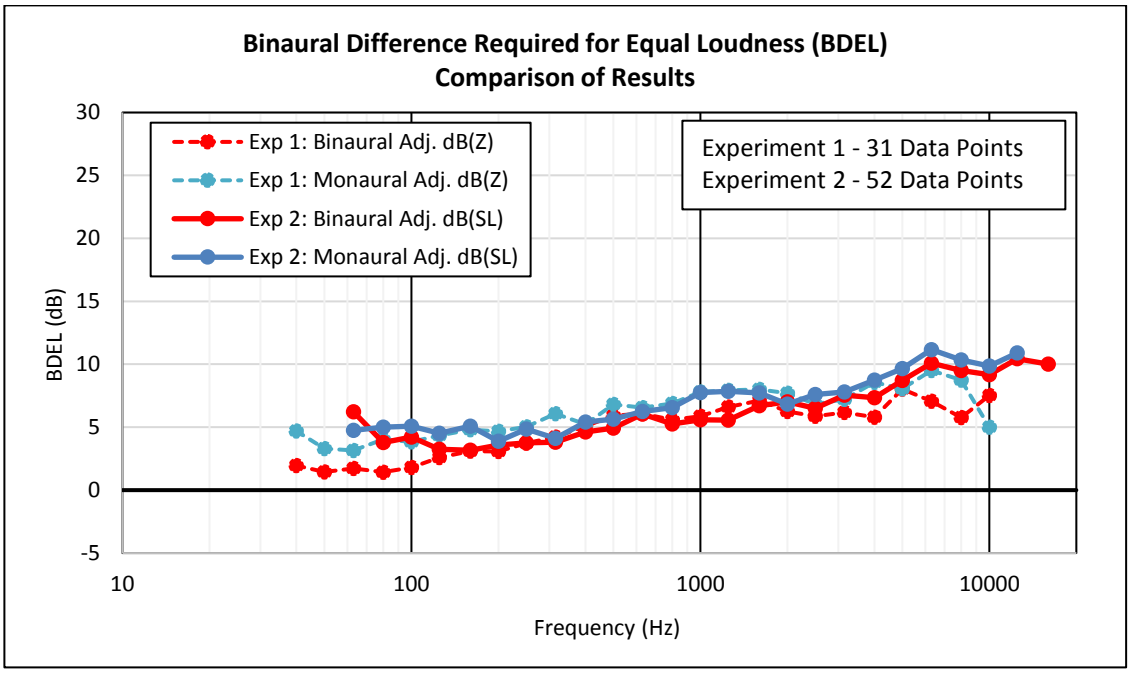


Figure 5.9 - Comparison of results from the first two experiments.

It was useful at this point to compare the results of the second experiment with those from the initial investigation. The comparison greatly increased the confidence in the data as seen in **Figure 5.9**; both sets of data were compared unaltered on a common plot. The agreement between both data-sets is quite good despite the large procedural and experimental differences between the investigations. Recall that for Experiment 1 signals were pre-recorded and presented at constant amplitude via a mixing board while loudness matches were made through a 10-turn potentiometer. Experiment 2 on the other hand was strictly electronic as pure tones were generated during each trial and calibrated to include headphone FRF functions. Additionally, the pure tones in Experiment 2 were amplitude dependent as a sensation level or a constant SPL above the individual's threshold of hearing. Loudness adjustments were then made electronically via the USB keyboard which provided incremental adjustments and removed the errors associated with antiquated hardware.

5.2.1 The Binaural Summation Algorithm

After the review of the sensation level investigation an algorithm for the binaural summation mechanism was derived as a logarithmic function with respect to frequency. The algorithm and the averaged data it was based on have been summarized on a common plot in **Figure 5.10** below.

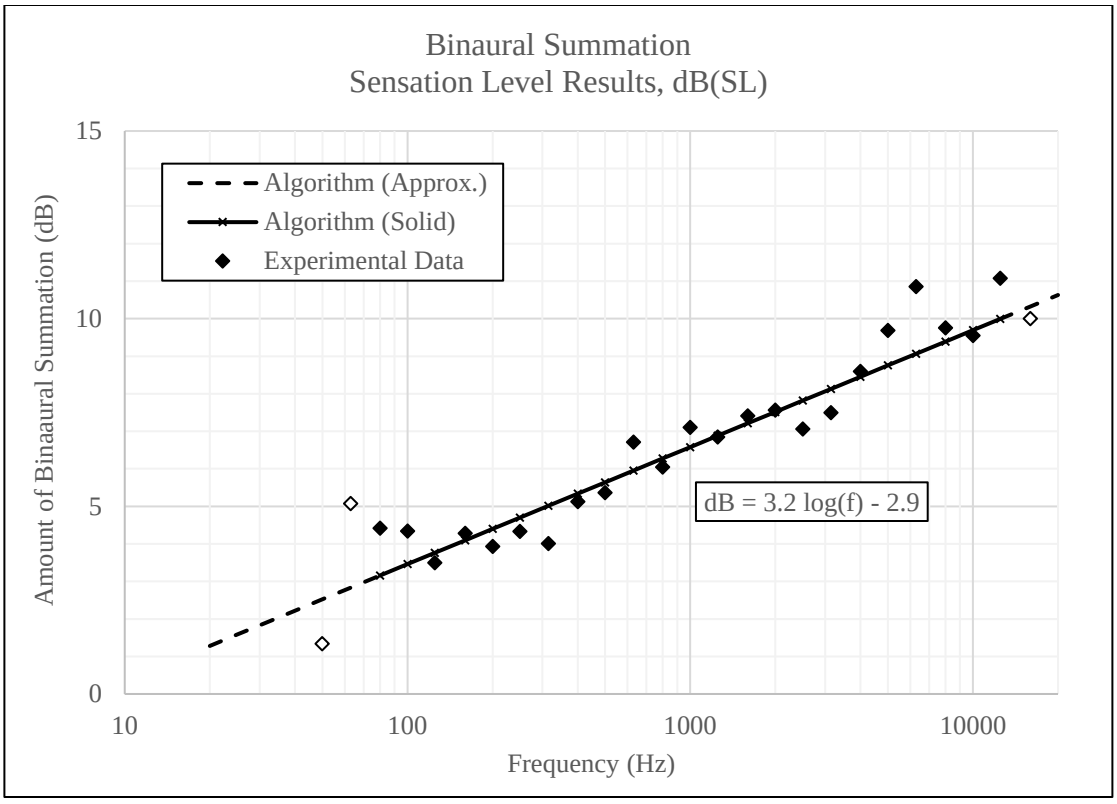


Figure 5.10 - Final summary of experimental data. Binaural summation algorithm to be used for the TVL model.

The derived function shows an accurate representation of the experimental data with an R^2 value of 0.91. Note that hardware limitations and subject sensitivity restricted the amount of information available below 80 Hz and above 12.5 kHz. As such the data beyond these extents was not included in deriving the final binaural summation algorithm. Several curve fitting methods were investigated to match the data including the logarithmic functions as well as multi-order polynomials (see **Appendix C** for a comparison of the results). While the 6^{th} order polynomial was found to have the highest correlation with an R^2 value of 0.97, the upper frequency trends were difficult to extrapolate beyond the experimental data-set as required for implementation into the TVL model. Based on the unknowns associated with the trends at these frequencies it was

determined that the logarithmic function (as previously indicated in **Figure 5.10**) had the most appropriate shape for extrapolating. The caveat being that outside of the range of 80 Hz to 12.5 kHz this function is only an approximation while the rest of the data contains a high degree of confidence.

At this point, it was determined that sufficient information was available to improve upon the existing TVL model. Experimental results confirmed that binaural summation does exist and varies as function of frequency. This information disproves the assumption that a binaural signal is either an average of the two channels or a constant SPL increase over the monaural signal, (i.e. 3 dB for the acoustical energy sum). Next consider the constant ratio summation. Recall from **Figure 3.1** that a doubling of loudness does appear as an increasing function with frequency derived from the equal loudness contours. For comparison with the current algorithm this information has been reproduced on **Figure 5.11** below. For this comparison keep in mind that the summation trends derived from the ISO 226 equal loudness contours are only indicative of the binaural loudness summation for pure tones. For a summation ratio of 2.0, the BDEL was calculated as the difference between the 40 phon contour (1.0 sone) and the 50 phon contour (2.0 sone). Additionally, for this comparison the BDEL contour for a summation ratio of 1.5 (as used in [66]), was derived as the difference between the 40 phon contour (1.0 sone) and the 45 phon contour (1.5 sone).

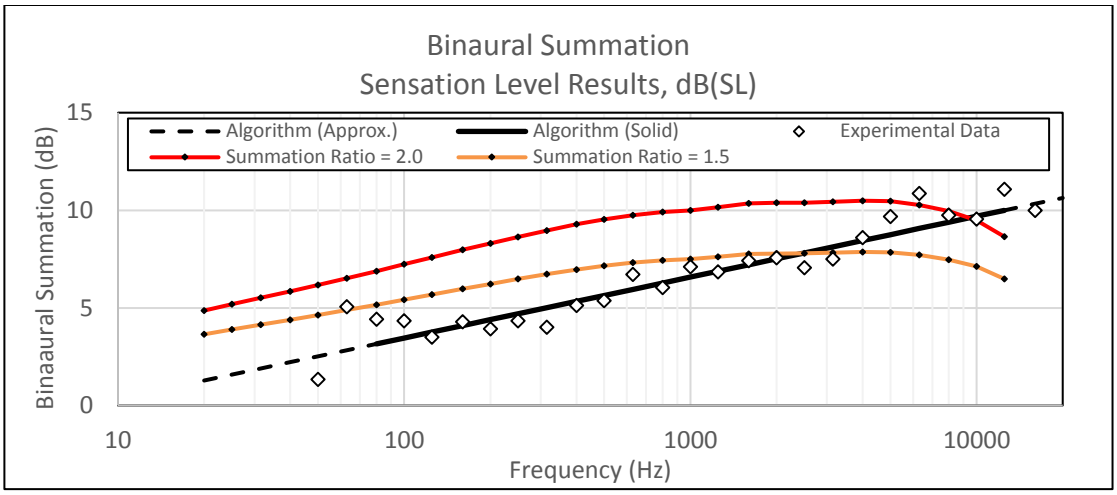


Figure 5.11 - Comparison of loudness algorithm with summation ratios for pure tones.

A review of the contours present in **Figure 5.11** concludes that the current information does not follow a clear summation ratio based on the ISO 226 equal loudness contours. While the current algorithm largely follows the slope of the 2.0 ratio, the amplitudes for BDEL are considerably different between the models. Additionally, from the above comparisons it was clear that the derived summation algorithm was in fact a new contribution to the binaural summation literature. To verify the performance of this novel method, the summation algorithm has been implemented into the TVL metric below where its performance could be verified through further jury test investigations.

5.3 Improvements on the Model

Once a binaural summation mechanism was derived using the above procedures the resulting algorithm could then be implemented into the existing TVL model. Unfortunately, as the original model applied the correction only after the monaural loudness had been calculated, the programmed position of the binaural correction required adjustment as well.

5.3.1 Improvements to the TVL Code

Due to copyright issues, the full code of the Glasberg and Moore TVL model could not be reproduced here. For the purpose of identifying improvements to the model refer to **Figure 5.12** below (altered from **Figure 2.11** above).

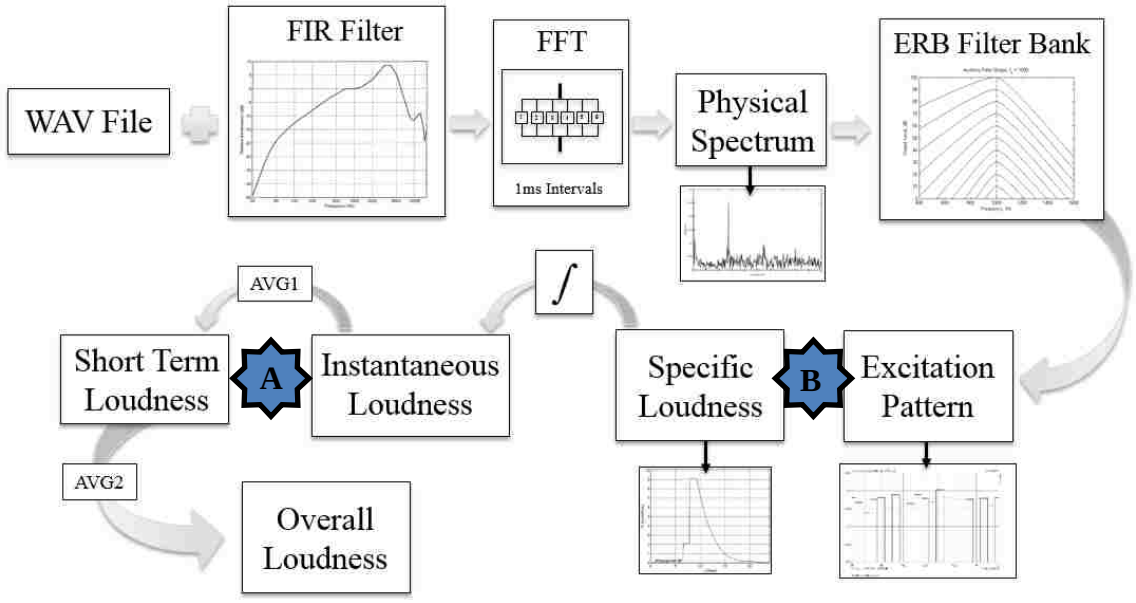


Figure 5.12 - Alterations made to the TVL model,
(Reproduced from Figure 2.10).

As mentioned in **Section 2.4.2**, the original TVL model applied the binaural correction as a perfect sum of the instantaneous loudness; see at '**Position A'** in **Figure 5.12**. To accomplish this, the following line of code was introduced, (see **Equation 6**).

```
if(!monaural) sones = sones * 2.0;
```
… (6)

The C-programming logic used here stated that if the 'WAV File' was **not** identified as a monaural signal, it was assumed that the presentation was binaural and diotic. The resulting loudness was therefore equal to twice that of the monaural loudness,

(i.e. the value in sones was multiplied by 2.0 in accordance with the theory of perfect summation). Recall that the 'sones' value above was calculated as the area under the specific loudness curve and was a value without spectral definition. While the suggested improvements retained that binaural summation was a constant increase regardless of the amplitude, the new summation model recognised that the increase varied with respect to frequency. Therefore, in order to apply the improved algorithm, the correction was relocated to a position where the signal retained the spectral information required for the calculation. To achieve this, the binaural approximation was moved to '**Position B**,' determined to be the most ideal location for the algorithm. At this point the signal still retained spectral information in the format of amplitude per ERB. To implement the algorithm, the previous correction was removed and the if-statement provided by **Equation 7** was inserted at '**Position B**' in the TVL code.

```
if(!monaural)                                          … (7)
       { compdB_1[i] = compdB_1[i] + (3.2*log10(compfq[i])-2.9); }
```

The updated logic stated that if the 'WAV File' had **not** been identified as a monaural signal, it was assumed that the presentation is binaural and diotic. The resulting sound pressure level (in decibels), at each ERB would then be applied as a correction factor, based on the logarithmic equation derived in **Section 4.3**; a constant decibel-increase over the monaural signal which was a function of frequency. Once compiled the performance of the updated TVL model required verification in order to confirm that the performance of the updated summation mechanism was in line with the anticipated results.

5.4 Model Verification and Validation Results

It was determined that the most appropriate manner of presenting the verification test results would be a plot of the loudness difference between the monaural and binaural signals at each amplitude. Recall that as the signal amplitudes were selected based on **Equation 5** above, the ideal resultant of this difference would be zero at all frequencies.

5.4.1 Direct Feed Verification

Manufactured tones indicate the true potential performance of the algorithm when all other factors are removed. In response to the Direct Feed pure tones **Figure 5.13** reiterates how the summation algorithm varied with amplitude as the signal varied across the frequency spectrum.

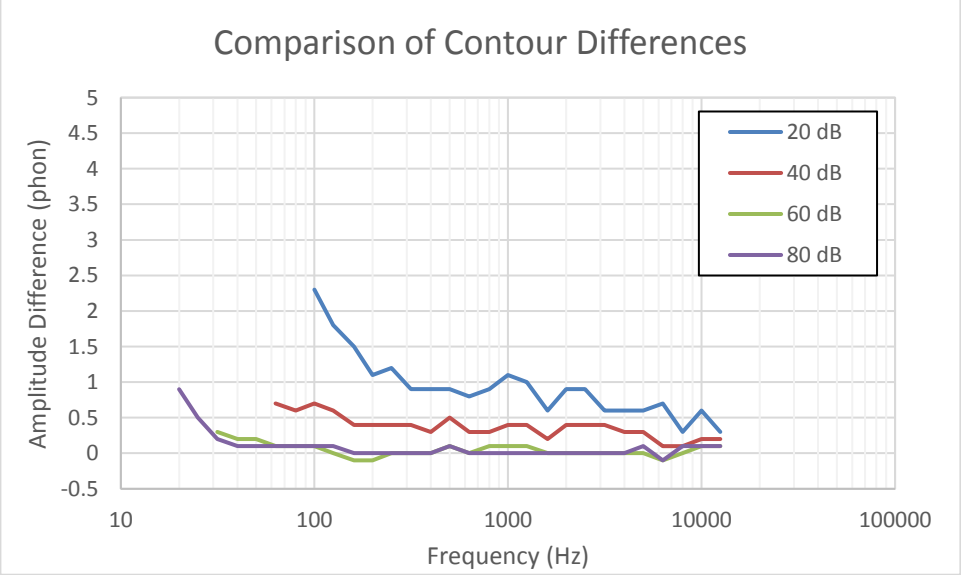


Figure 5.13 - Direct feed results for verification of TVL binaural summation mechanism.

Recall from **Section 5.1** that pure tones of constant amplitude quickly approach the threshold of hearing as the frequency decreases. In terms of the TVL loudness model, the proximity of the threshold contour has a large impact on the predicted loudness level, particularly for binaural signals. During the development of the model the authors included a check-function in the procedure where any ERB segments found below the threshold contour are simply to be ignored. For the current comparison, this was most evident in the response to the 20 dB pure tones in the lower frequencies region. As a result of the improved summation mechanism, adjacent ERB segments which would have otherwise been ignored for the monaural sample become audible once the summation algorithm is applied; adjacent bands which were below the threshold contour are increased to levels above threshold. The result is an increase in the area under the specific loudness contour for all binaural loudness predictions near the threshold. The result of this difference can readily be seen in the low frequency region of **Figure 5.13**.

As the amplitude of the signal was increased, the performance of the binaural summation algorithm closely mimicked the anticipated results. This was most evident in the 60 dB and 80 dB contours where above 50 Hz the difference between predictions were within +/- 0.1 phons.

Regardless of the sensitivities at low amplitudes, the binaural summation mechanism performed as expected with an acceptable level of accuracy for direct feed signals (+/- 2.5 phons within the spectral range of an average human listener). As the direct feed results were mathematically manufactured to exclude external influences, the repeatability of this trial was assured.

5.4.2 Semi-Anechoic Recorded Verification Response

Following the same procedure as the Direct Feed verification, the result from the Semi-Anechoic Recordings are summarized in **Figure 5.14** below.

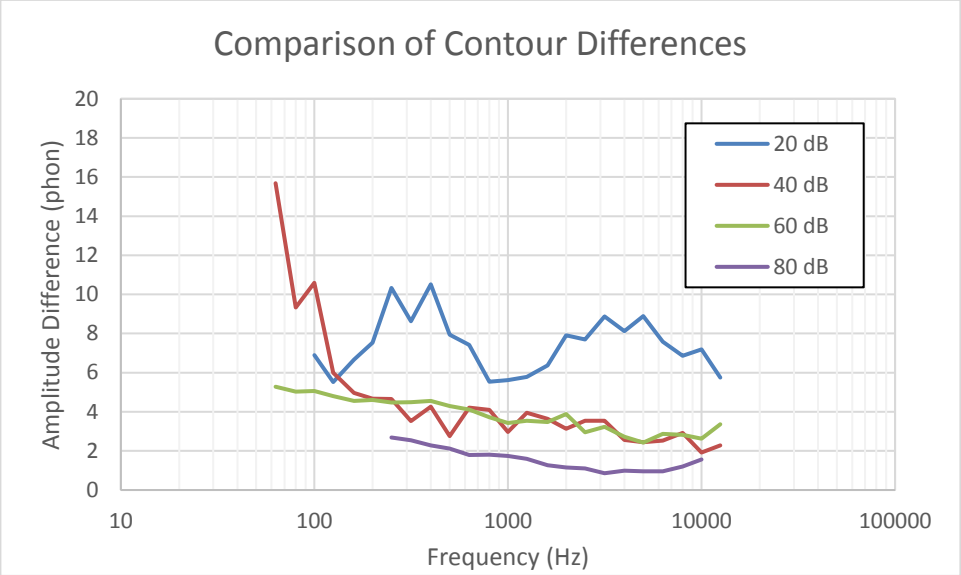


Figure 5.14 - Results of pure tone verification check using semi-anechoic recordings.

Like the results presented in **Figure 5.13** the lower amplitude signals exhibited larger fluctuations with regard to the sensitivity of the loudness model. Unlike the previous investigation however, this instance included the influence of several external factors which could not be controlled. These factors likely resulted in the significant binaural-to-monaural differences exhibited for each of the respective contours. To investigate the source of these errors, a brief signal processing check was conducted as outlined below.

To understand the varied response of the model to recorded pure tones it was important to appreciate how each signal was altered through the presentation process.

During the jury test trials each signal followed a similar acquisition process to calculate the loudness: (i) the electronic signal was produced via the loudness matching program; (ii) fed though the sound card and into the headphones; (iii) converted to an audible signal and recorded via the head and torso simulating device; (iv) transferred back to the data acquisition system as an electronic signal; (v) converted into a WAV file for export; and (vi) resampled for post processing via the TVL program. It was anticipated that during these transitions the signal would be altered slightly. To demonstrate the impact, a 60 dB 1 kHz tone was followed through the loudness process from presentation-to-calculation as shown in **Figure 5.15** below.

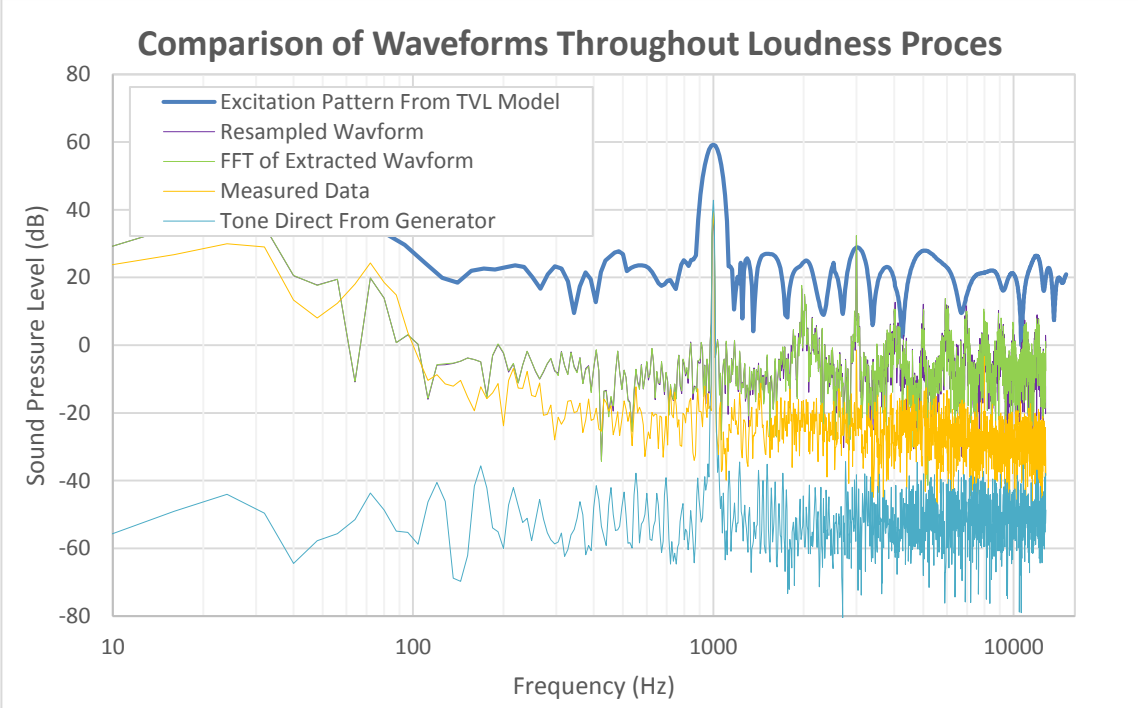


Figure 5.15 - Wavform comparison from different stages in loudness process.

A pure tone produced from the signal generator is free of any distortion as any deterministic sinusoidal wave. It is the goal of any pair of headphones to reproduce an audio signal as near to the original recording as possible. When this is achieved the audio-transducer is said to have 'high fidelity' because of the high-quality components which are used in the design. However, no transducer can be considered 'perfect' as small alterations unavoidably occur during this reproduction. Harmonic distortions are often present which repeat portions of sinusoidal energy at multiples of the fundamental frequency. In the present example, the 1 kHz tone exhibits such energy as echoes observed in **Figure 5.15** at 2 kHz, 3 kHz, 4 kHz, etc. Such harmonics are often unavoidable in acoustics in addition to other non-linear influences such as the ambient noise spectra which strongly influences the results. In the present case, all of these alterations resulted in the spectra produced from the generator (**light blue contour** in **Figure 5.15**), being converted into the much higher spectra recorded via the HATS transducer (**orange contour** above).

To apply the TVL model to the waveform, it was then necessary to convert the recorded DAT sample into a WAV file using the PULSE post-processing software. As with any conversion process this altered the waveform once again and introduced additional noise features while amplifying the existing harmonic distortions present (**green contour** in **Figure 5.15**). Lastly, as per the TVL software requirements, the waveform required resampling from 65.5 kHz to 32 kHz; this process only marginally altered the waveform as the contours overlapped in **Figure 5.15** (see the **purple contour** which is shielded by the **green contour**). For informational purposes the excitation pattern

was included in the figure to demonstrate how the TVL program extracts pressure levels from the waveforms as a measure of sound pressure per ERB.

From this investigation, it is likely that in addition to ambient noise levels, the small fluctuations present in **Figure 5.13** (from the Direct Feed response), were amplified as the signal was recorded in a semi-anechoic environment and converted to a useable format. It was evident that the extraction process had the greatest impact on the calculated resultant. With the inclusion of significant peaks after the conversion froma DAT file to the WAV file, it was clear that this process had a negative impact on the resulting verification. The unavoidable process was considered insignificant in this review as the ideal implement of the TVL model would be a direct calculation on a recorded waveform, removing any need for this extraction to take place. Further, the influence of threshold corrections (as observed in **Section 4.3.2**) were likely exemplified in the recorded response as the ambient SPL closely followed the threshold contour (as observed in **Figure 4.8** above).

Similar to the previous test, the sensitivity drastically improved as the amplitude of the signal was increased. This can be attributed to the spectral masking associated with higher amplitude signals which would overshadow unwanted ambient noise and the resulting by-product of harmonic echoes. While not nearly as focused as the Direct Feed results, the Recorded Response investigation indicated ± 6 phons for pure tones greater than 40 dB and 100 Hz; an acceptable range of loudness variation particularly considering the accuracy of previous results. For each of the amplitudes investigated statistical details of the response have been included as **Appendix B.2**. In summary, the standard deviation for each of the amplitudes investigated varied from a maximum of 6.0 phons for the 20 dB

contour down to only 0.1 phons for the 80 dB contour, (see **Appendix B.2** for complete details). With the sensitivity confirmed, the above analysis verifies that the improved TVL model is functioning as intended to provide improved binaural summation.

5.4.3 Jury Test Validation

Ten (10) volunteers took part in the verification test including nine males and one (1) female, all between the ages of 23 and 43 years of age. Each of the volunteers had experience with loudness matching experiments and had previously taken part in the sensation level investigation outlined in **Section 4.2** above. The following figures summarize the binaural matches per each participant. For comparison, both the 'Original' TVL model (with the theory of perfect summation) and the updated TVL model 'TVL-JC' have been presented on a common plot. Keep in mind that the closer the response was to zero (highlighted as a grey horizontal line), the more accurately that model corresponded to the participant's response.

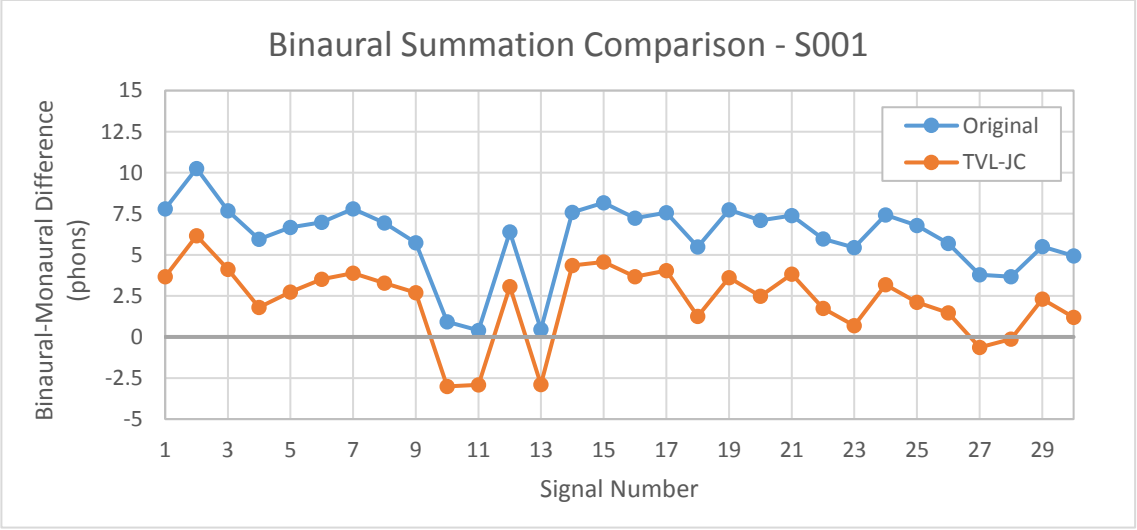


Figure 5.16 - Verification response of subject S001

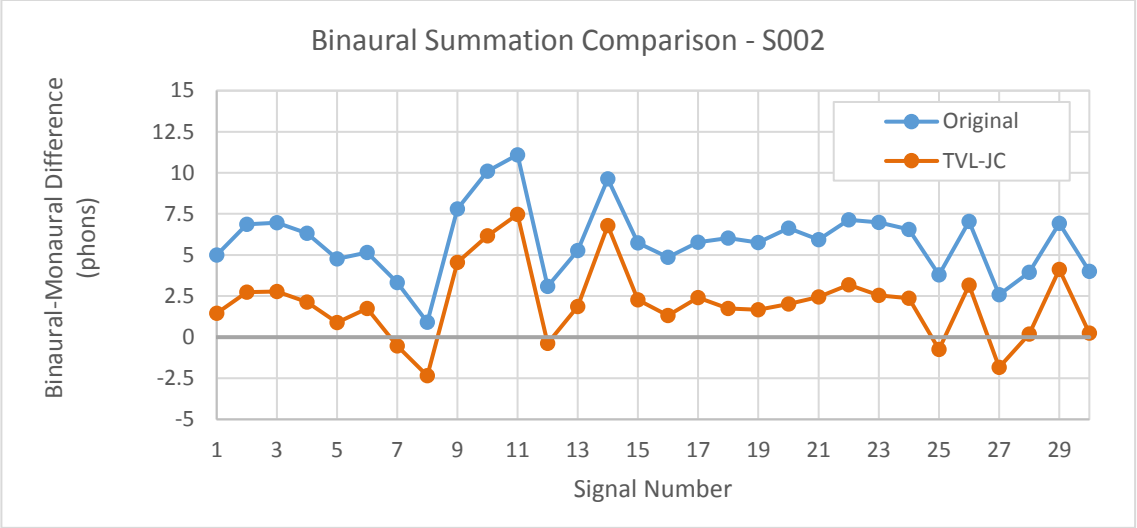


Figure 5.17 - Verification response of subject S002

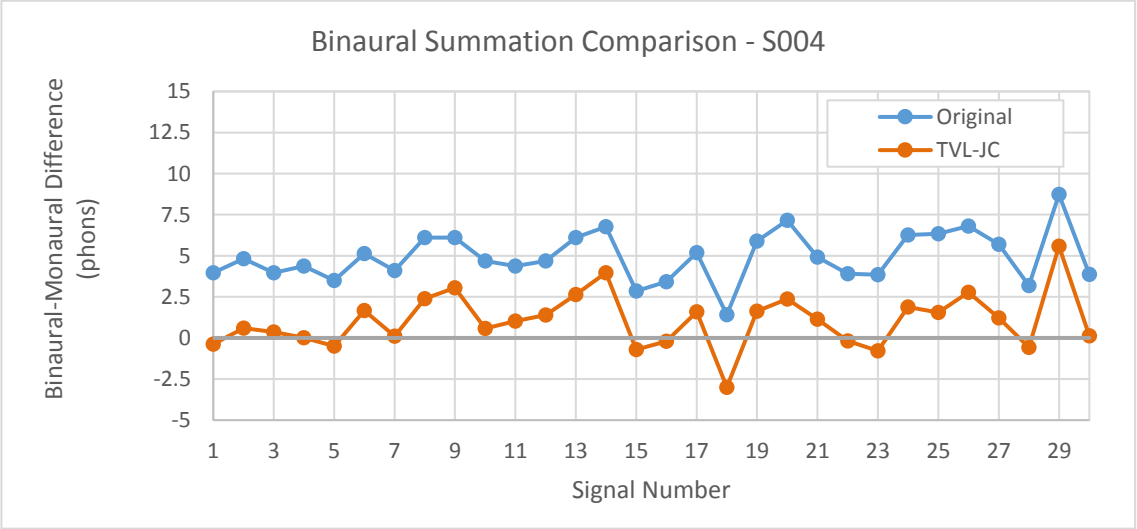


Figure 5.18 - Verification response of subject S004

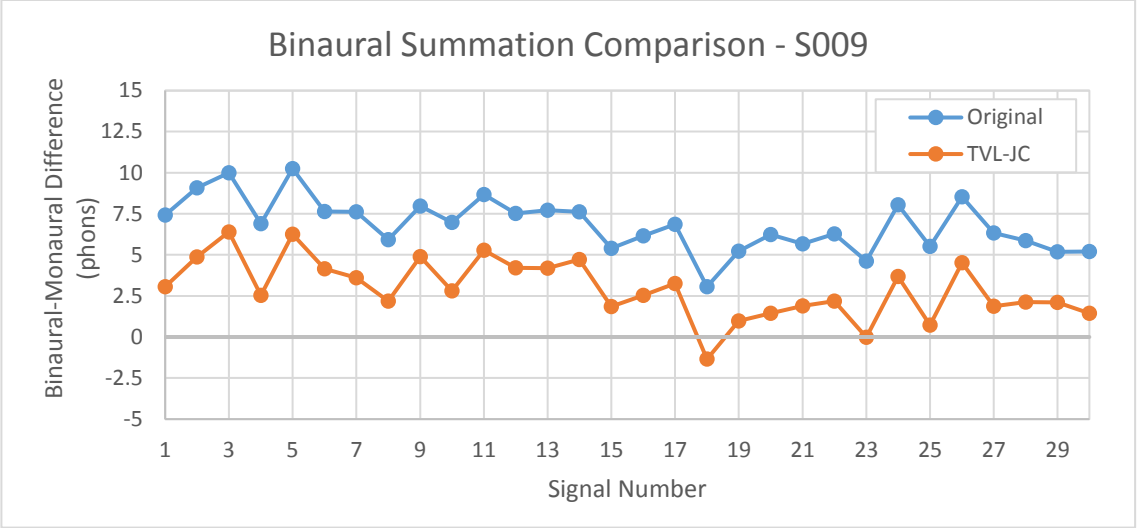


Figure 5.19 - Verification response of subject S009

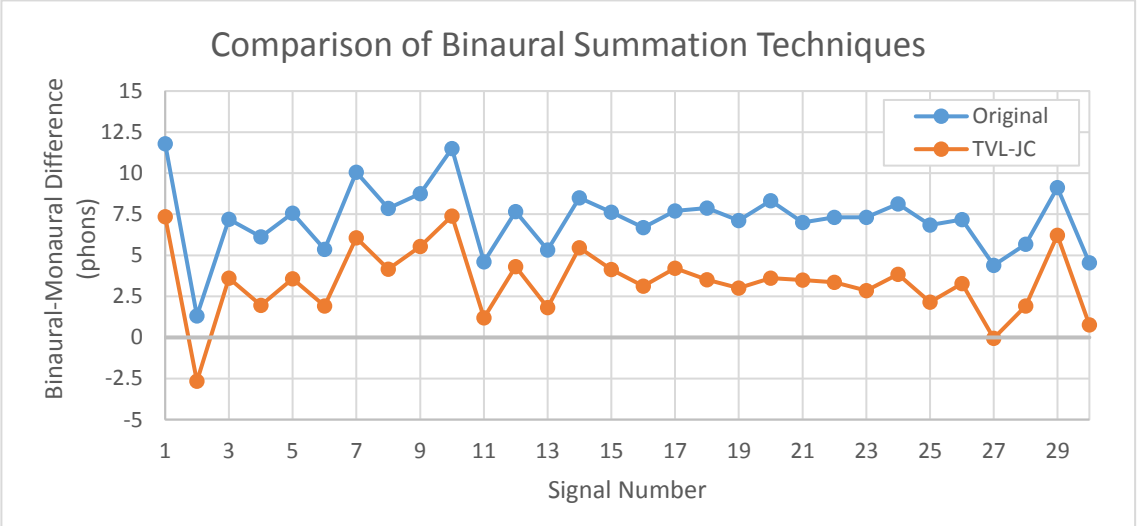


Figure 5. 20 - Verification response of subject S018

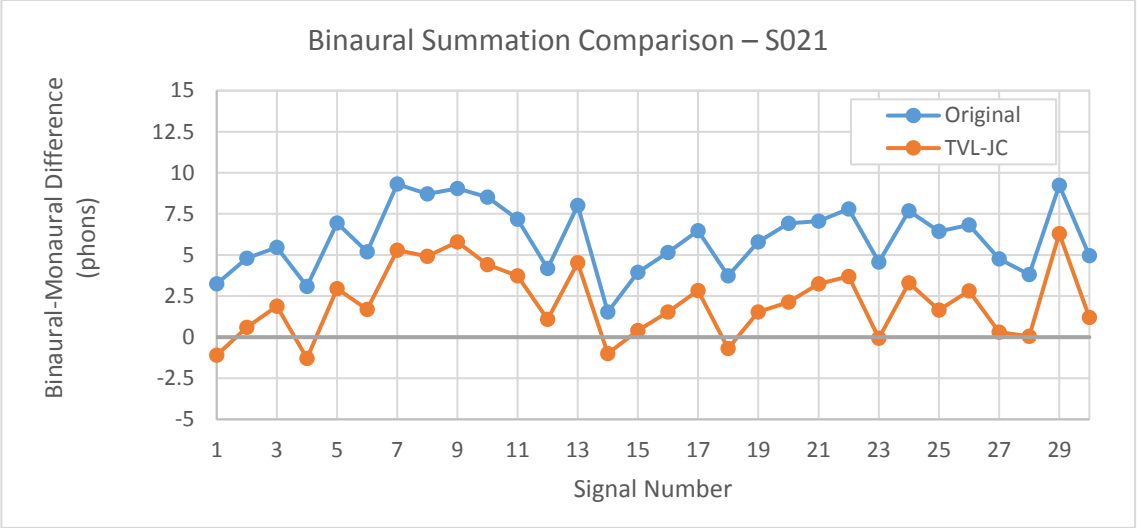


Figure 5.21 - Verification response of subject S021

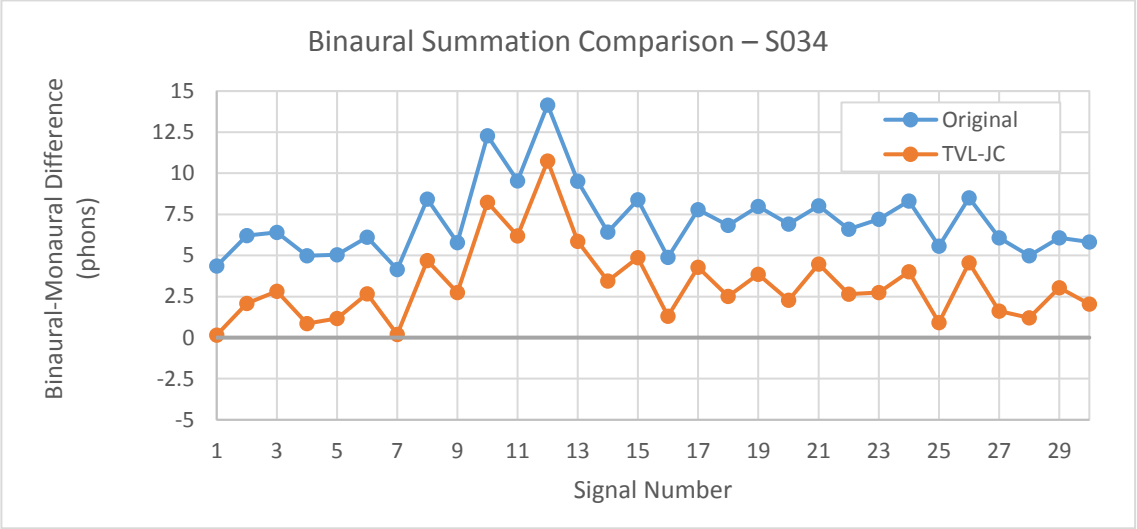


Figure 5.22 - Verification response of subject S034

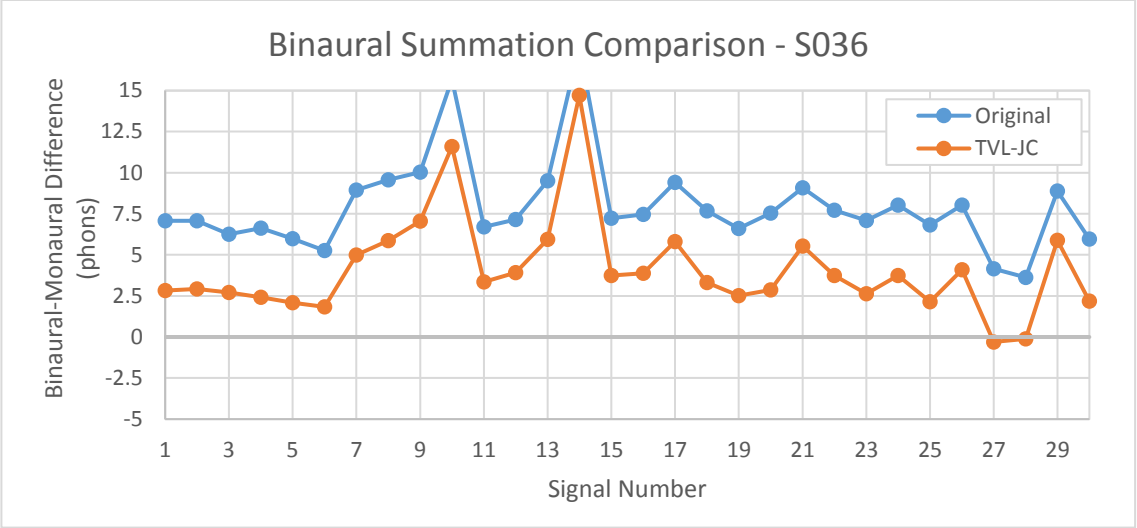


Figure 5.23 - Verification response of subject S036

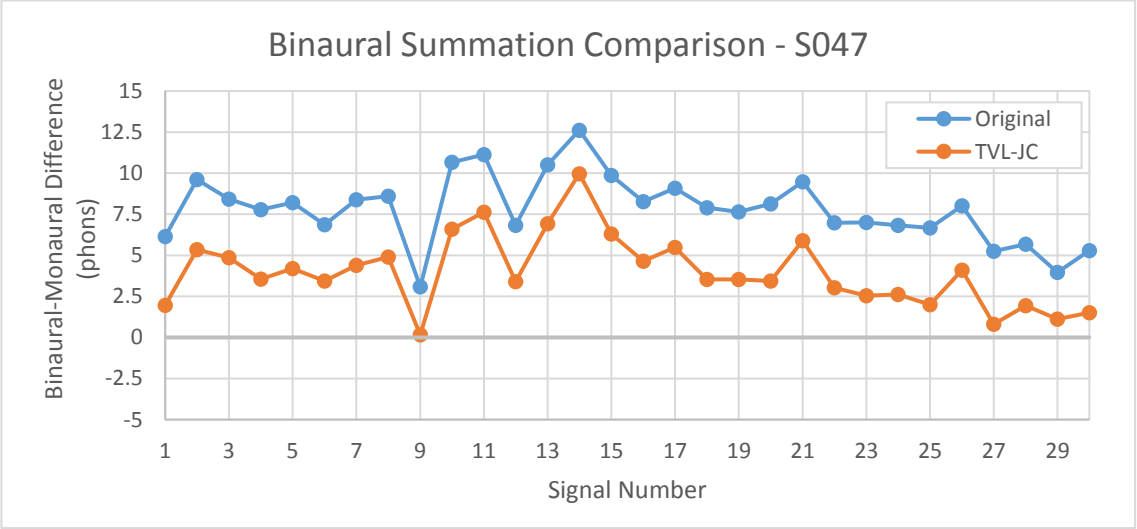


Figure 5.24 - Verification response of subject S047

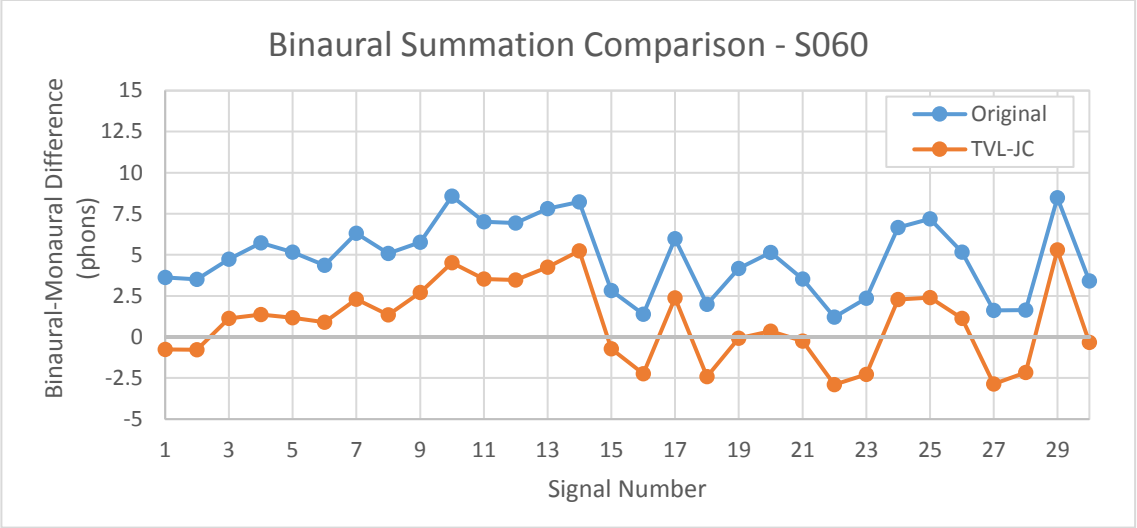


Figure 5.25 - Verification response of subject S060

A review of participant results (**Figure 5.16** though **Figure 5.25**), reveals several important observations; most importantly that the updated TVL model correlated better with the jury participant results than the original TVL approximation. On average the updated TVL model predicted loudness levels which were 3.8 phons closer to the observed responses than the original. Secondly, while the results of each participant varied markedly (i.e. the amount of summation was not consistent between participants), it appeared that the updated algorithm produced results which were consistently offset from the original TVL model results. To investigate this further the difference between the updated algorithm and the original metric was compared in **Figure 5.26** below.

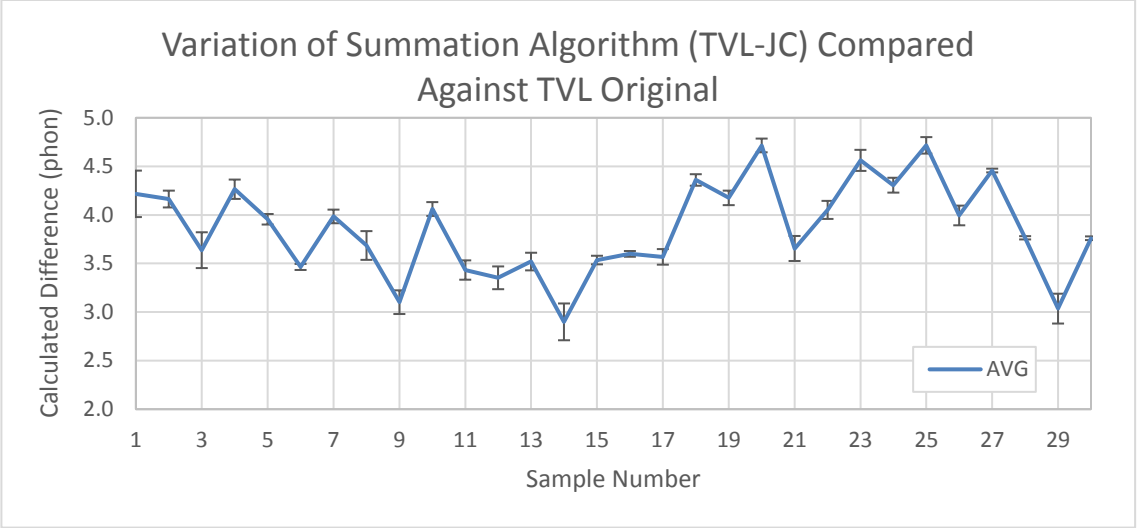


Figure 5.26 - Calculated difference between the original TVL model results and the updated binaural summation algorithm.

Regardless of the amount of summation the participant indicated, **Figure 5.26** confirmed that the difference between the two models was consistently the same (with a maximum standard deviation of only 0.10 phons across the signals). Additionally, the updated algorithm was consistently 3-5 phon lower than the standardized method.

To ensure that the current findings were an improvement over the most recent information, a secondary investigation was carried out for direct comparison towards the 2007 conclusions from Moore and Glasberg concerning binaural summation, (see [66]). Recall from **Section 3.1.3** that their recent investigation revealed a binaural to monaural summation ratio of 1.5 for diotic listening conditions using the stationary loudness model. Using the original TVL model as a base, this updated correction factor was inserted into the model and the participant results were re-examined. **Figure 5.27** below compares all three loudness models on a common plot using S002's response.

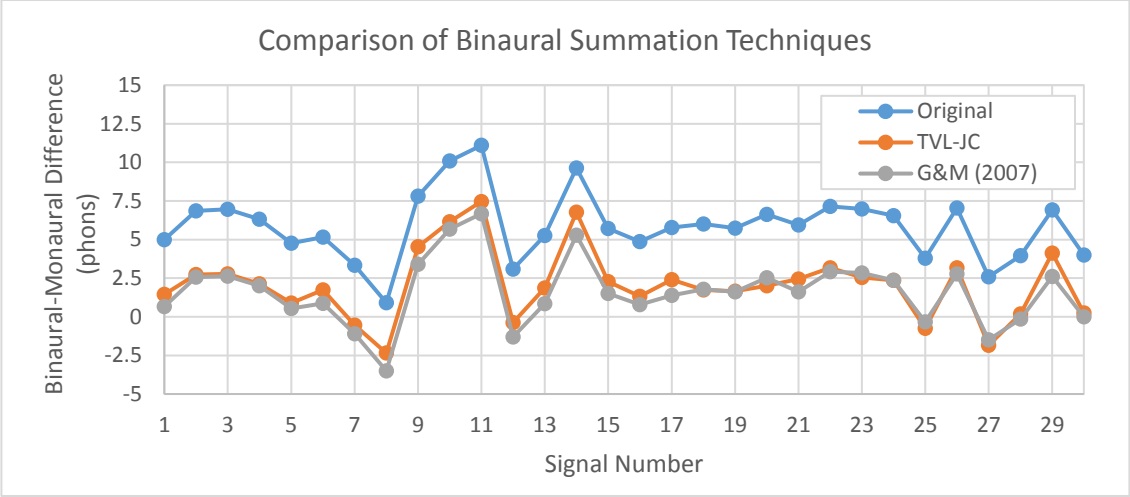


Figure 5.27 - Comparison of loudness model results for participant S002's response.

From **Figure 5.27** it was clear that the binaural summation mechanisms of the updated TVL model (TVL-JC) and the Moore and Glasberg 1.5 ratio (G&M (2007)) produced nearly identical results. Further investigations revealed that these similarities were observed for nearly all participants in the follow-up investigation as summarized through averaged results in **Figure 5.28**.

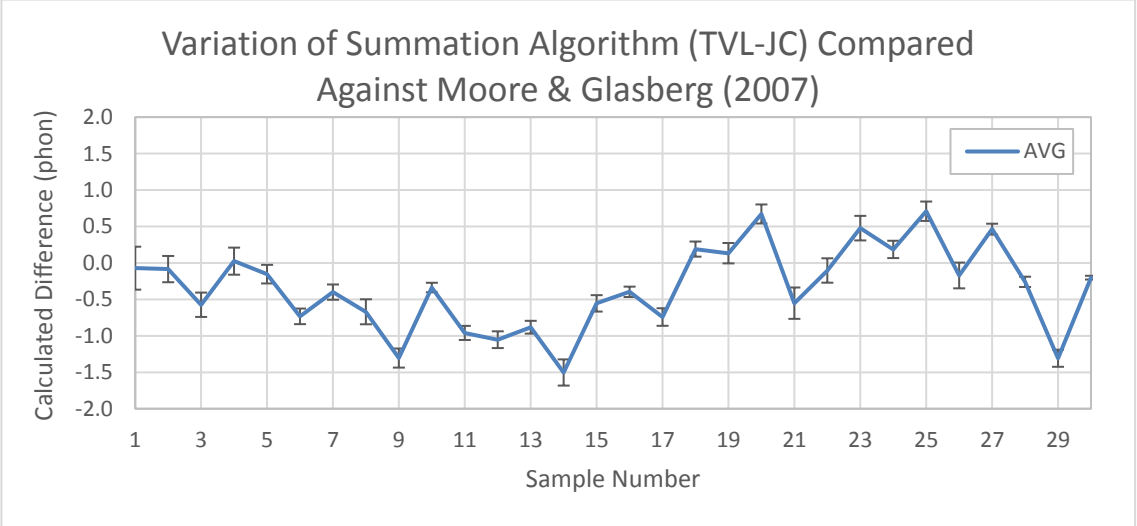


Figure 5.28 - Averaged comparison of updated binaural summation algorithm compared against the Moore and Glasberg binaural to monaural ratio model.

After reviewing **Figure 5.28** it was clear that the two loudness models approximate the same information. This was surprising as the two algorithms were based on entirely different calculation methods. The new algorithm applies a correction based on a sound pressure level increase as a function of frequency. Alternatively, the Moore and Glasberg approach was applied only after the monaural loudness had been calculated as an area under the specific loudness contour. Despite this, the results of the two algorithms are within +/- 2 phons of one-another.

The data in **Figure 5.28** represents a direct difference between updated TVL model and the Moore and Glasberg 2007 approximation, a negative value indicates that the updated model predicted a higher value while a positive value indicates the opposite. Based on the signal content from each sample, this review suggested that for stationary and quasi-stationary signals the updated model consistently produced a higher estimate of the loudness than the Moore and Glasberg approximation. Alternatively, for the 'real' signals identified as samples 16 through 30 (as opposed to fabricated ones), the updated model appeared to produce a lower response. Despite one being greater than the other, the difference between the two estimates was well within an acceptable margin of error.

The data presented in this chapter has demonstrated that the updated binaural summation algorithm was a considerable improvement over the state of the art approach for predicting the perceived binaural loudness. The improved function retained the ease of use from the original model and allowed for additionally accurate estimates of diotic listening conditions from monaural WAV files. It has been clearly demonstrated that the

revised time-varying loudness model is a significant improvement over the previously standardized approach. As additional discussion, the following chapter will review any outlying uncertainties and sources of error encountered during this development as well as any remaining limitations.

VI. UNCERTAINTY ANALYSIS AND SOURCES OF ERROR

As with any experiment, the above procedures were subject to several unavoidable uncertainties and sources of error. Once identified, efforts were made to control the known variables as outlined in the following subsections. Before the final conclusions were drawn, the remaining sources of error and the limitations of the model which could not be controlled were summarized thereafter.

6.1 Uncertainties through Participant Response

When dealing with jury test measurements, Robinson and Dadson made note of three significant sources of variance, namely: (i) deviations between participants; (ii) measured variance between each individual's consecutive observations; and, (iii) the variance resulting from the order of presentation, [10]. All three of these factors were likely present in the above research and have been accounted for using the following considerations.

6.1.1 Variance Amongst Persons: Participant Pool Size

The results of Robinson and Dadson confirmed that the largest variance was observed when comparing the results between participants (i.e. 89% of their total variance), [10]. In the current investigation, this dispersion was accounted for through a large sampling of the population and the resulting average from the data. By definition, loudness is a subjective quantity which strongly depends on an individual's opinion. As such, it was determined that a large participant pool was required to accurately represent the binaural summation mechanism for an average human listener. To determine if the number of participants selected was in fact large enough to represent the population, a sensitivity check was performed on the data. During this assessment, the experimenter

posed the question: ‘What happens when 10 random data-sets are removed from the average?’

To answer this question a MATLAB program was written to examine the variance through 100,000 iterations. Where data was limited between 63 Hz and 12.5 kHz, the resulting average varied between 1.2 dB and 3.2 dB after removing 10 random data-sets through the iterations. These results confirmed that had the study only included 40 participants, the resulting average function would have closely correlated with the current trends, (i.e. within ±3 dB). This was considered an acceptable amount of uncertainty given the small spread of data. The fact that the study was expanded to include 50 participants improved upon the confidence in the results considerably. Based on this analysis, the results confirmed that additional testing beyond the participant pool size of 50 test subjects was unnecessary and the variance between participants had been taken into account. Details regarding the sensitivity check algorithm and complete results from the application have been included as **Appendix D**.

6.1.2 Variance Between Consecutive Observations

The second largest contribution to variance was determined to be the likely shift in response in consecutive measurements from the same observer. Robinson and Dadson attributed 9% of their overall variance to this factor, [10]. Ideally, consecutive measurements would produce identical results; however, many factors influence the perceived loudness for an individual including alertness, state of mind, recognition and listener fatigue. With so many variables it would be nearly impossible to completely define the variance from a participant pool without repeat trials from the same observers. The current investigation accounted for these irregularities in a manner similar to Fletcher

and Munson's approach towards interaural differences. It was assumed that the resulting average from a sufficiently large number of listeners would overshadow the small variations from each observer individually, [72]. Additionally, each participant was permitted to position their own headphones during trials, the literature confirmed that this practice sufficiently reduced the spread of consecutive observations, [99].

6.1.3 Variance Resulting from Presentation Order

The final measure of variance observed by Robinson and Dadson was the impact of presentation order during any loudness matching paradigm. Recall from **Section 3.1.2** that the best method of accounting for and reducing the amount of variance from presentation bias was found to be the balanced presentation method identified by Hellman and Zwislocki, [50]. Through procedural reversal for half the participants the bias associated with either presentation method was successfully counterbalanced when compared to the other. Review of **Figure 5.2** and **Figure 5.6** confirmed this theory.

6.2 Additional Sources of Error

Unfortunately, not all sources of error could be completely controlled. A complete review of these issues was essential for ensuring that all sources of error that could be addressed were given the appropriate attention. The remaining uncontrollable sources have been summarized as follows.

6.2.1 Headphone Use and Loudness Matching Paradigms

The approach used in the present study was a monaural-to-diotic binaural comparison method presented over headphones. Critics of this approach rightly argue that this form of listening rarely occurs in the natural environment. When presented over

headphones the response creates a fabricated state which appears to be localized from inside the listener's head. Literature confirmed that this scenario was difficult to avoid, for example see [106]. The results raised a question regarding the appropriateness of using headphones during loudness investigations, specifically should an internally localized signal be used for the derivation of binaural loudness summation?

Epstein and Florentine investigated this question where it was shown that the amount of summation was lowered when the signal was presented via loudspeaker in place of headphones, [107]. During their investigation, a monaural loudness scenario was approximated using a monaural earplug capable of a 20 dB attenuation. While a direct comparison between the two listening conditions was difficult, the findings of Epstein and Florentine suggested that more information may be necessary. For the purpose of the current investigation, it was decided to continue with a strict headphone comparison approach given the difficulty of achieving a true monaural-to-binaural presentation using a loudspeaker. This decision was made with the understanding that the actual summation function may be lower than predicted based on Epstein and Florentine's results.

6.2.2 Consistency During Post Processing and Extraction

Each match of equal loudness was monitored and recorded via the head and torso simulator located inside the semi-anechoic chamber. The resulting DAT file provided a 12-second sample of the alternating signal presentations from each matched pair. As each presentation comprised a 2 second signal followed by a period of silence, each DAT file therefore included two iterations of each monaural and binaural presentation. In order to post-process this information, each monaural and diotic signal set was extracted from the sample before the respective binaural summation conclusions could be drawn.

During post processing, it was observed that selecting the identical time signatures for both the binaural and monaural signals was nearly impossible. During the verification procedure, this was particularly important given the temporal characteristics of the signals as the resulting loudness could vary unless similar time segments were used. To avoid this scenario, considerable effort was made to sample and resample each data point while ensuring the appropriate time signatures were being investigated. Unfortunately, this was an unavoidable source of error given the current experimental set-up. Future investigations should examine methods of repeatability for data-collection in order to limit this potential source of error.

6.2.3 Discussion of Overall Uncertainty Results

While several points of uncertainty and sources of error have been identified above, a considerable amount of confidence remains in the findings due to the overlapping procedures, verification of results and measures taken to reduce the variance in the data. Based on this analysis, it was determined that the results of the binaural loudness investigation were accurately depicted based on current information.

6.3 Limitations of the Model

The current investigation improved the TVL loudness metric through a revised estimate of the diotic case of binaural loudness. While the improvements enhanced the accuracy of predictions, several known mechanisms of loudness still remain unaccounted for from limitations in the procedure. The most influential of which was the monaural WAV file restriction which limited predictions to either a monaural loudness estimate or the binaural approximation as a diotic listening condition. As with the comparisons via headphones, critics of the diotic approach are quick to point out that a true diotic signal is

rarely encountered in natural environments. To improve upon the algorithm, this limitation must first be removed to permit stereo signal predictions from two-channel HATS measurements. Only after this restriction has been removed will complex loudness predictions be possible including off-angle listening, localization, and binaural inhibition. The following chapter will include recommendations to investigate these mechanisms further and revise the model.

VII. CONCLUSIONS AND RECOMMENDATIONS

The following review summarizes the observations and conclusions drawn from the above work with reference to the previously stated objectives for this research. It also identifies how the contributions provided by this work can enhance the field of psychoacoustics through the development of more meaningful loudness metrics. Finally, several recommendations will be provided for future work in this field to further refine this important descriptor.

7.1 Conclusions

Upon reviewing the results of this study the following is a presentation of conclusions that have been reached:

1. The initial purpose of this study was to verify the existence of binaural summation through an independent jury test investigation. Using a loudness matching experiment and pure tones, it was verified that binaural summation does in fact exist and is not a simple averaging or addition of the loudness from the left and right ears.
2. Results of a follow-up investigation using pure tones of constant sensation level confirmed the previous results. The binaural summation mechanism was determined to be a function that varies with frequency as an increasing logarithmic trend. Once derived, the binaural loudness algorithm could readily be adapted into existing loudness metrics for accurate predictions of diotic binaural loudness using monaural WAV files as an input.
3. To demonstrate this, the resulting algorithm was successfully implemented into an updated Time-Varying Loudness (TVL) metric using the Moore and Glasberg's original code as a base. The updated TVL model operated in a similar manner to the

original program while revising the summation location and the form of the binaural approximation.

4. The results of the updated TVL model were verified with measurements of computer generated pure tones as direct feed and recorded signals in addition to actual binaural summation estimates from a series of human listeners.
5. It was confirmed that, as a result of the revised algorithm, a more accurate representation of the binaural summation had been achieved. The updated TVL model results were consistently closer to the participant response than the 'perfect summation' approximation included in the original program.

7.2 Contributions

In addition to the observations made above, the following is a summary of the major contributions to the previously identified state of the art which can be attributed to the work presented in this dissertation.

1. The review identified that several theories exist for how binaural summation should be applied where conflicting results exist even in the most recent investigations. The comprehensive literature review provided in this document summarized all the existing research as it applied to each theory. A review of this information provides a quick comparison between each set of results in a convenient tabular format.
2. A review of the published literature confirmed that the binaural summation mechanism specifically influenced by frequency had not been targeted in the past. The specific focus of this research was therefore an innovative approach providing

an in-depth look into the perception of loudness. While the results presented have demonstrated that frequency plays an important role in the binaural difference required for equal loudness (BDEL values), they have also opened new potential avenues of investigation. Suggested follow-up studies have been included in the recommendations section below.

3. The updated TVL model developed successfully improved upon the original loudness model through the inclusion of the novel binaural summation algorithm. Results of a verification test confirmed that the updated model was a significant improvement over the original procedure.
4. The primary objective of this model was focused on frontal incident sounds which would produce equal sensations at both the left and right ears (an approximate diotic condition). This research may readily be extended to include other angles of incidence through additional jury testing.

7.3 Future Work and Recommendations

The update TVL model provided significant improvements over the original procedure. As identified in **Section 6.3** however, there are areas where additional improvements can be made. Additional research, as outlined by the following procedures may provide important insight as to how these improvements should be implemented.

1. The first and most direct extension of this work would be a repeat of the loudness matching experiment while including the effects of masking. Recall that while the revised metric demonstrated a significant improvement over the original version, verification trials suggested more information was necessary. By presenting the pure tones in the presence of a masking signal, new information may be revealed as

the binaural summation trends may vary because of the increased acoustic energy exposed to the listener. This hypothesis stemmed from the observation that loudness matches from recorded signals often do not match the anticipated increase suggested by the pure tone results. The perceived summation may therefore be found to change in the presence of off-frequency noise while re-examining each pure tone in the spectrum.

2. The second recommended extension targets the binaural summation of signals with varying levels of interaural differences. The level of binaural summation has been observed to vary as presentation deviates from the diotic case. A follow-up investigation should be conducted to measure this level of variance. The recommended procedure involves two parts: a headphone comparison similar to the previous experiment; and a loudspeaker comparison for additional insight into the appropriateness of headphones for loudness matching paradigms.
    a. The first experiment is relatively straightforward with little change from the previous experiments. Diotic pure tones would be presented to each participant and compared against the dichotic case where the amplitude varies between ears. The observed difference between the pair of matched signals would then present a measure of binaural summation for each pure tone frequency and level of interaural difference. As the entire frequency spectrum would ideally be investigated, it is anticipated that this experiment would be quite involved when done properly. Presenting the difference required for equal loudness as a measure of interaural difference could be done using a simple plot for each frequency with the amplitude of the left ear as the ordinate and the amplitude

of the right ear as abscissa; this presentation method is similar to one used by Irwin in 1965 as in [72]. Note that as Irwin's investigation focused only on one signal type (thermal noise), it may have overlooked the influence of frequency on summation.

b. The second experiment would involve a set-up similar to that used by Robinson and Whittle as in [70]. Speakers located at specific angles around the observer would be used to present pure tones at a variety of incident angles. Each angle would then be compared against a frontal incident reference as the diotic approximation point. Once matched the resulting difference required for equal loudness may provide additional insight into the influence of listening angle on loudness as the frequency of the stimulus is varied. When compared against the results from the first experiment, the loudspeaker investigation should closely mimic the headphone presentation results if each presentation method is acceptable.

BIBLIOGRAPHY

[1] H. Fletcher and W. Munson, "Loudness, its Definition, Measurement and Calculation," *J. Acoust. Soc. Am.*, vol. 5, pp. 82-108, 1933.

[2] B. R. Glasberg and B. C. J. Moore, "A model of loudness applicable to time-varying sounds," *Journal of the Audio Engineering Society*, vol. 50, no. 5, pp. 331-42, 2002.

[3] International Organization for Standardization, "ISO226 Acoustics – Normal Equal-Loudness Contours," International Organization for Standardization, Geneva, 2003.

[4] International Organization for Standardization, "ISO 2204 Acoustics - Guide to the Measurement of Airborne Acoustical Noise and Evaluation of its Effects on Man," International Organization for Standardization, Geneva, Switzerland, 1973.

[5] W. Yost and M. Killion, "Hearing Thresholds," in *Encyclopedia of Acoustics*, vol. 3, M. Crocker, Ed., Hoboken, NJ: John Wiley & Sons, Inc., 2007, pp. 1545-1554.

[6] B. Kingsbury, "A Direct Comparison of the Loudness of Pure Tones," *Phys.*, vol. 29, p. 588, 1927.

[7] "Handbook for Acoustical Ecology," 1999. [Online]. Available: http://www.sfu.ca/sonic-studio/handbook/Threshold_of_Pain.html. [Accessed 21 11 2016].

[8] L. Sivian and S. White, "On Minimum Audible Sound Fields," *J. Acoust. Soc. Am.* , vol. 4, pp. 288-321, 1933.

[9] B. G. Churcher and A. J. King, "The performance of noise meters in terms of the primary standard," *J. Inst. Electr. Eng.*, vol. 81, pp. 57-90, 1937.

[10] D. Robinson and R. Dadson, "A Re-Determination of the Equal-Loudness Relations for Pure Tones," *British Journal of Applied Physics*, vol. 7, pp. 166-181, 1956.

[11] E. Sengpiel, "Eberhard Sengpiel: Calculations for microphone recording techniques, electro and psycho acoustics and recording practique," sengpielaudio, 01 07 2014. [Online]. Available: http://www.sengpielaudio.com/Acoustics226-2003.pdf. [Accessed 16 07 2015].

[12] American National Standards Institute, "ANSI S1.11 - Specification for Octave-Band and Fractional-Octave-Band Analog and Digital Filters," Acoustical Society of America, Melville, NY, 2004.

[13] E. Zwicker, "Subdivision of the Audible Frequency Range into Critical Bands (Frequenzgruppen)," *J. Acoust. Soc. Am.,* vol. 33, no. 2, p. 248, 1961.

[14] B. C. J. Moore and B. R. Glasberg, "A revision of Zwicker's loudness model," *Acustica - Acta Acustica,* vol. 82, no. 2, pp. 335-345, 1996.

[15] B. C. J. Moore and B. R. Glasberg, "Formulae describing frequency selectivity as a function of frequency and level, and their use in calculating excitation patterns," *Hearing Research,* vol. 28, pp. 209-225, 1987.

[16] R. Patterson, I. Nimmo-Smith, D. Weber and R. Milroy, "The deterioration of hearing with age: frequency selectivity, the critical ratio, the audiogram, and speech threshold," *J. Acoust. Soc. Am.,* vol. 72, pp. 1788-1803, 1982.

[17] R. Tyler, "Frequency resolultion in hearing-impariment listeners," in *Frequency Selectivity in Hearing,* B. Moore, Ed., London, Academic Press, 1986, pp. 309-371.

[18] H. Fastl and E. Zwicker, Psychoacoustics: Facts and Models, Berlin: Springer, 2007.

[19] J. Steinberg, "The Loudness of a Sound and Its Physical Stimulus," *Phys. Rev.* , vol. 26, pp. 507-523, 1925.

[20] F. A. Everest, Master Handbook of Acoustics - Fourth Edition, New York: McGraw-Hill, 2001.

[21] P. D. S. Y. Schomer and F. Saito, "Evaluation of loudness-level weightings ro assessing the annoyance of environmental noise," *J. Acoust. Soc. Am.,* vol. 110, no. 5, pp. 2390-2397, 2001.

[22] S. Stevens, "Calculation of loudness of complex-noise," *Acoust. Soc. Am.,* vol. 28, no. 5, pp. 807-829, 1956.

[23] S. Stevens, "A scale for the measurement of psychological magnitude: loudness," *Psychological Review,* no. 43, pp. 405-416, 1936.

[24] International Organization for Standardization, "ISO532 Acoustics - Method for calculating loudness level," International Organization for Standardization, Geneva, 1975.

[25] Deutsches Institut für Normung, (German Institute for Standardization), "DIN 45631 - Procedure for calculating loudness level and loudness," DIN, Berlin, Germany, 1991.

[26] E. Zwicker, H. Fastl, E. Widmann, K. Kurakata, S. Kuwano and S. Namba, "Program for calculating loudness according to DIN 45631 (ISO 532B)," *J. Acoust. Soc. Jap. (E),* vol. 12, no. 1, pp. 39-42, 1991.

[27] American National Standards Institute, Inc. (ANSI), "ANSI/ASA S3.4-2007 (R 2012) Procedure for the Computation of Loudness of Steady Sounds," Standards Sceretariat, Melville, NY, 2012.

[28] American National Standards Institute, "ANSI S3.4:2007 - American National Standard Procedure for the Computation of Loudness of Steady Sounds," Acoustical Society of America, Melville, NY, 2007.

[29] S. Kuwano and S. Namba, "Chapter 6 Loudness in the Laboratory, Part II: Non-Steady-State Sounds," in *Loudness*, New York, Springer, 2011, pp. 145-168.

[30] S. Namba, S. Kuwano and T. Kato, "The Loudness of Sound with Intensity Increment," *Jpn. J. Psychol. Res.,* vol. 18, pp. 63-72, 1976.

[31] B. Scharf, "Loudness," in *Handbook of Perception, Vol. IV: Hearing*, New York, Academic, 1978, pp. 187-242.

[32] M. Florentine, S. Buus and T. Poulsen, "Temporal Integration of Loudnessas a Function of Level," *J. Acoust. Soc. Am.,* vol. 99, pp. 1633-1644, 1996.

[33] S. Buus, M. Florentine and T. Poulsen, "Temporal Integration of Loudness, Loudness Discrimination, and the Form of the Loudness Function," *J. Acoust. Soc. Am.,* vol. 101, pp. 669-680, 1997.

[34] T. Painter and A. Spanias, "Perceptual Coding of Digital Audio," in *IEEE, VOL 88, NO. 4,* 2000.

[35] R. J. Cassidy and J. O. I. Smith, "Center for Computer Research in Music and Acoustics," 2009. [Online]. Available: https://ccrma.stanford.edu/~jos/psychoacoustics/psychoacoustics.pdf. [Accessed 12 08 2015].

[36] Deutsches Institut für Normung, (German Institute for Standardization), "DIN 45631/A1 - Calculation of loudness level and loudness from the sound spectrum - Zwicker method - Amendment 1: Calculatoin of the loudness of time-variant sound," DIN, 2007.

[37] H. Bauch, "Die Bedeutung der Frequenzgrupp fur die Lautheit von Krangen," *Acustica,* vol. 6, pp. 40-45, 1956.

[38] B. R. Glasberg and B. C. Moore, "Derivation of Auditory Filter Shapes From Notched-Noise Data," *Hearing Research,* vol. 47, pp. 103-138, 1990.

[39] R. Hellman, "Perceived Magnitude of Two-Tone-Noise Complexes: Loudness, Annoyance, and Noisiness," *J. Acoust. Soc. Am.,* vol. 77, pp. 1497-1504, 1985.

[40] H. Fastl, "Loudness and Masking Patterns of Narrow Noise Bands," *Acustica*, vol. 33, pp. 266-271, 1975.

[41] B. Moore, S. Launer, D. Vickers and T. Baer, "Loudness of Modulated Sounds as a Function of Modulation Rate, Modulation Depth, Modulation Waveform, and Overall Level," in *Psychophysical and Physiological Advances in Hearing*, A. Palmer, A. Rees, A. Summerfield and R. Meddis, Eds., London, Whurr, 1998, pp. 456-471.

[42] B. Moore, D. Vickers, T. Baer and S. Launer, "Factors Affecting the Loudness of Modulated Sounds," *J. Acoust. Soc. Am.,* vol. 105, pp. 2757-2772, 1999.

[43] C. Zhang and F. Zeng, "Loudness of Dynamic Stimuli in Acoustics and Electric Hearing," *J. Acoust. Soc. Am. ,* vol. 102, pp. 2925-2934, 1997.

[44] B. R. Glasberg and B. C. J. Moore, "Development and Evaluation of a Model for Predicting the Audibility of Time-Varying Sounds in the Presence of Background Sounds," *J. Audio Eng. Soc.,* vol. 53, no. 10, pp. 906-918, 2005.

[45] J. Charbonneau, Comparison of loudness calculation procedure results to equal loudness contours, Windsor, Ontario: Thesis (M.A.Sc.) - University of Windsor, 2010., 2010.

[46] T. Brand, "Loudness Scaling," in *8th EFAS Congress*, Heidelberg, 2007.

[47] C. Elberling, "Loudness Scaling Revisited," *J. Am. Acad. Audiol.,* vol. 10, pp. 248-260, 1999.

[48] International Organization for Standardization, "ISO 16832 Acoustics - Loudness Scaling by Means of Categories," International Organization for Standardization, Geneva, Switzerland, 2006.

[49] L. Marks, "Binaural Summation of the Loudness of Pure Tones," *J. Acoust. Soc. Am. ,* vol. 64, pp. 107-113, 1978.

[50] R. Hellman and J. Zwislocki, "Monaural Loudness Function at 1000 cps, and Interaural Summation," *J. Acoust. Soc. Am. ,* vol. 35, pp. 856-865, 1963.

[51] S. Stevens and H. Greenbaum, "Regression effect in psychophysical judgement," *Percept. Psychophys.,* vol. 1, pp. 439-446, 1966.

[52] G. S. Reynolds and S. Stevens, "Binaural Summation of Loudness," *J. Acoust. Soc. Am. ,* vol. 32, no. 10, pp. 1337-1344, 1960.

[53] S. N. Joshi and W. Jesteadt, "Sequential Dependencies in Magnitude Scaling of Loudness," in *Meeting on Acoustics, Vol. 19, Acoustical Society of America*, Montreal, Canada, 2013.

[54] S. Stevens and E. Poulton, "The Estimation of Loudness by Unpracticed Observers," *J. Exp. Psychol.*, vol. 51, pp. 71-78, 1956.

[55] W. Levelt, J. Riemersma and A. Bunt, "Binaural Additivity of Loudness," *Brit. J. Math. Stat. Psychol.*, vol. 25, pp. 51-68, 1972.

[56] B. Bond and S. Stevens, "Cross-modality matching of brightness to loudness by 5-year-olds," *Perception & Psychophysics*, vol. 6, pp. 337-339, 1969.

[57] M. Teghtsoonian, "Children's scales of length and loudness: A developmental application of cross-modal matching.," *Journal of Experimental Child Psychology.*, vol. 30, pp. 290-307, 1980.

[58] S. Stevens, "The Direct Estimation of Sensory Magnitudes - Loudness," *Am. J. Psychol.*, vol. 69, p. 1, 1956.

[59] B. Scharf and D. Fishken, "Binaural summation of loudness: reconsidered," *Journal of Experimental Psychology*, vol. 86, no. 3, pp. 374-379, 1970.

[60] S. Stevens, Psychoacoustics. Introduction to its perceptual, neural and social prospects, G. Stevens, Ed., New York: Wiley, 1975.

[61] R. P. Hellman, "Growth of Loudness at 1000 and 3000 Hz," *J. Acoust. Soc. Am.*, vol. 60, no. 3, pp. 672-679, 1976.

[62] B. A. Schneider and A. J. Cohen, "Binaural additivity of loudness in children and adults," *Perception & Psychophysics*, vol. 59, no. 5, pp. 655-664, 1997.

[63] P. Zahorik and F. L. Wightman, "Loudness constancy with varying sound source distance," *Nature neuroscience*, vol. 4, no. 1, pp. 78-83, 2001.

[64] M. Epstein and M. Florentine, "Binaural Loudness Summation for Speech and Tones Presented via Earphones and Loudspeakers," *Ear Hear.*, vol. 30, pp. 234-237, 2009.

[65] J. Marozeau, M. Epstein, M. Florentine and B. Daley, "A test of the binaural equal-loudness ratio hypothesis for tones," *J. Acoust. Soc. Am.*, vol. 120, no. 6, pp. 3870-3877, 2006.

[66] B. Moore and B. Glasberg, "Modeling Binaural Loudness," *J. Acoust. Soc. Am.*, vol. 121, pp. 1604-1612, 2007.

[67] S. Stevens and H. Davis, Hearing, Its Psychology and Physiology, New York: John Wiley & Sons, Inc., 1938.

[68] D. Algom, B. Ben-Aharon and L. Cohen-Raz, "Dichotic, diotic, and monaural summation of loudness: A comprehensive analysis of composition and psychophysical functions," *Perception & Psychophysics,* vol. 46, no. 6, pp. 567-578, 1989.

[69] B. Scharf, "Dichotic Summation of Loudness," *J. Acoust. Soc. Amer.,* vol. 45, no. 5, pp. 1193-1205, 1969.

[70] D. Robinson and L. Whittle, "The Loudness of Directional Sound Fields," *Acustica,* vol. 10, pp. 74-80, 1960.

[71] V. Sivonen and W. Ellermeier, "Binaural Loudness for Artificial-Head Measurements in Directional Sound Fields," *J. Audio Eng. Soc.,* vol. 56, no. 6, pp. 452-461, 2008.

[72] R. Irwin, "Binaural summation of thermal noises of equal and unequal power in each ear," *The American Journal of Psychology,* vol. 78, no. 1, pp. 57-65, 1965.

[73] D. Algom, A. Rubin and L. Cohen-Raz, "Binaural and temporal integration of the loudness of tones and noises," *Perception & Psychophysics,* vol. 46, no. 2, pp. 155-166, 1989.

[74] I. Jankovic and D. Cross, "On the binaural additivity of loudness," in *Meeting of the Eastern Psychological Association*, Boston, 1977.

[75] G. Gigerenzer and G. Strube, "Are there limits to binaural additivity of loudness?," *J. Experimental Psychology: Human Perception and Performance,* vol. 9, no. 1, pp. 126-136, 1983.

[76] E. Zwicker and U. Zwicker, "Dependence of binaural loudness summation on interaural level differences, spectral distribution, and temporal distribution.," *J. Acoust. Soc. Am.,* vol. 89, no. 2, pp. 756-764, 1991.

[77] B. Mulligan, "Binaural Loudness Summation: Better and Less Than "Perfect"," in *123rd Meeting: Acoustical Society of America*, Salt Lake City, 1992.

[78] L. E. Marks, E. Galanter and J. C. Baird, "Binaural summation after learning psychophysical functions for loudness," *Perception & Psychophysics,* vol. 57, no. 8, pp. 1209-1216, 1995.

[79] V. P. Sivonen and W. Ellermeier, "Chapter 7 Binaural Loudness," in *Loudness*, M. Florentine, A. N. Popper and R. R. Fay, Eds., New York, Springer, 2011, pp. 169-198.

[80] V. Sivonen and W. Ellermeier, "Directional Loudness in an Anechoic Sound Field, Head-Related Transfer Functions, and Binaural Summation," *J. Acoust. Soc. Am.* , vol. 119, pp. 2965-2980, 2006.

[81] V. P. Sivonen, "Directional loudness and binaural summation for wideband and reverberant sounds," *J. Acoust. Soc. Am.,* vol. 121, no. 5, pp. 2852-2861, 2007.

[82] J. Culling and B. Edmonds, "Interaural correlation and loudness," in *Hearing - From sensory processing to perception*, B. Kollmeier, G. Klump, V. Hohmann, U. Langemann, M. Mauermann, S. Uppenkamp and J. Verhey, Eds., Heidelberg, Springer Verlag, 2007, pp. 359-368.

[83] S. Whilby, M. Florentine, E. Wagner and J. Marozeau, "Monaural and binaural loudness of 5- and 200-ms tones in normal and impaired hearing," *J. Acoust. Soc. Am.,* vol. 119, no. 6, pp. 3931-3939, 2006.

[84] B. A. Edmonds and J. F. Culling, "Interaural correlation and the binaural summation of loudness," *J. Acoust. Soc. Am.,* vol. 125, no. 6, pp. 3865-3870, 2009.

[85] V. Sivonen and W. Ellermeier, "Binaural Loudness," in *Loudness*, M. Florentine, A. Popper and R. Fay, Eds., New York City, NY, USA, Springer, 2011, pp. Chapter 7, 169-179.

[86] R. Caussé and P. Chavasse, "Études sur la fatigue auditive," *L'année psychologique,* Vols. 43-44, pp. 265-298, 1942.

[87] R. Porsolt and R. Irwin, "Binaural summation in loudness of two tones as a funciton of their bandwidth," *Amer. J. Psychol,* vol. 80, pp. 384-390, 1967.

[88] B. Scharf, "Binaural loudness summation as a function of bandwidth," *Rep. Int. Congr. Acoust.,* vol. 6th, no. Paper A-3-5, pp. 25-28, 1968.

[89] J. Zhang and D. Mao, "Dependence of binaural loudness summation on interaural level difference and frequency for pure tones," *Science China Physics, Mechanics & Astronomy,* vol. 53, no. 5, pp. 834-841, 2010.

[90] I. Hirsh, "Binaural Summation and Interaural Inhibition as a Function of the Level of Masking Noise," *Am. J. of Psychology,* vol. 61, no. 2, pp. 205-213, 1948.

[91] M. Epstein and M. Florentine, "Binaural loudness summation for speech and tones presented via earphones and loudspeakers," *Ear Hear.,* vol. 30, pp. 234-237, 2009.

[92] D. Pralong and S. Carlile, "The role of individualized headphone calibration for the generation of high fidelity virtual auditory space," *J. Acoust. Soc. Am.,* vol. 100, no. 6, pp. 3785-3793, 1996.

[93] A. Kulkarni and H. Colburn, "Variability in the characterization of the headphone transfer function," *J. Acoust. Soc. Am.*, vol. 107, no. 2, pp. 1071-1074, 2000.

[94] K. McAnally and R. Martin, "Variability in the headphone-to-ear-canal transfer function," *Journal of the Audio Engineering Society*, vol. 50, no. 4, pp. 263-266, 2002.

[95] F. Toole, "Binaural record/reproduction systems and their use in psychoacoustic investigations," in *Presented at the AES 91sst convention*, New York, NY, USA, 1991 October 4-8.

[96] F. Wightman and D. Kistler, "Headphone simulation of free-field listening, I: Stimulus synthesis," *J. Acoust. Soc. Am.*, vol. 85, no. 2, pp. 858-867, 1989.

[97] D. Green, "Chapter 5: Pure-tone air conduction threshold testing," in *J. Katz, Handbook of clinical audiology*, Baltimore, MD, Williams & Wilkins, 1994, pp. 97-108.

[98] M. Paquier and V. Koehl, "Audibility of headphone positioning variability," in *Auidio Engineering Society 128th Convention*, London, UK, 2010 May 22-25.

[99] M. Paquier, V. Koehl and B. Jantzem, "Influence of headphone position in pure-tone audiometry," in *Acoustics 2012*, Nantes Fance, 23-27 April 2012.

[100] Digital Recordings, Canada, "WWW HEARING TEST. Digital Audiometer - Professional v6.5," 2012. [Online]. Available: http://www.digital-recordings.com/. [Accessed 02 September 2015].

[101] D. V. Ferrari, E. A. Lopez, A. C. Lopes, C. P. Aiello and P. R. Jokura, "Results obtained with a low cost software-based audiometer for hearing screening," *Int. Arch. Otorhinolaryngol.*, vol. 17, no. 3, pp. 257-264, 2013.

[102] International Organization for Standardization, "ISO 8253-1:1989/R Acoustics -- Audiometric test methods -- Part 1: Pure-tone air and bone conduction audiometry," International Organization for Standardization, Geneva, 1989.

[103] S. Purdy and W. Warwick, "guideline for diagnosing occupational noise-induced hearing loss," 2012. [Online]. Available: http://www.acc.co.nz. [Accessed 13 03 2015].

[104] K. Wojcicki, "MATLAB Central File Exchange," 01 December 2011. [Online]. Available: http://www.mathworks.com/matlabcentral/fileexchange/34046-fade-signal. [Accessed 04 November 2015].

[105] K. Wojcicki, "MATLAB Central File Exchange," 02 December 2011. [Online]. Available: http://www.mathworks.com/matlabcentral/fileexchange/34058-pure-tone-generator. [Accessed 04 November 2015].

[106] B. Moore, "Chapter 7. Space Perception," in *An Introduction to the Psychology of Hearing*, sixth ed., Brill, Leiden, The Netherlands, 2012, pp. 245-250.

[107] M. Epstein and M. Florentine, "Binaural Loudness Summation for Speech Presented via Earphones and Loudspeaker with and without Visual Cues," *J. Acoust. Soc. Am.*, vol. 131, no. 5, pp. 3981-3988, 2012.

[108] O. M. o. t. Environment, "Model Municipal Noise Control By-Law, Final Report (NPC-100)," Ontario Government Bookstore, Toronto, August 1978.

[109] N. Anderson, "Algebraic Models in Perception," in *Handbook of Perception* , vol. II, E. Carterette and M. Friedman, Eds., New York, Academic, 1974.

[110] B. C. Moore, B. R. Glasberg and T. Baer, "A Model for the Prediction of Thresholds, Loudness, and Partial Loudness," *J. Audio Eng. Soc.*, vol. 45, no. 4, pp. 224-240, 1997.

[111] American National Standards Institute, "ANSI S3.4:2005 - American National Standard Procedure for the Computation of Loudness of Steady Sounds," Acoustical Society of America, Melville, NY, 2005.

[112] International Organization for Standardization, "ISO226 Acoustics – Normal Equal-Loudness Contours," International Organization for Standardization, Geneva, 1987.

[113] N. Chouard, "Loudness and Unpleasantness perception in dichotic conditions. Dissertation for doctoral degree," University of Oldenburg, Oldenburg, 1997.

[114] E. Bloch, "Das Binaurale Hören," *Z. Ohrenhlk.*, vol. 24, pp. 25-85, 1983.

[115] G. Békésy, "Zur Theorie des Hörens: Über die eben merkbare Amplituden-und Frequenzanderung eines Tones. Die Theorie der Schwebungen," *Physik. Z.* , vol. 30, pp. 721-745, 1929.

[116] J. Hugues, "The monaural threshold: effect of subliminal and audible contralateral and ipsilateral stimuli," *Proc. roy. Soc.*, vol. 128, no. B, pp. 144-152, 1938.

[117] F. LeRoux, "Sur les perceptions binauriculaires," *Gaz. hebd. Med. Chir.*, vol. 19, p. 296, 1875.

[118] R. Feldtkeller, E. Zwicker and E. Port, "Lautstarke, Verhaltnislautheit and Summenlautheit," *Frequenz*, vol. 4, p. 108, 1959.

[119] E. Zwicker, "Über psychologische und Methodische Grundlagen der Lautheit," *Acustica,* vol. 8, pp. 237-258, 1958.

[120] D. Robinson, "The Subjective Loudness Scale," *Acustica,* vol. 7, p. 217, 1957.

[121] V. Koehl and M. Paquier, "Loudness of Low-Frequency Pure Tones Lateralized by Interaural Time Differences," *Journal of the Acoustical Society of America,* vol. 137, no. 2, pp. 1040-1043, 2015.

[122] B. Scharf and J. Stevens, "The Form of the Loudness Function near Threshold," in *Proceedings of the Third International Congress on Acoustics*, Stuttgart, 1961.

[123] L. Miranda and D. Cabrera, "Evaluation of Binaural Loudness Models with Signals of Different Diffusivity," in *Acoustics 2008*, Geelong, Victoria, Australia, 2008.

[124] M. Paquier, V. Koehl and B. Jantzem, "Influence of headphone position in pure-tone audiometry," in *Acoustics 2012 Nantes Conference*, Nantes, France, 2012.

[125] M. Paquier and V. Koehl, "Audibility of headphone positioning variability," in *Audio Engineering Society - 128th Convention*, London, UK, 2010.

[126] W. Rudmose, "The case of the missing 6 dB," *J. Acoust. Soc. Am.* , vol. 71, pp. 650-659, 1982.

[127] V. Sivonen, P. Minnaar and W. Ellermeier, "Effect of direction on loudness in individual binaural synthesis," in *Proceedings of the Audio Engineering Society 118th Conference*, Barcelona, Spain, 2005.

[128] E. Langendijk and A. Bronkorst, "Fidelity of three-dimensional-sound reproduction using a virtual auditory display," *J. Acoust. Soc. Am.,* vol. 107, pp. 528-537, 2000.

[129] R. P. Hellman and J. J. Zwislocki, "Monaural loudness function at 1000 cps and interaural summation," *J. Acoust. Soc. Am.*, vol. 35, pp. 856-865, 1963.

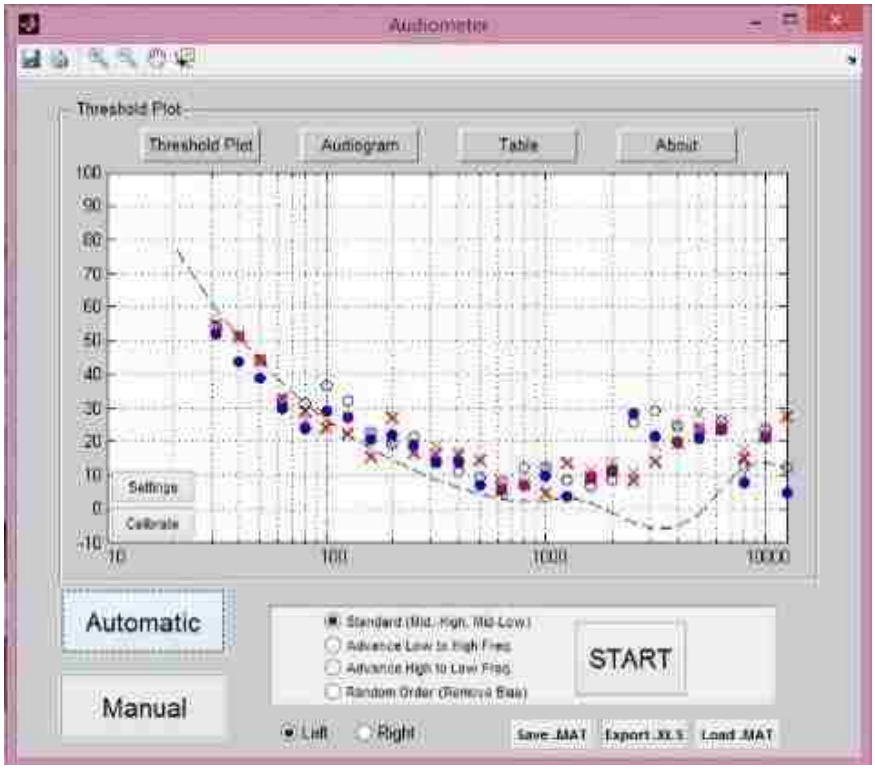
APPENDICES

APPENDIX A: Listening Test Development

Appendix A: Audiometer and Listening Test Development

A.1 – Air Conduction Pure Tone Audiometer

A.1.1 Audiometer GUI & Code (ExpAudiometerGUIv2.m)



MATLAB CODE

```
function AudiometerGUI ()
clc
global NewValue y Audiogram_Data ReferenceContour Resultant Cal_Row Check Freq playtime pausetime Increase1 Increase2 StartingValue PlayOrder
Font8 Font10 Font12 Font16


ErrorChecking=0; %Change to 1 to enable the feedback

% Default Values
Vol=0.3; % Volume of Start and Stop SOUNDS
playtime=1;
pausetime=0.2; %Pausetime is now given by (playtime(1+rand())) where the
%pause length has been ensured to be larger than the tone length while
%remaining random between itereations, (up to 100% larger).
Increase1=5; Increase2=2.5;
Fade=200;

Vol=0.3; % Volume of Start and Stop SOUNDS


if exist('UserSettings.txt')
        DefaultUserSettings=csvread('UserSettings.txt');
            playtime1               = DefaultUserSettings(1,5);
            pausetime1              = DefaultUserSettings(2,5);
            Fade1                   = DefaultUserSettings(3,5);
            Phase1                  = DefaultUserSettings(4,5);
            playtime2               = DefaultUserSettings(5,5);
            pausetime2              = DefaultUserSettings(6,5);
            Fade2                   = DefaultUserSettings(7,5);
```

```
            Phase2                  = DefaultUserSettings(8,5);
            Cal_Playback_Length     = DefaultUserSettings(9,5);
            Cal_Fade                = DefaultUserSettings(10,5);
            Increase1               = DefaultUserSettings(11,5);
            Increase2               = DefaultUserSettings(12,5);
            OctStep                 = DefaultUserSettings(13,5);
            Startvalue              = DefaultUserSettings(14,5);
            Endvalue                = DefaultUserSettings(15,5);
            Cur_Increase1           = DefaultUserSettings(16,5);
            Cur_Increase2           = DefaultUserSettings(17,5);
            Default_Oct             = DefaultUserSettings(18,5);
            Default_startvalue      = DefaultUserSettings(19,5);
            Default_endvalue        = DefaultUserSettings(20,5);
            AMeterPlayback          = DefaultUserSettings(21,5);
            AMeterFade              = DefaultUserSettings(22,5);
            AMeterIncrease1         = DefaultUserSettings(23,5);
            AMeterIncrease2         = DefaultUserSettings(24,5);
            AMeterOctValue          = DefaultUserSettings(25,5);
            AMeterStartValue        = DefaultUserSettings(26,5);
            AMeterEndValue          = DefaultUserSettings(27,5);
            PauseCheck              = DefaultUserSettings(28,5);
            TestdBZ                 = DefaultUserSettings(29,5);
            AudFunc                 = DefaultUserSettings(30,5);

            Increase1   = AMeterIncrease1;
            Increase2   = AMeterIncrease2;
            OctStep     = AMeterOctValue;
            Startvalue  = AMeterStartValue;
            Endvalue    = AMeterEndValue;

        % Threshold values as per ISO 226:2003, reproduced in ANSI S3.4:2007
            ReadDefaultSettings=csvread('UserSettings.txt');
            ReferenceContour =[Freq.',ReadDefaultSettings(:,4)];
            MAX_SPL_Lim =[Freq.',ReadDefaultSettings(:,3)]; % Load Maximum SPL Limit for Each Frequency
            AmbientSPL=[Freq.',ReadDefaultSettings(:,1),ReadDefaultSettings(:,2)]; % Load Reference

        OLDvNEW=size(Resultant);

        if OLDvNEW(1,1)==29         % OLD Audiograms stopped at 12,500 Hz
            freqcheck=1;

            for freqcheck=1:29
                ExpApp= getappdata(0,'evalue');
                ExpApp(freqcheck,5)=Resultant(freqcheck,3); % Column 5: Resultant L-Small
                ExpApp(freqcheck,6)=Resultant(freqcheck,5); % Column 6: Resultant R-Small
                ExpApp(30,5)=0.0;ExpApp(31,5)=0.0;
                ExpApp(30,6)=0;ExpApp(31,6)=0;
                setappdata(0,'evalue',ExpApp) % Placeholder for Important Values
            end

        else                        % NEW Audiograms go up to 20,000 Hz
            ExpApp= getappdata(0,'evalue');
            ExpApp(:,5)=Resultant(:,3); % Column 5: Resultant L-Small
            ExpApp(:,6)=Resultant(:,5); % Column 6: Resultant R-Small
            setappdata(0,'evalue',ExpApp) % Placeholder for Important Values
        end

else
            playtime            = 1;
            pausetime           = 0.25;
            Fade                = 200;
            Phase               = 0;
            OctStep             = 1;
            Startvalue          = 3;
            Endvalue            = 29;
            PlayCount=0;
            BreakOpt=0;
            PauseCheck=0;
            TestdBZ=0;
            AudFunc=0;
            ReferenceContour    =
[20,78.5;25,68.7;31.5,59.5;40,51.1;50,44;63,37.5;80,31.5;100,26.5;125,22.1;160,17.9;200,14.4;250,11.4;315,8.60;400,6.20;500,4.40;630,3;800,2.2
0;1000,2.20;1250,3.50;1600,1.70;2000,-1.30;2500,-4.20;3150,-6;4000,-5.40;5000,-1.50;6300,6;8000,12.6;10000,13.9;12500,12.3;16000,0;20000,0;];
            MAX_SPL_Lim = [20,60;   25,60;  31.5,60;    40,65;  50,80;  63,80;    80,80;    100,80;   125,80;   160,80;   200,80;   250,80;
315,80;   400,80;   500,80;   630,80;  800,80;   1000,80;  1250,80;  1600,80;  2000,80; 2500,80; 3150,80;    4000,80;    5000,80;    6300,80;
8000,80;  10000,80;     12500,80;     16000,80;    20000,80;];
            AmbientSPL=[Freq.',zeros(31,1),zeros(31,1)]; % Load Reference
end


Max00020_00031_5=60;
Max00031_5_00250=70;
Max00250_00500=90;
Max00500_04000=90; % Recommended 120
Max04000_12500=90; % Recommended 110

Default_Oct=1; %Third Octaves for Automatic Step Size
Default_startvalue=3; % 31.5Hz set as Lower Frequency Default
Default_endvalue=29;  % 12500Hz set as Upper Frequency

if Cal_Playback_Length ==0 %Setting Default where settings not saved.
    Cal_Playback_Length=10;
    Cal_Fade=200;
end

Sec_Drop=10+round(rand(1)*10)-5;

% Inserting Data to Plot
Freq=[20, 25, 31.5, 40, 50, 63, 80, 100, 125, 160, 200, 250, 315, 400, 500, 630, 800, 1000, 1250, 1600, 2000, 2500, 3150, 4000, 5000, 6300,
8000, 10000, 12500,16000, 20000];
FreqOct=[31.5, 63, 125, 250, 500, 1000, 2000, 4000, 8000, 16000];

NegativeTens=[-20; -20; -20; -20; -20; -20; -20; -20; -20; -20; -20; -20; -20; -20; -20; -20; -20; -20; -20; -20; -20; -20; -20; -20; -20; -
20; -20; -20; -20; -20; -20];
ZeroVector=[0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0];


ExpAppZeros =[0 0 0 playtime1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;              %1
    0 0 0 pausetime1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
    0 0 0 Fade1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
    0 0 0 Phase1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
    0 0 0 playtime2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;                        %5
    0 0 0 pausetime2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
    0 0 0 Fade2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
```

```
    0 0 0 Phase2 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
    0 0 0 Cal_Playback_Length 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
    0 0 0 Cal_Fade 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;                          %10
    0 0 0 Increase1 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
    0 0 0 Increase2 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
    0 0 0 OctStep 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
    0 0 0 Startvalue 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
    0 0 0 Endvalue 0 0 0 0 0 0 0 0 0 0 0 0 0 0;                            %15
    0 0 0 Cur_Increase1 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
    0 0 0 Cur_Increase2 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
    0 0 0 Default_Oct 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
    0 0 0 Default_startvalue 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
    0 0 0 Default_endvalue 0 0 0 0 0 0 0 0 0 0 0 0 0 0;                    %20
    0 0 0 AMeterPlayback 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
    0 0 0 AMeterFade 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
    0 0 0 AMeterIncrease1 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
    0 0 0 AMeterIncrease2 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
    0 0 0 AMeterOctValue 0 0 0 0 0 0 0 0 0 0 0 0 0 0;                      %25
    0 0 0 AMeterStartValue 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
    0 0 0 AMeterEndValue 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
    0 0 0 PauseCheck 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;                                         %30
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0];

    Increase1   =  AMeterIncrease1;
    Increase2   =  AMeterIncrease2;
    OctStep     =  AMeterOctValue;
    Startvalue  =  AMeterStartValue;
    Endvalue    =  AMeterEndValue;

    ExpAppZeros(:,11)=AmbientSPL(:,2);
    ExpAppZeros(:,12)=AmbientSPL(:,3);
    ExpAppZeros(:,13)=MAX_SPL_Lim(:,2);
    ExpAppZeros(:,14)=ReferenceContour(:,2);

    setappdata(0,'evalue',ExpAppZeros)


Calibration_File = getappdata(0,'evalue');

% evalue Placeholders
% Column 1: Frequency
% Column 2: Loaded Cal Left
% Column 3: Loaded Cal Right
% Column 4: User Settings
% Column 5: Resultant L-Small (:,3)
% Column 6: Resultant R-Small (:,5)
% Column 7: Headphones 1 - Left
% Column 8: Headphones 1 - Right
% Column 9: Headphones 2 - Left
% Column 10: Headphones 2 - Right
% Column 11: Ambient Conditions Left
% Column 12: Ambient Conditions Right
% Column 13: Maximum Limits
% Column 14: Reference Contour
% Column 15: Loaded Binaural Cal Headphones 1
% Column 16: Loaded Binaural Cal Headphones 2

Audiogram_Data = ReferenceContour(:,2);

Resultant=Freq.'; %Column reserved for Frequency
%Column reserved for Left Ear, Large increment
Resultant(:,2)=NegativeTens(:,1);
%Column reserved for Left Ear, Small increment
Resultant(:,3)=NegativeTens(:,1);
%Column reserved for Right Ear, Large increment
Resultant(:,4)=NegativeTens(:,1);
%Column reserved for Right Ear, Small increment
Resultant(:,5)=NegativeTens(:,1);
%Column reserved for Left Ear, Not Heard
Resultant(:,6)=NegativeTens(:,1);
%Column reserved for Right Ear, Not Heard
Resultant(:,7)=NegativeTens(:,1);

% Resultant Matrix
% Col 1 - Frequencies
% Col 2 - Left Ear, Large increment
% Col 3 - Left Ear, Small increment
% Col 4 - Right Ear, Large increment
% Col 5 - Right Ear, Small increment
% Col 6 - Left Ear, Not Heard
% Col 7 - Right Ear, Not Heard


StartingValue=0; %Starting Value Variable
%Setting Up Starting Values for Auto Program
StartingValues_Auto(:,1)=NegativeTens(:,1);

% Basic Graphical User Interface (GUI) without using GUIDE

%FIGURE DIMENSIONS
set(0,'units','pixels');
RelFont = get(0,'ScreenSize'); %#ok<NASGU>
set(0,'Units','normalized');
scnsize = get(0,'ScreenSize'); %#ok<NASGU>
MainWindow2 = figure('units', 'normalized',...
    'Position',[0.3,0.25,0.35,0.5], 'Visible', 'On',...
    'Name','Audiometer','NumberTitle','Off');
set(MainWindow2,'MenuBar','none');
set(MainWindow2,'Toolbar','figure');

h1Toolbar = findall(MainWindow2,'tag','FigureToolBar');
delete(findall(h1Toolbar,'tag','Standard.NewFigure'))
delete(findall(h1Toolbar,'tag','Standard.FileOpen'))
delete(findall(h1Toolbar,'tag','Plottools.PlottoolsOn'))
delete(findall(h1Toolbar,'tag','Plottools.PlottoolsOff'))
delete(findall(h1Toolbar,'tag','Annotation.InsertColorbar'))
delete(findall(h1Toolbar,'tag','DataManager.Linking'))
delete(findall(h1Toolbar,'tag','Standard.EditPlot'))
delete(findall(h1Toolbar,'tag','Exploration.Rotate'))
delete(findall(h1Toolbar,'tag','Exploration.Brushing'))
delete(findall(h1Toolbar,'tag','Annotation.InsertLegend'))
```

```
Font6=round(6/1080*RelFont(1,4));
Font8=round(8/1080*RelFont(1,4)); %Font 8 on 1920x1080
Font10=round(10/1080*RelFont(1,4));
Font12=round(12/1080*RelFont(1,4));
Font16=round(16/1080*RelFont(1,4));



% SETTINGS AND CALIBRATION

% PUSHBUTTON LOCATION - Calibration and Settings
CSPH = 0.112;
CSPV = 0.321;
CSW = 0.1;
CSH = 0.05;


Settings = uicontrol('style', 'pushbutton','string', 'Settings','units',...
    'normalized','Visible','on','position', [CSPH CSPV+0.054 CSW CSH],...
    'FontSize', Font8,'callback', {@Settings_Callback}); %#ok<NASGU>

    function Settings_Callback(~,~)
        UserSettings
    end

Calibrate = uicontrol('style', 'pushbutton','string', 'Calibrate','units',...
    'normalized','Visible','on','position', [CSPH CSPV CSW CSH],...
    'FontSize', Font8,'callback', {@Cal_Callback,MainWindow2}); %#ok<NASGU>

    function Cal_Callback(~,~,MainWindow)
        set(Calibrate,'BackgroundColor',[0.941176 0.941176 0.941176])
        ExpCalibration
    end

Calibration_File = getappdata(0,'evalue');

if Calibration_File(:,2)==ZeroVector(:,1)
    set(Calibrate,'BackgroundColor',[1 0 0])
end

bh=0.20;
c=(1/5)*(1-4*bh);
h = uibuttongroup('Visible','on','Position',[0.3 0.075 0.6 0.15]);
% Standard presentation method as per ISO 8253-1:1989
u00 = uicontrol('Style','Radio','String','Standard (Mid.-High, Mid-Low)',...
    'units', 'normalized','pos',[0.1 4*c+3*bh 0.5 bh],'parent',h,...
    'HandleVisibility','off','Value',1);
u0 = uicontrol('Style','Radio','String','Advance Low to High Freq.',...
    'units', 'normalized','pos',[0.1 3*c+2*bh 0.5 bh],'parent',h,...
    'HandleVisibility','off');
u1 = uicontrol('Style','Radio','String','Advance High to Low Freq.',...
    'units', 'normalized','pos',[0.1 2*c+bh 0.5 bh],...
    'parent',h,'HandleVisibility','off');
u2 = uicontrol('Style','Radio','String','Random Order (Remove Bias)',...
    'units', 'normalized','pos',[0.1 c 0.5 bh],...
    'parent',h,'HandleVisibility','off');
%set(h,'SelectedObject',[]);  % No selection


%create a "push button" for MANUAL START
but_manh = uicontrol('style', 'pushbutton','string', 'START','units',...
    'normalized','Visible','off','FontSize', Font16,...
    'position', [0.725 0.075 0.175 0.125],...
    'callback', {@RunManual}); %Function Run when Button Pressed

Initial_Freq = 20;
%MANUAL DISPLAY
%FREQUENCY DISPLAY
Box=uicontrol('style','frame',...
    'units', 'normalized','position',[0.295 0.145 0.41 .06],...
    'string', Initial_Freq,'Visible','off');

Frequency_Disp = uicontrol('style','text',...
    'units', 'normalized',...
    'position', [0.3 0.15 0.4 0.05],...
    'string', Initial_Freq,...
    'Visible','off',...
    'FontSize', Font16);

%MANUAL SLIDER
%create Slider object
Slider = uicontrol('style', 'Slider',...
    'Min',1,'Max',31,'Value',1,...
    'units', 'normalized',...
    'Visible','off',...
    'SliderStep', [1/28 0.1],...
    'position',[0.3 0.075 0.4 0.06],...
    'callback', {@change_freq,NewValue,Frequency_Disp});

%MANUAL AND AUTOMATIC RADIOBUTTON
Left_Right = uibuttongroup('visible','off','Visible','on',...
    'units', 'normalized',...
    'Position',[0.3 0.01 0.2 0.05],'ShadowColor',[0.8 0.8 0.8],...
    'BorderType','none','BackgroundColor',[0.8 0.8 0.8]);
Left_button = uicontrol('Style','Radio','String','Left',...
    'units', 'normalized','pos',[0.05 0.025 0.4 1],'parent',...
    Left_Right,'HandleVisibility','on','FontSize', Font10,...
    'BackgroundColor',[0.8 0.8 0.8]);
Right_button = uicontrol('Style','Radio','String','Right',...
    'units', 'normalized','pos',[0.5 0.025 0.4 1],'parent',...
    Left_Right,'HandleVisibility','on','FontSize', Font10,...
    'BackgroundColor',[0.8 0.8 0.8]); %#ok<NASGU>

% Setting initial frequency point at 20 Hz
set(Frequency_Disp,'string',20)
set(Frequency_Disp,'UserData',20)
y=get(Frequency_Disp,'UserData');

%AUTOMATIC START BUTTON
but_h = uicontrol('style', 'pushbutton','string', 'START',...
    'FontSize',Font16,'units','normalized', 'Visible','on',...
    'position', [0.6 0.075 0.225 0.8],...
    'parent',h,'SelectionHighlight','on',...
    'callback', {@RunAuto,Left_button,u0,u1,u2}); %AUTOSTART PUSHBUTTON
```

```
%create a "push button" for AUTOMATIC PROGRAM
but_AutoP = uicontrol('style', 'pushbutton',...
    'string', 'Automatic',...
    'units', 'normalized',...
    'Visible','on',...
    'FontSize', Font16,...
    'position', [0.05 0.15 0.2 0.1],...
    'callback', {@AutoSelect,but_manh,Slider,but_h,h,Frequency_Disp,...
    Box,u00}); %#ok<NASGU> %Function Run when Button Pressed

% Save Button Positions
SBh=0.58;
SBv=0.01;
SBw=0.105;
SBt=0.05;
SaveData = uicontrol('style', 'pushbutton','string', 'Save .MAT',...
    'units','normalized','Visible','on',...
    'position', [SBh SBv SBw SBt],...
    'FontSize', Font8, 'FontWeight',...
    'bold','callback', {@Save_Data,MainWindow2,u00,u0,u1,u2});

SaveXLS = uicontrol('style', 'pushbutton','string', 'Export .XLS',...
    'units','normalized','Visible','on',...
    'position', [SBh+SBw+0.005 SBv SBw SBt],...
    'FontSize', Font8, 'FontWeight',...
    'bold','callback', {@Save_XLS,MainWindow2,u00,u0,u1,u2});

LOAD_M = uicontrol('style', 'pushbutton','string', 'Load .MAT',...
    'units','normalized','Visible','on',...
    'position', [SBh+2*(SBw+0.005) SBv SBw SBt],...
    'FontSize', Font8, 'FontWeight',...
    'bold','callback', {@Load_Data,MainWindow2});

Return_Button = uicontrol('style', 'pushbutton','string', 'RETURN',...
    'units','normalized','Visible','on',...
    'position', [SBh+3*(SBw+0.005) SBv SBw*0.80 SBt],...
    'FontSize', Font8, 'FontWeight',...
    'bold','callback', {@Return,MainWindow2});

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%% Clear Data button %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Cannot get the plot to refresh once data is cleared.

% %
% % ClearData = uicontrol('style', 'pushbutton','string', 'Clear',...
% %     'units','normalized','Visible','on',...
% %     'position', [SBh+3*(SBw+0.005) SBv SBw*0.66 SBt],...
% %     'FontSize', Font8, 'FontWeight',...
% %     'bold','callback', {@Clear_Data,MainWindow,Resultant});

% Function to HIDE all of the MANUAL buttons
    function AutoSelect (~,~,but_manh,Slider,but_h,h,Frequency_Disp,Box,u00)
        set(but_manh,'Visible','off')
        set(Slider,'Visible','off')
        set(but_h,'Visible','on','SelectionHighlight','Off')
        set(h,'Visible','on')
        set(Frequency_Disp,'Visible','off')
        set(Box,'Visible','off')
        set(Left_Right,'Visible','on')
        set(SaveData,'Visible','on')
        set(SaveXLS,'Visible','on')
        set(LOAD_M,'Visible','on')
        set(Return_Button,'Visible','on')
        set(u00,'Value',1)
    end

% Function to HIDE all of the AUTOMATIC buttons
    function ManuSelect (~,~,but_manh,Slider,but_h,h,Frequency_Disp,Box)
        set(but_manh,'Visible','on')
        set(Slider,'Visible','on')
        set(but_h,'Visible','off')
        set(h,'Visible','off')
        set(Frequency_Disp,'Visible','on')
        set(Box,'Visible','on')
        set(Left_Right,'Visible','on')
        set(SaveData,'Visible','on')
        set(SaveXLS,'Visible','on')
        set(LOAD_M,'Visible','on')
        set(Return_Button,'Visible','on')
    end

%create a "push button" for MANUAL PROGRAM
but_ManP = uicontrol('style', 'pushbutton',...
    'string', 'Manual',...
    'units', 'normalized',...
    'Visible','on',...
    'position', [0.05 0.025 0.2 0.1],...
    'FontSize', Font16,...
    'callback', {@ManuSelect,but_manh,Slider,but_h,h,...
    Frequency_Disp,Box}); %#ok<NASGU> %Function Run when Button Pressed


% Plot_Frame_Text = uicontrol('style','text', 'units', 'normalized',...
%     'position', [0.095 0.295 0.81 0.56], 'string', 'General Settings',...
%     'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left');

Plot_Frame1 = uipanel('Title','Threshold Plot', 'units', 'normalized',...
    'position', [0.05 0.26 0.9 0.71], 'visible','on',...
    'FontSize', Font10,'BackgroundColor',[0.8 0.8 0.8],...
    'Visible','On');

Plot_Frame2 = uipanel('Title','Audiogram Plot', 'units', 'normalized',...
    'position', [0.05 0.26 0.9 0.71], 'visible','on',...
    'FontSize', Font10,'BackgroundColor',[0.8 0.8 0.8],...
    'Visible','Off');

Plot_Frame3 = uipanel('Title','Audiogram Table', 'units', 'normalized',...
    'position', [0.05 0.26 0.9 0.71], 'visible','on',...
    'FontSize', Font10,'BackgroundColor',[0.8 0.8 0.8],...
    'Visible','Off');

Plot_Frame4 = uipanel('Title','About Program', 'units', 'normalized',...
    'position', [0.05 0.26 0.9 0.71], 'visible','on',...
```

```
'FontSize', Font10,'BackgroundColor',[0.8 0.8 0.8],...
'Visible','Off');



function Plotting_Function
    %% Threshold Plot

    Picture1=subplot(1,1,1,'Parent',Plot_Frame1);

    PlotAxes = semilogx(ReferenceContour(:,1), ReferenceContour(:,2), '--k',...
                        AmbientSPL(:,1), AmbientSPL(:,2),'g',...           % Axis2 - Ambient Sound Pressure Levels (Freq, AMBIENT)
                        Resultant(:,1), Resultant(:,2),'ob',...             % Axis3 - Left Ear Big Increment    (Freq, LeftBig)
                        Resultant(:,1), Resultant(:,3),'ob',...             % Axis4 - Left Ear Small Increment  (Freq, Left)
                        Resultant(:,1), Resultant(:,4),'xr',...             % Axis5 - Right Ear Big Increment   (Freq, RightBig)
                        Resultant(:,1), Resultant(:,5),'xr',...             % Axis6 - Right Ear Small Increment (Freq, Right)
                        Resultant(:,1), Resultant(:,6),'xb',...             % Axis7 - Left Not Heard  (Freq, Left-x)
                        Resultant(:,1), Resultant(:,7),'xr',...             % Axis8 - Right Not Heard (Freq, Right-x)
                        'DisplayName', 'SemiLog Plot');                     % Name Assigned To Plot

    % Formatting the 'X' Markers to be larger and bolder
    set(PlotAxes(5),'MarkerSize',10,'LineWidth', 0.5)   %Right Small
    set(PlotAxes(6),'MarkerSize',10,'LineWidth', 2)     %Right Big

    % Formatting the 'Small' Markers to fill the markers
    set(PlotAxes(4),'MarkerFaceColor','b')  % Left Small
    set(PlotAxes(6),'MarkerFaceColor','r')  % Right Small


    set(PlotAxes(2),'LineWidth', 1)  % Right Small

    ax_PlotAxes = gca; %Collect info on current axes values to gca
    curtick = get(gca, 'XTick'); % Collecting current x-labels
    set(ax_PlotAxes, 'units', 'normalized',...
        'OuterPosition', [0 0.2 1 0.75],...
        'Position', [0.06 0.08 0.9 0.805],...
        'XLim', [10 20000],...
        'YLim', [-10 100],...
        'XTickLabel', cellstr(num2str(curtick(:)))); % Setting the x-labels to non-scientific
    grid
    freq_Value = get(Frequency_Disp,'string'); %#ok<NASGU>
    %% Audiogram Plot
    Picture2=subplot(1,1,1,'Parent',Plot_Frame2);

    AudiogramDat(:,1)=Resultant(:,1);
    AudiogramDat(:,2)=Resultant(:,3)-ReferenceContour(:,2);
    AudiogramDat(:,3)=Resultant(:,5);

    zerox=[10 12500];
    zeroy=[0 0];
    Audiogram=semilogx(Resultant(:,1), (Resultant(:,3)-ReferenceContour(:,2)),'ob',... % Left Ear Small
                       Resultant(:,1), (Resultant(:,5)-ReferenceContour(:,2)),'xr',... % Right Ear Small
                       zerox,zeroy,':k',...
                       'DisplayName', 'SemiLog Plot','Visible','on');


    set(Audiogram(1),'MarkerFaceColor','b')

    set(Audiogram(2),'MarkerSize',10,'LineWidth', 2)
    set(Audiogram(2),'MarkerFaceColor','r')

    ax_Audiogram = gca; %Collect info on current axes values to gca
    set(ax_Audiogram, 'units', 'normalized',...
        'OuterPosition', [0 0.2 1 0.75],...
        'Position', [0.06 0.08 0.9 0.805],...
        'XLim', [10 20000],...
        'YLim', [-10 100],...
        'XTickLabel', cellstr(num2str(curtick(:)))); % Setting the x-labels to non-scientific

    fillx=[10 12500];
    filly=[-10 15 25 40 55 70 90 100];

    hold(ax_Audiogram, 'on');

    SS=0.5;

    % Text Positioning
    TLeft=0.115;
    Tw=0.25;
    TvSp=0.41;
    Th=0.025;

    % Normal Hearing (-10)-15 dB HL
    Area1=fill([fillx(1) fillx(1) fillx(2) fillx(2)],...
        [filly(1) filly(2) filly(2) filly(1)],...
        [0.2 1.0 0.0],'FaceAlpha',0,'EdgeColor','none',...
        'SpecularStrength',SS);

    Text1 = uicontrol('style','text', 'units', 'normalized',...
        'position', [TLeft TvSp 0.11 Th], 'string', 'Normal Hearing',...
        'FontSize', Font8,'BackgroundColor',[0.2 1.0 0.0],...
        'HorizontalAlignment','left','Visible','off');

    gap=0.050;


    % Slight Hearing Loss 15-25 dB HL
    Area2=fill([fillx(1) fillx(1) fillx(2) fillx(2)],...
        [filly(2) filly(3) filly(3) filly(2)],...
        [1.0 1.0 0.0],'FaceAlpha',0,'EdgeColor','none',...
        'SpecularStrength',SS);

    Text2 = uicontrol('style','text', 'units', 'normalized',...
        'position', [TLeft TvSp+gap Tw Th], 'string', 'Slight Hearing Loss',...
        'FontSize', Font8,'BackgroundColor',[1.0 1.0 0.0],...
        'HorizontalAlignment','left','Visible','off');


    % Mild Hearing Loss 25-40 dB HL
    Area3=fill([fillx(1) fillx(1) fillx(2) fillx(2)],...
        [filly(3) filly(4) filly(4) filly(3)],...
        [1.0 0.6 0.0],'FaceAlpha',0,'EdgeColor','none',...
        'SpecularStrength',SS);
    gap=gap+0.075;
```

```
        Text3 = uicontrol('style','text', 'units', 'normalized',...
            'position', [TLeft TvSp+gap Tw Th], 'string', 'Mild Hearing Loss',...
            'FontSize', Font8,'BackgroundColor',[1.0 0.6 0.0],...
            'HorizontalAlignment','left','Visible','off');


        % Moderate Hearing Loss 40-55 dB HL
        Area4=fill([fillx(1) fillx(1) fillx(2) fillx(2)],...
            [filly(4) filly(5) filly(5) filly(4)],...
            [1.0 0.4 0.0],'FaceAlpha',0,'EdgeColor','none',...
            'SpecularStrength',SS);
        gap=gap+0.075;

        Text4 = uicontrol('style','text', 'units', 'normalized',...
            'position', [TLeft TvSp+gap Tw Th], 'string', 'Moderate Hearing Loss',...
            'FontSize', Font8,'BackgroundColor',[1.0 0.4 0.0],...
            'HorizontalAlignment','left','Visible','off');


        % Moderately Severe Hearing Loss 55-70 dB HL
        Area5=fill([fillx(1) fillx(1) fillx(2) fillx(2)],...
            [filly(5) filly(6) filly(6) filly(5)],...
            [1.0 0.0 0.0],'FaceAlpha',0,'EdgeColor','none',...
            'SpecularStrength',SS);
        gap=gap+0.075;

        Text5 = uicontrol('style','text', 'units', 'normalized',...
            'position', [TLeft TvSp+gap Tw Th], 'string', 'Moderately Severe Hearing Loss',...
            'FontSize', Font8,'BackgroundColor',[1.0 0.0 0.0],...
            'HorizontalAlignment','left','Visible','off');


        % Severe Hearing Loss 70-90 dB HL
        Area6=fill([fillx(1) fillx(1) fillx(2) fillx(2)],...
            [filly(6) filly(7) filly(7) filly(6)],...
            [0.8 0.0 0.2],'FaceAlpha',0,'EdgeColor','none',...
            'SpecularStrength',SS);
        gap=gap+0.075+0.025;

        Text6 = uicontrol('style','text', 'units', 'normalized',...
            'position', [TLeft TvSp+gap Tw Th], 'string', 'Severe Hearing Loss',...
            'FontSize', Font8,'BackgroundColor',[0.8 0.0 0.2],...
            'HorizontalAlignment','left','Visible','off');


        % Profound Hearing Loss >90 dB HL
        Area7=fill([fillx(1) fillx(1) fillx(2) fillx(2)],...
            [filly(7) filly(8) filly(8) filly(7)],...
            [0.6 0.0 0.4],'FaceAlpha',0,'EdgeColor','none',...
            'SpecularStrength',SS);
        gap=gap+0.075-0.025;

        Text7 = uicontrol('style','text', 'units', 'normalized',...
            'position', [TLeft TvSp+gap Tw Th], 'string', 'Profound Hearing Loss',...
            'FontSize', Font8,'BackgroundColor',[0.6 0.0 0.4],...
            'HorizontalAlignment','left','Visible','off');

        %hold %Release Plot
        hold(ax_Audiogram, 'off');
        %% Audiogram Table
        %           if dB_HL > 90;  'Profound Hearing Loss'
        %           if dB_HL <= 90;  'Severe Hearing Loss'
        %           if dB_HL <= 70;  'Moderately Severe Hearing Loss'
        %           if dB_HL <= 55;  'Moderate Hearing Loss'
        %           if dB_HL <= 40;  'Mild Hearing Loss'
        %           if dB_HL <= 25;  'Slight Hearing Loss'
        %           if dB_HL <= 15;  'Normal Hearing'


        cnames = {'Left','Level'};
        rnames = {'20 Hz','25 Hz','31.5 Hz','40 Hz','50 Hz','63 Hz','80 Hz','100 Hz','125 Hz','160 Hz','200 Hz','250 Hz','315 Hz','400
Hz','500 Hz','630 Hz','800 Hz','1 kHz','1.25 kHz','1.6 kHz','2 kHz','2.5 kHz','3.15 kHz','4 kHz','5 kHz','6.3 kHz','8 kHz','10 kHz','12.5
kHz','16 kHz','20kHz'};
        dat1(:,1)=(Resultant(:,3)-ReferenceContour(:,2));
        dat2(:,1)=(Resultant(:,5)-ReferenceContour(:,2));


        for i=1:31

            if dat1(i,1)<=15;
                dat1b(i,1)={sprintf('Normal Hearing')};
            end
            LvlCheck=-10;
            if dat1(i,1)<LvlCheck;
                dat1b(i,1)={sprintf('N/A')};
            end


            if dat1(i,1)>15;
                dat1b(i,1)={sprintf('Slight Hearing Loss')};
                if dat1(i,1)>25;
                    dat1b(i,1)={sprintf('Mild Hearing Loss')};
                    if dat1(i,1)>40;
                        dat1b(i,1)={sprintf('Moderate Hearing Loss')};
                        if dat1(i,1)>55;
                            dat1b(i,1)={sprintf('Moderately Severe Hearing Loss')};
                            if dat1(i,1)>70;
                                dat1b(i,1)={sprintf('Severe Hearing Loss')};
                                if dat1(i,1)>90;
                                    dat1b(i,1)={sprintf('Profound Hearing Loss')};
                                end
                            end
                        end
                    end
                end
            end
            if dat2(i,1)<=15;
                dat2b(i,1)={sprintf('Normal Hearing')};
            end
            if dat2(i,1)<LvlCheck;
                dat2b(i,1)={sprintf('N/A')};
            end
            if dat2(i,1)>15;
```

```
                dat2b(i,1)={sprintf('Slight Hearing Loss')};
                if dat2(i,1)>25;
                    dat2b(i,1)={sprintf('Mild Hearing Loss')};
                    if dat2(i,1)>40;
                        dat2b(i,1)={sprintf('Moderate Hearing Loss')};
                        if dat2(i,1)>55;
                            dat2b(i,1)={sprintf('Moderately Severe Hearing Loss')};
                            if dat2(i,1)>70;
                                dat2b(i,1)={sprintf('Severe Hearing Loss')};
                                if dat2(i,1)>90;
                                    dat2b(i,1)={sprintf('Profound Hearing Loss')};
                                end
                            end
                        end
                    end
                end
            end
        end


for i=1:31
    dat1(i,1)=str2double(sprintf('%.0f',dat1(i,1)));
    dat2(i,1)=str2double(sprintf('%.0f',dat2(i,1)));
end

dataset1=[num2cell(dat1),dat1b];
dataset2=[num2cell(dat2),dat2b];

Tabx=0.01;
Taby=0.05;
Tabw=0.475;
Tabv=0.8;
TabGap=0.5;

% %'ColumnWidth','auto'
% %'ColumnFormat',{'short','char'},
%
% a = [1;2];
% b = ['test1';'test2'];

Result_Table1=uitable('Data', dataset1, 'units', 'normalized',...
    'RowName',rnames,...
    'ColumnName',cnames,'FontSize',Font8,...
    'parent',Plot_Frame3, 'units', 'normalized',...
    'Position',[Tabx Taby Tabw Tabv],...
    'ColumnWidth',{50 175},'ColumnFormat',{'short','char'},...
    'Visible','off');

Tabx=Tabx+TabGap;
Result_Table2=uitable('Data', dataset2, 'units', 'normalized',...
    'RowName',rnames,...
    'ColumnName',cnames,'FontSize',Font8,...
    'parent',Plot_Frame3, ...
    'Position',[Tabx Taby Tabw Tabv],...
    'ColumnWidth',{50 175},'ColumnFormat',{'short','char'},...
    'Visible','off');
        %% About Program Tab

Desc = '  AUDIOMETER PROGRAM';
Desc1 = '    This audiometer program was developed as a means of determining the threshold of hearing for a given';
Desc1b = '    listener. There are two options which the program may be run in:';
Desc2 = '       (i) an automatic mode where the software automatically steps through each frequency; and';
Desc3 = '       (ii) a manual program where the user may select which frequency is played and over which ear.';
Desc4 = '    For improved accuracy, please calibrate, or load an existing calibration file into the model prior to starting ';
Desc4b = '    either test method. This can be done by using the included calibration option which is located in the lower ';
Desc4c = '    left of the "Threshold Plot" tab.';
DescSp = ' ';
CopyR = '                                                                                                    Copyright 2014 -
Jeremy Charbonneau';

Description_txt = uicontrol('style','text',...
    'units', 'normalized',...
    'position', [0.06 0.08 0.9 0.805],...
    'string', {Desc,DescSp,Desc1,Desc1b,
DescSp,Desc2,Desc3,DescSp,Desc4,Desc4b,Desc4c,DescSp,DescSp,DescSp,DescSp,DescSp,DescSp,DescSp,DescSp,CopyR },...
    'Visible','off',...
    'FontSize',Font8 ,...
    'HorizontalAlignment','left',...
    'Parent',Plot_Frame4);
        %% Figure Select Buttons
bh2=0.05;
bv=0.876;
bh=0.1;
bw=0.14;
wc=(1/5)*(0.8-4*bw);

% Standard presentation method as per ISO 8253-1:1989
FOpt1 = uicontrol('Style','PushButton','String','Threshold Plot',...
    'units', 'normalized','pos',[bh+wc bv bw bh2],...
    'HandleVisibility','off','Value',1,...
    'BackgroundColor',[0.8 0.8 0.8],'FontSize',Font10);
FOpt2 = uicontrol('Style','PushButton','String','Audiogram',...
    'units', 'normalized','pos',[bh+2*wc+bw bv bw bh2],...
    'HandleVisibility','off','FontSize',Font10,...
    'BackgroundColor',[0.8 0.8 0.8]);
FOpt3 = uicontrol('Style','PushButton','String','Table',...
    'units', 'normalized','pos',[bh+3*wc+2*bw bv bw bh2],...
    'HandleVisibility','off',...
    'BackgroundColor',[0.8 0.8 0.8],'FontSize',Font10);
FOpt4 = uicontrol('Style','PushButton','String','About',...
    'units', 'normalized','pos',[bh+4*wc+3*bw bv bw bh2],...
    'HandleVisibility','off',...
    'BackgroundColor',[0.8 0.8 0.8],'FontSize',Font10);

set(FOpt1,'Callback',{@Figure1,Plot_Frame1,Plot_Frame2,...
    Plot_Frame3,Plot_Frame4,Settings,Calibrate,Text1,...
    Text2,Text3,Text4,Text5,Text6,Text7,Result_Table2, Result_Table1, Description_txt,...
            LOAD_M,SaveXLS,SaveData,Return_Button})
set(FOpt2,'Callback',{@Figure2,Plot_Frame1,Plot_Frame2,...
    Plot_Frame3,Plot_Frame4,Settings,Calibrate,Text1,...
    Text2,Text3,Text4,Text5,Text6,Text7,Result_Table2, Result_Table1, Description_txt,...
            LOAD_M,SaveXLS,SaveData,Return_Button})
set(FOpt3,'Callback',{@Figure3,Plot_Frame1,Plot_Frame2,...
    Plot_Frame3,Plot_Frame4,Settings,Calibrate,Text1,...
```

```
    Text2,Text3,Text4,Text5,Text6,Text7,Result_Table2, Result_Table1, Description_txt,...
            LOAD_M,SaveXLS,SaveData,Return_Button})
set(FOpt4,'Callback',{@Figure4,Plot_Frame1,Plot_Frame2,...
    Plot_Frame3,Plot_Frame4,Settings,Calibrate,Text1,...
    Text2,Text3,Text4,Text5,Text6,Text7,Result_Table2, Result_Table1, Description_txt,...
            LOAD_M,SaveXLS,SaveData,Return_Button})

    function Figure1(~,~,Plot_Frame1,Plot_Frame2,Plot_Frame3,Plot_Frame4,Settings,Calibrate,Text1,...
            Text2,Text3,Text4,Text5,Text6,Text7,Result_Table2, Result_Table1, Description_txt,...
            LOAD_M,SaveXLS,SaveData,Return_Button)
        set(Plot_Frame1,'Visible','on')
        set(Plot_Frame2,'Visible','off')
        set(Plot_Frame3,'Visible','off')
        set(Plot_Frame4,'Visible','off')
        set(Settings,'Visible','on')
        set(Calibrate,'Visible','on')
        set(Text1,'Visible','off')
        set(Text2,'Visible','off')
        set(Text3,'Visible','off')
        set(Text4,'Visible','off')
        set(Text5,'Visible','off')
        set(Text6,'Visible','off')
        set(Text7,'Visible','off')
        set(Result_Table2,'Visible','off')
        set(Result_Table1,'Visible','off')
        set(Description_txt,'Visible','off')
        set(LOAD_M,'Visible','on')
        set(SaveXLS,'Visible','on')
        set(SaveData,'Visible','on')
        set(Return_Button,'Visible','on')
    end

    function Figure2(~,~,Plot_Frame1,Plot_Frame2,Plot_Frame3,Plot_Frame4,Settings,Calibrate,Text1,...
            Text2,Text3,Text4,Text5,Text6,Text7,Result_Table2, Result_Table1, Description_txt,...
            LOAD_M,SaveXLS,SaveData,Return_Button)
        set(Plot_Frame1,'Visible','off')
        set(Plot_Frame2,'Visible','on')
        set(Plot_Frame3,'Visible','off')
        set(Plot_Frame4,'Visible','off')
        set(Settings,'Visible','off')
        set(Calibrate,'Visible','off')
        set(Text1,'Visible','on')
        set(Text2,'Visible','on')
        set(Text3,'Visible','on')
        set(Text4,'Visible','on')
        set(Text5,'Visible','on')
        set(Text6,'Visible','on')
        set(Text7,'Visible','on')
        set(Result_Table2,'Visible','off')
        set(Result_Table1,'Visible','off')
        set(Description_txt,'Visible','off')
        set(LOAD_M,'Visible','off')
        set(SaveXLS,'Visible','off')
        set(SaveData,'Visible','off')
        set(Return_Button,'Visible','off')
    end

    function Figure3(~,~,Plot_Frame1,Plot_Frame2,Plot_Frame3,Plot_Frame4,Settings,Calibrate,Text1,...
            Text2,Text3,Text4,Text5,Text6,Text7,Result_Table2, Result_Table1, Description_txt,...
            LOAD_M,SaveXLS,SaveData,Return_Button)
        set(Plot_Frame1,'Visible','off')
        set(Plot_Frame2,'Visible','off')
        set(Plot_Frame3,'Visible','on')
        set(Plot_Frame4,'Visible','off')
        set(Settings,'Visible','off')
        set(Calibrate,'Visible','off')
        set(Text1,'Visible','off')
        set(Text2,'Visible','off')
        set(Text3,'Visible','off')
        set(Text4,'Visible','off')
        set(Text5,'Visible','off')
        set(Text6,'Visible','off')
        set(Text7,'Visible','off')
        set(Result_Table2,'Visible','on')
        set(Result_Table1,'Visible','on')
        set(Description_txt,'Visible','off')
        set(LOAD_M,'Visible','off')
        set(SaveXLS,'Visible','off')
        set(SaveData,'Visible','off')
        set(Return_Button,'Visible','off')
    end

    function Figure4(~,~,Plot_Frame1,Plot_Frame2,Plot_Frame3,Plot_Frame4,Settings,Calibrate,Text1,...
            Text2,Text3,Text4,Text5,Text6,Text7,Result_Table2, Result_Table1, Description_txt,...
            LOAD_M,SaveXLS,SaveData,Return_Button)
        set(Plot_Frame1,'Visible','off')
        set(Plot_Frame2,'Visible','off')
        set(Plot_Frame3,'Visible','off')
        set(Plot_Frame4,'Visible','on')
        set(Settings,'Visible','off')
        set(Calibrate,'Visible','off')
        set(Text1,'Visible','off')
        set(Text2,'Visible','off')
        set(Text3,'Visible','off')
        set(Text4,'Visible','off')
        set(Text5,'Visible','off')
        set(Text6,'Visible','off')
        set(Text7,'Visible','off')
        set(Result_Table2,'Visible','off')
        set(Result_Table1,'Visible','off')
        set(Description_txt,'Visible','on')
        set(LOAD_M,'Visible','off')
        set(SaveXLS,'Visible','off')
        set(SaveData,'Visible','off')
        set(Return_Button,'Visible','off')
    end
    end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% %%%%%%%%%%%%%%%     Automatic Program           %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [Value_Disp] = RunAuto(~,~,Left_button,...
```

```
    u0,u1,u2,Quit_Button) %#ok<STOUT,INUSD>
%During a AUTOMATIC run, the frequency order will be selected by
%the user where tones will be played in succession after each match
%is made. Once the pushbutton is initiated, the tone will play back
%in 4 second loops, at increasing amplitudes of 5dB increments
%until it is heard by the user (spacebar has been struck). At this
%point, the data point will be recorded and the process repeated at
% smaller interval increments (ie. 2dB), before moving onto the
% next frequency. Once complete, this process can be repeated for
% the other ear.
%Once complete, some sort of indication should play
%% Determining which ear is being targetted
    function [k1] = KeyPress(~,~)
        k1='stop'; %indicates button has been pressed
        sound(0.05)
        pause(0.001)
        set(but_h,'UserData',k1);

%         %Saving Feedback for User
%         Save_Disp = uicontrol('style','text',...
%             'units', 'normalized',...
%             'position', [0.375 0.8 0.25 0.05],...
%             'string', 'Saving...',...
%             'FontSize', Font16);
%         pause(0.5)
%         delete(Save_Disp)
    end

    function [k1] = breaker(~,~)
        k1='stop'; %indicates button has been pressed

        set(but_h,'UserData',k1);
    end

    function QuitProgram(~,~)
        WhileStop1=0;
        WhileStop2=0;
        BreakValue=1;
    end

    function Display_on_graph
        if ErrorChecking==1
            delete(Description_txt)
            if OctStep==1; OctStepString='1/3 Octaves'; end
            if OctStep==0; OctStepString='1/1 Octaves'; end
            Display_String = sprintf(' Step Size: %s \n Start/End: %.0f / %.0f \n Frequency: %.0f %.1f \n Amplitude 1: %.1f \n FreqRow 1: %.1f
\n "i" Value: %.1f \n Play Order: %.1f',OctStepString,Startvalue,Endvalue,FreqRow,FDesc,PlotAmp,FreqRow,i,PlayOrder(i,1));
            Description_txt = uicontrol('style','text',...
                'units', 'normalized',...
                'position', [0.6 0.60 0.2 0.3],...
                'string', {Display_String},...
                'Visible','on',...
                'FontSize',Font8 ,...
                'HorizontalAlignment','left','BackgroundColor',[1 1 1]);
        end
    end


    function CloseFcn(~,~)
        delete(Finished_Prompt)
    end

    function CloseFcn2(~,~)
        delete(NoCal_Prompt)
        % Adjusts calibration term to have a value slightly greater
        % than zero.
        Calibration_File(:,7)=ZeroVector(:,1)+0.0000001;
        Temp = getappdata(0,'evalue');
        Temp(:,2)=ZeroVector(:,1)+0.0000001; % Cal Left
        setappdata(0,'evalue',Temp);
    end

    function CloseFcn3(~,~)
        delete(NoCal_Prompt)
        ExpCalibration
    end

    function Cond_2_Function(~,~)
        Cond_2=0;
        if FDesc<31.5;
            if count>=Max00020_00031_5;
                Cond_2=1;
            end
        else %Frequency is greater than 100
            if FDesc>=250;
                if FDesc>=500;
                    if FDesc>=4000;
                        %Frequency is above 4000
                        if count>=Max04000_12500; count=Max04000_12500; Cond_2=1;end
                    else
                        %Frequency is between than 500 and 400
                        if count>=Max00500_04000; count=Max00500_04000; Cond_2=1;end
                    end
                else
                    %Frequency is between than 250 and 500
                    if count>=Max00250_00500; count=Max00250_00500; Cond_2=1;end
                end
            else
                %Frequency is between than 100 and 250
                if count>=Max00031_5_00250; count=Max00031_5_00250; Cond_2=1;end
            end
        end
    end

    function Cond_3_Function(~,~)
        % LIMITING FOR HEADPHONES
        if FDesc<31.5
            if count>=Max00020_00031_5;
                count=Max00020_00031_5;
            end
        else %Frequency is greater than 100
            if FDesc>=250;
                if FDesc>=500;
                    if FDesc>=4000;
```

```
                        %Frequency is above 4000
                        if count>=Max04000_12500; count=Max04000_12500; end
                    else
                        %Frequency is between than 500 and 400
                        if count>=Max00500_04000; count=Max00500_04000; end
                    end
                else
                    %Frequency is between than 250 and 500
                    if count>=Max00250_00500; count=Max00250_00500; end
                end
            else
                %Frequency is between than 100 and 250
                if count>=Max00031_5_00250; count=Max00031_5_00250; end
            end
        end
    end


Calibration_File = getappdata(0,'evalue');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%% CALIBRATION FILE CHECK %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Calibration_File(:,2);

if Calibration_File(:,2)==ZeroVector(:,1)
    disp('No Calibration Loaded')
    NoCal_Prompt= figure('units', 'normalized','Position',[0.4 0.4 0.15 0.1], 'Visible', 'On');
    set(NoCal_Prompt,'MenuBar','none');

    FinishedText = uicontrol('style','text',...
        'units', 'normalized',...
        'position', [0.05 0.6 0.9 0.3],...
        'string', 'No Calibration File Loaded',...
        'Visible','on',...
        'parent',NoCal_Prompt,...
        'FontSize', Font12,...
        'BackgroundColor',[0.8 0.8 0.8]); %#ok<NASGU>


    Continue_txt = ('Continue Without Calibration');

    Continue1 = uicontrol('style', 'pushbutton',...
        'string', Continue_txt,...
        'units', 'normalized',...
        'position', [0.2 0.4 0.6 0.25],...
        'parent',NoCal_Prompt,...
        'FontSize', Font8,...
        'callback', @CloseFcn2); %#ok<NASGU> %Function Run when Button Pressed

    Quit1 = uicontrol('style', 'pushbutton',...
        'string', 'Load Calibration',...
        'units', 'normalized',...
        'position', [0.2 0.1 0.6 0.25],...
        'parent',NoCal_Prompt,...
        'FontSize', Font8,...
        'callback', @CloseFcn3); %#ok<NASGU> %Function Run when Button Pressed


else

    % Play Starting Sound
    [x,y]=wavread('Speech_On.wav');
    sound(Vol*x,y)

    %BUTTON TO QUIT PROGRAM
    Quit_Button= uicontrol('style', 'pushbutton','string','Quit',...
        'FontSize', Font16,'units','normalized', 'Visible','on',...
        'position', [0.85 0.075 0.125 0.8],'parent',h,...
        'SelectionHighlight','Off',...
        'callback', {@QuitProgram}); %AUTOSTART PUSHBUTTON

    Calibration_File    = getappdata(0,'evalue');
    CurrentSettings     = getappdata(0,'evalue');
    playtime            = CurrentSettings(21,4);
    pausetime           = CurrentSettings(2,4);
    Fade                = CurrentSettings(22,4);
    Increase1           = CurrentSettings(23,4);
    Increase2           = CurrentSettings(24,4);
    OctStep             = CurrentSettings(25,4);
    Startvalue          = CurrentSettings(26,4);
    Endvalue            = CurrentSettings(27,4);

    L_R=get(Left_button,'Value');
    if L_R==1;
        ear='left';
        d=7;
    else
        ear='right';
        d=8;
    end

    %% Adding correction value from calibration to Starting point from ANSI data (only once)
    if OctStep==1;
        for i=1:31
            StartingValues_Auto(i,1)=ReferenceContour(i,2)+Calibration_File(i,d)-10;
        end
    end
    if OctStep==0;
        for i=1:10
            StartingValues_Auto(i,1)=ReferenceContour((3*i),2)+Calibration_File((3*i),d)-10;
        end
    end

    %% Setting up Frequencies and Playback order from Radio Button
    if OctStep==1; CurrentFreq=Freq.'; end  %#ok<NASGU> %Column of Frequencies used for Looping Selection

    if OctStep==0; CurrentFreq=FreqOct.'; end  %#ok<NASGU>



    u00Value=get(u00,'Value');%Standard Advance plays 1000Hz to upper limit followed by 1000Hz to lower limit - as per ISO 8253-1:1989
    u0Value=get(u0,'Value');%Advance Low to High Freq
```

```
    u1Value=get(u1,'Value');%Advance High to Low Freq
    u2Value=get(u2,'Value');%Advance Random Freq

    if u00Value==1;FreqOrder=100; %#ok<NASGU>
        if OctStep==1; PlayOrder=[18:31 fliplr(1:18)].'; end
        if OctStep==0; PlayOrder=[6:10 fliplr(1:6)].'; end
    end %Advance Low to High Freq

    if u0Value==1;FreqOrder=1; %#ok<NASGU>
        if OctStep==1; PlayOrder=(1:31).'; end
        if OctStep==0; PlayOrder=(1:10).'; end
    end %Advance Low to High Freq

    if u1Value==1;FreqOrder=2; %#ok<NASGU>
        if OctStep==1; PlayOrder=(fliplr(1:31)).'; end
        if OctStep==0; PlayOrder=(fliplr(1:10)).'; end
    end %Advance High to Low Freq

    if u2Value==1;FreqOrder=3; %#ok<NASGU>
        if OctStep==1; PlayOrder=randperm(31,31).'; end
        if OctStep==0; PlayOrder=randperm(10,10).'; end
        %generates 29 numbers in a random order from 1 to 29 with no
        %repeating values
    end


    %% Keystroke/Pushbutton Functions
    A{1,1} = 'SAVE'; A{1,2} = 'Spacebar';
    MultiText = sprintf('%s\n%s',A{1,1},A{1,2});

    set(but_h,'callback',{@breaker},'string',MultiText,'FontSize', Font8);

    set(but_h,'UserData','zero');

    % With Figure open, when anykey pressed run command
    set(gcf, 'WindowKeyPressFcn', {@KeyPress})


    BreakValue=0;
    %% Setting Initial Values for While Functions
    WhileStop1=1; WhileStop2=2;
    %Necessary values to start the while loops

    %FreqRow will be the designator for which frequency will play and
    %what order the playback will be in. The potential playback orders
    %are: (i) the low-to-high progression will start with FreqRow=1 and
    %iterate until FreqRow=29; (ii)the high-to-low progression will be
    %the opposite of (i); and (iii) the random progression will
    %generate a random ordering of the values 1 through 29 and the
    %frequencies will proceed in this random order until done.

    increase=Increase1;


    if OctStep==1;
        OctSize=31;
        if u00Value==1;OctSize=30;end
    end

    if OctStep==0;
        OctSize=10;
        if u00Value==1;OctSize=11;end
    end


    for i=1:OctSize
        FreqRow=PlayOrder(i,1);
        count=StartingValues_Auto(FreqRow,1);



        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        while WhileStop2==2 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

            if FreqRow<Startvalue;
%                fprintf('FreqRow < Startvalue')
                break;
            end
            if FreqRow>Endvalue;
%                fprintf('FreqRow > Endvalue')
                break;
            end

            % BreakValue included for QUIT Fuction
            if BreakValue==1; set(but_h,'callback',{@RunAuto,Left_button,u0,u1,u2},'string','START','FontSize', Font16);
set(Quit_Button,'Visible','off'); break; end


            while WhileStop1==1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                fprintf('small increment loop \n')
                % BreakValue included for QUIT Fuction
                if BreakValue==1; set(but_h,'callback',{@RunAuto,Left_button,u0,u1,u2},'string','START','FontSize', Font16);
set(Quit_Button,'Visible','off'); break; end

                k1=get(but_h,'UserData');

                %% Cond 1: Signal Was Heard, Save Requested
                if k1=='stop'; %#ok<STCMP>

                    if OctStep==0; PlotAmp=count-Calibration_File(3*FreqRow,d); end
                    if OctStep==1; PlotAmp=count-Calibration_File(FreqRow,d); end

                    set(but_h,'UserData','zero');
                    count=count-increase-Sec_Drop; if count<=StartingValues_Auto(FreqRow,1);count=StartingValues_Auto(FreqRow,1); end

                    if d==7; Rcol=2; end %Left Ear, Large Increment
                    if d==8; Rcol=4; end %Right Ear, Large Increment

                    %% Placing Resultant Values for Plot

                    if OctStep==0; Resultant(3*FreqRow,Rcol)=PlotAmp; FDesc=Freq(1,3*FreqRow); end
                    if OctStep==1; Resultant(FreqRow,Rcol)=PlotAmp; FDesc=Freq(1,FreqRow);end
```

```
                    %% Plotting Function

                    Plotting_Function


                    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                    %%%% DISPLAY ON GRAPH

                    Display_on_graph

                    %% Setting Iteration Values
                    increase=Increase2; %Advance to Smaller Increments
                    WhileStop1=2; % Prevent while loop 1 from restarting
                    break; % Break loop 1

                end


                %% Cond 2: Signal Not Heard, Amplitude Exceeded
                if OctStep==0; FDesc=(Freq(1,3*FreqRow)); end
                if OctStep==1; FDesc=(Freq(1,FreqRow)); end
                Cond_2_Function

                if Cond_2==1;
                    %Indicating a tone was not heard

                    PlotAmp=count;
                    if d==7; Rcol=6; end
                    if d==8; Rcol=7; end

                    %% Placing Resultant Values for Plot
                    if OctStep==0; Resultant(3*FreqRow,Rcol)=PlotAmp; FDesc=Freq(1,3*FreqRow); end
                    if OctStep==1; Resultant(FreqRow,Rcol)=PlotAmp; FDesc=Freq(1,FreqRow); end

                    %% Plotting Function
                    Plotting_Function

                    %% Turning Appropriate Buttons On/Off and Breaking
                    %set(SPL_Display1,'Visible','off')

                    %set(but_h,'callback',{@RunAuto,Left_button,u0,u1,u2},'string','START','FontSize', Font16);


                    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                    WhileStop1=2; %Prevent while loop 1 from restarting
                    WhileStop2=0; %Prevent while loop 2 from restarting
                    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

                    break;
                end

                %% Cond 3: Signal Not Heard, Amplitude Not-Exceeded
                count=count+increase;

                Cond_3_Function

                %set(SPL_Display1,'String',{'Count:',count});
                %set(SPL_Display2,'string',{'Actual (after correction):',count-Calibration_File(i,d+1)})

                if OctStep==0;
%                    fprintf('Freq: %d, SPL %d \n', FreqOct(1,PlayOrder(i,1)), count)
%                    PlotTone_adj_v3 (FreqOct(1,PlayOrder(i,1)), Fade, playtime*1000, count*[1,1], ear, 1, 0);
                      [  t, left, right, fs ] = PlotTone_adj_v3 (FreqOct(1,PlayOrder(i,1)), Fade, playtime*1000, count*[1,1], ear, 1, 0);
                        if d==7; player=audioplayer(left,fs); end
                        if d==8; player=audioplayer(right,fs); end
                        play(player)
                        pause(max(size(left))/fs);
                end
                if OctStep==1;
%                        fprintf('Freq: %d, SPL %d\n', Freq(1,PlayOrder(i,1)), count)

%                     PlotTone_adj_v3  (Freq(1,PlayOrder(i,1)), Fade, playtime*1000, count*[1,1], ear, 1, 0);
                      [  t, left, right, fs ] = PlotTone_adj_v3 (Freq(1,PlayOrder(i,1)), Fade, playtime*1000, count*[1,1], ear, 1, 0);
                        if d==7; player=audioplayer(left,fs); end
                        if d==8; player=audioplayer(right,fs); end
                        play(player)
                        pause(max(size(left))/fs);
                end

                pause(pausetime) %playtime*(1+0.5*rand()))

            end %%%%  End of While Loop 1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

            k1=get(but_h,'UserData');

            %% Cond 1: Signal Was Heard, Save Requested
            if k1=='stop'; %#ok<STCMP>

                if OctStep==0; PlotAmp=count-Calibration_File(3*FreqRow,d); FDesc=Freq(1,3*FreqRow); end
                if OctStep==1; PlotAmp=count-Calibration_File(FreqRow,d); FDesc=Freq(1,FreqRow); end


                set(but_h,'UserData','zero');

                count=count-increase-Sec_Drop; if count<=StartingValues_Auto(FreqRow,1);count=StartingValues_Auto(FreqRow,1) ;end %#ok<NASGU>

                if d==7; Rcol=3; end %Left Ear, Large Increment
                if d==8; Rcol=5; end %Right Ear, Large Increment

                %% Placing Resultant Values for Plot
                if OctStep==0; Resultant(3*FreqRow,Rcol)=PlotAmp; end
                if OctStep==1; Resultant(FreqRow,Rcol)=PlotAmp; end

                %% Plotting Function
                Plotting_Function

                %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                %%%% DISPLAY ON GRAPH

                Display_on_graph
                %% Setting Iteration Values
                increase=Increase2; %#ok<NASGU> %Advance to Smaller Increments
```

```
                WhileStop1=2; % Prevent while loop 1 from restarting
                break; % Break loop 1

            end

            %% Cond 2: Signal Not Heard, Amplitude Exceeded
            if OctStep==0; FDesc=(Freq(1,3*FreqRow)); end
            if OctStep==1; FDesc=(Freq(1,FreqRow)); end
            Cond_2_Function

            if Cond_2==1;
                %Indicating a tone was not heard

                PlotAmp=count;
                if d==7; Rcol=6; end
                if d==8; Rcol=7; end

                %% Placing Resultant Values for Plot
                if OctStep==0; Resultant(3*FreqRow,Rcol)=PlotAmp; FDesc=Freq(1,3*FreqRow); end
                if OctStep==1; Resultant(FreqRow,Rcol)=PlotAmp; FDesc=Freq(1,FreqRow); end
                %% Plotting Function
                Plotting_Function

                %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                %%%% DISPLAY ON GRAPH

                Display_on_graph
                %% Turning Appropriate Buttons On/Off and Breaking
                %set(SPL_Display1,'Visible','off')

                %set(but_h,'callback',{@RunAuto,Left_button,u0,u1,u2},'string','START','FontSize', Font16);

                %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                WhileStop1=2; %Prevent while loop 1 from restarting
                WhileStop2=0; %Prevent while loop 2 from restarting
                %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

                break;
            end

            %% Cond 3: Signal Not Heard, Amplitude Not-Exceeded
            count=count+increase;

            Cond_3_Function

            %set(SPL_Display1,'String',{'Count:',count});
            %set(SPL_Display2,'string',{'Actual (after correction):',count-Calibration_File(i,d+1)})


            if OctStep==0;
%                   fprintf('Freq: %d, SPL %d \n', FreqOct(1,PlayOrder(i,1)), count)
%                 PlotTone_adj_v3 (FreqOct(1,PlayOrder(i,1)), Fade, playtime*1000, count*[1,1], ear, 1, 0);
                      [  t, left, right, fs ] = PlotTone_adj_v3 (FreqOct(1,PlayOrder(i,1)), Fade, playtime*1000, count*[1,1], ear, 1, 0);
                          if d==7; player=audioplayer(left,fs); end
                          if d==8; player=audioplayer(right,fs); end
                          play(player)
                          pause(max(size(left))/fs);
            end
            if OctStep==1;
%                   fprintf('Freq: %d, SPL %d \n', Freq(1,PlayOrder(i,1)), count)
%                 PlotTone_adj_v3 (Freq(1,PlayOrder(i,1)), Fade, playtime*1000, count*[1,1], ear, 1, 0);
                      [  t, left, right, fs ] = PlotTone_adj_v3 (Freq(1,PlayOrder(i,1)), Fade, playtime*1000, count*[1,1], ear, 1, 0);
                          if d==7; player=audioplayer(left,fs); end
                          if d==8; player=audioplayer(right,fs); end
                          play(player)
                          pause(max(size(left))/fs);
            end


            pause(pausetime) %(playtime*(1+0.5*rand()))))

            if i==OctSize+1; set(but_h,'callback',{@RunAuto,Left_button,u0,u1,u2},'string','START','FontSize', Font16);
set(Quit_Button,'Visible','off'); WhileStop1=0;WhileStop2=0;end

        end %%%%  End of While Loop 2  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        if i==OctSize
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            % This Prompt was put in place to stop the accidental
            % restarting of the program. This way when a spacebar is
            % pressed after the autorun has completed, it will trigger
            % this pushbutton rather than the unwanted start command.


            Finished_Prompt= figure('units', 'normalized','Position',[0.4 0.4 0.15 0.10], 'Visible', 'On');
            set(Finished_Prompt,'MenuBar','none');

            FinishedText = uicontrol('style','text',...
                'units', 'normalized',...
                'position', [0.1 0.6 0.8 0.3],...
                'string', 'Auto Playback Complete',...
                'Visible','on',...
                'parent',Finished_Prompt,...
                'FontSize', Font16,...
                'BackgroundColor',[0.8 0.8 0.8]); %#ok<NASGU>


            FinishedOK= uicontrol('style', 'pushbutton',...
                'string', 'Finished',...
                'units', 'normalized',...
                'position', [0.3 0.2 0.4 0.3],...
                'parent',Finished_Prompt,...
                'FontSize', Font16,...
                'callback', @CloseFcn); %#ok<NASGU> %Function Run when Button Pressed
        end

        WhileStop1=1;
        increase=Increase1;
        WhileStop2=2;
```

```
    end
    set(but_h,'callback',{@RunAuto,Left_button,u0,u1,u2},'string','START','FontSize', Font16); set(Quit_Button,'Visible','off');

    % Play Quitting Sound
    [x,y]=wavread('Speech_Off.wav');
    sound(Vol*x,y)

end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% %%%%%%%%%%%%%%   Manual Program  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [ NewValue ] = change_freq(change_freq_handle,~,NewValue,Frequency_Disp) %#ok<INUSD>
% SLIDER FUNCTION - WORKS, DON'T TOUCH
% Simple funtion to change the slider value into the desired 1/3 Octaves


Freq_Lookup = [20;25;31.5;40;50;63;80;100;125;160;200;250;315;400;500;630;800;1000;1250;1600;2000;2500;3150;4000;5000;6300;8000;10000;12500];

% Get slider postion
slider_pos = round(get(change_freq_handle,'Value')); %Round to whole number incase user dragged slider bar
set(change_freq_handle, 'Value', slider_pos); %Set the slider position to match the rounded value

% If slider position is on a 1/3 Octave, change colour of frequency
% display

if ( 3 * round(double(slider_pos)/3) == slider_pos )==1
    set(Frequency_Disp,'BackgroundColor',[0.8 0.8 0.8])
else
    set(Frequency_Disp,'BackgroundColor',[0.941176 0.941176 0.941176])
end


NewValue=Freq_Lookup(slider_pos(1,1),1); %Lookup desired frequency from vector above

set(Frequency_Disp,'string',NewValue(1,1)) %Display the output frequency in the Frequency Display Bar
set(Frequency_Disp,'UserData',NewValue(1,1))
y=get(Frequency_Disp,'UserData');
end

function []=RunManual(~,~)
%%%%%%%%%%%%%%%%%%%  MANUAL PROGRAM FUNCTION %%%%%%%%%%%%%%%%%%%%%%%
%During a manual run, a frequency value will be selected by the
%user where the tone will be played. Once the pushbutton is
%initiated, the tone will play back in 4 second loops, at
%increasing amplitudes of 5dB increments until it is heard by the
%user (spacebar has been struck). At this point, the data point
%will be recorded and the process repeated at smaller interval
%increments (ie. 2dB)



Calibration_File = getappdata(0,'evalue');
CurrentSettings= getappdata(0,'evalue');
playtime = CurrentSettings(21,4);
pausetime = CurrentSettings(2,4);
Fade = CurrentSettings(22,4);
Increase1 = CurrentSettings(23,4);
Increase2 = CurrentSettings(24,4);


set(h,'Visible','off'); %Turn off auto buttons
set(but_manh,'visible','off')
x=get(Frequency_Disp,'UserData');


            % Play Starting Sound
            [xl,yl]=wavread('Speech_On.wav');
            sound(Vol*xl,yl)

%% Keystroke/Pushbutton Functions
%Push button 'placeholder' instructing to push Spacebar for match
SpaceButton = uicontrol('style', 'pushbutton',...
    'string', 'SAVE (Spacebar)',...
    'units', 'normalized',...
    'position', [0.725 0.075 0.175 0.125],...
    'callback', @breaker); %Function Run when Button Pressed
set(SpaceButton,'UserData','zero');


% With Figure open, when anykey pressed run command
set(gcf, 'WindowKeyPressFcn', {@KeyPress})

    function [k] = KeyPress(~,~)
        k='stop'; %indicates button has been pressed
        set(SpaceButton,'UserData',k);

%         %Saving Feedback for User
%         Save_Disp = uicontrol('style','text',...
%             'units', 'normalized',...
%             'position', [0.375 0.8 0.25 0.05],...
%             'string', 'Saving...',...
%             'FontSize', Font16);
%         pause(0.5)
%         delete(Save_Disp)

    end

    function [k] = breaker(~,~)
        k='stop'; %indicates button has been pressed
        set(SpaceButton,'UserData',k);

    end
g=1; Largei=2; %necessary values to start the while loops

Check = ismember(Freq,x).';
%% Collecting Correction Values
if Check(1,1)==1;Cal_Row=1;StartingValue = ReferenceContour(1,2)-10;end
if Check(2,1)==1;Cal_Row=2;StartingValue = ReferenceContour(2,2)-10;end
if Check(3,1)==1;Cal_Row=3;StartingValue = ReferenceContour(3,2)-10;end
```

```
if Check(4,1)==1;Cal_Row=4;StartingValue = ReferenceContour(4,2)-10;end
if Check(5,1)==1;Cal_Row=5;StartingValue = ReferenceContour(5,2)-10;end
if Check(6,1)==1;Cal_Row=6;StartingValue = ReferenceContour(6,2)-10;end
if Check(7,1)==1;Cal_Row=7;StartingValue = ReferenceContour(7,2)-10;end
if Check(8,1)==1;Cal_Row=8;StartingValue = ReferenceContour(8,2)-10;end
if Check(9,1)==1;Cal_Row=9;StartingValue = ReferenceContour(9,2)-10;end
if Check(10,1)==1;Cal_Row=10;StartingValue = ReferenceContour(10,2)-10;end
if Check(11,1)==1;Cal_Row=11;StartingValue = ReferenceContour(11,2)-10;end
if Check(12,1)==1;Cal_Row=12;StartingValue = ReferenceContour(12,2)-10;end
if Check(13,1)==1;Cal_Row=13;StartingValue = ReferenceContour(13,2)-10;end
if Check(14,1)==1;Cal_Row=14;StartingValue = ReferenceContour(14,2)-10;end
if Check(15,1)==1;Cal_Row=15;StartingValue = ReferenceContour(15,2)-10;end
if Check(16,1)==1;Cal_Row=16;StartingValue = ReferenceContour(16,2)-10;end
if Check(17,1)==1;Cal_Row=17;StartingValue = ReferenceContour(17,2)-10;end
if Check(18,1)==1;Cal_Row=18;StartingValue = ReferenceContour(18,2)-10;end
if Check(19,1)==1;Cal_Row=19;StartingValue = ReferenceContour(19,2)-10;end
if Check(20,1)==1;Cal_Row=20;StartingValue = ReferenceContour(20,2)-10;end
if Check(21,1)==1;Cal_Row=21;StartingValue = ReferenceContour(21,2)-10;end
if Check(22,1)==1;Cal_Row=22;StartingValue = ReferenceContour(22,2)-10;end
if Check(23,1)==1;Cal_Row=23;StartingValue = ReferenceContour(23,2)-10;end
if Check(24,1)==1;Cal_Row=24;StartingValue = ReferenceContour(24,2)-10;end
if Check(25,1)==1;Cal_Row=25;StartingValue = ReferenceContour(25,2)-10;end
if Check(26,1)==1;Cal_Row=26;StartingValue = ReferenceContour(26,2)-10;end
if Check(27,1)==1;Cal_Row=27;StartingValue = ReferenceContour(27,2)-10;end
if Check(28,1)==1;Cal_Row=28;StartingValue = ReferenceContour(28,2)-10;end
if Check(29,1)==1;Cal_Row=29;StartingValue = ReferenceContour(29,2)-10;end
if Check(30,1)==1;Cal_Row=30;StartingValue = ReferenceContour(30,2)-10;end
if Check(31,1)==1;Cal_Row=31;StartingValue = ReferenceContour(31,2)-10;end
%% Setting Initial Values before While Statement Begins

increase=Increase1;

Loopcheck=1;
count=StartingValue;
checkpoint=StartingValue+60;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    function Cond_2_Function(~,~)
        Cond_2=0;

        % Checking Max Amplitude Limits
        if FDesc<31.5;
            if count>=Max00020_00031_5;
                amp1=Max00020_00031_5;
                Cond_2=1;
            end
        else %Frequency is greater than 100
            if FDesc>=250;
                if FDesc>=500;
                    if FDesc>=4000;
                        %Frequency is above 4000
                        if count>=Max04000_12500; amp1=Max04000_12500; Cond_2=1;end
                    else
                        %Frequency is between than 500 and 400
                        if count>=Max00500_04000; amp1=Max00500_04000; Cond_2=1;end
                    end
                else
                    %Frequency is between than 250 and 500
                    if count>=Max00250_00500; amp1=Max00250_00500; Cond_2=1;end
                end
            else
                %Frequency is between than 100 and 250
                if count>=Max00031_5_00250; amp1=Max00031_5_00250; Cond_2=1;end
            end
        end
    end

    function Collect_Resultant(~,~)
        if Check(1,1)==1;Resultant(1,Rcol)=amp1;Cal_Row=1;end
        if Check(2,1)==1;Resultant(2,Rcol)=amp1;Cal_Row=2;end
        if Check(3,1)==1;Resultant(3,Rcol)=amp1;Cal_Row=3;end
        if Check(4,1)==1;Resultant(4,Rcol)=amp1;Cal_Row=4;end
        if Check(5,1)==1;Resultant(5,Rcol)=amp1;Cal_Row=5;end
        if Check(6,1)==1;Resultant(6,Rcol)=amp1;Cal_Row=6;end
        if Check(7,1)==1;Resultant(7,Rcol)=amp1;Cal_Row=7;end
        if Check(8,1)==1;Resultant(8,Rcol)=amp1;Cal_Row=8;end
        if Check(9,1)==1;Resultant(9,Rcol)=amp1;Cal_Row=9;end
        if Check(10,1)==1;Resultant(10,Rcol)=amp1;Cal_Row=10;end
        if Check(11,1)==1;Resultant(11,Rcol)=amp1;Cal_Row=11;end
        if Check(12,1)==1;Resultant(12,Rcol)=amp1;Cal_Row=12;end
        if Check(13,1)==1;Resultant(13,Rcol)=amp1;Cal_Row=13;end
        if Check(14,1)==1;Resultant(14,Rcol)=amp1;Cal_Row=14;end
        if Check(15,1)==1;Resultant(15,Rcol)=amp1;Cal_Row=15;end
        if Check(16,1)==1;Resultant(16,Rcol)=amp1;Cal_Row=16;end
        if Check(17,1)==1;Resultant(17,Rcol)=amp1;Cal_Row=17;end
        if Check(18,1)==1;Resultant(18,Rcol)=amp1;Cal_Row=18;end
        if Check(19,1)==1;Resultant(19,Rcol)=amp1;Cal_Row=19;end
        if Check(20,1)==1;Resultant(20,Rcol)=amp1;Cal_Row=20;end
        if Check(21,1)==1;Resultant(21,Rcol)=amp1;Cal_Row=21;end
        if Check(22,1)==1;Resultant(22,Rcol)=amp1;Cal_Row=22;end
        if Check(23,1)==1;Resultant(23,Rcol)=amp1;Cal_Row=23;end
        if Check(24,1)==1;Resultant(24,Rcol)=amp1;Cal_Row=24;end
        if Check(25,1)==1;Resultant(25,Rcol)=amp1;Cal_Row=25;end
        if Check(26,1)==1;Resultant(26,Rcol)=amp1;Cal_Row=26;end
        if Check(27,1)==1;Resultant(27,Rcol)=amp1;Cal_Row=27;end
        if Check(28,1)==1;Resultant(28,Rcol)=amp1;Cal_Row=28;end
        if Check(29,1)==1;Resultant(29,Rcol)=amp1;Cal_Row=29;end
        if Check(30,1)==1;Resultant(30,Rcol)=amp1;Cal_Row=30;end
        if Check(31,1)==1;Resultant(31,Rcol)=amp1;Cal_Row=31;end
    end

    function Collect_Resultant2(~,~)
        if Check(1,1)==1;Resultant(1,Rcol)=amp2;Cal_Row=1;end
        if Check(2,1)==1;Resultant(2,Rcol)=amp2;Cal_Row=2;end
        if Check(3,1)==1;Resultant(3,Rcol)=amp2;Cal_Row=3;end
        if Check(4,1)==1;Resultant(4,Rcol)=amp2;Cal_Row=4;end
        if Check(5,1)==1;Resultant(5,Rcol)=amp2;Cal_Row=5;end
        if Check(6,1)==1;Resultant(6,Rcol)=amp2;Cal_Row=6;end
        if Check(7,1)==1;Resultant(7,Rcol)=amp2;Cal_Row=7;end
        if Check(8,1)==1;Resultant(8,Rcol)=amp2;Cal_Row=8;end
        if Check(9,1)==1;Resultant(9,Rcol)=amp2;Cal_Row=9;end
        if Check(10,1)==1;Resultant(10,Rcol)=amp2;Cal_Row=10;end
        if Check(11,1)==1;Resultant(11,Rcol)=amp2;Cal_Row=11;end
```

```
        if Check(12,1)==1;Resultant(12,Rcol)=amp2;Cal_Row=12;end
        if Check(13,1)==1;Resultant(13,Rcol)=amp2;Cal_Row=13;end
        if Check(14,1)==1;Resultant(14,Rcol)=amp2;Cal_Row=14;end
        if Check(15,1)==1;Resultant(15,Rcol)=amp2;Cal_Row=15;end
        if Check(16,1)==1;Resultant(16,Rcol)=amp2;Cal_Row=16;end
        if Check(17,1)==1;Resultant(17,Rcol)=amp2;Cal_Row=17;end
        if Check(18,1)==1;Resultant(18,Rcol)=amp2;Cal_Row=18;end
        if Check(19,1)==1;Resultant(19,Rcol)=amp2;Cal_Row=19;end
        if Check(20,1)==1;Resultant(20,Rcol)=amp2;Cal_Row=20;end
        if Check(21,1)==1;Resultant(21,Rcol)=amp2;Cal_Row=21;end
        if Check(22,1)==1;Resultant(22,Rcol)=amp2;Cal_Row=22;end
        if Check(23,1)==1;Resultant(23,Rcol)=amp2;Cal_Row=23;end
        if Check(24,1)==1;Resultant(24,Rcol)=amp2;Cal_Row=24;end
        if Check(25,1)==1;Resultant(25,Rcol)=amp2;Cal_Row=25;end
        if Check(26,1)==1;Resultant(26,Rcol)=amp2;Cal_Row=26;end
        if Check(27,1)==1;Resultant(27,Rcol)=amp2;Cal_Row=27;end
        if Check(28,1)==1;Resultant(28,Rcol)=amp2;Cal_Row=28;end
        if Check(29,1)==1;Resultant(29,Rcol)=amp2;Cal_Row=29;end
        if Check(30,1)==1;Resultant(30,Rcol)=amp2;Cal_Row=30;end
        if Check(31,1)==1;Resultant(31,Rcol)=amp2;Cal_Row=31;end
    end


%% While Statement Block - One within the other (varying steps)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
while Largei==2 %While loop for 2nd iteration - Small increments
    %% Determining which ear is being targetted
    L_R=get(Left_button,'Value');
    if L_R==1;
        ear='left';
        d=7;            %Left Calibration stored in 'evalue'
%         fprintf('Small Inc. Left Ear \n')
    else
        ear='right';
        d=8;            %Right Calibration stored in 'evalue'
%         fprintf('Small Inc. Right Ear \n')
    end

        % Resultant Matrix
        % Col 1 - Frequencies
        % Col 2 - Left Ear, Large increment
        % Col 3 - Left Ear, Small increment
        % Col 4 - Right Ear, Large increment
        % Col 5 - Right Ear, Small increment
        % Col 6 - Left Ear, Not Heard
        % Col 7 - Right Ear, Not Heard

    %% Adding correction value from calibration file (only once)
    if  Loopcheck==1
        Correction=Calibration_File(Cal_Row,d);
        count = StartingValue+Correction; %Will adjust output so that the intended level is actually played
        Loopcheck=2;
    end
    %% Lookup command to determine frequency row
    Check = ismember(Freq,x).';
    %% Error Checking Display to indicate current amplitude
    %SPL_Display = uicontrol('style','text',...
    %                'units', 'normalized',...
    %                'position', [0.8 0.8 0.08 0.08],...
    %                'string', count,...
    %                'FontSize', 20);
    %set(SPL_Display,'string',count)

    %% Inner While Block - Larger Iteration Steps
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    while g==1 %While for 1st iteration - 10 dB increments
%
%         L_R=get(Left_button,'Value');
%         if L_R==1;
%             fprintf('Large Inc. Left Ear \n')
%         else
%             fprintf('Large Inc. Left Ear \n')
%         end

        %k will be used as a feedback indicator to determine when
        %stop/save command has been issued. In this case, the
        %command is given whenever a button is pressed via the
        %'KeyPress' or 'breaker' functions located above.

        % Check k value
        k=get(SpaceButton,'UserData');

        % If button has been pressed, k value will show 'stop.' On
        % 'stop' save all data to the Resultant Vector.
        if k=='stop'; %#ok<STCMP>

            freq1=x; %#ok<NASGU>
            amp1=count-Correction; %Saving the correct numerical value (correction was added only so the numerical value and the output were
the same)
            set(SpaceButton,'UserData','zero');

%             fprintf('Reset Count \n')
            count=count-increase-Sec_Drop;
            if count<=(StartingValue+Correction);count=StartingValue+Correction;end

            if d==7; Rcol=2; end
            if d==8; Rcol=4; end

            % Placing Resultant Values for Plot
%             fprintf('Collecting Resultant \n')
            Collect_Resultant

            % Plotting Function
%             fprintf('Plot \n')
            Plotting_Function
            % Setting Iteration Values
            increase=Increase2;
            g=2; %Prevent while loop 1 from restarting

            break;

        end
```

```
        %% CONDITION 2: Signal Not Heard, Amplitude Exceeded
        FDesc=x;
        Cond_2_Function  % Limit Check
        if Cond_2==1;    % Limit Has been Exceeded


            if d==7; Rcol=6; end
            if d==8; Rcol=7; end

            %% Placing Resultant Values for Plot
            Collect_Resultant


            %% Plotting Function
            Plotting_Function
            %% Turning Appropriate Buttons On/Off

            set(but_manh,'visible','on')
            %set(SPL_Display,'Visible','off')
            set(SpaceButton,'Visible','off')

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            g=2; %Prevent while loop 1 from restarting
            Largei=0; %Prevent while loop 2 from restarting
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

            break;
        end

        count=count+increase;

        %set(SPL_Display,'string',count)
        [  t, left, right, fs ] = PlotTone_adj_v3 (x, Fade, playtime*1000, count*[1,1], ear, 1, 0);

        if d==7; player=audioplayer(left,fs); end
        if d==8; player=audioplayer(right,fs); end

        play(player)
        pause(max(size(left))/fs);
        pause(pausetime) %(playtime*(1+0.5*rand())))

    end  %%%%  End of While Loop 1  %%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%     fprintf('\n Out of loop 1 \n')


    k=get(SpaceButton,'UserData');

    if k=='stop'; %#ok<STCMP>
        freq2=x; %#ok<NASGU>
        amp2=count-Correction;

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        Largei=0; %#ok<NASGU> %Prevent while loop 2 from restarting
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        if d==7; Rcol=3; end
        if d==8; Rcol=5; end

        %% Placing Resultant Values for Plot
        Collect_Resultant2

        %% Plotting Function
        Plotting_Function
        %% Setting Iteration Values
        increase=Increase2; %#ok<NASGU>
        set(SpaceButton,'Visible','off')
        %set(SPL_Display,'Visible','off')
        break;
    end

    %% CONDITION 2: Signal Not Heard, Amplitude Exceeded
    FDesc=x;
    Cond_2_Function

    if Cond_2==1;
%         fprintf('Max Exceeded in COND 2\n')
        %ampl=-10;

        if d==7; Rcol=6; end
        if d==8; Rcol=7; end

        %% Placing Resultant Values for Plot
        Collect_Resultant2

        %% Plotting Function
        Plotting_Function
        %% Setting Iteration Values

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        Largei=0; %#ok<NASGU> %Prevent while loop 2 from restarting
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        set(but_manh,'visible','on')
        %set(SPL_Display,'Visible','off')
        set(SpaceButton,'Visible','off')

        %fprintf('\n Break 2')%%%  %%%  %%%  %%%%
        break;
    end

     count=count+increase;

    [  t, left, right, fs ] = PlotTone_adj_v3 (x, Fade, playtime*1000, count*[1,1], ear, 1, 0);
        if d==7; player=audioplayer(left,fs); end
        if d==8; player=audioplayer(right,fs); end
        play(player)
        pause(max(size(left))/fs);

    pause(pausetime) %(playtime*(1+0.5*rand())))
```

```
end %%%%  End of While Loop 2  %%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Verifying that the While statements are complete and buttons are on/off

set(but_manh,'visible','on')
%set(SPL_Display,'Visible','off')

            % Play Quitting Sound
            [x2,y2]=wavread('Speech_Off.wav');
            sound(Vol*x2,y2)

end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% %%%%%%%%%%%%%%%%%%%%   End Code                    %%%%%%%%%%%%%%%%%%%%%%%%

Plotting_Function
%% Saving Functions
function Save_Data(~,~,MainWindow,u00,u0,u1,u2)
[filename, pathname] = uiputfile({'*.mat'}, 'Save Workspace as');
Cal_File=getappdata(0,'evalue');

% evalue Placeholders
% Column 1: Frequency
% Column 2: Audiometer Loaded Cal Left
% Column 3: Audiometer Loaded Cal Right
% Column 4: User Settings
% Column 5: Resultant L-Small (:,3)
% Column 6: Resultant R-Small (:,5)
% Column 7: Headphones 1 - Left
% Column 8: Headphones 1 - Right
% Column 9: Headphones 2 - Left
% Column 10: Headphones 2 - Right
% Column 11: Ambient Conditions Left
% Column 12: Ambient Conditions Right
% Column 13: Maximum Limits
% Column 14: Reference Contour
% Column 15: Loaded Binaural Cal Headphones 1
% Column 16: Loaded Binaural Cal Headphones 2
% Column 17: Resultant L-Large (:,2)
% Column 18: Resultant R-Large (:,4)

CalOutput(:,1)=Cal_File(:,1);
CalOutput(:,2)=Cal_File(:,2);
CalOutput(:,3)=Cal_File(:,3);
CalFileColumns = {'Frequency (Hz)','Left Channel','Right Channel'};

if isequal(filename,0) || isequal(pathname,0)
    disp('') %Do Nothing when cancelled or closed
else
    save ([pathname filename], 'Resultant', 'Cal_File');

    refresh(MainWindow);
end
end
%% LOAD AUDIOGRAM
function Load_Data(~,~,MainWindow)
[FileName,PathName] = uigetfile('*.mat','Select mat file');
if FileName==0, return, end

Struct1 = load( fullfile(PathName,FileName) );   %# pass file path as string
Structname = fieldnames(Struct1);                %# typo?
Loaded_Response=Struct1.(Structname{2,1});
Loaded_Cal=Struct1.(Structname{1,1});
Resultant=Loaded_Response;

        OLDvNEW=size(Resultant);

        if OLDvNEW(1,1)==29         % OLD Audiograms stopped at 12,500 Hz
            i=1;

            for i=1:29
                ExpApp= getappdata(0,'evalue');
                ExpApp(i,2)=Loaded_Cal(i,2); % Cal Left
                ExpApp(i,3)=Loaded_Cal(i,3); % Cal Right
                ExpApp(i,5)=Resultant(i,3); % Column 5: Resultant L-Small
                ExpApp(i,6)=Resultant(i,5); % Column 6: Resultant R-Small
                ExpApp(i,17)=Resultant(i,2); % Column 17: Resultant L-Large
                ExpApp(i,18)=Resultant(i,4); % Column 18: Resultant R-Large

                ExpApp(30,5)=0.0;   ExpApp(31,5)=0.0;
                ExpApp(30,6)=0.0;   ExpApp(31,6)=0.0;
                Resultant(30,3)=0.0;Resultant(31,3)=0.0;
                Resultant(30,4)=0.0;Resultant(31,4)=0.0;
                Resultant(30,5)=0.0;Resultant(31,5)=0.0;
                Resultant(30,6)=0.0;Resultant(31,6)=0.0;

                setappdata(0,'evalue',ExpApp) % Placeholder for Important Values
            end

        else                        % NEW Audiograms go up to 20,000 Hz
            ExpApp= getappdata(0,'evalue');
            ExpApp(:,2)=Loaded_Cal(:,2); % Cal Left
            ExpApp(:,3)=Loaded_Cal(:,3); % Cal Right
            ExpApp(:,5)=Resultant(:,3); % Column 5: Resultant L-Small
            ExpApp(:,17)=Resultant(:,2); % Column 17: Resultant L-Large
            ExpApp(:,6)=Resultant(:,5); % Column 6: Resultant R-Small
            ExpApp(:,18)=Resultant(:,4); % Column 18: Resultant R-Large
            setappdata(0,'evalue',ExpApp) % Placeholder for Important Values
        end

% setappdata(0,'evalue',Loaded_Cal);
Plotting_Function
refresh(MainWindow);
end
%% SAVE DATA
function Save_XLS(~,~,MainWindow,u00,u0,u1,u2)
Cal_File=getappdata(0,'evalue');
CalOutput(:,1)=Cal_File(:,1);CalOutput(:,2)=Cal_File(:,2);CalOutput(:,3)=Cal_File(:,3);
CalFileColumns = {'Frequency (Hz)','Left Channel','Right Channel'};
OutputColumns = {'Frequency (Hz)','Left_Large','Left_Small','Right_Large','Right_Small','Left_NotHeard','Right_NotHeard'};
[filename, pathname] = uiputfile({'*.xls'}, 'Save Workspace as');
```

```
Audiogram_ColL = {'Frequency (Hz)','Left Ear','Level'};
Audiogram_ColR = {'Frequency (Hz)','Left Ear','Level'};
% AUDIOGRAM DATA
        cnames = {'Left','Level'};
        Freq=[20, 25, 31.5, 40, 50, 63, 80, 100, 125, 160, 200, 250, 315, 400, 500, 630, 800, 1000, 1250, 1600, 2000, 2500, 3150, 4000, 5000,
6300, 8000, 10000, 12500, 16000, 20000];
        rnames = {'20 Hz','25 Hz','31.5 Hz','40 Hz','50 Hz','63 Hz','80 Hz','100 Hz','125 Hz','160 Hz','200 Hz','250 Hz','315 Hz','400
Hz','500 Hz','630 Hz','800 Hz','1 kHz','1.25 kHz','1.6 kHz','2 kHz','2.5 kHz','3.15 kHz','4 kHz','5 kHz','6.3 kHz','8 kHz','10 kHz','12.5
kHz','16 kHz','20 kHz'};
        dat1(:,1)=(Resultant(:,3)-ReferenceContour(:,2));
        dat2(:,1)=(Resultant(:,5)-ReferenceContour(:,2));
        for i=1:31

            if dat1(i,1)<=15;
                dat1b(i,1)={sprintf('Normal Hearing')};
            end
            LvlCheck=-10;
            if dat1(i,1)<LvlCheck;
                dat1b(i,1)={sprintf('N/A')};
            end


            if dat1(i,1)>15;
                dat1b(i,1)={sprintf('Slight Hearing Loss')};
                if dat1(i,1)>25;
                    dat1b(i,1)={sprintf('Mild Hearing Loss')};
                    if dat1(i,1)>40;
                        dat1b(i,1)={sprintf('Moderate Hearing Loss')};
                        if dat1(i,1)>55;
                            dat1b(i,1)={sprintf('Moderately Severe Hearing Loss')};
                            if dat1(i,1)>70;
                                dat1b(i,1)={sprintf('Severe Hearing Loss')};
                                if dat1(i,1)>90;
                                    dat1b(i,1)={sprintf('Profound Hearing Loss')};
                                end
                            end
                        end
                    end
                end
            end
            if dat2(i,1)<=15;
                dat2b(i,1)={sprintf('Normal Hearing')};
            end
            if dat2(i,1)<LvlCheck;
                dat2b(i,1)={sprintf('N/A')};
            end
            if dat2(i,1)>15;
                dat2b(i,1)={sprintf('Slight Hearing Loss')};
                if dat2(i,1)>25;
                    dat2b(i,1)={sprintf('Mild Hearing Loss')};
                    if dat2(i,1)>40;
                        dat2b(i,1)={sprintf('Moderate Hearing Loss')};
                        if dat2(i,1)>55;
                            dat2b(i,1)={sprintf('Moderately Severe Hearing Loss')};
                            if dat2(i,1)>70;
                                dat2b(i,1)={sprintf('Severe Hearing Loss')};
                                if dat2(i,1)>90;
                                    dat2b(i,1)={sprintf('Profound Hearing Loss')};
                                end
                            end
                        end
                    end
                end
            end
        end
for i=1:31
    dat1(i,1)=str2double(sprintf('%.0f',dat1(i,1)));
    dat2(i,1)=str2double(sprintf('%.0f',dat2(i,1)));
end

dataset1=[num2cell(dat1),dat1b];
dataset2=[num2cell(dat2),dat2b];

%     if dB_HL > 90; dB_HL_Text = 'Profound Hearing Loss'; end
%     if dB_HL <= 90; dB_HL_Text = 'Severe Hearing Loss'; end
%     if dB_HL <= 70; dB_HL_Text = 'Moderately Severe Hearing Loss'; end
%     if dB_HL <= 55; dB_HL_Text = 'Moderate Hearing Loss'; end
%     if dB_HL <= 40; dB_HL_Text = 'Mild Hearing Loss'; end
%     if dB_HL <= 25; dB_HL_Text = 'Slight Hearing Loss'; end
%     if dB_HL <= 15; dB_HL_Text = 'Normal Hearing'; end



if isequal(filename,0) || isequal(pathname,0)
    disp('') %Do Nothing when cancelled or closed
else
    fname = fullfile(pathname, filename);
    xlswrite(fname, OutputColumns, 'Output', 'B2');
    xlswrite(fname, Resultant, 'Output', 'B3');
    xlswrite(fname, CalFileColumns, 'CalFile', 'B2');
    xlswrite(fname, CalOutput, 'CalFile', 'B3');

    xlswrite(fname, Audiogram_ColL, 'Audiogram', 'B2');
    xlswrite(fname, Freq.', 'Audiogram', 'B3');
    xlswrite(fname, dataset1, 'Audiogram', 'C3');

    xlswrite(fname, Audiogram_ColR, 'Audiogram', 'F2');
    xlswrite(fname, Freq.', 'Audiogram', 'F3');
    xlswrite(fname, dataset2, 'Audiogram', 'G3');

% mlrange='E1:F15';   % or similar
%
% Excel = actxserver ('Excel.Application');
% Excel.Workbooks.Open(fname);
% TargetSheet = get(Excel.sheets,'item','A');
% TargetSheet.Activate
% ran = Excel.Activesheet.get('Range',mlrange);
%
% ran.font.Color=hex2dec('FF0000');    % works fine
% % ran.BorderAround;                          %doesn't seem to do anything
%
% Sheet.get('Range','A:A').Border.Item('xlEdgeRight').LineStyle = 3
% % %Other option that I could not get to work, not sure how to pass the arguments correctly...:
% % ran.Borders(xlEdgeBottom).LineStyle = 'xlContinuous'
```

```
% % ran.Borders(xlEdgeBottom).Weight = 'xlThick'

    refresh(MainWindow);

end
end
%% RETURN TO PROGRAM
    function Return(~,~,MainWindow)
        % Check to see if Audiogram has been measured/loaded

        % Set Audiogram to memory

        % Call function in ExperimentGUI to update plot

        refresh(MainWindow);
        close
    end

%% Introduction to tones
    function playthrough(~,~)

    end
end
```

A.1.2 Signal Generator Code (PlayTone_adj_v3.m)

```
% Create a sound signal at a given amplitude and frequency for playback on a specific ear. PlotTone (freq., fade_duration, duration,
amplitude_dB, filename, ear)
% PlotTone (10000, 300, 2000, 1,'10000Hz_tone.wav',L,1)
%[time,Left,Right,Fs]=PlotTone_adj_v3 (1000, 200, 1000, [40 40], 'L', 1, 0);

% NOTE: Monaural WAV files may not play as a monaural file over some laptop
% speakers. This is a hardware problem, not a problem with the MATLAB
% Signal generation (Verified - 2014-05-28)

function [  t, left, right, fs] = PlotTone_adj_v3 (frequency, fade_duration, duration, amplitude_dB, ear, on_off, phase)

          %frequency                          % pure tone frequency (Hz)
          %fade_duration                      % fade-in and fade-out duration (ms)
          %duration                           % pure tone duration (ms)
          %amplitude_dB                       % pure tone amplitude in decibels
          %filename                           % name which wav file is to be saved as (string value, ex: 'name.wav')
          %ear                                % mono channel for wav signal (L or R)
          %on_off                             % sound playback either on (1) or off (0)
          %phase = pi/3;                                % pure tone phase (rad/sec)

          fs   = 65536;                                % sampling frequency (Hz)
          amplitudeL = exp((amplitude_dB(1,1)-90.753)/8.6848); % pure tone amplitude
          amplitudeR = exp((amplitude_dB(1,2)-90.753)/8.6848); % pure tone amplitude

          fade_window1 = @(N)( hanning(N).^2 );          % fade-in and fade-out window function handle

          % PlotTone_adj (19, 200, 5000, [90,90], 'B', 1,0);

          %%
            default_phase= pi/3;

          % generate a pure tone using program by Author: Kamil Wojcicki, UTD, November 2011 (reproduced at bottom
          [ toneL, time ] = tone_generator( fs, duration, amplitudeL, frequency, default_phase, fade_duration, fade_window1 );
          [ toneR, time ] = tone_generator( fs, duration, amplitudeR, frequency, (default_phase+phase), fade_duration, fade_window1 );
            t=time.';

              % Check for which ear indicator was used (isequal returns a value of '1' for a true match)
              a = isequal(ear, 'left'); b = isequal(ear, 'L'); c = isequal(ear, 'Left');
              d = isequal(ear, 'right'); e = isequal(ear, 'R'); f = isequal(ear, 'Right');
              m = isequal(ear, 'binaural'); n = isequal(ear, 'B'); o = isequal(ear, 'Binaural');

                % Creating Stero Audio files from tone generator with a mono-signal on either the left or right channel.
                    pad=zeros(length(toneL),1);     % blank channel

              if (a+b+c)>0; Ear=1; %Left ear was specified
                    left=[toneL.',pad];       % add blank channel so right is silent
                    right=[pad,pad];

              elseif (d+e+f)>0; Ear=2; %Right ear was specified
                    right=[pad,toneR.'];     % add blank channel so left is silent
                    left=[pad,pad];

              elseif (m+n+o)>0; Ear=3; %Binaural was specified
                   left=[toneL.'];
                   right=[toneR.'];
                    binaural=[toneL.',toneR.'];

              else % Display error message for incorrect value.
                  error('Incorrect value for "EAR" selection. For right ear use either: R, right, or Right; for left ear use either: L, left,
or Left')

              end

function [ tone, time ] = tone_generator( sampling_frequency, duration, amplitudes, frequencies, phases, fade_durations, fade_windows )
%   Tone generator function adopted from online source
%   Author: Kamil Wojcicki, UTD, November 2011. <----------*********
%
% Copyright (c) 2011, Kamil Wojcicki
% All rights reserved.
%
% Redistribution and use in source and binary forms, with or without
% modification, are permitted provided that the following conditions are
% met:
%
%     * Redistributions of source code must retain the above copyright
%       notice, this list of conditions and the following disclaimer.
%     * Redistributions in binary form must reproduce the above copyright
%       notice, this list of conditions and the following disclaimer in
%       the documentation and/or other materials provided with the distribution
%
```

```
% THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
% AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
% IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
% ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
% LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
% CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
% SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
% INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
% CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
% ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
% POSSIBILITY OF SUCH DAMAGE.

    % sampling period (s)
    sampling_period = 1/sampling_frequency;

    % time vector
    time = [ 0:sampling_period:duration*1E-3 ];

    % generate separate tone (as row vectors)
    tone = diag( amplitudes ) * cos( 2*pi*diag(frequencies)*repmat(time,length(frequencies),1) + repmat(phases.',1,length(time)) );

    % combine sinusoid components
    tone = sum( tone, 1 );

    % return if fade-in and fade-out is not needed
    if nargin==5 || ( islogical(fade_durations) && fade_durations==false ), return; end;

    % fade signal boundaries
    tone = fade( tone, sampling_frequency, fade_durations, fade_windows );
    % EOF
end

function signal = fade( signal, fs, T, window )
% FADE Fade (taper) signal at extremities.
%
%   FADE(SIGNAL,FS,T,WINDOW) returns SIGNAL with leading and trailing
%   samples faded-in and faded-out, respectively. The signal is sampled
%   at FS Hz. The fade-in and fade-out durations are specified in T.
%   The window function used for fading is supplied in WINDOW as a
%   function handle.
%
%   Inputs
%           SIGNAL input signal a vector.
%
%           FS sampling frequency (Hz).
%
%           T fade-in and fade-out durations (ms), as scalar (if same),
%             or as a two element vector.
%
%           WINDOW function handles for window functions used for
%                  fading-in and fading-out of the input signal,
%                  as a single function handle, if same window
%                  is to be used for fading-in and fading-out,
%                  or as a two element cell array of window function
%                  handles, if different window functions are to be used.
%
%   Outputs
%           SIGNAL input signal with faded leading and trailing samples.
%
%   Examples
%           clear all; close all; clc;          % clear MATLAB environment
%
%           fs = 16E3;                          % signal sampling frequency (Hz)
%           duration = 1E3;                     % signal duration (ms)
%           length = round(duration*1E-3*fs);   % signal length (samples)
%           time = [ 0:length-1 ] / fs;         % signal time vector (s)
%           fade_durations = [ 350 100 ];       % fade-in and fade-out durations (ms)
%
%           % fade-in and fade-out window function handles
%           fade_windows = { @(N)(hanning(N).^2) @(N)(chebwin(N,100)) };
%
%           original = ones( 1, length );
%           faded = fade( original, fs, fade_durations, fade_windows );
%
%           % plot generated tones
%           hfig = figure( 'Position', [ 10 10 500 300 ], 'PaperPositionMode', 'auto', 'color', 'w');
%           plot( time, original, 'r--', 'linewidth', 1.25 ); hold on;
%           plot( time, faded, 'b' );
%           xlim( [ min(time) max(time) ] );
%           ylim( [ min(faded)-0.05*max(faded) 1.05*max(faded) ] );
%           xlabel( 'Time (s)', 'FontSize', 7 );
%           ylabel( 'Amplitude', 'FontSize', 7 );
%           hleg = legend( 'Original', 'Tapered', 4 );
%           set( hleg, 'box', 'off', 'Position', get(hleg,'Position')-[0.05 0 0 0] );
%           set( gca, 'box', 'off', 'FontSize', 7 );
%
%           % print figure to png file
%           print( '-dpng', 'fade.png' );

%   Author: Kamil Wojcicki, UTD, November 2011.
%
% Copyright (c) 2011, Kamil Wojcicki
% All rights reserved.
%
% Redistribution and use in source and binary forms, with or without
% modification, are permitted provided that the following conditions are
% met:
%
%     * Redistributions of source code must retain the above copyright
%       notice, this list of conditions and the following disclaimer.
%     * Redistributions in binary form must reproduce the above copyright
%       notice, this list of conditions and the following disclaimer in
%       the documentation and/or other materials provided with the distribution
%
% THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
% AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
% IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
% ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
% LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
% CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
% SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
% INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
% CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
% ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
```

```
% POSSIBILITY OF SUCH DAMAGE.


    % check that required inputs have been supplied
    if nargin~=4, help(mfilename); return; end;

    % check that input is a vector
    if min( size(signal) )>1
        error( 'Input signal has to be a vector.\n' );
    end

    % if only one fade was duration specified,
    % use it for both fade-in and fade-out durations
    if length(T)==1, T = [ T T ]; end;

    % if only one window was specified,
    % use it for both fade-in and fade-out windows
    if length(window)==1, window = { window window }; end;

    % get input signal length
    L = length( signal );

    % determine fade durations (samples)
    N = round( T*1E-3*fs );

    % determine whether the input signal is in row or column form
    if size(signal,1)==1, form='row';
    else size(signal,2)==1, form='col';
    end

    % design fade-in window
    fadein = fade_window( N(1), 'fadein', window{1}, form );

    % generate fade-out indices
    index = 1:N(1);

    % apply fade-in window
    signal(index) = signal(index) .* fadein;


    % design fade-out window
    fadeout = fade_window( N(2), 'fadeout', window{2}, form );

    % generate fade-out indices
    index = L-N(2)+1:L;

    % apply fade-out window
    signal(index) = signal(index) .* fadeout;

end

function window = fade_window( N, type, custom, form )
% FADE_WINDOW Design window for signal fading-in or fading-out.
%
%   FADE_WINDOW(N,TYPE,CUSTOM) returns WINDOW of length N samples
%   for leading or trailing sample tapering of a signal as specified
%   by TYPE. Custom window function can be specified.
%
%   Inputs
%           N fading duration (samples).
%
%           TYPE fading direction as string,
%                i.e., 'fade-in' or 'fade-out'
%
%           CUSTOM function handle for tapering window function to
%                  be used for fade-in or fade-out window design.
%                  Note that symmetric tapered window is assumed,
%                  and that for design purposes window of double
%                  the length requested will be generated and then
%                  cut in half to retain the slope-up portion.
%
%           FORM specifies (as string) if the window should be
%                in row (i.e., 'row') or column (i.e., 'col') form.
%
%   Outputs
%           WINDOW window to be used for fading.

%   Author: Kamil Wojcicki, UTD, November 2011.

    % generate window samples
    if false
        % some hard-coded examples
%       window = triang( 2*N );
%       window = chebwin( 2*N, 100 );
        window = hanning( 2*N ).^2;
%       window = exp( linspace( -4,0,N ) );
    else
        % use a custom window instead
        window = custom( 2*N );
    end

    % ensure correct vector form
    switch lower( form )
    case 'col', window = window(:);
    case 'row', window = window(:).';
    end

    % truncate to half length (if 2*N was used)
    window = window(1:N);

    % scale to unit magnitude
    window = window / max( abs(window) );

    % time reverse for 'fade-out' option, ignore otherwise
    switch lower( type )
    case { 'fadeout', 'fade-out' },window = window(end:-1:1);
    end

% EOF

end
end
```

A.1.3 User Settings for Audiometer code (UserSettings.m)

```
function UserSettings ()
clc
% Callout Figure allowing for an easy change of operating settings
%   - Tone Length
%   - Pause Length
%   - Fade Value
%   - Fade Window Value

global Octave_String1 ThirdOctave_String1

%%%%%%%%%%%%%%%% CHANGE OUT!!!!!!!! %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% This information will be stored in the App Data Vector

 Octave_String1 = '31.5 Hz|63 Hz|125 Hz|250 Hz|500 Hz|1,000 Hz|2,000 Hz|4,000 Hz|8,000 Hz|16,000 Hz';
 ThirdOctave_String1 = '20 Hz|25 Hz|31.5 Hz|40 Hz|50 Hz|63 Hz|80 Hz|100 Hz|125 Hz|160 Hz|200 Hz|250 Hz|315 Hz|400 Hz|500 Hz|630 Hz|800
Hz|1,000 Hz|1,250 Hz|1,600 Hz|2,000 Hz|2,500 Hz|3,150 Hz|4,000 Hz|5,000 Hz|6,300 Hz|8,000 Hz|10,000 Hz|12,500 Hz|16,000 Hz|20,000 Hz';

Freq=[20, 25, 31.5, 40, 50, 63, 80, 100, 125, 160, 200, 250, 315, 400, 500, 630, 800, 1000, 1250, 1600, 2000, 2500, 3150, 4000, 5000, 6300,
8000, 10000, 12500, 16000, 20000];
FreqOct=[31.5, 63, 125, 250, 500, 1000, 2000, 4000, 8000, 16000];

%% Setting up Figure
set(0,'units','pixels');
RelFont = get(0,'ScreenSize'); %#ok<NASGU>
Font8=round(8/1080*RelFont(1,4)); %Font 8 on 1920x1080
Font10=round(10/1080*RelFont(1,4));
Font12=round(12/1080*RelFont(1,4));
Font16=round(16/1080*RelFont(1,4));


set(0,'Units','normalized');
scnsize = get(0,'ScreenSize'); %#ok<NASGU>
Background_Color=[0.8 0.8 0.8];

%set(MainWindow,'Visible','off')
Settings_Box= figure('units', 'normalized','Position',[0.3125 0.4 0.25 0.2],...
    'Color',Background_Color);
set(Settings_Box,'name','User Settings','numbertitle','off')
set(Settings_Box,'MenuBar','none');

a=get(Settings_Box,'Position');
%% Collecting Settings


if exist('UserSettings.txt')
        DefaultUserSettings=csvread('UserSettings.txt');
            Cur_Playback            = DefaultUserSettings(1,5);
            Cur_Pause               = DefaultUserSettings(2,5);
            Cur_Fade                = DefaultUserSettings(3,5);
            Phase                   = DefaultUserSettings(4,5);
            playtime2               = DefaultUserSettings(5,5);
            pausetime2              = DefaultUserSettings(6,5);
            Fade2                   = DefaultUserSettings(7,5);
            Phase2                  = DefaultUserSettings(8,5);
            Cal_Playback_Length     = DefaultUserSettings(9,5);
            Cal_Fade                = DefaultUserSettings(10,5);
            Increase1               = DefaultUserSettings(11,5);
            Increase2               = DefaultUserSettings(12,5);
            OctStep                 = DefaultUserSettings(13,5);
            Startvalue              = DefaultUserSettings(14,5);
            Endvalue                = DefaultUserSettings(15,5);
            Cur_Increase1           = DefaultUserSettings(16,5);
            Cur_Increase2           = DefaultUserSettings(17,5);
            Default_Oct             = DefaultUserSettings(18,5);
            Default_startvalue      = DefaultUserSettings(19,5);
            Default_endvalue        = DefaultUserSettings(20,5);

            AMeterPlayback          = DefaultUserSettings(21,5);
            AMeterFade              = DefaultUserSettings(22,5);
            AMeterIncrease1         = DefaultUserSettings(23,5);
            AMeterIncrease2         = DefaultUserSettings(24,5);
            AMeterOctValue          = DefaultUserSettings(25,5);
            AMeterStartValue        = DefaultUserSettings(26,5);
            AMeterEndValue          = DefaultUserSettings(27,5);

            Cur_Playback    = AMeterPlayback;
            Cur_Pause       = 0.2;
            Cur_Fade        = AMeterFade;
            Cur_Increase1   = AMeterIncrease1;
            Cur_Increase2   = AMeterIncrease2;
            Default_Oct     = AMeterOctValue;
            Default_startvalue  = AMeterStartValue;
            Default_endvalue    = AMeterEndValue;

            % Threshold values as per ISO 226:2003, reproduced in ANSI S3.4:2007
            ReadDefaultSettings=csvread('UserSettings.txt');
            ReferenceContour =[Freq.',ReadDefaultSettings(:,4)];
            MAX_SPL_Lim =[Freq.',ReadDefaultSettings(:,3)]; % Load Maximum SPL Limit for Each Frequency
            AmbientSPL=[Freq.',ReadDefaultSettings(:,1),ReadDefaultSettings(:,2)]; % Load Reference

            ExpAppZeros = getappdata(0,'evalue');
            OLDvNEW=size(DefaultUserSettings);
            ExpSize=size(ExpAppZeros);
            Comp1 = ExpSize(1,1);
            Comp2 = OLDvNEW(1,1);

            if Comp1 > Comp2;
                O_E_Diff = ExpSize(1,1) - OLDvNEW (1,1)
                for delt = Comp2:Comp1  % i.e. 31, 32, 33, 34, 35, 36
                    ExpAppZeros(delt,4)=0;
                    ExpAppZeros(delt,11)=0;
                    ExpAppZeros(delt,12)=0;
                    ExpAppZeros(delt,13)=0;
                    ExpAppZeros(delt,14)=0;
                end
                setappdata(0,'evalue',ExpAppZeros)
            else
                ExpAppZeros(:,4)=DefaultUserSettings(:,5);
```

```
                ExpAppZeros(:,11)=AmbientSPL(:,2);
                ExpAppZeros(:,12)=AmbientSPL(:,3);
                ExpAppZeros(:,13)=MAX_SPL_Lim(:,2);
                ExpAppZeros(:,14)=ReferenceContour(:,2);
                setappdata(0,'evalue',ExpAppZeros)
            end

else
    Cur_Playback        = 1;
    Cur_Pause           = 0.2;
    Cur_Fade            = 200;
    ReferenceContour    =
[20,78.5;25,68.7;31.5,59.5;40,51.1;50,44;63,37.5;80,31.5;100,26.5;125,22.1;160,17.9;200,14.4;250,11.4;315,8.60;400,6.20;500,4.40;630,3;800,2.2
0;1000,2.20;1250,3.50;1600,1.70;2000,-1.30;2500,-4.20;3150,-6;4000,-5.40;5000,-1.50;6300,6;8000,12.6;10000,13.9;12500,12.3;16000,0;20000,0;];
    MAX_SPL_Lim         = [20,60;   25,60;  31.5,60;    40,65;  50,80;  63,80;    80,80;    100,80;   125,80;   160,80;   200,80;   250,80;
315,80;   400,80;   500,80;   630,80;  800,80;   1000,80;  1250,80;  1600,80;  2000,80; 2500,80; 3150,80;    4000,80;    5000,80;    6300,80;
8000,80;  10000,80;     12500,80;     16000,80;    20000,80;];
    AmbientSPL          = [Freq.',zeros(31,1),zeros(31,1)]; % Load Reference
    Cur_Increase1       = 5;
    Cur_Increase2       = 2.5;
    Default_Oct         = 1;
    Default_startvalue  = 3;
    Default_endvalue    = 29;

end

%% Settings Buttons and Edit Boxes

%Button Positioning

%1=2*Left_Align+Width1+HoriSpace+Width2
%Left_Align=0.5*(1-Width1+HoriSpace+Width2);

Bottom_Align=0.8;
Width1=0.35/2;
Width2=0.3/4;
Width3=0.35/6;
Height1=(0.075);
Height2=(0.09);
dh=(Height2-Height1)*0.5;

Bottom_Block=0.35;
HoriSpace=0.05;

VertSpace=(1-Bottom_Block-5*Height2)*(1/5);
Left_Align=0.075;
UnitAlign = (Left_Align+Width1+HoriSpace+Width2+0.01);

%% Variable Descriptions
    function PlaybackAbout (~,~)
        TH=0.2; %Text Height
        TL1=0.1;   %Text Left
        TW1=0.2; %Text Width - Variable
        TW2=(1-TL1-TW1-TL1); %Text Width - Description
        VS=(1/5)*(1-4*TH); %Vertical Spacing
        TB1=1-VS-TH; %Text Bottom

        AboutBox=figure;
        set(AboutBox,'name','Variable Descriptions','numbertitle','off')
        set(AboutBox,'MenuBar','none');

        PlayDesc='The Playback length determines how long each pure tone will be presented during the assessment';
        PauseDesc='The Pause length determines how the pause will last between tone presentations';
        FadeDesc=' Fade defines the leading and trailing faded-in and faded-out lengths used during the signal presentation. The fade-in and
fade-out durations are specified in unit milliseconds. ';
        Increase1Desc='The large increase defines the initial iteration steps used to gather a rough estimate of the threshold of hearing for
the participant, (default 5 decibels (dB)).';
        Increase2Desc='Once the large increase has been perceived by the participant, the amplitude drops back down and the proceedure
restarts with the smaller iteration value. Using this approach a higher resolution may be obtained for the participants threshold contour.';
        OctavesDesc='Frequency values progress at every octave. This approach has the benefit of quickly deriving the hearing threshold shape,
only with a relatively low resolution.';
        ThirdOctavesDesc='Frequency values progress at every 1/3 octave. This setting allows for a higher resolution contour but may be more
time consuming.';
        LowerFreqDesc='The lower frequency which the low to high approach starts from.';
        UpperFreqDesc='The upper frequency which the low to high approach ends on. Note that the upper frequency must be higher than the lower
frequency limit to operate correctly. For the case where the user specifies an upper limit which is less than the lower limit, the program
will automatically set the upper and lower limits the same.';

        Playback_Desc = uicontrol('style','text', 'units', 'normalized',...
            'position', [TL1 TB1 TW1 TH], 'string', 'Playback Length',...
            'visible','on', 'FontSize', Font10,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left','FontWeight','Bold');
        Playback_Desc2 = uicontrol('style','text', 'units', 'normalized',...
            'position', [TL1+TW1 TB1 TW2 TH], 'string', PlayDesc,...
            'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left');

        Pause_Desc = uicontrol('style','text', 'units', 'normalized',...
            'position', [TL1 TB1-(VS+TH) TW1 TH], 'string', 'Pause Length',...
            'visible','on', 'FontSize', Font10,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left','FontWeight','Bold');
        Pause_Desc2 = uicontrol('style','text', 'units', 'normalized',...
            'position', [TL1+TW1 TB1-(VS+TH) TW2 TH], 'string', PauseDesc,...
            'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left');

        Fade_Desc = uicontrol('style','text', 'units', 'normalized',...
            'position', [TL1 TB1-2*(VS+TH) TW1 TH], 'string', 'Fade Length',...
            'visible','on', 'FontSize', Font10,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left','FontWeight','Bold');
        Fade_Desc2 = uicontrol('style','text', 'units', 'normalized',...
            'position', [TL1+TW1 TB1-2*(VS+TH) TW2 TH], 'string', FadeDesc,...
            'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left');

        Increase1_Desc = uicontrol('style','text', 'units', 'normalized',...
            'position', [TL1 TB1-3*(VS+TH) TW1 TH], 'string', 'Small Increase',...
            'visible','on', 'FontSize', Font10,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left','FontWeight','Bold');
        Increase1_Desc2 = uicontrol('style','text', 'units', 'normalized',...
            'position', [TL1+TW1 TB1-3*(VS+TH) TW2 TH], 'string', Increase1Desc,...
            'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left');

        Increase2_Desc = uicontrol('style','text', 'units', 'normalized',...
            'position', [TL1 TB1-4*(VS+TH) TW1 TH], 'string', 'Large Increase',...
            'visible','on', 'FontSize', Font10,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left','FontWeight','Bold');
        Increase2_Desc2 = uicontrol('style','text', 'units', 'normalized',...
            'position', [TL1+TW1 TB1-4*(VS+TH) TW2 TH], 'string', Increase2Desc,...
            'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left');

    end
```

```
PlaybackMenu=uicontextmenu;
item1=uimenu(PlaybackMenu,'Label','What''s This?','Callback',@PlaybackAbout);
%% Setting Controls
GeneralFrame = uicontrol('style','frame', 'units', 'normalized',...
    'position', [0.05 0.25 0.40 0.70], 'visible','on', 'FontSize', Font10,'BackgroundColor',[0.8 0.8
0.8],'HorizontalAlignment','Left','FontWeight','Bold');

Frame1_Text = uicontrol('style','text', 'units', 'normalized',...
    'position', [0.075 0.90 Width1 Height1], 'string', 'General Settings',...
    'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left');

Playback_Text = uicontrol('style','text', 'units', 'normalized',...
    'position', [Left_Align Bottom_Align+dh Width1 Height1], 'string', 'Playback Length',...
    'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left');
Playback_Edit = uicontrol('style','edit', 'units', 'normalized',...
    'position', [Left_Align+Width1+HoriSpace Bottom_Align Width2 Height2], 'string', Cur_Playback,...
    'visible','on', 'FontSize', Font8,'UIContextMenu',PlaybackMenu);
Playback_Units = uicontrol('style','text', 'units', 'normalized',...
    'position', [UnitAlign Bottom_Align+dh Width3 Height1], 'string', 'Sec.',...
    'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left');

% Pause_Text = uicontrol('style','text', 'units', 'normalized',...
%     'position', [Left_Align Bottom_Align-(VertSpace+Height2)+dh Width1 Height1], 'string', 'Pause Length',...
%     'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left');
% Pause_Edit = uicontrol('style','edit', 'units', 'normalized',...
%     'position', [Left_Align+Width1+HoriSpace Bottom_Align-(VertSpace+Height2) Width2 Height2], 'string', Cur_Pause,...
%     'visible','on', 'FontSize', Font8,'UIContextMenu',PlaybackMenu);
% Pause_Units = uicontrol('style','text', 'units', 'normalized',...
%     'position', [UnitAlign Bottom_Align-(VertSpace+Height2)+dh Width3 Height1], 'string', 'Sec.',...
%     'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left');


Fade_Text = uicontrol('style','text', 'units', 'normalized',...
    'position', [Left_Align Bottom_Align-1*(VertSpace+Height2)+dh Width1 Height1], 'string', 'Fade Length',...
    'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left');
Fade_Edit = uicontrol('style','edit', 'units', 'normalized',...
    'position', [Left_Align+Width1+HoriSpace Bottom_Align-1*(VertSpace+Height2) Width2 Height2], 'string', Cur_Fade,...
    'visible','on', 'FontSize', Font8,'UIContextMenu',PlaybackMenu);
Fade_Units = uicontrol('style','text', 'units', 'normalized',...
    'position', [UnitAlign Bottom_Align-1*(VertSpace+Height2)+dh Width3 Height1], 'string', 'ms.',...
    'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left');

Increase1_Text = uicontrol('style','text', 'units', 'normalized',...
    'position', [Left_Align Bottom_Align-2*(VertSpace+Height2)+dh Width1 Height1], 'string', 'Large Increase',...
    'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left');
Increase1_Edit = uicontrol('style','edit', 'units', 'normalized',...
    'position', [Left_Align+Width1+HoriSpace Bottom_Align-2*(VertSpace+Height2) Width2 Height2], 'string', Cur_Increase1,...
    'visible','on', 'FontSize', Font8,'UIContextMenu',PlaybackMenu);
Increase1_Units = uicontrol('style','text', 'units', 'normalized',...
    'position', [UnitAlign Bottom_Align-2*(VertSpace+Height2)+dh Width3 Height1], 'string', 'dB.',...
    'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left');


Increase2_Text = uicontrol('style','text', 'units', 'normalized',...
    'position', [Left_Align Bottom_Align-3*(VertSpace+Height2)+dh Width1 Height1], 'string', 'Small Increase',...
    'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left');
Increase2_Edit = uicontrol('style','edit', 'units', 'normalized',...
    'position', [Left_Align+Width1+HoriSpace Bottom_Align-3*(VertSpace+Height2) Width2 Height2], 'string', Cur_Increase2,...
    'visible','on', 'FontSize', Font8,'UIContextMenu',PlaybackMenu);
Increase2_Units = uicontrol('style','text', 'units', 'normalized',...
    'position', [UnitAlign Bottom_Align-3*(VertSpace+Height2)+dh Width3 Height1], 'string', 'dB.',...
    'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left');

R_Width1=0.15;
DB_Width=0.20;

AutomaticFrame = uicontrol('style','frame', 'units', 'normalized',...
    'position', [0.5 0.25 0.45 0.70], 'visible','on', 'FontSize', Font10,'BackgroundColor',[0.8 0.8
0.8],'HorizontalAlignment','Left','FontWeight','Bold');

Frame2_Text = uicontrol('style','text', 'units', 'normalized',...
    'position', [0.525 0.90 0.29 Height1], 'string', 'Automatic Program Settings',...
    'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left');

Oct_Text = uicontrol('style','text', 'units', 'normalized',...
    'position', [0.525 0.8 R_Width1*1.6 Height1], 'string', 'Octave Steps',...
    'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left');

Oct_Group = uibuttongroup('Visible','on','Position',[0.525 0.5 0.4 0.3]);
Oct_u0 = uicontrol('Style','Radio','String','1/1 Octaves',...
    'units', 'normalized','pos',[0.1 0.75 0.8 0.28],'parent',Oct_Group,...
    'HandleVisibility','on','BackgroundColor',[0.8 0.8 0.8], 'Enable', 'Inactive');
Oct_u1 = uicontrol('Style','Radio','String','1/3 Octaves',...
    'units', 'normalized','pos',[0.1 0.45 0.8 0.28],...
    'parent',Oct_Group,'HandleVisibility','on','BackgroundColor',[0.8 0.8 0.8], 'Enable',
'Inactive','ButtonDownFcn',{@UpdateFreq2,Oct_u0},'Value',Default_Oct);

%Defining functions for when RadioButtons are Pressed
%  Had to wait until both functions were defined as they refer to each
%  other
set(Oct_u0,'ButtonDownFcn',{@UpdateFreq1,Oct_u1})
set(Oct_u1,'ButtonDownFcn',{@UpdateFreq2,Oct_u0})


Freq_Text = uicontrol('style','text', 'units', 'normalized',...
    'position', [0.525 0.51 R_Width1*1.6 Height1], 'string', 'Frequency Limits',...
    'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left');

%%%% ALSO REPEATED BELOW
% Octave_String1 = '31.5 Hz|63 Hz|125 Hz|250 Hz|500 Hz|1,000 Hz|2,000 Hz|4,000 Hz|8,000 Hz';
% ThirdOctave_String1 = '20 Hz|25 Hz|31.5 Hz|40 Hz|50 Hz|63 Hz|80 Hz|100 Hz|125 Hz|160 Hz|200 Hz|250 Hz|315 Hz|400 Hz|500 Hz|630 Hz|800
Hz|1,000 Hz|1,250 Hz|1,600 Hz|2,000 Hz|2,500 Hz|3,150 Hz|4,000 Hz|5,000 Hz|6,300 Hz|8,000 Hz|10,000 Hz|12,500 Hz';

Oct_Value0=get(Oct_u0,'Value');
Oct_Value1=get(Oct_u1,'Value');
if Oct_Value0==1
    Drop_String=Octave_String1;
end
if Oct_Value1==1
    Drop_String=ThirdOctave_String1;
end
```

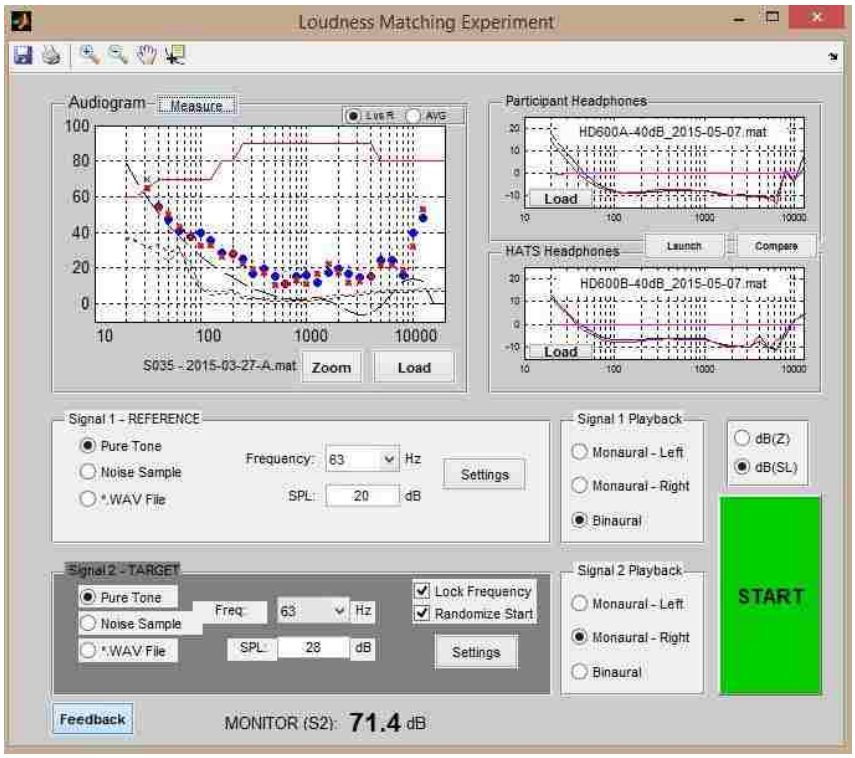
```
%% Testing

LowerFreq_Text = uicontrol('style','text', 'units', 'normalized',...
    'position', [0.575 0.43 R_Width1 Height1], 'string', 'Lower Freq.',...
    'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left');
LowerFreq_DropBox = uicontrol('style','popup', 'units', 'normalized',...
    'position', [0.575+R_Width1 0.4375 DB_Width Height1], 'string', Drop_String,...
    'visible','on', 'FontSize', Font8,'Value',Default_startvalue);

UpperFreq_Text = uicontrol('style','text', 'units', 'normalized',...
    'position', [0.575 0.325 R_Width1 Height1], 'string', 'Upper Freq.',...
    'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left');
UpperFreq_DropBox = uicontrol('style','popup', 'units', 'normalized',...
    'position', [0.575+R_Width1 0.3275 DB_Width Height1], 'string', Drop_String,...
    'visible','on', 'FontSize', Font8,'Value',Default_endvalue);

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            %%%%%%%  Frequency Limit Check  %%%%%%%%%%%%%%%%%%
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            set(LowerFreq_Text,'Callback',{@quickCorrLow2});
            set(UpperFreq_Text,'Callback',{@quickCorrHigh2});

            function quickCorrLow2(~,~)
                check_f1 = get(LowerFreq_Text,'Value');
                check_f2 = get(UpperFreq_Text,'Value');

                if check_f2<check_f1;
                    check_f1=check_f2;
                    set(LowerFreq_Text,'Value',check_f1);
                end
            end

            function quickCorrHigh2(~,~)
                check_f1 = get(LowerFreq_Text,'Value');
                check_f2 = get(UpperFreq_Text,'Value');

                if check_f2<check_f1;
                    check_f2=check_f1;
                    set(UpperFreq_Text,'Value',check_f2);
                end
            end

% Functions Called when RadioButtons are pressed
%  Turn on button, update lower and upper frequency options with respective
%  octave/third octave values.
    function UpdateFreq1(~,~,Oct_u1)
        set(Oct_u0,'Value',[1])
                delete(LowerFreq_DropBox)
        delete(UpperFreq_DropBox)

        % Check Radiobuttons again
        Oct_Value0=get(Oct_u0,'Value');
        Oct_Value1=get(Oct_u1,'Value');

        if Oct_Value0==1                    % 1/1 OCTAVES
            Drop_String=Octave_String1;
            endvalue=10;
        end
        if Oct_Value1==1                    % 1/3 OCTAVES
            Drop_String=ThirdOctave_String1;
            endvalue=31;
        end


        LowerFreq_DropBox = uicontrol('style','popup', 'units', 'normalized',...
            'position', [0.575+R_Width1 0.4375 DB_Width Height1], 'string', Drop_String,...
            'visible','on', 'FontSize', Font8);
%         if endvalue==31; set(LowerFreq_DropBox,'Value',3); end
        UpperFreq_DropBox = uicontrol('style','popup', 'units', 'normalized',...
            'position', [0.575+R_Width1 0.3275 DB_Width Height1], 'string', Drop_String,...
            'visible','on', 'FontSize', Font8,'Value',endvalue);

             %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            %%%%%%%  Frequency Limit Check  %%%%%%%%%%%%%%%%%%
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            set(LowerFreq_DropBox,'Callback',{@quickCorrLow2});
            set(UpperFreq_DropBox,'Callback',{@quickCorrHigh2});

            function quickCorrLow2(~,~)
                check_f1 = get(LowerFreq_DropBox,'Value');
                check_f2 = get(UpperFreq_DropBox,'Value');

                if check_f2<check_f1;
                    check_f1=check_f2;
                    set(LowerFreq_DropBox,'Value',check_f1);
                end
            end

            function quickCorrHigh2(~,~)
                check_f1 = get(LowerFreq_DropBox,'Value');
                check_f2 = get(UpperFreq_DropBox,'Value');

                if check_f2<check_f1;
                    check_f2=check_f1;
                    set(UpperFreq_DropBox,'Value',check_f2);
                end
            end

    end

    function UpdateFreq2(~,~,Oct_u0)
        set(Oct_u1,'Value',[1])
                delete(LowerFreq_DropBox)
        delete(UpperFreq_DropBox)
        Oct_Value0=get(Oct_u0,'Value');
        Oct_Value1=get(Oct_u1,'Value');
        if Oct_Value0==1
            Drop_String=Octave_String1;
            endvalue=10;
        end
        if Oct_Value1==1
            Drop_String=ThirdOctave_String1;
            endvalue=31;
        end
```

```
        LowerFreq_DropBox = uicontrol('style','popup', 'units', 'normalized',...
            'position', [0.575+R_Width1 0.4375 DB_Width Height1], 'string', Drop_String,...
            'visible','on', 'FontSize', Font8);
        UpperFreq_DropBox = uicontrol('style','popup', 'units', 'normalized',...
            'position', [0.575+R_Width1 0.3275 DB_Width Height1], 'string', Drop_String,...
            'visible','on', 'FontSize', Font8,'Value',endvalue);

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            %%%%%%%  Frequency Limit Check  %%%%%%%%%%%%%%%%%%
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            set(LowerFreq_DropBox,'Callback',{@quickCorrLow2});
            set(UpperFreq_DropBox,'Callback',{@quickCorrHigh2});

            function quickCorrLow2(~,~)
                check_f1 = get(LowerFreq_DropBox,'Value');
                check_f2 = get(UpperFreq_DropBox,'Value');

                if check_f2<check_f1;
                    check_f1=check_f2;
                    set(LowerFreq_DropBox,'Value',check_f1);
                end
            end

            function quickCorrHigh2(~,~)
                check_f1 = get(LowerFreq_DropBox,'Value');
                check_f2 = get(UpperFreq_DropBox,'Value');

                if check_f2<check_f1;
                    check_f2=check_f1;
                    set(UpperFreq_DropBox,'Value',check_f2);
                end
            end

    end



%% Button Positions
Button_Width =0.3;
Spacing=0.1;
Button_Bottom=0.05;
Button_Height=0.15;
Button_Left = 0.5*(1-Spacing-2*Button_Width);

%SAVE SETTINGS
Save_Button=uicontrol('style', 'pushbutton','string', 'Save','units',...
    'normalized','Visible','on','position', [Button_Left Button_Bottom Button_Width Button_Height],...
    'FontSize', Font12, 'FontWeight', 'bold','callback', {@Save_Settings});

    function Save_Settings(~,~)
        CurrentSettings= getappdata(0,'evalue');

        DefaultUserSettings=csvread('UserSettings.txt');


            NewValue_Playback        = str2num(get(Playback_Edit,'string'));
            %NewValue_Pause           = str2num(get(Pause_Edit,'string'));
            NewValue_Fade            = str2num(get(Fade_Edit,'string'));
            NewValue_Increase1       = str2num(get(Increase1_Edit,'string'));
            NewValue_Increase2       = str2num(get(Increase2_Edit,'string'));
            NewValue_Oct             = get(Oct_u1,'Value');
            NewValue_startvalue      = get(LowerFreq_DropBox,'Value');
            NewValue_endvalue        = get(UpperFreq_DropBox,'Value');



        CurrentSettings(21,4)=NewValue_Playback;
        %CurrentSettings(2,4)=NewValue_Pause;
        CurrentSettings(22,4)=NewValue_Fade;
        CurrentSettings(23,4)=NewValue_Increase1;
        CurrentSettings(24,4)=NewValue_Increase2;
        CurrentSettings(25,4)=NewValue_Oct;
        CurrentSettings(26,4)=NewValue_startvalue;
        CurrentSettings(27,4)=NewValue_endvalue;
        setappdata(0,'evalue',CurrentSettings);


          DefaultUserSettings(21,5) = NewValue_Playback;
          %DefaultUserSettings(22,5) = NewValue_Pause
          DefaultUserSettings(22,5) = NewValue_Fade;
          DefaultUserSettings(23,5) = NewValue_Increase1;
          DefaultUserSettings(24,5) = NewValue_Increase2;
          DefaultUserSettings(25,5) = NewValue_Oct;
          DefaultUserSettings(26,5) = NewValue_startvalue;
          DefaultUserSettings(27,5) = NewValue_endvalue;

            csvwrite('UserSettings.txt',DefaultUserSettings);


        disp('Settings Saved')
        close
    end

%CANCEL SETTINGS
Cancel_Button=uicontrol('style', 'pushbutton','string', 'Cancel','units',...
    'normalized','Visible','on','position', [Button_Left+Button_Width+Spacing Button_Bottom Button_Width Button_Height],...
    'FontSize', Font12, 'FontWeight', 'bold','callback', {@Cancel_Settings});

    function Cancel_Settings(~,~)
        disp('Settings Not Saved')
        close
    end

%CANCEL SETTINGS CHANGE
end
```

A.1.4 User Settings Example (UserSettings.txt)

```
35.91,35.31,60,78.5,2
36.47,35.83,60,68.7,0.25
40.19,39.16,65,59.5,200
32.81,32.47,70,51.1,0
26.61,26.1,70,44,2
29.1,26.92,70,37.5,0.25
20.49,21.14,70,31.5,200
9.07,12.15,70,26.5,0
7.13,7.61,70,22.1,10
4.56,6.14,80,17.9,200
4.22,5.94,80,14.4,0
2.58,4.44,90,11.4,0
2.18,4.26,90,8.6,0
1.95,3.41,90,6.2,0
1.66,3.47,90,4.4,0
1.95,3.37,90,3,0
2.42,3.56,90,2.2,0
2.82,4.04,90,2.4,0
2.93,4.09,90,3.5,0
3.28,4.35,90,1.7,0
3.99,5.01,90,-1.3,1
4.61,5.71,90,-4.2,200
5.46,6.64,90,-6,5
5.92,6.94,90,-5.4,2.5
6.74,7.63,80,-1.5,1
7.11,7.62,80,6,7
6.91,8.03,80,12.6,31
6.88,8.08,80,13.9,0
7.41,8.51,80,12.3,0
7.79,9.34,50,0,0
8.4,10.45,50,0,0
```

A.2 – Program Development for Flexible Loudness Matching



A.2.1 Experimental GUI & Code (ExperimentGUI_v9.m)

```
function ExperimentGUI_v9 ()
% Signal presentation method to be used at the University of Windsor for
% Loudness Comparison Experiments.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clc
global Audiogram_Data ReferenceContour Resultant Freq playtime1 pausetime1 Increase1 Increase2 Font8 Font10 Font12 Font16
Errorfigcount=1; %

%% Signal Feedback for Participant
%
SignalFeedback = figure('units', 'normalized', 'Visible', 'off',...
    'Name','Participant Feedback','NumberTitle','Off','toolbar','none','MenuBar','none');

% Background Colours used in the GUI
            Colour0 = [0.9 0.9 0.9];
            Colour1 = [0 0.3 1];
            Colour2 = [1 0 0];    % Colour formatting [R G B]

%% SETTING APP-DATA VECTOR
% Default Values to be transfered between functions and windows

Vol=0.3; % Volume of Start and Stop SOUNDS

        DefaultUserSettings=csvread('UserSettings.txt');
            playtime1               = DefaultUserSettings( 1,5);
            pausetime1              = DefaultUserSettings( 2,5);
            Fade1                   = DefaultUserSettings( 3,5);
            Phase1                  = DefaultUserSettings( 4,5);
            playtime2               = DefaultUserSettings( 5,5);
            pausetime2              = DefaultUserSettings( 6,5);
            Fade2                   = DefaultUserSettings( 7,5);
            Phase2                  = DefaultUserSettings( 8,5);
            Cal_Playback_Length     = DefaultUserSettings( 9,5);
            Cal_Fade                = DefaultUserSettings(10,5);
            Increase1               = DefaultUserSettings(11,5);
            Increase2               = DefaultUserSettings(12,5);
            OctStep                 = DefaultUserSettings(13,5);
            Startvalue              = DefaultUserSettings(14,5);
            Endvalue                = DefaultUserSettings(15,5);
            Cur_Increase1           = DefaultUserSettings(16,5);
            Cur_Increase2           = DefaultUserSettings(17,5);
            Default_Oct             = DefaultUserSettings(18,5);
            Default_startvalue      = DefaultUserSettings(19,5);
            Default_endvalue        = DefaultUserSettings(20,5);
            AMeterPlayback          = DefaultUserSettings(21,5);
            AMeterFade              = DefaultUserSettings(22,5);
            AMeterIncrease1         = DefaultUserSettings(23,5);
            AMeterIncrease2         = DefaultUserSettings(24,5);
            AMeterOctValue          = DefaultUserSettings(25,5);
            AMeterStartValue        = DefaultUserSettings(26,5);
            AMeterEndValue          = DefaultUserSettings(27,5);
            PauseCheck              = DefaultUserSettings(28,5);

            %Convert Phase angle in degrees to radians
            Phase=60*((2*pi)/60);       % Default Phase Angle

            %Phase difference added to RIGHT ear of binaural presentation
            Phase1=Phase1*((2*pi)/60);  % Binaural Phase Difference between ears on Signal 1
            Phase2=Phase2*((2*pi)/60);  % Binaural Phase Difference between ears on Signal 2



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% DEFAULT VALUES / INITIAL VALUES AND PLACEHOLDERS %%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

PlayCount   = 0;
BreakOpt    = 0;
PauseCheck  = 0;
TestdBZ     = 0;
AudFunc     = 0;


% Standard 1/3 Octave Frequencies
Freq = [20, 25, 31.5, 40, 50, 63, 80, 100, 125, 160, 200, 250, 315, 400,...
        500, 630, 800, 1000, 1250, 1600, 2000, 2500, 3150, 4000, 5000,...
        6300, 8000, 10000, 12500, 16000, 20000];


% Standard 1/1 Octave Frequencies
FreqOct = [31.5, 63, 125, 250, 500, 1000, 2000, 4000, 8000, 16000];

% Reading UserSettings File which allows user settings to be saved between
% successive program runs.
ReadDefaultSettings = csvread('UserSettings.txt');

for loadref = 1:31
    ReadDefaultSettings(loadref,1); % Default settings (playtime, fade, ect).
    ReferenceContour(loadref,1) = (Freq(1,loadref)); ReferenceContour(loadref,2) = (ReadDefaultSettings(loadref,4));
% Threshold values as per ISO 226:2003, reproduced in ANSI S3.4:2007
    MAX_SPL_Lim(loadref,1) = (Freq(1,loadref)); MAX_SPL_Lim(loadref,2) = (ReadDefaultSettings(loadref,3));
% Load Maximum SPL Limit for Each Frequency
    AmbientSPL(loadref,1) = (Freq(1,loadref)); AmbientSPL(loadref,2) = (ReadDefaultSettings(loadref,1)); AmbientSPL(loadref,3) =
(ReadDefaultSettings(loadref,2)); % Load Ambient Levels from Limits
end

% Initial Values for Audiometer plot where no data has been loaded
NegativeTens=[-20; -20; -20; -20; -20; -20; -20; -20; -20; -20; -20; -20;...
    -20; -20; -20; -20; -20; -20; -20; -20; -20; -20; -20; -20;...
    -20; -20; -20; -20; -20; -20; -20];

% ZeroVector placeholder as an initial value
ZeroVector=[0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0;...
    0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0];

% Inserting loaded reference data into the programs memory bank
ExpAppZeros =[  0 0 0 playtime1     0 0 0 0 0 0 0 0 0 0 0 0 0 0;         %1
```

```
                0 0 0 pausetime1    0 0 0 0 0 0 0 0 0 0 0 0 0 0;
                0 0 0 Fade1         0 0 0 0 0 0 0 0 0 0 0 0 0 0;
                0 0 0 Phase1        0 0 0 0 0 0 0 0 0 0 0 0 0 0;
                0 0 0 playtime2     0 0 0 0 0 0 0 0 0 0 0 0 0 0;         %5
                0 0 0 pausetime2    0 0 0 0 0 0 0 0 0 0 0 0 0 0;
                0 0 0 Fade2         0 0 0 0 0 0 0 0 0 0 0 0 0 0;
                0 0 0 Phase2        0 0 0 0 0 0 0 0 0 0 0 0 0 0;
                0 0 0 Cal_Playback_Length 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
                0 0 0 Cal_Fade      0 0 0 0 0 0 0 0 0 0 0 0 0 0;        %10
                0 0 0 Increase1     0 0 0 0 0 0 0 0 0 0 0 0 0 0;
                0 0 0 Increase2     0 0 0 0 0 0 0 0 0 0 0 0 0 0;
                0 0 0 OctStep       0 0 0 0 0 0 0 0 0 0 0 0 0 0;
                0 0 0 Startvalue    0 0 0 0 0 0 0 0 0 0 0 0 0 0;
                0 0 0 Endvalue      0 0 0 0 0 0 0 0 0 0 0 0 0 0;        %15
                0 0 0 Cur_Increase1 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
                0 0 0 Cur_Increase2 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
                0 0 0 Default_Oct   0 0 0 0 0 0 0 0 0 0 0 0 0 0;
                0 0 0 Default_startvalue 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
                0 0 0 Default_endvalue 0 0 0 0 0 0 0 0 0 0 0 0 0 0;     %20
                0 0 0 AMeterPlayback 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
                0 0 0 AMeterFade    0 0 0 0 0 0 0 0 0 0 0 0 0 0;
                0 0 0 AMeterIncrease1 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
                0 0 0 AMeterIncrease2 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
                0 0 0 AMeterOctValue 0 0 0 0 0 0 0 0 0 0 0 0 0 0;       %25
                0 0 0 AMeterStartValue 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
                0 0 0 AMeterEndValue 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
                0 0 0 PauseCheck    0 0 0 0 0 0 0 0 0 0 0 0 0 0;
                0 0 0 TestdBZ       0 0 0 0 0 0 0 0 0 0 0 0 0 0;
                0 0 0 AudFunc       0 0 0 0 0 0 0 0 0 0 0 0 0 0;        %30
                0 0 0 0             0 0 0 0 0 0 0 0 0 0 0 0 0 0
                0 0 0 0             0 0 0 0 0 0 0 0 0 0 0 0 0 0
                0 0 0 0             0 0 0 0 0 0 0 0 0 0 0 0 0 0
                0 0 0 0             0 0 0 0 0 0 0 0 0 0 0 0 0 0
                0 0 0 0             0 0 0 0 0 0 0 0 0 0 0 0 0 0         %35
                0 0 0 0             0 0 0 0 0 0 0 0 0 0 0 0 0 0];

            % Loading reference contours into the memory bank
            for loadref = 1:31
                ExpAppZeros(loadref,11) = AmbientSPL(loadref,2);        % 11th Column = Ambient Left
                ExpAppZeros(loadref,12) = AmbientSPL(loadref,3);        % 12th Column = Ambient Right
                ExpAppZeros(loadref,13) = MAX_SPL_Lim(loadref,2);       % 13th Column = MAX SPL limit for headphones
                ExpAppZeros(loadref,14) = ReferenceContour(loadref,2);  % 14th Column = Reference Threshold
            end

            % Store ExpAppZeros
            setappdata(0,'evalue',ExpAppZeros) % Placeholder for Important Values

            % evalue Placeholder Legend
                % Column 1: Frequency
                % Column 2: Audiometer Loaded Cal Left
                % Column 3: Audiometer Loaded Cal Right
                % Column 4: User Settings***
                % Column 5: Resultant L-Small (:,3)
                % Column 6: Resultant R-Small (:,5)
                % Column 7: Headphones 1 - Left
                % Column 8: Headphones 1 - Right
                % Column 9: Headphones 2 - Left
                % Column 10: Headphones 2 - Right
                % Column 11: Ambient Conditions Left
                % Column 12: Ambient Conditions Right
                % Column 13: Maximum Limits
                % Column 14: Reference Contour
                % Column 15: Loaded Binaural Cal Headphones 1
                % Column 16: Loaded Binaural Cal Headphones 2
                % Column 17: Resultant L-Large (:,2)
                % Column 18: Resultant R-Large (:,4)

%% Setting Up the Plot options for the Audiogram Data
    AudFileName=0;
    Audiogram_Data = ReferenceContour(:,1);
    Resultant      = Freq.';                %Column reserved for Frequency
    Resultant(:,2) = NegativeTens(:,1);   %Column reserved for Left Ear, Large increment
    Resultant(:,3) = NegativeTens(:,1);   %Column reserved for Left Ear, Small increment
    Resultant(:,4) = NegativeTens(:,1);   %Column reserved for Right Ear, Large increment
    Resultant(:,5) = NegativeTens(:,1);   %Column reserved for Right Ear, Small increment
    Resultant(:,6) = NegativeTens(:,1);   %Column reserved for Left Ear, Not Heard
    Resultant(:,7) = NegativeTens(:,1);   %Column reserved for Right Ear, Not Heard

% Setting spaceholder for Signal WAV File names
    S1FileName = 0; S2FileName = 0;

% Setting up the Plot options and spaceholders for the Headphone
% Calibrations
    H1FileName = 0; H2FileName = 0;

    Headphones_A = Freq.'; %Column reserved for Frequency
    Headphones_B = Freq.'; %Column reserved for Frequency

    Headphones_A(:,2) = NegativeTens(:,1); % Res. for Headphones A:Left
    Headphones_A(:,3) = NegativeTens(:,1); % Res. for Headphones A:Right
    Headphones_A(:,4) = NegativeTens(:,1); % Res. for Headphones A:Binaural

    Headphones_B(:,2) = NegativeTens(:,1); % Res. for Headphones B:Left
    Headphones_B(:,3) = NegativeTens(:,1); % Res. for Headphones B:Right
    Headphones_B(:,4) = NegativeTens(:,1); % Res. for Headphones B:Binaural


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Basic Graphical User Interface (GUI) without using GUIDE %%%%%%%%%%%%%%
%% Setting up the GUI dimensions                            %%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%FIGURE DIMENSIONS
    set(0,'units','pixels'); %Setting unit of preference to pixels
    RelFont = get(0,'ScreenSize'); %#ok<NASGU> %Gathering current screen size
    %--> %#ok<NASGU> - Tells the program to ignore specific lines of code/error

    set(0,'Units','normalized'); %Normalizing units for sizing
    scnsize = get(0,'ScreenSize'); %#ok<NASGU>

    MainWindow = figure('units', 'normalized',...
        'Position',[0.3,0.25,0.35,0.5], 'Visible', 'On',...
        'Name','Loudness Matching Experiment','NumberTitle','Off');
```

```
    set(MainWindow,'MenuBar','none'); %Turn off Menu Bar
    set(MainWindow,'Toolbar','figure'); %Turn on the figure toolbar

    player = audioplayer(0.1,80); % Setup AudioPlayer(Y,Fs), adjusted later

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%% Look into setting player as a %%%%%%%%%
    %%%% "persistent" variable type.   %%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    % KeyPress Funtion used to adjust volume
    set(MainWindow,'KeyPressFcn', {@keyPress,player});

% Adjusting the Default Figure Toolbar Options
    h1Toolbar = findall(MainWindow,'tag','FigureToolBar');
        % Removing buttons that are not needed.
        delete(findall(h1Toolbar,'tag','Standard.NewFigure'))
        delete(findall(h1Toolbar,'tag','Standard.FileOpen'))
        delete(findall(h1Toolbar,'tag','Plottools.PlottoolsOn'))
        delete(findall(h1Toolbar,'tag','Plottools.PlottoolsOff'))
        delete(findall(h1Toolbar,'tag','Annotation.InsertColorbar'))
        delete(findall(h1Toolbar,'tag','DataManager.Linking'))
        delete(findall(h1Toolbar,'tag','Standard.EditPlot'))
        delete(findall(h1Toolbar,'tag','Exploration.Rotate'))
        delete(findall(h1Toolbar,'tag','Exploration.Brushing'))
        delete(findall(h1Toolbar,'tag','Annotation.InsertLegend'))

% Setting font size to be consistent and rel. to screen size (1920x1080)
    Font6 = round(6/1080*RelFont(1,4));   % Font 6  - rel scale
    Font8 = round(8/1080*RelFont(1,4));   % Font 8  - rel scale
    Font10 = round(10/1080*RelFont(1,4)); % Font 10 - rel scale
    Font12 = round(12/1080*RelFont(1,4)); % Font 12 - rel scale
    Font16 = round(16/1080*RelFont(1,4)); % Font 16 - rel scale

%% Feedback Buttons
%   Creates a window which informs the participant which signal is being
%   presented to them. This option works best when a second monitor is used
%   allowing the experimentor to remain outside the listening room.

    StartFeedback = uicontrol('style', 'togglebutton','string', 'Feedback',...
        'units','normalized','Visible','on',...
        'position', [0.05 0.015 0.1 0.05],...
        'FontSize', Font8, 'FontWeight',...
        'bold','callback', {@TurnOn});

    function TurnOn (~,~) % When button pressed the window opens

        if(SignalFeedback)

            SignalFeedback = figure('units', 'normalized', 'Visible', 'on',...
            'Name','Figure Feedback','NumberTitle','Off','toolbar','none','MenuBar','none');

            SignalFrame = uicontrol('Style','frame','units', 'normalized',...
            'pos',[0.1 0.1 0.8 0.8], 'parent',SignalFeedback,'HandleVisibility','off',...
            'BackgroundColor',[0.9 0.9 0.9],'Visible','On');

            SignalLabel = uicontrol('Style','Edit','String','Reference Signal','units', 'normalized',...
            'pos',[0.3 0.4 0.4 0.2], 'parent',SignalFeedback,'HandleVisibility','off',...
            'BackgroundColor',[1 1 1],'Visible','On','FontSize', Font12);

        else

            SignalFeedback = figure('units', 'normalized', 'Visible', 'on',...
            'Name','Figure Feedback','NumberTitle','Off','toolbar','none','MenuBar','none');

        end

    end % EOF


    % Default settings for feedback window
    SignalFrame = uicontrol('Style','frame','units', 'normalized',...
        'pos',[0.1 0.1 0.8 0.8], 'parent',SignalFeedback,'HandleVisibility','off',...
                'BackgroundColor',[0.9 0.9 0.9],'Visible','On');

    SignalLabel = uicontrol('Style','Edit','String','Reference Signal','units', 'normalized',...
        'pos',[0.3 0.4 0.4 0.2], 'parent',SignalFeedback,'HandleVisibility','off',...
                    'BackgroundColor',[1 1 1],'Visible','On','FontSize', Font12);



%% SETTINGS AND CALIBRATION PUSHBUTTONS

% PUSHBUTTON LOCATION - Calibration and Settings
    CSPH = 0.52;   CSPV = 0.155;   CSW = 0.1;   CSH = 0.05;

%Creating Settings Button
    PTSettings1 = uicontrol('style', 'pushbutton','string', 'Settings',...
        'units','normalized','Visible','on',...
        'position', [CSPH CSPV+0.225 CSW CSH],...
        'FontSize', Font8,'callback', {@Settings_Callback}); %#ok<NASGU>

    NSSettings1 = uicontrol('style', 'pushbutton','string', 'Settings',...
        'units','normalized','Visible','off',...
        'position', [CSPH CSPV+0.21 CSW CSH],...
        'FontSize', Font8,'callback', {@NSSettings_Callback}); %#ok<NASGU>

    % WAVSettings1 = uicontrol('style', 'pushbutton','string', 'Settings','units',...
    %     'normalized','Visible','off','position', [CSPH CSPV+0.235 CSW CSH],...
    %     'FontSize', Font8,'callback', {@WAVSettings_Callback}); %#ok<NASGU>

%Creating Settings Button
    PTSettings2 = uicontrol('style', 'pushbutton','string', 'Settings','units',...
        'normalized','Visible','on','position', [CSPH-0.01 CSPV-0.04 CSW CSH],...
        'FontSize', Font8,'callback', {@Settings_Callback}); %#ok<NASGU>

    NSSettings2 = uicontrol('style', 'pushbutton','string', 'Settings','units',...
        'normalized','Visible','off','position', [CSPH-0.01 CSPV-0.015 CSW CSH],...
        'FontSize', Font8,'callback', {@NSSettings_Callback}); %#ok<NASGU>

    % WAVSettings2 = uicontrol('style', 'pushbutton','string', 'Settings','units',...
    %     'normalized','Visible','off','position', [CSPH-0.01 CSPV+0.01 CSW CSH],...
    %     'FontSize', Font8,'callback', {@WAVSettings_Callback}); %#ok<NASGU>
```

```
    % Window Called When Pure Tone Settings Button Pressed
        function Settings_Callback(~,~)
            ExpUserSettings
        end


    % Window Called When Noise Settings Button Pressed
        function NSSettings_Callback(~,~)
            ExpNoiseSettings
        end

    % Window Called When WAV Settings Button Pressed
    %     function WAVSettings_Callback(~,~)
    %
    %     end

%% SPL Reference Selection dB(Z) or db(SL)
    Ref1= uibuttongroup('Title','','Visible','on',...
        'FontSize', Font8,'Position',[0.86 0.385 0.1 0.1]);
    % Linear Reference, dB(Z)
    R1 = uicontrol('Style','Radio','String','dB(Z)',...
        'units', 'normalized','pos',[0.05 0.5 0.9 0.5],...
        'FontSize', Font8,'parent',Ref1,...
        'HandleVisibility','off','Value',1);
    % Sensation Level Reference, dB(SL)
    R2 = uicontrol('Style','Radio','String','dB(SL)',...
        'FontSize', Font8,...
        'units', 'normalized','pos',[0.05 0.05 0.9 0.5],'parent',Ref1,...
        'HandleVisibility','off', 'value',1);

    set(R1,'Callback',{@Greyout})
    set(R2,'Callback',{@Greyout})


%% Signal One Button Group - REFERENCE
    %[Left Bottom Width Height]

    BGH =0.2; %Button Group Height (For Text Height)
    bh=0.20; %Setting Spacing Value
    c=(1/5)*(1-4*bh); %Bottom Location of Text

    TonesOnOff='on';
    NoiseOnOff='off';
    WAVOnOff='off';

    Sig1 = uibuttongroup('Title','Signal 1 - REFERENCE','Visible','on','Position',[0.05 0.3 0.6 BGH]);
        set(Sig1,'SelectionChangeFcn',@ColourUiGroup1);
        set(Sig1,'SelectedObject',[]);
        set(Sig1,'Visible','on');

    %% Pure Tones
    u00 = uicontrol('Style','Radio','String','Pure Tone',...
        'units', 'normalized','pos',[0.05 4*c+3*bh 0.2 bh],'parent',Sig1,...
        'HandleVisibility','off','Value',1);

    %Frequency
        FShiftH=0.2;
        FShiftV=-0.1;
        u00FreqText = uicontrol('Style','text','String','Frequency:',...
            'units', 'normalized','pos',[0.20+FShiftH-0.02 4*c+3*bh-0.025+FShiftV-0.025 0.15 bh],'parent',Sig1,...
            'HandleVisibility','off','Visible',TonesOnOff);
        u00Freq = uicontrol('Style','popupmenu','String',Freq,...
            'units', 'normalized','pos',[0.35+FShiftH 4*c+3*bh+FShiftV 0.15 bh],'parent',Sig1,...
            'HandleVisibility','off','BackgroundColor',[1 1 1],'Visible',TonesOnOff);
        u00FreqUnit = uicontrol('Style','text','String','Hz',...
            'units', 'normalized','pos',[0.5+FShiftH 4*c+3*bh-0.025+FShiftV-0.025 0.05 bh],'parent',Sig1,...
            'HandleVisibility','off','Visible',TonesOnOff);

    %Pure Tone Amplitude
        AShiftH=-0.125;
        AShiftV=-0.45;
        u00AmpLabel = uicontrol('Style','text','String','SPL:',...
            'units', 'normalized','pos',[0.575+AShiftH 4*c+3*bh-0.025+AShiftV 0.1 bh],'parent',Sig1,...
            'HandleVisibility','off','Visible',TonesOnOff);
        u00Amp = uicontrol('Style','Edit','String',0.0,...
            'units', 'normalized','pos',[0.675+AShiftH 4*c+3*bh+AShiftV 0.15 bh],'parent',Sig1,...
            'HandleVisibility','off','BackgroundColor',[1 1 1],'Visible',TonesOnOff);
        u00AmpText = uicontrol('Style','text','String','dB',...
            'units', 'normalized','pos',[0.825+AShiftH 4*c+3*bh-0.025+AShiftV 0.05 bh],'parent',Sig1,...
            'HandleVisibility','off','Visible',TonesOnOff);

           set(u00Freq,'Callback',{@quickLock});

            function quickLock(~,~)
                check_lock = get(u10LockFreq,'Value');

                if check_lock==1
                    Val1 = get(u00Freq,'Value');
                    set(u10Freq,'Value',Val1);
                    set(u10Freq,'BackgroundColor',[0.9 0.9 0.9]);
                    else
                        set(u10Freq,'BackgroundColor',[1 1 1]);
                end
            end
   %% Noise Signals
    u0 = uicontrol('Style','Radio','String','Noise Sample',...
        'units', 'normalized','pos',[0.05 3*c+2*bh 0.25 bh],'parent',Sig1,...
        'HandleVisibility','off');

    %Noise Type Drop-Down Box
        NShift_H=0.0;
        NShift_V=0.0;
        u0NoiseText = uicontrol('Style','text','String','Noise Colour:',...
            'units', 'normalized','pos',[0.3+NShift_H 4*c+3*bh+NShift_V 0.20 bh],'parent',Sig1,...
            'HandleVisibility','off','Visible',NoiseOnOff);

        u0Type = uicontrol('Style','popupmenu','String',{'Violet (+6 dB/oct.)','Blue (+3 dB/oct.)','White (+0 dB/oct.)','Pink (-3
dB/oct.)','Brownian (-6 dB/oct.)', 'Grey (Inv. A-Weight)'},...
            'units', 'normalized','pos',[0.5+NShift_H 4*c+3*bh+NShift_V 0.30 bh],'parent',Sig1,...
            'HandleVisibility','off','Value',3,'Visible',NoiseOnOff);

    %Noise Amplitude
        AShift_H=0.1;
        AShift_V=-0.4;
```

```
    u0AmpText = uicontrol('Style','text','String','SPL:',...
        'units', 'normalized','pos',[0.575+AShift_H 3*c+2*bh-0.025+AShift_V 0.1 bh],'parent',Sig1,...
        'HandleVisibility','off','Visible',NoiseOnOff);
    u0Amp = uicontrol('Style','Edit','String',0.0,...
        'units', 'normalized','pos',[0.675+AShift_H 3*c+2*bh+AShift_V 0.15 bh],'parent',Sig1,...
        'HandleVisibility','off','Visible',NoiseOnOff);
    u0AmpUnit = uicontrol('Style','text','String','dB',...
        'units', 'normalized','pos',[0.825+AShift_H 3*c+2*bh-0.025+AShift_V 0.05 bh],'parent',Sig1,...
        'HandleVisibility','off','Visible',NoiseOnOff);

% No filter available for 20 kHz
FreqN = [20, 25, 31.5, 40, 50, 63, 80, 100, 125, 160, 200, 250, 315, 400,...
        500, 630, 800, 1000, 1250, 1600, 2000, 2500, 3150, 4000, 5000,...
        6300, 8000, 10000, 12500, 16000];

%Lower Frequency
    LFShift_H=0.1;
    LFShift_V=-0.1;
    u0LowFreqText = uicontrol('Style','text','String','Lower Freq:',...
        'units', 'normalized','pos',[0.15+LFShift_H+0.02 3*c+2*bh-0.025+LFShift_V 0.18 bh],'parent',Sig1,...
        'HandleVisibility','off','Visible',NoiseOnOff);
    u0LowFreq = uicontrol('Style','popupmenu','String',FreqN,...
        'units', 'normalized','pos',[0.35+LFShift_H 3*c+2*bh+LFShift_V 0.15 bh],'parent',Sig1,...
        'HandleVisibility','off','Value',1,'Visible',NoiseOnOff);
    u0LowFreqUnit = uicontrol('Style','text','String','Hz',...
        'units', 'normalized','pos',[0.5+LFShift_H 3*c+2*bh-0.025+LFShift_V 0.05 bh],'parent',Sig1,...
        'HandleVisibility','off','Visible',NoiseOnOff);

%Upper Frequency
    UFShift_H=0.1;
    UFShift_V=-0.1;
    u0UpFreqText = uicontrol('Style','text','String','Upper Freq:',...
        'units', 'normalized','pos',[0.15+UFShift_H+0.02 2*c+1*bh-0.025-0.025+UFShift_V 0.18 bh],'parent',Sig1,...
        'HandleVisibility','off','Visible',NoiseOnOff);
    u0UpFreq = uicontrol('Style','popupmenu','String',FreqN,...
        'units', 'normalized','pos',[0.35+UFShift_H 2*c+1*bh-0.025+UFShift_V 0.15 bh],'parent',Sig1,...
        'HandleVisibility','off','Value',29,'Visible',NoiseOnOff);
    u0UpFreqUnit = uicontrol('Style','text','String','Hz',...
        'units', 'normalized','pos',[0.50+UFShift_H 2*c+1*bh-0.025-0.025+UFShift_V 0.05 bh],'parent',Sig1,...
        'HandleVisibility','off','Visible',NoiseOnOff);

      % Preventing the lower limit from being higher than the upper
      % limit and visa versa.

        set(u0LowFreq,'Callback',{@quickCorrLow});
        set(u0UpFreq,'Callback',{@quickCorrHigh});

        function quickCorrLow(~,~)
            check_f1 = get(u0LowFreq,'Value');
            check_f2 = get(u0UpFreq,'Value');

            if check_f2<check_f1;
                check_f1=check_f2;
                set(u0LowFreq,'Value',check_f1);
            end
        end

        function quickCorrHigh(~,~)
            check_f1 = get(u0LowFreq,'Value');
            check_f2 = get(u0UpFreq,'Value');

            if check_f2<check_f1;
                check_f2=check_f1;
                set(u0UpFreq,'Value',check_f2);
            end
        end
%% WAV Files
 u1 = uicontrol('Style','Radio','String','*.WAV File',...
     'units', 'normalized','pos',[0.05 2*c+1*bh 0.2 bh],...
     'parent',Sig1,'HandleVisibility',WAVOnOff);

        WAV1_Name = uicontrol('Style','Edit','units', 'normalized',...
        'pos',[0.3 0.75 0.4 0.2], 'parent',Sig1,'HandleVisibility','off',...
        'BackgroundColor',[1 1 1],'Visible','Off','FontSize', Font12);
         WAV1_Loc = uicontrol('Style','Edit','units', 'normalized',...
        'pos',[0.3 0.25 0.4 0.2], 'parent',Sig1,'HandleVisibility','off',...
        'BackgroundColor',[1 1 1],'Visible','Off','FontSize', Font12);

%File Load
    WavShiftH=0.1;
    WavShiftV=0.75;
    u1Name = uicontrol('Style','text','String','No WAV File Loaded',...
        'units', 'normalized','pos',[0.25+WavShiftH 1*c-0.025+WavShiftV 0.30 bh],'parent',Sig1,...
        'HandleVisibility','off','Visible',WAVOnOff);
    u1LoadButton = uicontrol('Style','pushbutton','String','Load',...
        'units', 'normalized','pos',[0.55+WavShiftH 1*c+WavShiftV 0.10 bh],'parent',Sig1,...
        'HandleVisibility','off','callback', {@Load_S1_WAV,MainWindow,u1Name,u1,WAV1_Name,WAV1_Loc},'Visible',WAVOnOff);

%Gain Adjust
    GainShiftH=-0.25;
    GainShiftV=0.4625;
    u0GainText = uicontrol('Style','text','String','Gain:',...
        'units', 'normalized','pos',[0.65+GainShiftH 1*c-0.025+GainShiftV 0.1 bh],'parent',Sig1,...
        'HandleVisibility','off','Visible',WAVOnOff);
    u0Gain = uicontrol('Style','Edit','String',100.0,...
        'units', 'normalized','pos',[0.75+GainShiftH 1*c+GainShiftV 0.15 bh],'parent',Sig1,...
        'HandleVisibility','off','Visible',WAVOnOff);
    u0GainUnit = uicontrol('Style','text','String','%',...
        'units', 'normalized','pos',[0.9+GainShiftH 1*c-0.025+GainShiftV 0.05 bh],'parent',Sig1,...
        'HandleVisibility','off','Visible',WAVOnOff);



%WAV Details
    T1ShiftH=-0.05;
    T1ShiftV=0.0;
    WAV1_Text1 = uicontrol('Style','text','String','Fs:',...
        'units', 'normalized','pos',[0.3+T1ShiftH 0.25+T1ShiftV 0.1 0.2],'parent',Sig1,...
        'HandleVisibility','off','Visible',WAVOnOff, 'FontWeight','bold');
            WAV1_Text1val = uicontrol('Style','text','String','_____',...
               'units', 'normalized','pos',[0.4+T1ShiftH 0.25+T1ShiftV 0.1 0.2],'parent',Sig1,...
               'HandleVisibility','off','Visible',WAVOnOff);
            WAV1_Text1val2 = uicontrol('Style','text','String','Samples per Sec.',...
               'units', 'normalized','pos',[0.5+T1ShiftH 0.25+T1ShiftV 0.225 0.2],'parent',Sig1,...
```

```
                'HandleVisibility','off','Visible',WAVOnOff);

        T2ShiftH=-0.05;
        T2ShiftV=0.0;
        WAV1_Text2 = uicontrol('Style','text','String','nbits:',...
            'units', 'normalized','pos',[0.3+T2ShiftH 0.05+T2ShiftV 0.1 0.2],'parent',Sig1,...
            'HandleVisibility','off','Visible',WAVOnOff, 'FontWeight','bold');
                WAV1_Text2val = uicontrol('Style','text','String','_______',...
                    'units', 'normalized','pos',[0.4+T2ShiftH 0.05+T2ShiftV 0.1 0.2],'parent',Sig1,...
                    'HandleVisibility','off','Visible',WAVOnOff);
                WAV1_Text2val2 = uicontrol('Style','text','String','Bits per Sample',...
                    'units', 'normalized','pos',[0.5+T2ShiftH 0.05+T2ShiftV 0.225 0.2],'parent',Sig1,...
                    'HandleVisibility','off','Visible',WAVOnOff);

        T3ShiftH=0.35;
        T3ShiftV=0.0;
%         WAV1_Text3 = uicontrol('Style','text','String','Fs:',...
%             'units', 'normalized','pos',[0.3+T3ShiftH 0.25+T3ShiftV 0.1 0.2],'parent',Sig1,...
%             'HandleVisibility','off','Visible',WAVOnOff);
                WAV1_Text3val = uicontrol('Style','text','String','______',...
                    'units', 'normalized','pos',[0.4+T3ShiftH 0.25+T3ShiftV 0.1 0.2],'parent',Sig1,...
                    'HandleVisibility','off','Visible',WAVOnOff);
                WAV1_Text3val2 = uicontrol('Style','text','String','Seconds',...
                    'units', 'normalized','pos',[0.5+T3ShiftH 0.25+T3ShiftV 0.125 0.2],'parent',Sig1,...
                    'HandleVisibility','off','Visible',WAVOnOff);

        T4ShiftH=0.35;
        T4ShiftV=0.0;
%         WAV1_Text4 = uicontrol('Style','text','String','nbits:',...
%             'units', 'normalized','pos',[0.3+T4ShiftH 0.05+T4ShiftV 0.1 0.2],'parent',Sig1,...
%             'HandleVisibility','off','Visible',WAVOnOff);
                WAV1_Text4val = uicontrol('Style','text','String','_______',...
                    'units', 'normalized','pos',[0.4+T4ShiftH 0.05+T4ShiftV 0.1 0.2],'parent',Sig1,...
                    'HandleVisibility','off','Visible',WAVOnOff);
                WAV1_Text4val2 = uicontrol('Style','text','String','Channels',...
                    'units', 'normalized','pos',[0.5+T4ShiftH 0.05+T4ShiftV 0.125 0.2],'parent',Sig1,...
                    'HandleVisibility','off','Visible',WAVOnOff);

                 WAV1_EQ = uicontrol('Style','checkbox','String','Equalizer',...
                    'units', 'normalized','pos',[0.8 0.85 0.2 0.2],'parent',Sig1,...
                    'HandleVisibility','off','Visible','off','FontSize', Font8,...
                    'value',1.0);


    %% Colour Changing Function
     function ColourUiGroup1(~,~)
        %fprintf('Color Group Called\n')
        a1=get(u00,'Value');
        b1=get(u0,'Value');
        c1=get(u1,'Value');
        BgWhite=[1 1 1];
        BgGrey=[0.9 0.9 0.9];
        if a1==1
            %fprintf('Tones Selected\n')
            %Turn White
            set(u00Freq,'BackgroundColor',BgWhite)
            set(u00Amp,'BackgroundColor',BgWhite)
            %Turn Grey

            set(u0Type,'BackgroundColor',BgGrey)
            set(u0Amp,'BackgroundColor',BgGrey)
            set(u0LowFreq,'BackgroundColor',BgGrey)
            set(u0UpFreq,'BackgroundColor',BgGrey)
            set(u0Gain,'BackgroundColor',BgGrey)


            %Turn ON
            set(u00FreqText,'Visible','on')
            set(u00Freq,'Visible','on')
            set(u00FreqUnit,'Visible','on')
            set(u00AmpLabel,'Visible','on')
            set(u00Amp,'Visible','on')
            set(u00AmpText,'Visible','on')
            set(PTSettings1,'Visible','on')

            %Turn OFF
            set(WAV1_Text1,'Visible','off')
            set(WAV1_Text2,'Visible','off')
            set(WAV1_Text1val,'Visible','off')
            set(WAV1_Text2val,'Visible','off')
            set(WAV1_Text1val2,'Visible','off')
            set(WAV1_Text2val2,'Visible','off')
            %set(WAV1_Text3,'Visible','off')
            set(WAV1_Text3val,'Visible','off')
            set(WAV1_Text3val2,'Visible','off')
            %set(WAV1_Text4,'Visible','off')
            set(WAV1_Text4val,'Visible','off')
            set(WAV1_Text4val2,'Visible','off')
            set(u0NoiseText,'Visible','off')
            set(u0Type,'Visible','off')
            set(u0AmpText,'Visible','off')
            set(u0Amp,'Visible','off')
            set(u0AmpUnit,'Visible','off')
            set(u0LowFreqText,'Visible','off')
            set(u0LowFreq,'Visible','off')
            set(u0LowFreqUnit,'Visible','off')
            set(u0UpFreqText,'Visible','off')
            set(u0UpFreq,'Visible','off')
            set(u0UpFreqUnit,'Visible','off')
            set(u1Name,'Visible','off')
            set(u1LoadButton,'Visible','off')
            set(u0GainText,'Visible','off')
            set(u0Gain,'Visible','off')
            set(u0GainUnit,'Visible','off')
            set(NSSettings1,'Visible','off')
%             set(WAVSettings1,'Visible','off')
            set(WAV1_EQ,'Visible','off')

        end

        if b1==1
             %fprintf('Noise Selected\n')
            %Turn White
            set(u0Type,'BackgroundColor',BgWhite)
```

```
    set(u0Amp,'BackgroundColor',BgWhite)
    set(u0LowFreq,'BackgroundColor',BgWhite)
    set(u0UpFreq,'BackgroundColor',BgWhite)

    %Turn Grey
    set(u00Freq,'BackgroundColor',BgGrey)
    set(u00Amp,'BackgroundColor',BgGrey)
    set(u0Gain,'BackgroundColor',BgGrey)


    %Turn ON
    set(u0NoiseText,'Visible','on')
    set(u0Type,'Visible','on')
    set(u0AmpText,'Visible','on')
    set(u0Amp,'Visible','on')
    set(u0AmpUnit,'Visible','on')
    set(u0LowFreqText,'Visible','on')
    set(u0LowFreq,'Visible','on')
    set(u0LowFreqUnit,'Visible','on')
    set(u0UpFreqText,'Visible','on')
    set(u0UpFreq,'Visible','on')
    set(u0UpFreqUnit,'Visible','on')
    set(NSSettings1,'Visible','on')

    %Turn OFF
    set(WAV1_Text1,'Visible','off')
    set(WAV1_Text2,'Visible','off')
    set(WAV1_Text1val,'Visible','off')
    set(WAV1_Text2val,'Visible','off')
    set(WAV1_Text1val2,'Visible','off')
    set(WAV1_Text2val2,'Visible','off')
    %set(WAV1_Text3,'Visible','off')
    set(WAV1_Text3val,'Visible','off')
    set(WAV1_Text3val2,'Visible','off')
    %set(WAV1_Text4,'Visible','off')
    set(WAV1_Text4val,'Visible','off')
    set(WAV1_Text4val2,'Visible','off')
    set(u1Name,'Visible','off')
    set(u1LoadButton,'Visible','off')
    set(u0GainText,'Visible','off')
    set(u0Gain,'Visible','off')
    set(u0GainUnit,'Visible','off')
    set(u00FreqText,'Visible','off')
    set(u00Freq,'Visible','off')
    set(u00FreqUnit,'Visible','off')
    set(u00AmpLabel,'Visible','off')
    set(u00Amp,'Visible','off')
    set(u00AmpText,'Visible','off')
    set(PTSettings1,'Visible','off')
%       set(WAVSettings1,'Visible','off')
    set(WAV1_EQ,'Visible','off')

end

if c1==1
     %fprintf('WAV Selected\n')
    %Turn White
    set(u0Gain,'BackgroundColor',BgWhite)
    %Turn Grey

    set(u00Freq,'BackgroundColor',BgGrey)
    set(u00Amp,'BackgroundColor',BgGrey)
    set(u0Type,'BackgroundColor',BgGrey)
    set(u0Amp,'BackgroundColor',BgGrey)
    set(u0LowFreq,'BackgroundColor',BgGrey)
    set(u0UpFreq,'BackgroundColor',BgGrey)


    %Turn White
    set(WAV1_Text1,'Visible','on')
    set(WAV1_Text2,'Visible','on')
    set(WAV1_Text1val,'Visible','on')
    set(WAV1_Text2val,'Visible','on')
    set(WAV1_Text1val2,'Visible','on')
    set(WAV1_Text2val2,'Visible','on')
    %set(WAV1_Text3,'Visible','on')
    set(WAV1_Text3val,'Visible','on')
    set(WAV1_Text3val2,'Visible','on')
    %set(WAV1_Text4,'Visible','on')
    set(WAV1_Text4val,'Visible','on')
    set(WAV1_Text4val2,'Visible','on')
    set(u1Name,'Visible','on')
    set(u1LoadButton,'Visible','on')
    set(u0GainText,'Visible','on')
    set(u0Gain,'Visible','on')
    set(u0GainUnit,'Visible','on')
%       set(WAVSettings1,'Visible','on')
    set(WAV1_EQ,'Visible','on')

    %Turn Grey
    set(u0NoiseText,'Visible','off')
    set(u00FreqText,'Visible','off')
    set(u00Freq,'Visible','off')
    set(u00FreqUnit,'Visible','off')
    set(u00AmpLabel,'Visible','off')
    set(u00Amp,'Visible','off')
    set(u00AmpText,'Visible','off')
    set(u0Type,'Visible','off')
    set(u0AmpText,'Visible','off')
    set(u0Amp,'Visible','off')
    set(u0AmpUnit,'Visible','off')
    set(u0LowFreqText,'Visible','off')
    set(u0LowFreq,'Visible','off')
    set(u0LowFreqUnit,'Visible','off')
    set(u0UpFreqText,'Visible','off')
    set(u0UpFreq,'Visible','off')
    set(u0UpFreqUnit,'Visible','off')
    set(PTSettings1,'Visible','off')
    set(NSSettings1,'Visible','off')
end

if get(u1,'value')==1
    set(WAV1_LnR,'Visible','on')
    %set(WAV1_LnR,'pos',[0.15 0.775 0.9 bh/1.5])
```

```
            set(u1L,'pos',[0.05 3*c+2*bh+0.05 0.9 bh])
            set(u1R,'pos',[0.05 2*c+1*bh-0.05 0.9 bh])
            set(u1B,'pos',[0.05 1*c 0.9 bh])
        else
            set(WAV1_LnR,'Visible','off')
            set(u1L,'pos',[0.05 3*c+2*bh 0.9 bh])
            set(u1R,'pos',[0.05 2*c+1*bh 0.9 bh])
            set(u1B,'pos',[0.05 1*c 0.9 bh])
        end

    end
%% Signal One Presentation Method
    %[Left Bottom Width Height]

    BGH =0.2; %Button Group Height (For Text Height)
    bh=0.20; %Setting Spacing Value
    c=(1/4)*(1-3*bh); %Bottom Location of Text

    Sig1Pres = uibuttongroup('Title','Signal 1 Playback','Visible','on','Position',[0.66 0.3 0.175 BGH]);

    % Monaural Left
    u1L = uicontrol('Style','Radio','String','Monaural - Left',...
        'units', 'normalized','pos',[0.05 3*c+2*bh 0.9 bh],'parent',Sig1Pres,...
        'HandleVisibility','off','Value',1);

    % Monaural Right
    u1R = uicontrol('Style','Radio','String','Monaural - Right',...
        'units', 'normalized','pos',[0.05 2*c+1*bh 0.9 bh],'parent',Sig1Pres,...
        'HandleVisibility','off');

    % Binaural Presentation
    u1B = uicontrol('Style','Radio','String','Binaural',...
        'units', 'normalized','pos',[0.05 1*c 0.9 bh],...
        'parent',Sig1Pres,'HandleVisibility','on');

                  WAV1_LnR = uicontrol('Style','checkbox','String','Average L+R',...
                     'units', 'normalized','pos',[0.2 0.575 0.8 bh/1.25],'parent',Sig1Pres,...
                     'HandleVisibility','off','Visible','off','FontSize', Font8);



%% Signal Two Button Group - TARGET
    %[Left Bottom Width Height]

    BGH =0.2; %Button Group Height (For Text Height)
    bh=0.20; %Setting Spacing Value
    c=(1/5)*(1-4*bh); %Bottom Location of Text

    Sig2 = uibuttongroup('Title','Signal 2 - TARGET','Visible','on','Position',[0.05 0.075 0.6 BGH]);
        set(Sig2,'SelectionChangeFcn',@ColourUiGroup2);
        set(Sig2,'SelectedObject',[]);
        set(Sig2,'Visible','on');

    %% Pure Tones
    u10 = uicontrol('Style','Radio','String','Pure Tone',...
        'units', 'normalized','pos',[0.05 4*c+3*bh 0.2 bh],'parent',Sig2,...
        'HandleVisibility','off','Value',1);

    %Frequency
        FShiftH=0.1;
        FShiftV=-0.1;
        u10FreqText = uicontrol('Style','text','String','Freq:',...
            'units', 'normalized','pos',[0.20+FShiftH-0.02 4*c+3*bh-0.025+FShiftV-0.025 0.15 bh],'parent',Sig2,...
            'HandleVisibility','off','Visible',TonesOnOff);
        u10Freq = uicontrol('Style','popupmenu','String',Freq,...
            'units', 'normalized','pos',[0.35+FShiftH 4*c+3*bh+FShiftV 0.15 bh],'parent',Sig2,...
            'HandleVisibility','off','BackgroundColor',[1 1 1],'Visible',TonesOnOff);
        u10FreqUnit = uicontrol('Style','text','String','Hz',...
            'units', 'normalized','pos',[0.5+FShiftH 4*c+3*bh-0.025+FShiftV-0.025 0.05 bh],'parent',Sig2,...
            'HandleVisibility','off','Visible',TonesOnOff);

    %Amplitude
        AShiftH=-0.225;
        AShiftV=-0.45;
        u10AmpLabel = uicontrol('Style','text','String','SPL:',...
            'units', 'normalized','pos',[0.575+AShiftH 4*c+3*bh-0.025+AShiftV 0.1 bh],'parent',Sig2,...
            'HandleVisibility','off','Visible',TonesOnOff);
        u10Amp = uicontrol('Style','Edit','String',0.0,...
            'units', 'normalized','pos',[0.675+AShiftH 4*c+3*bh+AShiftV 0.15 bh],'parent',Sig2,...
            'HandleVisibility','off','BackgroundColor',[1 1 1],'Visible',TonesOnOff);
        u10AmpText = uicontrol('Style','text','String','dB',...
            'units', 'normalized','pos',[0.825+AShiftH 4*c+3*bh-0.025+AShiftV 0.05 bh],'parent',Sig2,...
            'HandleVisibility','off','Visible',TonesOnOff);

     %Lock Frequency
             % Locks Frequency between Signal 1 and Signal 2
             u10LockFreq = uicontrol('Style','checkbox','String','Lock Frequency',...
            'units', 'normalized','pos',[0.575+AShiftH+0.375 4*c+3*bh-0.025+AShiftV+0.525 0.25 bh],'parent',Sig2,...
            'HandleVisibility','off','Visible',TonesOnOff);

                set(u10Freq,'Callback',{@quickLock3});
                set(u10LockFreq,'Callback',{@quickLock2});

                function quickLock2(~,~)
                    check_lock = get(u10LockFreq,'Value');

                    if check_lock==1
                        Val1 = get(u00Freq,'Value');
                        set(u10Freq,'Value',Val1);
                        set(u10Freq,'BackgroundColor',[0.9 0.9 0.9]);
                    else
                        set(u10Freq,'BackgroundColor',[1 1 1]);
                    end
                end

                function quickLock3(~,~)
                    check_lock = get(u10LockFreq,'Value');

                    if check_lock==1
                        Val1 = get(u00Freq,'Value');
                        Val2 = get(u10Freq,'Value');
                        set(u00Freq,'Value',Val2);
                        set(u10Freq,'BackgroundColor',[0.9 0.9 0.9]);
```

```
                    else
                        set(u10Freq,'BackgroundColor',[1 1 1]);
                    end
                end

      %Randomize Starting SPL For Pure Tones
            plusminus = 5;  % Random value range

            u10Random = uicontrol('Style','checkbox','String','Randomize Start',...
            'units', 'normalized','pos',[0.575+AShiftH+0.375 4*c+3*bh-0.025+AShiftV+0.325 0.25 bh],'parent',Sig2,...
            'HandleVisibility','off','Visible',TonesOnOff);

                set(u10Random,'Callback',{@randomstart});

                function randomstart(~,~)
                    S1_ToneAmp = str2num(get(u00Amp ,'string'));
                    S2_ToneAmp = S1_ToneAmp+randi([-1*plusminus plusminus],1,1);
                    set(u10Amp ,'string',num2str(S2_ToneAmp));
                end

    %% Noise Signals
     u11 = uicontrol('Style','Radio','String','Noise Sample',...
         'units', 'normalized','pos',[0.05 3*c+2*bh 0.25 bh],'parent',Sig2,...
         'HandleVisibility','off');

     %Noise Type Drop-Down Box
         NShift_H=-0.04;
         NShift_V=0.0;
         u11NoiseText = uicontrol('Style','text','String','Noise Colour:',...
             'units', 'normalized','pos',[0.3+NShift_H 4*c+3*bh+NShift_V-0.05 0.20 bh],'parent',Sig2,...
             'HandleVisibility','off','Visible',NoiseOnOff);
         u11Type = uicontrol('Style','popupmenu','String',{'Violet (+6 dB/oct.)','Blue (+3 dB/oct.)','White (+0 dB/oct.)','Pink (-3
dB/oct.)','Brownian (-6 dB/oct.)', 'Grey (Inv. A-Weight)'},...
             'units', 'normalized','pos',[0.5+NShift_H 4*c+3*bh+NShift_V 0.30 bh],'parent',Sig2,...
             'HandleVisibility','off','Value',3,'Visible',NoiseOnOff);

     %Amplitude
         AShift_H=0.1;
         AShift_V=-0.4;
         u11AmpText = uicontrol('Style','text','String','SPL:',...
             'units', 'normalized','pos',[0.575+AShift_H 3*c+2*bh-0.025+AShift_V 0.1 bh],'parent',Sig2,...
             'HandleVisibility','off','Visible',NoiseOnOff);
         u11Amp = uicontrol('Style','Edit','String',0.0,...
             'units', 'normalized','pos',[0.675+AShift_H 3*c+2*bh+AShift_V 0.15 bh],'parent',Sig2,...
             'HandleVisibility','off','Visible',NoiseOnOff);
         u11AmpUnit = uicontrol('Style','text','String','dB',...
             'units', 'normalized','pos',[0.825+AShift_H 3*c+2*bh-0.025+AShift_V 0.05 bh],'parent',Sig2,...
             'HandleVisibility','off','Visible',NoiseOnOff);

     %Lower Frequency
         LFShift_H=0.1;
         LFShift_V=-0.1;
         u11LowFreqText = uicontrol('Style','text','String','Lower Freq:',...
             'units', 'normalized','pos',[0.15+LFShift_H+0.02 3*c+2*bh-0.025+LFShift_V-0.025 0.18 bh],'parent',Sig2,...
             'HandleVisibility','off','Visible',NoiseOnOff);
         u11LowFreq = uicontrol('Style','popupmenu','String',FreqN,...
             'units', 'normalized','pos',[0.35+LFShift_H 3*c+2*bh+LFShift_V 0.15 bh],'parent',Sig2,...
             'HandleVisibility','off','Value',1,'Visible',NoiseOnOff);
         u11LowFreqUnit = uicontrol('Style','text','String','Hz',...
             'units', 'normalized','pos',[0.5+LFShift_H 3*c+2*bh-0.025+LFShift_V-0.025 0.05 bh],'parent',Sig2,...
             'HandleVisibility','off','Visible',NoiseOnOff);

     %Upper Frequency
         UFShift_H=0.1;
         UFShift_V=-0.1;
         u11UpFreqText = uicontrol('Style','text','String','Upper Freq:',...
             'units', 'normalized','pos',[0.15+UFShift_H+0.02 2*c+1*bh-0.025-0.025+UFShift_V-0.025 0.18 bh],'parent',Sig2,...
             'HandleVisibility','off','Visible',NoiseOnOff);
         u11UpFreq = uicontrol('Style','popupmenu','String',FreqN,...
             'units', 'normalized','pos',[0.35+UFShift_H 2*c+1*bh-0.025+UFShift_V 0.15 bh],'parent',Sig2,...
             'HandleVisibility','off','Value',29,'Visible',NoiseOnOff);
         u11UpFreqUnit = uicontrol('Style','text','String','Hz',...
             'units', 'normalized','pos',[0.50+UFShift_H 2*c+1*bh-0.025-0.025+UFShift_V-0.025 0.05 bh],'parent',Sig2,...
             'HandleVisibility','off','Visible',NoiseOnOff);

           % Preventing the lower limit from being higher than the upper
           % limit and visa versa.

             set(u11LowFreq,'Callback',{@quickCorrLow2});
             set(u11UpFreq,'Callback',{@quickCorrHigh2});

             function quickCorrLow2(~,~)
                 check_f1 = get(u11LowFreq,'Value');
                 check_f2 = get(u11UpFreq,'Value');

                 if check_f2<check_f1;
                     check_f1=check_f2;
                     set(u11LowFreq,'Value',check_f1);
                 end
             end

             function quickCorrHigh2(~,~)
                 check_f1 = get(u11LowFreq,'Value');
                 check_f2 = get(u11UpFreq,'Value');

                 if check_f2<check_f1;
                     check_f2=check_f1;
                     set(u11UpFreq,'Value',check_f2);
                 end
             end

%$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
%$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

             u11Lock = uicontrol('Style','checkbox','String','Lock N1 & N2',...
             'units', 'normalized','pos',[0.575+AShiftH+0.425 4*c+6*bh-0.025+AShiftV-0.025 0.22 bh],'parent',Sig2,...
             'HandleVisibility','off','Visible',NoiseOnOff);

                     set(u11Lock,'Callback',{@quickLockn1});

                     set(u0Type,'Callback',{@quickLockn2});
                     set(u0LowFreq,'Callback',{@quickLockn2});
                     set(u0UpFreq,'Callback',{@quickLockn2});
```

```
            function quickLockn1(~,~)
                check_lockn = get(u11Lock,'Value');

                if check_lockn==1
                    Ntype = get(u0Type,'Value');
                    NFreqLow = get(u0LowFreq,'Value');
                    NFreqHigh = get(u0UpFreq,'Value');

                    set(u11Type,'Value',Ntype)
                    set(u11LowFreq,'Value',NFreqLow)
                    set(u11UpFreq,'Value',NFreqHigh)

                    set(u11Type,'BackgroundColor',[0.9 0.9 0.9]);
                    set(u11LowFreq,'BackgroundColor',[0.9 0.9 0.9]);
                    set(u11UpFreq,'BackgroundColor',[0.9 0.9 0.9]);
                else
                    set(u11Type,'BackgroundColor',[1 1 1]);
                    set(u11LowFreq,'BackgroundColor',[1 1 1]);
                    set(u11UpFreq,'BackgroundColor',[1 1 1]);
                end
            end

            function quickLockn2(~,~)
                check_lockn = get(u11Lock,'Value');

                if check_lockn==1
                    Ntype = get(u0Type,'Value');
                    NFreqLow = get(u0LowFreq,'Value');
                    NFreqHigh = get(u0UpFreq,'Value');

                    set(u11Type,'Value',Ntype)
                    set(u11LowFreq,'Value',NFreqLow)
                    set(u11UpFreq,'Value',NFreqHigh)

                    set(u11Type,'BackgroundColor',[0.9 0.9 0.9]);
                    set(u11LowFreq,'BackgroundColor',[0.9 0.9 0.9]);
                    set(u11UpFreq,'BackgroundColor',[0.9 0.9 0.9]);
                else
                    set(u11Type,'BackgroundColor',[1 1 1]);
                    set(u11LowFreq,'BackgroundColor',[1 1 1]);
                    set(u11UpFreq,'BackgroundColor',[1 1 1]);
                end
            end

        plusminusN = 5;  % Random value range

        u11Random = uicontrol('Style','checkbox','String','Random Start',...
        'units', 'normalized','pos',[0.575+AShiftH+0.425 4*c+3*bh-0.025+AShiftV+0.40-0.025 0.22 bh],'parent',Sig2,...
        'HandleVisibility','off','Visible',NoiseOnOff,'Value',1);

            set(u11Random,'Callback',{@randomstartN});

            function randomstartN(~,~)
                S1_NoiseAmp = str2num(get(u0Amp ,'string'));
                S2_NoiseAmp = S1_NoiseAmp+randi([-1*plusminusN plusminusN],1,1);
                set(u11Amp ,'string',num2str(S2_NoiseAmp));
            end


%% WAV Files
 u12 = uicontrol('Style','Radio','String','*.WAV File',...
     'units', 'normalized','pos',[0.05 2*c+1*bh 0.2 bh],...
     'parent',Sig2,'HandleVisibility','on');

            WAV2_Name = uicontrol('Style','Edit','units', 'normalized',...
         'pos',[0.3 0.75 0.4 0.2], 'parent',Sig1,'HandleVisibility','off',...
         'BackgroundColor',[1 1 1],'Visible','Off','FontSize', Font12);
          WAV2_Loc = uicontrol('Style','Edit','units', 'normalized',...
         'pos',[0.3 0.25 0.4 0.2], 'parent',Sig1,'HandleVisibility','off',...
         'BackgroundColor',[1 1 1],'Visible','Off','FontSize', Font12);

 %File Load
     WavShiftH=0.1;
     WavShiftV=0.75;
     u12Name = uicontrol('Style','text','String','No WAV File Loaded',...
         'units', 'normalized','pos',[0.25+WavShiftH 1*c-0.025+WavShiftV 0.30 bh],'parent',Sig2,...
         'HandleVisibility','off','Visible',WAVOnOff);
     u12LoadButton = uicontrol('Style','pushbutton','String','Load',...
         'units', 'normalized','pos',[0.55+WavShiftH 1*c+WavShiftV 0.10 bh],'parent',Sig2,...
         'HandleVisibility','off','Visible',WAVOnOff,'callback', {@Load_S2_WAV,MainWindow,u12Name,u12,WAV2_Name,WAV2_Loc});

 %Gain Adjust
     GainShiftH=-0.25;
     GainShiftV=0.45;
     u12GainText = uicontrol('Style','text','String','Gain:',...
         'units', 'normalized','pos',[0.65+GainShiftH 1*c-0.025+GainShiftV 0.1 bh],'parent',Sig2,...
         'HandleVisibility','off','Visible',WAVOnOff);
     u12Gain = uicontrol('Style','Edit','String',100.0,...
         'units', 'normalized','pos',[0.75+GainShiftH 1*c+GainShiftV 0.15 bh],'parent',Sig2,...
         'HandleVisibility','off','Visible',WAVOnOff);
     u12GainUnit = uicontrol('Style','text','String','%',...
         'units', 'normalized','pos',[0.9+GainShiftH 1*c-0.025+GainShiftV 0.05 bh],'parent',Sig2,...
         'HandleVisibility','off','Visible',WAVOnOff);


 %WAV Details
     T1ShiftH=-0.05;
     T1ShiftV=0.0;
     WAV2_Text1 = uicontrol('Style','text','String','Fs:',...
         'units', 'normalized','pos',[0.3+T1ShiftH 0.25+T1ShiftV 0.1 0.2],'parent',Sig2,...
         'HandleVisibility','off','Visible',WAVOnOff);
              WAV2_Text1val = uicontrol('Style','text','String','_____',...
                 'units', 'normalized','pos',[0.4+T1ShiftH 0.25+T1ShiftV 0.1 0.2],'parent',Sig2,...
                 'HandleVisibility','off','Visible',WAVOnOff);
             WAV2_Text1val2 = uicontrol('Style','text','String','Samples per Sec.',...
                 'units', 'normalized','pos',[0.5+T1ShiftH 0.25+T1ShiftV 0.225 0.2],'parent',Sig2,...
                 'HandleVisibility','off','Visible',WAVOnOff);

     T2ShiftH=-0.05;
     T2ShiftV=0.0;
     WAV2_Text2 = uicontrol('Style','text','String','nbits:',...
         'units', 'normalized','pos',[0.3+T2ShiftH 0.05+T2ShiftV 0.1 0.2],'parent',Sig2,...
         'HandleVisibility','off','Visible',WAVOnOff);
             WAV2_Text2val = uicontrol('Style','text','String','______',...
```

```
                'units', 'normalized','pos',[0.4+T2ShiftH 0.05+T2ShiftV 0.1 0.2],'parent',Sig2,...
                'HandleVisibility','off','Visible',WAVOnOff);
            WAV2_Text2val2 = uicontrol('Style','text','String','Bits per Sample',...
                'units', 'normalized','pos',[0.5+T2ShiftH 0.05+T2ShiftV 0.225 0.2],'parent',Sig2,...
                'HandleVisibility','off','Visible',WAVOnOff);

        T3ShiftH=0.35;
        T3ShiftV=0.0;
             WAV2_Text3val = uicontrol('Style','text','String','______',...
                'units', 'normalized','pos',[0.4+T3ShiftH 0.25+T3ShiftV 0.1 0.2],'parent',Sig2,...
                'HandleVisibility','off','Visible',WAVOnOff);
             WAV2_Text3val2 = uicontrol('Style','text','String','Seconds',...
                'units', 'normalized','pos',[0.5+T3ShiftH 0.25+T3ShiftV 0.125 0.2],'parent',Sig2,...
                'HandleVisibility','off','Visible',WAVOnOff);

        T4ShiftH=0.35;
        T4ShiftV=0.0;
            WAV2_Text4val = uicontrol('Style','text','String','_______',...
                'units', 'normalized','pos',[0.4+T4ShiftH 0.05+T4ShiftV 0.1 0.2],'parent',Sig2,...
                'HandleVisibility','off','Visible',WAVOnOff);
            WAV2_Text4val2 = uicontrol('Style','text','String','Channels',...
                'units', 'normalized','pos',[0.5+T4ShiftH 0.05+T4ShiftV 0.125 0.2],'parent',Sig2,...
                'HandleVisibility','off','Visible',WAVOnOff);


                 WAV2_EQ = uicontrol('Style','checkbox','String','Equalizer',...
                    'units', 'normalized','pos',[0.8 0.85 0.2 0.2],'parent',Sig2,...
                    'HandleVisibility','off','Visible','off','FontSize', Font8,...
                    'value',1.0);


                 WAV2_Lock = uicontrol('Style','checkbox','String','Lock WAV',...
                    'units', 'normalized','pos',[0.8 0.65 0.2 0.2],'parent',Sig2,...
                    'HandleVisibility','off','Visible',WAVOnOff,'FontSize', Font8,...
                    'value',1.0);


                WavLockCheck = get (WAV2_Lock,'Value');


set(u1LoadButton,'callback',{@Load_S1_WAV,MainWindow,u1Name,u1,WAV1_Name,WAV1_Loc,u12Name,u12,WAV2_Name,WAV2_Loc,WAV2_Lock})

set(u12LoadButton,'callback',{@Load_S2_WAV,MainWindow,u12Name,u12,WAV2_Name,WAV2_Loc,u1Name,u1,WAV1_Name,WAV1_Loc,WAV2_Lock})

                 RdmWAV = uicontrol('Style','checkbox','String','Randomize',...
                    'units', 'normalized','pos',[0.8 0.45 0.2 0.2],'parent',Sig2,...
                    'HandleVisibility','off','Visible',WAVOnOff,'FontSize', Font8,...
                    'value',1.0);

            % Setting Default Values
            S2y = 0;
            S2Fs = 0;
            S2nbit = 0;
            S1y = 0;
            S1Fs = 0;
            S1nbit = 0;

%$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
%$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
        plusminusWAV = 15;  % Random value range


            set(RdmWAV,'Callback',{@randomstartW});

            function randomstartW(~,~)
                S1_WAVGain = str2num(get(u0Gain ,'string'));
                S2_WAVAmp = S1_WAVGain+randi([-1*plusminusWAV plusminusWAV],1,1);
                set(u12Gain ,'string',num2str(S2_WAVAmp));
            end



   %% Colour Changing Function
        function ColourUiGroup2(~,~)

            %fprintf('Color Group Called\n')
            a2=get(u10,'Value');
            b2=get(u11,'Value');
            c2=get(u12,'Value');
            BgWhite=[1 1 1];
            BgGrey=[0.9 0.9 0.9];
            if a2==1
                %fprintf('Tones Selected\n')
                %Turn White
                set(u10Freq,'BackgroundColor',BgWhite)
                set(u10Amp,'BackgroundColor',BgWhite)

                %Turn Grey
                set(u11Type,'BackgroundColor',BgGrey)
                set(u11Amp,'BackgroundColor',BgGrey)
                set(u11LowFreq,'BackgroundColor',BgGrey)
                set(u11UpFreq,'BackgroundColor',BgGrey)
                set(u12Gain,'BackgroundColor',BgGrey)

                %Turn ON
                set(u10FreqText,'Visible','on')
                set(u10Freq,'Visible','on')
                set(u10FreqUnit,'Visible','on')
                set(u10AmpLabel,'Visible','on')
                set(u10Amp,'Visible','on')
                set(u10AmpText,'Visible','on')
                set(PTSettings2,'Visible','on')
                set(u10LockFreq,'Visible','on')
                set(u10Random,'Visible','on')
                set(u11Random,'Visible','off')
                set(u11Lock,'Visible','off')

                %Turn OFF
                set(WAV2_Text1,'Visible','off')
                set(WAV2_Text2,'Visible','off')
                set(WAV2_Text1val,'Visible','off')
                set(WAV2_Text2val,'Visible','off')
                set(WAV2_Text1val2,'Visible','off')
```

```
    set(WAV2_Text2val2,'Visible','off')
    set(WAV2_Text3val,'Visible','off')
    set(WAV2_Text3val2,'Visible','off')
    set(WAV2_Text4val,'Visible','off')
    set(WAV2_Text4val2,'Visible','off')
    set(u11NoiseText,'Visible','off')
    set(u11Type,'Visible','off')
    set(u11AmpText,'Visible','off')
    set(u11Amp,'Visible','off')
    set(u11AmpUnit,'Visible','off')
    set(u11LowFreqText,'Visible','off')
    set(u11LowFreq,'Visible','off')
    set(u11LowFreqUnit,'Visible','off')
    set(u11UpFreqText,'Visible','off')
    set(u11UpFreq,'Visible','off')
    set(u11UpFreqUnit,'Visible','off')
    set(u12Name,'Visible','off')
    set(u12LoadButton,'Visible','off')
    set(u12GainText,'Visible','off')
    set(u12Gain,'Visible','off')
    set(u12GainUnit,'Visible','off')
    set(NSSettings2,'Visible','off')
    set(WAV2_EQ,'Visible','off')
    set(WAV2_Lock,'Visible','off')
    set(RdmWAV,'Visible','off')

end

if b2==1
     %fprintf('Noise Selected\n')
    %Turn White
    set(u11Type,'BackgroundColor',BgWhite)
    set(u11Amp,'BackgroundColor',BgWhite)
    set(u11LowFreq,'BackgroundColor',BgWhite)
    set(u11UpFreq,'BackgroundColor',BgWhite)

    %Turn Grey
    set(u10Freq,'BackgroundColor',BgGrey)
    set(u10Amp,'BackgroundColor',BgGrey)
    set(u12Gain,'BackgroundColor',BgGrey)

    %Turn ON
    set(u11NoiseText,'Visible','on')
    set(u11Type,'Visible','on')
    set(u11AmpText,'Visible','on')
    set(u11Amp,'Visible','on')
    set(u11AmpUnit,'Visible','on')
    set(u11LowFreqText,'Visible','on')
    set(u11LowFreq,'Visible','on')
    set(u11LowFreqUnit,'Visible','on')
    set(u11UpFreqText,'Visible','on')
    set(u11UpFreq,'Visible','on')
    set(u11UpFreqUnit,'Visible','on')
    set(NSSettings2,'Visible','on')
    set(u11Random,'Visible','on')
    set(u11Lock,'Visible','on')

    %Turn OFF
    set(WAV2_Text1,'Visible','off')
    set(WAV2_Text2,'Visible','off')
    set(WAV2_Text1val,'Visible','off')
    set(WAV2_Text2val,'Visible','off')
    set(WAV2_Text1val2,'Visible','off')
    set(WAV2_Text2val2,'Visible','off')
    set(WAV2_Text3val,'Visible','off')
    set(WAV2_Text3val2,'Visible','off')
    set(WAV2_Text4val,'Visible','off')
    set(WAV2_Text4val2,'Visible','off')
    set(u12Name,'Visible','off')
    set(u12LoadButton,'Visible','off')
    set(u12GainText,'Visible','off')
    set(u12Gain,'Visible','off')
    set(u12GainUnit,'Visible','off')
    set(u10FreqText,'Visible','off')
    set(u10Freq,'Visible','off')
    set(u10FreqUnit,'Visible','off')
    set(u10AmpLabel,'Visible','off')
    set(u10Amp,'Visible','off')
    set(u10AmpText,'Visible','off')
    set(u10LockFreq,'Visible','off')
    set(u10Random,'Visible','off')
    set(PTSettings2,'Visible','off')
    set(WAV2_EQ,'Visible','off')
    set(WAV2_Lock,'Visible','off')
    set(RdmWAV,'Visible','off')
end

if c2==1
    %Turn White
    set(u12Gain,'BackgroundColor',BgWhite)

    %Turn Grey
    set(u10Freq,'BackgroundColor',BgGrey)
    set(u10Amp,'BackgroundColor',BgGrey)
    set(u11Type,'BackgroundColor',BgGrey)
    set(u11Amp,'BackgroundColor',BgGrey)
    set(u11LowFreq,'BackgroundColor',BgGrey)
    set(u11UpFreq,'BackgroundColor',BgGrey)

    %Turn White
    set(WAV2_Text1,'Visible','on')
    set(WAV2_Text2,'Visible','on')
    set(WAV2_Text1val,'Visible','on')
    set(WAV2_Text2val,'Visible','on')
    set(WAV2_Text1val2,'Visible','on')
    set(WAV2_Text2val2,'Visible','on')
    set(WAV2_Text3val,'Visible','on')
    set(WAV2_Text3val2,'Visible','on')
    set(WAV2_Text4val,'Visible','on')
    set(WAV2_Text4val2,'Visible','on')
    set(u12Name,'Visible','on')
    set(u12LoadButton,'Visible','on')
    set(u12GainText,'Visible','on')
    set(u12Gain,'Visible','on')
```

```
                set(u12GainUnit,'Visible','on')
                set(WAV2_EQ,'Visible','on')
                set(WAV2_Lock,'Visible','on')
                set(RdmWAV,'Visible','on')

                %Turn Off
                set(u11NoiseText,'Visible','off')
                set(u10FreqText,'Visible','off')
                set(u10Freq,'Visible','off')
                set(u10FreqUnit,'Visible','off')
                set(u10AmpLabel,'Visible','off')
                set(u10Amp,'Visible','off')
                set(u10AmpText,'Visible','off')
                set(u11Type,'Visible','off')
                set(u11AmpText,'Visible','off')
                set(u11Amp,'Visible','off')
                set(u11AmpUnit,'Visible','off')
                set(u11LowFreqText,'Visible','off')
                set(u11LowFreq,'Visible','off')
                set(u11LowFreqUnit,'Visible','off')
                set(u11UpFreqText,'Visible','off')
                set(u11UpFreq,'Visible','off')
                set(u11UpFreqUnit,'Visible','off')
                set(u10LockFreq,'Visible','off')
                set(u10Random,'Visible','off')
                set(PTSettings2,'Visible','off')
                set(NSSettings2,'Visible','off')
                set(u11Random,'Visible','off')
                set(u11Lock,'Visible','off')
            end

            if get(u12,'value')==1
                set(WAV2_LnR,'Visible','on')
                set(u2L,'pos',[0.05 3*c+2*bh+0.05 0.9 bh])
                set(u2R,'pos',[0.05 2*c+1*bh-0.05 0.9 bh])
                set(u2B,'pos',[0.05 1*c 0.9 bh])
            else
                set(WAV2_LnR,'Visible','off')
                set(u2L,'pos',[0.05 3*c+2*bh 0.9 bh])
                set(u2R,'pos',[0.05 2*c+1*bh 0.9 bh])
                set(u2B,'pos',[0.05 1*c 0.9 bh])
            end

        end % EOF

    % ---------------------------------------------------------------------

%% Signal Two Presentation Method
    %[Left Bottom Width Height]

    BGH =0.2; %Button Group Height (For Text Height)
    bh=0.20; %Setting Spacing Value
    c=(1/4)*(1-3*bh); %Bottom Location of Text

    Sig2Pres = uibuttongroup('Title','Signal 2 Playback','Visible','on','Position',[0.66 0.075 0.175 BGH]);

    % Monaural Left
    u2L = uicontrol('Style','Radio','String','Monaural - Left',...
        'units', 'normalized','pos',[0.05 3*c+2*bh 0.9 bh],'parent',Sig2Pres,...
        'HandleVisibility','off','Value',1);

    % Monaural Right
    u2R = uicontrol('Style','Radio','String','Monaural - Right',...
        'units', 'normalized','pos',[0.05 2*c+1*bh 0.9 bh],'parent',Sig2Pres,...
        'HandleVisibility','off');

    % Binaural Presentation
    u2B = uicontrol('Style','Radio','String','Binaural',...
        'units', 'normalized','pos',[0.05 1*c 0.9 bh],...
        'parent',Sig2Pres,'HandleVisibility','on');

             WAV2_LnR = uicontrol('Style','checkbox','String','Average L+R',...
                'units', 'normalized','pos',[0.2 0.575 0.8 bh/1.25],'parent',Sig2Pres,...
                'HandleVisibility','off','Visible','off','FontSize', Font8);

%% START/STOP Button Positions
    SSh=0.85;
    SSv=0.075;
    SSw=0.125;
    SSt=0.30;

    PlayCount=0;

    % Defining New Buttons for Saving Data and Loading Data
    StartPlayback = uicontrol('style', 'togglebutton','string', 'START',...
        'units','normalized','Visible','on',...
        'position', [SSh SSv SSw SSt],...
        'FontSize', Font12, 'FontWeight',...
        'bold','callback', {@StartSig,Resultant,Headphones_A,...
        Headphones_B,u00Freq,u00Amp,u0Type, u0Amp, u0LowFreq,...
        u0UpFreq, u1Name, u0Gain, u10Freq,u10Amp, u11Type, u11Amp,...
        u11LowFreq, u11UpFreq, u12Name, u12Gain, PlayCount, R1, R2,...
        WAV1_Name,WAV1_Loc,WAV2_Name,WAV2_Loc,WAV1_LnR,WAV2_LnR,...
        WAV1_EQ,WAV2_EQ,WAV2_Lock,RdmWAV});

    set(StartPlayback,'BackgroundColor',[0 0.8 0])

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% %%%%%%%%      AUDIOGRAM PLOT FUNCTION                          %%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    Plot_Frame1 = uipanel('Title','Audiogram', 'units', 'normalized',...
        'position', [0.05 0.525 0.5 0.45], 'visible','on',...
        'FontSize', Font10,'BackgroundColor',[0.8 0.8 0.8],...
        'Visible','On');

    function Plotting_Function

        % Grey out if dBZ is selected
        ExpAppZeros = getappdata(0,'evalue');
        TestdBZ = ExpAppZeros(29,4);
        AudFunc=ExpAppZeros(30,4);

        if get(A1,'Value') == 1; AudFunc = 1; end
```

```
if get(A2,'Value') == 1; AudFunc = 2; end
if get(A3,'Value') == 1; AudFunc = 3; end
if get(A4,'Value') == 1; AudFunc = 4; end

AVG = (Resultant(:,3)+Resultant(:,5))/2;

for StepAVG = 1:size(AVG)
    if Resultant(StepAVG,3) == -20
        AVG(StepAVG,1)=-20;
    end

    if Resultant(StepAVG,5) == -20
        AVG(StepAVG,1)=-20;
    end
end

Picture1=subplot(1,1,1,'Parent',Plot_Frame1);
PlotAxes = semilogx(ReferenceContour(:,1), ReferenceContour(:,2), '--k',... % (1) ANSI Reference Contour
                    AmbientSPL(:,1), AmbientSPL(:,2),':b',...               % (2) Ambient - Left Signal
                    AmbientSPL(:,1), AmbientSPL(:,3),':r',...               % (3) Ambient - Right Signal
                    Resultant(:,1), Resultant(:,2),'ob',...                 % (4) Large Left
                    Resultant(:,1), Resultant(:,3),'ob',...                 % (5) Small Left
                    Resultant(:,1), Resultant(:,4),'xr',...                 % (6) Large Right
                    Resultant(:,1), Resultant(:,5),'xr',...                 % (7) Small Right
                    Resultant(:,1), Resultant(:,6),'xb',...                 % (8) Did Not Hear - Left
                    Resultant(:,1), Resultant(:,7),'xr',...                 % (9) Did Not Hear - Right
                    MAX_SPL_Lim(:,1),MAX_SPL_Lim(:,2),'-r',...              % (10) LIMIT Line
                    Resultant(:,1), AVG(:,1),'dm',...                % (9) AVERAGE
    'DisplayName', 'SemiLog Plot'); %, 'Parent', handles.graph

if AudFunc == 1; % L + R
    set(PlotAxes(4),'MarkerSize',1,'LineWidth', 0.5) % Large Left (Hide)
    set(PlotAxes(5),'MarkerSize',5,'LineWidth', 0.5,'MarkerFaceColor','b') % Small Left (Hide)
    set(PlotAxes(6),'MarkerSize',1,'LineWidth', 0.5) % Large Right (Hide)
    set(PlotAxes(7),'MarkerSize',5,'LineWidth', 2,'MarkerFaceColor','r') % Small Right (Hide)
    set(PlotAxes(11),'MarkerSize',1,'LineWidth', 0.5,'MarkerFaceColor','k') % Small Right (Hide)

end

if AudFunc == 2;  % L
    set(PlotAxes(4),'MarkerSize',1,'LineWidth', 0.5) % Large Left (Hide)
    set(PlotAxes(5),'MarkerSize',5,'LineWidth', 0.5,'MarkerFaceColor','b') % Small Left (Hide)
    set(PlotAxes(6),'MarkerSize',1,'LineWidth', 0.5) % Large Right (Hide)
    set(PlotAxes(7),'MarkerSize',1,'LineWidth', 0.5,'MarkerFaceColor','k') % Small Right (Hide)
    set(PlotAxes(11),'MarkerSize',1,'LineWidth', 0.5,'MarkerFaceColor','k') % Small Right (Hide)
end

if AudFunc == 3;  % R
    set(PlotAxes(4),'MarkerSize',1,'LineWidth', 0.5) % Large Left (Hide)
    set(PlotAxes(5),'MarkerSize',1,'LineWidth', 0.5,'MarkerFaceColor','k') % Small Left (Hide)
    set(PlotAxes(6),'MarkerSize',1,'LineWidth', 0.5) % Large Right (Hide)
    set(PlotAxes(7),'MarkerSize',5,'LineWidth', 2,'MarkerFaceColor','r') % Small Right (Hide)
    set(PlotAxes(11),'MarkerSize',1,'LineWidth', 0.5,'MarkerFaceColor','k') % Small Right (Hide)
end

if AudFunc == 4;  % AVG(L,R)
    set(PlotAxes(4),'MarkerSize',1,'LineWidth', 0.5) % Large Left (Hide)
    set(PlotAxes(5),'MarkerSize',1,'LineWidth', 0.5,'MarkerFaceColor','k') % Small Left (Hide)
    set(PlotAxes(6),'MarkerSize',1,'LineWidth', 0.5) % Large Right (Hide)
    set(PlotAxes(7),'MarkerSize',1,'LineWidth', 0.5,'MarkerFaceColor','k') % Small Right (Hide)
    set(PlotAxes(11),'MarkerSize',5,'LineWidth', 2,'MarkerFaceColor','m') % Small Right (Hide)
end

ax_PlotAxes = gca; %Collect info on current axes values to gca
curtick = get(gca, 'XTick'); % Collecting current x-labels
set(ax_PlotAxes, 'units', 'normalized',...
    'OuterPosition', [0 0.2 1 0.75],...
    'Position', [0.1 0.25 0.85 0.705],...
    'XLim', [10 20000],...
    'YLim', [-10 100],...
    'XTickLabel', cellstr(num2str(curtick(:)))); % Setting the x-labels to non-scientific
grid


% When dB(Z) is selected, GREY-OUT the background
if TestdBZ == 1
    set(gca,'Color',[0.8 0.8 0.8]);
    set(PlotAxes(4),'MarkerSize',1,'LineWidth', 0.5) % Large Left (Hide)
    set(PlotAxes(5),'MarkerSize',1,'MarkerFaceColor','b')
    set(PlotAxes(6),'MarkerSize',2,'LineWidth', 0.5) % Large Right (Hide)
    set(PlotAxes(7),'MarkerSize',2,'MarkerFaceColor','r')
    set(PlotAxes(7),'MarkerSize',2,'LineWidth', 0.5)   % Magnifying Small Right
    set(PlotAxes(11),'MarkerSize',2,'LineWidth', 0.5)   % Magnifying Small Right
end


% When dB(SL) is selected, Highlight the background
if TestdBZ == 0
    set(gca,'Color',[1 1 1]);
end

AuidiogramFile = uicontrol('Style','text',...
    'units', 'normalized','pos',[0.05 0.05 0.7 0.075],'parent',Plot_Frame1,...
    'HandleVisibility','off','BackgroundColor',[0.8 0.8 0.8]);

if AudFileName==0
    set(AuidiogramFile,'String','No Audiogram Input...')
else
    set(AuidiogramFile,'String',AudFileName)
end

AuidiogramLoad = uicontrol('Style','pushbutton','String','Load',...
    'units', 'normalized','pos',[0.775 0.025 0.2 0.125],'parent',Plot_Frame1,...
    'HandleVisibility','off',...
    'FontSize', Font8, 'FontWeight',...
    'bold','callback', {@Load_Audiogram,MainWindow});

        ExpApp= getappdata(0,'evalue');

AuidiogramZoom = uicontrol('Style','pushbutton','String','Zoom',...
    'units', 'normalized','pos',[0.6 0.025 0.15 0.125],'parent',Plot_Frame1,...
    'HandleVisibility','off',...
    'FontSize', Font8, 'FontWeight',...
    'bold','callback', {@Zoom_Audiogram,ExpApp});
```

```
    end % EOF

    % ---------------------------------------------------------------------

        % SPL Reference Selection dB(Z) or db(SL)
        AudFnc= uibuttongroup('Title','','Visible','on',...
            'FontSize', Font6,'Position',[0.45 0.955 0.54 0.07],'parent',Plot_Frame1);

        % Linear Reference, dB(Z)
            A1 = uicontrol('Style','Radio','String','L + R',...
                'units', 'normalized','pos',[0.0 0.0 0.25 1],...
                'FontSize', Font6,'parent',AudFnc,...
                'HandleVisibility','off','Value',1,'BackgroundColor',[0.8 0.8 0.8]);
        % Linear Reference, dB(Z)
            A2 = uicontrol('Style','Radio','String','LEFT',...
                'units', 'normalized','pos',[0.25 0.0 0.25 1],...
                'FontSize', Font6,'parent',AudFnc,...
                'HandleVisibility','off','Value',0,'BackgroundColor',[0.8 0.8 0.8]);
        % Linear Reference, dB(Z)
            A3 = uicontrol('Style','Radio','String','RIGHT',...
                'units', 'normalized','pos',[0.5 0.0 0.25 1],...
                'FontSize', Font6,'parent',AudFnc,...
                'HandleVisibility','off','Value',0,'BackgroundColor',[0.8 0.8 0.8]);
        % Sensation Level Reference, dB(SL)
            A4 = uicontrol('Style','Radio','String','AVG',...
                'FontSize', Font6,...
                'units', 'normalized','pos',[0.75 0.0 0.25 1],'parent',...
                AudFnc, 'HandleVisibility','off', 'value',0,'BackgroundColor',[0.8 0.8 0.8]);


        set(A1,'Callback',{@AudCall,A1,A2,A3,A4})        % A1 - L + R
        set(A2,'Callback',{@AudCall,A1,A2,A3,A4})        % A2 - Left
        set(A3,'Callback',{@AudCall,A1,A2,A3,A4})        % A3 - Right
        set(A4,'Callback',{@AudCall,A1,A2,A3,A4})        % A4 - AVG

    function AudCall(~,~,A1,A2,A3,A4)

        if get(A1,'Value') == 1; AudFunc = 1; end
        if get(A2,'Value') == 1; AudFunc = 2; end
        if get(A3,'Value') == 1; AudFunc = 3; end
        if get(A4,'Value') == 1; AudFunc = 4; end

         ExpAppZeros(30,4)=AudFunc;
         setappdata(0,'evalue',ExpAppZeros) % Placeholder for Important Values

         Plotting_Function

    end % EOF

    % ---------------------------------------------------------------------

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% %%%%%%%%      HEADPHONE CAL FUNCTION 1                                %%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

HP1 = uipanel('Title','Participant Headphones', 'units', 'normalized',...
    'position', [0.575 0.75 0.4 0.225], 'visible','on',...
    'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],...
    'Visible','On');

    function Headphone_Cal_Function_1

        xAxis = Freq.';

        Picture2=subplot(1,1,1,'Parent',HP1);
        PlotAxesH1 = semilogx(xAxis, Headphones_A(:,2),'-b',...
                              xAxis, Headphones_A(:,3),'-r',...
                              xAxis, Headphones_A(:,4),'-m',...
                              'DisplayName','SemiLog Plot');
                              %, 'Parent', handles.graph

        set(PlotAxesH1(1),'MarkerSize',2,'LineWidth', 0.5)
        set(PlotAxesH1(2),'MarkerSize',2,'LineWidth', 0.5)

        set(PlotAxesH1(1),'MarkerFaceColor','b')
        set(PlotAxesH1(2),'MarkerFaceColor','r')

        Label=[10,100,1000,10000,100000];
        ax_PlotAxesH1 = gca; %Collect info on current axes values to gca
        set(ax_PlotAxesH1, 'units', 'normalized',...
            'OuterPosition', [0 0.2 1 0.75],...
            'Position', [0.1 0.25 0.85 0.705],...
            'XLim', [10 12500],...
            'YLim', [-15 25],...
            'FontSize', Font6,...
            'XTickLabel', Label); % Setting the x-labels to non-scientific
        grid

        HeadphoneFile1 = uicontrol('Style','text','String','No Cal. Input...',...
            'units', 'normalized','pos',[0.2 0.75 0.7 0.15],'parent',HP1,...
            'HandleVisibility','off','BackgroundColor',[0.8 0.8 0.8]);

        if H1FileName==0
            set(HeadphoneFile1,'String','No Cal. Input...')
        else
            set(HeadphoneFile1,'String',H1FileName)
            set(HeadphoneFile1,'BackgroundColor',[1 1 1])
        end

        HeadphoneLoad1 = uicontrol('Style','pushbutton','String','Load',...
            'units', 'normalized','pos',[0.1125 0.25 0.2 0.15],'parent',HP1,...
            'HandleVisibility','off',...
            'FontSize', Font8, 'FontWeight',...
            'bold','callback', {@Load_Headphones1,MainWindow});

    end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% %%%%%%%%      HEADPHONE CAL FUNCTION 2                                %%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

HP2 = uipanel('Title','HATS Headphones', 'units', 'normalized',...
    'position', [0.575 0.525 0.4 0.225], 'visible','on',...
```

```
    'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],...
    'Visible','On');

    function Headphone_Cal_Function_2

        xAxis = Freq.';

        Picture3=subplot(1,1,1,'Parent',HP2);
        PlotAxesH2 = semilogx(xAxis, Headphones_B(:,2),'-b',...
                              xAxis, Headphones_B(:,3),'-r',...
                              xAxis, Headphones_B(:,4),'-m',...
                              'DisplayName', 'SemiLog Plot');
                              %, 'Parent', handles.graph

        set(PlotAxesH2(1),'MarkerSize',2,'LineWidth', 0.5)
        set(PlotAxesH2(2),'MarkerSize',2,'LineWidth', 0.5)

        set(PlotAxesH2(1),'MarkerFaceColor','b')
        set(PlotAxesH2(2),'MarkerFaceColor','r')

        ax_PlotAxesH2 = gca; %Collect info on current axes values to gca

        Label=[10,100,1000,10000,100000];
        set(ax_PlotAxesH2, 'units', 'normalized',...
            'OuterPosition', [0 0.2 1 0.75],...
            'Position', [0.1 0.25 0.85 0.705],...
            'XLim', [10 12500],...
            'YLim', [-15 25],...
            'FontSize', Font6,...
            'XTickLabel', Label);
            % Setting the x-labels to non-scientific

        grid

        HeadphoneFile1 = uicontrol('Style','text','String','No Cal. Input...',...
            'units', 'normalized','pos',[0.2 0.75 0.7 0.15],'parent',HP2,...
            'HandleVisibility','off','BackgroundColor',[0.8 0.8 0.8]);

        if H2FileName==0
            set(HeadphoneFile1,'String','No Cal. Input...')
        else
            set(HeadphoneFile1,'String',H2FileName)
            set(HeadphoneFile1,'BackgroundColor',[1 1 1])
        end

        HeadphoneLoad1 = uicontrol('Style','pushbutton','String','Load',...
            'units', 'normalized','pos',[0.1125 0.25 0.2 0.125],'parent',HP2,...
            'HandleVisibility','off',...
            'FontSize', Font8, 'FontWeight',...
            'bold','callback', {@Load_Headphones2,MainWindow});

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% %%%%%%%%%%%%%%%     Looping Program            %%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% DISPLAY FOR MONITOR REFERENCE
%   The monitor is in place as a reference. As the signal produced by the
%   signal generator has been adjusted to account for the patricpant's
%   headphones, a similar adjustment will occur for the headphones
%   positioned on the HATS listening device. As each pair of headphones
%   requires its own signal adjustments, the paricipat and the HATS WILL
%   NOT receive the same signal amplitude; the MONITOR therefore indicates
%   the anticipated sound pressure level at each ear for the reference
%   purposes of the experimentor.
%
%   SIGNAL --> HEADPHONES-1 --> PARTICIPANT (TARGET)
%       SIGNAL = TARGET + HEADPHONES-1
%        66    =   60    +     6
%           (ie. to achieve 60 dB at the listener, a signal of 66 dB
%                must be generated)
%
%   SIGNAL --> HEADPHONES-2 --> HATS (MONITOR)
%       (MONITOR) = [TARGET + HEADPHONES-1] - HEADPHONES-2
%           62    = [  60   +      6     ]  -     4
%
Monitor_Color = [0.8 0.8 0.8];

    Monitor_Label = uicontrol('Style','text','String','MONITOR:',...
        'units', 'normalized','pos',[0.25 0.0225 0.15 0.025],...
        'HandleVisibility','off','Visible','on','FontSize',...
         Font10,'BackgroundColor', Monitor_Color);

        Monitor_Value = uicontrol('Style','text','String','00.0',...
        'units', 'normalized','pos',[0.4 0.01 0.075 0.05],...
        'HandleVisibility','off','Visible','on','FontSize',...
         Font16, 'FontWeight', 'bold','BackgroundColor', Monitor_Color);

        Monitor_Units = uicontrol('Style','text','String','dB',...
        'units', 'normalized','pos',[0.475 0.0225 0.025 0.025],...
        'HandleVisibility','off','Visible','on','FontSize',...
         Font10,'BackgroundColor', Monitor_Color);

function [Value_Disp] = StartSig(~,~,...
        Resultant,Headphones_A,Headphones_B,...
        u00Freq,u00Amp,...
        u0Type, u0Amp, u0LowFreq, u0UpFreq,...
        u1Name, u0Gain,...
        u10Freq,u10Amp,...
        u11Type, u11Amp, u11LowFreq, u11UpFreq,...
        u12Name, u12Gain,...
        PlayCount, R1, R2, WAV1_Name,WAV1_Loc,WAV2_Name,WAV2_Loc,WAV1_LnR,WAV2_LnR, WAV1_EQ,WAV2_EQ,WAV2_Lock,RdmWAV) %#ok<STOUT,INUSD>


        ExpApp= getappdata(0,'evalue');

        for GetData = 1:31
            Resultant(GetData,3)=ExpApp(GetData,5);% Column 5: Resultant L-Small (:,3)
            Resultant(GetData,5)=ExpApp(GetData,6);% Column 6: Resultant R-Small (:,5)
            Headphones_A(GetData,2)=ExpApp(GetData,7); % Column 7: Headphones 1 - Left
            Headphones_A(GetData,3)=ExpApp(GetData,8); % Column 8: Headphones 1 - Right
            Headphones_A(GetData,4)=ExpApp(GetData,15); % Column 8: Headphones 1 - Right
```

```
        Headphones_B(GetData,2)=ExpApp(GetData,9); % Column 9: Headphones 2 - Left
        Headphones_B(GetData,3)=ExpApp(GetData,10); % Column 10: Headphones 2 - Right
        Headphones_B(GetData,4)=ExpApp(GetData,16); % Column 10: Headphones 2 - Right
    end

     set(Sig1,'BackgroundColor',[0.9412    0.9412    0.9412])
     set(Sig2,'BackgroundColor',[0.9412    0.9412    0.9412])

%Signal Type for Signal 1
Sig1Type1=get(u00,'Value');
Sig1Type2=get(u0,'Value');
Sig1Type3=get(u1,'Value');

%Signal Type for Singal 2
Sig2Type1=get(u10,'Value');
Sig2Type2=get(u11,'Value');
Sig2Type3=get(u12,'Value');


% Ear Presentation for Signal 1
Sig1PresMLVal=get(u1L,'Value');
Sig1PresMRVal=get(u1R,'Value');
Sig1PresMBVal=get(u1B,'Value');

% Ear Presentation for Signal 2
Sig2PresMLVal=get(u2L,'Value');
Sig2PresMRVal=get(u2R,'Value');
Sig2PresMBVal=get(u2B,'Value');

spacer='    ';
spacer11='    ';spacer12='    ';spacer13='    ';
spacer21='    ';spacer22='    ';spacer23='    ';

if Sig1Type1==1; spacer11='--->'; end
if Sig1Type2==1; spacer12='--->'; end
if Sig1Type3==1; spacer13='--->'; end
if Sig2Type1==1; spacer21='--->'; end
if Sig2Type2==1; spacer22='--->'; end
if Sig2Type3==1; spacer23='--->'; end

if Sig1PresMLVal==1; ear1='L';  end
if Sig1PresMRVal==1; ear1='R';  end
if Sig1PresMBVal==1; ear1='B';  end
if Sig2PresMLVal==1; ear2='L';  end
if Sig2PresMRVal==1; ear2='R';  end
if Sig2PresMBVal==1; ear2='B';  end


%Singal 1 Playback Settings
    %S1_ToneFreq = str2num(get(u00Freq ,'string'))
    S1_PureToneFreq = get(u00Freq ,'value');
    S1_ToneFreq=Freq(1,S1_PureToneFreq);
    S1_ToneAmp = str2num(get(u00Amp ,'string'));
    S1_NoiseType = get(u0Type ,'value');
    S1_NoiseAmp = str2num(get(u0Amp ,'string'));
    S1_NoiseLFreq = str2num(get(u0LowFreq ,'string'));
    S1_NoiseUFreq = str2num(get(u0UpFreq ,'string'));
    S1_WAVName = get(u1Name ,'string');
    S1_WAVGain = str2num(get(u0Gain ,'string'));
        if S1_NoiseType == 1; S1_NType='White'; end
        if S1_NoiseType == 2; S1_NType='Pink'; end
        if S1_NoiseType == 3; S1_NType='Brown'; end

%Signal 2 Playback Settings

runcount=get (StartPlayback,'Value');
   if runcount==1
    RandomCheck = get(u10Random,'Value'); %Check for Random Start Box
        if RandomCheck == 1
            S2_ToneAmp = S1_ToneAmp+randi([-1*plusminus plusminus],1,1);
            set(u10Amp ,'string',num2str(S2_ToneAmp));
            runcount=0;
        else
            S2_ToneAmp = str2num(get(u10Amp ,'string'));
            runcount=0;
        end

    RandomCheck2 = get(u11Random,'Value'); %Check for NOISE Random Start Box
        if RandomCheck2 == 1
            S2_NoiseAmp = S1_NoiseAmp+randi([-1*plusminusN plusminusN],1,1);
            set(u11Amp ,'string',num2str(S2_NoiseAmp));
            runcount=0;
        else
            S2_NoiseAmp = str2num(get(u11Amp ,'string'));
            runcount=0;
        end

    RandomCheck3 = get(RdmWAV,'Value'); %Check for WAV Random Start Box
        if RandomCheck3 == 1
            S2_WAVAmp = S1_WAVGain+randi([-1*plusminusWAV plusminusWAV],1,1);
            set(u12Gain ,'string',num2str(S2_WAVAmp));
            runcount=0;
        else
            S2_WAVAmp = str2num(get(u12Gain ,'string'));
            runcount=0;
        end
   end

    %S2_ToneFreq = str2num(get(u10Freq ,'string'))
    S2_PureToneFreq = get(u10Freq ,'value');
    S2_ToneFreq=Freq(1,S2_PureToneFreq);
    S2_NoiseType = get(u11Type ,'value');
    S2_NoiseAmp = str2num(get(u11Amp ,'string'));
    S2_NoiseLFreq = str2num(get(u11LowFreq ,'string'));
    S2_NoiseUFreq = str2num(get(u11UpFreq ,'string'));
    S2_WAVName = get(u12Name ,'string');
    S2_WAVGain = str2num(get(u12Gain ,'string'));
        if S2_NoiseType == 1; S2_NType='White'; end
        if S2_NoiseType == 2; S2_NType='Pink'; end
        if S2_NoiseType == 3; S2_NType='Brown'; end


Linear=0; Audiogram=0;
```

```
% Reference Used
if get(R1,'value')==1; Linear=1; end
if get(R2,'value')==1; Audiogram=1; end

AudL=0;AudR=0;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% CORRECTION FACTORS TO USE %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
S1_ROW = get(u00Freq ,'value'); %Frequency Row Value
S1_ToneFreq=Freq(1,S1_ROW);     %Frequency

AudL1 = Resultant(S1_ROW,3);     %Audiometer L-Sensitivity
AudR1 = Resultant(S1_ROW,5);     %Audiometer R-Sensitivity

if get(A1,'value')==1; AudL = AudL1; AudR = AudR1; end    % A1 - L + R
if get(A2,'value')==1; AudL = AudL1; AudR = AudL1; end    % A2 - Left
if get(A3,'value')==1; AudL = AudR1; AudR = AudR1; end    % A3 - Right
if get(A4,'value')==1; AudL = (AudL1+AudR1)/2; AudR = (AudL1+AudR1)/2; end    % A4 - AVG

H1_L = Headphones_A(S1_ROW,2);  %Participant Headphones L-Adjustment
H1_R = Headphones_A(S1_ROW,3);  %Participant Headphones R-Adjustment
H1_B = Headphones_A(S1_ROW,4);  %Participant Headphones B-Adjustment

H2_L = Headphones_B(S1_ROW,2);  %HATS Headphones L-Adjustment
H2_R = Headphones_B(S1_ROW,3);  %HATS Headphones R-Adjustment
H2_B = Headphones_B(S1_ROW,4);  %HATS Headphones B-Adjustment

%%% Setting Correction for Reference Used

    %%% Linear Reference
        if Linear==1;
            %fprintf('\nLinear dB(Z)\n')
            H1Corr_S1L = H1_L; %S1L + H1L (Singal 1 on Headphones 1)
            H1Corr_S1R = H1_R; %S1R + H1R (Singal 1 on Headphones 1)
            H1Corr_S2L = H1_L; %S2L + H1L (Singal 2 on Headphones 1)
            H1Corr_S2R = H1_R; %S2R + H1R (Singal 2 on Headphones 1)

            H2Corr_S1L = H2_L; %S1L + H2L (Singal 1 on Headphones 2)
            H2Corr_S1R = H2_R; %S1R + H2R (Singal 1 on Headphones 2)
            H2Corr_S2L = H2_L; %S2L + H2L (Singal 2 on Headphones 2)
            H2Corr_S2R = H2_R; %S2R + H2R (Singal 2 on Headphones 2)
        end

    %%% Sensation Level Reference
        if Audiogram==1;
            %fprintf('\nSensation Level dB(SL)\n')
            H1Corr_S1L = H1_L + AudL; %S1L + H1L + AudCorrL (Singal 1 on Headphones 1 as SL)
            H1Corr_S1R = H1_R + AudR; %S1R + H1R + AudCorrR (Singal 1 on Headphones 1 as SL)
            H1Corr_S2L = H1_L + AudL; %S2L + H1L + AudCorrL (Singal 2 on Headphones 1 as SL)
            H1Corr_S2R = H1_R + AudR; %S2R + H1R + AudCorrR (Singal 2 on Headphones 1 as SL)

            H2Corr_S1L = H2_L; %S1L + H2L + AudCorrL (Singal 1 on Headphones 2 as SL)
            H2Corr_S1R = H2_R; %S1R + H2R + AudCorrR (Singal 1 on Headphones 2 as SL)
            H2Corr_S2L = H2_L; %S2L + H2L + AudCorrL (Singal 2 on Headphones 2 as SL)
            H2Corr_S2R = H2_R; %S2R + H2R + AudCorrR (Singal 2 on Headphones 2 as SL)

        end

PlayCount=0; % Loop Break Safeguard

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Collect WAV File Information before While Loop

    if Sig1Type3==1;  % Signal 1 WAV File
            FileLoc1 = get(WAV1_Loc ,'string');
            FileNam1 = get(WAV1_Name ,'string');
            [S1y, S1Fs,S1nbit] = wavread( fullfile(FileLoc1,FileNam1) );
            Info=size(S1y);
            Length = (Info(1,1))/S1Fs;
            Length = round((Length*1000))/1000; % Rounding Decimal Places
            Channels=Info(1,2);
            Factor = 1/100;
            S1y = (S1_WAVGain * Factor) * S1y;
    end

    if Sig2Type3==1;  % Signal 2 WAV File
            FileLoc2 = get(WAV2_Loc ,'string');
            FileNam2 = get(WAV2_Name ,'string');
            [S2y, S2Fs,S2nbit] = wavread( fullfile(FileLoc2,FileNam2) );
            Info2=size(S2y);
            Length2=(Info2(1,1))/S2Fs;
            Length2 = round((Length2*1000))/1000; % Rounding Decimal Places
            Channels2=Info2(1,2);
            Factor = 1/100;
            S2y = (S2_WAVGain * Factor) * S2y;
            AvgLnR1 = get(WAV1_LnR,'value');
    end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%-------------------------
 % Setting Up Equalizer Filter Bank - Chebyshev Filters
%-------------------------
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

 Fs=44100;

 Const_Rp = 0.25;
 Const_Atten = 30;

 CL = zeros(1,33);
 RS = zeros(1,33);
 Rp = zeros(1,33); % Ripple Value
 BandSlope = zeros(1,33); % Ripple Value

     for Val_Set = 1:34
         CL(Val_Set) = 1;
         Rs(Val_Set) = Const_Atten;
         Rp(Val_Set) = Const_Rp;
         BandSlope(Val_Set) = -30;

         if Val_Set > 2;  BandSlope(Val_Set) = -30;  end
         if Val_Set > 3;  BandSlope(Val_Set) = -50;  end
         if Val_Set > 7;  BandSlope(Val_Set) = -90;  end
```

```
            if Val_Set > 15; BandSlope(Val_Set) = -100; end
            if Val_Set > 30; BandSlope(Val_Set) = -150; end
            if Val_Set > 32; BandSlope(Val_Set) = -150; end
            if Val_Set > 33; BandSlope(Val_Set) = -150; end
        end

    % Constants for Caclulating Important Frequnencies (ANSI S1.11-2004)
        G = 10^(3/10);
        b = 3;
        fref = 1000;

% LOW PASS FILTER
    % 0-25 Hz (25Hz has a upper bandedge = 28.18)

    BN = 13; % Lowpass Filter               (25 Hz)
        FN = BN-11;
        Slope = BandSlope(FN);

    fm = (G^((BN-30)/b))*(fref);
        f_low = (G^(-1/(2*b)))*(fm);
        f_upp = (G^(+1/(2*b)))*(fm);

        y_int_upp = (Rs(1)-3)-Slope*log(f_upp); % (x0,y0)

    fsu = exp((Rs(1)-y_int_upp)/(Slope));
    fpu = exp(((0)-y_int_upp)/(Slope));

    Fp1 = fsu / (Fs/2);
    Fs1 = fpu /(Fs/2);
    n1 = cheblord(Fp1,Fs1,Rp(1),Rs(1));
    [b1,a1] = chebyl(n1,Rp(1),Fp1,'low');

    RowCount = 1;

    b_coef(1,RowCount) = {b1};
    a_coef(1,RowCount) = {a1};

% BAND-PASS FILTER BANK

for filt_plot = 13:41
    BN = filt_plot + 1;               % 15-42
        FN = BN-11;
        Slope = BandSlope(FN);

    fm = (G^((BN-30)/b))*(fref);
        f_low = (G^(-1/(2*b)))*(fm);
        f_upp = (G^(+1/(2*b)))*(fm);

    y_int_low = (Rs(FN)-3)-(-1*Slope)*log(f_low); % (x0,y0)
        fpl = exp(((0)-y_int_low)/(-1*Slope));
        f_low;
        fsl = exp((Rs(FN)-y_int_low)/(-1*Slope));


    y_int_upp = (Rs(FN)-3)-Slope*log(f_upp); % (x0,y0)
        fsu = exp((Rs(FN)-y_int_upp)/(Slope));
        f_upp;
        fpu = exp(((0)-y_int_upp)/(Slope));

    Fp2=[fsl,fsu]/(Fs/2);
    Fs2=[fpl,fpu]/(Fs/2);
    n2 = cheblord(Fp2,Fs2,Rp(FN),Rs(FN));
    [b2,a2] = chebyl(n2,Rp(FN),Fp2);                %BAND-PASS FILTER

    H = 0;
        Acc = 2048*2;                                   %Resolution of Plot
        H = H+10^(CL(FN)/20)*abs(freqz(b2,a2,Acc/2));

    RowCount = RowCount + 1;

    b_coef(1,RowCount) = {b2};
    a_coef(1,RowCount) = {a2};
end

    BN = 43; % Lowpass Filter
        FN = BN-11;
        Slope = BandSlope(FN);

    fm = (G^((BN-30)/b))*(fref);
        f_low = (G^(-1/(2*b)))*(fm);
        f_upp = (G^(+1/(2*b)))*(fm);

    y_int_low = (Rs(FN)-3)-(-1*Slope)*log(f_low); % (x0,y0)
        fpl = exp(((0)-y_int_low)/(-1*Slope));
        f_low;
        fsl = exp((Rs(FN)-y_int_low)/(-1*Slope));


    Fp1 = fsl / (Fs/2);
    Fs1 = fpl /(Fs/2);
    n1 = cheblord(Fp1,Fs1,Rp(FN),Rs(FN));
    [b1,a1] = chebyl(n1,Rp(FN),Fp1,'high');
    RowCount = RowCount + 1;

    b_coef(1,RowCount) = {b1};
    a_coef(1,RowCount) = {a1};


 H=0;
Acc = 2048*2;
for i=1:30

     H=H+10^(CL(i)/20)*abs(freqz(b_coef{i},a_coef{i},Acc/2));
end

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   %-------------------------
    %While Loop For Playback
   %-------------------------
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    while get (StartPlayback,'Value')
        set(StartPlayback,'string', 'STOP') %Replace with STOP Button
        set(StartPlayback,'BackgroundColor',[1 0 0]) %Turn Button Red
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%  CHECK TO MAKE SURE CALIBRATION FILES ARE LOADED  %%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if H1_L==0; set(StartPlayback,'Value',0); headphonecheck; break; end
if H1_R==0; set(StartPlayback,'Value',0); headphonecheck; break; end
if H2_L==0; set(StartPlayback,'Value',0); headphonecheck; break; end
if H2_R==0; set(StartPlayback,'Value',0); headphonecheck; break; end


%%% Check specifically for AUDIOGRAM and SENSATION LOSS
if Audiogram==1;
        % If there is no threshold information present
        % (tone was not heard)
        if AudL ==-20; set(StartPlayback,'Value',0); AudCheck1; break;  end
        if AudR ==-20; set(StartPlayback,'Value',0); AudCheck1; break;  end
        % If there is no audiogram loaded
        if AudL==0; set(StartPlayback,'Value',0); AudCheck2; break; end
        if AudR==0; set(StartPlayback,'Value',0); AudCheck2; break; end
end

%CHECK BREAK
if PlayCount==500;
    break;
end

%----------------------
%%%%%%% SIGNAL 1
%----------------------

set(SignalFeedback,'Color',Colour1)
set(SignalLabel,'String','Reference Signal')

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %%% CORRECTION FACTORS TO USE %%%
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        S1_ROW = get(u00Freq ,'value'); %Frequency Row Value
        S1_ToneFreq=Freq(1,S1_ROW); %Frequency
%         AudL = Resultant(S1_ROW,3);  %Audiometer L-Sensitivity
%         AudR = Resultant(S1_ROW,5);  %Audiometer R-Sensitivity

        AudL1 = Resultant(S1_ROW,3);     %Audiometer L-Sensitivity
        AudR1 = Resultant(S1_ROW,5);     %Audiometer R-Sensitivity

        if get(A1,'value')==1; AudL = AudL1; AudR = AudR1; end    % A1 - L + R
        if get(A2,'value')==1; AudL = AudL1; AudR = AudL1; end    % A2 - Left
        if get(A3,'value')==1; AudL = AudR1; AudR = AudR1; end    % A3 - Right
        if get(A4,'value')==1; AudL = (AudL1+AudR1)/2; AudR = (AudL1+AudR1)/2; end    % A4 - AVG

        H1_L = Headphones_A(S1_ROW,2);  %Participant Headphones L-Adjustment
        H1_R = Headphones_A(S1_ROW,3);  %Participant Headphones R-Adjustment
        H1_B = Headphones_A(S1_ROW,4);  %Participant Headphones B-Adjustment

        H2_L = Headphones_B(S1_ROW,2);  %HATS Headphones L-Adjustment
        H2_R = Headphones_B(S1_ROW,3);  %HATS Headphones R-Adjustment
        H2_B = Headphones_B(S1_ROW,4);  %HATS Headphones B-Adjustment

        %%% Setting Correction for Reference Used
            %%% Linear Reference
                if Linear==1;
                    %fprintf('\nLinear dB(Z)\n')
                    H1Corr_S1L = H1_L; %S1L + H1L (Singal 1 on Headphones 1)
                    H1Corr_S1R = H1_R; %S1R + H1R (Singal 1 on Headphones 1)
                    H1Corr_S2L = H1_L; %S2L + H1L (Singal 2 on Headphones 1)
                    H1Corr_S2R = H1_R; %S2R + H1R (Singal 2 on Headphones 1)

                    H2Corr_S1L = H2_L; %S1L + H2L (Singal 1 on Headphones 2)
                    H2Corr_S1R = H2_R; %S1R + H2R (Singal 1 on Headphones 2)
                    H2Corr_S2L = H2_L; %S2L + H2L (Singal 2 on Headphones 2)
                    H2Corr_S2R = H2_R; %S2R + H2R (Singal 2 on Headphones 2)
                end

            %%% Sensation Level Reference
                if Audiogram==1;
                    %fprintf('\nSensation Level dB(SL)\n')
                    H1Corr_S1L = H1_L + AudL; %S1L + H1L + AudCorrL (Singal 1 on Headphones 1 as SL)
                    H1Corr_S1R = H1_R + AudR; %S1R + H1R + AudCorrR (Singal 1 on Headphones 1 as SL)
                    H1Corr_S2L = H1_L + AudL; %S2L + H1L + AudCorrL (Singal 2 on Headphones 1 as SL)
                    H1Corr_S2R = H1_R + AudR; %S2R + H1R + AudCorrR (Singal 2 on Headphones 1 as SL)

                    H2Corr_S1L = H2_L; %S1L + H2L + AudCorrL (Singal 1 on Headphones 2 as SL)
                    H2Corr_S1R = H2_R; %S1R + H2R + AudCorrR (Singal 1 on Headphones 2 as SL)
                    H2Corr_S2L = H2_L; %S2L + H2L + AudCorrL (Singal 2 on Headphones 2 as SL)
                    H2Corr_S2R = H2_R; %S2R + H2R + AudCorrR (Singal 2 on Headphones 2 as SL)

                end


set(Sig1,'BackgroundColor',[0.5 0.5 0.5])
set(Sig2,'BackgroundColor',[0.9412    0.9412    0.9412])


    % Collect User Settings Information
    CurrentSettings= getappdata(0,'evalue');
    Cur_Playback1 = CurrentSettings(1,4);
    Cur_Pause1 = CurrentSettings(2,4);
    Cur_Fade1 = CurrentSettings(3,4);
    Cur_Phase1 = CurrentSettings(4,4);

%%%% PURE TONE
if Sig1Type1==1;

    % Determine which ear will be presented to
    if ear1 == 'L'
        S1_L_Monitor=S1_ToneAmp+H1Corr_S1L-H2Corr_S1L;
        Amplitude(1,1)=S1_ToneAmp+H1Corr_S1L;
        Amplitude(1,2)=S1_ToneAmp+H1Corr_S1R;

        set(Monitor_Label,'String','MONITOR (S1):');
        set(Monitor_Value,'String',S1_L_Monitor);

        if S1_L_Monitor>=ExpAppZeros(S1_PureToneFreq,13); set(StartPlayback,'Value',0); MaxSPLCheck; break; end
```

```
                [  t, left, right, fs ] = PlotTone_adj_v3 (S1_ToneFreq, Cur_Fade1, Cur_Playback1*1000, Amplitude, ear1, 1, Phase);
                  player=audioplayer(left,fs);
%                   fprintf('Left Player \n');
                  play(player)

                  pause(max(size(left))/fs);

            end

            if ear1 == 'R'
                S1_R_Monitor = S1_ToneAmp + H1Corr_S1R - H2Corr_S1R;
                Amplitude(1,1) = S1_ToneAmp + H1Corr_S1L;
                Amplitude(1,2) = S1_ToneAmp + H1Corr_S1R;

%                 fprintf('Participant Left SPL (%d dB)\n', Amplitude(1,1));
%                 fprintf('Participant Right SPL (%d dB)\n', Amplitude(1,2));

                set(Monitor_Label,'String','MONITOR (S1):');
                set(Monitor_Value,'String',S1_R_Monitor);

                if S1_R_Monitor>=ExpAppZeros(S1_PureToneFreq,13); set(StartPlayback,'Value',0); MaxSPLCheck; break; end

                [  t, left, right, fs ] = PlotTone_adj_v3 (S1_ToneFreq, Cur_Fade1, Cur_Playback1*1000, Amplitude, ear1, 1, Phase);
                  player=audioplayer(right,fs);
%                   fprintf('Right Player \n');
                  play(player)
                  pause(max(size(right))/fs);
            end

            if ear1 == 'B'
                S1_L_Monitor=S1_ToneAmp+H1Corr_S1L-H2Corr_S1L;
                S1_R_Monitor=S1_ToneAmp+H1Corr_S1R-H2Corr_S1R;

                Amplitude(1,1)=S1_ToneAmp+H1Corr_S1L+H1_B;
                Amplitude(1,2)=S1_ToneAmp+H1Corr_S1R+H1_B;

                set(Monitor_Label,'String','Bin. Mon. (S1-R):');
                set(Monitor_Value,'String',S1_R_Monitor);

                if S1_L_Monitor>=ExpAppZeros(S1_PureToneFreq,13); set(StartPlayback,'Value',0); MaxSPLCheck; break;
                elseif S1_R_Monitor>=ExpAppZeros(S1_PureToneFreq,13); set(StartPlayback,'Value',0); MaxSPLCheck; break; end

                [  t, left, right, fs ] = PlotTone_adj_v3 (S1_ToneFreq, Cur_Fade1, Cur_Playback1*1000, Amplitude, 'B', 1, Phase);
                  player=audioplayer([left,right],fs);
%                   fprintf('Binaural Player \n');
                  play(player)
                  pause(max(size(left))/fs);
                %[  t, left, right, fs ] = PlotTone_adj_v3 (S1_ToneFreq, Cur_Fade1, Cur_Playback1*1000, S1_ToneAmp, 'R', 1, Phase+Phase1);
%                   PhaseLConv=Phase*360/(2*pi);
%                   PhaseRConv=(Phase+Phase1)*360/(2*pi);
%                  fprintf('\nPhase angle of Left Channel = %d \nPhase angle of Right Channel = %d\n',PhaseLConv,PhaseRConv);
            end
        end

        %%% NOISE SAMPLE
        if Sig1Type2==1;
            CurrentSettings= getappdata(0,'evalue');
             Cur_Playback1 = CurrentSettings(1,4);
             Cur_Pause1 = CurrentSettings(2,4);
             Cur_Fade1 = CurrentSettings(3,4);
             Cur_Phase1 = CurrentSettings(4,4);

            H1_L = Headphones_A(S1_ROW,2);  %Participant Headphones L-Adjustment
            H1_R = Headphones_A(S1_ROW,3);  %Participant Headphones R-Adjustment
            H1_B = Headphones_A(S1_ROW,4);  %Participant Headphones B-Adjustment

            Fs = 44100;

            S1_NoiseAmp     = str2num(get(u0Amp ,'string'));
            S1_NoiseType    = get(u0Type ,'value');
            S1_NoiseLFreq   = str2num(get(u0LowFreq ,'string'));
            S1_NoiseUFreq   = str2num(get(u0UpFreq ,'string'));


              % No filter available for 20 kHz
                FreqN = [20, 25, 31.5, 40, 50, 63, 80, 100, 125, 160, 200, 250, 315, 400,...
                        500, 630, 800, 1000, 1250, 1600, 2000, 2500, 3150, 4000, 5000,...
                        6300, 8000, 10000, 12500, 16000].';

                dBA = [-50.5, -44.7, -39.4, -34.6, -30.2, -26.2, -22.5, -19.1, -16.1, -13.4, -10.9, -8.6, -6.6, -4.8,...
                         -3.2, -1.9, -0.8, 0.0, 0.6, 1.0, 1.2, 1.3, 1.2, 1.0, 0.5,...
                         -0.1, -1.1, -2.5, -4.3, -6.6].';

                A_Weight = [FreqN; dBA];    % For Grey Noise Presentation
                BCorr = zeros(length(dBA),1);

                if S2_NoiseType == 6;
                    BCorr = -1 * dBA;
                end

            LowFreqRow = get(u0LowFreq ,'value'); % 20 = 1,  12500 = 29, 20000 = 31
            UppFreqRow = get(u0UpFreq  ,'value');

            LpCheck = 0;

            if S1_NoiseType == 1; % 'Violet (+6 dB/oct.)',      +2 dB/3rd.
                ThrdAdd =  2;
                AmpShft = 0;
                Lp=S1_NoiseAmp;
                LpTot=0;
                for Lpi = 1:31
                    if Lpi >= LowFreqRow
                       if Lpi <= UppFreqRow
                           LpTot=10*log10(10^((S1_NoiseAmp+((Lpi-1)*(2)))/10) + 10^(LpTot/10));
                           LpCheck(:,Lpi) = (S1_NoiseAmp+((Lpi-1)*(2)));
                       end
                    end
                end
                AmpShft = -1*(LpTot - S1_NoiseAmp); % Shift to ensure target SPL is met given noise band width
            end

            if S1_NoiseType == 2;  % 'Blue (+3 dB/oct.)',        +1 dB/3rd.
                ThrdAdd =  1;
                AmpShft = 0;
```

```
    Lp=S1_NoiseAmp;
    LpTot=0;
    for Lpi = 1:31
        if Lpi >= LowFreqRow
           if Lpi <= UppFreqRow
               LpTot=10*log10(10^((S1_NoiseAmp+((Lpi-1)*(1)))/10) + 10^(LpTot/10));
               LpCheck(:,Lpi) = (S1_NoiseAmp+((Lpi-1)*(1)));
           end
        end
    end
    AmpShft = -1*(LpTot - S1_NoiseAmp);
end

if S1_NoiseType == 3;  % 'White (+0 dB/oct.)',       +0 dB/3rd.
    ThrdAdd =  0;
    AmpShft = 0;
    Lp=S1_NoiseAmp;
    LpTot=0;
    for Lpi = 1:31
        if Lpi >= LowFreqRow
           if Lpi <= UppFreqRow
               LpTot=10*log10(10^((S1_NoiseAmp+((Lpi-1)*(0)))/10) + 10^(LpTot/10));
               LpCheck(:,Lpi) = (S1_NoiseAmp+((Lpi-1)*(0)));
           end
        end
    end
    AmpShft = -1*(LpTot - S1_NoiseAmp);
end

if S1_NoiseType == 4;  % 'Pink (-3 dB/oct.)',        -1 dB/3rd.
    ThrdAdd = -1;
    AmpShft = 0;
    Lp=S1_NoiseAmp;
    LpTot=0;
    for Lpi = 1:31
        if Lpi >= LowFreqRow
           if Lpi <= UppFreqRow
               LpTot=10*log10(10^((S1_NoiseAmp+((Lpi-1)*(-1)))/10) + 10^(LpTot/10));
               LpCheck(:,Lpi) = (S1_NoiseAmp+((Lpi-1)*(-1)));
           end
        end
    end
    AmpShft = -1*(LpTot - S1_NoiseAmp);
end

if S1_NoiseType == 5;  % 'Brownian (-6 dB/oct.)',    -2 dB/3rd.
    ThrdAdd = -2;
    AmpShft = 0;
    Lp=S1_NoiseAmp;
    LpTot=0;
    for Lpi = 1:31
        if Lpi >= LowFreqRow
           if Lpi <= UppFreqRow
               LpTot=10*log10(10^((S1_NoiseAmp+((Lpi-1)*(-2)))/10) + 10^(LpTot/10));
               LpCheck(:,Lpi) = (S1_NoiseAmp+((Lpi-1)*(-2)));
           end
        end
    end
    AmpShft = -1*(LpTot - S1_NoiseAmp);
end

if S1_NoiseType == 6;  % 'Grey (Inv. A-Weight)'
    ThrdAdd =  0;
    AmpShft = 0;
    Lp=S1_NoiseAmp;
    LpTot=0;
    for Lpi = 1:31
        if Lpi >= LowFreqRow
           if Lpi <= UppFreqRow
               LpTot=10*log10(10^((S1_NoiseAmp+(BCorr(Lpi,1)))/10) + 10^(LpTot/10));
               LpCheck(:,Lpi) = (S1_NoiseAmp+(BCorr(Lpi,1)));
           end
        end
    end
    AmpShft = -1*(LpTot - S1_NoiseAmp);
end

H_L(:,1) = Headphones_A(:,2);
H_R(:,1) = Headphones_A(:,3);  %Participant Headphones R-Adjustment

    dBA = [-50.5, -44.7, -39.4, -34.6, -30.2, -26.2, -22.5, -19.1, -16.1, -13.4, -10.9, -8.6, -6.6, -4.8,...
             -3.2, -1.9, -0.8, 0.0, 0.6, 1.0, 1.2, 1.3, 1.2, 1.0, 0.5,...
             -0.1, -1.1, -2.5, -4.3, -6.6].';

    BCorr = zeros(length(dBA),1);

    if S1_NoiseType == 6;
        BCorr = -1 * dBA;
    end

CL = 0; CR = 0;

[CL,CR] = Apply_filter(S1_NoiseAmp, ThrdAdd, AmpShft, H_L, H_R,LowFreqRow,UppFreqRow,S1_NoiseType);


x = wgn(Cur_Playback1*Fs, 1, 0);
yL = zeros(length(x),30);
yR = zeros(length(x),30);

% i=1;  % --> 25 Hz   (2)
% i=2;  % --> 31.5 Hz (3)
% i=3;  % --> 40 Hz
% i=20; % --> 2 kHz
% i=17; % --> 1 kHz   (18)
% i=24; % --> 5 kHz
% i=26; % --> 8 kHz
% i=27; % --> 10 kHz
% i=28; % --> 12.5 kHz
% i=29; % --> 16 kHz (1 is too high, 0.5 --> 73dB)
% i=30; %  --> 16 kHz (1 is too high, 0.5 --> 74dB)

for filt = 1:30
    fi=filt;
    yL(:,fi) = filter((CL(fi))*b_coef{fi},a_coef{fi},x);
```

```
                yR(:,fi) = filter((CR(fi))*b_coef{fi},a_coef{fi},x);
            end

            yLtot = yL(:,1) + yL(:,2) + yL(:,3) + yL(:,4) + yL(:,5) + yL(:,6) + yL(:,7) + yL(:,8) + yL(:,9) + yL(:,10) + yL(:,11) + yL(:,12) +
yL(:,13) + yL(:,14) + yL(:,15) + yL(:,16) + yL(:,17) + yL(:,18) + yL(:,19) + yL(:,20) + yL(:,21) + yL(:,22) + yL(:,23) + yL(:,24) + yL(:,25) +
yL(:,26) + yL(:,27) + yL(:,28) + yL(:,29) + yL(:,30);
            yRtot = yR(:,1) + yR(:,2) + yR(:,3) + yR(:,4) + yR(:,5) + yR(:,6) + yR(:,7) + yR(:,8) + yR(:,9) + yR(:,10) + yR(:,11) + yR(:,12) +
yR(:,13) + yR(:,14) + yR(:,15) + yR(:,16) + yR(:,17) + yR(:,18) + yR(:,19) + yR(:,20) + yR(:,21) + yR(:,22) + yR(:,23) + yR(:,24) + yR(:,25) +
yR(:,26) + yR(:,27) + yR(:,28) + yR(:,29) + yR(:,30);

            fade_window1 = @(N)( hanning(N).^2 );
            yLtot = fade( yLtot, Fs, 200, fade_window1);
            yRtot = fade( yRtot, Fs, 200, fade_window1);

            Y = [yLtot yRtot];
            y0 = zeros(size(yLtot));

            if ear1 == 'L'; Y = [yLtot y0]; end
            if ear1 == 'R'; Y = [y0 yRtot]; end
            if ear1 == 'B'; Y = [yLtot yRtot]; end

            player=audioplayer(Y, Fs);
                play(player)
                pause(Cur_Playback1);

        end

        %%%%%%%%%%%%%%%
        %%% WAV FILE %%%
        %%%%%%%%%%%%%%%

        if Sig1Type3==1;

            AvgLnR1 = get(WAV1_LnR,'value');

                S1_WAVGain = str2num(get(u0Gain ,'string'));
                EQ_Check = get(WAV1_EQ ,'value');

                % Check Number of Channels
                InfoS1=size(S1y);
                ChannelsS1=InfoS1(1,2);

                S1yTemp = zeros(size(S1y));


                % size(Headphones_A) = 31 4

                if ChannelsS1 == 2;         % Stereo File

                    AvgLnR1 = get(WAV1_LnR,'value');

                    if ear1 == 'L';
                        S1yTemp(:,1) = (S1_WAVGain * Factor) * S1y(:,1);
                        % Option 2: Combine Left and Right onto Left
                        if AvgLnR1 == 1
                            S1yTemp2ch = (S1_WAVGain * Factor) * S1y;
                            S1yTemp(:,1) = (S1yTemp2ch(:,1) + S1yTemp2ch(:,2)) / 2;
                        end
                    end

                    if ear1 == 'R';
                        S1yTemp(:,2) = (S1_WAVGain * Factor) * S1y(:,2);
                        % Option 2: Combine Left and Right onto Left
                        if AvgLnR1 == 1
                            S1yTemp2ch = (S1_WAVGain * Factor) * S1y;
                            S1yTemp(:,2) = (S1yTemp2ch(:,1) + S1yTemp2ch(:,2)) / 2;
                        end
                    end

                    if ear1 == 'B';
                        S1yTemp = (S1_WAVGain * Factor) * S1y;
                    end

                else                        % Monaural File (Duplicate ch.s)

                    S1yTemp = [zeros(size(S1y)), zeros(size(S1y))];

                    if ear1 == 'L';
                        S1yTemp(:,1) = (S1_WAVGain * Factor) * S1y(:,1);
                    end

                    if ear1 == 'R';
                        S1yTemp(:,2) = (S1_WAVGain * Factor) * S1y(:,1);
                    end

                    if ear1 == 'B';
                        S1yTempMon = (S1_WAVGain * Factor) * S1y;
                        S1yTemp = [S1yTempMon, S1yTempMon];
                    end

                end


                if EQ_Check == 1;
                    H_L(:,1) = Headphones_A(:,2);
                    H_R(:,1) = Headphones_A(:,3);  %Participant Headphones R-Adjustment

%                     CL = exp((H_L-60.635)/8.9141);
%                     CR = exp((H_R-60.635)/8.9141);

                    CL = exp((H_L + S1_WAVGain * Factor)/8.9141);
                    CR = exp((H_R + S1_WAVGain * Factor)/8.9141);

                    yL = zeros(length(S1yTemp(:,1)),30);
                    yR = zeros(length(S1yTemp(:,2)),30);

                    for filt = 1:30
                        fi=filt;
                        yL(:,fi) = filter((CL(fi))*b_coef{fi},a_coef{fi},S1yTemp(:,1));
                        yR(:,fi) = filter((CR(fi))*b_coef{fi},a_coef{fi},S1yTemp(:,2));
                    end
```

```
                    yLtot = yL(:,1) + yL(:,2) + yL(:,3) + yL(:,4) + yL(:,5) + yL(:,6) + yL(:,7) + yL(:,8) + yL(:,9) + yL(:,10) + yL(:,11) +
yL(:,12) + yL(:,13) + yL(:,14) + yL(:,15) + yL(:,16) + yL(:,17) + yL(:,18) + yL(:,19) + yL(:,20) + yL(:,21) + yL(:,22) + yL(:,23) + yL(:,24) +
yL(:,25) + yL(:,26) + yL(:,27) + yL(:,28) + yL(:,29) + yL(:,30);
                    yRtot = yR(:,1) + yR(:,2) + yR(:,3) + yR(:,4) + yR(:,5) + yR(:,6) + yR(:,7) + yR(:,8) + yR(:,9) + yR(:,10) + yR(:,11) +
yR(:,12) + yR(:,13) + yR(:,14) + yR(:,15) + yR(:,16) + yR(:,17) + yR(:,18) + yR(:,19) + yR(:,20) + yR(:,21) + yR(:,22) + yR(:,23) + yR(:,24) +
yR(:,25) + yR(:,26) + yR(:,27) + yR(:,28) + yR(:,29) + yR(:,30);

                    Y = [yLtot yRtot];
                    y0 = zeros(size(yLtot));

                    if ear1 == 'L'; S1yTemp = [yLtot y0]; end
                    if ear1 == 'R'; S1yTemp = [y0 yRtot]; end
                    if ear1 == 'B'; S1yTemp = [yLtot yRtot]; end

                end


                    fade_window1 = @(N)( hanning(N).^2 );
                    S1yTemp(:,1) = fade( S1yTemp(:,1), Fs, 100, fade_window1);
                    S1yTemp(:,2) = fade( S1yTemp(:,2), Fs, 100, fade_window1);

                player=audioplayer(S1yTemp,S1Fs,S1nbit);

%               fprintf('Right Player \n');
                play(player)
                pause(max(size(S1yTemp))/S1Fs);

            end


            %%%%%%% PAUSE 1
             set(Sig1,'BackgroundColor',[0.9412    0.9412    0.9412])

            pause(Cur_Pause1)


                % Collect User Settings Information for Signal 2
                CurrentSettings= getappdata(0,'evalue');
                Cur_Playback2 = CurrentSettings(5,4);
                Cur_Pause2 = CurrentSettings(6,4);
                Cur_Fade2 = CurrentSettings(7,4);
                Cur_Phase2 = CurrentSettings(8,4);


            %----------------------
            %%%%%%% SIGNAL 2 %%%%%%%
            %----------------------

            set(Signal1Feedback,'Color',Colour2) %BgGrey
            set(Signal1Label,'String','Target Signal')

                        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                        %%% CORRECTION FACTORS TO USE %%%
                        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                S2_ToneAmp = str2num(get(ul0Amp ,'string'));
                                S2_NoiseAmp = str2num(get(ul1Amp ,'string'));
                                S2_WAVGain = str2num(get(ul2Gain ,'string'));

                        S1_ROW = get(ul0Freq ,'value'); %Frequency Row Value
                        S1_ToneFreq=Freq(1,S1_ROW); %Frequency
%                         AudL = Resultant(S1_ROW,3);  %Audiometer L-Sensitivity
%                         AudR = Resultant(S1_ROW,5);  %Audiometer R-Sensitivity

                    AudL1 = Resultant(S1_ROW,3);     %Audiometer L-Sensitivity
                    AudR1 = Resultant(S1_ROW,5);     %Audiometer R-Sensitivity

                    if get(A1,'value')==1; AudL = AudL1; AudR = AudR1; end    % A1 - Use both Left and Rigth Thresholds
                    if get(A2,'value')==1; AudL = AudL1; AudR = AudL1; end    % A2 - Use Left Threshold Only
                    if get(A3,'value')==1; AudL = AudR1; AudR = AudR1; end    % A3 - Use Right Threshold Only
                    if get(A4,'value')==1; AudL = (AudL1+AudR1)/2; AudR = (AudL1+AudR1)/2; end    % A4 - Use AVG of Left and Right Thresholds

                        H1_L = Headphones_A(S1_ROW,2);  %Participant Headphones L-Adjustment
                        H1_R = Headphones_A(S1_ROW,3);  %Participant Headphones R-Adjustment
                        H1_B = Headphones_A(S1_ROW,4);  %Participant Headphones B-Adjustment

                        H2_L = Headphones_B(S1_ROW,2);  %HATS Headphones L-Adjustment
                        H2_R = Headphones_B(S1_ROW,3);  %HATS Headphones R-Adjustment
                        H2_B = Headphones_B(S1_ROW,4);  %HATS Headphones B-Adjustment

                        %%% Setting Correction for Reference Used
                            %%% Linear Reference
                                if Linear==1;
                                    %fprintf('\nLinear dB(Z)\n')
                                    H1Corr_S1L = H1_L; %S1L + H1L (Singal 1 on Headphones 1)
                                    H1Corr_S1R = H1_R; %S1R + H1R (Singal 1 on Headphones 1)
                                    H1Corr_S2L = H1_L; %S2L + H1L (Singal 2 on Headphones 1)
                                    H1Corr_S2R = H1_R; %S2R + H1R (Singal 2 on Headphones 1)

                                    H2Corr_S1L = H2_L; %S1L + H2L (Singal 1 on Headphones 2)
                                    H2Corr_S1R = H2_R; %S1R + H2R (Singal 1 on Headphones 2)
                                    H2Corr_S2L = H2_L; %S2L + H2L (Singal 2 on Headphones 2)
                                    H2Corr_S2R = H2_R; %S2R + H2R (Singal 2 on Headphones 2)
                                end

                            %%% Sensation Level Reference
                                if Audiogram==1;
                                    %fprintf('\nSensation Level dB(SL)\n')
                                    H1Corr_S1L = H1_L + AudL; %S1L + H1L + AudCorrL (Singal 1 on Headphones 1 as SL)
                                    H1Corr_S1R = H1_R + AudR; %S1R + H1R + AudCorrR (Singal 1 on Headphones 1 as SL)
                                    H1Corr_S2L = H1_L + AudL; %S2L + H1L + AudCorrL (Singal 2 on Headphones 1 as SL)
                                    H1Corr_S2R = H1_R + AudR; %S2R + H1R + AudCorrR (Singal 2 on Headphones 1 as SL)

                                    H2Corr_S1L = H2_L; %S1L + H2L + AudCorrL (Singal 1 on Headphones 2 as SL)
                                    H2Corr_S1R = H2_R; %S1R + H2R + AudCorrR (Singal 1 on Headphones 2 as SL)
                                    H2Corr_S2L = H2_L; %S2L + H2L + AudCorrL (Singal 2 on Headphones 2 as SL)
                                    H2Corr_S2R = H2_R; %S2R + H2R + AudCorrR (Singal 2 on Headphones 2 as SL)

                                end

             set(Sig1,'BackgroundColor',[0.9412    0.9412    0.9412])
             set(Sig2,'BackgroundColor',[0.5 0.5 0.5])

             %%%% PURE TONE
            if Sig2Type1==1;
```

```
% Collect User Settings Information
CurrentSettings= getappdata(0,'evalue');
Cur_Playback2 = CurrentSettings(5,4);
Cur_Pause2 = CurrentSettings(6,4);
Cur_Fade2 = CurrentSettings(7,4);
Cur_Phase2 = CurrentSettings(8,4);

% Determine which ear will be presented to
if ear2 == 'L'
    S2_ToneAmp = str2num(get(u10Amp ,'string'));
    S2_L_Monitor = S2_ToneAmp+H1Corr_S2L-H2Corr_S2L;
    Amplitude(1,1)=S2_ToneAmp+H1Corr_S2L;
    Amplitude(1,2)=S2_ToneAmp+H1Corr_S2R;

    set(Monitor_Label,'String','MONITOR (S2):');
    set(Monitor_Value,'String',S2_L_Monitor);

    if S2_L_Monitor>=ExpAppZeros(S2_PureToneFreq,13); set(StartPlayback,'Value',0); MaxSPLCheck; break; end

    [  t, left, right, fs ] = PlotTone_adj_v3 (S2_ToneFreq, Cur_Fade2, Cur_Playback2*1000, Amplitude, ear2, 1, Phase);
         player=audioplayer(left,fs);
%          fprintf('Left Player \n');
         play(player)

         ToneLength=(max(size(left))/fs); %gives the tone length in seconds
         PauseCount=0;

         PauseInterval=100;
         for PauseRun = 1:PauseInterval
           ExpAppZeros = getappdata(0,'evalue');
           PauseCheck = ExpAppZeros(28,4);

             if PauseCheck ==0
               pause(ToneLength/PauseInterval);         % (Length, sec.) / 1000
               PauseCount=PauseCount+1;        % 1-1000
             else
               Playtime=(ToneLength-(PauseCount/PauseInterval)*ToneLength)*1000;

               S2_ToneAmp = str2num(get(u10Amp ,'string'));
               S2_L_Monitor = S2_ToneAmp+H1Corr_S2L-H2Corr_S2L;
               set(Monitor_Value,'String',S2_L_Monitor);
               Amplitude(1,1)=S2_ToneAmp+H1Corr_S2L;
               Amplitude(1,2)=S2_ToneAmp+H1Corr_S2R;

               [  t, left, right, fs ] = PlotTone_adj_v3 (S2_ToneFreq, 10, Playtime, Amplitude, ear2, 1, Phase);
               player=audioplayer(left,fs);
               play(player)

               ExpAppZeros(28,4)=0;
               setappdata(0,'evalue',ExpAppZeros);
               PauseCount=PauseCount+1;
             end
         end

end

if ear2 == 'R'
    S2_R_Monitor = S2_ToneAmp+H1Corr_S2R-H2Corr_S2R;
    Amplitude(1,1)=S2_ToneAmp+H1Corr_S2L;
    Amplitude(1,2)=S2_ToneAmp+H1Corr_S2R;

    set(Monitor_Label,'String','MONITOR (S2):');
    set(Monitor_Value,'String',S2_R_Monitor);

    if S2_R_Monitor>=ExpAppZeros(S2_PureToneFreq,13); set(StartPlayback,'Value',0); MaxSPLCheck; break; end

    [  t, left, right, fs ] = PlotTone_adj_v3 (S2_ToneFreq, Cur_Fade2, Cur_Playback2*1000, Amplitude, ear2, 1, Phase);
         player=audioplayer(right,fs);
%          fprintf('Right Player \n');
         play(player)

         ToneLength=(max(size(right))/fs); %gives the tone length in seconds
         PauseCount=0;

         PauseInterval=10;
         for PauseRun = 1:PauseInterval
           ExpAppZeros = getappdata(0,'evalue');
           PauseCheck = ExpAppZeros(28,4);

             if PauseCheck ==0
               pause(ToneLength/PauseInterval);         % (Length, sec.) / 1000
               PauseCount=PauseCount+1;        % 1-1000
             else
               Playtime=(ToneLength-(PauseCount/PauseInterval)*ToneLength)*1000;

               S2_ToneAmp = str2num(get(u10Amp ,'string'));
               S2_L_Monitor = S2_ToneAmp+H1Corr_S2L-H2Corr_S2L;
               set(Monitor_Value,'String',S2_L_Monitor);
               Amplitude(1,1)=S2_ToneAmp+H1Corr_S2L;
               Amplitude(1,2)=S2_ToneAmp+H1Corr_S2R;

               [  t, left, right, fs ] = PlotTone_adj_v3 (S2_ToneFreq, 10, Playtime, Amplitude, ear2, 1, Phase);
               player=audioplayer(right,fs);
               play(player)

               ExpAppZeros(28,4)=0;
               setappdata(0,'evalue',ExpAppZeros);
               PauseCount=PauseCount+1;
             end
         end


end

if ear2 == 'B'
    S2_L_Monitor = S2_ToneAmp+H1Corr_S2L-H2Corr_S2L;
    S2_R_Monitor = S2_ToneAmp+H1Corr_S2R-H2Corr_S2R;
    Amplitude(1,1)=S2_ToneAmp+H1Corr_S2L;
    Amplitude(1,2)=S2_ToneAmp+H1Corr_S2R;

    set(Monitor_Label,'String','Bin. Mon. (S2-R):');
    set(Monitor_Value,'String',S2_R_Monitor);
```

```
                if S2_L_Monitor>=ExpAppZeros(S2_PureToneFreq,13); set(StartPlayback,'Value',0); MaxSPLCheck; break;
                elseif S2_R_Monitor>=ExpAppZeros(S2_PureToneFreq,13); set(StartPlayback,'Value',0); MaxSPLCheck; break; end


                [  t, left, right, fs ] = PlotTone_adj_v3 (S2_ToneFreq, Cur_Fade2, Cur_Playback2*1000, Amplitude, 'B', 1, Phase);
                    player=audioplayer([left,right],fs);
%                     fprintf('Binaural Player \n');
                    play(player)

                    ToneLength=(max(size(left))/fs); %gives the tone length in seconds
                    PauseCount=0;

                    PauseInterval=10;
                    for PauseRun = 1:PauseInterval
                      ExpAppZeros = getappdata(0,'evalue');
                      PauseCheck = ExpAppZeros(28,4);

                        if PauseCheck ==0
                          pause(ToneLength/PauseInterval);         % (Length, sec.) / 1000
                          PauseCount=PauseCount+1;        % 1-1000
                        else
                          Playtime=(ToneLength-(PauseCount/PauseInterval)*ToneLength)*1000;

                          S2_ToneAmp = str2num(get(u10Amp ,'string'));
                          S2_L_Monitor = S2_ToneAmp+H1Corr_S2L-H2Corr_S2L;
                          set(Monitor_Value,'String',S2_L_Monitor);
                          Amplitude(1,1)=S2_ToneAmp+H1Corr_S2L;
                          Amplitude(1,2)=S2_ToneAmp+H1Corr_S2R;

                          [  t, left, right, fs ] = PlotTone_adj_v3 (S2_ToneFreq, 10, Playtime, Amplitude, 'B', 1, Phase);
                          player=audioplayer([left,right],fs);
                          play(player)

                          ExpAppZeros(28,4)=0;
                          setappdata(0,'evalue',ExpAppZeros);
                          PauseCount=PauseCount+1;
                        end
                    end

            end


        end

        %%% NOISE SAMPLE
        if Sig2Type2==1;

            S2_NoiseType = get(u11Type ,'value');
            S2_NoiseAmp = str2num(get(u11Amp ,'string'));
            S2_NoiseLFreq = str2num(get(u11LowFreq ,'string'));
            S2_NoiseUFreq = str2num(get(u11UpFreq ,'string'));

            H2_L = Headphones_B(S1_ROW,2);  %HATS Headphones L-Adjustment
            H2_R = Headphones_B(S1_ROW,3);  %HATS Headphones R-Adjustment
            H2_B = Headphones_B(S1_ROW,4);  %HATS Headphones B-Adjustment

            % Collect User Settings Information
            CurrentSettings= getappdata(0,'evalue');
            Cur_Playback2 = CurrentSettings(5,4);
            Cur_Pause2 = CurrentSettings(6,4);
            Cur_Fade2 = CurrentSettings(7,4);
            Cur_Phase2 = CurrentSettings(8,4);

                        % No filter available for 20 kHz
                FreqN = [20, 25, 31.5, 40, 50, 63, 80, 100, 125, 160, 200, 250, 315, 400,...
                        500, 630, 800, 1000, 1250, 1600, 2000, 2500, 3150, 4000, 5000,...
                        6300, 8000, 10000, 12500, 16000].';

                dBA = [-50.5, -44.7, -39.4, -34.6, -30.2, -26.2, -22.5, -19.1, -16.1, -13.4, -10.9, -8.6, -6.6, -4.8,...
                         -3.2, -1.9, -0.8, 0.0, 0.6, 1.0, 1.2, 1.3, 1.2, 1.0, 0.5,...
                         -0.1, -1.1, -2.5, -4.3, -6.6].';

                A_Weight = [FreqN; dBA];    % For Grey Noise Presentation
                BCorr = zeros(length(dBA),1);

                if S2_NoiseType == 6;
                    BCorr = -1 * dBA;
                end

            Fs = 44100;

            LowFreqRow = get(u11LowFreq ,'value'); % 20 = 1,  12500 = 29, 20000 = 31
            UppFreqRow = get(u11UpFreq  ,'value');

            LpCheck = 0;

            if S2_NoiseType == 1; % 'Violet (+6 dB/oct.)',      +2 dB/3rd.
                ThrdAdd =  2;
                AmpShft = 0;
                Lp=S2_NoiseAmp;
                LpTot=0;
                for Lpi = 1:31
                    if Lpi >= LowFreqRow
                       if Lpi <= UppFreqRow
                           LpTot=10*log10(10^((S2_NoiseAmp+((Lpi-1)*(2)))/10) + 10^(LpTot/10));
                           LpCheck(:,Lpi) = (S2_NoiseAmp+((Lpi-1)*(2)));
                       end
                    end
                end
                AmpShft = -1*(LpTot - S2_NoiseAmp);
            end

            if S2_NoiseType == 2;  % 'Blue (+3 dB/oct.)',        +1 dB/3rd.
                ThrdAdd =  1;
                AmpShft = 0;
                Lp=S2_NoiseAmp;
                LpTot=0;
                for Lpi = 1:31
                    if Lpi >= LowFreqRow
                       if Lpi <= UppFreqRow
                           LpTot=10*log10(10^((S2_NoiseAmp+((Lpi-1)*(1)))/10) + 10^(LpTot/10));
                           LpCheck(:,Lpi) = (S2_NoiseAmp+((Lpi-1)*(1)));
                       end
                    end
```

```
            end
            AmpShft = -1*(LpTot - S2_NoiseAmp);
        end

        if S2_NoiseType == 3;  % 'White (+0 dB/oct.)',       +0 dB/3rd.
            ThrdAdd =  0;
            AmpShft = 0;
            Lp=S2_NoiseAmp;
            LpTot=0;
            for Lpi = 1:31
                if Lpi >= LowFreqRow
                   if Lpi <= UppFreqRow
                       LpTot=10*log10(10^((S2_NoiseAmp+((Lpi-1)*(0)))/10) + 10^(LpTot/10));
                       LpCheck(:,Lpi) = (S2_NoiseAmp+((Lpi-1)*(0)));
                   end
                end
            end
            AmpShft = -1*(LpTot - S2_NoiseAmp);
        end

        if S2_NoiseType == 4;  % 'Pink (-3 dB/oct.)',        -1 dB/3rd.
            ThrdAdd = -1;
            AmpShft = 0;
            Lp=S2_NoiseAmp;
            LpTot=0;
            for Lpi = 1:31
                if Lpi >= LowFreqRow
                   if Lpi <= UppFreqRow
                       LpTot=10*log10(10^((S2_NoiseAmp+((Lpi-1)*(-1)))/10) + 10^(LpTot/10));
                       LpCheck(:,Lpi) = (S2_NoiseAmp+((Lpi-1)*(-1)));
                   end
                end
            end
            AmpShft = -1*(LpTot - S2_NoiseAmp);
        end

        if S2_NoiseType == 5;  % 'Brownian (-6 dB/oct.)',    -2 dB/3rd.
            ThrdAdd = -2;
            AmpShft = 0;
            Lp=S2_NoiseAmp;
            LpTot=0;
            for Lpi = 1:31
                if Lpi >= LowFreqRow
                   if Lpi <= UppFreqRow
                       LpTot=10*log10(10^((S2_NoiseAmp+((Lpi-1)*(-2)))/10) + 10^(LpTot/10));
                       LpCheck(:,Lpi) = (S2_NoiseAmp+((Lpi-1)*(-2)));
                   end
                end
            end
            AmpShft = -1*(LpTot - S2_NoiseAmp);
        end

        if S2_NoiseType == 6;  % 'Grey (Inv. A-Weight)'
            ThrdAdd =  0;
            AmpShft = 0;
            Lp=S2_NoiseAmp;
            LpTot=0;
            for Lpi = 1:31
                if Lpi >= LowFreqRow
                   if Lpi <= UppFreqRow
                       LpTot=10*log10(10^((S2_NoiseAmp+(BCorr(Lpi,1)))/10) + 10^(LpTot/10));
                       LpCheck(:,Lpi) = (S2_NoiseAmp+(BCorr(Lpi,1)));
                   end
                end
            end
            AmpShft = -1*(LpTot - S2_NoiseAmp);
        end

        H_L(:,1) = Headphones_B(:,2);
        H_R(:,1) = Headphones_B(:,3);  %Participant Headphones R-Adjustment

        [CL,CR] = Apply_filter(S2_NoiseAmp, ThrdAdd, AmpShft, H_L, H_R,LowFreqRow,UppFreqRow,S2_NoiseType);


        x = wgn(Cur_Playback2*Fs, 1, 0);
        yL = zeros(length(x),30);
        yR = zeros(length(x),30);

        for filt = 1:30
            fi=filt;
            yL(:,fi) = filter((CL(fi))*b_coef{fi},a_coef{fi},x);
            yR(:,fi) = filter((CR(fi))*b_coef{fi},a_coef{fi},x);
        end

        yLtot = yL(:,1) + yL(:,2) + yL(:,3) + yL(:,4) + yL(:,5) + yL(:,6) + yL(:,7) + yL(:,8) + yL(:,9) + yL(:,10) + yL(:,11) + yL(:,12) +
yL(:,13) + yL(:,14) + yL(:,15) + yL(:,16) + yL(:,17) + yL(:,18) + yL(:,19) + yL(:,20) + yL(:,21) + yL(:,22) + yL(:,23) + yL(:,24) + yL(:,25) +
yL(:,26) + yL(:,27) + yL(:,28) + yL(:,29) + yL(:,30);
        yRtot = yR(:,1) + yR(:,2) + yR(:,3) + yR(:,4) + yR(:,5) + yR(:,6) + yR(:,7) + yR(:,8) + yR(:,9) + yR(:,10) + yR(:,11) + yR(:,12) +
yR(:,13) + yR(:,14) + yR(:,15) + yR(:,16) + yR(:,17) + yR(:,18) + yR(:,19) + yR(:,20) + yR(:,21) + yR(:,22) + yR(:,23) + yR(:,24) + yR(:,25) +
yR(:,26) + yR(:,27) + yR(:,28) + yR(:,29) + yR(:,30);

        fade_window1 = @(N)( hanning(N).^2 );
        yLtot = fade( yLtot, Fs, 200, fade_window1);
        yRtot = fade( yRtot, Fs, 200, fade_window1);

        Y = [yLtot yRtot];
        y0 = zeros(size(yLtot));

        if ear2 == 'L'; Y = [yLtot y0]; end
        if ear2 == 'R'; Y = [y0 yRtot]; end
        if ear2 == 'B'; Y = [yLtot yRtot]; end

        player=audioplayer(Y, Fs);
            play(player)

                ToneLength=(max(size(Y(:,1)))/Fs); %gives the tone length in seconds
                PauseCount=0;

                PauseInterval=10;
                for PauseRun = 1:PauseInterval
                  ExpAppZeros = getappdata(0,'evalue');
                  PauseCheck = ExpAppZeros(28,4);

                    if PauseCheck ==0
```

```
                        pause(ToneLength/PauseInterval);         % (Length, sec.) / 1000
%                         PauseCount=PauseCount+1;        % 1-1000
                      else
                        Playtime=(ToneLength-(PauseCount/PauseInterval)*ToneLength)*1000;

                            S2_NoiseAmp = str2num(get(u11Amp ,'string'));

                            [CL,CR] = Apply_filter(S2_NoiseAmp, ThrdAdd, AmpShft, H_L, H_R,LowFreqRow,UppFreqRow,S2_NoiseType);

                            x = wgn(Cur_Playback2*Fs, 1, 0);
                            yL = zeros(length(x),30);
                            yR = zeros(length(x),30);

                            for filt = 1:30
                                fi=filt;
                                yL(:,fi) = filter((CL(fi))*b_coef{fi},a_coef{fi},x);
                                yR(:,fi) = filter((CR(fi))*b_coef{fi},a_coef{fi},x);
                            end

                            yLtot = yL(:,1) + yL(:,2) + yL(:,3) + yL(:,4) + yL(:,5) + yL(:,6) + yL(:,7) + yL(:,8) + yL(:,9) + yL(:,10) +
yL(:,11) + yL(:,12) + yL(:,13) + yL(:,14) + yL(:,15) + yL(:,16) + yL(:,17) + yL(:,18) + yL(:,19) + yL(:,20) + yL(:,21) + yL(:,22) + yL(:,23) +
yL(:,24) + yL(:,25) + yL(:,26) + yL(:,27) + yL(:,28) + yL(:,29) + yL(:,30);
                            yRtot = yR(:,1) + yR(:,2) + yR(:,3) + yR(:,4) + yR(:,5) + yR(:,6) + yR(:,7) + yR(:,8) + yR(:,9) + yR(:,10) +
yR(:,11) + yR(:,12) + yR(:,13) + yR(:,14) + yR(:,15) + yR(:,16) + yR(:,17) + yR(:,18) + yR(:,19) + yR(:,20) + yR(:,21) + yR(:,22) + yR(:,23) +
yR(:,24) + yR(:,25) + yR(:,26) + yR(:,27) + yR(:,28) + yR(:,29) + yR(:,30);

                            fade_window1 = @(N)( hanning(N).^2 );
                            yLtot = fade( yLtot, Fs, 200, fade_window1);
                            yRtot = fade( yRtot, Fs, 200, fade_window1);

                            Y = [yLtot yRtot];
                            y0 = zeros(size(yLtot));

                            if ear2 == 'L'; Y = [yLtot y0]; end
                            if ear2 == 'R'; Y = [y0 yRtot]; end
                            if ear2 == 'B'; Y = [yLtot yRtot]; end

                        player=audioplayer(Y, Fs);
                        play(player)

                        ExpAppZeros(28,4)=0;
                        setappdata(0,'evalue',ExpAppZeros);
                        PauseCount=PauseCount+1;
                      end
                  end

        end

        %%% WAV FILE
        if Sig2Type3==1;
            % Collect User Settings Information
            CurrentSettings= getappdata(0,'evalue');
            Cur_Playback2 = CurrentSettings(5,4);
            Cur_Pause2 = CurrentSettings(6,4);
            Cur_Fade2 = CurrentSettings(7,4);
            Cur_Phase2 = CurrentSettings(8,4);
                S2_WAVGain = str2num(get(u12Gain ,'string'));
                EQ_Check = get(WAV2_EQ ,'value');

                Lock_Check = get(WAV2_Lock ,'value');

                % Check Number of Channels
                InfoS2=size(S2y);
                ChannelsS2=InfoS2(1,2);

                S2yTemp = zeros(size(S2y));

                if ChannelsS2 == 2;         % Stereo File

                    AvgLnR2 = get(WAV2_LnR,'value');

                    if ear2 == 'L';
                        S2yTemp(:,1) = (S2_WAVGain * Factor) * S2y(:,1);
                        % Option 2: Combine Left and Right onto Left
                        if AvgLnR2 == 1
                            S2yTemp2ch = (S2_WAVGain * Factor) * S2y;
                            S2yTemp(:,1) = (S2yTemp2ch(:,1) + S2yTemp2ch(:,2)) / 2;
                        end
                    end

                    if ear2 == 'R';
                        S2yTemp(:,2) = (S2_WAVGain * Factor) * S2y(:,2);
                        % Option 2: Combine Left and Right onto Left
                        if AvgLnR2 == 1
                            S2yTemp2ch = (S2_WAVGain * Factor) * S2y;
                            S2yTemp(:,2) = (S2yTemp2ch(:,1) + S2yTemp2ch(:,2)) / 2;
                        end
                    end

                    if ear2 == 'B';
                        S2yTemp = (S2_WAVGain * Factor) * S2y;

                        if AvgLnR2 == 1
                            S2yTemp2ch = (S2_WAVGain * Factor) * S2y;
                            S2yTemp(:,1) = (S2yTemp2ch(:,1) + S2yTemp2ch(:,2)) / 2;
                            S2yTemp(:,2) = (S2yTemp2ch(:,1) + S2yTemp2ch(:,2)) / 2;
                        end
                    end

                else                        % Monaural File (Duplicate ch.s)

                    S2yTemp = [zeros(size(S2y)), zeros(size(S2y))];

                    if ear2 == 'L';
                        S2yTemp(:,1) = (S2_WAVGain * Factor) * S2y(:,1);
                    end

                    if ear2 == 'R';
                        S2yTemp(:,2) = (S2_WAVGain * Factor) * S2y(:,1);
                    end

                    if ear2 == 'B';
                        S2yTempMon = (S2_WAVGain * Factor) * S2y;
                        S2yTemp = [S2yTempMon, S2yTempMon];
```

```
    end

end


if EQ_Check == 1;
    H_L(:,1) = Headphones_B(:,2);
    H_R(:,1) = Headphones_B(:,3);  %Participant Headphones R-Adjustment

    CL = exp((H_L + S2_WAVGain * Factor)/8.9141);
    CR = exp((H_R + S2_WAVGain * Factor)/8.9141);

    yL = zeros(length(S2yTemp(:,1)),30);
    yR = zeros(length(S2yTemp(:,2)),30);

    for filt = 1:30
        fi=filt;
        yL(:,fi) = filter((CL(fi))*b_coef{fi},a_coef{fi},S2yTemp(:,1));
        yR(:,fi) = filter((CR(fi))*b_coef{fi},a_coef{fi},S2yTemp(:,2));
    end

    yLtot = yL(:,1) + yL(:,2) + yL(:,3) + yL(:,4) + yL(:,5) + yL(:,6) + yL(:,7) + yL(:,8) + yL(:,9) + yL(:,10) + yL(:,11) +
yL(:,12) + yL(:,13) + yL(:,14) + yL(:,15) + yL(:,16) + yL(:,17) + yL(:,18) + yL(:,19) + yL(:,20) + yL(:,21) + yL(:,22) + yL(:,23) + yL(:,24) +
yL(:,25) + yL(:,26) + yL(:,27) + yL(:,28) + yL(:,29) + yL(:,30);
    yRtot = yR(:,1) + yR(:,2) + yR(:,3) + yR(:,4) + yR(:,5) + yR(:,6) + yR(:,7) + yR(:,8) + yR(:,9) + yR(:,10) + yR(:,11) +
yR(:,12) + yR(:,13) + yR(:,14) + yR(:,15) + yR(:,16) + yR(:,17) + yR(:,18) + yR(:,19) + yR(:,20) + yR(:,21) + yR(:,22) + yR(:,23) + yR(:,24) +
yR(:,25) + yR(:,26) + yR(:,27) + yR(:,28) + yR(:,29) + yR(:,30);


    Y = [yLtot yRtot];
    y0 = zeros(size(yLtot));

    if ear2 == 'L'; S2yTemp = [yLtot y0]; end
    if ear2 == 'R'; S2yTemp = [y0 yRtot]; end
    if ear2 == 'B'; S2yTemp = [yLtot yRtot]; end

end

    fade_window1 = @(N)( hanning(N).^2 );
    S2yTemp(:,1) = fade( S2yTemp(:,1), Fs, 100, fade_window1);
    S2yTemp(:,2) = fade( S2yTemp(:,2), Fs, 100, fade_window1);



player=audioplayer(S2yTemp,S2Fs,S2nbit);
play(player)


ToneLength=(max(size(S2yTemp(:,1)))/Fs); %gives the tone length in seconds
PauseCount=0;

PauseInterval=10;
for PauseRun = 1:PauseInterval
    ExpAppZeros = getappdata(0,'evalue');
    PauseCheck = ExpAppZeros(28,4);

    if PauseCheck ==0
        pause(ToneLength/PauseInterval);         % (Length, sec.) / 1000
    else
        S2_WAVGain = str2num(get(u12Gain ,'string'));

        if ChannelsS2 == 2;         % Stereo File

                AvgLnR2 = get(WAV2_LnR,'value');

                if ear2 == 'L';
                    S2yTemp(:,1) = (S2_WAVGain * Factor) * S2y(:,1);
                    % Option 2: Combine Left and Right onto Left
                    if AvgLnR2 == 1
                        S2yTemp2ch = (S2_WAVGain * Factor) * S2y;
                        S2yTemp(:,1) = (S2yTemp2ch(:,1) + S2yTemp2ch(:,2)) / 2;
                    end
                end

                if ear2 == 'R';
                    S2yTemp(:,2) = (S2_WAVGain * Factor) * S2y(:,2);
                    % Option 2: Combine Left and Right onto Left
                    if AvgLnR2 == 1
                        S2yTemp2ch = (S2_WAVGain * Factor) * S2y;
                        S2yTemp(:,2) = (S2yTemp2ch(:,1) + S2yTemp2ch(:,2)) / 2;
                    end
                end

                if ear2 == 'B';
                    S2yTemp = (S2_WAVGain * Factor) * S2y;

                    % Option 2: Average Left and Right
                    %   Present to both.
                    if AvgLnR2 == 1
                        S2yTemp2ch = (S2_WAVGain * Factor) * S2y;
                        S2yTemp(:,1) = (S2yTemp2ch(:,1) + S2yTemp2ch(:,2)) / 2;
                        S2yTemp(:,2) = (S2yTemp2ch(:,1) + S2yTemp2ch(:,2)) / 2;
                    end
                end

            else                     % Monaural File (Duplicate ch.s)

                S2yTemp = [zeros(size(S2y)), zeros(size(S2y))];

                if ear2 == 'L';
                    S2yTemp(:,1) = (S2_WAVGain * Factor) * S2y(:,1);
                end

                if ear2 == 'R';
                    S2yTemp(:,2) = (S2_WAVGain * Factor) * S2y(:,1);
                end

                if ear2 == 'B';
                    S2yTempMon = (S2_WAVGain * Factor) * S2y;
                    S2yTemp = [S2yTempMon, S2yTempMon];
                end

        end
```

```
                            if EQ_Check == 1;
                                H_L(:,1) = Headphones_B(:,2);
                                H_R(:,1) = Headphones_B(:,3);  %Participant Headphones R-Adjustment

                                CL = exp((H_L + S2_WAVGain * Factor)/8.9141);
                                CR = exp((H_R + S2_WAVGain * Factor)/8.9141);

                                yL = zeros(length(S2yTemp(:,1)),30);
                                yR = zeros(length(S2yTemp(:,2)),30);

                                for filt = 1:30
                                    fi=filt;
                                    yL(:,fi) = filter((CL(fi))*b_coef{fi},a_coef{fi},S2yTemp(:,1));
                                    yR(:,fi) = filter((CR(fi))*b_coef{fi},a_coef{fi},S2yTemp(:,2));
                                end

                                yLtot = yL(:,1) + yL(:,2) + yL(:,3) + yL(:,4) + yL(:,5) + yL(:,6) + yL(:,7) + yL(:,8) + yL(:,9) + yL(:,10) +
yL(:,11) + yL(:,12) + yL(:,13) + yL(:,14) + yL(:,15) + yL(:,16) + yL(:,17) + yL(:,18) + yL(:,19) + yL(:,20) + yL(:,21) + yL(:,22) + yL(:,23) +
yL(:,24) + yL(:,25) + yL(:,26) + yL(:,27) + yL(:,28) + yL(:,29) + yL(:,30);
                                yRtot = yR(:,1) + yR(:,2) + yR(:,3) + yR(:,4) + yR(:,5) + yR(:,6) + yR(:,7) + yR(:,8) + yR(:,9) + yR(:,10) +
yR(:,11) + yR(:,12) + yR(:,13) + yR(:,14) + yR(:,15) + yR(:,16) + yR(:,17) + yR(:,18) + yR(:,19) + yR(:,20) + yR(:,21) + yR(:,22) + yR(:,23) +
yR(:,24) + yR(:,25) + yR(:,26) + yR(:,27) + yR(:,28) + yR(:,29) + yR(:,30);


                                Y = [yLtot yRtot];
                                y0 = zeros(size(yLtot));

                                if ear2 == 'L'; S2yTemp = [yLtot y0]; end
                                if ear2 == 'R'; S2yTemp = [y0 yRtot]; end
                                if ear2 == 'B'; S2yTemp = [yLtot yRtot]; end

                                fade_window1 = @(N)( hanning(N).^2 );
                                S2yTemp(:,1) = fade( S2yTemp(:,1), Fs, 100, fade_window1);
                                S2yTemp(:,2) = fade( S2yTemp(:,2), Fs, 100, fade_window1);
                            end


                        player=audioplayer(S2yTemp,S2Fs,S2nbit);
                        play(player)

                        ExpAppZeros(28,4)=0;
                        setappdata(0,'evalue',ExpAppZeros);
                        PauseCount=PauseCount+1;
                    end
                end
        end

        %%%%%%% PAUSE 2
        set(Sig2,'BackgroundColor',[0.9412    0.9412    0.9412])
        pause(Cur_Pause2)

        PlayCount=PlayCount+1;
    end

    set(StartPlayback,'string', 'START')
    set(StartPlayback,'BackgroundColor',[0 0.8 0])
    set(SignalFeedback,'Color',Colour0)

end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% %%%%%%%%%%%%%%%%%%    End Code                        %%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Plotting_Function
Headphone_Cal_Function_1
Headphone_Cal_Function_2

            function [CL,CR] = Apply_filter(S2_NoiseAmp, ThrdAdd, AmpShft, H_L, H_R,LowFreqRow,UppFreqRow,S2_NoiseType)
            CL = 0; CR = 0;

                % No filter available for 20 kHz
                FreqN = [20, 25, 31.5, 40, 50, 63, 80, 100, 125, 160, 200, 250, 315, 400,...
                        500, 630, 800, 1000, 1250, 1600, 2000, 2500, 3150, 4000, 5000,...
                        6300, 8000, 10000, 12500, 16000].';

                dBA = [-50.5, -44.7, -39.4, -34.6, -30.2, -26.2, -22.5, -19.1, -16.1, -13.4, -10.9, -8.6, -6.6, -4.8,...
                         -3.2, -1.9, -0.8, 0.0, 0.6, 1.0, 1.2, 1.3, 1.2, 1.0, 0.5,...
                         -0.1, -1.1, -2.5, -4.3, -6.6].';

                A_Weight = [FreqN; dBA];
                BCorr = zeros(length(dBA),1);

                if S2_NoiseType == 6;
                    BCorr = -1 * dBA;
                end

                for Cset = 1:34
                    CL(Cset,1) = S2_NoiseAmp;

                        LNoiseAdj = [-79.834, -76.346, -69.480, -70.512, -67.196, -70.820, -63.132, -65.113, -65.760, -65.984, -68.229, -
69.331, -69.435, -71.064, -71.296, -72.130, -73.335, -74.992, -75.105, -75.950, -78.235, -79.027, -82.302, -78.128, -80.381, -93.336, -83.038,
-92.668, -93.978, -88.435];

                        CLtemp(1,1)  = exp((S2_NoiseAmp + BCorr(1,1) + ThrdAdd* 1 + AmpShft +  H_L( 2,1)+LNoiseAdj(1,1))/8.9141);
                        CLtemp(2,1)  = exp((S2_NoiseAmp + BCorr(2,1) + ThrdAdd* 2 + AmpShft +  H_L( 3,1)+LNoiseAdj(1,2))/9.6987);
                        CLtemp(3,1)  = exp((S2_NoiseAmp + BCorr(3,1) + ThrdAdd* 3 + AmpShft +  H_L( 4,1)+LNoiseAdj(1,3))/9.0276);
                        CLtemp(4,1)  = exp((S2_NoiseAmp + BCorr(4,1) + ThrdAdd* 4 + AmpShft +  H_L( 5,1)+LNoiseAdj(1,4))/9.0438);
                        CLtemp(5,1)  = exp((S2_NoiseAmp + BCorr(5,1) + ThrdAdd* 5 + AmpShft +  H_L( 6,1)+LNoiseAdj(1,5))/9.3716);
                        CLtemp(6,1)  = exp((S2_NoiseAmp + BCorr(6,1) + ThrdAdd* 6 + AmpShft +  H_L( 7,1)+LNoiseAdj(1,6))/8.6569);
                        CLtemp(7,1)  = exp((S2_NoiseAmp + BCorr(7,1) + ThrdAdd* 7 + AmpShft +  H_L( 8,1)+LNoiseAdj(1,7))/8.4304);
                        CLtemp(8,1)  = exp((S2_NoiseAmp + BCorr(8,1) + ThrdAdd* 8 + AmpShft +  H_L( 9,1)+LNoiseAdj(1,8))/8.9141);
                        CLtemp(9,1)  = exp((S2_NoiseAmp + BCorr(9,1) + ThrdAdd* 9 + AmpShft +  H_L(10,1)+LNoiseAdj(1,9))/8.8249);
                        CLtemp(10,1) = exp((S2_NoiseAmp + BCorr(10,1) + ThrdAdd*10 + AmpShft +  H_L(11,1)+LNoiseAdj(1,10))/8.6526);
                        CLtemp(11,1) = exp((S2_NoiseAmp + BCorr(11,1) + ThrdAdd*11 + AmpShft +  H_L(12,1)+LNoiseAdj(1,11))/8.8063);
                        CLtemp(12,1) = exp((S2_NoiseAmp + BCorr(12,1) + ThrdAdd*12 + AmpShft +  H_L(13,1)+LNoiseAdj(1,12))/8.9410);
                        CLtemp(13,1) = exp((S2_NoiseAmp + BCorr(13,1) + ThrdAdd*13 + AmpShft +  H_L(14,1)+LNoiseAdj(1,13))/8.7501);
                        CLtemp(14,1) = exp((S2_NoiseAmp + BCorr(14,1) + ThrdAdd*14 + AmpShft +  H_L(15,1)+LNoiseAdj(1,14))/8.8295);
```

```
                        CLtemp(15,1) = exp((S2_NoiseAmp + BCorr(15,1) + ThrdAdd*15 + AmpShft +  H_L(16,1)+LNoiseAdj(1,15))/8.7235);
                        CLtemp(16,1) = exp((S2_NoiseAmp + BCorr(16,1) + ThrdAdd*16 + AmpShft +  H_L(17,1)+LNoiseAdj(1,16))/8.7006);
                        CLtemp(17,1) = exp((S2_NoiseAmp + BCorr(17,1) + ThrdAdd*17 + AmpShft +  H_L(18,1)+LNoiseAdj(1,17))/8.7220);
                        CLtemp(18,1) = exp((S2_NoiseAmp + BCorr(18,1) + ThrdAdd*18 + AmpShft +  H_L(19,1)+LNoiseAdj(1,18))/8.7227);
                        CLtemp(19,1) = exp((S2_NoiseAmp + BCorr(19,1) + ThrdAdd*19 + AmpShft +  H_L(20,1)+LNoiseAdj(1,19))/8.6702);
                        CLtemp(20,1) = exp((S2_NoiseAmp + BCorr(20,1) + ThrdAdd*20 + AmpShft +  H_L(21,1)+LNoiseAdj(1,20))/8.7238);
                        CLtemp(21,1) = exp((S2_NoiseAmp + BCorr(21,1) + ThrdAdd*21 + AmpShft +  H_L(22,1)+LNoiseAdj(1,21))/8.7045);
                        CLtemp(22,1) = exp((S2_NoiseAmp + BCorr(22,1) + ThrdAdd*22 + AmpShft +  H_L(23,1)+LNoiseAdj(1,22))/8.7093);
                        CLtemp(23,1) = exp((S2_NoiseAmp + BCorr(23,1) + ThrdAdd*23 + AmpShft +  H_L(24,1)+LNoiseAdj(1,23))/8.7068);
                        CLtemp(24,1) = exp((S2_NoiseAmp + BCorr(24,1) + ThrdAdd*24 + AmpShft +  H_L(25,1)+LNoiseAdj(1,24))/8.7715);
                        CLtemp(25,1) = exp((S2_NoiseAmp + BCorr(25,1) + ThrdAdd*25 + AmpShft +  H_L(26,1)+LNoiseAdj(1,25))/8.7100);
                        CLtemp(26,1) = exp((S2_NoiseAmp + BCorr(26,1) + ThrdAdd*26 + AmpShft +  H_L(27,1)+LNoiseAdj(1,26))/8.7832);
                        CLtemp(27,1) = exp((S2_NoiseAmp + BCorr(27,1) + ThrdAdd*27 + AmpShft +  H_L(28,1)+LNoiseAdj(1,27))/8.5797);
                        CLtemp(28,1) = exp((S2_NoiseAmp + BCorr(28,1) + ThrdAdd*28 + AmpShft +  H_L(29,1)+LNoiseAdj(1,28))/8.4993);
                        CLtemp(29,1) = exp((S2_NoiseAmp + BCorr(29,1) + ThrdAdd*29 + AmpShft +  H_L(30,1)+LNoiseAdj(1,29))/8.6223);
                        CLtemp(30,1) = exp((S2_NoiseAmp + BCorr(30,1) + ThrdAdd*30 + AmpShft +  H_L(31,1)+LNoiseAdj(1,30))/8.7175);

                        if Cset < LowFreqRow;
                            %CL(Cset,1) = exp((-50)/8.7175);
                            CL(Cset,1) = 0;
                        else
                            if Cset > UppFreqRow;
                                %CL(Cset,1) = exp((-50)/8.7175);
                                CL(Cset,1) = 0;
                            else
                                CL(Cset,1) = CLtemp(Cset,1);
                            end
                        end

                    CR(Cset,1) = S2_NoiseAmp;

                        RNoiseAdj = [-79.834, -76.346, -69.480, -70.512, -67.196, -70.820, -63.132, -65.113, -65.760, -65.984, -68.229, -
69.331, -69.435, -71.064, -71.296, -72.130, -73.335, -74.992, -75.105, -75.950, -78.235, -79.027, -82.302, -78.128, -80.381, -93.336, -89.038,
-90.668, -90.478, -84.935];

                        CRtemp(1,1)  = exp((S2_NoiseAmp + BCorr(1,1) + ThrdAdd* 1 + AmpShft +  H_R( 2,1)+RNoiseAdj(1,1))/8.9141);
                        CRtemp(2,1)  = exp((S2_NoiseAmp + BCorr(2,1) + ThrdAdd* 2 + AmpShft +  H_R( 3,1)+RNoiseAdj(1,2))/9.6987);
                        CRtemp(3,1)  = exp((S2_NoiseAmp + BCorr(3,1) + ThrdAdd* 3 + AmpShft +  H_R( 4,1)+RNoiseAdj(1,3))/9.0438);
                        CRtemp(4,1)  = exp((S2_NoiseAmp + BCorr(4,1) + ThrdAdd* 4 + AmpShft +  H_R( 5,1)+RNoiseAdj(1,4))/9.0438);
                        CRtemp(5,1)  = exp((S2_NoiseAmp + BCorr(5,1) + ThrdAdd* 5 + AmpShft +  H_R( 6,1)+RNoiseAdj(1,5))/9.3716);
                        CRtemp(6,1)  = exp((S2_NoiseAmp + BCorr(6,1) + ThrdAdd* 6 + AmpShft +  H_R( 7,1)+RNoiseAdj(1,6))/8.6569);
                        CRtemp(7,1)  = exp((S2_NoiseAmp + BCorr(7,1) + ThrdAdd* 7 + AmpShft +  H_R( 8,1)+RNoiseAdj(1,7))/8.4304);
                        CRtemp(8,1)  = exp((S2_NoiseAmp + BCorr(8,1) + ThrdAdd* 8 + AmpShft +  H_R( 9,1)+RNoiseAdj(1,8))/8.9141);
                        CRtemp(9,1)  = exp((S2_NoiseAmp + BCorr(9,1) + ThrdAdd* 9 + AmpShft +  H_R(10,1)+RNoiseAdj(1,9))/8.8249);
                        CRtemp(10,1) = exp((S2_NoiseAmp + BCorr(10,1) + ThrdAdd*10 + AmpShft +  H_R(11,1)+RNoiseAdj(1,10))/8.6526);
                        CRtemp(11,1) = exp((S2_NoiseAmp + BCorr(11,1) + ThrdAdd*11 + AmpShft +  H_R(12,1)+RNoiseAdj(1,11))/8.8063);
                        CRtemp(12,1) = exp((S2_NoiseAmp + BCorr(12,1) + ThrdAdd*12 + AmpShft +  H_R(13,1)+RNoiseAdj(1,12))/8.9410);
                        CRtemp(13,1) = exp((S2_NoiseAmp + BCorr(13,1) + ThrdAdd*13 + AmpShft +  H_R(14,1)+RNoiseAdj(1,13))/8.7501);
                        CRtemp(14,1) = exp((S2_NoiseAmp + BCorr(14,1) + ThrdAdd*14 + AmpShft +  H_R(15,1)+RNoiseAdj(1,14))/8.8295);
                        CRtemp(15,1) = exp((S2_NoiseAmp + BCorr(15,1) + ThrdAdd*15 + AmpShft +  H_R(16,1)+RNoiseAdj(1,15))/8.7235);
                        CRtemp(16,1) = exp((S2_NoiseAmp + BCorr(16,1) + ThrdAdd*16 + AmpShft +  H_R(17,1)+RNoiseAdj(1,16))/8.7006);
                        CRtemp(17,1) = exp((S2_NoiseAmp + BCorr(17,1) + ThrdAdd*17 + AmpShft +  H_R(18,1)+RNoiseAdj(1,17))/8.7220); %800
                        CRtemp(18,1) = exp((S2_NoiseAmp + BCorr(18,1) + ThrdAdd*18 + AmpShft +  H_R(19,1)+RNoiseAdj(1,18))/8.7227);
                        CRtemp(19,1) = exp((S2_NoiseAmp + BCorr(19,1) + ThrdAdd*19 + AmpShft +  H_R(20,1)+RNoiseAdj(1,19))/8.6702);
                        CRtemp(20,1) = exp((S2_NoiseAmp + BCorr(20,1) + ThrdAdd*20 + AmpShft +  H_R(21,1)+RNoiseAdj(1,20))/8.7238);
                        CRtemp(21,1) = exp((S2_NoiseAmp + BCorr(21,1) + ThrdAdd*21 + AmpShft +  H_R(22,1)+RNoiseAdj(1,21))/8.7045);
                        CRtemp(22,1) = exp((S2_NoiseAmp + BCorr(22,1) + ThrdAdd*22 + AmpShft +  H_R(23,1)+RNoiseAdj(1,22))/8.7093);
                        CRtemp(23,1) = exp((S2_NoiseAmp + BCorr(23,1) + ThrdAdd*23 + AmpShft +  H_R(24,1)+RNoiseAdj(1,23))/8.7068);
                        CRtemp(24,1) = exp((S2_NoiseAmp + BCorr(24,1) + ThrdAdd*24 + AmpShft +  H_R(25,1)+RNoiseAdj(1,24))/8.7715);
                        CRtemp(25,1) = exp((S2_NoiseAmp + BCorr(25,1) + ThrdAdd*25 + AmpShft +  H_R(26,1)+RNoiseAdj(1,25))/8.7100);
                        CRtemp(26,1) = exp((S2_NoiseAmp + BCorr(26,1) + ThrdAdd*26 + AmpShft +  H_R(27,1)+RNoiseAdj(1,26))/8.7832);
                        CRtemp(27,1) = exp((S2_NoiseAmp + BCorr(27,1) + ThrdAdd*27 + AmpShft +  H_R(28,1)+RNoiseAdj(1,27))/8.5797);
                        CRtemp(28,1) = exp((S2_NoiseAmp + BCorr(28,1) + ThrdAdd*28 + AmpShft +  H_R(29,1)+RNoiseAdj(1,28))/8.4993);
                        CRtemp(29,1) = exp((S2_NoiseAmp + BCorr(29,1) + ThrdAdd*29 + AmpShft +  H_R(30,1)+RNoiseAdj(1,29))/8.6223);
                        CRtemp(30,1) = exp((S2_NoiseAmp + BCorr(30,1) + ThrdAdd*30 + AmpShft +  H_R(31,1)+RNoiseAdj(1,30))/8.7175);

                        if Cset < LowFreqRow;
                            CR(Cset,1) = 0;
                        else
                            if Cset > UppFreqRow;
                                CR(Cset,1) = 0;
                            else
                                CR(Cset,1) = CRtemp(Cset,1);
                            end
                        end
                end
            end


%% Load Audiogram Data
    function Load_Audiogram(~,~,MainWindow)
        [AudFileName,PathName] = uigetfile('*.mat','Select mat file');
        if AudFileName==0, return, end

        Struct1 = load( fullfile(PathName,AudFileName) );   %# pass file path as string
        Structname = fieldnames(Struct1);                   %# typo?
        Loaded_Response=Struct1.(Structname{2,1});
        Resultant=Loaded_Response;
        OLDvNEW=size(Resultant); % 31 x 7, Resultant = [Freq L-Big L-Sm R-Big R-Sm L-DNH R-DNH]


        if OLDvNEW(1,1)==29         % OLD Audiograms stopped at 12,500 Hz
            i=1;

            for i=1:29
                ExpApp= getappdata(0,'evalue');
                ExpApp(i,5)=Resultant(i,3); % Column 5: Resultant L-Small
                ExpApp(i,6)=Resultant(i,5); % Column 6: Resultant R-Small
                ExpApp(30,5)=0.0;ExpApp(31,5)=0.0;
                ExpApp(30,6)=0;ExpApp(31,6)=0;
                setappdata(0,'evalue',ExpApp) % Placeholder for Important Values
            end

        else                        % NEW Audiograms go up to 20,000 Hz
            ExpApp= getappdata(0,'evalue');
            for LoadRes = 1:size(Resultant)
                ExpApp(LoadRes,5)=Resultant(LoadRes,3); % Column 5: Resultant L-Small
                ExpApp(LoadRes,6)=Resultant(LoadRes,5); % Column 6: Resultant R-Small
            end
            setappdata(0,'evalue',ExpApp) % Placeholder for Important Values
        end

        Plotting_Function
        Headphone_Cal_Function_1
```

```
        Headphone_Cal_Function_2
        refresh(MainWindow);

    end

%% Zoom Audiogram
    function Zoom_Audiogram(~,~,ExpApp)
        ZoomAudiogramGUI(ExpApp)
    end


%Creating Audiogram Button
AudLaunch = uicontrol('style', 'pushbutton','string', 'Measure','units',...
    'normalized','Visible','on','position', [0.175 0.938 0.08 0.035],...
    'FontSize', Font8,'callback', {@LoadAudiogram}); %#ok<NASGU>

    function LoadAudiogram (~,~)
        ExpAudiometerGUI
    end

%% Load Headphone Data 1
    function Load_Headphones1(~,~,MainWindow)
        [H1FileName,PathName] = uigetfile('*.mat','Select Calibration File For Participant Headphones');
        if H1FileName==0, return, end

        Struct2 = load( fullfile(PathName,H1FileName) );   %# pass file path as string
        Structname = fieldnames(Struct2);                  %# typo?
        Loaded_Response2=Struct2.(Structname{1,1});
        Headphones_A=Loaded_Response2;

        ExpApp= getappdata(0,'evalue');
        for LoadVal = 1:size(Headphones_A)
            ExpApp(LoadVal,7)=Headphones_A(LoadVal,2); % Column 7: Headphones 1 - Left
            ExpApp(LoadVal,8)=Headphones_A(LoadVal,3); % Column 8: Headphones 1 - Right
            ExpApp(LoadVal,15)=Headphones_A(LoadVal,4); % Column 15: Headphones 1 - Binaural
        end
        setappdata(0,'evalue',ExpApp) % Placeholder for Important Values



        Headphone_Cal_Function_1
        refresh(MainWindow);

    end
%% Load Headphone Data 2
    function Load_Headphones2(~,~,MainWindow)
        [H2FileName,PathName] = uigetfile('*.mat','Select Calibration File For Monitor Headphones');
        if H2FileName==0, return, end

        Struct3 = load( fullfile(PathName,H2FileName) );   %# pass file path as string
        Structname = fieldnames(Struct3);                  %# typo?
        Loaded_Response3=Struct3.(Structname{1,1});
        Headphones_B=Loaded_Response3;

        ExpApp= getappdata(0,'evalue');
        for LoadVal = 1:size(Headphones_B)
            ExpApp(LoadVal,9)=Headphones_B(LoadVal,2); % Column 9: Headphones 2 - Left
            ExpApp(LoadVal,10)=Headphones_B(LoadVal,3); % Column 10: Headphones 2 - Right
            ExpApp(LoadVal,16)=Headphones_B(LoadVal,4); % Column 16: Headphones 2 - Binaural
        end
        setappdata(0,'evalue',ExpApp) % Placeholder for Important Values

        Headphone_Cal_Function_2
        refresh(MainWindow);

    end
%% Launch Calibrator
LaunchCal = uicontrol('style', 'pushbutton','string', 'Launch',...
    'units','normalized','Visible','on',...
    'position', [0.76 0.725 0.1 0.04],...
    'FontSize', Font6, 'FontWeight',...
    'bold','callback', {@ExpCal_Callback});

    function ExpCal_Callback(~,~)
        ExpCalibration
    end
%% Comparison Plot
ComparePB = uicontrol('style', 'pushbutton','string', 'Compare',...
    'units','normalized','Visible','on',...
    'position', [0.87 0.725 0.1 0.04],...
    'FontSize', Font6, 'FontWeight',...
    'bold','callback', {@ComparePlot});

    function ComparePlot(~,~)
        CompPlot=figure('units', 'normalized',...
            'Position',[0.25,0.3,0.45,0.4], 'Visible', 'On',...
            'Name','Loudness Matching Experiment - Headphone Sensitivity Comparison','NumberTitle','Off');
        %Position - [Left Bottom Width Height]
        set(CompPlot,'MenuBar','none'); %Turn off Menu Bar
        set(CompPlot,'Toolbar','figure'); %Turn on the figure toolbar
        %Left on for Print and Save functions

        h2Toolbar = findall(CompPlot,'tag','FigureToolBar');
        % Removing buttons that are not needed.
        delete(findall(h2Toolbar,'tag','Standard.NewFigure'))
        delete(findall(h2Toolbar,'tag','Standard.FileOpen'))
        delete(findall(h2Toolbar,'tag','Plottools.PlottoolsOn'))
        delete(findall(h2Toolbar,'tag','Plottools.PlottoolsOff'))
        delete(findall(h2Toolbar,'tag','Annotation.InsertColorbar'))
        delete(findall(h2Toolbar,'tag','DataManager.Linking'))
        delete(findall(h2Toolbar,'tag','Standard.EditPlot'))
        delete(findall(h2Toolbar,'tag','Exploration.Rotate'))
        delete(findall(h2Toolbar,'tag','Exploration.Brushing'))
        delete(findall(h2Toolbar,'tag','Annotation.InsertLegend'))

        Freq=[20, 25, 31.5, 40, 50, 63, 80, 100, 125, 160, 200, 250,...
             315, 400, 500, 630, 800, 1000, 1250, 1600, 2000, 2500,...
             3150, 4000, 5000, 6300, 8000, 10000, 12500, 16000, 20000];

        ExpApp= getappdata(0,'evalue');
        for getdata = 1:31
            Headphones_A(getdata,2)=ExpApp(getdata,7); % Column 7: Headphones 1 - Left
            Headphones_A(getdata,3)=ExpApp(getdata,8); % Column 8: Headphones 1 - Right
            Headphones_A(getdata,4)=ExpApp(getdata,15); % Column 8: Headphones 1 - Binaural
```

```
            Headphones_B(getdata,2)=ExpApp(getdata,9); % Column 9: Headphones 2 - Left
            Headphones_B(getdata,3)=ExpApp(getdata,10); % Column 10: Headphones 2 - Right
            Headphones_B(getdata,4)=ExpApp(getdata,16); % Column 10: Headphones 2 - Binaural
        end


        CompPlotArea=subplot(1,1,1,'Parent',CompPlot);
        PlotAxesComp = semilogx(Freq.', Headphones_A(:,2),':b',...
                            Freq.', Headphones_A(:,3),':r',...
                            Freq.', Headphones_A(:,4),':m',...
                            Freq.', Headphones_B(:,2),':b',...
                            Freq.', Headphones_B(:,3),':r',...
                            Freq.', Headphones_B(:,4),':m',...
                            'DisplayName', 'SemiLog Plot');
                            %, 'Parent', handles.graph

                            set(PlotAxesComp(1),'MarkerSize',5,'LineWidth', 2.5)
                            set(PlotAxesComp(2),'MarkerSize',5,'LineWidth', 2.5)
                            set(PlotAxesComp(3),'MarkerSize',5,'LineWidth', 2.5)
                            set(PlotAxesComp(4),'MarkerSize',5,'LineWidth', 5)
                            set(PlotAxesComp(5),'MarkerSize',5,'LineWidth', 5)
                            set(PlotAxesComp(6),'MarkerSize',5,'LineWidth', 5)

%                             set(PlotAxesComp(1),'MarkerFaceColor','b')
%                             set(PlotAxesComp(2),'MarkerFaceColor','r')

        ax_PlotAxesComp = gca; %Collect info on current axes values to gca
        curtickComp = get(gca, 'XTick'); % Collecting current x-labels
        set(ax_PlotAxesComp, 'units', 'normalized',...
            'OuterPosition', [0 0.05 1 0.95],...
            'Position', [0.05 0.1 0.9 0.8],...
            'XLim', [10 20000],...
            'YLim', [-15 25],...
            'FontSize', Font12,...
            'XTickLabel', cellstr(num2str(curtickComp(:)))); % Setting the x-labels to non-scientific
        grid

        LE=0.05; %Left Edge
        LW=0.02; %Line Width
        WW=0.125; %Word Width
        G=(1/6)*(1-(4*LW+4*WW)); % Gap Between Words

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %%%% PARTICIPANT HEADPHONES  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        Legend_SYM1 = uicontrol('style', 'text','string',...
            '---','units','normalized',...
            'Visible','on','position', [G 0.935 LW 0.04],...
            'FontSize', Font10,...
            'BackgroundColor',[0.8 0.8 0.8],...
            'ForegroundColor',[0 0 1],'FontWeight','bold');
        Legend_1a = uicontrol('style', 'text','string',...
            'Participant - L','units','normalized',...
            'Visible','on','position', [(G+LW) 0.925 WW 0.05],...
            'FontSize', Font12, 'FontWeight','bold',...
            'BackgroundColor',[0.8 0.8 0.8]);

         Legend_SYM2 = uicontrol('style', 'text','string',...
            '---','units','normalized',...
            'Visible','on','position', [(2*G+LW+WW) 0.935 LW 0.04],...
            'FontSize', Font10,...
            'BackgroundColor',[0.8 0.8 0.8],...
            'ForegroundColor',[1 0 0],'FontWeight','bold');
        Legend_2a = uicontrol('style', 'text','string',...
            'Participant - R','units','normalized',...
            'Visible','on','position', [(2*G+2*LW+WW) 0.925 WW 0.05],...
            'FontSize', Font12, 'FontWeight','bold',...
            'BackgroundColor',[0.8 0.8 0.8]);

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %%%% HATS HEADPHONES  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        Legend_SYM3 = uicontrol('style', 'text','string',...
            '---','units','normalized',...
            'Visible','on','position', [(4*G+2*LW+2*WW) 0.9475 LW 0.04],...
            'FontSize', Font16,...
            'BackgroundColor',[0.8 0.8 0.8],...
            'ForegroundColor',[0 0 1],'FontWeight','bold');
        Legend_3a = uicontrol('style', 'text','string',...
            'HATS - L','units','normalized',...
            'Visible','on','position', [(4*G+3*LW+2*WW) 0.925 WW 0.05],...
            'FontSize', Font12, 'FontWeight','bold',...
            'BackgroundColor',[0.8 0.8 0.8]);

         Legend_SYM4 = uicontrol('style', 'text','string',...
            '---','units','normalized',...
            'Visible','on','position', [(5*G+3*LW+3*WW) 0.9475 LW 0.04],...
            'FontSize', Font16,...
            'BackgroundColor',[0.8 0.8 0.8],...
            'ForegroundColor',[1 0 0],'FontWeight','bold');
        Legend_4a = uicontrol('style', 'text','string',...
            'HATS - R','units','normalized',...
            'Visible','on','position', [(5*G+4*LW+3*WW) 0.925 WW 0.05],...
            'FontSize', Font12, 'FontWeight','bold',...
            'BackgroundColor',[0.8 0.8 0.8]);


    end
%% Initial Values for WAV Files
    S1y = 0; S1Fs = 0; S1nbit = 0; % Setting up initial values
    S2y = 0; S2Fs = 0; S2nbit = 0; % Setting up initial values

%% Load Signal 1 WAV File
    function [S1y, S1Fs,S1nbit] = Load_S1_WAV(~,~,MainWindow,u1Name,u1,WAV1_Name,WAV1_Loc,u12Name,u12,WAV2_Name,WAV2_Loc,WAV2_Lock)
        [S1FileName,S1PathName] = uigetfile('*.wav','Select audio file');
        if S1FileName==0, return, end

                    WavLockCheck = get (WAV2_Lock,'Value');

        [S1y, S1Fs,S1nbit] = wavread( fullfile(S1PathName,S1FileName) );   %# pass file path as string
        Info=size(S1y);
        Length=(Info(1,1))/S1Fs;
        Length = round((Length*1000))/1000; % Rounding Decimal Places
```

```
        Channels=Info(1,2);

        if S1FileName==0
            set(u1Name,'String','No WAV File Loaded')
        else
            set(u1Name,'String',S1FileName, 'FontWeight','bold')
            %set(u1Name,'BackgroundColor',[1 1 1])
            set(u1,'value',1)

               set(WAV1_Text1val,'String',num2str(S1Fs));
               set(WAV1_Text2val,'String',num2str(S1nbit));
               set(WAV1_Text3val,'String',Length);
               set(WAV1_Text4val,'String',Channels);

            ColourUiGroup1
        end

               set(WAV1_Name,'String',S1FileName);
               set(WAV1_Loc,'String',S1PathName);

        refresh(MainWindow);

        if WavLockCheck==1
            set(u12Name,'String',S1FileName, 'FontWeight','bold')
            %set(u12Name,'BackgroundColor',[1 1 1])
            set(u12,'value',1)

            S2y = S1y;
            S2Fs = S1Fs;
            S2nbit = S1nbit;

            set(WAV2_Name,'String',S1FileName);
            set(WAV2_Loc,'String',S1PathName);

            set(WAV2_Text1val,'String',num2str(S1Fs));
            set(WAV2_Text2val,'String',num2str(S1nbit));
            set(WAV2_Text3val,'String',Length);
            set(WAV2_Text4val,'String',Channels);

            refresh(MainWindow);
        end

    end
%% Load Signal 2 WAV File
    function Load_S2_WAV(~,~,MainWindow,u12Name,u12,WAV2_Name,WAV2_Loc,u1Name,u1,WAV1_Name,WAV1_Loc,WAV2_Lock)
        [S2FileName,S2PathName] = uigetfile('*.wav','Select audio file');
        if S2FileName==0, return, end

                    WavLockCheck = get (WAV2_Lock,'Value');

        [S2y, S2Fs,S2nbit] = wavread( fullfile(S2PathName,S2FileName) );   %# pass file path as string
        Info2=size(S2y);
        Length2=(Info2(1,1))/S2Fs;
        Length2 = round((Length2*1000))/1000; % Rounding Decimal Places
        Channels2=Info2(1,2);

        if S2FileName==0
            set(u12Name,'String','No WAV File Loaded')
        else
            set(u12Name,'String',S2FileName, 'FontWeight','bold')
            %set(u12Name,'BackgroundColor',[1 1 1])
            set(u12,'value',1)

               set(WAV2_Text1val,'String',num2str(S2Fs));
               set(WAV2_Text2val,'String',num2str(S2nbit));
               set(WAV2_Text3val,'String',Length2);
               set(WAV2_Text4val,'String',Channels2);
            ColourUiGroup2
        end


               set(WAV2_Name,'String',S2FileName);
               set(WAV2_Loc,'String',S2PathName);

        refresh(MainWindow);

        if WavLockCheck==1
            set(u1Name,'String',S2FileName, 'FontWeight','bold')
            %set(u1Name,'BackgroundColor',[1 1 1])
            set(u1,'value',1)

            S1y = S2y;
            S1Fs = S2Fs;
            S1nbit = S2nbit;

            set(WAV1_Name,'String',S2FileName);
            set(WAV1_Loc,'String',S2PathName);

            set(WAV1_Text1val,'String',num2str(S2Fs));
            set(WAV1_Text2val,'String',num2str(S2nbit));
            set(WAV1_Text3val,'String',Length2);
            set(WAV1_Text4val,'String',Channels2);

            refresh(MainWindow);
        end

    end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% %%%% CHECK HEADPHONE FILES LOADED                    %%%%%%%%%%%%%%%%%%%%%%%%
%%%%%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function headphonecheck(~,~)
    Prompt= figure('units', 'normalized','Position',[0.4 0.4 0.15 0.1], 'Visible', 'On');
    set(Prompt,'MenuBar','none','Name','Calibration Check','NumberTitle','Off');

    FinishedText = uicontrol('style','text',...
        'units', 'normalized',...
        'position', [0.05 0.6 0.9 0.3],...
        'string', 'No Calibration File Loaded',...
        'Visible','on',...
        'parent',Prompt,...
        'FontSize', Font12,...
        'BackgroundColor',[0.8 0.8 0.8]); %#ok<NASGU>
```

```
    Quit1 = uicontrol('style', 'pushbutton',...
        'string', 'Go Back',...
        'units', 'normalized',...
        'position', [0.2 0.1 0.6 0.4],...
        'parent',Prompt,...
        'FontSize', Font12,...
        'callback', @CloseFcn3); %#ok<NASGU> %Function Run when Button Pressed

    function CloseFcn3(~,~)
    delete(Prompt)
    end
end
%% Audiometer Check 1
function AudCheck1(~,~)
    % Quick check to verify that the participant could hear the pure tone
    % being analysed. Any tone not perceived was given a value of '-20' in
    % order to keep the plot point off of the graph. A quick check for this
    % value will determine the case present.

    AudPrompt= figure('units', 'normalized','Position',[0.4 0.4 0.15 0.125], 'Visible', 'On');
    set(AudPrompt,'MenuBar','none','Name','Frequency Check','NumberTitle','Off');

    FinishedText = uicontrol('style','text',...
        'units', 'normalized',...
        'position', [0.05 0.6 0.9 0.3],...
        'string', 'No Threshold Perceived at This Frequency.',...
        'Visible','on',...
        'parent',AudPrompt,...
        'FontSize', Font12,...
        'BackgroundColor',[0.8 0.8 0.8]); %#ok<NASGU>


    Quit1 = uicontrol('style', 'pushbutton',...
        'string', 'Go Back',...
        'units', 'normalized',...
        'position', [0.2 0.1 0.6 0.35],...
        'parent',AudPrompt,...
        'FontSize', Font12,...
        'callback', @CloseFcn4); %#ok<NASGU> %Function Run when Button Pressed

    function CloseFcn4(~,~)
    delete(AudPrompt)
    end
end
%% Audiometer Check
function AudCheck2(~,~)
    % Only runs when the dB(SL) option is selected. In this case the target
    % tone is a measure of the perceived level above the threshold.

    AudPrompt2= figure('units', 'normalized','Position',[0.4 0.4 0.15 0.125], 'Visible', 'On');
    set(AudPrompt2,'MenuBar','none','Name','Audiometer Check','NumberTitle','Off');

    FinishedText = uicontrol('style','text',...
        'units', 'normalized',...
        'position', [0.05 0.6 0.9 0.3],...
        'string', 'Audiometer File Has NOT Been Loaded.',...
        'Visible','on',...
        'parent',AudPrompt2,...
        'FontSize', Font12,...
        'BackgroundColor',[0.8 0.8 0.8]); %#ok<NASGU>


    Quit1 = uicontrol('style', 'pushbutton',...
        'string', 'Go Back',...
        'units', 'normalized',...
        'position', [0.2 0.1 0.6 0.35],...
        'parent',AudPrompt2,...
        'FontSize', Font12,...
        'callback', @CloseFcn5); %#ok<NASGU> %Function Run when Button Pressed

    function CloseFcn5(~,~)
    delete(AudPrompt2)
    end
end

function MaxSPLCheck(~,~)
    % Only runs when the dB(SL) option is selected. In this case the target
    % tone is a measure of the perceived level above the threshold.

    SPLPrompt= figure('units', 'normalized','Position',[0.4 0.4 0.15 0.125], 'Visible', 'On');
    set(SPLPrompt,'MenuBar','none','Name','Maximum SPL Exceeded','NumberTitle','Off');

    FinishedText = uicontrol('style','text',...
        'units', 'normalized',...
        'position', [0.05 0.6 0.9 0.3],...
        'string', 'Selected Level is above Max Allowable SPL.',...
        'Visible','on',...
        'parent',SPLPrompt,...
        'FontSize', Font12,...
        'BackgroundColor',[0.8 0.8 0.8]); %#ok<NASGU>


    Quit1 = uicontrol('style', 'pushbutton',...
        'string', 'Go Back',...
        'units', 'normalized',...
        'position', [0.2 0.1 0.6 0.35],...
        'parent',SPLPrompt,...
        'FontSize', Font12,...
        'callback', @CloseFcn6); %#ok<NASGU> %Function Run when Button Pressed

    function CloseFcn6(~,~)
    delete(SPLPrompt)
    end
end
%% Volume Control Function

MyButton = uicontrol('Style', 'pushbutton','Visible','off','Callback',@task);

    MinVol      =     -20;    % Minimum Volume

        Sig2Type1=get(u10,'Value');
        Sig2Type2=get(u11,'Value');
```

```
    Sig2Type3=get(u12,'Value');


% Function run when a Case-Key is pressed
function task(src, e)
    refresh(MainWindow);         % Refresh the Figure
end

% Function run when any Key is pressed
function keyPress(src, e, player)

    S2_PureToneFreq =   get(u10Freq ,'value');
    S2_ToneFreq     =   Freq(1,S2_PureToneFreq);
    MaxVolL          =   ExpAppZeros(S2_PureToneFreq,13);

      VolTone   = str2num(get(u10Amp,'String'));
      VolNoise  = str2num(get(u11Amp,'String'));
      VolWav    = str2num(get(u12Gain,'String'));

      if Sig2Type1==1; StartVol=VolTone; end
      if Sig2Type2==1; StartVol=VolNoise; end
      if Sig2Type3==1; StartVol=VolWav; end

      if get(R2,'value')==1;
          Audiogram=1;          % Use Audiogram
      else
          Audiogram=0;          % Ignore Audiogram
      end


          S1_ROW = get(u10Freq ,'value'); %Frequency Row Value
          S1_ToneFreq=Freq(1,S1_ROW); %Frequency

          H1_L = Headphones_A(S1_ROW,2);  %Participant Headphones L-Adjustment
          H1_R = Headphones_A(S1_ROW,3);  %Participant Headphones R-Adjustment
          H1_B = Headphones_A(S1_ROW,4);  %Participant Headphones B-Adjustment

          H2_L = Headphones_B(S1_ROW,2);  %HATS Headphones L-Adjustment
          H2_R = Headphones_B(S1_ROW,3);  %HATS Headphones R-Adjustment
          H2_B = Headphones_B(S1_ROW,4);  %HATS Headphones B-Adjustment


          H1Corr_S1L = H1_L; %S1L + H1L + AudCorrL (Singal 1 on Headphones 1 as SL)
          H1Corr_S1R = H1_R; %S1R + H1R + AudCorrR (Singal 1 on Headphones 1 as SL)
          H1Corr_S2L = H1_L; %S2L + H1L + AudCorrL (Singal 2 on Headphones 1 as SL)
          H1Corr_S2R = H1_R; %S2R + H1R + AudCorrR (Singal 2 on Headphones 1 as SL)

          H2Corr_S1L = H2_L; %S1L + H2L + AudCorrL (Singal 1 on Headphones 2 as SL)
          H2Corr_S1R = H2_R; %S1R + H2R + AudCorrR (Singal 1 on Headphones 2 as SL)
          H2Corr_S2L = H2_L; %S2L + H2L + AudCorrL (Singal 2 on Headphones 2 as SL)
          H2Corr_S2R = H2_R; %S2R + H2R + AudCorrR (Singal 2 on Headphones 2 as SL)

    if Audiogram==1;


            AudL1 = Resultant(S1_ROW,3);     %Audiometer L-Sensitivity
            AudR1 = Resultant(S1_ROW,5);     %Audiometer R-Sensitivity

            if get(A1,'value')==1; AudL = AudL1; AudR = AudR1; end    % A1 - L + R
            if get(A2,'value')==1; AudL = AudL1; AudR = AudL1; end    % A2 - Left
            if get(A3,'value')==1; AudL = AudR1; AudR = AudR1; end    % A3 - Right
            if get(A4,'value')==1; AudL = (AudL1+AudR1)/2; AudR = (AudL1+AudR1)/2; end    % A4 - AVG

          %fprintf('\nSensation Level dB(SL)\n')

          % The following is used to verify that perticipants receive
          % safe sound pressure levels.

            H1Corr_S1L = H1_L + AudL; %S1L + H1L + AudCorrL (Singal 1 on Headphones 1 as SL)
            H1Corr_S1R = H1_R + AudR; %S1R + H1R + AudCorrR (Singal 1 on Headphones 1 as SL)
            H1Corr_S2L = H1_L + AudL; %S2L + H1L + AudCorrL (Singal 2 on Headphones 1 as SL)
            H1Corr_S2R = H1_R + AudR; %S2R + H1R + AudCorrR (Singal 2 on Headphones 1 as SL)

            H2Corr_S1L = H2_L + AudL; %S1L + H2L + AudCorrL (Singal 1 on Headphones 2 as SL)
            H2Corr_S1R = H2_R + AudR; %S1R + H2R + AudCorrR (Singal 1 on Headphones 2 as SL)
            H2Corr_S2L = H2_L + AudL; %S2L + H2L + AudCorrL (Singal 2 on Headphones 2 as SL)
            H2Corr_S2R = H2_R + AudR; %S2R + H2R + AudCorrR (Singal 2 on Headphones 2 as SL)
    end

          MaxSet=ExpAppZeros(S2_PureToneFreq,13);
          MaxVolL = MaxSet-H1Corr_S2L;
          MaxVolR = MaxSet-H1Corr_S2R;

          if (MaxSet-H1Corr_S2R)<=MaxVolL;
              MaxVolL=(MaxSet-H1Corr_S2R);
          end

          if Sig2Type2==1;
              MaxVolL = 100;
          end

          if Sig2Type3==1;
              MaxVolL = 500; % 500% or 5x as loud !!!! Check Max Level Test !!!!
          end

          if StartVol>=MaxVolL;
              MaxVolL=floor(MaxVolL);
              if Sig2Type1==1;
                  set(u10Amp,'String',num2str(MaxVolL));
              end

              if Sig2Type2==1;
                  set(u11Amp,'String',num2str(MaxVolL));
              end

              if Sig2Type3==1;
                  set(u12Gain,'String',num2str(MaxVolL));
              end
          end



          %----------------------
```

```
  IncDecLarge = 5;
  WAVLarge = 25;
  IncDecSmall = 1;
  WAVSmall = 5;
  %---------------------

switch e.Key

% Case where right-arrow is pressed
case 'rightarrow'
    task(MyButton, []);
    if StartVol<MaxVolL
                if Sig2Type1==1;
                    StartVol=StartVol+IncDecLarge;
                    set(u10Amp,'String',num2str(StartVol));
                end

                if Sig2Type2==1;
                    StartVol=StartVol+IncDecLarge;
                    set(u11Amp,'String',num2str(StartVol));
                end

                if Sig2Type3==1;
                    StartVol=StartVol+WAVLarge;
                    set(u12Gain,'String',num2str(StartVol));
                end

%                fprintf('Increase, volume = %d\n',StartVol)
    else
%                fprintf('Increase, volume = MAX\n')
    end

    ExpAppZeros = getappdata(0,'evalue');
    PauseCheck = 1;
    ExpAppZeros(28,4) = PauseCheck;
    setappdata(0,'evalue',ExpAppZeros)


% Case where left-arrow is pressed
case 'leftarrow'
    task(MyButton, []);
    if StartVol>MinVol

        if Sig2Type1==1;
            StartVol=StartVol-IncDecLarge;
            set(u10Amp,'String',num2str(StartVol));
        end

        if Sig2Type2==1;
            StartVol=StartVol-IncDecLarge;
            set(u11Amp,'String',num2str(StartVol));
        end

        if Sig2Type3==1;
            StartVol=StartVol-WAVLarge;
            set(u12Gain,'String',num2str(StartVol));
        end
        %                           fprintf('Decrease, volume = %d\n',StartVol)
    else
%                fprintf('Decrease, volume = MIN\n')
    end

    ExpAppZeros = getappdata(0,'evalue');
    PauseCheck = 1;
    ExpAppZeros(28,4) = PauseCheck;
    setappdata(0,'evalue',ExpAppZeros)


% Case where up-arrow is pressed
case 'uparrow'
    task(MyButton, []);
    if StartVol<MaxVolL
                if Sig2Type1==1;
                    StartVol=StartVol+IncDecSmall;
                    set(u10Amp,'String',num2str(StartVol));
                end

                if Sig2Type2==1;
                    StartVol=StartVol+IncDecSmall;
                    set(u11Amp,'String',num2str(StartVol));
                end

                if Sig2Type3==1;
                    StartVol=StartVol+WAVSmall;
                    set(u12Gain,'String',num2str(StartVol));
                end

%                fprintf('Increase, volume = %d\n',StartVol)
    else
%                fprintf('Increase, volume = MAX\n')
    end

    ExpAppZeros = getappdata(0,'evalue');
    PauseCheck = 1;
    ExpAppZeros(28,4) = PauseCheck;
    setappdata(0,'evalue',ExpAppZeros)


% Case where down-arrow is pressed
case 'downarrow'
    task(MyButton, []);
    if StartVol>MinVol

                if Sig2Type1==1;
                    StartVol=StartVol-IncDecSmall;
                    set(u10Amp,'String',num2str(StartVol));
                end

                if Sig2Type2==1;
                    StartVol=StartVol-IncDecSmall;
                    set(u11Amp,'String',num2str(StartVol));
                end

                if Sig2Type3==1;
```

```
                                StartVol=StartVol-WAVSmall;
                                set(ul2Gain,'String',num2str(StartVol));
                            end
%                           fprintf('Decrease, volume = %d\n',StartVol)
                        else
%                           fprintf('Decrease, volume = MIN\n')
                        end

                        ExpAppZeros = getappdata(0,'evalue');
                        PauseCheck = 1;
                        ExpAppZeros(28,4) = PauseCheck;
                        setappdata(0,'evalue',ExpAppZeros)

                    % Case where a-button is pressed (CAPS or Non-CAPS)
                    case 'a'
                        pause(player)
                        fprintf('(A) button pressed\n')

                    end

            end

        %% Function to Notify User when the Audiogram is in use.
            function Greyout(~,~)
                TestdBZ = get(R1,'Value');

                ExpAppZeros = getappdata(0,'evalue');
                ExpAppZeros(29,4) = TestdBZ;
                setappdata(0,'evalue',ExpAppZeros) % Placeholder for Important Values

                Plotting_Function
            end


        function signal = fade( signal, fs, T, window )
        %   Author: Kamil Wojcicki, UTD, November 2011.
        % FADE Fade (taper) signal at extremities.
        %
        %   FADE(SIGNAL,FS,T,WINDOW) returns SIGNAL with leading and trailing
        %   samples faded-in and faded-out, respectively. The signal is sampled
        %   at FS Hz. The fade-in and fade-out durations are specified in T.
        %   The window function used for fading is supplied in WINDOW as a
        %   function handle.
        %
        %   Inputs
        %           SIGNAL input signal a vector.
        %
        %           FS sampling frequency (Hz).
        %
        %           T fade-in and fade-out durations (ms), as scalar (if same),
        %             or as a two element vector.
        %
        %           WINDOW function handles for window functions used for
        %                  fading-in and fading-out of the input signal,
        %                  as a single function handle, if same window
        %                  is to be used for fading-in and fading-out,
        %                  or as a two element cell array of window function
        %                  handles, if different window functions are to be used.
        %
        %   Outputs
        %           SIGNAL input signal with faded leading and trailing samples.
        %
        %   Author: Kamil Wojcicki, UTD, November 2011.


            % check that required inputs have been supplied
            if nargin~=4, help(mfilename); return; end;

            % check that input is a vector
            if min( size(signal) )>1
                error( 'Input signal has to be a vector.\n' );
            end

            % if only one fade was duration specified,
            % use it for both fade-in and fade-out durations
            if length(T)==1, T = [ T T ]; end; % Fade length (200 ms)

            % if only one window was specified,
            % use it for both fade-in and fade-out windows
            if length(window)==1, window = { window window }; end;

            % get input signal length
            L = length( signal );

            % determine fade durations (samples)
            N = round( T*1E-3*fs );

            % determine whether the input signal is in row or column form
            if size(signal,1)==1, form='row';
            else size(signal,2)==1, form='col';
            end

            % design fade-in window
            fadein = fade_window( N(1), 'fadein', window{1}, form );

            % generate fade-out indices
            index = 1:N(1);

            % apply fade-in window
            signal(index) = signal(index) .* fadein;


            % design fade-out window
            fadeout = fade_window( N(2), 'fadeout', window{2}, form );

            % generate fade-out indices
            index = L-N(2)+1:L;

            % apply fade-out window
            signal(index) = signal(index) .* fadeout;

        end

        function window = fade_window( N, type, custom, form )
```

```
%   Author: Kamil Wojcicki, UTD, November 2011.
% FADE_WINDOW Design window for signal fading-in or fading-out.
%
%   FADE_WINDOW(N,TYPE,CUSTOM) returns WINDOW of length N samples
%   for leading or trailing sample tapering of a signal as specified
%   by TYPE. Custom window function can be specified.
%
%   Inputs
%           N fading duration (samples).
%
%           TYPE fading direction as string,
%                i.e., 'fade-in' or 'fade-out'
%
%           CUSTOM function handle for tapering window function to
%                  be used for fade-in or fade-out window design.
%                  Note that symmetric tapered window is assumed,
%                  and that for design purposes window of double
%                  the length requested will be generated and then
%                  cut in half to retain the slope-up portion.
%
%           FORM specifies (as string) if the window should be
%                in row (i.e., 'row') or column (i.e., 'col') form.
%
%   Outputs
%           WINDOW window to be used for fading.

%   Author: Kamil Wojcicki, UTD, November 2011.


    % generate window samples
    if false
        % some hard-coded examples
%       window = triang( 2*N );
%       window = chebwin( 2*N, 100 );
        window = hanning( 2*N ).^2;
%       window = exp( linspace( -4,0,N ) );
    else
        % use a custom window instead
        window = custom( 2*N );
    end

    % ensure correct vector form
    switch lower( form )
    case 'col', window = window(:);
    case 'row', window = window(:).';
    end

    % truncate to half length (if 2*N was used)
    window = window(1:N);

    % scale to unit magnitude
    window = window / max( abs(window) );

    % time reverse for 'fade-out' option, ignore otherwise
    switch lower( type )
    case { 'fadeout', 'fade-out' },window = window(end:-1:1);
    end
% EOF
end
end % EOF (Experiment_v8.m)
```

A.2.2 Headphone Calibration (ExpCalibration.m)

```
 %cd 'C:\Users\Jeremy\Documents\PhD\PhD Project\Loudness Models\ANSI-S3S4-2005\ToneGenerator\Audiometer'

function ExpCalibration ()
% FIGURE DIMENSIONS
clc

CurrentSettings = getappdata(0,'evalue');

DefaultUserSettings = csvread('UserSettings.txt');
    playtime1               = DefaultUserSettings(1,5);
    pausetime1              = DefaultUserSettings(2,5);
    Fade1                   = DefaultUserSettings(3,5);
    Phase1                  = DefaultUserSettings(4,5);
    playtime2               = DefaultUserSettings(5,5);
    pausetime2              = DefaultUserSettings(6,5);
    Fade2                   = DefaultUserSettings(7,5);
    Phase2                  = DefaultUserSettings(8,5);
    Cal_Playback_Length     = DefaultUserSettings(9,5);
    Cal_Fade                = DefaultUserSettings(10,5);

amplitude_dB=[0,0];

if Cal_Playback_Length ==0 %Setting Default where settings not saved.
    Cal_Playback_Length=10;
    Cal_Fade=200;
end

Cal_File = getappdata(0,'evalue');
        % evalue Placeholders
        % Column 1: Frequency
        % Column 2: Loaded Cal Left
        % Column 3: Loaded Cal Right
        % Column 4: User Settings
        % Column 5: Resultant L-Small (:,3)
        % Column 6: Resultant R-Small (:,5)
        % Column 7: Headphones 1 - Left
        % Column 8: Headphones 1 - Right
        % Column 9: Headphones 2 - Left
        % Column 10: Headphones 2 - Right
        % Column 11: Ambient Conditions Left
        % Column 12: Ambient Conditions Right
        % Column 13: Maximum Limits
        % Column 14: Reference Contour
```

```
        % Column 15: Loaded Binaural Cal Headphones 1
        % Column 16: Loaded Binaural Cal Headphones 2

set(0,'units','pixels');
    RelFont = get(0,'ScreenSize');
    set(0,'Units','normalized');
    scnsize = get(0,'ScreenSize'); %#ok<NASGU>
    Background_Color=[0.8 0.8 0.8];

Font8   = round(8/1080*RelFont(1,4)); %Font 8 on 1920x1080
Font10  = round(10/1080*RelFont(1,4));
Font12  = round(12/1080*RelFont(1,4));
Font16  = round(16/1080*RelFont(1,4));

%set(MainWindow,'Visible','off')
% cal_f = figure('units', 'normalized','Position',[0.3125,0.075,0.425,0.85],...
%     'Color',Background_Color);
cal_f = figure('units', 'normalized','Position',[0.3125,0.05,0.425,0.90],...
    'Color',Background_Color);
    set(cal_f,'name','','numbertitle','off')
    set(cal_f,'MenuBar','none');

RelFigure=get(cal_f,'Position');

%ZeroVector=[0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0];
%Cal_File(:,1)=ZeroVector(:,1); Cal_File(:,2)=ZeroVector(:,1); Cal_File(:,3)=ZeroVector(:,1);

VertSpace   = 0.0275;
Bottom      = 0.0025;
% VertSpacing = zeros(1,29);
VertSpacing = zeros(1,31);

j=1;

% for j=1:29
% VertSpacing(1,j)=(Bottom+j*VertSpace);
% end

for j=1:31
VertSpacing(1,j)=(Bottom+j*VertSpace);
end
%Collect Values into one calibration array
% Freq=[20, 25, 31.5, 40, 50, 63, 80, 100, 125, 160, 200, 250, 315, 400,...
%     500, 630, 800, 1000, 1250, 1600, 2000, 2500, 3150, 4000, 5000, 6300,...
%     8000, 10000, 12500];
Freq=[20, 25, 31.5, 40, 50, 63, 80, 100, 125, 160, 200, 250, 315, 400,...
    500, 630, 800, 1000, 1250, 1600, 2000, 2500, 3150, 4000, 5000, 6300,...
    8000, 10000, 12500, 16000, 20000];


Title = uicontrol('style','text', 'units', 'normalized',...
    'position', [0.1 0.96 0.8 0.03], 'string', 'Calibration Values for Headphones',...
    'visible','on', 'FontSize', Font16,'HorizontalAlignment','Center',...
    'BackgroundColor',Background_Color, 'FontWeight', 'bold');

RelFigWidth = RelFigure(1,3);

W_Button    = 0.05/RelFigWidth;
Gap2        = 0.001/RelFigWidth;

Gap1        = (1/5)*(1-4*(2*W_Button+Gap2));

L_LGroup    = Gap1;
R_LGroup    = L_LGroup+Gap2+W_Button;

L_RGroup    = R_LGroup+Gap1+W_Button;
R_RGroup    = L_RGroup+Gap2+W_Button;

L_BGroup    = R_RGroup+Gap1+W_Button;
R_BGroup    = L_BGroup+Gap2+W_Button;

SPL_Group   = R_BGroup+Gap1+W_Button;

B_Height    = 0.024; %Button Height


L_Left      = L_LGroup;  %Left Cal Pushbutton Position
LC_Left     = R_LGroup;  %Left Cal text Position
R_Left      = L_RGroup;
RC_Left     = R_RGroup;
B_Left      = L_BGroup;
BC_Left     = R_BGroup;

L_Width     = W_Button;  %Left Cal Pushbutton Width
L_Height    = B_Height;  %Left Cal Pushbutton Height
LC_Width    = W_Button;  %Left Cal text Width
LC_Height   = B_Height;  %Left Cal text Height

R_Width     = W_Button;
R_Height    = B_Height;
RC_Width    = W_Button;
RC_Height   = B_Height;

B_Width     = W_Button;
BC_Width    = W_Button;
BC_Height   = B_Height;

Right_Edge = BC_Left+W_Button;
%% Target SPL Block at top of GUI
% Initial Text

SPL_Left=0.21;
SPL_Bottom=0.92;
% TW=0.35;
TW=2*RC_Width;
SPLW=0.1;
DH=0.005;

SPL_Gap=0.5*(1-(TW+2*DH+2*SPLW));

Target = uicontrol('style','text', 'units', 'normalized',...
    'position', [SPL_Gap SPL_Bottom TW 0.03], 'string', 'Target SPL Value: ',...
    'visible','on', 'FontSize', Font12,'HorizontalAlignment','Right',...
    'BackgroundColor',Background_Color);
```

```
% Edit box to enter Target SPL Value
SPL = uicontrol('style','edit', 'units', 'normalized',...
    'position', [SPL_Gap+TW+DH SPL_Bottom+0.005 SPLW 0.025], 'string',0,...
    'visible','on', 'FontSize', Font12);
% Units indicator
SPL_Units = uicontrol('style','text', 'units', 'normalized',...
    'position', [SPL_Gap+TW+2*DH+SPLW SPL_Bottom SPLW 0.03], 'string', ' dB',...
    'visible','on', 'FontSize', Font12,'HorizontalAlignment','Left',...
    'BackgroundColor',Background_Color);
%% Overall Correction Value Set Button
%    - Adds an equal amount over all frequencies as a broad correction
% Edit box to enter Correction Value

% % L: Text
TopRow=27; %Top row of Left/Right Adding Function
BoxSize=5*(VertSpacing(1,1));

Addition_Frame = uicontrol('style','frame', 'units', 'normalized',...
    'position', [SPL_Group VertSpacing(1,TopRow-3) 2*W_Button BoxSize],...
    'visible','on');

Increase= uicontrol('style','text', 'units', 'normalized',...
    'position', [SPL_Group+0.01 VertSpacing(1,TopRow+1)+.005 1.75*W_Button 0.025], 'string', 'Increase All Equally',...
    'visible','on', 'FontSize', Font10, 'FontWeight','Bold'); %,...'BackgroundColor',Background_Color);

L_Add= uicontrol('style','text', 'units', 'normalized',...
    'position', [SPL_Group+0.01 VertSpacing(1,TopRow) W_Button*.66 0.025], 'string', 'Left: ',...
    'visible','on', 'FontSize', Font12); %,...'BackgroundColor',Background_Color);

OVL_CORR_L = uicontrol('style','edit', 'units', 'normalized',...
    'position', [Right_Edge+W_Button+Gap2 VertSpacing(1,TopRow) 1.5*0.075 0.025], 'string',0,...
    'visible','on', 'FontSize', Font12);

R_Add= uicontrol('style','text', 'units', 'normalized',...
    'position', [SPL_Group+0.01 VertSpacing(1,TopRow-1) W_Button*.66 0.025], 'string', 'Right: ',...
    'visible','on', 'FontSize', Font12,'HorizontalAlignment','Center');

    %'BackgroundColor',Background_Color);

OVL_CORR_R = uicontrol('style','edit', 'units', 'normalized',...
    'position', [Right_Edge+W_Button+Gap2 VertSpacing(1,TopRow-1) 1.5*0.075 0.025], 'string',0,...
    'visible','on', 'FontSize', Font12);

OVL_CORR_BUTTON= uicontrol('style', 'pushbutton','string', 'Add to All','units',...
    'normalized','Visible','on','position', [SPL_Group+0.5*(W_Button) VertSpacing(1,TopRow-3)+.015 W_Button 0.03],...
    'FontSize', Font8,'callback', {@OVL_CORR_FCN});
%% Left and Right Ear Column Labels
Ext=0.075; %Extra space required for "BINAURAL"
Shift=((2*(W_Button+Gap2)-(W_Button+Ext)))/2; %Shift to position text in center
L_Label = uicontrol('style','text', 'units', 'normalized',...
    'position', [L_Left+Shift 0.875 W_Button+Ext 0.03], 'string', 'LEFT',...
    'visible','on', 'FontSize', Font12,'HorizontalAlignment','CENTER',...
    'BackgroundColor',Background_Color, 'FontWeight', 'bold');

R_Label = uicontrol('style','text', 'units', 'normalized',...
    'position', [R_Left+Shift 0.875 W_Button+Ext 0.03], 'string', 'RIGHT',...
    'visible','on', 'FontSize', Font12,'HorizontalAlignment','CENTER',...
    'BackgroundColor',Background_Color, 'FontWeight', 'bold');

B_Label = uicontrol('style','text', 'units', 'normalized',...
    'position', [B_Left+Shift 0.875 W_Button+Ext 0.03], 'string', 'BINAURAL',...
    'visible','on', 'FontSize', Font12,'HorizontalAlignment','CENTER',...
    'BackgroundColor',Background_Color, 'FontWeight', 'bold');
%% %%%%%%%%%%%%%%%%%%     LEFT EAR CALIBRATION VALUES      %%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%



L00020Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [LC_Left VertSpacing(1,1) LC_Width LC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

L00025Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [LC_Left VertSpacing(1,2) LC_Width LC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

L000315Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [LC_Left VertSpacing(1,3) LC_Width LC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

L00040Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [LC_Left VertSpacing(1,4) LC_Width LC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

L00050Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [LC_Left VertSpacing(1,5) LC_Width LC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

L00063Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [LC_Left VertSpacing(1,6) LC_Width LC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

L00080Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [LC_Left VertSpacing(1,7) LC_Width LC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

L00100Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [LC_Left VertSpacing(1,8) LC_Width LC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

L00125Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [LC_Left VertSpacing(1,9) LC_Width LC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

L00160Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [LC_Left VertSpacing(1,10) LC_Width LC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

L00200Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [LC_Left VertSpacing(1,11) LC_Width LC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);
```

```
L00250Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [LC_Left VertSpacing(1,12) LC_Width LC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

L00315Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [LC_Left VertSpacing(1,13) LC_Width LC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

L00400Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [LC_Left VertSpacing(1,14) LC_Width LC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

L00500Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [LC_Left VertSpacing(1,15) LC_Width LC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

L00630Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [LC_Left VertSpacing(1,16) LC_Width LC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

L00800Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [LC_Left VertSpacing(1,17) LC_Width LC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

L01000Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [LC_Left VertSpacing(1,18) LC_Width LC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

L01250Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [LC_Left VertSpacing(1,19) LC_Width LC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

L01600Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [LC_Left VertSpacing(1,20) LC_Width LC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

L02000Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [LC_Left VertSpacing(1,21) LC_Width LC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

L02500Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [LC_Left VertSpacing(1,22) LC_Width LC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

L03150Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [LC_Left VertSpacing(1,23) LC_Width LC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

L04000Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [LC_Left VertSpacing(1,24) LC_Width LC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

L05000Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [LC_Left VertSpacing(1,25) LC_Width LC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

L06300Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [LC_Left VertSpacing(1,26) LC_Width LC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

L08000Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [LC_Left VertSpacing(1,27) LC_Width LC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

L10000Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [LC_Left VertSpacing(1,28) LC_Width LC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

L12500Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [LC_Left VertSpacing(1,29) LC_Width LC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

L16000Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [LC_Left VertSpacing(1,30) LC_Width LC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

L20000Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [LC_Left VertSpacing(1,31) LC_Width LC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% %%%%%%%%%%%%%%%%%%     LEFT EAR PUSHBUTTONS              %%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%



L_00020Hz = uicontrol('style', 'pushbutton','string', '20 Hz','units',...
    'normalized','Visible','on','position', [L_Left VertSpacing(1,1) L_Width L_Height],...
    'FontSize', Font8,'callback', {@Left_00020Hz, SPL,L00020Hz_Corr});

    function Left_00020Hz(~,~,SPL,L00020Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corr=str2num(get(L00020Hz_Corr,'string'));
        amplitude_dB(1,1)=SPL_Level+corr;
        PlotTone_adj (20, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'left', 1,0);
    end

L_00025Hz = uicontrol('style', 'pushbutton','string', '25 Hz','units',...
    'normalized','Visible','on','position', [L_Left VertSpacing(1,2) L_Width L_Height],...
    'FontSize', Font8,'callback', {@Left_00025Hz, SPL,L00025Hz_Corr});

    function Left_00025Hz(~,~,SPL,L00025Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corr=str2num(get(L00025Hz_Corr,'string'));
        amplitude_dB(1,1)=SPL_Level+corr;
        PlotTone_adj (25, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'left', 1,0);
```

```
    end

L_000315Hz = uicontrol('style', 'pushbutton','string', '31.5 Hz','units',...
    'normalized','Visible','on','position', [L_Left VertSpacing(1,3) L_Width L_Height],...
    'FontSize', Font8,'callback', {@Left_000315Hz, SPL,L000315Hz_Corr});

    function Left_000315Hz(~,~,SPL,L000315Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corr=str2num(get(L000315Hz_Corr,'string'));
        amplitude_dB(1,1)=SPL_Level+corr;
        PlotTone_adj (31.5, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'left', 1,0);
    end

L_00040Hz = uicontrol('style', 'pushbutton','string', '40 Hz','units',...
    'normalized','Visible','on','position', [L_Left VertSpacing(1,4) L_Width L_Height],...
    'FontSize', Font8,'callback', {@Left_00040Hz, SPL,L00040Hz_Corr});

    function Left_00040Hz(~,~,SPL,L00040Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corr=str2num(get(L00040Hz_Corr,'string'));
        amplitude_dB(1,1)=SPL_Level+corr;
        PlotTone_adj (40, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'left', 1,0);
    end

L_00050Hz = uicontrol('style', 'pushbutton','string', '50 Hz','units',...
    'normalized','Visible','on','position', [L_Left VertSpacing(1,5) L_Width L_Height],...
    'FontSize', Font8,'callback', {@Left_00050Hz, SPL,L00050Hz_Corr});

    function Left_00050Hz(~,~,SPL,L00050Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corr=str2num(get(L00050Hz_Corr,'string'));
        amplitude_dB(1,1)=SPL_Level+corr;
        PlotTone_adj (50, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'left', 1,0);
    end

L_00063Hz = uicontrol('style', 'pushbutton','string', '63 Hz','units',...
    'normalized','Visible','on','position', [L_Left VertSpacing(1,6) L_Width L_Height],...
    'FontSize', Font8,'callback', {@Left_00063Hz, SPL,L00063Hz_Corr});

    function Left_00063Hz(~,~,SPL,L00063Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corr=str2num(get(L00063Hz_Corr,'string'));
        amplitude_dB(1,1)=SPL_Level+corr;
        PlotTone_adj (63, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'left', 1,0);
    end

L_00080Hz = uicontrol('style', 'pushbutton','string', '80 Hz','units',...
    'normalized','Visible','on','position', [L_Left VertSpacing(1,7) L_Width L_Height],...
    'FontSize', Font8,'callback', {@Left_00080Hz, SPL,L00080Hz_Corr});

    function Left_00080Hz(~,~,SPL,L00080Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corr=str2num(get(L00080Hz_Corr,'string'));
        amplitude_dB(1,1)=SPL_Level+corr;
        PlotTone_adj (80, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'left', 1,0);
    end

L_00100Hz = uicontrol('style', 'pushbutton','string', '100 Hz','units',...
    'normalized','Visible','on','position', [L_Left VertSpacing(1,8) L_Width L_Height],...
    'FontSize', Font8,'callback', {@Left_00100Hz, SPL,L00100Hz_Corr});

    function Left_00100Hz(~,~,SPL,L00100Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corr=str2num(get(L00100Hz_Corr,'string'));
        amplitude_dB(1,1)=SPL_Level+corr;
        PlotTone_adj (100, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'left', 1,0);
    end

L_00125Hz = uicontrol('style', 'pushbutton','string', '125 Hz','units',...
    'normalized','Visible','on','position', [L_Left VertSpacing(1,9) L_Width L_Height],...
    'FontSize', Font8,'callback', {@Left_00125Hz, SPL,L00125Hz_Corr});

    function Left_00125Hz(~,~,SPL,L00125Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corr=str2num(get(L00125Hz_Corr,'string'));
        amplitude_dB(1,1)=SPL_Level+corr;
        PlotTone_adj (125, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'left', 1,0);
    end

L_00160Hz = uicontrol('style', 'pushbutton','string', '160 Hz','units',...
    'normalized','Visible','on','position', [L_Left VertSpacing(1,10) L_Width L_Height],...
    'FontSize', Font8,'callback', {@Left_00160Hz, SPL,L00160Hz_Corr});

    function Left_00160Hz(~,~,SPL,L00160Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corr=str2num(get(L00160Hz_Corr,'string'));
        amplitude_dB(1,1)=SPL_Level+corr;
        PlotTone_adj (160, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'left', 1,0);
    end
```

```
L_00200Hz = uicontrol('style', 'pushbutton','string', '200 Hz','units',...
    'normalized','Visible','on','position', [L_Left VertSpacing(1,11) L_Width L_Height],...
    'FontSize', Font8,'callback', {@Left_00200Hz, SPL,L00200Hz_Corr});

    function Left_00200Hz(~,~,SPL,L00200Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corr=str2num(get(L00200Hz_Corr,'string'));
        amplitude_dB(1,1)=SPL_Level+corr;
        PlotTone_adj (200, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'left', 1,0);
    end

L_00250Hz = uicontrol('style', 'pushbutton','string', '250 Hz','units',...
    'normalized','Visible','on','position', [L_Left VertSpacing(1,12) L_Width L_Height],...
    'FontSize', Font8,'callback', {@Left_00250Hz, SPL,L00250Hz_Corr});

    function Left_00250Hz(~,~,SPL,L00250Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corr=str2num(get(L00250Hz_Corr,'string'));
        amplitude_dB(1,1)=SPL_Level+corr;
        PlotTone_adj (250, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'left', 1,0);
    end

L_00315Hz = uicontrol('style', 'pushbutton','string', '315 Hz','units',...
    'normalized','Visible','on','position', [L_Left VertSpacing(1,13) L_Width L_Height],...
    'FontSize', Font8,'callback', {@Left_00315Hz, SPL,L00315Hz_Corr});

    function Left_00315Hz(~,~,SPL,L00315Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corr=str2num(get(L00315Hz_Corr,'string'));
        amplitude_dB(1,1)=SPL_Level+corr;
        PlotTone_adj (315, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'left', 1,0);
    end

L_00400Hz = uicontrol('style', 'pushbutton','string', '400 Hz','units',...
    'normalized','Visible','on','position', [L_Left VertSpacing(1,14) L_Width L_Height],...
    'FontSize', Font8,'callback', {@Left_00400Hz, SPL,L00400Hz_Corr});

    function Left_00400Hz(~,~,SPL,L00400Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corr=str2num(get(L00400Hz_Corr,'string'));
        amplitude_dB(1,1)=SPL_Level+corr;
        PlotTone_adj (400, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'left', 1,0);
    end

L_00500Hz = uicontrol('style', 'pushbutton','string', '500 Hz','units',...
    'normalized','Visible','on','position', [L_Left VertSpacing(1,15) L_Width L_Height],...
    'FontSize', Font8,'callback', {@Left_00500Hz, SPL,L00500Hz_Corr});

    function Left_00500Hz(~,~,SPL,L00500Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corr=str2num(get(L00500Hz_Corr,'string'));
        amplitude_dB(1,1)=SPL_Level+corr;
        PlotTone_adj (500, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'left', 1,0);
    end

L_00630Hz = uicontrol('style', 'pushbutton','string', '630 Hz','units',...
    'normalized','Visible','on','position', [L_Left VertSpacing(1,16) L_Width L_Height],...
    'FontSize', Font8,'callback', {@Left_00630Hz, SPL,L00630Hz_Corr});

    function Left_00630Hz(~,~,SPL,L00630Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corr=str2num(get(L00630Hz_Corr,'string'));
        amplitude_dB(1,1)=SPL_Level+corr;
        PlotTone_adj (630, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'left', 1,0);
    end

L_00800Hz = uicontrol('style', 'pushbutton','string', '800 Hz','units',...
    'normalized','Visible','on','position', [L_Left VertSpacing(1,17) L_Width L_Height],...
    'FontSize', Font8,'callback', {@Left_00800Hz, SPL,L00800Hz_Corr});

    function Left_00800Hz(~,~,SPL,L00800Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corr=str2num(get(L00800Hz_Corr,'string'));
        amplitude_dB(1,1)=SPL_Level+corr;
        PlotTone_adj (800, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'left', 1,0);
    end

L_01000Hz = uicontrol('style', 'pushbutton','string', '1000 Hz','units',...
    'normalized','Visible','on','position', [L_Left VertSpacing(1,18) L_Width L_Height],...
    'FontSize', Font8,'callback', {@Left_01000Hz, SPL,L01000Hz_Corr});

    function Left_01000Hz(~,~,SPL,L01000Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corr=str2num(get(L01000Hz_Corr,'string'));
        amplitude_dB(1,1)=SPL_Level+corr;
        PlotTone_adj (1000, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'left', 1,0);
    end

L_01250Hz = uicontrol('style', 'pushbutton','string', '1250 Hz','units',...
    'normalized','Visible','on','position', [L_Left VertSpacing(1,19) L_Width L_Height],...
```

```
    'FontSize', Font8,'callback', {@Left_01250Hz, SPL,L01250Hz_Corr});

    function Left_01250Hz(~,~,SPL,L01250Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corr=str2num(get(L01250Hz_Corr,'string'));
        amplitude_dB(1,1)=SPL_Level+corr;
        PlotTone_adj (1250, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'left', 1,0);
    end

L_01600Hz = uicontrol('style', 'pushbutton','string', '1600 Hz','units',...
    'normalized','Visible','on','position', [L_Left VertSpacing(1,20) L_Width L_Height],...
    'FontSize', Font8,'callback', {@Left_01600Hz, SPL,L01600Hz_Corr});

    function Left_01600Hz(~,~,SPL,L01600Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corr=str2num(get(L01600Hz_Corr,'string'));
        amplitude_dB(1,1)=SPL_Level+corr;
        PlotTone_adj (1600, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'left', 1,0);
    end

L_02000Hz = uicontrol('style', 'pushbutton','string', '2000 Hz','units',...
    'normalized','Visible','on','position', [L_Left VertSpacing(1,21) L_Width L_Height],...
    'FontSize', Font8,'callback', {@Left_02000Hz, SPL,L02000Hz_Corr});

    function Left_02000Hz(~,~,SPL,L02000Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corr=str2num(get(L02000Hz_Corr,'string'));
        amplitude_dB(1,1)=SPL_Level+corr;
        PlotTone_adj (2000, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'left', 1,0);
    end

L_02500Hz = uicontrol('style', 'pushbutton','string', '2500 Hz','units',...
    'normalized','Visible','on','position', [L_Left VertSpacing(1,22) L_Width L_Height],...
    'FontSize', Font8,'callback', {@Left_02500Hz, SPL,L02500Hz_Corr});

    function Left_02500Hz(~,~,SPL,L02500Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corr=str2num(get(L02500Hz_Corr,'string'));
        amplitude_dB(1,1)=SPL_Level+corr;
        PlotTone_adj (2500, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'left', 1,0);
    end

L_03150Hz = uicontrol('style', 'pushbutton','string', '3150 Hz','units',...
    'normalized','Visible','on','position', [L_Left VertSpacing(1,23) L_Width L_Height],...
    'FontSize', Font8,'callback', {@Left_03150Hz, SPL,L03150Hz_Corr});

    function Left_03150Hz(~,~,SPL,L03150Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corr=str2num(get(L03150Hz_Corr,'string'));
        amplitude_dB(1,1)=SPL_Level+corr;
        PlotTone_adj (3150, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'left', 1,0);
    end

L_04000Hz = uicontrol('style', 'pushbutton','string', '4000 Hz','units',...
    'normalized','Visible','on','position', [L_Left VertSpacing(1,24) L_Width L_Height],...
    'FontSize', Font8,'callback', {@Left_04000Hz, SPL,L04000Hz_Corr});

    function Left_04000Hz(~,~,SPL,L04000Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corr=str2num(get(L04000Hz_Corr,'string'));
        amplitude_dB(1,1)=SPL_Level+corr;
        PlotTone_adj (4000, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'left', 1,0);
    end

L_05000Hz = uicontrol('style', 'pushbutton','string', '5000 Hz','units',...
    'normalized','Visible','on','position', [L_Left VertSpacing(1,25) L_Width L_Height],...
    'FontSize', Font8,'callback', {@Left_05000Hz, SPL,L05000Hz_Corr});

    function Left_05000Hz(~,~,SPL,L05000Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corr=str2num(get(L05000Hz_Corr,'string'));
        amplitude_dB(1,1)=SPL_Level+corr;
        PlotTone_adj (5000, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'left', 1,0);
    end

L_06300Hz = uicontrol('style', 'pushbutton','string', '6300 Hz','units',...
    'normalized','Visible','on','position', [L_Left VertSpacing(1,26) L_Width L_Height],...
    'FontSize', Font8,'callback', {@Left_06300Hz, SPL,L06300Hz_Corr});

    function Left_06300Hz(~,~,SPL,L06300Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corr=str2num(get(L06300Hz_Corr,'string'));
        amplitude_dB(1,1)=SPL_Level+corr;
        PlotTone_adj (6300, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'left', 1,0);
    end

L_08000Hz = uicontrol('style', 'pushbutton','string', '8000 Hz','units',...
    'normalized','Visible','on','position', [L_Left VertSpacing(1,27) L_Width L_Height],...
    'FontSize', Font8,'callback', {@Left_08000Hz, SPL,L08000Hz_Corr});
```

```
    function Left_08000Hz(~,~,SPL,L08000Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corr=str2num(get(L08000Hz_Corr,'string'));
        amplitude_dB(1,1)=SPL_Level+corr;
        PlotTone_adj (8000, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'left', 1,0);
    end

L_10000Hz = uicontrol('style', 'pushbutton','string', '10000 Hz','units',...
    'normalized','Visible','on','position', [L_Left VertSpacing(1,28) L_Width L_Height],...
    'FontSize', Font8,'callback', {@Left_10000Hz, SPL,L10000Hz_Corr});

    function Left_10000Hz(~,~,SPL,L10000Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corr=str2num(get(L10000Hz_Corr,'string'));
        amplitude_dB(1,1)=SPL_Level+corr;
        PlotTone_adj (10000, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'left', 1,0);
    end

L_12500Hz = uicontrol('style', 'pushbutton','string', '12500 Hz','units',...
    'normalized','Visible','on','position', [L_Left VertSpacing(1,29) L_Width L_Height],...
    'FontSize', Font8,'callback', {@Left_12500Hz, SPL,L12500Hz_Corr});

    function Left_12500Hz(~,~,SPL,L12500Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corr=str2num(get(L12500Hz_Corr,'string'));
        amplitude_dB(1,1)=SPL_Level+corr;
        PlotTone_adj (12500, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'left', 1,0);
    end

L_16000Hz = uicontrol('style', 'pushbutton','string', '16000 Hz','units',...
    'normalized','Visible','on','position', [L_Left VertSpacing(1,30) L_Width L_Height],...
    'FontSize', Font8,'callback', {@Left_16000Hz, SPL,L16000Hz_Corr});

    function Left_16000Hz(~,~,SPL,L16000Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corr=str2num(get(L16000Hz_Corr,'string'));
        amplitude_dB(1,1)=SPL_Level+corr;
        PlotTone_adj (16000, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'left', 1,0);
    end

L_20000Hz = uicontrol('style', 'pushbutton','string', '20000 Hz','units',...
    'normalized','Visible','on','position', [L_Left VertSpacing(1,31) L_Width L_Height],...
    'FontSize', Font8,'callback', {@Left_20000Hz, SPL,L20000Hz_Corr});

    function Left_20000Hz(~,~,SPL,L20000Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corr=str2num(get(L20000Hz_Corr,'string'));
        amplitude_dB(1,1)=SPL_Level+corr;
        PlotTone_adj (20000, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'left', 1,0);
    end
%% %%%%%%%%%%%%%%%%%     RIGHT EAR CALIBRATION VALUES     %%%%%%%%%%%%%%%%%


R00020Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [RC_Left VertSpacing(1,1) RC_Width RC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

R00025Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [RC_Left VertSpacing(1,2) RC_Width RC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

R000315Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [RC_Left VertSpacing(1,3) RC_Width RC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

R00040Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [RC_Left VertSpacing(1,4) RC_Width RC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

R00050Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [RC_Left VertSpacing(1,5) RC_Width RC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

R00063Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [RC_Left VertSpacing(1,6) RC_Width RC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

R00080Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [RC_Left VertSpacing(1,7) RC_Width RC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

R00100Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [RC_Left VertSpacing(1,8) RC_Width RC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

R00125Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [RC_Left VertSpacing(1,9) RC_Width RC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

R00160Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [RC_Left VertSpacing(1,10) RC_Width RC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

R00200Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [RC_Left VertSpacing(1,11) RC_Width RC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

R00250Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [RC_Left VertSpacing(1,12) RC_Width RC_Height], 'string', 0.0,...
```

```
    'visible','on', 'FontSize', Font8);

R00315Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [RC_Left VertSpacing(1,13) RC_Width RC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

R00400Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [RC_Left VertSpacing(1,14) RC_Width RC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

R00500Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [RC_Left VertSpacing(1,15) RC_Width RC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

R00630Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [RC_Left VertSpacing(1,16) RC_Width RC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

R00800Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [RC_Left VertSpacing(1,17) RC_Width RC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

R01000Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [RC_Left VertSpacing(1,18) RC_Width RC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

R01250Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [RC_Left VertSpacing(1,19) RC_Width RC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

R01600Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [RC_Left VertSpacing(1,20) RC_Width RC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

R02000Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [RC_Left VertSpacing(1,21) RC_Width RC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

R02500Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [RC_Left VertSpacing(1,22) RC_Width RC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

R03150Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [RC_Left VertSpacing(1,23) RC_Width RC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

R04000Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [RC_Left VertSpacing(1,24) RC_Width RC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

R05000Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [RC_Left VertSpacing(1,25) RC_Width RC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

R06300Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [RC_Left VertSpacing(1,26) RC_Width RC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

R08000Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [RC_Left VertSpacing(1,27) RC_Width RC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

R10000Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [RC_Left VertSpacing(1,28) RC_Width RC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

R12500Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [RC_Left VertSpacing(1,29) RC_Width RC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

R16000Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [RC_Left VertSpacing(1,30) RC_Width RC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

R20000Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [RC_Left VertSpacing(1,31) RC_Width RC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% %%%%%%%%%%%%%%%%%     RIGHT EAR PUSHBUTTONS            %%%%%%%%%%%%%%%%%

R_00020Hz = uicontrol('style', 'pushbutton','string', '20 Hz','units',...
    'normalized','Visible','on','position', [R_Left VertSpacing(1,1) R_Width R_Height],...
    'FontSize', Font8,'callback', {@Right_00020Hz,SPL,R00020Hz_Corr});

    function Right_00020Hz(~,~,SPL,R00020Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corr=str2num(get(R00020Hz_Corr,'string'));
        amplitude_dB(1,2)=SPL_Level+corr;
        PlotTone_adj (20, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'Right', 1,0);
    end

R_00025Hz = uicontrol('style', 'pushbutton','string', '25 Hz','units',...
    'normalized','Visible','on','position', [R_Left VertSpacing(1,2) R_Width R_Height],...
    'FontSize', Font8,'callback', {@Right_00025Hz,SPL,R00025Hz_Corr});

    function Right_00025Hz(~,~,SPL,R00025Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corr=str2num(get(R00025Hz_Corr,'string'));
        amplitude_dB(1,2)=SPL_Level+corr;
        PlotTone_adj (25, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'Right', 1,0);
    end

R_000315Hz = uicontrol('style', 'pushbutton','string', '31.5 Hz','units',...
    'normalized','Visible','on','position', [R_Left VertSpacing(1,3) R_Width R_Height],...
    'FontSize', Font8,'callback', {@Right_000315Hz,SPL,R000315Hz_Corr});

    function Right_000315Hz(~,~,SPL,R000315Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
```

```
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corr=str2num(get(R000315Hz_Corr,'string'));
        amplitude_dB(1,2)=SPL_Level+corr;
        PlotTone_adj (31.5, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'Right', 1,0);
    end

R_00040Hz = uicontrol('style', 'pushbutton','string', '40 Hz','units',...
    'normalized','Visible','on','position', [R_Left VertSpacing(1,4) R_Width R_Height],...
    'FontSize', Font8,'callback', {@Right_00040Hz,SPL,R00040Hz_Corr});

    function Right_00040Hz(~,~,SPL,R00040Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corr=str2num(get(R00040Hz_Corr,'string'));
        amplitude_dB(1,2)=SPL_Level+corr;
        PlotTone_adj (40, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'Right', 1,0);
    end

R_00050Hz = uicontrol('style', 'pushbutton','string', '50 Hz','units',...
    'normalized','Visible','on','position', [R_Left VertSpacing(1,5) R_Width R_Height],...
    'FontSize', Font8,'callback', {@Right_00050Hz,SPL,R00050Hz_Corr});

    function Right_00050Hz(~,~,SPL,R00050Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corr=str2num(get(R00050Hz_Corr,'string'));
        amplitude_dB(1,2)=SPL_Level+corr;
        PlotTone_adj (50, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'Right', 1,0);
    end

R_00063Hz = uicontrol('style', 'pushbutton','string', '63 Hz','units',...
    'normalized','Visible','on','position', [R_Left VertSpacing(1,6) R_Width R_Height],...
    'FontSize', Font8,'callback', {@Right_00063Hz,SPL,R00063Hz_Corr});

    function Right_00063Hz(~,~,SPL,R00063Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corr=str2num(get(R00063Hz_Corr,'string'));
        amplitude_dB(1,2)=SPL_Level+corr;
        PlotTone_adj (63, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'Right', 1,0);
    end

R_00080Hz = uicontrol('style', 'pushbutton','string', '80 Hz','units',...
    'normalized','Visible','on','position', [R_Left VertSpacing(1,7) R_Width R_Height],...
    'FontSize', Font8,'callback', {@Right_00080Hz,SPL,R00080Hz_Corr});

    function Right_00080Hz(~,~,SPL,R00080Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corr=str2num(get(R00080Hz_Corr,'string'));
        amplitude_dB(1,2)=SPL_Level+corr;
        PlotTone_adj (80, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'Right', 1,0);
    end

R_00100Hz = uicontrol('style', 'pushbutton','string', '100 Hz','units',...
    'normalized','Visible','on','position', [R_Left VertSpacing(1,8) R_Width R_Height],...
    'FontSize', Font8,'callback', {@Right_00100Hz,SPL,R00100Hz_Corr});

    function Right_00100Hz(~,~,SPL,R00100Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corr=str2num(get(R00100Hz_Corr,'string'));
        amplitude_dB(1,2)=SPL_Level+corr;
        PlotTone_adj (100, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'Right', 1,0);
    end

R_00125Hz = uicontrol('style', 'pushbutton','string', '125 Hz','units',...
    'normalized','Visible','on','position', [R_Left VertSpacing(1,9) R_Width R_Height],...
    'FontSize', Font8,'callback', {@Right_00125Hz,SPL,R00125Hz_Corr});

    function Right_00125Hz(~,~,SPL,R00125Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corr=str2num(get(R00125Hz_Corr,'string'));
        amplitude_dB(1,2)=SPL_Level+corr;
        PlotTone_adj (125, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'Right', 1,0);
    end

R_00160Hz = uicontrol('style', 'pushbutton','string', '160 Hz','units',...
    'normalized','Visible','on','position', [R_Left VertSpacing(1,10) R_Width R_Height],...
    'FontSize', Font8,'callback', {@Right_00160Hz,SPL,R00160Hz_Corr});

    function Right_00160Hz(~,~,SPL,R00160Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corr=str2num(get(R00160Hz_Corr,'string'));
        amplitude_dB(1,2)=SPL_Level+corr;
        PlotTone_adj (160, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'Right', 1,0);
    end

R_00200Hz = uicontrol('style', 'pushbutton','string', '200 Hz','units',...
    'normalized','Visible','on','position', [R_Left VertSpacing(1,11) R_Width R_Height],...
    'FontSize', Font8,'callback', {@Right_00200Hz,SPL,R00200Hz_Corr});

    function Right_00200Hz(~,~,SPL,R00200Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
```

```
        SPL_Level=str2num(get(SPL,'string'));
        corr=str2num(get(R00200Hz_Corr,'string'));
        amplitude_dB(1,2)=SPL_Level+corr;
        PlotTone_adj (200, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'Right', 1,0);
    end

R_00250Hz = uicontrol('style', 'pushbutton','string', '250 Hz','units',...
    'normalized','Visible','on','position', [R_Left VertSpacing(1,12) R_Width R_Height],...
    'FontSize', Font8,'callback', {@Right_00250Hz,SPL,R00250Hz_Corr});

    function Right_00250Hz(~,~,SPL,R00250Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corr=str2num(get(R00250Hz_Corr,'string'));
        amplitude_dB(1,2)=SPL_Level+corr;
        PlotTone_adj (250, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'Right', 1,0);
    end

R_00315Hz = uicontrol('style', 'pushbutton','string', '315 Hz','units',...
    'normalized','Visible','on','position', [R_Left VertSpacing(1,13) R_Width R_Height],...
    'FontSize', Font8,'callback', {@Right_00315Hz,SPL,R00315Hz_Corr});

    function Right_00315Hz(~,~,SPL,R00315Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corr=str2num(get(R00315Hz_Corr,'string'));
        amplitude_dB(1,2)=SPL_Level+corr;
        PlotTone_adj (315, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'Right', 1,0);
    end

R_00400Hz = uicontrol('style', 'pushbutton','string', '400 Hz','units',...
    'normalized','Visible','on','position', [R_Left VertSpacing(1,14) R_Width R_Height],...
    'FontSize', Font8,'callback', {@Right_00400Hz,SPL,R00400Hz_Corr});

    function Right_00400Hz(~,~,SPL,R00400Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corr=str2num(get(R00400Hz_Corr,'string'));
        amplitude_dB(1,2)=SPL_Level+corr;
        PlotTone_adj (400, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'Right', 1,0);
    end

R_00500Hz = uicontrol('style', 'pushbutton','string', '500 Hz','units',...
    'normalized','Visible','on','position', [R_Left VertSpacing(1,15) R_Width R_Height],...
    'FontSize', Font8,'callback', {@Right_00500Hz,SPL,R00500Hz_Corr});

    function Right_00500Hz(~,~,SPL,R00500Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corr=str2num(get(R00500Hz_Corr,'string'));
        amplitude_dB(1,2)=SPL_Level+corr;
        PlotTone_adj (500, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'Right', 1,0);
    end

R_00630Hz = uicontrol('style', 'pushbutton','string', '630 Hz','units',...
    'normalized','Visible','on','position', [R_Left VertSpacing(1,16) R_Width R_Height],...
    'FontSize', Font8,'callback', {@Right_00630Hz,SPL,R00630Hz_Corr});

    function Right_00630Hz(~,~,SPL,R00630Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corr=str2num(get(R00630Hz_Corr,'string'));
        amplitude_dB(1,2)=SPL_Level+corr;
        PlotTone_adj (630, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'Right', 1,0);
    end

R_00800Hz = uicontrol('style', 'pushbutton','string', '800 Hz','units',...
    'normalized','Visible','on','position', [R_Left VertSpacing(1,17) R_Width R_Height],...
    'FontSize', Font8,'callback', {@Right_00800Hz,SPL,R00800Hz_Corr});

    function Right_00800Hz(~,~,SPL,R00800Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corr=str2num(get(R00800Hz_Corr,'string'));
        amplitude_dB(1,2)=SPL_Level+corr;
        PlotTone_adj (800, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'Right', 1,0);
    end

R_01000Hz = uicontrol('style', 'pushbutton','string', '1000 Hz','units',...
    'normalized','Visible','on','position', [R_Left VertSpacing(1,18) R_Width R_Height],...
    'FontSize', Font8,'callback', {@Right_01000Hz,SPL,R01000Hz_Corr});

    function Right_01000Hz(~,~,SPL,R01000Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corr=str2num(get(R01000Hz_Corr,'string'));
%         a=Cal_Fade
%         b=Cal_Playback_Length*1000
%         c=SPL_Level+corr
        amplitude_dB(1,2)=SPL_Level+corr;
        PlotTone_adj (1000, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'Right', 1,0);
    end

R_01250Hz = uicontrol('style', 'pushbutton','string', '1250 Hz','units',...
    'normalized','Visible','on','position', [R_Left VertSpacing(1,19) R_Width R_Height],...
    'FontSize', Font8,'callback', {@Right_01250Hz,SPL,R01250Hz_Corr});

    function Right_01250Hz(~,~,SPL,R01250Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
```

```
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corr=str2num(get(R01250Hz_Corr,'string'));
        amplitude_dB(1,2)=SPL_Level+corr;
        PlotTone_adj (1250, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'Right', 1,0);
    end

R_01600Hz = uicontrol('style', 'pushbutton','string', '1600 Hz','units',...
    'normalized','Visible','on','position', [R_Left VertSpacing(1,20) R_Width R_Height],...
    'FontSize', Font8,'callback', {@Right_01600Hz,SPL,R01600Hz_Corr});

    function Right_01600Hz(~,~,SPL,R01600Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corr=str2num(get(R01600Hz_Corr,'string'));
        amplitude_dB(1,2)=SPL_Level+corr;
        PlotTone_adj (1600, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'Right', 1,0);
    end

R_02000Hz = uicontrol('style', 'pushbutton','string', '2000 Hz','units',...
    'normalized','Visible','on','position', [R_Left VertSpacing(1,21) R_Width R_Height],...
    'FontSize', Font8,'callback', {@Right_02000Hz,SPL,R02000Hz_Corr});

    function Right_02000Hz(~,~,SPL,R02000Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corr=str2num(get(R02000Hz_Corr,'string'));
        amplitude_dB(1,2)=SPL_Level+corr;
        PlotTone_adj (2000, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'Right', 1,0);
    end

R_02500Hz = uicontrol('style', 'pushbutton','string', '2500 Hz','units',...
    'normalized','Visible','on','position', [R_Left VertSpacing(1,22) R_Width R_Height],...
    'FontSize', Font8,'callback', {@Right_02500Hz,SPL,R02500Hz_Corr});

    function Right_02500Hz(~,~,SPL,R02500Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corr=str2num(get(R02500Hz_Corr,'string'));
        amplitude_dB(1,2)=SPL_Level+corr;
        PlotTone_adj (2500, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'Right', 1,0);
    end

R_03150Hz = uicontrol('style', 'pushbutton','string', '3150 Hz','units',...
    'normalized','Visible','on','position', [R_Left VertSpacing(1,23) R_Width R_Height],...
    'FontSize', Font8,'callback', {@Right_03150Hz,SPL,R03150Hz_Corr});

    function Right_03150Hz(~,~,SPL,R03150Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corr=str2num(get(R03150Hz_Corr,'string'));
        amplitude_dB(1,2)=SPL_Level+corr;
        PlotTone_adj (3150, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'Right', 1,0);
    end

R_04000Hz = uicontrol('style', 'pushbutton','string', '4000 Hz','units',...
    'normalized','Visible','on','position', [R_Left VertSpacing(1,24) R_Width R_Height],...
    'FontSize', Font8,'callback', {@Right_04000Hz,SPL,R04000Hz_Corr});

    function Right_04000Hz(~,~,SPL,R04000Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corr=str2num(get(R04000Hz_Corr,'string'));
        amplitude_dB(1,2)=SPL_Level+corr;
        PlotTone_adj (4000, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'Right', 1,0);
    end

R_05000Hz = uicontrol('style', 'pushbutton','string', '5000 Hz','units',...
    'normalized','Visible','on','position', [R_Left VertSpacing(1,25) R_Width R_Height],...
    'FontSize', Font8,'callback', {@Right_05000Hz,SPL,R05000Hz_Corr});

    function Right_05000Hz(~,~,SPL,R05000Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corr=str2num(get(R05000Hz_Corr,'string'));
        amplitude_dB(1,2)=SPL_Level+corr;
        PlotTone_adj (5000, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'Right', 1,0);
    end

R_06300Hz = uicontrol('style', 'pushbutton','string', '6300 Hz','units',...
    'normalized','Visible','on','position', [R_Left VertSpacing(1,26) R_Width R_Height],...
    'FontSize', Font8,'callback', {@Right_06300Hz,SPL,R06300Hz_Corr});

    function Right_06300Hz(~,~,SPL,R06300Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corr=str2num(get(R06300Hz_Corr,'string'));
        amplitude_dB(1,2)=SPL_Level+corr;
        PlotTone_adj (6300, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'Right', 1,0);
    end

R_08000Hz = uicontrol('style', 'pushbutton','string', '8000 Hz','units',...
    'normalized','Visible','on','position', [R_Left VertSpacing(1,27) R_Width R_Height],...
    'FontSize', Font8,'callback', {@Right_08000Hz,SPL,R08000Hz_Corr});

    function Right_08000Hz(~,~,SPL,R08000Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
```

```
        corr=str2num(get(R08000Hz_Corr,'string'));
        amplitude_dB(1,2)=SPL_Level+corr;
        PlotTone_adj (8000, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'Right', 1,0);
    end

R_10000Hz = uicontrol('style', 'pushbutton','string', '10000 Hz','units',...
    'normalized','Visible','on','position', [R_Left VertSpacing(1,28) R_Width R_Height],...
    'FontSize', Font8,'callback', {@Right_10000Hz,SPL,R10000Hz_Corr});

    function Right_10000Hz(~,~,SPL,R10000Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corr=str2num(get(R10000Hz_Corr,'string'));
        amplitude_dB(1,2)=SPL_Level+corr;
        PlotTone_adj (10000, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'Right', 1,0);
    end

R_12500Hz = uicontrol('style', 'pushbutton','string', '12500 Hz','units',...
    'normalized','Visible','on','position', [R_Left VertSpacing(1,29) R_Width R_Height],...
    'FontSize', Font8,'callback', {@Right_12500Hz,SPL,R12500Hz_Corr});

    function Right_12500Hz(~,~,SPL,R12500Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corr=str2num(get(R12500Hz_Corr,'string'));
        amplitude_dB(1,2)=SPL_Level+corr;
        PlotTone_adj (12500, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'Right', 1,0);
    end

R_16000Hz = uicontrol('style', 'pushbutton','string', '16000 Hz','units',...
    'normalized','Visible','on','position', [R_Left VertSpacing(1,30) R_Width R_Height],...
    'FontSize', Font8,'callback', {@Right_16000Hz,SPL,R16000Hz_Corr});

    function Right_16000Hz(~,~,SPL,R16000Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corr=str2num(get(R16000Hz_Corr,'string'));
        amplitude_dB(1,2)=SPL_Level+corr;
        PlotTone_adj (16000, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'Right', 1,0);
    end

R_20000Hz = uicontrol('style', 'pushbutton','string', '20000 Hz','units',...
    'normalized','Visible','on','position', [R_Left VertSpacing(1,31) R_Width R_Height],...
    'FontSize', Font8,'callback', {@Right_20000Hz,SPL,R20000Hz_Corr});

    function Right_20000Hz(~,~,SPL,R20000Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corr=str2num(get(R20000Hz_Corr,'string'));
        amplitude_dB(1,2)=SPL_Level+corr;
        PlotTone_adj (20000, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'Right', 1,0);
    end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% %%%%%%%%%%%%%%%%%     Binaural CALIBRATION VALUES      %%%%%%%%%%%%%%%%%


B00020Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [BC_Left VertSpacing(1,1) BC_Width BC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

B00025Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [BC_Left VertSpacing(1,2) BC_Width BC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

B000315Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [BC_Left VertSpacing(1,3) BC_Width BC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

B00040Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [BC_Left VertSpacing(1,4) BC_Width BC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

B00050Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [BC_Left VertSpacing(1,5) BC_Width BC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

B00063Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [BC_Left VertSpacing(1,6) BC_Width BC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

B00080Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [BC_Left VertSpacing(1,7) BC_Width BC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

B00100Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [BC_Left VertSpacing(1,8) BC_Width BC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

B00125Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [BC_Left VertSpacing(1,9) BC_Width BC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

B00160Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [BC_Left VertSpacing(1,10) BC_Width BC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

B00200Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [BC_Left VertSpacing(1,11) BC_Width BC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

B00250Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [BC_Left VertSpacing(1,12) BC_Width BC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);
```

```
B00315Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [BC_Left VertSpacing(1,13) BC_Width BC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

B00400Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [BC_Left VertSpacing(1,14) BC_Width BC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

B00500Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [BC_Left VertSpacing(1,15) BC_Width BC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

B00630Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [BC_Left VertSpacing(1,16) BC_Width BC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

B00800Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [BC_Left VertSpacing(1,17) BC_Width BC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

B01000Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [BC_Left VertSpacing(1,18) BC_Width BC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

B01250Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [BC_Left VertSpacing(1,19) BC_Width BC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

B01600Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [BC_Left VertSpacing(1,20) BC_Width BC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

B02000Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [BC_Left VertSpacing(1,21) BC_Width BC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

B02500Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [BC_Left VertSpacing(1,22) BC_Width BC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

B03150Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [BC_Left VertSpacing(1,23) BC_Width BC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

B04000Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [BC_Left VertSpacing(1,24) BC_Width BC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

B05000Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [BC_Left VertSpacing(1,25) BC_Width BC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

B06300Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [BC_Left VertSpacing(1,26) BC_Width BC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

B08000Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [BC_Left VertSpacing(1,27) BC_Width BC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

B10000Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [BC_Left VertSpacing(1,28) BC_Width BC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

B12500Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [BC_Left VertSpacing(1,29) BC_Width BC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

B16000Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [BC_Left VertSpacing(1,30) BC_Width BC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);

B20000Hz_Corr = uicontrol('style','edit', 'units', 'normalized',...
    'position', [BC_Left VertSpacing(1,31) BC_Width BC_Height], 'string', 0.0,...
    'visible','on', 'FontSize', Font8);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% %%%%%%%%%%%%%%%%%     Binaual   PUSHBUTTONS            %%%%%%%%%%%%%%%%%

B_00020Hz = uicontrol('style', 'pushbutton','string', '20 Hz','units',...
    'normalized','Visible','on','position', [B_Left VertSpacing(1,1) B_Width B_Height],...
    'FontSize', Font8,'callback', {@Binaural_00020Hz,SPL,B00020Hz_Corr,L00020Hz_Corr,R00020Hz_Corr});

    function Binaural_00020Hz(~,~,SPL,B00020Hz_Corr,L00020Hz_Corr,R00020Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corrL=str2num(get(B00020Hz_Corr,'string'))+str2num(get(L00020Hz_Corr,'string'));
        corrR=str2num(get(B00020Hz_Corr,'string'))+str2num(get(R00020Hz_Corr,'string'));
        amplitude_dB(1,1)=SPL_Level+corrL;
        amplitude_dB(1,2)=SPL_Level+corrR;
        PlotTone_adj (20, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'Binaural', 1,0);
    end

B_00025Hz = uicontrol('style', 'pushbutton','string', '25 Hz','units',...
    'normalized','Visible','on','position', [B_Left VertSpacing(1,2) B_Width B_Height],...
    'FontSize', Font8,'callback', {@Binaural_00025Hz,SPL,B00025Hz_Corr,L00025Hz_Corr,R00025Hz_Corr});

    function Binaural_00025Hz(~,~,SPL,B00025Hz_Corr,L00025Hz_Corr,R00025Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corrL=str2num(get(B00025Hz_Corr,'string'))+str2num(get(L00025Hz_Corr,'string'));
        corrR=str2num(get(B00025Hz_Corr,'string'))+str2num(get(R00025Hz_Corr,'string'));
        amplitude_dB(1,1)=SPL_Level+corrL;
        amplitude_dB(1,2)=SPL_Level+corrR;
        PlotTone_adj (25, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'Binaural', 1,0);
    end

B_000315Hz = uicontrol('style', 'pushbutton','string', '31.5 Hz','units',...
    'normalized','Visible','on','position', [B_Left VertSpacing(1,3) B_Width B_Height],...
    'FontSize', Font8,'callback', {@Binaural_000315Hz,SPL,B000315Hz_Corr,L000315Hz_Corr,R000315Hz_Corr});
```

```
    function Binaural_000315Hz(~,~,SPL,B000315Hz_Corr,L000315Hz_Corr,R000315Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corrL=str2num(get(B000315Hz_Corr,'string'))+str2num(get(L000315Hz_Corr,'string'));
        corrR=str2num(get(B000315Hz_Corr,'string'))+str2num(get(R000315Hz_Corr,'string'));
        amplitude_dB(1,1)=SPL_Level+corrL;
        amplitude_dB(1,2)=SPL_Level+corrR;
        PlotTone_adj (31.5, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'Binaural', 1,0);
    end

B_00040Hz = uicontrol('style', 'pushbutton','string', '40 Hz','units',...
    'normalized','Visible','on','position', [B_Left VertSpacing(1,4) B_Width B_Height],...
    'FontSize', Font8,'callback', {@Binaural_00040Hz,SPL,B00040Hz_Corr,L00040Hz_Corr,R00040Hz_Corr});

    function Binaural_00040Hz(~,~,SPL,B00040Hz_Corr,L00040Hz_Corr,R00040Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corrL=str2num(get(B00040Hz_Corr,'string'))+str2num(get(L00040Hz_Corr,'string'));
        corrR=str2num(get(B00040Hz_Corr,'string'))+str2num(get(R00040Hz_Corr,'string'));
        amplitude_dB(1,1)=SPL_Level+corrL;
        amplitude_dB(1,2)=SPL_Level+corrR;
        PlotTone_adj (40, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'Binaural', 1,0);
    end

B_00050Hz = uicontrol('style', 'pushbutton','string', '50 Hz','units',...
    'normalized','Visible','on','position', [B_Left VertSpacing(1,5) B_Width B_Height],...
    'FontSize', Font8,'callback', {@Binaural_00050Hz,SPL,B00050Hz_Corr,L00050Hz_Corr,R00050Hz_Corr});

    function Binaural_00050Hz(~,~,SPL,B00050Hz_Corr,L00050Hz_Corr,R00050Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corrL=str2num(get(B00050Hz_Corr,'string'))+str2num(get(L00050Hz_Corr,'string'));
        corrR=str2num(get(B00050Hz_Corr,'string'))+str2num(get(R00050Hz_Corr,'string'));
        amplitude_dB(1,1)=SPL_Level+corrL;
        amplitude_dB(1,2)=SPL_Level+corrR;
        PlotTone_adj (50, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'Binaural', 1,0);
    end

B_00063Hz = uicontrol('style', 'pushbutton','string', '63 Hz','units',...
    'normalized','Visible','on','position', [B_Left VertSpacing(1,6) B_Width B_Height],...
    'FontSize', Font8,'callback', {@Binaural_00063Hz,SPL,B00063Hz_Corr,L00063Hz_Corr,R00063Hz_Corr});

    function Binaural_00063Hz(~,~,SPL,B00063Hz_Corr,L00063Hz_Corr,R00063Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corrL=str2num(get(B00063Hz_Corr,'string'))+str2num(get(L00063Hz_Corr,'string'));
        corrR=str2num(get(B00063Hz_Corr,'string'))+str2num(get(R00063Hz_Corr,'string'));
        amplitude_dB(1,1)=SPL_Level+corrL;
        amplitude_dB(1,2)=SPL_Level+corrR;
        PlotTone_adj (63, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'Binaural', 1,0);
    end

B_00080Hz = uicontrol('style', 'pushbutton','string', '80 Hz','units',...
    'normalized','Visible','on','position', [B_Left VertSpacing(1,7) B_Width B_Height],...
    'FontSize', Font8,'callback', {@Binaural_00080Hz,SPL,B00080Hz_Corr,L00080Hz_Corr,R00080Hz_Corr});

    function Binaural_00080Hz(~,~,SPL,B00080Hz_Corr,L00080Hz_Corr,R00080Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corrL=str2num(get(B00080Hz_Corr,'string'))+str2num(get(L00080Hz_Corr,'string'));
        corrR=str2num(get(B00080Hz_Corr,'string'))+str2num(get(R00080Hz_Corr,'string'));
        amplitude_dB(1,1)=SPL_Level+corrL;
        amplitude_dB(1,2)=SPL_Level+corrR;
        PlotTone_adj (80, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'Binaural', 1,0);
    end

B_00100Hz = uicontrol('style', 'pushbutton','string', '100 Hz','units',...
    'normalized','Visible','on','position', [B_Left VertSpacing(1,8) B_Width B_Height],...
    'FontSize', Font8,'callback', {@Binaural_00100Hz,SPL,B00100Hz_Corr,L00100Hz_Corr,R00100Hz_Corr});

    function Binaural_00100Hz(~,~,SPL,B00100Hz_Corr,L00100Hz_Corr,R00100Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corrL=str2num(get(B00100Hz_Corr,'string'))+str2num(get(L00100Hz_Corr,'string'));
        corrR=str2num(get(B00100Hz_Corr,'string'))+str2num(get(R00100Hz_Corr,'string'));
        amplitude_dB(1,1)=SPL_Level+corrL;
        amplitude_dB(1,2)=SPL_Level+corrR;
        PlotTone_adj (100, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'Binaural', 1,0);
    end

B_00125Hz = uicontrol('style', 'pushbutton','string', '125 Hz','units',...
    'normalized','Visible','on','position', [B_Left VertSpacing(1,9) B_Width B_Height],...
    'FontSize', Font8,'callback', {@Binaural_00125Hz,SPL,B00125Hz_Corr,L00125Hz_Corr,R00125Hz_Corr});

    function Binaural_00125Hz(~,~,SPL,B00125Hz_Corr,L00125Hz_Corr,R00125Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corrL=str2num(get(B00125Hz_Corr,'string'))+str2num(get(L00125Hz_Corr,'string'));
        corrR=str2num(get(B00125Hz_Corr,'string'))+str2num(get(R00125Hz_Corr,'string'));
        amplitude_dB(1,1)=SPL_Level+corrL;
        amplitude_dB(1,2)=SPL_Level+corrR;
        PlotTone_adj (125, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'Binaural', 1,0);
    end

B_00160Hz = uicontrol('style', 'pushbutton','string', '160 Hz','units',...
    'normalized','Visible','on','position', [B_Left VertSpacing(1,10) B_Width B_Height],...
    'FontSize', Font8,'callback', {@Binaural_00160Hz,SPL,B00160Hz_Corr,L00160Hz_Corr,R00160Hz_Corr});

    function Binaural_00160Hz(~,~,SPL,B00160Hz_Corr,L00160Hz_Corr,R00160Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
```

```
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corrL=str2num(get(B00160Hz_Corr,'string'))+str2num(get(L00160Hz_Corr,'string'));
        corrR=str2num(get(B00160Hz_Corr,'string'))+str2num(get(R00160Hz_Corr,'string'));
        amplitude_dB(1,1)=SPL_Level+corrL;
        amplitude_dB(1,2)=SPL_Level+corrR;
        PlotTone_adj (160, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'Binaural', 1,0);
    end

B_00200Hz = uicontrol('style', 'pushbutton','string', '200 Hz','units',...
    'normalized','Visible','on','position', [B_Left VertSpacing(1,11) B_Width B_Height],...
    'FontSize', Font8,'callback', {@Binaural_00200Hz,SPL,B00200Hz_Corr,L00200Hz_Corr,R00200Hz_Corr});

    function Binaural_00200Hz(~,~,SPL,B00200Hz_Corr,L00200Hz_Corr,R00200Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corrL=str2num(get(B00200Hz_Corr,'string'))+str2num(get(L00200Hz_Corr,'string'));
        corrR=str2num(get(B00200Hz_Corr,'string'))+str2num(get(R00200Hz_Corr,'string'));
        amplitude_dB(1,1)=SPL_Level+corrL;
        amplitude_dB(1,2)=SPL_Level+corrR;
        PlotTone_adj (200, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'Binaural', 1,0);
    end

B_00250Hz = uicontrol('style', 'pushbutton','string', '250 Hz','units',...
    'normalized','Visible','on','position', [B_Left VertSpacing(1,12) B_Width B_Height],...
    'FontSize', Font8,'callback', {@Binaural_00250Hz,SPL,B00250Hz_Corr,L00250Hz_Corr,R00250Hz_Corr});

    function Binaural_00250Hz(~,~,SPL,B00250Hz_Corr,L00250Hz_Corr,R00250Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corrL=str2num(get(B00250Hz_Corr,'string'))+str2num(get(L00250Hz_Corr,'string'));
        corrR=str2num(get(B00250Hz_Corr,'string'))+str2num(get(R00250Hz_Corr,'string'));
        amplitude_dB(1,1)=SPL_Level+corrL;
        amplitude_dB(1,2)=SPL_Level+corrR;
        PlotTone_adj (250, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'Binaural', 1,0);
    end

B_00315Hz = uicontrol('style', 'pushbutton','string', '315 Hz','units',...
    'normalized','Visible','on','position', [B_Left VertSpacing(1,13) B_Width B_Height],...
    'FontSize', Font8,'callback', {@Binaural_00315Hz,SPL,B00315Hz_Corr,L00315Hz_Corr,R00315Hz_Corr});

    function Binaural_00315Hz(~,~,SPL,B00315Hz_Corr,L00315Hz_Corr,R00315Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corrL=str2num(get(B00315Hz_Corr,'string'))+str2num(get(L00315Hz_Corr,'string'));
        corrR=str2num(get(B00315Hz_Corr,'string'))+str2num(get(R00315Hz_Corr,'string'));
        amplitude_dB(1,1)=SPL_Level+corrL;
        amplitude_dB(1,2)=SPL_Level+corrR;
        PlotTone_adj (315, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'Binaural', 1,0);
    end

B_00400Hz = uicontrol('style', 'pushbutton','string', '400 Hz','units',...
    'normalized','Visible','on','position', [B_Left VertSpacing(1,14) B_Width B_Height],...
    'FontSize', Font8,'callback', {@Binaural_00400Hz,SPL,B00400Hz_Corr,L00400Hz_Corr,R00400Hz_Corr});

    function Binaural_00400Hz(~,~,SPL,B00400Hz_Corr,L00400Hz_Corr,R00400Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corrL=str2num(get(B00400Hz_Corr,'string'))+str2num(get(L00400Hz_Corr,'string'));
        corrR=str2num(get(B00400Hz_Corr,'string'))+str2num(get(R00400Hz_Corr,'string'));
        amplitude_dB(1,1)=SPL_Level+corrL;
        amplitude_dB(1,2)=SPL_Level+corrR;
        PlotTone_adj (400, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'Binaural', 1,0);
    end

B_00500Hz = uicontrol('style', 'pushbutton','string', '500 Hz','units',...
    'normalized','Visible','on','position', [B_Left VertSpacing(1,15) B_Width B_Height],...
    'FontSize', Font8,'callback', {@Binaural_00500Hz,SPL,B00500Hz_Corr,L00500Hz_Corr,R00500Hz_Corr});

    function Binaural_00500Hz(~,~,SPL,B00500Hz_Corr,L00500Hz_Corr,R00500Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corrL=str2num(get(B00500Hz_Corr,'string'))+str2num(get(L00500Hz_Corr,'string'));
        corrR=str2num(get(B00500Hz_Corr,'string'))+str2num(get(R00500Hz_Corr,'string'));
        amplitude_dB(1,1)=SPL_Level+corrL;
        amplitude_dB(1,2)=SPL_Level+corrR;
        PlotTone_adj (500, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'Binaural', 1,0);
    end

B_00630Hz = uicontrol('style', 'pushbutton','string', '630 Hz','units',...
    'normalized','Visible','on','position', [B_Left VertSpacing(1,16) B_Width B_Height],...
    'FontSize', Font8,'callback', {@Binaural_00630Hz,SPL,B00630Hz_Corr,L00630Hz_Corr,R00630Hz_Corr});

    function Binaural_00630Hz(~,~,SPL,B00630Hz_Corr,L00630Hz_Corr,R00630Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corrL=str2num(get(B00630Hz_Corr,'string'))+str2num(get(L00630Hz_Corr,'string'));
        corrR=str2num(get(B00630Hz_Corr,'string'))+str2num(get(R00630Hz_Corr,'string'));
        amplitude_dB(1,1)=SPL_Level+corrL;
        amplitude_dB(1,2)=SPL_Level+corrR;
        PlotTone_adj (630, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'Binaural', 1,0);
    end

B_00800Hz = uicontrol('style', 'pushbutton','string', '800 Hz','units',...
    'normalized','Visible','on','position', [B_Left VertSpacing(1,17) B_Width B_Height],...
    'FontSize', Font8,'callback', {@Binaural_00800Hz,SPL,B00800Hz_Corr,L00800Hz_Corr,R00800Hz_Corr});

    function Binaural_00800Hz(~,~,SPL,B00800Hz_Corr,L00800Hz_Corr,R00800Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
```

```
        SPL_Level=str2num(get(SPL,'string'));
        corrL=str2num(get(B00800Hz_Corr,'string'))+str2num(get(L00800Hz_Corr,'string'));
        corrR=str2num(get(B00800Hz_Corr,'string'))+str2num(get(R00800Hz_Corr,'string'));
        amplitude_dB(1,1)=SPL_Level+corrL;
        amplitude_dB(1,2)=SPL_Level+corrR;
        PlotTone_adj (800, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'Binaural', 1,0);
    end

B_01000Hz = uicontrol('style', 'pushbutton','string', '1000 Hz','units',...
    'normalized','Visible','on','position', [B_Left VertSpacing(1,18) B_Width B_Height],...
    'FontSize', Font8,'callback', {@Binaural_01000Hz,SPL,B01000Hz_Corr,L01000Hz_Corr,R01000Hz_Corr});

    function Binaural_01000Hz(~,~,SPL,B01000Hz_Corr,L01000Hz_Corr,R01000Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corrL=str2num(get(B01000Hz_Corr,'string'))+str2num(get(L01000Hz_Corr,'string'));
        corrR=str2num(get(B01000Hz_Corr,'string'))+str2num(get(R01000Hz_Corr,'string'));
        amplitude_dB(1,1)=SPL_Level+corrL;
        amplitude_dB(1,2)=SPL_Level+corrR;
%         a=Cal_Fade
%         b=Cal_Playback_Length*1000
%         c=SPL_Level+corr
        PlotTone_adj (1000, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'Binaural', 1,0);
    end

B_01250Hz = uicontrol('style', 'pushbutton','string', '1250 Hz','units',...
    'normalized','Visible','on','position', [B_Left VertSpacing(1,19) B_Width B_Height],...
    'FontSize', Font8,'callback', {@Binaural_01250Hz,SPL,B01250Hz_Corr,L01250Hz_Corr,R01250Hz_Corr});

    function Binaural_01250Hz(~,~,SPL,B01250Hz_Corr,L01250Hz_Corr,R01250Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corrL=str2num(get(B01250Hz_Corr,'string'))+str2num(get(L01250Hz_Corr,'string'));
        corrR=str2num(get(B01250Hz_Corr,'string'))+str2num(get(R01250Hz_Corr,'string'));
        amplitude_dB(1,1)=SPL_Level+corrL;
        amplitude_dB(1,2)=SPL_Level+corrR;
        PlotTone_adj (1250, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'Binaural', 1,0);
    end

B_01600Hz = uicontrol('style', 'pushbutton','string', '1600 Hz','units',...
    'normalized','Visible','on','position', [B_Left VertSpacing(1,20) B_Width B_Height],...
    'FontSize', Font8,'callback', {@Binaural_01600Hz,SPL,B01600Hz_Corr,L01600Hz_Corr,R01600Hz_Corr});

    function Binaural_01600Hz(~,~,SPL,B01600Hz_Corr,L01600Hz_Corr,R01600Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corrL=str2num(get(B01600Hz_Corr,'string'))+str2num(get(L01600Hz_Corr,'string'));
        corrR=str2num(get(B01600Hz_Corr,'string'))+str2num(get(R01600Hz_Corr,'string'));
        amplitude_dB(1,1)=SPL_Level+corrL;
        amplitude_dB(1,2)=SPL_Level+corrR;
        PlotTone_adj (1600, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'Binaural', 1,0);
    end

B_02000Hz = uicontrol('style', 'pushbutton','string', '2000 Hz','units',...
    'normalized','Visible','on','position', [B_Left VertSpacing(1,21) B_Width B_Height],...
    'FontSize', Font8,'callback', {@Binaural_02000Hz,SPL,B02000Hz_Corr,L02000Hz_Corr,R02000Hz_Corr});

    function Binaural_02000Hz(~,~,SPL,B02000Hz_Corr,L02000Hz_Corr,R02000Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corrL=str2num(get(B02000Hz_Corr,'string'))+str2num(get(L02000Hz_Corr,'string'));
        corrR=str2num(get(B02000Hz_Corr,'string'))+str2num(get(R02000Hz_Corr,'string'));
        amplitude_dB(1,1)=SPL_Level+corrL;
        amplitude_dB(1,2)=SPL_Level+corrR;
        PlotTone_adj (2000, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'Binaural', 1,0);
    end

B_02500Hz = uicontrol('style', 'pushbutton','string', '2500 Hz','units',...
    'normalized','Visible','on','position', [B_Left VertSpacing(1,22) B_Width B_Height],...
    'FontSize', Font8,'callback', {@Binaural_02500Hz,SPL,B02500Hz_Corr,L02500Hz_Corr,R02500Hz_Corr});

    function Binaural_02500Hz(~,~,SPL,B02500Hz_Corr,L02500Hz_Corr,R02500Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corrL=str2num(get(B02500Hz_Corr,'string'))+str2num(get(L02500Hz_Corr,'string'));
        corrR=str2num(get(B02500Hz_Corr,'string'))+str2num(get(R02500Hz_Corr,'string'));
        amplitude_dB(1,1)=SPL_Level+corrL;
        amplitude_dB(1,2)=SPL_Level+corrR;
        PlotTone_adj (2500, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'Binaural', 1,0);
    end

B_03150Hz = uicontrol('style', 'pushbutton','string', '3150 Hz','units',...
    'normalized','Visible','on','position', [B_Left VertSpacing(1,23) B_Width B_Height],...
    'FontSize', Font8,'callback', {@Binaural_03150Hz,SPL,B03150Hz_Corr,L03150Hz_Corr,R03150Hz_Corr});

    function Binaural_03150Hz(~,~,SPL,B03150Hz_Corr,L03150Hz_Corr,R03150Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corrL=str2num(get(B03150Hz_Corr,'string'))+str2num(get(L03150Hz_Corr,'string'));
        corrR=str2num(get(B03150Hz_Corr,'string'))+str2num(get(R03150Hz_Corr,'string'));
        amplitude_dB(1,1)=SPL_Level+corrL;
        amplitude_dB(1,2)=SPL_Level+corrR;
        PlotTone_adj (3150, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'Binaural', 1,0);
    end

B_04000Hz = uicontrol('style', 'pushbutton','string', '4000 Hz','units',...
    'normalized','Visible','on','position', [B_Left VertSpacing(1,24) B_Width B_Height],...
    'FontSize', Font8,'callback', {@Binaural_04000Hz,SPL,B04000Hz_Corr,L04000Hz_Corr,R04000Hz_Corr});

    function Binaural_04000Hz(~,~,SPL,B04000Hz_Corr,L04000Hz_Corr,R04000Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
```

```
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corrL=str2num(get(B04000Hz_Corr,'string'))+str2num(get(L04000Hz_Corr,'string'));
        corrR=str2num(get(B04000Hz_Corr,'string'))+str2num(get(R04000Hz_Corr,'string'));
        amplitude_dB(1,1)=SPL_Level+corrL;
        amplitude_dB(1,2)=SPL_Level+corrR;
        PlotTone_adj (4000, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'Binaural', 1,0);
    end

B_05000Hz = uicontrol('style', 'pushbutton','string', '5000 Hz','units',...
    'normalized','Visible','on','position', [B_Left VertSpacing(1,25) B_Width B_Height],...
    'FontSize', Font8,'callback', {@Binaural_05000Hz,SPL,B05000Hz_Corr,L05000Hz_Corr,R05000Hz_Corr});

    function Binaural_05000Hz(~,~,SPL,B05000Hz_Corr,L05000Hz_Corr,R05000Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corrL=str2num(get(B05000Hz_Corr,'string'))+str2num(get(L05000Hz_Corr,'string'));
        corrR=str2num(get(B05000Hz_Corr,'string'))+str2num(get(R05000Hz_Corr,'string'));
        amplitude_dB(1,1)=SPL_Level+corrL;
        amplitude_dB(1,2)=SPL_Level+corrR;
        PlotTone_adj (5000, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'Binaural', 1,0);
    end

B_06300Hz = uicontrol('style', 'pushbutton','string', '6300 Hz','units',...
    'normalized','Visible','on','position', [B_Left VertSpacing(1,26) B_Width B_Height],...
    'FontSize', Font8,'callback', {@Binaural_06300Hz,SPL,B06300Hz_Corr,L06300Hz_Corr,R06300Hz_Corr});

    function Binaural_06300Hz(~,~,SPL,B06300Hz_Corr,L06300Hz_Corr,R06300Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corrL=str2num(get(B06300Hz_Corr,'string'))+str2num(get(L06300Hz_Corr,'string'));
        corrR=str2num(get(B06300Hz_Corr,'string'))+str2num(get(R06300Hz_Corr,'string'));
        amplitude_dB(1,1)=SPL_Level+corrL;
        amplitude_dB(1,2)=SPL_Level+corrR;
        PlotTone_adj (6300, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'Binaural', 1,0);
    end

B_08000Hz = uicontrol('style', 'pushbutton','string', '8000 Hz','units',...
    'normalized','Visible','on','position', [B_Left VertSpacing(1,27) B_Width B_Height],...
    'FontSize', Font8,'callback', {@Binaural_08000Hz,SPL,B08000Hz_Corr,L08000Hz_Corr,R08000Hz_Corr});

    function Binaural_08000Hz(~,~,SPL,B08000Hz_Corr,L08000Hz_Corr,R08000Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corrL=str2num(get(B08000Hz_Corr,'string'))+str2num(get(L08000Hz_Corr,'string'));
        corrR=str2num(get(B08000Hz_Corr,'string'))+str2num(get(R08000Hz_Corr,'string'));
        amplitude_dB(1,1)=SPL_Level+corrL;
        amplitude_dB(1,2)=SPL_Level+corrR;
        PlotTone_adj (8000, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'Binaural', 1,0);
    end

B_10000Hz = uicontrol('style', 'pushbutton','string', '10000 Hz','units',...
    'normalized','Visible','on','position', [B_Left VertSpacing(1,28) B_Width B_Height],...
    'FontSize', Font8,'callback', {@Binaural_10000Hz,SPL,B10000Hz_Corr,L10000Hz_Corr,R10000Hz_Corr});

    function Binaural_10000Hz(~,~,SPL,B10000Hz_Corr,L10000Hz_Corr,R10000Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corrL=str2num(get(B10000Hz_Corr,'string'))+str2num(get(L10000Hz_Corr,'string'));
        corrR=str2num(get(B10000Hz_Corr,'string'))+str2num(get(R10000Hz_Corr,'string'));
        amplitude_dB(1,1)=SPL_Level+corrL;
        amplitude_dB(1,2)=SPL_Level+corrR;
        PlotTone_adj (10000, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'Binaural', 1,0);
    end

B_12500Hz = uicontrol('style', 'pushbutton','string', '12500 Hz','units',...
    'normalized','Visible','on','position', [B_Left VertSpacing(1,29) B_Width B_Height],...
    'FontSize', Font8,'callback', {@Binaural_12500Hz,SPL,B12500Hz_Corr,L12500Hz_Corr,R12500Hz_Corr});

    function Binaural_12500Hz(~,~,SPL,B12500Hz_Corr,L12500Hz_Corr,R12500Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corrL=str2num(get(B12500Hz_Corr,'string'))+str2num(get(L12500Hz_Corr,'string'));
        corrR=str2num(get(B12500Hz_Corr,'string'))+str2num(get(R12500Hz_Corr,'string'));
        amplitude_dB(1,1)=SPL_Level+corrL;
        amplitude_dB(1,2)=SPL_Level+corrR;
        PlotTone_adj (12500, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'Binaural', 1,0);
    end

B_16000Hz = uicontrol('style', 'pushbutton','string', '16000 Hz','units',...
    'normalized','Visible','on','position', [B_Left VertSpacing(1,30) B_Width B_Height],...
    'FontSize', Font8,'callback', {@Binaural_16000Hz,SPL,B16000Hz_Corr,L16000Hz_Corr,R16000Hz_Corr});

    function Binaural_16000Hz(~,~,SPL,B16000Hz_Corr,L16000Hz_Corr,R16000Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
        corrL=str2num(get(B16000Hz_Corr,'string'))+str2num(get(L16000Hz_Corr,'string'));
        corrR=str2num(get(B16000Hz_Corr,'string'))+str2num(get(R16000Hz_Corr,'string'));
        amplitude_dB(1,1)=SPL_Level+corrL;
        amplitude_dB(1,2)=SPL_Level+corrR;
        PlotTone_adj (16000, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'Binaural', 1,0);
    end

B_20000Hz = uicontrol('style', 'pushbutton','string', '20000 Hz','units',...
    'normalized','Visible','on','position', [B_Left VertSpacing(1,31) B_Width B_Height],...
    'FontSize', Font8,'callback', {@Binaural_20000Hz,SPL,B20000Hz_Corr,L20000Hz_Corr,R20000Hz_Corr});

    function Binaural_20000Hz(~,~,SPL,B20000Hz_Corr,L20000Hz_Corr,R20000Hz_Corr)
        CurrentSettings= getappdata(0,'evalue');
        Cal_Playback_Length = CurrentSettings(9,4);
        Cal_Fade = CurrentSettings(10,4);
        SPL_Level=str2num(get(SPL,'string'));
```

```
        corrL=str2num(get(B20000Hz_Corr,'string'))+str2num(get(L20000Hz_Corr,'string'));
        corrR=str2num(get(B20000Hz_Corr,'string'))+str2num(get(R20000Hz_Corr,'string'));
        amplitude_dB(1,1)=SPL_Level+corrL;
        amplitude_dB(1,2)=SPL_Level+corrR;
        PlotTone_adj (20000, Cal_Fade, Cal_Playback_Length*1000, amplitude_dB, 'Binaural', 1,0);
    end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Previously Loaded Data
        set(L00020Hz_Corr,'string',Cal_File(1,7));
        set(L00025Hz_Corr,'string',Cal_File(2,7));
        set(L000315Hz_Corr,'string',Cal_File(3,7));
        set(L00040Hz_Corr,'string',Cal_File(4,7));
        set(L00050Hz_Corr,'string',Cal_File(5,7));
        set(L00063Hz_Corr,'string',Cal_File(6,7));
        set(L00080Hz_Corr,'string',Cal_File(7,7));
        set(L00100Hz_Corr,'string',Cal_File(8,7));
        set(L00125Hz_Corr,'string',Cal_File(9,7));
        set(L00160Hz_Corr,'string',Cal_File(10,7));
        set(L00200Hz_Corr,'string',Cal_File(11,7));
        set(L00250Hz_Corr,'string',Cal_File(12,7));
        set(L00315Hz_Corr,'string',Cal_File(13,7));
        set(L00400Hz_Corr,'string',Cal_File(14,7));
        set(L00500Hz_Corr,'string',Cal_File(15,7));
        set(L00630Hz_Corr,'string',Cal_File(16,7));
        set(L00800Hz_Corr,'string',Cal_File(17,7));
        set(L01000Hz_Corr,'string',Cal_File(18,7));
        set(L01250Hz_Corr,'string',Cal_File(19,7));
        set(L01600Hz_Corr,'string',Cal_File(20,7));
        set(L02000Hz_Corr,'string',Cal_File(21,7));
        set(L02500Hz_Corr,'string',Cal_File(22,7));
        set(L03150Hz_Corr,'string',Cal_File(23,7));
        set(L04000Hz_Corr,'string',Cal_File(24,7));
        set(L05000Hz_Corr,'string',Cal_File(25,7));
        set(L06300Hz_Corr,'string',Cal_File(26,7));
        set(L08000Hz_Corr,'string',Cal_File(27,7));
        set(L10000Hz_Corr,'string',Cal_File(28,7));
        set(L12500Hz_Corr,'string',Cal_File(29,7));
        set(L16000Hz_Corr,'string',Cal_File(30,7));
        set(L20000Hz_Corr,'string',Cal_File(31,7));

        set(R00020Hz_Corr,'string',Cal_File(1,8));
        set(R00025Hz_Corr,'string',Cal_File(2,8));
        set(R000315Hz_Corr,'string',Cal_File(3,8));
        set(R00040Hz_Corr,'string',Cal_File(4,8));
        set(R00050Hz_Corr,'string',Cal_File(5,8));
        set(R00063Hz_Corr,'string',Cal_File(6,8));
        set(R00080Hz_Corr,'string',Cal_File(7,8));
        set(R00100Hz_Corr,'string',Cal_File(8,8));
        set(R00125Hz_Corr,'string',Cal_File(9,8));
        set(R00160Hz_Corr,'string',Cal_File(10,8));
        set(R00200Hz_Corr,'string',Cal_File(11,8));
        set(R00250Hz_Corr,'string',Cal_File(12,8));
        set(R00315Hz_Corr,'string',Cal_File(13,8));
        set(R00400Hz_Corr,'string',Cal_File(14,8));
        set(R00500Hz_Corr,'string',Cal_File(15,8));
        set(R00630Hz_Corr,'string',Cal_File(16,8));
        set(R00800Hz_Corr,'string',Cal_File(17,8));
        set(R01000Hz_Corr,'string',Cal_File(18,8));
        set(R01250Hz_Corr,'string',Cal_File(19,8));
        set(R01600Hz_Corr,'string',Cal_File(20,8));
        set(R02000Hz_Corr,'string',Cal_File(21,8));
        set(R02500Hz_Corr,'string',Cal_File(22,8));
        set(R03150Hz_Corr,'string',Cal_File(23,8));
        set(R04000Hz_Corr,'string',Cal_File(24,8));
        set(R05000Hz_Corr,'string',Cal_File(25,8));
        set(R06300Hz_Corr,'string',Cal_File(26,8));
        set(R08000Hz_Corr,'string',Cal_File(27,8));
        set(R10000Hz_Corr,'string',Cal_File(28,8));
        set(R12500Hz_Corr,'string',Cal_File(29,8));
        set(R16000Hz_Corr,'string',Cal_File(30,8));
        set(R20000Hz_Corr,'string',Cal_File(31,8));

        set(B00020Hz_Corr,'string',Cal_File(1,15));
        set(B00025Hz_Corr,'string',Cal_File(2,15));
        set(B000315Hz_Corr,'string',Cal_File(3,15));
        set(B00040Hz_Corr,'string',Cal_File(4,15));
        set(B00050Hz_Corr,'string',Cal_File(5,15));
        set(B00063Hz_Corr,'string',Cal_File(6,15));
        set(B00080Hz_Corr,'string',Cal_File(7,15));
        set(B00100Hz_Corr,'string',Cal_File(8,15));
        set(B00125Hz_Corr,'string',Cal_File(9,15));
        set(B00160Hz_Corr,'string',Cal_File(10,15));
        set(B00200Hz_Corr,'string',Cal_File(11,15));
        set(B00250Hz_Corr,'string',Cal_File(12,15));
        set(B00315Hz_Corr,'string',Cal_File(13,15));
        set(B00400Hz_Corr,'string',Cal_File(14,15));
        set(B00500Hz_Corr,'string',Cal_File(15,15));
        set(B00630Hz_Corr,'string',Cal_File(16,15));
        set(B00800Hz_Corr,'string',Cal_File(17,15));
        set(B01000Hz_Corr,'string',Cal_File(18,15));
        set(B01250Hz_Corr,'string',Cal_File(19,15));
        set(B01600Hz_Corr,'string',Cal_File(20,15));
        set(B02000Hz_Corr,'string',Cal_File(21,15));
        set(B02500Hz_Corr,'string',Cal_File(22,15));
        set(B03150Hz_Corr,'string',Cal_File(23,15));
        set(B04000Hz_Corr,'string',Cal_File(24,15));
        set(B05000Hz_Corr,'string',Cal_File(25,15));
        set(B06300Hz_Corr,'string',Cal_File(26,15));
        set(B08000Hz_Corr,'string',Cal_File(27,15));
        set(B10000Hz_Corr,'string',Cal_File(28,15));
        set(B12500Hz_Corr,'string',Cal_File(29,15));
        set(B16000Hz_Corr,'string',Cal_File(30,15));
        set(B20000Hz_Corr,'string',Cal_File(31,15));
%% %%%%%%%%%%%%%%%%%     BUTTON POSITIONS                 %%%%%%%%%%%%%%%%%%
Spacing = 0.01;  BW = 0.2;
BH = 0.04;
Gap=0.5*(1-(R_RGroup+W_Button)-BW);
BL=SPL_Group;
LBB = VertSpacing(1,TopRow-10); %Lower Button Bottom
DBH = BH + Spacing; %Delta for next button up
%% %%%%%%%%%%%%%%%%%     Calibrations Settings Button     %%%%%%%%%%%%%%%%%%
W=0.1;
```

```
G=0.5*(BW-W);

LOAD = uicontrol('style', 'pushbutton','string', 'Settings','units',...
    'normalized','Visible','on','position', [BL LBB+3*DBH BW BH],...
    'FontSize', Font10, 'FontWeight', 'bold','callback', {@Load_ExpCal_Settings});

    function Load_ExpCal_Settings(~,~)
        ExpCal_Settings
    end
%% %%%%%%%%%%%%%%%%%     LOAD Button                    %%%%%%%%%%%%%%%%%
LOAD = uicontrol('style', 'pushbutton','string', 'LOAD','units',...
    'normalized','Visible','on','position', [BL LBB+2*DBH BW BH],...
    'FontSize', Font10, 'FontWeight', 'bold','callback', {@Load_Cal});
%% %%%%%%%%%%%%%%%%%     LOAD Function                   %%%%%%%%%%%%%%%%%
    function Load_Cal(~,~)
        [FileName,PathName] = uigetfile('*.mat','Select mat file');
        if FileName==0, return, end

        Structl = load( fullfile(PathName,FileName) );   %# pass file path as string
        Structname = fieldnames(Struct1);                %# typo?
        Loaded_Cal_Tone=Structl.(Structname{1,1});

        set(L00020Hz_Corr,'string',Loaded_Cal_Tone(1,2));
        set(L00025Hz_Corr,'string',Loaded_Cal_Tone(2,2));
        set(L000315Hz_Corr,'string',Loaded_Cal_Tone(3,2));
        set(L00040Hz_Corr,'string',Loaded_Cal_Tone(4,2));
        set(L00050Hz_Corr,'string',Loaded_Cal_Tone(5,2));
        set(L00063Hz_Corr,'string',Loaded_Cal_Tone(6,2));
        set(L00080Hz_Corr,'string',Loaded_Cal_Tone(7,2));
        set(L00100Hz_Corr,'string',Loaded_Cal_Tone(8,2));
        set(L00125Hz_Corr,'string',Loaded_Cal_Tone(9,2));
        set(L00160Hz_Corr,'string',Loaded_Cal_Tone(10,2));
        set(L00200Hz_Corr,'string',Loaded_Cal_Tone(11,2));
        set(L00250Hz_Corr,'string',Loaded_Cal_Tone(12,2));
        set(L00315Hz_Corr,'string',Loaded_Cal_Tone(13,2));
        set(L00400Hz_Corr,'string',Loaded_Cal_Tone(14,2));
        set(L00500Hz_Corr,'string',Loaded_Cal_Tone(15,2));
        set(L00630Hz_Corr,'string',Loaded_Cal_Tone(16,2));
        set(L00800Hz_Corr,'string',Loaded_Cal_Tone(17,2));
        set(L01000Hz_Corr,'string',Loaded_Cal_Tone(18,2));
        set(L01250Hz_Corr,'string',Loaded_Cal_Tone(19,2));
        set(L01600Hz_Corr,'string',Loaded_Cal_Tone(20,2));
        set(L02000Hz_Corr,'string',Loaded_Cal_Tone(21,2));
        set(L02500Hz_Corr,'string',Loaded_Cal_Tone(22,2));
        set(L03150Hz_Corr,'string',Loaded_Cal_Tone(23,2));
        set(L04000Hz_Corr,'string',Loaded_Cal_Tone(24,2));
        set(L05000Hz_Corr,'string',Loaded_Cal_Tone(25,2));
        set(L06300Hz_Corr,'string',Loaded_Cal_Tone(26,2));
        set(L08000Hz_Corr,'string',Loaded_Cal_Tone(27,2));
        set(L10000Hz_Corr,'string',Loaded_Cal_Tone(28,2));
        set(L12500Hz_Corr,'string',Loaded_Cal_Tone(29,2));
        set(L16000Hz_Corr,'string',Loaded_Cal_Tone(30,2));
        set(L20000Hz_Corr,'string',Loaded_Cal_Tone(31,2));

        set(R00020Hz_Corr,'string',Loaded_Cal_Tone(1,3));
        set(R00025Hz_Corr,'string',Loaded_Cal_Tone(2,3));
        set(R000315Hz_Corr,'string',Loaded_Cal_Tone(3,3));
        set(R00040Hz_Corr,'string',Loaded_Cal_Tone(4,3));
        set(R00050Hz_Corr,'string',Loaded_Cal_Tone(5,3));
        set(R00063Hz_Corr,'string',Loaded_Cal_Tone(6,3));
        set(R00080Hz_Corr,'string',Loaded_Cal_Tone(7,3));
        set(R00100Hz_Corr,'string',Loaded_Cal_Tone(8,3));
        set(R00125Hz_Corr,'string',Loaded_Cal_Tone(9,3));
        set(R00160Hz_Corr,'string',Loaded_Cal_Tone(10,3));
        set(R00200Hz_Corr,'string',Loaded_Cal_Tone(11,3));
        set(R00250Hz_Corr,'string',Loaded_Cal_Tone(12,3));
        set(R00315Hz_Corr,'string',Loaded_Cal_Tone(13,3));
        set(R00400Hz_Corr,'string',Loaded_Cal_Tone(14,3));
        set(R00500Hz_Corr,'string',Loaded_Cal_Tone(15,3));
        set(R00630Hz_Corr,'string',Loaded_Cal_Tone(16,3));
        set(R00800Hz_Corr,'string',Loaded_Cal_Tone(17,3));
        set(R01000Hz_Corr,'string',Loaded_Cal_Tone(18,3));
        set(R01250Hz_Corr,'string',Loaded_Cal_Tone(19,3));
        set(R01600Hz_Corr,'string',Loaded_Cal_Tone(20,3));
        set(R02000Hz_Corr,'string',Loaded_Cal_Tone(21,3));
        set(R02500Hz_Corr,'string',Loaded_Cal_Tone(22,3));
        set(R03150Hz_Corr,'string',Loaded_Cal_Tone(23,3));
        set(R04000Hz_Corr,'string',Loaded_Cal_Tone(24,3));
        set(R05000Hz_Corr,'string',Loaded_Cal_Tone(25,3));
        set(R06300Hz_Corr,'string',Loaded_Cal_Tone(26,3));
        set(R08000Hz_Corr,'string',Loaded_Cal_Tone(27,3));
        set(R10000Hz_Corr,'string',Loaded_Cal_Tone(28,3));
        set(R12500Hz_Corr,'string',Loaded_Cal_Tone(29,3));
        set(R16000Hz_Corr,'string',Loaded_Cal_Tone(30,3));
        set(R20000Hz_Corr,'string',Loaded_Cal_Tone(31,3));

        set(B00020Hz_Corr,'string',Loaded_Cal_Tone(1,4));
        set(B00025Hz_Corr,'string',Loaded_Cal_Tone(2,4));
        set(B000315Hz_Corr,'string',Loaded_Cal_Tone(3,4));
        set(B00040Hz_Corr,'string',Loaded_Cal_Tone(4,4));
        set(B00050Hz_Corr,'string',Loaded_Cal_Tone(5,4));
        set(B00063Hz_Corr,'string',Loaded_Cal_Tone(6,4));
        set(B00080Hz_Corr,'string',Loaded_Cal_Tone(7,4));
        set(B00100Hz_Corr,'string',Loaded_Cal_Tone(8,4));
        set(B00125Hz_Corr,'string',Loaded_Cal_Tone(9,4));
        set(B00160Hz_Corr,'string',Loaded_Cal_Tone(10,4));
        set(B00200Hz_Corr,'string',Loaded_Cal_Tone(11,4));
        set(B00250Hz_Corr,'string',Loaded_Cal_Tone(12,4));
        set(B00315Hz_Corr,'string',Loaded_Cal_Tone(13,4));
        set(B00400Hz_Corr,'string',Loaded_Cal_Tone(14,4));
        set(B00500Hz_Corr,'string',Loaded_Cal_Tone(15,4));
        set(B00630Hz_Corr,'string',Loaded_Cal_Tone(16,4));
        set(B00800Hz_Corr,'string',Loaded_Cal_Tone(17,4));
        set(B01000Hz_Corr,'string',Loaded_Cal_Tone(18,4));
        set(B01250Hz_Corr,'string',Loaded_Cal_Tone(19,4));
        set(B01600Hz_Corr,'string',Loaded_Cal_Tone(20,4));
        set(B02000Hz_Corr,'string',Loaded_Cal_Tone(21,4));
        set(B02500Hz_Corr,'string',Loaded_Cal_Tone(22,4));
        set(B03150Hz_Corr,'string',Loaded_Cal_Tone(23,4));
        set(B04000Hz_Corr,'string',Loaded_Cal_Tone(24,4));
        set(B05000Hz_Corr,'string',Loaded_Cal_Tone(25,4));
        set(B06300Hz_Corr,'string',Loaded_Cal_Tone(26,4));
        set(B08000Hz_Corr,'string',Loaded_Cal_Tone(27,4));
        set(B10000Hz_Corr,'string',Loaded_Cal_Tone(28,4));
```

```
        set(B12500Hz_Corr,'string',Loaded_Cal_Tone(29,4));
        set(B16000Hz_Corr,'string',Loaded_Cal_Tone(30,4));
        set(B20000Hz_Corr,'string',Loaded_Cal_Tone(31,4));

    end
%% %%%%%%%%%%%%%%%%%     Overall Correction Function, add to all
    function OVL_CORR_FCN (~,~)
        Corr_Addition_L=str2num(get(OVL_CORR_L,'string'));
        if isempty(Corr_Addition_L); set(OVL_CORR_L,'string','0'); Corr_Addition_L=str2num(get(OVL_CORR_L,'string'));end
        Corr_Addition_R=str2num(get(OVL_CORR_R,'string'));
        if isempty(Corr_Addition_R); set(OVL_CORR_R,'string','0'); Corr_Addition_R=str2num(get(OVL_CORR_R,'string'));end

        TempL01=str2num(get(L00020Hz_Corr,'string')); set(L00020Hz_Corr,'string',TempL01+Corr_Addition_L)
        TempL02=str2num(get(L00025Hz_Corr,'string')); set(L00025Hz_Corr,'string',TempL02+Corr_Addition_L)
        TempL03=str2num(get(L000315Hz_Corr,'string')); set(L000315Hz_Corr,'string',TempL03+Corr_Addition_L)
        TempL04=str2num(get(L00040Hz_Corr,'string')); set(L00040Hz_Corr,'string',TempL04+Corr_Addition_L)
        TempL05=str2num(get(L00050Hz_Corr,'string')); set(L00050Hz_Corr,'string',TempL05+Corr_Addition_L)
        TempL06=str2num(get(L00063Hz_Corr,'string')); set(L00063Hz_Corr,'string',TempL06+Corr_Addition_L)
        TempL07=str2num(get(L00080Hz_Corr,'string')); set(L00080Hz_Corr,'string',TempL07+Corr_Addition_L)
        TempL08=str2num(get(L00100Hz_Corr,'string')); set(L00100Hz_Corr,'string',TempL08+Corr_Addition_L)
        TempL09=str2num(get(L00125Hz_Corr,'string')); set(L00125Hz_Corr,'string',TempL09+Corr_Addition_L)
        TempL10=str2num(get(L00160Hz_Corr,'string')); set(L00160Hz_Corr,'string',TempL10+Corr_Addition_L)
        TempL11=str2num(get(L00200Hz_Corr,'string')); set(L00200Hz_Corr,'string',TempL11+Corr_Addition_L)
        TempL12=str2num(get(L00250Hz_Corr,'string')); set(L00250Hz_Corr,'string',TempL12+Corr_Addition_L)
        TempL13=str2num(get(L00315Hz_Corr,'string')); set(L00315Hz_Corr,'string',TempL13+Corr_Addition_L)
        TempL14=str2num(get(L00400Hz_Corr,'string')); set(L00400Hz_Corr,'string',TempL14+Corr_Addition_L)
        TempL15=str2num(get(L00500Hz_Corr,'string')); set(L00500Hz_Corr,'string',TempL15+Corr_Addition_L)
        TempL16=str2num(get(L00630Hz_Corr,'string')); set(L00630Hz_Corr,'string',TempL16+Corr_Addition_L)
        TempL17=str2num(get(L00800Hz_Corr,'string')); set(L00800Hz_Corr,'string',TempL17+Corr_Addition_L)
        TempL18=str2num(get(L01000Hz_Corr,'string')); set(L01000Hz_Corr,'string',TempL18+Corr_Addition_L)
        TempL19=str2num(get(L01250Hz_Corr,'string')); set(L01250Hz_Corr,'string',TempL19+Corr_Addition_L)
        TempL20=str2num(get(L01600Hz_Corr,'string')); set(L01600Hz_Corr,'string',TempL20+Corr_Addition_L)
        TempL21=str2num(get(L02000Hz_Corr,'string')); set(L02000Hz_Corr,'string',TempL21+Corr_Addition_L)
        TempL22=str2num(get(L02500Hz_Corr,'string')); set(L02500Hz_Corr,'string',TempL22+Corr_Addition_L)
        TempL23=str2num(get(L03150Hz_Corr,'string')); set(L03150Hz_Corr,'string',TempL23+Corr_Addition_L)
        TempL24=str2num(get(L04000Hz_Corr,'string')); set(L04000Hz_Corr,'string',TempL24+Corr_Addition_L)
        TempL25=str2num(get(L05000Hz_Corr,'string')); set(L05000Hz_Corr,'string',TempL25+Corr_Addition_L)
        TempL26=str2num(get(L06300Hz_Corr,'string')); set(L06300Hz_Corr,'string',TempL26+Corr_Addition_L)
        TempL27=str2num(get(L08000Hz_Corr,'string')); set(L08000Hz_Corr,'string',TempL27+Corr_Addition_L)
        TempL28=str2num(get(L10000Hz_Corr,'string')); set(L10000Hz_Corr,'string',TempL28+Corr_Addition_L)
        TempL29=str2num(get(L12500Hz_Corr,'string')); set(L12500Hz_Corr,'string',TempL29+Corr_Addition_L)
        TempL30=str2num(get(L16000Hz_Corr,'string')); set(L16000Hz_Corr,'string',TempL30+Corr_Addition_L)
        TempL31=str2num(get(L20000Hz_Corr,'string')); set(L20000Hz_Corr,'string',TempL31+Corr_Addition_L)

        TempR01=str2num(get(R00020Hz_Corr,'string')); set(R00020Hz_Corr,'string',TempR01+Corr_Addition_R)
        TempR02=str2num(get(R00025Hz_Corr,'string')); set(R00025Hz_Corr,'string',TempR02+Corr_Addition_R)
        TempR03=str2num(get(R000315Hz_Corr,'string')); set(R000315Hz_Corr,'string',TempR03+Corr_Addition_R)
        TempR04=str2num(get(R00040Hz_Corr,'string')); set(R00040Hz_Corr,'string',TempR04+Corr_Addition_R)
        TempR05=str2num(get(R00050Hz_Corr,'string')); set(R00050Hz_Corr,'string',TempR05+Corr_Addition_R)
        TempR06=str2num(get(R00063Hz_Corr,'string')); set(R00063Hz_Corr,'string',TempR06+Corr_Addition_R)
        TempR07=str2num(get(R00080Hz_Corr,'string')); set(R00080Hz_Corr,'string',TempR07+Corr_Addition_R)
        TempR08=str2num(get(R00100Hz_Corr,'string')); set(R00100Hz_Corr,'string',TempR08+Corr_Addition_R)
        TempR09=str2num(get(R00125Hz_Corr,'string')); set(R00125Hz_Corr,'string',TempR09+Corr_Addition_R)
        TempR10=str2num(get(R00160Hz_Corr,'string')); set(R00160Hz_Corr,'string',TempR10+Corr_Addition_R)
        TempR11=str2num(get(R00200Hz_Corr,'string')); set(R00200Hz_Corr,'string',TempR11+Corr_Addition_R)
        TempR12=str2num(get(R00250Hz_Corr,'string')); set(R00250Hz_Corr,'string',TempR12+Corr_Addition_R)
        TempR13=str2num(get(R00315Hz_Corr,'string')); set(R00315Hz_Corr,'string',TempR13+Corr_Addition_R)
        TempR14=str2num(get(R00400Hz_Corr,'string')); set(R00400Hz_Corr,'string',TempR14+Corr_Addition_R)
        TempR15=str2num(get(R00500Hz_Corr,'string')); set(R00500Hz_Corr,'string',TempR15+Corr_Addition_R)
        TempR16=str2num(get(R00630Hz_Corr,'string')); set(R00630Hz_Corr,'string',TempR16+Corr_Addition_R)
        TempR17=str2num(get(R00800Hz_Corr,'string')); set(R00800Hz_Corr,'string',TempR17+Corr_Addition_R)
        TempR18=str2num(get(R01000Hz_Corr,'string')); set(R01000Hz_Corr,'string',TempR18+Corr_Addition_R)
        TempR19=str2num(get(R01250Hz_Corr,'string')); set(R01250Hz_Corr,'string',TempR19+Corr_Addition_R)
        TempR20=str2num(get(R01600Hz_Corr,'string')); set(R01600Hz_Corr,'string',TempR20+Corr_Addition_R)
        TempR21=str2num(get(R02000Hz_Corr,'string')); set(R02000Hz_Corr,'string',TempR21+Corr_Addition_R)
        TempR22=str2num(get(R02500Hz_Corr,'string')); set(R02500Hz_Corr,'string',TempR22+Corr_Addition_R)
        TempR23=str2num(get(R03150Hz_Corr,'string')); set(R03150Hz_Corr,'string',TempR23+Corr_Addition_R)
        TempR24=str2num(get(R04000Hz_Corr,'string')); set(R04000Hz_Corr,'string',TempR24+Corr_Addition_R)
        TempR25=str2num(get(R05000Hz_Corr,'string')); set(R05000Hz_Corr,'string',TempR25+Corr_Addition_R)
        TempR26=str2num(get(R06300Hz_Corr,'string')); set(R06300Hz_Corr,'string',TempR26+Corr_Addition_R)
        TempR27=str2num(get(R08000Hz_Corr,'string')); set(R08000Hz_Corr,'string',TempR27+Corr_Addition_R)
        TempR28=str2num(get(R10000Hz_Corr,'string')); set(R10000Hz_Corr,'string',TempR28+Corr_Addition_R)
        TempR29=str2num(get(R12500Hz_Corr,'string')); set(R12500Hz_Corr,'string',TempR29+Corr_Addition_R)
        TempR30=str2num(get(R16000Hz_Corr,'string')); set(R16000Hz_Corr,'string',TempR30+Corr_Addition_R)
        TempR31=str2num(get(R20000Hz_Corr,'string')); set(R20000Hz_Corr,'string',TempR31+Corr_Addition_R)

    end
%% %%%%%%%%%%%%%%%%%     Plotting Button

PLOT = uicontrol('style', 'pushbutton','string', 'PLOT','units',...
    'normalized','Visible','on','position', [BL LBB+1*DBH BW BH],...
    'FontSize', Font10, 'FontWeight', 'bold','callback', {@Plot_Cal});
%% %%%%%%%%%%%%%%%%%     Plotting Function
    function Plot_Cal(~,~)
if isempty(str2num(get(L00020Hz_Corr,'string'))); set(L00020Hz_Corr,'string',0.0);str2num(get(L00020Hz_Corr,'string'));end
            if isempty(str2num(get(L00025Hz_Corr,'string'))); set(L00025Hz_Corr,'string',0.0);str2num(get(L00025Hz_Corr,'string'));end
            if isempty(str2num(get(L000315Hz_Corr,'string'))); set(L000315Hz_Corr,'string',0.0);str2num(get(L000315Hz_Corr,'string'));end
            if isempty(str2num(get(L00040Hz_Corr,'string'))); set(L00040Hz_Corr,'string',0.0);str2num(get(L00040Hz_Corr,'string'));end
            if isempty(str2num(get(L00050Hz_Corr,'string'))); set(L00050Hz_Corr,'string',0.0);str2num(get(L00050Hz_Corr,'string'));end
            if isempty(str2num(get(L00063Hz_Corr,'string'))); set(L00063Hz_Corr,'string',0.0);str2num(get(L00063Hz_Corr,'string'));end
            if isempty(str2num(get(L00080Hz_Corr,'string'))); set(L00080Hz_Corr,'string',0.0);str2num(get(L00080Hz_Corr,'string'));end
            if isempty(str2num(get(L00100Hz_Corr,'string'))); set(L00100Hz_Corr,'string',0.0);str2num(get(L00100Hz_Corr,'string'));end
            if isempty(str2num(get(L00125Hz_Corr,'string'))); set(L00125Hz_Corr,'string',0.0);str2num(get(L00125Hz_Corr,'string'));end
            if isempty(str2num(get(L00160Hz_Corr,'string'))); set(L00160Hz_Corr,'string',0.0);str2num(get(L00160Hz_Corr,'string'));end
            if isempty(str2num(get(L00200Hz_Corr,'string'))); set(L00200Hz_Corr,'string',0.0);str2num(get(L00200Hz_Corr,'string'));end
            if isempty(str2num(get(L00250Hz_Corr,'string'))); set(L00250Hz_Corr,'string',0.0);str2num(get(L00250Hz_Corr,'string'));end
            if isempty(str2num(get(L00315Hz_Corr,'string'))); set(L00315Hz_Corr,'string',0.0);str2num(get(L00315Hz_Corr,'string'));end
            if isempty(str2num(get(L00400Hz_Corr,'string'))); set(L00400Hz_Corr,'string',0.0);str2num(get(L00400Hz_Corr,'string'));end
            if isempty(str2num(get(L00500Hz_Corr,'string'))); set(L00500Hz_Corr,'string',0.0);str2num(get(L00500Hz_Corr,'string'));end
            if isempty(str2num(get(L00630Hz_Corr,'string'))); set(L00630Hz_Corr,'string',0.0);str2num(get(L00630Hz_Corr,'string'));end
            if isempty(str2num(get(L00800Hz_Corr,'string'))); set(L00800Hz_Corr,'string',0.0);str2num(get(L00800Hz_Corr,'string'));end
            if isempty(str2num(get(L01000Hz_Corr,'string'))); set(L01000Hz_Corr,'string',0.0);str2num(get(L01000Hz_Corr,'string'));end
            if isempty(str2num(get(L01250Hz_Corr,'string'))); set(L01250Hz_Corr,'string',0.0);str2num(get(L01250Hz_Corr,'string'));end
            if isempty(str2num(get(L01600Hz_Corr,'string'))); set(L01600Hz_Corr,'string',0.0);str2num(get(L01600Hz_Corr,'string'));end
            if isempty(str2num(get(L02000Hz_Corr,'string'))); set(L02000Hz_Corr,'string',0.0);str2num(get(L02000Hz_Corr,'string'));end
            if isempty(str2num(get(L02500Hz_Corr,'string'))); set(L02500Hz_Corr,'string',0.0);str2num(get(L02500Hz_Corr,'string'));end
            if isempty(str2num(get(L03150Hz_Corr,'string'))); set(L03150Hz_Corr,'string',0.0);str2num(get(L03150Hz_Corr,'string'));end
            if isempty(str2num(get(L04000Hz_Corr,'string'))); set(L04000Hz_Corr,'string',0.0);str2num(get(L04000Hz_Corr,'string'));end
            if isempty(str2num(get(L05000Hz_Corr,'string'))); set(L05000Hz_Corr,'string',0.0);str2num(get(L05000Hz_Corr,'string'));end
            if isempty(str2num(get(L06300Hz_Corr,'string'))); set(L06300Hz_Corr,'string',0.0);str2num(get(L06300Hz_Corr,'string'));end
            if isempty(str2num(get(L08000Hz_Corr,'string'))); set(L08000Hz_Corr,'string',0.0);str2num(get(L08000Hz_Corr,'string'));end
            if isempty(str2num(get(L10000Hz_Corr,'string'))); set(L10000Hz_Corr,'string',0.0);str2num(get(L10000Hz_Corr,'string'));end
            if isempty(str2num(get(L12500Hz_Corr,'string'))); set(L12500Hz_Corr,'string',0.0);str2num(get(L12500Hz_Corr,'string'));end
            if isempty(str2num(get(L16000Hz_Corr,'string'))); set(L16000Hz_Corr,'string',0.0);str2num(get(L16000Hz_Corr,'string'));end
```

```
    if isempty(str2num(get(L20000Hz_Corr,'string'))); set(L20000Hz_Corr,'string',0.0);str2num(get(L20000Hz_Corr,'string'));end


    if isempty(str2num(get(R00020Hz_Corr,'string'))); set(R00020Hz_Corr,'string',0.0);str2num(get(R00020Hz_Corr,'string'));end
    if isempty(str2num(get(R00025Hz_Corr,'string'))); set(R00025Hz_Corr,'string',0.0);str2num(get(R00025Hz_Corr,'string'));end
    if isempty(str2num(get(R000315Hz_Corr,'string'))); set(R000315Hz_Corr,'string',0.0);str2num(get(R000315Hz_Corr,'string'));end
    if isempty(str2num(get(R00040Hz_Corr,'string'))); set(R00040Hz_Corr,'string',0.0);str2num(get(R00040Hz_Corr,'string'));end
    if isempty(str2num(get(R00050Hz_Corr,'string'))); set(R00050Hz_Corr,'string',0.0);str2num(get(R00050Hz_Corr,'string'));end
    if isempty(str2num(get(R00063Hz_Corr,'string'))); set(R00063Hz_Corr,'string',0.0);str2num(get(R00063Hz_Corr,'string'));end
    if isempty(str2num(get(R00080Hz_Corr,'string'))); set(R00080Hz_Corr,'string',0.0);str2num(get(R00080Hz_Corr,'string'));end
    if isempty(str2num(get(R00100Hz_Corr,'string'))); set(R00100Hz_Corr,'string',0.0);str2num(get(R00100Hz_Corr,'string'));end
    if isempty(str2num(get(R00125Hz_Corr,'string'))); set(R00125Hz_Corr,'string',0.0);str2num(get(R00125Hz_Corr,'string'));end
    if isempty(str2num(get(R00160Hz_Corr,'string'))); set(R00160Hz_Corr,'string',0.0);str2num(get(R00160Hz_Corr,'string'));end
    if isempty(str2num(get(R00200Hz_Corr,'string'))); set(R00200Hz_Corr,'string',0.0);str2num(get(R00200Hz_Corr,'string'));end
    if isempty(str2num(get(R00250Hz_Corr,'string'))); set(R00250Hz_Corr,'string',0.0);str2num(get(R00250Hz_Corr,'string'));end
    if isempty(str2num(get(R00315Hz_Corr,'string'))); set(R00315Hz_Corr,'string',0.0);str2num(get(R00315Hz_Corr,'string'));end
    if isempty(str2num(get(R00400Hz_Corr,'string'))); set(R00400Hz_Corr,'string',0.0);str2num(get(R00400Hz_Corr,'string'));end
    if isempty(str2num(get(R00500Hz_Corr,'string'))); set(R00500Hz_Corr,'string',0.0);str2num(get(R00500Hz_Corr,'string'));end
    if isempty(str2num(get(R00630Hz_Corr,'string'))); set(R00630Hz_Corr,'string',0.0);str2num(get(R00630Hz_Corr,'string'));end
    if isempty(str2num(get(R00800Hz_Corr,'string'))); set(R00800Hz_Corr,'string',0.0);str2num(get(R00800Hz_Corr,'string'));end
    if isempty(str2num(get(R01000Hz_Corr,'string'))); set(R01000Hz_Corr,'string',0.0);str2num(get(R01000Hz_Corr,'string'));end
    if isempty(str2num(get(R01250Hz_Corr,'string'))); set(R01250Hz_Corr,'string',0.0);str2num(get(R01250Hz_Corr,'string'));end
    if isempty(str2num(get(R01600Hz_Corr,'string'))); set(R01600Hz_Corr,'string',0.0);str2num(get(R01600Hz_Corr,'string'));end
    if isempty(str2num(get(R02000Hz_Corr,'string'))); set(R02000Hz_Corr,'string',0.0);str2num(get(R02000Hz_Corr,'string'));end
    if isempty(str2num(get(R02500Hz_Corr,'string'))); set(R02500Hz_Corr,'string',0.0);str2num(get(R02500Hz_Corr,'string'));end
    if isempty(str2num(get(R03150Hz_Corr,'string'))); set(R03150Hz_Corr,'string',0.0);str2num(get(R03150Hz_Corr,'string'));end
    if isempty(str2num(get(R04000Hz_Corr,'string'))); set(R04000Hz_Corr,'string',0.0);str2num(get(R04000Hz_Corr,'string'));end
    if isempty(str2num(get(R05000Hz_Corr,'string'))); set(R05000Hz_Corr,'string',0.0);str2num(get(R05000Hz_Corr,'string'));end
    if isempty(str2num(get(R06300Hz_Corr,'string'))); set(R06300Hz_Corr,'string',0.0);str2num(get(R06300Hz_Corr,'string'));end
    if isempty(str2num(get(R08000Hz_Corr,'string'))); set(R08000Hz_Corr,'string',0.0);str2num(get(R08000Hz_Corr,'string'));end
    if isempty(str2num(get(R10000Hz_Corr,'string'))); set(R10000Hz_Corr,'string',0.0);str2num(get(R10000Hz_Corr,'string'));end
    if isempty(str2num(get(R12500Hz_Corr,'string'))); set(R12500Hz_Corr,'string',0.0);str2num(get(R12500Hz_Corr,'string'));end
    if isempty(str2num(get(R16000Hz_Corr,'string'))); set(R16000Hz_Corr,'string',0.0);str2num(get(R16000Hz_Corr,'string'));end
    if isempty(str2num(get(R20000Hz_Corr,'string'))); set(R20000Hz_Corr,'string',0.0);str2num(get(R20000Hz_Corr,'string'));end


    if isempty(str2num(get(B00020Hz_Corr,'string'))); set(B00020Hz_Corr,'string',0.0);str2num(get(B00020Hz_Corr,'string'));end
    if isempty(str2num(get(B00025Hz_Corr,'string'))); set(B00025Hz_Corr,'string',0.0);str2num(get(B00025Hz_Corr,'string'));end
    if isempty(str2num(get(B000315Hz_Corr,'string'))); set(B000315Hz_Corr,'string',0.0);str2num(get(B000315Hz_Corr,'string'));end
    if isempty(str2num(get(B00040Hz_Corr,'string'))); set(B00040Hz_Corr,'string',0.0);str2num(get(B00040Hz_Corr,'string'));end
    if isempty(str2num(get(B00050Hz_Corr,'string'))); set(B00050Hz_Corr,'string',0.0);str2num(get(B00050Hz_Corr,'string'));end
    if isempty(str2num(get(B00063Hz_Corr,'string'))); set(B00063Hz_Corr,'string',0.0);str2num(get(B00063Hz_Corr,'string'));end
    if isempty(str2num(get(B00080Hz_Corr,'string'))); set(B00080Hz_Corr,'string',0.0);str2num(get(B00080Hz_Corr,'string'));end
    if isempty(str2num(get(B00100Hz_Corr,'string'))); set(B00100Hz_Corr,'string',0.0);str2num(get(B00100Hz_Corr,'string'));end
    if isempty(str2num(get(B00125Hz_Corr,'string'))); set(B00125Hz_Corr,'string',0.0);str2num(get(B00125Hz_Corr,'string'));end
    if isempty(str2num(get(B00160Hz_Corr,'string'))); set(B00160Hz_Corr,'string',0.0);str2num(get(B00160Hz_Corr,'string'));end
    if isempty(str2num(get(B00200Hz_Corr,'string'))); set(B00200Hz_Corr,'string',0.0);str2num(get(B00200Hz_Corr,'string'));end
    if isempty(str2num(get(B00250Hz_Corr,'string'))); set(B00250Hz_Corr,'string',0.0);str2num(get(B00250Hz_Corr,'string'));end
    if isempty(str2num(get(B00315Hz_Corr,'string'))); set(B00315Hz_Corr,'string',0.0);str2num(get(B00315Hz_Corr,'string'));end
    if isempty(str2num(get(B00400Hz_Corr,'string'))); set(B00400Hz_Corr,'string',0.0);str2num(get(B00400Hz_Corr,'string'));end
    if isempty(str2num(get(B00500Hz_Corr,'string'))); set(B00500Hz_Corr,'string',0.0);str2num(get(B00500Hz_Corr,'string'));end
    if isempty(str2num(get(B00630Hz_Corr,'string'))); set(B00630Hz_Corr,'string',0.0);str2num(get(B00630Hz_Corr,'string'));end
    if isempty(str2num(get(B00800Hz_Corr,'string'))); set(B00800Hz_Corr,'string',0.0);str2num(get(B00800Hz_Corr,'string'));end
    if isempty(str2num(get(B01000Hz_Corr,'string'))); set(B01000Hz_Corr,'string',0.0);str2num(get(B01000Hz_Corr,'string'));end
    if isempty(str2num(get(B01250Hz_Corr,'string'))); set(B01250Hz_Corr,'string',0.0);str2num(get(B01250Hz_Corr,'string'));end
    if isempty(str2num(get(B01600Hz_Corr,'string'))); set(B01600Hz_Corr,'string',0.0);str2num(get(B01600Hz_Corr,'string'));end
    if isempty(str2num(get(B02000Hz_Corr,'string'))); set(B02000Hz_Corr,'string',0.0);str2num(get(B02000Hz_Corr,'string'));end
    if isempty(str2num(get(B02500Hz_Corr,'string'))); set(B02500Hz_Corr,'string',0.0);str2num(get(B02500Hz_Corr,'string'));end
    if isempty(str2num(get(B03150Hz_Corr,'string'))); set(B03150Hz_Corr,'string',0.0);str2num(get(B03150Hz_Corr,'string'));end
    if isempty(str2num(get(B04000Hz_Corr,'string'))); set(B04000Hz_Corr,'string',0.0);str2num(get(B04000Hz_Corr,'string'));end
    if isempty(str2num(get(B05000Hz_Corr,'string'))); set(B05000Hz_Corr,'string',0.0);str2num(get(B05000Hz_Corr,'string'));end
    if isempty(str2num(get(B06300Hz_Corr,'string'))); set(B06300Hz_Corr,'string',0.0);str2num(get(B06300Hz_Corr,'string'));end
    if isempty(str2num(get(B08000Hz_Corr,'string'))); set(B08000Hz_Corr,'string',0.0);str2num(get(B08000Hz_Corr,'string'));end
    if isempty(str2num(get(B10000Hz_Corr,'string'))); set(B10000Hz_Corr,'string',0.0);str2num(get(B10000Hz_Corr,'string'));end
    if isempty(str2num(get(B12500Hz_Corr,'string'))); set(B12500Hz_Corr,'string',0.0);str2num(get(B12500Hz_Corr,'string'));end
    if isempty(str2num(get(B16000Hz_Corr,'string'))); set(B16000Hz_Corr,'string',0.0);str2num(get(B16000Hz_Corr,'string'));end
    if isempty(str2num(get(B20000Hz_Corr,'string'))); set(B20000Hz_Corr,'string',0.0);str2num(get(B20000Hz_Corr,'string'));end


Cal_Plot(:,1)=Freq;
Cal_Plot(:,2)=[str2num(get(L00020Hz_Corr,'string'))
    str2num(get(L00025Hz_Corr,'string'))
    str2num(get(L000315Hz_Corr,'string'))
    str2num(get(L00040Hz_Corr,'string'))
    str2num(get(L00050Hz_Corr,'string'))
    str2num(get(L00063Hz_Corr,'string'))
    str2num(get(L00080Hz_Corr,'string'))
    str2num(get(L00100Hz_Corr,'string'))
    str2num(get(L00125Hz_Corr,'string'))
    str2num(get(L00160Hz_Corr,'string'))
    str2num(get(L00200Hz_Corr,'string'))
    str2num(get(L00250Hz_Corr,'string'))
    str2num(get(L00315Hz_Corr,'string'))
    str2num(get(L00400Hz_Corr,'string'))
    str2num(get(L00500Hz_Corr,'string'))
    str2num(get(L00630Hz_Corr,'string'))
    str2num(get(L00800Hz_Corr,'string'))
    str2num(get(L01000Hz_Corr,'string'))
    str2num(get(L01250Hz_Corr,'string'))
    str2num(get(L01600Hz_Corr,'string'))
    str2num(get(L02000Hz_Corr,'string'))
    str2num(get(L02500Hz_Corr,'string'))
    str2num(get(L03150Hz_Corr,'string'))
    str2num(get(L04000Hz_Corr,'string'))
    str2num(get(L05000Hz_Corr,'string'))
    str2num(get(L06300Hz_Corr,'string'))
    str2num(get(L08000Hz_Corr,'string'))
    str2num(get(L10000Hz_Corr,'string'))
    str2num(get(L12500Hz_Corr,'string'))
    str2num(get(L16000Hz_Corr,'string'))
    str2num(get(L20000Hz_Corr,'string'))];

Cal_Plot(:,3)=[str2num(get(R00020Hz_Corr,'string'))
    str2num(get(R00025Hz_Corr,'string'))
    str2num(get(R000315Hz_Corr,'string'))
    str2num(get(R00040Hz_Corr,'string'))
    str2num(get(R00050Hz_Corr,'string'))
    str2num(get(R00063Hz_Corr,'string'))
    str2num(get(R00080Hz_Corr,'string'))
    str2num(get(R00100Hz_Corr,'string'))
    str2num(get(R00125Hz_Corr,'string'))
    str2num(get(R00160Hz_Corr,'string'))
    str2num(get(R00200Hz_Corr,'string'))
    str2num(get(R00250Hz_Corr,'string'))
```

```
str2num(get(R00315Hz_Corr,'string'))
str2num(get(R00400Hz_Corr,'string'))
str2num(get(R00500Hz_Corr,'string'))
str2num(get(R00630Hz_Corr,'string'))
str2num(get(R00800Hz_Corr,'string'))
str2num(get(R01000Hz_Corr,'string'))
str2num(get(R01250Hz_Corr,'string'))
str2num(get(R01600Hz_Corr,'string'))
str2num(get(R02000Hz_Corr,'string'))
str2num(get(R02500Hz_Corr,'string'))
str2num(get(R03150Hz_Corr,'string'))
str2num(get(R04000Hz_Corr,'string'))
str2num(get(R05000Hz_Corr,'string'))
str2num(get(R06300Hz_Corr,'string'))
str2num(get(R08000Hz_Corr,'string'))
str2num(get(R10000Hz_Corr,'string'))
str2num(get(R12500Hz_Corr,'string'))
str2num(get(R16000Hz_Corr,'string'))
str2num(get(R20000Hz_Corr,'string'))];

    Cal_Plot(:,4)=[str2num(get(B00020Hz_Corr,'string'))
str2num(get(B00025Hz_Corr,'string'))
str2num(get(B000315Hz_Corr,'string'))
str2num(get(B00040Hz_Corr,'string'))
str2num(get(B00050Hz_Corr,'string'))
str2num(get(B00063Hz_Corr,'string'))
str2num(get(B00080Hz_Corr,'string'))
str2num(get(B00100Hz_Corr,'string'))
str2num(get(B00125Hz_Corr,'string'))
str2num(get(B00160Hz_Corr,'string'))
str2num(get(B00200Hz_Corr,'string'))
str2num(get(B00250Hz_Corr,'string'))
str2num(get(B00315Hz_Corr,'string'))
str2num(get(B00400Hz_Corr,'string'))
str2num(get(B00500Hz_Corr,'string'))
str2num(get(B00630Hz_Corr,'string'))
str2num(get(B00800Hz_Corr,'string'))
str2num(get(B01000Hz_Corr,'string'))
str2num(get(B01250Hz_Corr,'string'))
str2num(get(B01600Hz_Corr,'string'))
str2num(get(B02000Hz_Corr,'string'))
str2num(get(B02500Hz_Corr,'string'))
str2num(get(B03150Hz_Corr,'string'))
str2num(get(B04000Hz_Corr,'string'))
str2num(get(B05000Hz_Corr,'string'))
str2num(get(B06300Hz_Corr,'string'))
str2num(get(B08000Hz_Corr,'string'))
str2num(get(B10000Hz_Corr,'string'))
str2num(get(B12500Hz_Corr,'string'))
str2num(get(B16000Hz_Corr,'string'))
str2num(get(B20000Hz_Corr,'string'))];

            Plot_Calibration=figure;
            set(Plot_Calibration,'name','Calibration Plot','numbertitle','off')
            PlotAxes = semilogx(Cal_Plot(:,1), Cal_Plot(:,2), 'ob',...
                                Cal_Plot(:,1), Cal_Plot(:,3),'or',...
                                Cal_Plot(:,1), Cal_Plot(:,2), '-b',...
                                Cal_Plot(:,1), Cal_Plot(:,3),'-r',...
                                Cal_Plot(:,1), Cal_Plot(:,4),'-dm'); %#ok<NASGU> %, 'Parent', handles.graph

            ax_PlotAxes = gca; %Collect info on current axes values to gca
            curtick = get(gca, 'XTick'); % Collecting current x-labels
            set(ax_PlotAxes, 'units', 'normalized',...
                'OuterPosition', [0 0.2 1 0.6],...
                'Position',[0.1 0.175 0.8 0.7],...
                'XLim', [10 20000],...
                'YLim', [-10 30],...
                'XTickLabel', cellstr(num2str(curtick(:)))); % Setting the x-labels to non-scientific
            grid

            set(Plot_Calibration,'Toolbar','figure');
            set(Plot_Calibration,'MenuBar','none');

            hToolbar = findall(Plot_Calibration,'tag','FigureToolBar');
            delete(findall(hToolbar,'tag','Standard.NewFigure'))
            delete(findall(hToolbar,'tag','Standard.FileOpen'))
            delete(findall(hToolbar,'tag','Plottools.PlottoolsOn'))
            delete(findall(hToolbar,'tag','Plottools.PlottoolsOff'))
            delete(findall(hToolbar,'tag','Annotation.InsertColorbar'))
            delete(findall(hToolbar,'tag','DataManager.Linking'))
            delete(findall(hToolbar,'tag','Standard.EditPlot'))
            delete(findall(hToolbar,'tag','Exploration.Rotate'))
            delete(findall(hToolbar,'tag','Exploration.Brushing'))
            delete(findall(hToolbar,'tag','Annotation.InsertLegend'))

            %Center Legend Box
            L_Width=0.22;
            L_Edge=0.5*(1-L_Width);
            L_Edge1=L_Edge+0.015;
            LegendBox = uicontrol('style','frame', 'units', 'normalized',...
                'position', [L_Edge 0.74 L_Width 0.125],...
                'BackgroundColor',[1 1 1]);
            PlotLabel1 = uicontrol('style','text', 'units', 'normalized',...
                'position', [L_Edge1+0.05 0.8 0.15 0.06], 'string', 'Left Ear',...
                'visible','on', 'FontSize', Font12,'HorizontalAlignment','Left',...
                'BackgroundColor',[1 1 1]);
            PlotLabel1mark = uicontrol('style','text', 'units', 'normalized',...
                'position', [L_Edge1 0.8 0.05 0.06], 'string', '-o-',...
                'visible','on', 'FontSize', Font12,'HorizontalAlignment','Left',...
                'BackgroundColor',[1 1 1],'ForegroundColor',[0 0 1]);
            PlotLabel2 = uicontrol('style','text', 'units', 'normalized',...
                'position', [L_Edge1+0.05 0.75 0.15 0.06], 'string', 'Right Ear',...
                'visible','on', 'FontSize', Font12,'HorizontalAlignment','Left',...
                'BackgroundColor',[1 1 1]);
            PlotLabel2mark = uicontrol('style','text', 'units', 'normalized',...
                'position', [L_Edge1 0.75 0.05 0.06], 'string', '-o-',...
                'visible','on', 'FontSize', Font12,'HorizontalAlignment','Left',...
                'BackgroundColor',[1 1 1],'ForegroundColor',[1 0 0]);


            PlotTitle = uicontrol('style','text', 'units', 'normalized',...
                'position', [0.1 0.90 0.8 0.06], 'string', 'Calibration Plot of Correction Values (dB)',...
                'visible','on', 'FontSize', Font16,'HorizontalAlignment','Center',...
                'BackgroundColor',Background_Color, 'FontWeight', 'bold');
```

```
                    PlotX_Axis = uicontrol('style','text', 'units', 'normalized',...
                        'position', [0.1 0.05 0.8 0.06], 'string', 'Frequency (Hz)',...
                        'visible','on', 'FontSize', Font12,'HorizontalAlignment','Center',...
                        'BackgroundColor',Background_Color, 'FontWeight', 'bold');

    end
%% %%%%%%%%%%%%%%%%%     SAVE Button                        %%%%%%%%%%%%%%%%%%
SAVE = uicontrol('style', 'pushbutton','string', 'SAVE','units',...
    'normalized','Visible','on','position', [BL LBB+0*DBH BW BH],...
    'FontSize', Font10, 'FontWeight', 'bold','callback', {@Save_Cal});
%% %%%%%%%%%%%%%%%%%     SAVE                               %%%%%%%%%%%%%%%%%%
    function Save_Cal(~,~)


            if isempty(str2num(get(L00020Hz_Corr,'string'))); set(L00020Hz_Corr,'string',0.0);str2num(get(L00020Hz_Corr,'string'));end
            if isempty(str2num(get(L00025Hz_Corr,'string'))); set(L00025Hz_Corr,'string',0.0);str2num(get(L00025Hz_Corr,'string'));end
            if isempty(str2num(get(L000315Hz_Corr,'string'))); set(L000315Hz_Corr,'string',0.0);str2num(get(L000315Hz_Corr,'string'));end
            if isempty(str2num(get(L00040Hz_Corr,'string'))); set(L00040Hz_Corr,'string',0.0);str2num(get(L00040Hz_Corr,'string'));end
            if isempty(str2num(get(L00050Hz_Corr,'string'))); set(L00050Hz_Corr,'string',0.0);str2num(get(L00050Hz_Corr,'string'));end
            if isempty(str2num(get(L00063Hz_Corr,'string'))); set(L00063Hz_Corr,'string',0.0);str2num(get(L00063Hz_Corr,'string'));end
            if isempty(str2num(get(L00080Hz_Corr,'string'))); set(L00080Hz_Corr,'string',0.0);str2num(get(L00080Hz_Corr,'string'));end
            if isempty(str2num(get(L00100Hz_Corr,'string'))); set(L00100Hz_Corr,'string',0.0);str2num(get(L00100Hz_Corr,'string'));end
            if isempty(str2num(get(L00125Hz_Corr,'string'))); set(L00125Hz_Corr,'string',0.0);str2num(get(L00125Hz_Corr,'string'));end
            if isempty(str2num(get(L00160Hz_Corr,'string'))); set(L00160Hz_Corr,'string',0.0);str2num(get(L00160Hz_Corr,'string'));end
            if isempty(str2num(get(L00200Hz_Corr,'string'))); set(L00200Hz_Corr,'string',0.0);str2num(get(L00200Hz_Corr,'string'));end
            if isempty(str2num(get(L00250Hz_Corr,'string'))); set(L00250Hz_Corr,'string',0.0);str2num(get(L00250Hz_Corr,'string'));end
            if isempty(str2num(get(L00315Hz_Corr,'string'))); set(L00315Hz_Corr,'string',0.0);str2num(get(L00315Hz_Corr,'string'));end
            if isempty(str2num(get(L00400Hz_Corr,'string'))); set(L00400Hz_Corr,'string',0.0);str2num(get(L00400Hz_Corr,'string'));end
            if isempty(str2num(get(L00500Hz_Corr,'string'))); set(L00500Hz_Corr,'string',0.0);str2num(get(L00500Hz_Corr,'string'));end
            if isempty(str2num(get(L00630Hz_Corr,'string'))); set(L00630Hz_Corr,'string',0.0);str2num(get(L00630Hz_Corr,'string'));end
            if isempty(str2num(get(L00800Hz_Corr,'string'))); set(L00800Hz_Corr,'string',0.0);str2num(get(L00800Hz_Corr,'string'));end
            if isempty(str2num(get(L01000Hz_Corr,'string'))); set(L01000Hz_Corr,'string',0.0);str2num(get(L01000Hz_Corr,'string'));end
            if isempty(str2num(get(L01250Hz_Corr,'string'))); set(L01250Hz_Corr,'string',0.0);str2num(get(L01250Hz_Corr,'string'));end
            if isempty(str2num(get(L01600Hz_Corr,'string'))); set(L01600Hz_Corr,'string',0.0);str2num(get(L01600Hz_Corr,'string'));end
            if isempty(str2num(get(L02000Hz_Corr,'string'))); set(L02000Hz_Corr,'string',0.0);str2num(get(L02000Hz_Corr,'string'));end
            if isempty(str2num(get(L02500Hz_Corr,'string'))); set(L02500Hz_Corr,'string',0.0);str2num(get(L02500Hz_Corr,'string'));end
            if isempty(str2num(get(L03150Hz_Corr,'string'))); set(L03150Hz_Corr,'string',0.0);str2num(get(L03150Hz_Corr,'string'));end
            if isempty(str2num(get(L04000Hz_Corr,'string'))); set(L04000Hz_Corr,'string',0.0);str2num(get(L04000Hz_Corr,'string'));end
            if isempty(str2num(get(L05000Hz_Corr,'string'))); set(L05000Hz_Corr,'string',0.0);str2num(get(L05000Hz_Corr,'string'));end
            if isempty(str2num(get(L06300Hz_Corr,'string'))); set(L06300Hz_Corr,'string',0.0);str2num(get(L06300Hz_Corr,'string'));end
            if isempty(str2num(get(L08000Hz_Corr,'string'))); set(L08000Hz_Corr,'string',0.0);str2num(get(L08000Hz_Corr,'string'));end
            if isempty(str2num(get(L10000Hz_Corr,'string'))); set(L10000Hz_Corr,'string',0.0);str2num(get(L10000Hz_Corr,'string'));end
            if isempty(str2num(get(L12500Hz_Corr,'string'))); set(L12500Hz_Corr,'string',0.0);str2num(get(L12500Hz_Corr,'string'));end
            if isempty(str2num(get(L16000Hz_Corr,'string'))); set(L16000Hz_Corr,'string',0.0);str2num(get(L16000Hz_Corr,'string'));end
            if isempty(str2num(get(L20000Hz_Corr,'string'))); set(L20000Hz_Corr,'string',0.0);str2num(get(L20000Hz_Corr,'string'));end


            if isempty(str2num(get(R00020Hz_Corr,'string'))); set(R00020Hz_Corr,'string',0.0);str2num(get(R00020Hz_Corr,'string'));end
            if isempty(str2num(get(R00025Hz_Corr,'string'))); set(R00025Hz_Corr,'string',0.0);str2num(get(R00025Hz_Corr,'string'));end
            if isempty(str2num(get(R000315Hz_Corr,'string'))); set(R000315Hz_Corr,'string',0.0);str2num(get(R000315Hz_Corr,'string'));end
            if isempty(str2num(get(R00040Hz_Corr,'string'))); set(R00040Hz_Corr,'string',0.0);str2num(get(R00040Hz_Corr,'string'));end
            if isempty(str2num(get(R00050Hz_Corr,'string'))); set(R00050Hz_Corr,'string',0.0);str2num(get(R00050Hz_Corr,'string'));end
            if isempty(str2num(get(R00063Hz_Corr,'string'))); set(R00063Hz_Corr,'string',0.0);str2num(get(R00063Hz_Corr,'string'));end
            if isempty(str2num(get(R00080Hz_Corr,'string'))); set(R00080Hz_Corr,'string',0.0);str2num(get(R00080Hz_Corr,'string'));end
            if isempty(str2num(get(R00100Hz_Corr,'string'))); set(R00100Hz_Corr,'string',0.0);str2num(get(R00100Hz_Corr,'string'));end
            if isempty(str2num(get(R00125Hz_Corr,'string'))); set(R00125Hz_Corr,'string',0.0);str2num(get(R00125Hz_Corr,'string'));end
            if isempty(str2num(get(R00160Hz_Corr,'string'))); set(R00160Hz_Corr,'string',0.0);str2num(get(R00160Hz_Corr,'string'));end
            if isempty(str2num(get(R00200Hz_Corr,'string'))); set(R00200Hz_Corr,'string',0.0);str2num(get(R00200Hz_Corr,'string'));end
            if isempty(str2num(get(R00250Hz_Corr,'string'))); set(R00250Hz_Corr,'string',0.0);str2num(get(R00250Hz_Corr,'string'));end
            if isempty(str2num(get(R00315Hz_Corr,'string'))); set(R00315Hz_Corr,'string',0.0);str2num(get(R00315Hz_Corr,'string'));end
            if isempty(str2num(get(R00400Hz_Corr,'string'))); set(R00400Hz_Corr,'string',0.0);str2num(get(R00400Hz_Corr,'string'));end
            if isempty(str2num(get(R00500Hz_Corr,'string'))); set(R00500Hz_Corr,'string',0.0);str2num(get(R00500Hz_Corr,'string'));end
            if isempty(str2num(get(R00630Hz_Corr,'string'))); set(R00630Hz_Corr,'string',0.0);str2num(get(R00630Hz_Corr,'string'));end
            if isempty(str2num(get(R00800Hz_Corr,'string'))); set(R00800Hz_Corr,'string',0.0);str2num(get(R00800Hz_Corr,'string'));end
            if isempty(str2num(get(R01000Hz_Corr,'string'))); set(R01000Hz_Corr,'string',0.0);str2num(get(R01000Hz_Corr,'string'));end
            if isempty(str2num(get(R01250Hz_Corr,'string'))); set(R01250Hz_Corr,'string',0.0);str2num(get(R01250Hz_Corr,'string'));end
            if isempty(str2num(get(R01600Hz_Corr,'string'))); set(R01600Hz_Corr,'string',0.0);str2num(get(R01600Hz_Corr,'string'));end
            if isempty(str2num(get(R02000Hz_Corr,'string'))); set(R02000Hz_Corr,'string',0.0);str2num(get(R02000Hz_Corr,'string'));end
            if isempty(str2num(get(R02500Hz_Corr,'string'))); set(R02500Hz_Corr,'string',0.0);str2num(get(R02500Hz_Corr,'string'));end
            if isempty(str2num(get(R03150Hz_Corr,'string'))); set(R03150Hz_Corr,'string',0.0);str2num(get(R03150Hz_Corr,'string'));end
            if isempty(str2num(get(R04000Hz_Corr,'string'))); set(R04000Hz_Corr,'string',0.0);str2num(get(R04000Hz_Corr,'string'));end
            if isempty(str2num(get(R05000Hz_Corr,'string'))); set(R05000Hz_Corr,'string',0.0);str2num(get(R05000Hz_Corr,'string'));end
            if isempty(str2num(get(R06300Hz_Corr,'string'))); set(R06300Hz_Corr,'string',0.0);str2num(get(R06300Hz_Corr,'string'));end
            if isempty(str2num(get(R08000Hz_Corr,'string'))); set(R08000Hz_Corr,'string',0.0);str2num(get(R08000Hz_Corr,'string'));end
            if isempty(str2num(get(R10000Hz_Corr,'string'))); set(R10000Hz_Corr,'string',0.0);str2num(get(R10000Hz_Corr,'string'));end
            if isempty(str2num(get(R12500Hz_Corr,'string'))); set(R12500Hz_Corr,'string',0.0);str2num(get(R12500Hz_Corr,'string'));end
            if isempty(str2num(get(R16000Hz_Corr,'string'))); set(R16000Hz_Corr,'string',0.0);str2num(get(R16000Hz_Corr,'string'));end
            if isempty(str2num(get(R20000Hz_Corr,'string'))); set(R20000Hz_Corr,'string',0.0);str2num(get(R20000Hz_Corr,'string'));end


            if isempty(str2num(get(B00020Hz_Corr,'string'))); set(B00020Hz_Corr,'string',0.0);str2num(get(B00020Hz_Corr,'string'));end
            if isempty(str2num(get(B00025Hz_Corr,'string'))); set(B00025Hz_Corr,'string',0.0);str2num(get(B00025Hz_Corr,'string'));end
            if isempty(str2num(get(B000315Hz_Corr,'string'))); set(B000315Hz_Corr,'string',0.0);str2num(get(B000315Hz_Corr,'string'));end
            if isempty(str2num(get(B00040Hz_Corr,'string'))); set(B00040Hz_Corr,'string',0.0);str2num(get(B00040Hz_Corr,'string'));end
            if isempty(str2num(get(B00050Hz_Corr,'string'))); set(B00050Hz_Corr,'string',0.0);str2num(get(B00050Hz_Corr,'string'));end
            if isempty(str2num(get(B00063Hz_Corr,'string'))); set(B00063Hz_Corr,'string',0.0);str2num(get(B00063Hz_Corr,'string'));end
            if isempty(str2num(get(B00080Hz_Corr,'string'))); set(B00080Hz_Corr,'string',0.0);str2num(get(B00080Hz_Corr,'string'));end
            if isempty(str2num(get(B00100Hz_Corr,'string'))); set(B00100Hz_Corr,'string',0.0);str2num(get(B00100Hz_Corr,'string'));end
            if isempty(str2num(get(B00125Hz_Corr,'string'))); set(B00125Hz_Corr,'string',0.0);str2num(get(B00125Hz_Corr,'string'));end
            if isempty(str2num(get(B00160Hz_Corr,'string'))); set(B00160Hz_Corr,'string',0.0);str2num(get(B00160Hz_Corr,'string'));end
            if isempty(str2num(get(B00200Hz_Corr,'string'))); set(B00200Hz_Corr,'string',0.0);str2num(get(B00200Hz_Corr,'string'));end
            if isempty(str2num(get(B00250Hz_Corr,'string'))); set(B00250Hz_Corr,'string',0.0);str2num(get(B00250Hz_Corr,'string'));end
            if isempty(str2num(get(B00315Hz_Corr,'string'))); set(B00315Hz_Corr,'string',0.0);str2num(get(B00315Hz_Corr,'string'));end
            if isempty(str2num(get(B00400Hz_Corr,'string'))); set(B00400Hz_Corr,'string',0.0);str2num(get(B00400Hz_Corr,'string'));end
            if isempty(str2num(get(B00500Hz_Corr,'string'))); set(B00500Hz_Corr,'string',0.0);str2num(get(B00500Hz_Corr,'string'));end
            if isempty(str2num(get(B00630Hz_Corr,'string'))); set(B00630Hz_Corr,'string',0.0);str2num(get(B00630Hz_Corr,'string'));end
            if isempty(str2num(get(B00800Hz_Corr,'string'))); set(B00800Hz_Corr,'string',0.0);str2num(get(B00800Hz_Corr,'string'));end
            if isempty(str2num(get(B01000Hz_Corr,'string'))); set(B01000Hz_Corr,'string',0.0);str2num(get(B01000Hz_Corr,'string'));end
            if isempty(str2num(get(B01250Hz_Corr,'string'))); set(B01250Hz_Corr,'string',0.0);str2num(get(B01250Hz_Corr,'string'));end
            if isempty(str2num(get(B01600Hz_Corr,'string'))); set(B01600Hz_Corr,'string',0.0);str2num(get(B01600Hz_Corr,'string'));end
            if isempty(str2num(get(B02000Hz_Corr,'string'))); set(B02000Hz_Corr,'string',0.0);str2num(get(B02000Hz_Corr,'string'));end
            if isempty(str2num(get(B02500Hz_Corr,'string'))); set(B02500Hz_Corr,'string',0.0);str2num(get(B02500Hz_Corr,'string'));end
            if isempty(str2num(get(B03150Hz_Corr,'string'))); set(B03150Hz_Corr,'string',0.0);str2num(get(B03150Hz_Corr,'string'));end
            if isempty(str2num(get(B04000Hz_Corr,'string'))); set(B04000Hz_Corr,'string',0.0);str2num(get(B04000Hz_Corr,'string'));end
            if isempty(str2num(get(B05000Hz_Corr,'string'))); set(B05000Hz_Corr,'string',0.0);str2num(get(B05000Hz_Corr,'string'));end
            if isempty(str2num(get(B06300Hz_Corr,'string'))); set(B06300Hz_Corr,'string',0.0);str2num(get(B06300Hz_Corr,'string'));end
            if isempty(str2num(get(B08000Hz_Corr,'string'))); set(B08000Hz_Corr,'string',0.0);str2num(get(B08000Hz_Corr,'string'));end
            if isempty(str2num(get(B10000Hz_Corr,'string'))); set(B10000Hz_Corr,'string',0.0);str2num(get(B10000Hz_Corr,'string'));end
            if isempty(str2num(get(B12500Hz_Corr,'string'))); set(B12500Hz_Corr,'string',0.0);str2num(get(B12500Hz_Corr,'string'));end
            if isempty(str2num(get(B16000Hz_Corr,'string'))); set(B16000Hz_Corr,'string',0.0);str2num(get(B16000Hz_Corr,'string'));end
            if isempty(str2num(get(B20000Hz_Corr,'string'))); set(B20000Hz_Corr,'string',0.0);str2num(get(B20000Hz_Corr,'string'));end
```

```
        Cal_FileSave(:,1)=Freq;
        Cal_FileSave(:,2)=[str2num(get(L00020Hz_Corr,'string'))
            str2num(get(L00025Hz_Corr,'string'))
            str2num(get(L000315Hz_Corr,'string'))
            str2num(get(L00040Hz_Corr,'string'))
            str2num(get(L00050Hz_Corr,'string'))
            str2num(get(L00063Hz_Corr,'string'))
            str2num(get(L00080Hz_Corr,'string'))
            str2num(get(L00100Hz_Corr,'string'))
            str2num(get(L00125Hz_Corr,'string'))
            str2num(get(L00160Hz_Corr,'string'))
            str2num(get(L00200Hz_Corr,'string'))
            str2num(get(L00250Hz_Corr,'string'))
            str2num(get(L00315Hz_Corr,'string'))
            str2num(get(L00400Hz_Corr,'string'))
            str2num(get(L00500Hz_Corr,'string'))
            str2num(get(L00630Hz_Corr,'string'))
            str2num(get(L00800Hz_Corr,'string'))
            str2num(get(L01000Hz_Corr,'string'))
            str2num(get(L01250Hz_Corr,'string'))
            str2num(get(L01600Hz_Corr,'string'))
            str2num(get(L02000Hz_Corr,'string'))
            str2num(get(L02500Hz_Corr,'string'))
            str2num(get(L03150Hz_Corr,'string'))
            str2num(get(L04000Hz_Corr,'string'))
            str2num(get(L05000Hz_Corr,'string'))
            str2num(get(L06300Hz_Corr,'string'))
            str2num(get(L08000Hz_Corr,'string'))
            str2num(get(L10000Hz_Corr,'string'))
            str2num(get(L12500Hz_Corr,'string'))
            str2num(get(L16000Hz_Corr,'string'))
            str2num(get(L20000Hz_Corr,'string'))];

        Cal_FileSave(:,3)=[str2num(get(R00020Hz_Corr,'string'))
            str2num(get(R00025Hz_Corr,'string'))
            str2num(get(R000315Hz_Corr,'string'))
            str2num(get(R00040Hz_Corr,'string'))
            str2num(get(R00050Hz_Corr,'string'))
            str2num(get(R00063Hz_Corr,'string'))
            str2num(get(R00080Hz_Corr,'string'))
            str2num(get(R00100Hz_Corr,'string'))
            str2num(get(R00125Hz_Corr,'string'))
            str2num(get(R00160Hz_Corr,'string'))
            str2num(get(R00200Hz_Corr,'string'))
            str2num(get(R00250Hz_Corr,'string'))
            str2num(get(R00315Hz_Corr,'string'))
            str2num(get(R00400Hz_Corr,'string'))
            str2num(get(R00500Hz_Corr,'string'))
            str2num(get(R00630Hz_Corr,'string'))
            str2num(get(R00800Hz_Corr,'string'))
            str2num(get(R01000Hz_Corr,'string'))
            str2num(get(R01250Hz_Corr,'string'))
            str2num(get(R01600Hz_Corr,'string'))
            str2num(get(R02000Hz_Corr,'string'))
            str2num(get(R02500Hz_Corr,'string'))
            str2num(get(R03150Hz_Corr,'string'))
            str2num(get(R04000Hz_Corr,'string'))
            str2num(get(R05000Hz_Corr,'string'))
            str2num(get(R06300Hz_Corr,'string'))
            str2num(get(R08000Hz_Corr,'string'))
            str2num(get(R10000Hz_Corr,'string'))
            str2num(get(R12500Hz_Corr,'string'))
            str2num(get(R16000Hz_Corr,'string'))
            str2num(get(R20000Hz_Corr,'string'))];

        Cal_FileSave(:,4)=[str2num(get(B00020Hz_Corr,'string'))
            str2num(get(B00025Hz_Corr,'string'))
            str2num(get(B000315Hz_Corr,'string'))
            str2num(get(B00040Hz_Corr,'string'))
            str2num(get(B00050Hz_Corr,'string'))
            str2num(get(B00063Hz_Corr,'string'))
            str2num(get(B00080Hz_Corr,'string'))
            str2num(get(B00100Hz_Corr,'string'))
            str2num(get(B00125Hz_Corr,'string'))
            str2num(get(B00160Hz_Corr,'string'))
            str2num(get(B00200Hz_Corr,'string'))
            str2num(get(B00250Hz_Corr,'string'))
            str2num(get(B00315Hz_Corr,'string'))
            str2num(get(B00400Hz_Corr,'string'))
            str2num(get(B00500Hz_Corr,'string'))
            str2num(get(B00630Hz_Corr,'string'))
            str2num(get(B00800Hz_Corr,'string'))
            str2num(get(B01000Hz_Corr,'string'))
            str2num(get(B01250Hz_Corr,'string'))
            str2num(get(B01600Hz_Corr,'string'))
            str2num(get(B02000Hz_Corr,'string'))
            str2num(get(B02500Hz_Corr,'string'))
            str2num(get(B03150Hz_Corr,'string'))
            str2num(get(B04000Hz_Corr,'string'))
            str2num(get(B05000Hz_Corr,'string'))
            str2num(get(B06300Hz_Corr,'string'))
            str2num(get(B08000Hz_Corr,'string'))
            str2num(get(B10000Hz_Corr,'string'))
            str2num(get(B12500Hz_Corr,'string'))
            str2num(get(B16000Hz_Corr,'string'))
            str2num(get(B20000Hz_Corr,'string'))];

        %Cal_File
        %  Column 01 - Frequency
        %  Column 02 - Cal Left
        %  Column 03 - Cal Right
        %  Column 04 - Cal Binaural

        [filename, pathname] = uiputfile('*.mat', 'Save Workspace as');
        save ([pathname filename], 'Cal_FileSave');

    end
%% %%%%%%%%%%%%%%%%     FINISH Button
DONE = uicontrol('style', 'pushbutton','string', 'FINISHED','units',...
    'normalized','Visible','on','position', [BL LBB-1*DBH BW BH],...
    'FontSize', Font10, 'FontWeight', 'bold','callback', {@Finish_Cal});
%% %%%%%%%%%%%%%%%%     FINISH
    function Finish_Cal(~,~)
```

```
if isempty(str2num(get(L00020Hz_Corr,'string'))); set(L00020Hz_Corr,'string',0.0);str2num(get(L00020Hz_Corr,'string'));end
if isempty(str2num(get(L00025Hz_Corr,'string'))); set(L00025Hz_Corr,'string',0.0);str2num(get(L00025Hz_Corr,'string'));end
if isempty(str2num(get(L000315Hz_Corr,'string'))); set(L000315Hz_Corr,'string',0.0);str2num(get(L000315Hz_Corr,'string'));end
if isempty(str2num(get(L00040Hz_Corr,'string'))); set(L00040Hz_Corr,'string',0.0);str2num(get(L00040Hz_Corr,'string'));end
if isempty(str2num(get(L00050Hz_Corr,'string'))); set(L00050Hz_Corr,'string',0.0);str2num(get(L00050Hz_Corr,'string'));end
if isempty(str2num(get(L00063Hz_Corr,'string'))); set(L00063Hz_Corr,'string',0.0);str2num(get(L00063Hz_Corr,'string'));end
if isempty(str2num(get(L00080Hz_Corr,'string'))); set(L00080Hz_Corr,'string',0.0);str2num(get(L00080Hz_Corr,'string'));end
if isempty(str2num(get(L00100Hz_Corr,'string'))); set(L00100Hz_Corr,'string',0.0);str2num(get(L00100Hz_Corr,'string'));end
if isempty(str2num(get(L00125Hz_Corr,'string'))); set(L00125Hz_Corr,'string',0.0);str2num(get(L00125Hz_Corr,'string'));end
if isempty(str2num(get(L00160Hz_Corr,'string'))); set(L00160Hz_Corr,'string',0.0);str2num(get(L00160Hz_Corr,'string'));end
if isempty(str2num(get(L00200Hz_Corr,'string'))); set(L00200Hz_Corr,'string',0.0);str2num(get(L00200Hz_Corr,'string'));end
if isempty(str2num(get(L00250Hz_Corr,'string'))); set(L00250Hz_Corr,'string',0.0);str2num(get(L00250Hz_Corr,'string'));end
if isempty(str2num(get(L00315Hz_Corr,'string'))); set(L00315Hz_Corr,'string',0.0);str2num(get(L00315Hz_Corr,'string'));end
if isempty(str2num(get(L00400Hz_Corr,'string'))); set(L00400Hz_Corr,'string',0.0);str2num(get(L00400Hz_Corr,'string'));end
if isempty(str2num(get(L00500Hz_Corr,'string'))); set(L00500Hz_Corr,'string',0.0);str2num(get(L00500Hz_Corr,'string'));end
if isempty(str2num(get(L00630Hz_Corr,'string'))); set(L00630Hz_Corr,'string',0.0);str2num(get(L00630Hz_Corr,'string'));end
if isempty(str2num(get(L00800Hz_Corr,'string'))); set(L00800Hz_Corr,'string',0.0);str2num(get(L00800Hz_Corr,'string'));end
if isempty(str2num(get(L01000Hz_Corr,'string'))); set(L01000Hz_Corr,'string',0.0);str2num(get(L01000Hz_Corr,'string'));end
if isempty(str2num(get(L01250Hz_Corr,'string'))); set(L01250Hz_Corr,'string',0.0);str2num(get(L01250Hz_Corr,'string'));end
if isempty(str2num(get(L01600Hz_Corr,'string'))); set(L01600Hz_Corr,'string',0.0);str2num(get(L01600Hz_Corr,'string'));end
if isempty(str2num(get(L02000Hz_Corr,'string'))); set(L02000Hz_Corr,'string',0.0);str2num(get(L02000Hz_Corr,'string'));end
if isempty(str2num(get(L02500Hz_Corr,'string'))); set(L02500Hz_Corr,'string',0.0);str2num(get(L02500Hz_Corr,'string'));end
if isempty(str2num(get(L03150Hz_Corr,'string'))); set(L03150Hz_Corr,'string',0.0);str2num(get(L03150Hz_Corr,'string'));end
if isempty(str2num(get(L04000Hz_Corr,'string'))); set(L04000Hz_Corr,'string',0.0);str2num(get(L04000Hz_Corr,'string'));end
if isempty(str2num(get(L05000Hz_Corr,'string'))); set(L05000Hz_Corr,'string',0.0);str2num(get(L05000Hz_Corr,'string'));end
if isempty(str2num(get(L06300Hz_Corr,'string'))); set(L06300Hz_Corr,'string',0.0);str2num(get(L06300Hz_Corr,'string'));end
if isempty(str2num(get(L08000Hz_Corr,'string'))); set(L08000Hz_Corr,'string',0.0);str2num(get(L08000Hz_Corr,'string'));end
if isempty(str2num(get(L10000Hz_Corr,'string'))); set(L10000Hz_Corr,'string',0.0);str2num(get(L10000Hz_Corr,'string'));end
if isempty(str2num(get(L12500Hz_Corr,'string'))); set(L12500Hz_Corr,'string',0.0);str2num(get(L12500Hz_Corr,'string'));end
if isempty(str2num(get(L16000Hz_Corr,'string'))); set(L16000Hz_Corr,'string',0.0);str2num(get(L16000Hz_Corr,'string'));end
if isempty(str2num(get(L20000Hz_Corr,'string'))); set(L20000Hz_Corr,'string',0.0);str2num(get(L20000Hz_Corr,'string'));end


if isempty(str2num(get(R00020Hz_Corr,'string'))); set(R00020Hz_Corr,'string',0.0);str2num(get(R00020Hz_Corr,'string'));end
if isempty(str2num(get(R00025Hz_Corr,'string'))); set(R00025Hz_Corr,'string',0.0);str2num(get(R00025Hz_Corr,'string'));end
if isempty(str2num(get(R000315Hz_Corr,'string'))); set(R000315Hz_Corr,'string',0.0);str2num(get(R000315Hz_Corr,'string'));end
if isempty(str2num(get(R00040Hz_Corr,'string'))); set(R00040Hz_Corr,'string',0.0);str2num(get(R00040Hz_Corr,'string'));end
if isempty(str2num(get(R00050Hz_Corr,'string'))); set(R00050Hz_Corr,'string',0.0);str2num(get(R00050Hz_Corr,'string'));end
if isempty(str2num(get(R00063Hz_Corr,'string'))); set(R00063Hz_Corr,'string',0.0);str2num(get(R00063Hz_Corr,'string'));end
if isempty(str2num(get(R00080Hz_Corr,'string'))); set(R00080Hz_Corr,'string',0.0);str2num(get(R00080Hz_Corr,'string'));end
if isempty(str2num(get(R00100Hz_Corr,'string'))); set(R00100Hz_Corr,'string',0.0);str2num(get(R00100Hz_Corr,'string'));end
if isempty(str2num(get(R00125Hz_Corr,'string'))); set(R00125Hz_Corr,'string',0.0);str2num(get(R00125Hz_Corr,'string'));end
if isempty(str2num(get(R00160Hz_Corr,'string'))); set(R00160Hz_Corr,'string',0.0);str2num(get(R00160Hz_Corr,'string'));end
if isempty(str2num(get(R00200Hz_Corr,'string'))); set(R00200Hz_Corr,'string',0.0);str2num(get(R00200Hz_Corr,'string'));end
if isempty(str2num(get(R00250Hz_Corr,'string'))); set(R00250Hz_Corr,'string',0.0);str2num(get(R00250Hz_Corr,'string'));end
if isempty(str2num(get(R00315Hz_Corr,'string'))); set(R00315Hz_Corr,'string',0.0);str2num(get(R00315Hz_Corr,'string'));end
if isempty(str2num(get(R00400Hz_Corr,'string'))); set(R00400Hz_Corr,'string',0.0);str2num(get(R00400Hz_Corr,'string'));end
if isempty(str2num(get(R00500Hz_Corr,'string'))); set(R00500Hz_Corr,'string',0.0);str2num(get(R00500Hz_Corr,'string'));end
if isempty(str2num(get(R00630Hz_Corr,'string'))); set(R00630Hz_Corr,'string',0.0);str2num(get(R00630Hz_Corr,'string'));end
if isempty(str2num(get(R00800Hz_Corr,'string'))); set(R00800Hz_Corr,'string',0.0);str2num(get(R00800Hz_Corr,'string'));end
if isempty(str2num(get(R01000Hz_Corr,'string'))); set(R01000Hz_Corr,'string',0.0);str2num(get(R01000Hz_Corr,'string'));end
if isempty(str2num(get(R01250Hz_Corr,'string'))); set(R01250Hz_Corr,'string',0.0);str2num(get(R01250Hz_Corr,'string'));end
if isempty(str2num(get(R01600Hz_Corr,'string'))); set(R01600Hz_Corr,'string',0.0);str2num(get(R01600Hz_Corr,'string'));end
if isempty(str2num(get(R02000Hz_Corr,'string'))); set(R02000Hz_Corr,'string',0.0);str2num(get(R02000Hz_Corr,'string'));end
if isempty(str2num(get(R02500Hz_Corr,'string'))); set(R02500Hz_Corr,'string',0.0);str2num(get(R02500Hz_Corr,'string'));end
if isempty(str2num(get(R03150Hz_Corr,'string'))); set(R03150Hz_Corr,'string',0.0);str2num(get(R03150Hz_Corr,'string'));end
if isempty(str2num(get(R04000Hz_Corr,'string'))); set(R04000Hz_Corr,'string',0.0);str2num(get(R04000Hz_Corr,'string'));end
if isempty(str2num(get(R05000Hz_Corr,'string'))); set(R05000Hz_Corr,'string',0.0);str2num(get(R05000Hz_Corr,'string'));end
if isempty(str2num(get(R06300Hz_Corr,'string'))); set(R06300Hz_Corr,'string',0.0);str2num(get(R06300Hz_Corr,'string'));end
if isempty(str2num(get(R08000Hz_Corr,'string'))); set(R08000Hz_Corr,'string',0.0);str2num(get(R08000Hz_Corr,'string'));end
if isempty(str2num(get(R10000Hz_Corr,'string'))); set(R10000Hz_Corr,'string',0.0);str2num(get(R10000Hz_Corr,'string'));end
if isempty(str2num(get(R12500Hz_Corr,'string'))); set(R12500Hz_Corr,'string',0.0);str2num(get(R12500Hz_Corr,'string'));end
if isempty(str2num(get(R16000Hz_Corr,'string'))); set(R16000Hz_Corr,'string',0.0);str2num(get(R16000Hz_Corr,'string'));end
if isempty(str2num(get(R20000Hz_Corr,'string'))); set(R20000Hz_Corr,'string',0.0);str2num(get(R20000Hz_Corr,'string'));end


if isempty(str2num(get(B00020Hz_Corr,'string'))); set(B00020Hz_Corr,'string',0.0);str2num(get(B00020Hz_Corr,'string'));end
if isempty(str2num(get(B00025Hz_Corr,'string'))); set(B00025Hz_Corr,'string',0.0);str2num(get(B00025Hz_Corr,'string'));end
if isempty(str2num(get(B000315Hz_Corr,'string'))); set(B000315Hz_Corr,'string',0.0);str2num(get(B000315Hz_Corr,'string'));end
if isempty(str2num(get(B00040Hz_Corr,'string'))); set(B00040Hz_Corr,'string',0.0);str2num(get(B00040Hz_Corr,'string'));end
if isempty(str2num(get(B00050Hz_Corr,'string'))); set(B00050Hz_Corr,'string',0.0);str2num(get(B00050Hz_Corr,'string'));end
if isempty(str2num(get(B00063Hz_Corr,'string'))); set(B00063Hz_Corr,'string',0.0);str2num(get(B00063Hz_Corr,'string'));end
if isempty(str2num(get(B00080Hz_Corr,'string'))); set(B00080Hz_Corr,'string',0.0);str2num(get(B00080Hz_Corr,'string'));end
if isempty(str2num(get(B00100Hz_Corr,'string'))); set(B00100Hz_Corr,'string',0.0);str2num(get(B00100Hz_Corr,'string'));end
if isempty(str2num(get(B00125Hz_Corr,'string'))); set(B00125Hz_Corr,'string',0.0);str2num(get(B00125Hz_Corr,'string'));end
if isempty(str2num(get(B00160Hz_Corr,'string'))); set(B00160Hz_Corr,'string',0.0);str2num(get(B00160Hz_Corr,'string'));end
if isempty(str2num(get(B00200Hz_Corr,'string'))); set(B00200Hz_Corr,'string',0.0);str2num(get(B00200Hz_Corr,'string'));end
if isempty(str2num(get(B00250Hz_Corr,'string'))); set(B00250Hz_Corr,'string',0.0);str2num(get(B00250Hz_Corr,'string'));end
if isempty(str2num(get(B00315Hz_Corr,'string'))); set(B00315Hz_Corr,'string',0.0);str2num(get(B00315Hz_Corr,'string'));end
if isempty(str2num(get(B00400Hz_Corr,'string'))); set(B00400Hz_Corr,'string',0.0);str2num(get(B00400Hz_Corr,'string'));end
if isempty(str2num(get(B00500Hz_Corr,'string'))); set(B00500Hz_Corr,'string',0.0);str2num(get(B00500Hz_Corr,'string'));end
if isempty(str2num(get(B00630Hz_Corr,'string'))); set(B00630Hz_Corr,'string',0.0);str2num(get(B00630Hz_Corr,'string'));end
if isempty(str2num(get(B00800Hz_Corr,'string'))); set(B00800Hz_Corr,'string',0.0);str2num(get(B00800Hz_Corr,'string'));end
if isempty(str2num(get(B01000Hz_Corr,'string'))); set(B01000Hz_Corr,'string',0.0);str2num(get(B01000Hz_Corr,'string'));end
if isempty(str2num(get(B01250Hz_Corr,'string'))); set(B01250Hz_Corr,'string',0.0);str2num(get(B01250Hz_Corr,'string'));end
if isempty(str2num(get(B01600Hz_Corr,'string'))); set(B01600Hz_Corr,'string',0.0);str2num(get(B01600Hz_Corr,'string'));end
if isempty(str2num(get(B02000Hz_Corr,'string'))); set(B02000Hz_Corr,'string',0.0);str2num(get(B02000Hz_Corr,'string'));end
if isempty(str2num(get(B02500Hz_Corr,'string'))); set(B02500Hz_Corr,'string',0.0);str2num(get(B02500Hz_Corr,'string'));end
if isempty(str2num(get(B03150Hz_Corr,'string'))); set(B03150Hz_Corr,'string',0.0);str2num(get(B03150Hz_Corr,'string'));end
if isempty(str2num(get(B04000Hz_Corr,'string'))); set(B04000Hz_Corr,'string',0.0);str2num(get(B04000Hz_Corr,'string'));end
if isempty(str2num(get(B05000Hz_Corr,'string'))); set(B05000Hz_Corr,'string',0.0);str2num(get(B05000Hz_Corr,'string'));end
if isempty(str2num(get(B06300Hz_Corr,'string'))); set(B06300Hz_Corr,'string',0.0);str2num(get(B06300Hz_Corr,'string'));end
if isempty(str2num(get(B08000Hz_Corr,'string'))); set(B08000Hz_Corr,'string',0.0);str2num(get(B08000Hz_Corr,'string'));end
if isempty(str2num(get(B10000Hz_Corr,'string'))); set(B10000Hz_Corr,'string',0.0);str2num(get(B10000Hz_Corr,'string'));end
if isempty(str2num(get(B12500Hz_Corr,'string'))); set(B12500Hz_Corr,'string',0.0);str2num(get(B12500Hz_Corr,'string'));end
if isempty(str2num(get(B16000Hz_Corr,'string'))); set(B16000Hz_Corr,'string',0.0);str2num(get(B16000Hz_Corr,'string'));end
if isempty(str2num(get(B20000Hz_Corr,'string'))); set(B20000Hz_Corr,'string',0.0);str2num(get(B20000Hz_Corr,'string'));end


Cal_File(1,7) = str2num(get(L00020Hz_Corr,'string'));
Cal_File(2,7) = str2num(get(L00025Hz_Corr,'string'));
Cal_File(3,7) = str2num(get(L000315Hz_Corr,'string'));
Cal_File(4,7) = str2num(get(L00040Hz_Corr,'string'));
Cal_File(5,7) = str2num(get(L00050Hz_Corr,'string'));
Cal_File(6,7) = str2num(get(L00063Hz_Corr,'string'));
Cal_File(7,7) = str2num(get(L00080Hz_Corr,'string'));
Cal_File(8,7) = str2num(get(L00100Hz_Corr,'string'));
Cal_File(9,7) = str2num(get(L00125Hz_Corr,'string'));
Cal_File(10,7) = str2num(get(L00160Hz_Corr,'string'));
Cal_File(11,7) = str2num(get(L00200Hz_Corr,'string'));
Cal_File(12,7) = str2num(get(L00250Hz_Corr,'string'));
```

```
            Cal_File(13,7) = str2num(get(L00315Hz_Corr,'string'));
            Cal_File(14,7) = str2num(get(L00400Hz_Corr,'string'));
            Cal_File(15,7) = str2num(get(L00500Hz_Corr,'string'));
            Cal_File(16,7) = str2num(get(L00630Hz_Corr,'string'));
            Cal_File(17,7) = str2num(get(L00800Hz_Corr,'string'));
            Cal_File(18,7) = str2num(get(L01000Hz_Corr,'string'));
            Cal_File(19,7) = str2num(get(L01250Hz_Corr,'string'));
            Cal_File(20,7) = str2num(get(L01600Hz_Corr,'string'));
            Cal_File(21,7) = str2num(get(L02000Hz_Corr,'string'));
            Cal_File(22,7) = str2num(get(L02500Hz_Corr,'string'));
            Cal_File(23,7) = str2num(get(L03150Hz_Corr,'string'));
            Cal_File(24,7) = str2num(get(L04000Hz_Corr,'string'));
            Cal_File(25,7) = str2num(get(L05000Hz_Corr,'string'));
            Cal_File(26,7) = str2num(get(L06300Hz_Corr,'string'));
            Cal_File(27,7) = str2num(get(L08000Hz_Corr,'string'));
            Cal_File(28,7) = str2num(get(L10000Hz_Corr,'string'));
            Cal_File(29,7) = str2num(get(L12500Hz_Corr,'string'));
            Cal_File(30,7) = str2num(get(L16000Hz_Corr,'string'));
            Cal_File(31,7) = str2num(get(L20000Hz_Corr,'string')); %#ok<*ST2NM>

            Cal_File(1,8) = str2num(get(R00020Hz_Corr,'string'));
            Cal_File(2,8) = str2num(get(R00025Hz_Corr,'string'));
            Cal_File(3,8) = str2num(get(R000315Hz_Corr,'string'));
            Cal_File(4,8) = str2num(get(R00040Hz_Corr,'string'));
            Cal_File(5,8) = str2num(get(R00050Hz_Corr,'string'));
            Cal_File(6,8) = str2num(get(R00063Hz_Corr,'string'));
            Cal_File(7,8) = str2num(get(R00080Hz_Corr,'string'));
            Cal_File(8,8) = str2num(get(R00100Hz_Corr,'string'));
            Cal_File(9,8) = str2num(get(R00125Hz_Corr,'string'));
            Cal_File(10,8) = str2num(get(R00160Hz_Corr,'string'));
            Cal_File(11,8) = str2num(get(R00200Hz_Corr,'string'));
            Cal_File(12,8) = str2num(get(R00250Hz_Corr,'string'));
            Cal_File(13,8) = str2num(get(R00315Hz_Corr,'string'));
            Cal_File(14,8) = str2num(get(R00400Hz_Corr,'string'));
            Cal_File(15,8) = str2num(get(R00500Hz_Corr,'string'));
            Cal_File(16,8) = str2num(get(R00630Hz_Corr,'string'));
            Cal_File(17,8) = str2num(get(R00800Hz_Corr,'string'));
            Cal_File(18,8) = str2num(get(R01000Hz_Corr,'string'));
            Cal_File(19,8) = str2num(get(R01250Hz_Corr,'string'));
            Cal_File(20,8) = str2num(get(R01600Hz_Corr,'string'));
            Cal_File(21,8) = str2num(get(R02000Hz_Corr,'string'));
            Cal_File(22,8) = str2num(get(R02500Hz_Corr,'string'));
            Cal_File(23,8) = str2num(get(R03150Hz_Corr,'string'));
            Cal_File(24,8) = str2num(get(R04000Hz_Corr,'string'));
            Cal_File(25,8) = str2num(get(R05000Hz_Corr,'string'));
            Cal_File(26,8) = str2num(get(R06300Hz_Corr,'string'));
            Cal_File(27,8) = str2num(get(R08000Hz_Corr,'string'));
            Cal_File(28,8) = str2num(get(R10000Hz_Corr,'string'));
            Cal_File(29,8) = str2num(get(R12500Hz_Corr,'string'));
            Cal_File(30,8) = str2num(get(R16000Hz_Corr,'string'));
            Cal_File(31,8) = str2num(get(R20000Hz_Corr,'string')); %#ok<*ST2NM>

            Cal_File(1,15) = str2num(get(B00020Hz_Corr,'string'));
            Cal_File(2,15) = str2num(get(B00025Hz_Corr,'string'));
            Cal_File(3,15) = str2num(get(B000315Hz_Corr,'string'));
            Cal_File(4,15) = str2num(get(B00040Hz_Corr,'string'));
            Cal_File(5,15) = str2num(get(B00050Hz_Corr,'string'));
            Cal_File(6,15) = str2num(get(B00063Hz_Corr,'string'));
            Cal_File(7,15) = str2num(get(B00080Hz_Corr,'string'));
            Cal_File(8,15) = str2num(get(B00100Hz_Corr,'string'));
            Cal_File(9,15) = str2num(get(B00125Hz_Corr,'string'));
            Cal_File(10,15) = str2num(get(B00160Hz_Corr,'string'));
            Cal_File(11,15) = str2num(get(B00200Hz_Corr,'string'));
            Cal_File(12,15) = str2num(get(B00250Hz_Corr,'string'));
            Cal_File(13,15) = str2num(get(B00315Hz_Corr,'string'));
            Cal_File(14,15) = str2num(get(B00400Hz_Corr,'string'));
            Cal_File(15,15) = str2num(get(B00500Hz_Corr,'string'));
            Cal_File(16,15) = str2num(get(B00630Hz_Corr,'string'));
            Cal_File(17,15) = str2num(get(B00800Hz_Corr,'string'));
            Cal_File(18,15) = str2num(get(B01000Hz_Corr,'string'));
            Cal_File(19,15) = str2num(get(B01250Hz_Corr,'string'));
            Cal_File(20,15) = str2num(get(B01600Hz_Corr,'string'));
            Cal_File(21,15) = str2num(get(B02000Hz_Corr,'string'));
            Cal_File(22,15) = str2num(get(B02500Hz_Corr,'string'));
            Cal_File(23,15) = str2num(get(B03150Hz_Corr,'string'));
            Cal_File(24,15) = str2num(get(B04000Hz_Corr,'string'));
            Cal_File(25,15) = str2num(get(B05000Hz_Corr,'string'));
            Cal_File(26,15) = str2num(get(B06300Hz_Corr,'string'));
            Cal_File(27,15) = str2num(get(B08000Hz_Corr,'string'));
            Cal_File(28,15) = str2num(get(B10000Hz_Corr,'string'));
            Cal_File(29,15) = str2num(get(B12500Hz_Corr,'string'));
            Cal_File(30,15) = str2num(get(B16000Hz_Corr,'string'));
            Cal_File(31,15) = str2num(get(B20000Hz_Corr,'string')); %#ok<*ST2NM>

           for GetCal = 1:size(Freq)
                Cal_File(GetCal,1)=Freq(GetCal,1);

                Cal_File(GetCal,2)=Cal_File(GetCal,7);
                Cal_File(GetCal,3)=Cal_File(GetCal,8);

            end

        setappdata(0,'evalue',Cal_File);
        TransferData=getappdata(0,'evalue');
        %set(MainWindow,'Visible','on')
        close
    end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%INSERT SIGNAL PLAYBACK HERE%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%(Code written by others) %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
End
```

A.2.3 Calibration Settings (ExpCal_Settings.m)

```
function Cal_Settings

set(0,'units','pixels');
RelFont = get(0,'ScreenSize'); %#ok<NASGU>
Font8=round(8/1080*RelFont(1,4)); %Font 8 on 1920x1080
Font10=round(10/1080*RelFont(1,4));
Font12=round(12/1080*RelFont(1,4));
Font16=round(16/1080*RelFont(1,4));

ScrnHeight=RelFont(1,3);

CurrentSettings= getappdata(0,'evalue');


set(0,'Units','normalized');
scnsize = get(0,'ScreenSize'); %#ok<NASGU>
Background_Color=[0.8 0.8 0.8];

Width=0.125;
CenterH=0.5-(Width/2);
Height=0.1/1920*ScrnHeight;
CenterV=0.5-(Height/2);

%set(MainWindow,'Visible','off')
Cal_Settings_Box= figure('units', 'normalized','Position',[CenterH CenterV Width Height],...
    'Color',Background_Color);
set(Cal_Settings_Box,'name','Cal. Settings','numbertitle','off')
set(Cal_Settings_Box,'MenuBar','none');

a=get(Cal_Settings_Box,'Position');


%% Collect Current Values
DefaultUserSettings=csvread('UserSettings.txt');
playtime1 = DefaultUserSettings(1,5);
pausetime1 = DefaultUserSettings(2,5);
Fade1 = DefaultUserSettings(3,5);
Phase1 = DefaultUserSettings(4,5);
playtime2 = DefaultUserSettings(5,5);
pausetime2 = DefaultUserSettings(6,5);
Fade2 = DefaultUserSettings(7,5);
Phase2 = DefaultUserSettings(8,5);
Default_Cal_Playback_Length = DefaultUserSettings(9,5);
Default_Cal_Fade = DefaultUserSettings(10,5);

Cur_Playback = CurrentSettings(1,4);
Cur_Pause = CurrentSettings(2,4);
Cur_Fade = CurrentSettings(3,4);
Cur_Increase1 = CurrentSettings(4,4);
Cur_Increase2 = CurrentSettings(5,4);
Default_Oct = CurrentSettings(6,4);
Default_startvalue = CurrentSettings(7,4);
Default_endvalue = CurrentSettings(8,4);
% Default_Cal_Playback_Length = CurrentSettings(9,4);
% Default_Cal_Fade = CurrentSettings(10,4);



Bottom_Align=0.75;
Width1=0.35;
Width2=0.3;
Width3=0.35;
Height1=(0.18)/1920*ScrnHeight;
Height2=(0.2)/1920*ScrnHeight;
dh=(Height2-Height1)*0.5;

Bottom_Block=0.15;
HoriSpace=0.05;

VertSpace=(1-Bottom_Block-2*Height2)*(1/6);
Left_Align=0.075;
UnitAlign = (Left_Align+Width1+HoriSpace+Width2+0.01);

%% Settings
Cal_Playback_Text = uicontrol('style','text', 'units', 'normalized',...
    'position', [Left_Align Bottom_Align+dh Width1 Height1], 'string', 'Playback Length',...
    'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Right');
Cal_Playback_Edit = uicontrol('style','edit', 'units', 'normalized',...
    'position', [Left_Align+Width1+HoriSpace Bottom_Align Width2 Height2], 'string', Default_Cal_Playback_Length,...
    'visible','on', 'FontSize', Font8);
Cal_Playback_Units = uicontrol('style','text', 'units', 'normalized',...
    'position', [UnitAlign Bottom_Align+dh Width3 Height1], 'string', 'Sec.',...
    'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left');

Cal_Fade_Text = uicontrol('style','text', 'units', 'normalized',...
    'position', [Left_Align Bottom_Align-(VertSpace+Height2)+dh Width1 Height1], 'string', 'Fade Length',...
    'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Right');
Cal_Fade_Edit = uicontrol('style','edit', 'units', 'normalized',...
    'position', [Left_Align+Width1+HoriSpace Bottom_Align-(VertSpace+Height2) Width2 Height2], 'string', Default_Cal_Fade,...
    'visible','on', 'FontSize', Font8);
Cal_Fade_Units = uicontrol('style','text', 'units', 'normalized',...
    'position', [UnitAlign Bottom_Align-(VertSpace+Height2)+dh Width3 Height1], 'string', 'ms.',...
    'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left');

%% SAVE AND CANCEL Button Positions
Button_Width =0.3;
Spacing=0.1;
Button_Bottom=0.05;
Button_Height=0.30/1920*ScrnHeight;
Button_Left = 0.5*(1-Spacing-2*Button_Width);

%SAVE SETTINGS
Cal_Save_Button=uicontrol('style', 'pushbutton','string', 'Save','units',...
    'normalized','Visible','on','position', [Button_Left Button_Bottom Button_Width Button_Height],...
    'FontSize', Font12, 'FontWeight', 'bold','callback', {@Cal_Save_Settings});

    function Cal_Save_Settings(~,~)
        NewValue_Cal_Playback=str2num(get(Cal_Playback_Edit,'string'));
        NewValue_Cal_Fade=str2num(get(Cal_Fade_Edit,'string'));
```

```
        CurrentSettings(9,4)=NewValue_Cal_Playback;
        CurrentSettings(10,4)=NewValue_Cal_Fade;
        setappdata(0,'evalue',CurrentSettings);

        DefaultUserSettings(9,5)=NewValue_Cal_Playback;
        DefaultUserSettings(10,5)=NewValue_Cal_Fade;
                  csvwrite('UserSettings.txt',DefaultUserSettings);


        disp('Calibration Settings Saved')
        close
    end

%CANCEL SETTINGS
Cal_Cancel_Button=uicontrol('style', 'pushbutton','string', 'Cancel','units',...
    'normalized','Visible','on','position', [Button_Left+Button_Width+Spacing Button_Bottom Button_Width Button_Height],...
    'FontSize', Font12, 'FontWeight', 'bold','callback', {@Cal_Cancel_Settings});

    function Cal_Cancel_Settings(~,~)
        disp('Calibration Settings Not Saved')
        close
    end

%CANCEL SETTINGS CHANGE
end
```

A.2.4 User Settings - Pure Tones (ExpUserSettings.m)

```
function UserSettings ()
% Callout Figure allowing for an easy change of operating settings
%   - Tone Length
%   - Pause Length
%   - Fade Value
%   - Fade Window Value

global Octave_String1 ThirdOctave_String1

%%%%%%%%%%%%%%%% CHANGE OUT!!!!!!!!! %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

set(0,'units','pixels');
RelFont = get(0,'ScreenSize'); %#ok<NASGU>
Font8=round(8/1080*RelFont(1,4)); %Font 8 on 1920x1080
Font10=round(10/1080*RelFont(1,4));
Font12=round(12/1080*RelFont(1,4));
Font16=round(16/1080*RelFont(1,4));



set(0,'Units','normalized');
scnsize = get(0,'ScreenSize'); %#ok<NASGU>
Background_Color=[0.8 0.8 0.8];

%set(MainWindow,'Visible','off')
Settings_Box= figure('units', 'normalized','Position',[0.3125 0.4 0.25 0.2],...
    'Color',Background_Color);
set(Settings_Box,'name','User Settings','numbertitle','off')
set(Settings_Box,'MenuBar','none');

a=get(Settings_Box,'Position');


%% Collect Current Values
CurrentSettings= getappdata(0,'evalue');

        DefaultUserSettings=csvread('UserSettings.txt');
            Cur_Playback1 = DefaultUserSettings(1,5);
            Cur_Pause1 = DefaultUserSettings(2,5);
            Cur_Fade1 = DefaultUserSettings(3,5);
            Cur_Phase1 = DefaultUserSettings(4,5);
            Cur_Playback2 = DefaultUserSettings(5,5);
            Cur_Pause2 = DefaultUserSettings(6,5);
            Cur_Fade2 = DefaultUserSettings(7,5);
            Cur_Phase2 = DefaultUserSettings(8,5);

% evalue Placeholders
% Column 1:
% Column 2:
% Column 3:
% Column 4: User Settings
% Column 5: Resultant L-Small (:,3)
% Column 6: Resultant R-Small (:,5)
% Column 7: Headphones 1 - Left
% Column 8: Headphones 1 - Right
% Column 9: Headphones 2 - Left
% Column 10: Headphones 2 - Right
% Column 11: Ambient Conditions Left
% Column 12: Ambient Conditions Right
% Column 13: Maximum Limits

%% Settings Buttons and Edit Boxes

%Button Positioning

Bottom_Align1=0.8;
Width11=0.35/2;
Width2=0.3/4;
Width3=0.35/6;
Height11=(0.075);
Height2=(0.09);
dh1=(Height2-Height11)*0.5;

Bottom_Block=0.35;
HoriSpace=0.05;

VertSpace=(1-Bottom_Block-5*Height2)*(1/5);
Left_Align1=0.075;
UnitAlign1 = (Left_Align1+Width11+HoriSpace+Width2+0.01);

Bottom_Align2=0.8;
```

```
Width21=0.35/2;
Width2=0.3/4;
Width3=0.35/6;
Height21=(0.075);
Height2=(0.09);
dh2=(Height2-Height21)*0.5;

Bottom_Block=0.35;
HoriSpace=0.05;

VertSpace=(1-Bottom_Block-5*Height2)*(1/5);
Left_Align2=0.525;
UnitAlign2 = (Left_Align2+Width21+HoriSpace+Width2+0.01);

%% Variable Descriptions
    function PlaybackAbout (~,~)
        TH=0.2; %Text Height
        TL1=0.1;   %Text Left
        TW1=0.2; %Text Width - Variable
        TW2=(1-TL1-TW1-TL1); %Text Width - Description
        VS=(1/5)*(1-4*TH); %Vertical Spacing
        TB1=1-VS-TH; %Text Bottom

        AboutBox=figure;
        set(AboutBox,'name','Variable Descriptions','numbertitle','off')
        set(AboutBox,'MenuBar','none');

        PlayDesc='The Playback length determines how long each pure tone will be presented during the assessment';
        PauseDesc='The Pause length determines how the pause will last between tone presentations';
        FadeDesc=' Fade defines the leading and trailing faded-in and faded-out lengths used during the signal presentation. The fade-in and
fade-out durations are specified in unit milliseconds. ';

        Playback_Desc = uicontrol('style','text', 'units', 'normalized',...
            'position', [TL1 TB1 TW1 TH], 'string', 'Playback Length',...
            'visible','on', 'FontSize', Font10,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left','FontWeight','Bold');
        Playback_Desc2 = uicontrol('style','text', 'units', 'normalized',...
            'position', [TL1+TW1 TB1 TW2 TH], 'string', PlayDesc,...
            'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left');

        Pause_Desc = uicontrol('style','text', 'units', 'normalized',...
            'position', [TL1 TB1-(VS+TH) TW1 TH], 'string', 'Pause Length',...
            'visible','on', 'FontSize', Font10,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left','FontWeight','Bold');
        Pause_Desc2 = uicontrol('style','text', 'units', 'normalized',...
            'position', [TL1+TW1 TB1-(VS+TH) TW2 TH], 'string', PauseDesc,...
            'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left');

        Fade_Desc = uicontrol('style','text', 'units', 'normalized',...
            'position', [TL1 TB1-2*(VS+TH) TW1 TH], 'string', 'Fade Length',...
            'visible','on', 'FontSize', Font10,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left','FontWeight','Bold');
        Fade_Desc2 = uicontrol('style','text', 'units', 'normalized',...
            'position', [TL1+TW1 TB1-2*(VS+TH) TW2 TH], 'string', FadeDesc,...
            'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left');

%         Fade_Desc = uicontrol('style','text', 'units', 'normalized',...
%             'position', [TL1 TB1-3*(VS+TH) TW1 TH], 'string', 'Fade Length',...
%             'visible','on', 'FontSize', Font10,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left','FontWeight','Bold');
%         Fade_Desc2 = uicontrol('style','text', 'units', 'normalized',...
%             'position', [TL1+TW1 TB1-3*(VS+TH) TW2 TH], 'string', FadeDesc,...
%             'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left');
%
    end

PlaybackMenu=uicontextmenu;
item1=uimenu(PlaybackMenu,'Label','What''s This?','Callback',@PlaybackAbout);

%% Setting Controls
GeneralFrame1 = uicontrol('style','frame', 'units', 'normalized',...
    'position', [0.05 0.25 0.40 0.70], 'visible','on', 'FontSize', Font10,'BackgroundColor',[0.8 0.8
0.8],'HorizontalAlignment','Left','FontWeight','Bold');

        Frame1_Text = uicontrol('style','text', 'units', 'normalized',...
            'position', [0.075 0.90 Width11 Height11], 'string', 'Signal 1 Settings',...
            'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],...
            'HorizontalAlignment','Left');

        Playback_Text = uicontrol('style','text', 'units', 'normalized',...
            'position', [Left_Align1 Bottom_Align1+dh1 Width11 Height11], 'string', 'Playback Length',...
            'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left');
        Playback_Edit1 = uicontrol('style','edit', 'units', 'normalized',...
            'position', [Left_Align1+Width11+HoriSpace Bottom_Align1 Width2 Height2], 'string', Cur_Playback1,...
            'visible','on', 'FontSize', Font8,'UIContextMenu',PlaybackMenu);
        Playback_Units = uicontrol('style','text', 'units', 'normalized',...
            'position', [UnitAlign1 Bottom_Align1+dh1 Width3 Height11], 'string', 'Sec.',...
            'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left');

        Fade_Text = uicontrol('style','text', 'units', 'normalized',...
            'position', [Left_Align1 Bottom_Align1-1*(VertSpace+Height2)+dh1 Width11 Height11], 'string', 'Fade Length',...
            'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left');
        Fade_Edit1 = uicontrol('style','edit', 'units', 'normalized',...
            'position', [Left_Align1+Width11+HoriSpace Bottom_Align1-1*(VertSpace+Height2) Width2 Height2], 'string', Cur_Fade1,...
            'visible','on', 'FontSize', Font8,'UIContextMenu',PlaybackMenu);
        Fade_Units = uicontrol('style','text', 'units', 'normalized',...
            'position', [UnitAlign1 Bottom_Align1-1*(VertSpace+Height2)+dh1 Width3 Height11], 'string', 'ms.',...
            'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left');

        Pause_Text = uicontrol('style','text', 'units', 'normalized',...
            'position', [Left_Align1 Bottom_Align1-2*(VertSpace+Height2)+dh1 Width11 Height11], 'string', 'Pause Length',...
            'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left');
        Pause_Edit1 = uicontrol('style','edit', 'units', 'normalized',...
            'position', [Left_Align1+Width11+HoriSpace Bottom_Align1-2*(VertSpace+Height2) Width2 Height2], 'string', Cur_Pause1,...
            'visible','on', 'FontSize', Font8,'UIContextMenu',PlaybackMenu);
        Pause_Units = uicontrol('style','text', 'units', 'normalized',...
            'position', [UnitAlign1 Bottom_Align1-2*(VertSpace+Height2)+dh1 Width3 Height11], 'string', 'sec.',...
            'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left');

        Phase_Text = uicontrol('style','text', 'units', 'normalized',...
            'position', [Left_Align1 Bottom_Align1-3*(VertSpace+Height2)+dh1 Width11+0.1 Height11], 'string', 'Binaural Phase Diff.',...
            'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left');
        Phase_Edit1 = uicontrol('style','edit', 'units', 'normalized',...
            'position', [Left_Align1+Width11+HoriSpace Bottom_Align1-3*(VertSpace+Height2) Width2 Height2], 'string', Cur_Phase1,...
            'visible','on', 'FontSize', Font8,'UIContextMenu',PlaybackMenu);
        Phase_Units = uicontrol('style','text', 'units', 'normalized',...
            'position', [UnitAlign1 Bottom_Align1-3*(VertSpace+Height2)+dh1 Width3 Height11], 'string', 'deg.',...
            'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left');
```

```
R_Width1=0.15;
DB_Width=0.20;


GeneralFrame2 = uicontrol('style','frame', 'units', 'normalized',...
    'position', [0.5 0.25 0.40 0.70], 'visible','on', 'FontSize', Font10,'BackgroundColor',[0.8 0.8
0.8],'HorizontalAlignment','Left','FontWeight','Bold');

        Frame2_Text = uicontrol('style','text', 'units', 'normalized',...
            'position', [0.525 0.90 Width21 Height21], 'string', 'Signal 2 Settings',...
            'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left');

        Playback_Text = uicontrol('style','text', 'units', 'normalized',...
            'position', [Left_Align2 Bottom_Align2+dh2 Width21 Height21], 'string', 'Playback Length',...
            'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left');
        Playback_Edit2 = uicontrol('style','edit', 'units', 'normalized',...
            'position', [Left_Align2+Width21+HoriSpace Bottom_Align2 Width2 Height2], 'string', Cur_Playback2,...
            'visible','on', 'FontSize', Font8,'UIContextMenu',PlaybackMenu);
        Playback_Units = uicontrol('style','text', 'units', 'normalized',...
            'position', [UnitAlign2 Bottom_Align2+dh2 Width3 Height21], 'string', 'Sec.',...
            'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left');

        Fade_Text = uicontrol('style','text', 'units', 'normalized',...
            'position', [Left_Align2 Bottom_Align2-1*(VertSpace+Height2)+dh2 Width21 Height21], 'string', 'Fade Length',...
            'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left');
        Fade_Edit2 = uicontrol('style','edit', 'units', 'normalized',...
            'position', [Left_Align2+Width21+HoriSpace Bottom_Align2-1*(VertSpace+Height2) Width2 Height2], 'string', Cur_Fade2,...
            'visible','on', 'FontSize', Font8,'UIContextMenu',PlaybackMenu);
        Fade_Units = uicontrol('style','text', 'units', 'normalized',...
            'position', [UnitAlign2 Bottom_Align2-1*(VertSpace+Height2)+dh2 Width3 Height21], 'string', 'ms.',...
            'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left');

        Pause_Text = uicontrol('style','text', 'units', 'normalized',...
            'position', [Left_Align2 Bottom_Align2-2*(VertSpace+Height2)+dh2 Width21 Height21], 'string', 'Pause Length',...
            'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left');
        Pause_Edit2 = uicontrol('style','edit', 'units', 'normalized',...
            'position', [Left_Align2+Width21+HoriSpace Bottom_Align2-2*(VertSpace+Height2) Width2 Height2], 'string', Cur_Pause2,...
            'visible','on', 'FontSize', Font8,'UIContextMenu',PlaybackMenu);
        Pause_Units = uicontrol('style','text', 'units', 'normalized',...
            'position', [UnitAlign2 Bottom_Align2-2*(VertSpace+Height2)+dh2 Width3 Height21], 'string', 'sec.',...
            'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left');

        Phase_Text = uicontrol('style','text', 'units', 'normalized',...
            'position', [Left_Align2 Bottom_Align2-3*(VertSpace+Height2)+dh2 Width21+0.1 Height21], 'string', 'Binaural Phase Diff.',...
            'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left');
        Phase_Edit2 = uicontrol('style','edit', 'units', 'normalized',...
            'position', [Left_Align2+Width21+HoriSpace Bottom_Align2-3*(VertSpace+Height2) Width2 Height2], 'string', Cur_Phase2,...
            'visible','on', 'FontSize', Font8,'UIContextMenu',PlaybackMenu);
        Phase_Units = uicontrol('style','text', 'units', 'normalized',...
            'position', [UnitAlign2 Bottom_Align2-3*(VertSpace+Height2)+dh2 Width3 Height21], 'string', 'deg.',...
            'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left');


%% Button Positions
Button_Width =0.3;
Spacing=0.025;
Button_Bottom=0.05;
Button_Height=0.15;
Button_Left = 0.025;

%SAVE SETTINGS
Save_Button=uicontrol('style', 'pushbutton','string', 'Save','units',...
    'normalized','Visible','on','position', [Button_Left Button_Bottom Button_Width Button_Height],...
    'FontSize', Font12, 'FontWeight', 'bold','callback', {@Save_Settings});


    function Save_Settings(~,~)
        CurrentSettings= getappdata(0,'evalue');

        DefaultUserSettings=csvread('UserSettings.txt');

        NewValue_Playback1=str2num(get(Playback_Edit1,'string'));
        NewValue_Pause1=str2num(get(Pause_Edit1,'string'));
        NewValue_Fade1=str2num(get(Fade_Edit1,'string'));
        NewValue_Phase1=str2num(get(Phase_Edit1,'string'));
        NewValue_Playback2=str2num(get(Playback_Edit2,'string'));
        NewValue_Pause2=str2num(get(Pause_Edit2,'string'));
        NewValue_Fade2=str2num(get(Fade_Edit2,'string'));
        NewValue_Phase2=str2num(get(Phase_Edit2,'string'));



        CurrentSettings(1,4)=NewValue_Playback1;
        CurrentSettings(2,4)=NewValue_Pause1;
        CurrentSettings(3,4)=NewValue_Fade1;
        CurrentSettings(4,4)=NewValue_Phase1;
        CurrentSettings(5,4)=NewValue_Playback2;
        CurrentSettings(6,4)=NewValue_Pause2;
        CurrentSettings(7,4)=NewValue_Fade2;
        CurrentSettings(8,4)=NewValue_Phase2;
        setappdata(0,'evalue',CurrentSettings);

        DefaultUserSettings=csvread('UserSettings.txt');
        DefaultUserSettings(1,5)=NewValue_Playback1;
        DefaultUserSettings(2,5)=NewValue_Pause1;
        DefaultUserSettings(3,5)=NewValue_Fade1;
        DefaultUserSettings(4,5)=NewValue_Phase1;
        DefaultUserSettings(5,5)=NewValue_Playback2;
        DefaultUserSettings(6,5)=NewValue_Pause2;
        DefaultUserSettings(7,5)=NewValue_Fade2;
        DefaultUserSettings(8,5)=NewValue_Phase2;

            csvwrite('UserSettings.txt',DefaultUserSettings);



        disp('Settings Saved')
        close
    end
```

```
%CANCEL SETTINGS
Cancel_Button=uicontrol('style', 'pushbutton','string', 'Cancel','units',...
    'normalized','Visible','on','position', [Button_Left+Button_Width+Spacing Button_Bottom Button_Width Button_Height],...
    'FontSize', Font12, 'FontWeight', 'bold','callback', {@Cancel_Settings});

    function Cancel_Settings(~,~)
        disp('Settings Not Saved')
        close
    end

%LIMIT SETTINGS
Limit_Button=uicontrol('style', 'pushbutton','string', 'Limits','units',...
    'normalized','Visible','on','position', [0.675 Button_Bottom Button_Width Button_Height],...
    'FontSize', Font12, 'FontWeight', 'bold','callback', {@Exp_Limit_Settings});

    function Exp_Limit_Settings(~,~)
        Limit_Settings
    end

end
```

A.2.5 User Settings - Noise Signals (ExpNoiseSettings.m)

```
function ExpNoiseSettings ()
% Callout Figure allowing for an easy change of operating settings
%   - Tone Length
%   - Pause Length
%   - Fade Value
%   - Fade Window Value

global Octave_String1 ThirdOctave_String1

%%%%%%%%%%%%%%%%% CHANGE OUT!!!!!!!!! %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

set(0,'units','pixels');
RelFont = get(0,'ScreenSize'); %#ok<NASGU>
Font8=round(8/1080*RelFont(1,4)); %Font 8 on 1920x1080
Font10=round(10/1080*RelFont(1,4));
Font12=round(12/1080*RelFont(1,4));
Font16=round(16/1080*RelFont(1,4));



set(0,'Units','normalized');
scnsize = get(0,'ScreenSize'); %#ok<NASGU>
Background_Color=[0.8 0.8 0.8];

%set(MainWindow,'Visible','off')
Settings_Box= figure('units', 'normalized','Position',[0.3125 0.4 0.25 0.2],...
    'Color',Background_Color);
set(Settings_Box,'name','User Settings','numbertitle','off')
set(Settings_Box,'MenuBar','none');

a=get(Settings_Box,'Position');


%% Collect Current Values
CurrentSettings= getappdata(0,'evalue');

        DefaultUserSettings=csvread('UserSettings.txt');
            Cur_Playback1 = DefaultUserSettings(1,5);
            Cur_Pause1 = DefaultUserSettings(2,5);
            Cur_Fade1 = DefaultUserSettings(3,5);
            Cur_Phase1 = DefaultUserSettings(4,5);
            Cur_Playback2 = DefaultUserSettings(5,5);
            Cur_Pause2 = DefaultUserSettings(6,5);
            Cur_Fade2 = DefaultUserSettings(7,5);
            Cur_Phase2 = DefaultUserSettings(8,5);

% evalue Placeholders
% Column 1:
% Column 2:
% Column 3:
% Column 4: User Settings
% Column 5: Resultant L-Small (:,3)
% Column 6: Resultant R-Small (:,5)
% Column 7: Headphones 1 - Left
% Column 8: Headphones 1 - Right
% Column 9: Headphones 2 - Left
% Column 10: Headphones 2 - Right
% Column 11: Ambient Conditions Left
% Column 12: Ambient Conditions Right
% Column 13: Maximum Limits

%% Settings Buttons and Edit Boxes

%Button Positioning

Bottom_Align1=0.8;
Width11=0.35/2;
Width2=0.3/4;
Width3=0.35/6;
Height11=(0.075);
Height2=(0.09);
dh1=(Height2-Height11)*0.5;

Bottom_Block=0.35;
HoriSpace=0.05;

VertSpace=(1-Bottom_Block-5*Height2)*(1/5);
Left_Align1=0.075;
UnitAlign1 = (Left_Align1+Width11+HoriSpace+Width2+0.01);

Bottom_Align2=0.8;
Width21=0.35/2;
Width2=0.3/4;
Width3=0.35/6;
Height21=(0.075);
Height2=(0.09);
```

```
dh2=(Height2-Height21)*0.5;

Bottom_Block=0.35;
HoriSpace=0.05;

VertSpace=(1-Bottom_Block-5*Height2)*(1/5);
Left_Align2=0.525;
UnitAlign2 = (Left_Align2+Width21+HoriSpace+Width2+0.01);

%% Variable Descriptions
    function PlaybackAbout (~,~)
        TH=0.2; %Text Height
        TL1=0.1;   %Text Left
        TW1=0.2; %Text Width - Variable
        TW2=(1-TL1-TW1-TL1); %Text Width - Description
        VS=(1/5)*(1-4*TH); %Vertical Spacing
        TB1=1-VS-TH; %Text Bottom

        AboutBox=figure;
        set(AboutBox,'name','Variable Descriptions','numbertitle','off')
        set(AboutBox,'MenuBar','none');

        PlayDesc='The Playback length determines how long each pure tone will be presented during the assessment';
        PauseDesc='The Pause length determines how the pause will last between tone presentations';
        FadeDesc=' Fade defines the leading and trailing faded-in and faded-out lengths used during the signal presentation. The fade-in and
fade-out durations are specified in unit milliseconds. ';

        Playback_Desc = uicontrol('style','text', 'units', 'normalized',...
            'position', [TL1 TB1 TW1 TH], 'string', 'Playback Length',...
            'visible','on', 'FontSize', Font10,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left','FontWeight','Bold');
        Playback_Desc2 = uicontrol('style','text', 'units', 'normalized',...
            'position', [TL1+TW1 TB1 TW2 TH], 'string', PlayDesc,...
            'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left');

        Pause_Desc = uicontrol('style','text', 'units', 'normalized',...
            'position', [TL1 TB1-(VS+TH) TW1 TH], 'string', 'Pause Length',...
            'visible','on', 'FontSize', Font10,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left','FontWeight','Bold');
        Pause_Desc2 = uicontrol('style','text', 'units', 'normalized',...
            'position', [TL1+TW1 TB1-(VS+TH) TW2 TH], 'string', PauseDesc,...
            'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left');

        Fade_Desc = uicontrol('style','text', 'units', 'normalized',...
            'position', [TL1 TB1-2*(VS+TH) TW1 TH], 'string', 'Fade Length',...
            'visible','on', 'FontSize', Font10,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left','FontWeight','Bold');
        Fade_Desc2 = uicontrol('style','text', 'units', 'normalized',...
            'position', [TL1+TW1 TB1-2*(VS+TH) TW2 TH], 'string', FadeDesc,...
            'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left');

%         Fade_Desc = uicontrol('style','text', 'units', 'normalized',...
%             'position', [TL1 TB1-3*(VS+TH) TW1 TH], 'string', 'Fade Length',...
%             'visible','on', 'FontSize', Font10,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left','FontWeight','Bold');
%         Fade_Desc2 = uicontrol('style','text', 'units', 'normalized',...
%             'position', [TL1+TW1 TB1-3*(VS+TH) TW2 TH], 'string', FadeDesc,...
%             'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left');
%
    end

PlaybackMenu=uicontextmenu;
item1=uimenu(PlaybackMenu,'Label','What''s This?','Callback',@PlaybackAbout);

%% Setting Controls
GeneralFrame1 = uicontrol('style','frame', 'units', 'normalized',...
    'position', [0.05 0.25 0.40 0.70], 'visible','on', 'FontSize', Font10,'BackgroundColor',[0.8 0.8
0.8],'HorizontalAlignment','Left','FontWeight','Bold');

        Frame1_Text = uicontrol('style','text', 'units', 'normalized',...
            'position', [0.075 0.90 Width11 Height11], 'string', 'Signal 1 Settings',...
            'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],...
            'HorizontalAlignment','Left');

        Playback_Text = uicontrol('style','text', 'units', 'normalized',...
            'position', [Left_Align1 Bottom_Align1+dh1 Width11 Height11], 'string', 'Playback Length',...
            'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left');
        Playback_Edit1 = uicontrol('style','edit', 'units', 'normalized',...
            'position', [Left_Align1+Width11+HoriSpace Bottom_Align1 Width2 Height2], 'string', Cur_Playback1,...
            'visible','on', 'FontSize', Font8,'UIContextMenu',PlaybackMenu);
        Playback_Units = uicontrol('style','text', 'units', 'normalized',...
            'position', [UnitAlign1 Bottom_Align1+dh1 Width3 Height11], 'string', 'Sec.',...
            'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left');

        Fade_Text = uicontrol('style','text', 'units', 'normalized',...
            'position', [Left_Align1 Bottom_Align1-1*(VertSpace+Height2)+dh1 Width11 Height11], 'string', 'Fade Length',...
            'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left');
        Fade_Edit1 = uicontrol('style','edit', 'units', 'normalized',...
            'position', [Left_Align1+Width11+HoriSpace Bottom_Align1-1*(VertSpace+Height2) Width2 Height2], 'string', Cur_Fade1,...
            'visible','on', 'FontSize', Font8,'UIContextMenu',PlaybackMenu);
        Fade_Units = uicontrol('style','text', 'units', 'normalized',...
            'position', [UnitAlign1 Bottom_Align1-1*(VertSpace+Height2)+dh1 Width3 Height11], 'string', 'ms.',...
            'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left');

        Pause_Text = uicontrol('style','text', 'units', 'normalized',...
            'position', [Left_Align1 Bottom_Align1-2*(VertSpace+Height2)+dh1 Width11 Height11], 'string', 'Pause Length',...
            'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left');
        Pause_Edit1 = uicontrol('style','edit', 'units', 'normalized',...
            'position', [Left_Align1+Width11+HoriSpace Bottom_Align1-2*(VertSpace+Height2) Width2 Height2], 'string', Cur_Pause1,...
            'visible','on', 'FontSize', Font8,'UIContextMenu',PlaybackMenu);
        Pause_Units = uicontrol('style','text', 'units', 'normalized',...
            'position', [UnitAlign1 Bottom_Align1-2*(VertSpace+Height2)+dh1 Width3 Height11], 'string', 'sec.',...
            'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left');

        Phase_Text = uicontrol('style','text', 'units', 'normalized',...
            'position', [Left_Align1 Bottom_Align1-3*(VertSpace+Height2)+dh1 Width11+0.1 Height11], 'string', 'Binaural Phase Diff.',...
            'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left');
        Phase_Edit1 = uicontrol('style','edit', 'units', 'normalized',...
            'position', [Left_Align1+Width11+HoriSpace Bottom_Align1-3*(VertSpace+Height2) Width2 Height2], 'string', Cur_Phase1,...
            'visible','on', 'FontSize', Font8,'UIContextMenu',PlaybackMenu);
        Phase_Units = uicontrol('style','text', 'units', 'normalized',...
            'position', [UnitAlign1 Bottom_Align1-3*(VertSpace+Height2)+dh1 Width3 Height11], 'string', 'deg.',...
            'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left');

R_Width1=0.15;
DB_Width=0.20;
```

```
GeneralFrame2 = uicontrol('style','frame', 'units', 'normalized',...
    'position', [0.5 0.25 0.40 0.70], 'visible','on', 'FontSize', Font10,'BackgroundColor',[0.8 0.8
0.8],'HorizontalAlignment','Left','FontWeight','Bold');

        Frame2_Text = uicontrol('style','text', 'units', 'normalized',...
            'position', [0.525 0.90 Width21 Height21], 'string', 'Signal 2 Settings',...
            'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left');

        Playback_Text = uicontrol('style','text', 'units', 'normalized',...
            'position', [Left_Align2 Bottom_Align2+dh2 Width21 Height21], 'string', 'Playback Length',...
            'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left');
        Playback_Edit2 = uicontrol('style','edit', 'units', 'normalized',...
            'position', [Left_Align2+Width21+HoriSpace Bottom_Align2 Width2 Height2], 'string', Cur_Playback2,...
            'visible','on', 'FontSize', Font8,'UIContextMenu',PlaybackMenu);
        Playback_Units = uicontrol('style','text', 'units', 'normalized',...
            'position', [UnitAlign2 Bottom_Align2+dh2 Width3 Height21], 'string', 'Sec.',...
            'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left');

        Fade_Text = uicontrol('style','text', 'units', 'normalized',...
            'position', [Left_Align2 Bottom_Align2-1*(VertSpace+Height2)+dh2 Width21 Height21], 'string', 'Fade Length',...
            'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left');
        Fade_Edit2 = uicontrol('style','edit', 'units', 'normalized',...
            'position', [Left_Align2+Width21+HoriSpace Bottom_Align2-1*(VertSpace+Height2) Width2 Height2], 'string', Cur_Fade2,...
            'visible','on', 'FontSize', Font8,'UIContextMenu',PlaybackMenu);
        Fade_Units = uicontrol('style','text', 'units', 'normalized',...
            'position', [UnitAlign2 Bottom_Align2-1*(VertSpace+Height2)+dh2 Width3 Height21], 'string', 'ms.',...
            'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left');

        Pause_Text = uicontrol('style','text', 'units', 'normalized',...
            'position', [Left_Align2 Bottom_Align2-2*(VertSpace+Height2)+dh2 Width21 Height21], 'string', 'Pause Length',...
            'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left');
        Pause_Edit2 = uicontrol('style','edit', 'units', 'normalized',...
            'position', [Left_Align2+Width21+HoriSpace Bottom_Align2-2*(VertSpace+Height2) Width2 Height2], 'string', Cur_Pause2,...
            'visible','on', 'FontSize', Font8,'UIContextMenu',PlaybackMenu);
        Pause_Units = uicontrol('style','text', 'units', 'normalized',...
            'position', [UnitAlign2 Bottom_Align2-2*(VertSpace+Height2)+dh2 Width3 Height21], 'string', 'sec.',...
            'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left');

        Phase_Text = uicontrol('style','text', 'units', 'normalized',...
            'position', [Left_Align2 Bottom_Align2-3*(VertSpace+Height2)+dh2 Width21+0.1 Height21], 'string', 'Binaural Phase Diff.',...
            'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left');
        Phase_Edit2 = uicontrol('style','edit', 'units', 'normalized',...
            'position', [Left_Align2+Width21+HoriSpace Bottom_Align2-3*(VertSpace+Height2) Width2 Height2], 'string', Cur_Phase2,...
            'visible','on', 'FontSize', Font8,'UIContextMenu',PlaybackMenu);
        Phase_Units = uicontrol('style','text', 'units', 'normalized',...
            'position', [UnitAlign2 Bottom_Align2-3*(VertSpace+Height2)+dh2 Width3 Height21], 'string', 'deg.',...
            'visible','on', 'FontSize', Font8,'BackgroundColor',[0.8 0.8 0.8],'HorizontalAlignment','Left');



%% Button Positions
Button_Width =0.3;
Spacing=0.025;
Button_Bottom=0.05;
Button_Height=0.15;
Button_Left = 0.025;

%SAVE SETTINGS
Save_Button=uicontrol('style', 'pushbutton','string', 'Save','units',...
    'normalized','Visible','on','position', [Button_Left Button_Bottom Button_Width Button_Height],...
    'FontSize', Font12, 'FontWeight', 'bold','callback', {@Save_Settings});


    function Save_Settings(~,~)
        CurrentSettings= getappdata(0,'evalue');

        DefaultUserSettings=csvread('UserSettings.txt');

        NewValue_Playback1=str2num(get(Playback_Edit1,'string'));
        NewValue_Pause1=str2num(get(Pause_Edit1,'string'));
        NewValue_Fade1=str2num(get(Fade_Edit1,'string'));
        NewValue_Phase1=str2num(get(Phase_Edit1,'string'));
        NewValue_Playback2=str2num(get(Playback_Edit2,'string'));
        NewValue_Pause2=str2num(get(Pause_Edit2,'string'));
        NewValue_Fade2=str2num(get(Fade_Edit2,'string'));
        NewValue_Phase2=str2num(get(Phase_Edit2,'string'));



        CurrentSettings(1,4)=NewValue_Playback1;
        CurrentSettings(2,4)=NewValue_Pause1;
        CurrentSettings(3,4)=NewValue_Fade1;
        CurrentSettings(4,4)=NewValue_Phase1;
        CurrentSettings(5,4)=NewValue_Playback2;
        CurrentSettings(6,4)=NewValue_Pause2;
        CurrentSettings(7,4)=NewValue_Fade2;
        CurrentSettings(8,4)=NewValue_Phase2;
        setappdata(0,'evalue',CurrentSettings);

        DefaultUserSettings=csvread('UserSettings.txt');
        DefaultUserSettings(1,5)=NewValue_Playback1;
        DefaultUserSettings(2,5)=NewValue_Pause1;
        DefaultUserSettings(3,5)=NewValue_Fade1;
        DefaultUserSettings(4,5)=NewValue_Phase1;
        DefaultUserSettings(5,5)=NewValue_Playback2;
        DefaultUserSettings(6,5)=NewValue_Pause2;
        DefaultUserSettings(7,5)=NewValue_Fade2;
        DefaultUserSettings(8,5)=NewValue_Phase2;

            csvwrite('UserSettings.txt',DefaultUserSettings);




        disp('Settings Saved')
        close
    end

%CANCEL SETTINGS
Cancel_Button=uicontrol('style', 'pushbutton','string', 'Cancel','units',...
    'normalized','Visible','on','position', [Button_Left+Button_Width+Spacing Button_Bottom Button_Width Button_Height],...
    'FontSize', Font12, 'FontWeight', 'bold','callback', {@Cancel_Settings});
```

```
    function Cancel_Settings(~,~)
        disp('Settings Not Saved')
        close
    end

%LIMIT SETTINGS
Limit_Button=uicontrol('style', 'pushbutton','string', 'Limits','units',...
    'normalized','Visible','on','position', [0.675 Button_Bottom Button_Width Button_Height],...
    'FontSize', Font12, 'FontWeight', 'bold','callback', {@Exp_Limit_Settings});

    function Exp_Limit_Settings(~,~)
        Limit_Settings
    end

end
```

A.2.6 Program Limit Settings (Limit_Settings.m)

```
function LimitSettings ()
%% CLEARING DATA AND COLLECTING SCREEN SIZE
clc
set(0,'units','pixels');
RelFont = get(0,'ScreenSize'); %#ok<NASGU>
Font6=round(6/1080*RelFont(1,4)); %Font 8 on 1920x1080
Font8=round(8/1080*RelFont(1,4)); %Font 8 on 1920x1080
Font10=round(10/1080*RelFont(1,4));
Font12=round(12/1080*RelFont(1,4));
Font16=round(16/1080*RelFont(1,4));

set(0,'Units','normalized');
scnsize = get(0,'ScreenSize'); %#ok<NASGU>
Background_Color=[0.8 0.8 0.8];
%% COLLECT CURRENT DATA FROM APP_DATA MATRIX
StoredAppData= getappdata(0,'evalue');
    AmbientL = StoredAppData(:,11);
    AmbientR = StoredAppData(:,12);
    Freq_Limits = StoredAppData(:,13);
    ReferenceContour = StoredAppData(:,14);

% evalue Placeholders
% Column 1:
% Column 2:
% Column 3:
% Column 4: User Settings
% Column 5: Resultant L-Small (:,3)
% Column 6: Resultant R-Small (:,5)
% Column 7: Headphones 1 - Left
% Column 8: Headphones 1 - Right
% Column 9: Headphones 2 - Left
% Column 10: Headphones 2 - Right
% Column 11: Ambient Conditions Left
% Column 12: Ambient Conditions Right
% Column 13: Maximum Limits
%% FIGURE DIMENSIONS
%set(MainWindow,'Visible','off')
Width=0.3;
CenterFig=0.5*(1-Width);
Settings_Box= figure('units', 'normalized','Position',[CenterFig 0.2 Width 0.64],...
    'Color',Background_Color);
set(Settings_Box,'name','Hardware Safety Limits','numbertitle','off')
set(Settings_Box,'MenuBar','none');

a=get(Settings_Box,'Position');
%% AMBIENT TABLE FORMATTING
AmbFrame1 = uicontrol('style','frame', 'units', 'normalized',...
    'position', [0.0125 0.07 0.35 0.92], 'visible','on',...
    'FontSize', Font10,'BackgroundColor',[0.8 0.8 0.8],...
    'HorizontalAlignment','Left','FontWeight','Bold');

Amb_Frame_Text = uicontrol('style','text', 'units', 'normalized',...
    'position', [0.03 0.9675 0.2 0.025],...
    'visible','on', 'FontSize', Font10,'BackgroundColor',[0.8 0.8 0.8],...
    'HorizontalAlignment','Left', 'string', 'Ambient Levels');


cnames = {'Left','Right'};
rnames = {'20','25','31.5','40','50','63','80','100','125','160','200',...
    '250','315','400','500','630','800','1000','1250','1600','2000',...
    '2500','3150','4000','5000','6300','8000','10000','12500','16000','20000'};
    Ambient_Combine(:,1)=AmbientL(:,1);
    Ambient_Combine(:,2)=AmbientR(:,1);

Amb_Table = uitable('Data',Ambient_Combine,...
    'ColumnName',cnames, 'RowName', rnames,'units', 'normalized',...
    'Position',[0.035 0.12 0.31 0.84]);

set(Amb_Table,'units', 'normalized','ColumnWidth',{40})
set(Amb_Table,'ColumnEditable',true);
%% PASTE AMBIENT LEFT - PUSHBUTTON
PasteButton1=uicontrol('style', 'pushbutton','units',...
    'normalized','Visible','on','position', [0.18 0.085 0.075 0.03],...
    'FontSize', Font6, 'FontWeight', 'bold','callback', {@copyExcel_callback1,Ambient_Combine},...
    'units', 'normalized','string','Paste');

    function copyExcel_callback1(~,~,Ambient_Combine)
        StoredAppData= getappdata(0,'evalue');
            AmbientL = StoredAppData(:,11);
            AmbientR = StoredAppData(:,12);
            Freq_Limits = StoredAppData(:,13);
        str1 = clipboard('paste');
        str1 = strrep(str1,',','.');
            import1(:,1)=AmbientL;
            import1(:,2)=AmbientR;
        import1(:,1)=str2num(str1);
        set(Amb_Table, 'Data', import1);

            StoredAppData(:,11)= import1(:,1);
            StoredAppData(:,12)=import1(:,2);
            StoredAppData(:,13)=Freq_Limits;
```

```
            setappdata(0,'evalue',StoredAppData);
    end
%% PASTE AMBIENT RIGHT - PUSHBUTTON
PasteButton2=uicontrol('style', 'pushbutton','units',...
    'normalized','Visible','on','position', [0.268 0.085 0.075 0.03],...
    'FontSize', Font6, 'FontWeight', 'bold','callback', {@copyExcel_callback2,Ambient_Combine},...
    'units', 'normalized','string','Paste');

    function copyExcel_callback2(~,~,Ambient_Combine)
        StoredAppData= getappdata(0,'evalue');
            AmbientL = StoredAppData(:,11)
            AmbientR = StoredAppData(:,12)
            Freq_Limits = StoredAppData(:,13);

            str2 = clipboard('paste');
            str2 = strrep(str2,',','.');
            import2(:,1)=AmbientL;
            import2(:,2)=AmbientR;
            import2(:,2)=str2num(str2);
            set(Amb_Table, 'Data', import2);

            StoredAppData(:,11)=import2(:,1)
            StoredAppData(:,12)=import2(:,2)
            StoredAppData(:,13)=Freq_Limits;

            setappdata(0,'evalue',StoredAppData);
    end
%% LOAD AMBIENT - PUSHBUTTON
LoadAmbButton=uicontrol('style', 'pushbutton','units',...
    'normalized','Visible','on','position', [0.055 0.085 0.1 0.03],...
    'FontSize', Font8, 'FontWeight', 'bold','callback', {@LoadAmb},...
    'units', 'normalized','string','Load');

    function LoadAmb(~,~)
            StoredAppData= getappdata(0,'evalue');
            AmbientL = StoredAppData(:,11);
            AmbientR = StoredAppData(:,12);
            Freq_Limits = StoredAppData(:,13);
            ReferenceContour = StoredAppData(:,14);


        [SPLFileName,PathName] = uigetfile('*.txt','Select TXT file');
        if SPLFileName==0, return, end

        AMB = csvread( fullfile(PathName,SPLFileName) );   %# pass file path as string


            StoredAppData(:,11)=AMB(:,2);
            StoredAppData(:,12)=AMB(:,3);
            set(Amb_Table,'Data',[AMB(:,2),AMB(:,3)]);

            setappdata(0,'evalue',StoredAppData);
    end
%% MAX SPL TABLE FORMATTING
AmbFrame2 = uicontrol('style','frame', 'units', 'normalized',...
    'position', [0.40 0.07 0.275 0.92], 'visible','on',...
    'FontSize', Font10,'BackgroundColor',[0.8 0.8 0.8],...
    'HorizontalAlignment','Left','FontWeight','Bold');

MAX_SPL_Frame_Text = uicontrol('style','text', 'units', 'normalized',...
    'position', [0.42 0.9675 0.25 0.025],...
    'visible','on', 'FontSize', Font10,'BackgroundColor',[0.8 0.8 0.8],...
    'HorizontalAlignment','Left', 'string', 'Max. SPL Allowed');


cnames2 = {'Max.'};
rnames2 = {'20','25','31.5','40','50','63','80','100','125','160','200',...
    '250','315','400','500','630','800','1000','1250','1600','2000',...
    '2500','3150','4000','5000','6300','8000','10000','12500','16000','20000'};
    MAX_SPL(:,1)=Freq_Limits(:,1);

MAX_SPL_Table = uitable('Data',MAX_SPL,...
    'ColumnName',cnames2, 'RowName', rnames2,'units', 'normalized',...
    'Position',[0.425 0.12 0.225 0.84]);

set(MAX_SPL_Table,'units', 'normalized','ColumnWidth',{40})
set(MAX_SPL_Table,'ColumnEditable',true);
%% LOAD SPL MAX - PUSHBUTTON
LoadMAXButton=uicontrol('style', 'pushbutton','units',...
    'normalized','Visible','on','position', [0.45 0.085 0.1 0.03],...
    'FontSize', Font8, 'FontWeight', 'bold','callback', {@LoadMaxSPL},...
    'units', 'normalized','string','Load');

    function LoadMaxSPL(~,~)
            StoredAppData= getappdata(0,'evalue');
            AmbientL = StoredAppData(:,11);
            AmbientR = StoredAppData(:,12);
            Freq_Limits = StoredAppData(:,13);
            ReferenceContour = StoredAppData(:,14);


        [SPLFileName,PathName] = uigetfile('*.txt','Select TXT file');
        if SPLFileName==0, return, end

        SPL_MAX = csvread( fullfile(PathName,SPLFileName) );   %# pass file path as string


            StoredAppData(:,13)=SPL_MAX(:,2);
            set(MAX_SPL_Table,'Data',SPL_MAX(:,2));

            setappdata(0,'evalue',StoredAppData);
    end
%% PASTE MAX SPL - PUSHBUTTON
PasteButton3=uicontrol('style', 'pushbutton','units',...
    'normalized','Visible','on','position', [0.575 0.085 0.075 0.03],...
    'FontSize', Font6, 'FontWeight', 'bold','callback', {@copyExcel_callback3,Freq_Limits},...
    'units', 'normalized','string','Paste');

    function copyExcel_callback3(~,~,Freq_Limits)
        StoredAppData= getappdata(0,'evalue');
            AmbientL = StoredAppData(:,11);
            AmbientR = StoredAppData(:,12);
            Freq_Limits = StoredAppData(:,13);
```

```
            str3 = clipboard('paste');
            str3 = strrep(str3,',','.');
            import3(:,1)=str2num(str3);
            set(MAX_SPL_Table, 'Data', import3);

            StoredAppData(:,13)=import3(:,1)

            setappdata(0,'evalue',StoredAppData);
    end
%% Reference TABLE FORMATTING
ReferenceFrame = uicontrol('style','frame', 'units', 'normalized',...
    'position', [0.7 0.07 0.275 0.92], 'visible','on',...
    'FontSize', Font10,'BackgroundColor',[0.8 0.8 0.8],...
    'HorizontalAlignment','Left','FontWeight','Bold');

Reference_Frame_Text = uicontrol('style','text', 'units', 'normalized',...
    'position', [0.7125 0.9675 0.125 0.025],...
    'visible','on', 'FontSize', Font10,'BackgroundColor',[0.8 0.8 0.8],...
    'HorizontalAlignment','Left', 'string', 'Reference');

cnames3 = {'Ref.'};
rnames3 = {'20','25','31.5','40','50','63','80','100','125','160','200',...
    '250','315','400','500','630','800','1000','1250','1600','2000',...
    '2500','3150','4000','5000','6300','8000','10000','12500','16000','20000'};
    REF_SPL(:,1)=ReferenceContour(:,1);

Reference_Table = uitable('Data',REF_SPL,...
    'ColumnName',cnames3, 'RowName', rnames3,'units', 'normalized',...
    'Position',[0.725 0.12 0.225 0.84]);

set(Reference_Table,'units', 'normalized','ColumnWidth',{40})
set(Reference_Table,'ColumnEditable',true);
%% LOAD REFERENCE - PUSHBUTTON
LoadRefButton=uicontrol('style', 'pushbutton','units',...
    'normalized','Visible','on','position', [0.75 0.085 0.1 0.03],...
    'FontSize', Font8, 'FontWeight', 'bold','callback', {@LoadRef},...
    'units', 'normalized','string','Load');

    function LoadRef(~,~)
            StoredAppData= getappdata(0,'evalue');
            AmbientL = StoredAppData(:,11);
            AmbientR = StoredAppData(:,12);
            Freq_Limits = StoredAppData(:,13);
            ReferenceContour = StoredAppData(:,14);

        [SPLFileName,PathName] = uigetfile('*.txt','Select TXT file');
        if SPLFileName==0, return, end

        REF = csvread( fullfile(PathName,SPLFileName) );   %# pass file path as string

            StoredAppData(:,14)=REF(:,2);
            set(Reference_Table,'Data',REF(:,2));

            setappdata(0,'evalue',StoredAppData);
    end
%% PASTE REFERECE - PUSHBUTTON
PasteButton4=uicontrol('style', 'pushbutton','units',...
    'normalized','Visible','on','position', [0.875 0.085 0.075 0.03],...
    'FontSize', Font6, 'FontWeight', 'bold','callback', {@PasteRef},...
    'units', 'normalized','string','Paste');

    function PasteRef(~,~)
        StoredAppData= getappdata(0,'evalue');
            AmbientL = StoredAppData(:,11);
            AmbientR = StoredAppData(:,12);
            Freq_Limits = StoredAppData(:,13);
            ReferenceContour = StoredAppData(:,14);

            str4 = clipboard('paste');
            str4 = strrep(str4,',','.');
            import4(:,1)=str2num(str4);
            set(Reference_Table, 'Data', import4);

            StoredAppData(:,14)=import4(:,1);

            setappdata(0,'evalue',StoredAppData);
    end
%% SAVE CHANGES - PUSHBUTTON
SaveButton=uicontrol('style', 'pushbutton','units',...
    'normalized','Visible','on','position', [0.1 0.01 0.8 0.05],...
    'FontSize', Font8, 'FontWeight', 'bold','callback', {@SaveChanges},...
    'units', 'normalized','string','Save Changes as Default');

    function SaveChanges(~,~)

        StoredAppData= getappdata(0,'evalue');

        Summary = csvread('UserSettings.txt');
        ReferenceContour=get(Reference_Table,'Data');
        Ambient =  get(Amb_Table,'Data');
        MaxData = get(MAX_SPL_Table,'Data');

        for GetData = 1:31
            AmbientL(GetData,1) = Ambient(GetData,1);
            AmbientR(GetData,1) = Ambient(GetData,2);
            Freq_Limits(GetData,1) =  MaxData(GetData,1);

            StoredAppData(GetData,11)=AmbientL(GetData,1);
            StoredAppData(GetData,12)=AmbientR(GetData,1);
            StoredAppData(GetData,13)=Freq_Limits(GetData,1);
            StoredAppData(GetData,14)=ReferenceContour(GetData,1);

            setappdata(0,'evalue',StoredAppData);

            Summary(GetData,1) = AmbientL(GetData,1);
            Summary(GetData,2) = AmbientR(GetData,1);
            Summary(GetData,3) = Freq_Limits(GetData,1);
            Summary(GetData,4) = ReferenceContour(GetData,1);
        end

            UserSettings =Summary;
            csvwrite('UserSettings.txt',UserSettings);
    end
end
```

A.2.7 Audiogram Zoom Option (ZoomAudiogramGUI.m)

```
function ZoomAudiogramGUI(~,~,ExpApp)

    %%FIGURE DIMENSIONS
    set(0,'units','pixels');
    RelFont = get(0,'ScreenSize'); %#ok<NASGU>
    set(0,'Units','normalized');
    scnsize = get(0,'ScreenSize'); %#ok<NASGU>

    MainWindow3 = figure('units', 'normalized',...
        'Position',[0.3,0.25,0.35,0.5], 'Visible', 'On',...
        'Name','Audiogram','NumberTitle','Off');

    set(MainWindow3,'MenuBar','none');
    set(MainWindow3,'Toolbar','figure');

    h1Toolbar = findall(MainWindow3,'tag','FigureToolBar');
    delete(findall(h1Toolbar,'tag','Standard.NewFigure'))
    delete(findall(h1Toolbar,'tag','Standard.FileOpen'))
    delete(findall(h1Toolbar,'tag','Plottools.PlottoolsOn'))
    delete(findall(h1Toolbar,'tag','Plottools.PlottoolsOff'))
    delete(findall(h1Toolbar,'tag','Annotation.InsertColorbar'))
    delete(findall(h1Toolbar,'tag','DataManager.Linking'))
    delete(findall(h1Toolbar,'tag','Standard.EditPlot'))
    delete(findall(h1Toolbar,'tag','Exploration.Rotate'))
    delete(findall(h1Toolbar,'tag','Exploration.Brushing'))
    delete(findall(h1Toolbar,'tag','Annotation.InsertLegend'))

    Font6=round(6/1080*RelFont(1,4));
    Font8=round(8/1080*RelFont(1,4)); %Font 8 on 1920x1080
    Font10=round(10/1080*RelFont(1,4));
    Font12=round(12/1080*RelFont(1,4));
    Font16=round(16/1080*RelFont(1,4));

    Plot_Frame2 = uipanel('Title','Audiogram', 'units', 'normalized',...
    'position', [0.05 0.15 0.7 0.7], 'visible','on',...
    'FontSize', Font10,'BackgroundColor',[0.8 0.8 0.8],...
    'Visible','On');

% Inserting Data to Plot
Freq=[20, 25, 31.5, 40, 50, 63, 80, 100, 125, 160, 200, 250, 315, 400, 500, 630, 800, 1000, 1250, 1600, 2000, 2500, 3150, 4000, 5000, 6300,
8000, 10000, 12500, 16000, 20000];
FreqOct=[31.5, 63, 125, 250, 500, 1000, 2000, 4000, 8000, 16000];

    ReadDefaultSettings=csvread('UserSettings.txt');

for loadref = 1:31
    ReadDefaultSettings(loadref,1);
    ReferenceContour(loadref,1) = (Freq(1,loadref)); ReferenceContour(loadref,2) = (ReadDefaultSettings(loadref,4));
    MAX_SPL_Lim(loadref,1) = (Freq(1,loadref)); MAX_SPL_Lim(loadref,2) = (ReadDefaultSettings(loadref,3)); % Load Maximum SPL Limit for Each
Frequency
    AmbientSPL(loadref,1) = (Freq(1,loadref)); AmbientSPL(loadref,2) = (ReadDefaultSettings(loadref,1)); AmbientSPL(loadref,3) =
(ReadDefaultSettings(loadref,2)); % Load Reference
end
        ExpApp= getappdata(0,'evalue');
        for GetData = 1:31
            Resultant(GetData,1) = Freq(1,GetData); % Column 5: Resultant L-Small (:,3)
            Resultant(GetData,3) = ExpApp(GetData,5); % Column 5: Resultant L-Small (:,3)
            Resultant(GetData,5) = ExpApp(GetData,6); % Column 6: Resultant R-Small (:,5)
        end

    Picture1=subplot(1,1,1,'Parent',Plot_Frame2);
        PlotAxes = semilogx(ReferenceContour(:,1), ReferenceContour(:,2), '--k',... % (1) ANSI Reference Contour
                            AmbientSPL(:,1), AmbientSPL(:,2),':b',...               % (2) Ambient - Left Signal
                            AmbientSPL(:,1), AmbientSPL(:,3),':r',...               % (3) Ambient - Right Signal
                            Resultant(:,1), Resultant(:,2),'ob',...                 % (4) Large Left
                            Resultant(:,1), Resultant(:,3),'ob',...                 % (5) Small Left
                            Resultant(:,1), Resultant(:,4),'xr',...                 % (6) Large Right
                            Resultant(:,1), Resultant(:,5),'xr',...                 % (7) Small Right
                            MAX_SPL_Lim(:,1),MAX_SPL_Lim(:,2),'-r',...              % (10) LIMIT Line
            'DisplayName', 'SemiLog Plot'); %, 'Parent', handles.graph

        set(PlotAxes(4),'MarkerSize',1,'LineWidth', 0.5) % Large Left (Hide)
        set(PlotAxes(6),'MarkerSize',1,'LineWidth', 0.5) % Large Right (Hide)
        set(PlotAxes(7),'MarkerSize',5,'LineWidth', 2)   % Magnifying Small Right

        % Filling Marker Colours for points of interest
        set(PlotAxes(5),'MarkerSize',5,'MarkerFaceColor','b')
        set(PlotAxes(7),'MarkerSize',5,'MarkerFaceColor','r')

        ax_PlotAxes = gca; %Collect info on current axes values to gca
        curtick = get(gca, 'XTick'); % Collecting current x-labels
        set(ax_PlotAxes, 'units', 'normalized',...
            'OuterPosition', [0 0.2 0.8 0.75],...
            'Position', [0.1 0.1 0.85 0.855],...
            'XLim', [10 20000],...
            'YLim', [-10 100],...
            'XTickLabel', cellstr(num2str(curtick(:)))); % Setting the x-labels to non-scientific
        grid

        textcount = 1;
        LS = 0.8;
        LS2 = 0.9;

            FreqText = uicontrol('style', 'text','string','Freq.',...
            'units','normalized',...
            'Visible','on','position', [LS 0.89 0.05 0.1],...
            'FontSize', Font8,...
            'BackgroundColor',[0.8 0.8 0.8],...
            'ForegroundColor',[0 0 1],'FontWeight','bold');

            SPLDiff = uicontrol('style', 'text','string','SPL. Diff',...
            'units','normalized',...
            'Visible','on','position', [LS2-0.025 0.89 0.1 0.1],...
            'FontSize', Font8,...
            'BackgroundColor',[0.8 0.8 0.8],...
            'ForegroundColor',[0 0 1],'FontWeight','bold');

        for text =1:31
            FreqText = uicontrol('style', 'text','string',Resultant(text,1),...
```

```
            'units','normalized',...
            'Visible','on','position', [LS 0.89-0.03*textcount 0.05 0.1],...
            'FontSize', Font8,...
            'BackgroundColor',[0.8 0.8 0.8],...
            'ForegroundColor',[0 0 1],'FontWeight','bold');

            SPLDiff = uicontrol('style', 'text','string',(Resultant(text,3)-Resultant(text,5)),...
            'units','normalized',...
            'Visible','on','position', [LS2 0.89-0.03*textcount 0.05 0.1],...
            'FontSize', Font8,...
            'BackgroundColor',[0.8 0.8 0.8],...
            'ForegroundColor',[0 0 1],'FontWeight','bold');

            if abs(Resultant(text,3)-Resultant(text,5)) > 5
                set(SPLDiff,'ForegroundColor',[1 0 0])
            end

            textcount=textcount+1;
        end

end
```

APPENDIX B: Loudness Model Check and Verification Signals

List of Appendix B Figures

Appendix B.1: Verification Signals

The following pages demonstrate the 30 acoustical signals used during the verification trials. Signals were selected in order to test the binaural summation mechanism for both stationary and non-stationary noise signatures. The selection included broad-spectrum noise signatures, complex tones, and various non-stationary recordings summarized as follows:

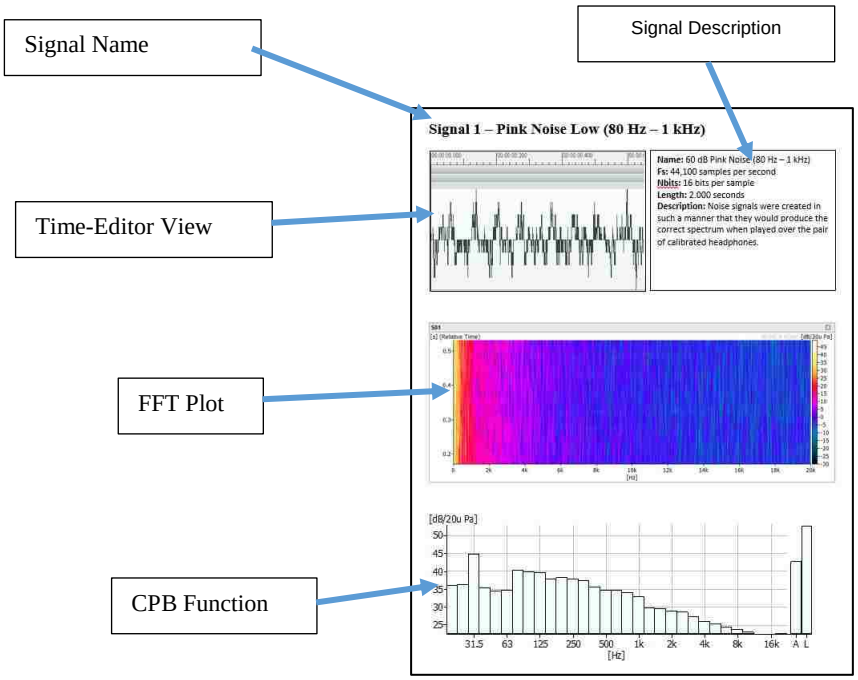
1) Pink Noise Low (80 Hz – 1 kHz)
2) Pink Noise Wide (80 Hz – 16 kHz)
3) Pink Noise High (1 kHz – 16 kHz)
4) White Noise Low (80 Hz – 1 kHz)
5) White Noise Wide (80 Hz – 16 kHz)
6) White Noise High (1 kHz – 16 kHz)
7) Low 1/1-Octaves (63 Hz – 1 kHz)
8) 1/1-Octaves (63 Hz – 8 kHz)
9) High 1/1-Octaves (1 kHz – 8 kHz)
10) Complex Tones 1 (63 Hz + 250 Hz)
11) Complex Tones 2 (500 Hz + 2 kHz)
12) Complex Tones 3 (250 Hz + 4 kHz)
13) Complex Tones 4 (250 Hz + 1 kHz + 4 kHz)
14) Complex Tones 5 (4 kHz + 16 kHz)
15) Drill Noise (Constant)
16) Drill Noise (Pulsed)
17) Bagpipe Recording [1]
18) Didgeridoo Recording [1]
19) Movie Introduction Music [2]
20) Vehicle Cabin Noise (Steady Speed)
21) Electronic Siren
22) Flute Music Sample [1]
23) Rumbler Siren Noise
24) Clarinet Music Sampler [1]
25) Tuba Musical Sample [1]
26) Violin Musical Sample [1]
27) Speech Segment (“Blade of Grass”) [1]
28) Phone Notification 1 [3]
29) Phone Notification 2 [3]
30) Phone Notification 3 [3]

For the following comparisons three plots were included to provide the most relevant information: temporal time-editor view, an FFT versus time plot, and the overall CPB function.

The **time-editor** enabled each signal to be viewed with the input time data as a function of overall pressure per millisecond.

In order to accurately demonstrate the spectral content of each signal, an **FFT plot** was included with a frequency span of 20 kHz and 3,200 FFT Lines. This provided an FFT resolution of 6.25 Hz and an averaging time of 53.33 ms. The averaging function was applied exponentially with a 66.7% overlapped Hanning window. To ensure adequate temporal resolution, the FFT data was sampled at 10 ms intervals.

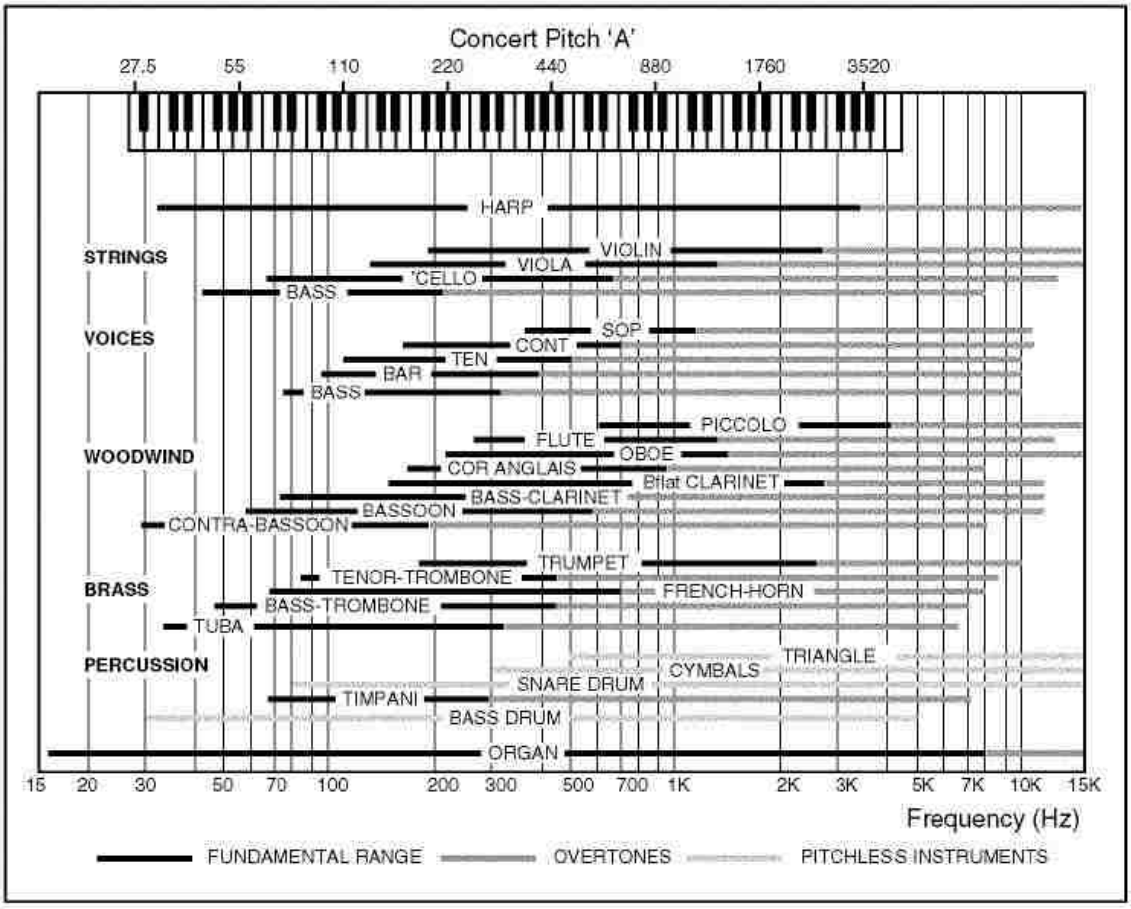
The standardized **CPB Function** had a 1/3-Octave resolution from 20 Hz to 20 kHz with equal emphasis over the entire signal (as oppose to Exponential where the emphasis was placed on the most recent data).



VIII.

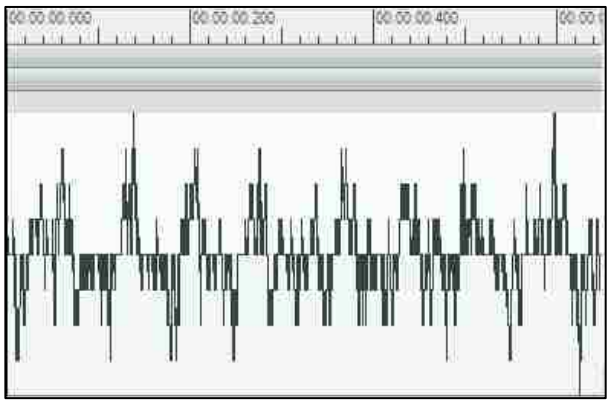
Figure B.1 - Verification Signal Layout

Several instrumentals were selected for the verification test in order to include a time-varying signal within each spectral region of interest. For reference, **Figure B.2** below demonstrates the fundamental range for a variety of concert instruments with pitch characteristics.

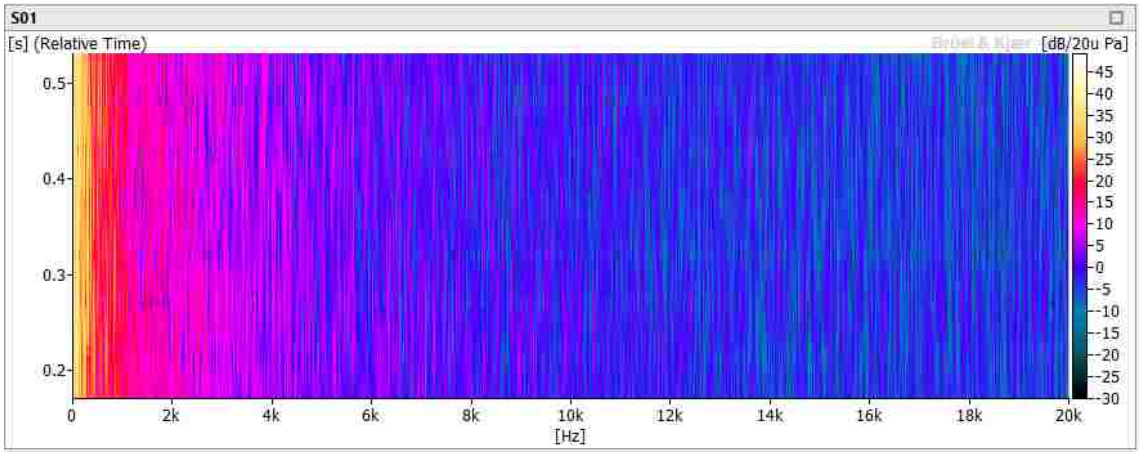


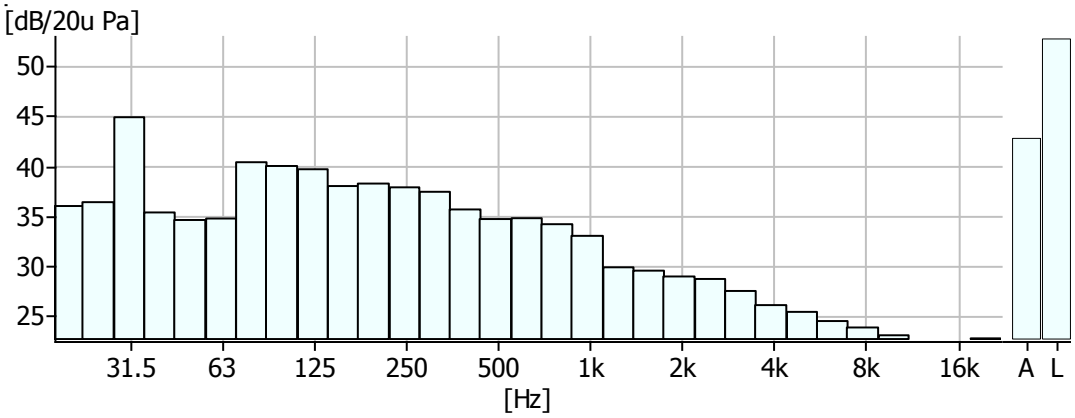
IX. Figure B.2 – Demonstration of the spectral range for common instrumental pitch

Signal 1 – Pink Noise Low (80 Hz – 1 kHz)

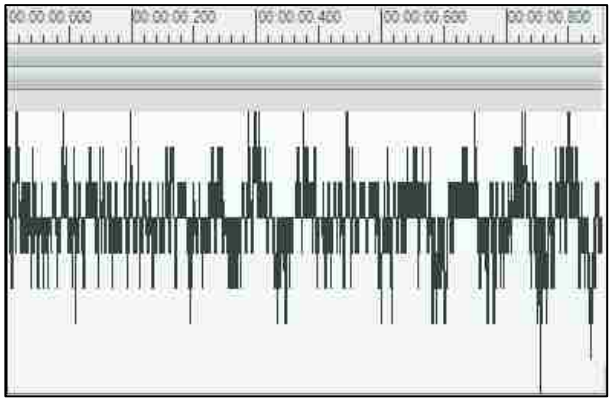


Name: 60 dB Pink Noise (80 Hz – 1 kHz)
Fs: 44,100 samples per second
Nbits: 16 bits per sample
Length: 2.000 seconds
Description: Noise signals were created in such a manner that they would produce the correct spectrum when played over the pair of calibrated headphones.

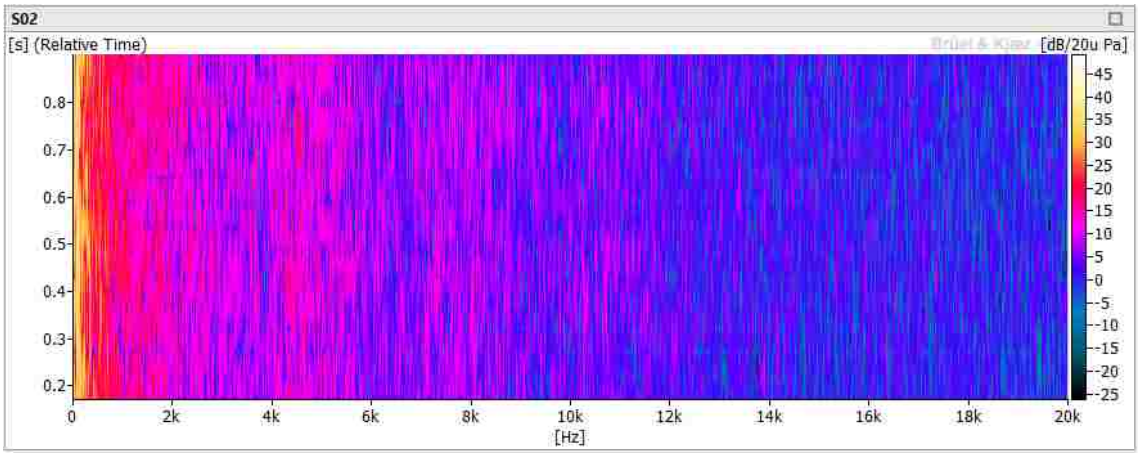


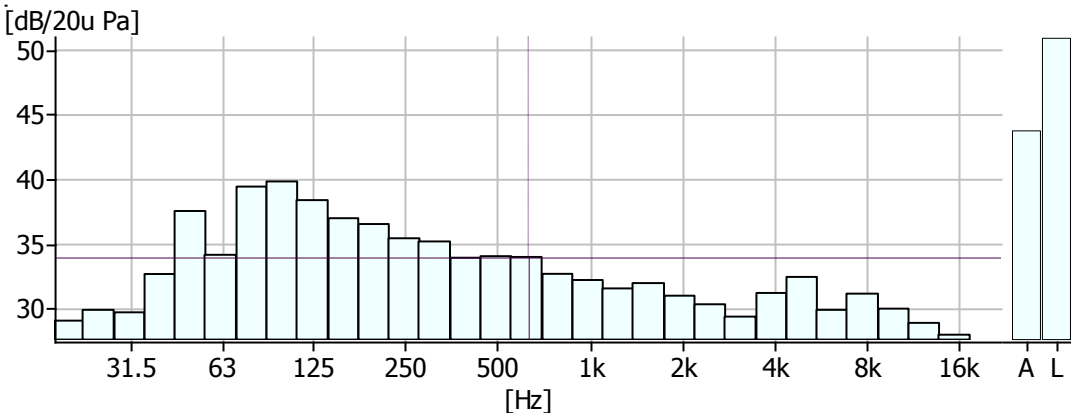


Signal 2 – Pink Noise Wide (80 Hz – 16 kHz)



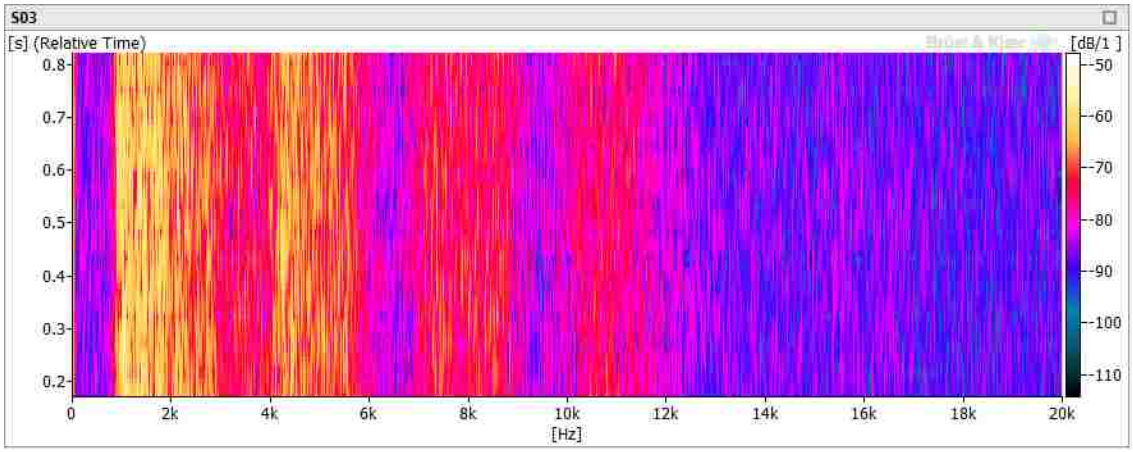
Name: 60 dB Pink Noise (80 Hz – 16 kHz)
Fs: 44,100 samples per second
Nbits: 16 bits per sample
Length: 2.000 seconds
Description: Noise signals were created in such a manner that they would produce the correct spectrum when played over the pair of calibrated headphones.

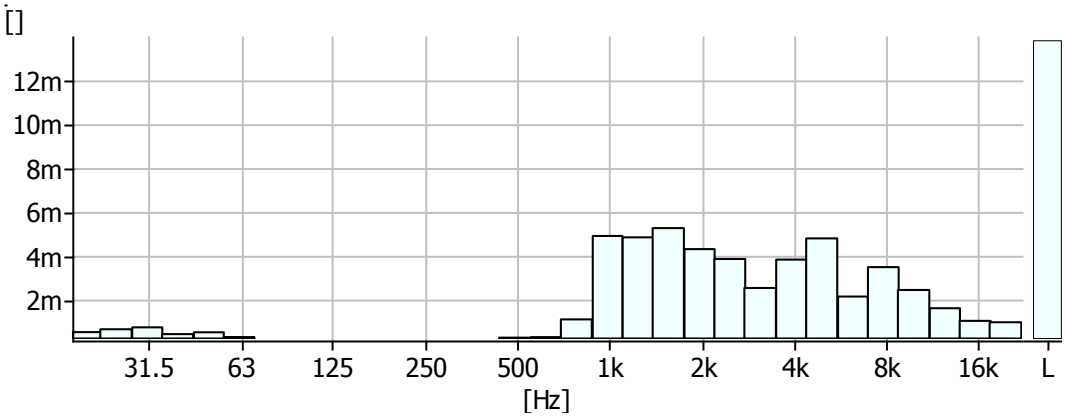




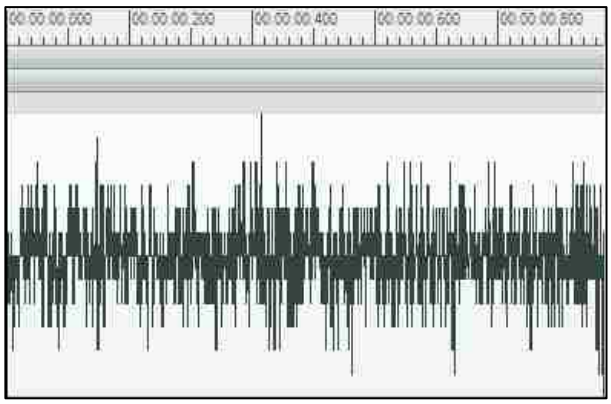
Signal 3 – Pink Noise High (1 kHz – 16 kHz)

Name: 60 dB Pink Noise (1 kHz – 16 kHz)
Fs: 44,100 samples per second
Nbits: 16 bits per sample
Length: 2.000 seconds
Description: Noise signals were created in such a manner that they would produce the correct spectrum when played over the pair of calibrated headphones.

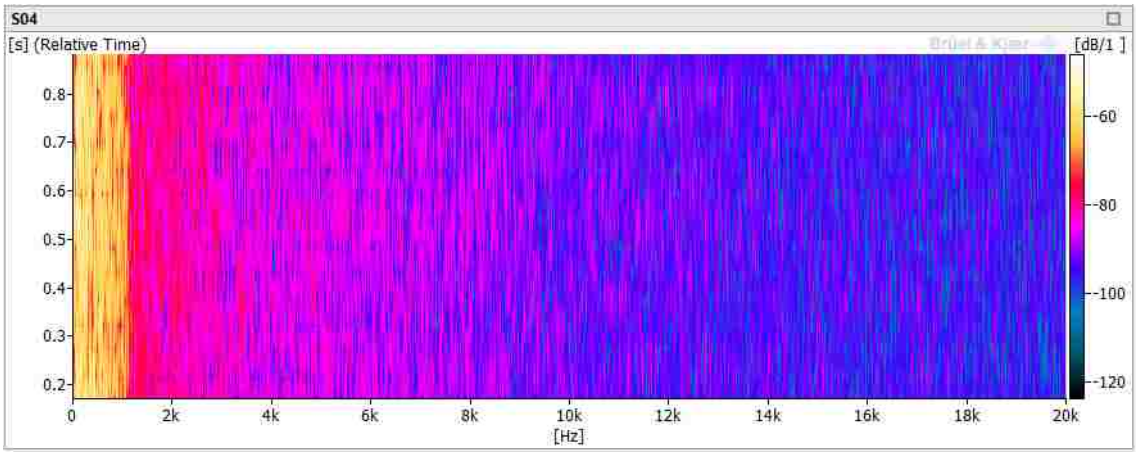


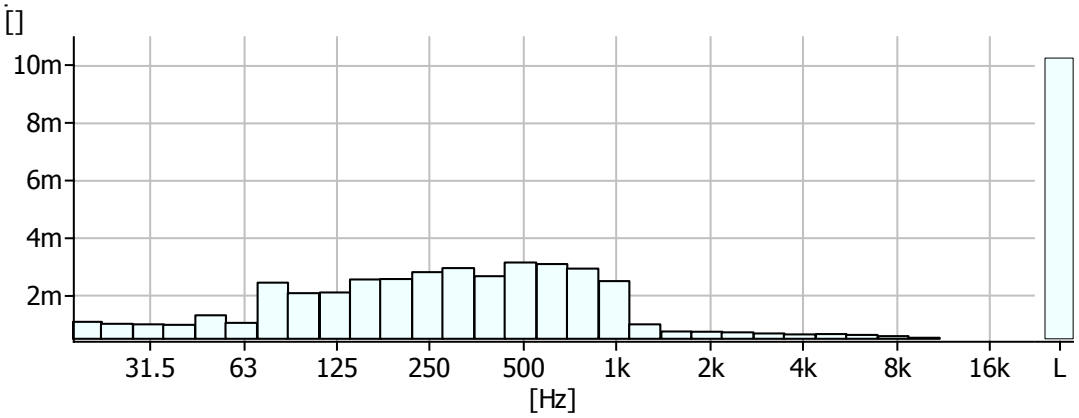


Signal 4 – White Noise Low (80 Hz – 1 kHz)

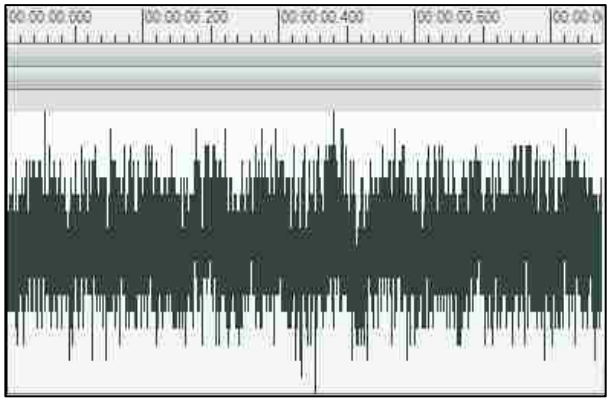


Name: 60 dB White Noise (80 Hz – 1 kHz)
Fs: 44,100 samples per second
Nbits: 16 bits per sample
Length: 2.000 seconds
Description: Noise signals were created in such a manner that they would produce the correct spectrum when played over the pair of calibrated headphones.

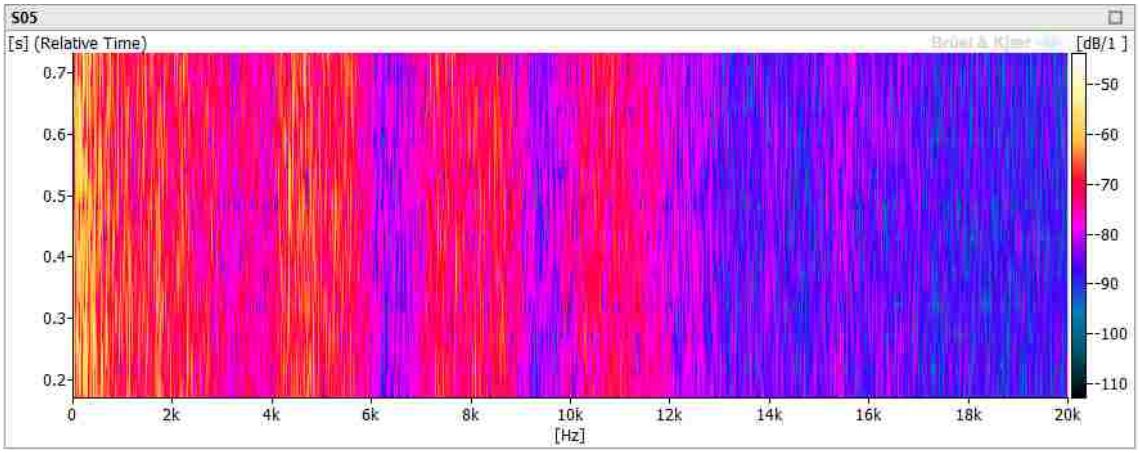


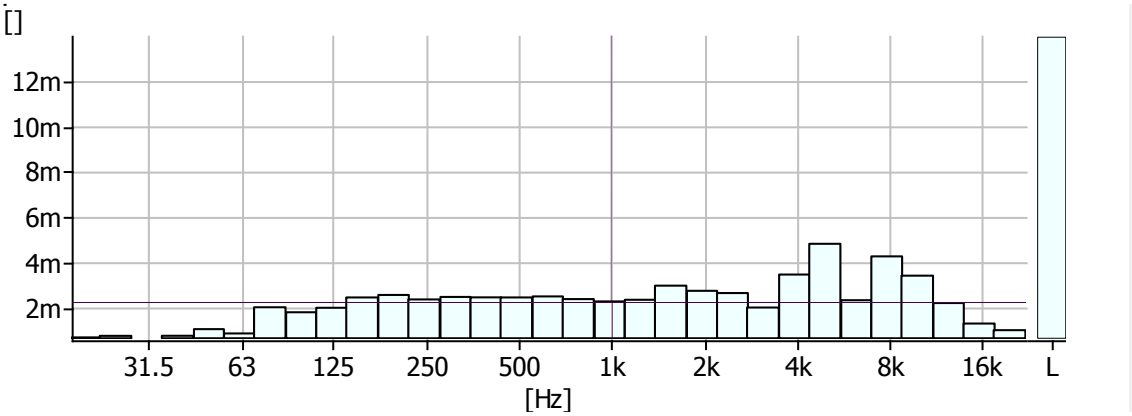


Signal 5 – White Noise Wide (80 Hz – 16 kHz)

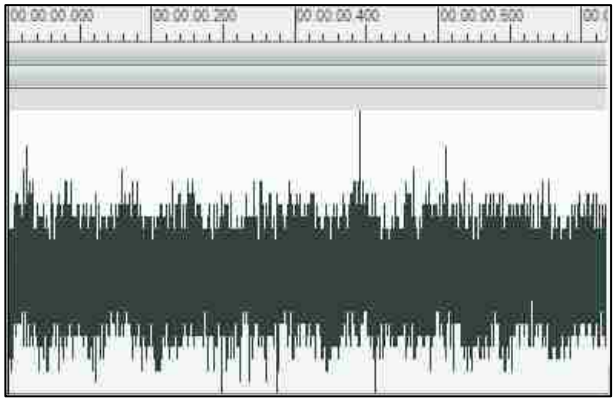


Name: 60 dB White Noise (80 Hz – 16 kHz)
Fs: 44,100 samples per second
Nbits: 16 bits per sample
Length: 2.000 seconds
Description: Noise signals were created in such a manner that they would produce the correct spectrum when played over the pair of calibrated headphones.

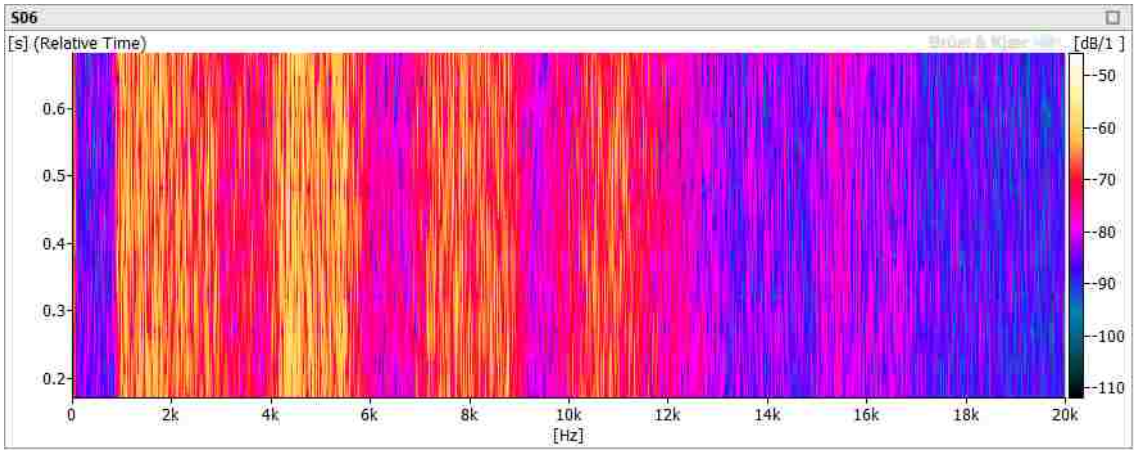


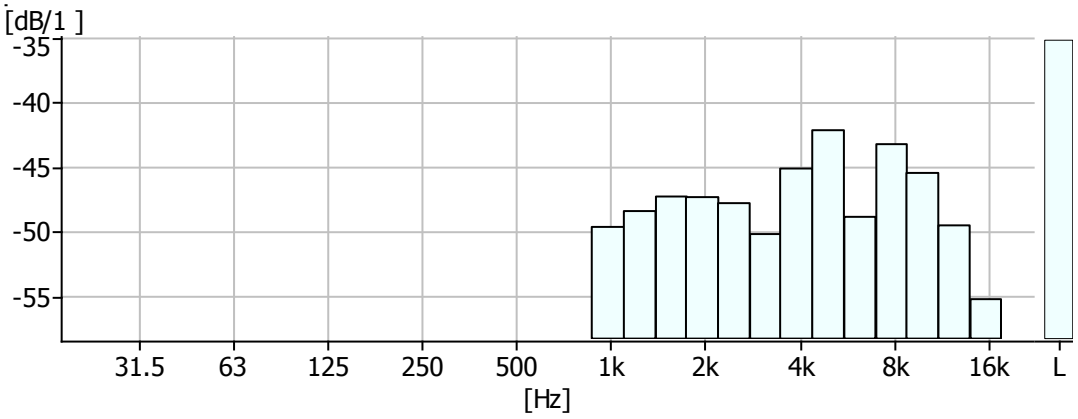


Signal 6 – White Noise High (1 kHz – 16 kHz)

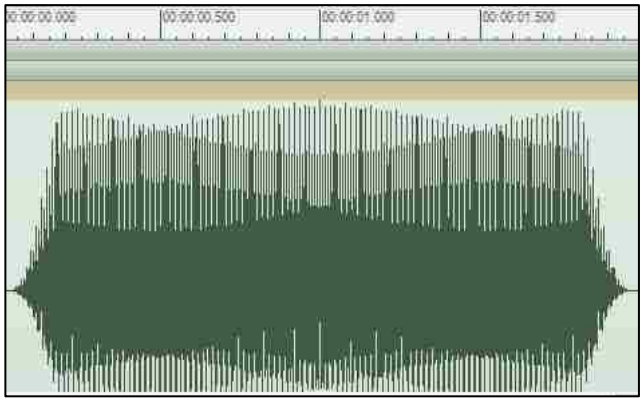


Name: 60 dB White Noise (1 kHz – 16 kHz)
Fs: 44,100 samples per second
Nbits: 16 bits per sample
Length: 2.000 seconds
Description: Noise signals were created in such a manner that they would produce the correct spectrum when played over the pair of calibrated headphones.

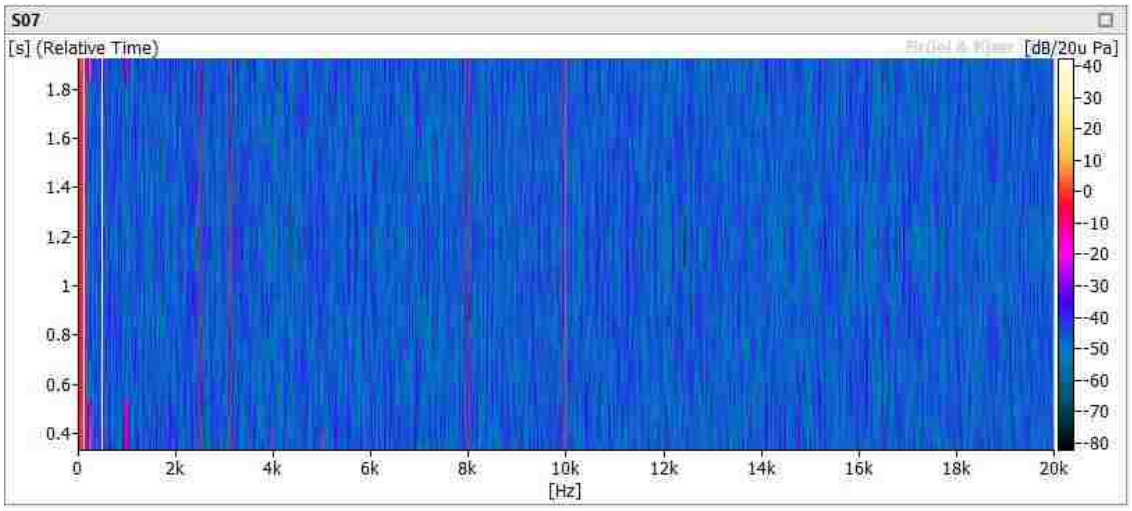


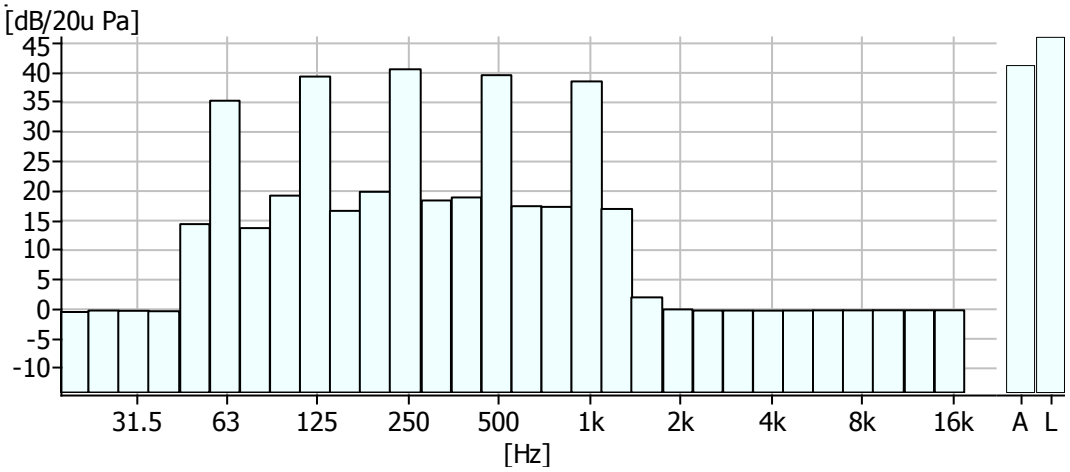


Signal 7 – Octaves Low (63 Hz – 1 kHz)

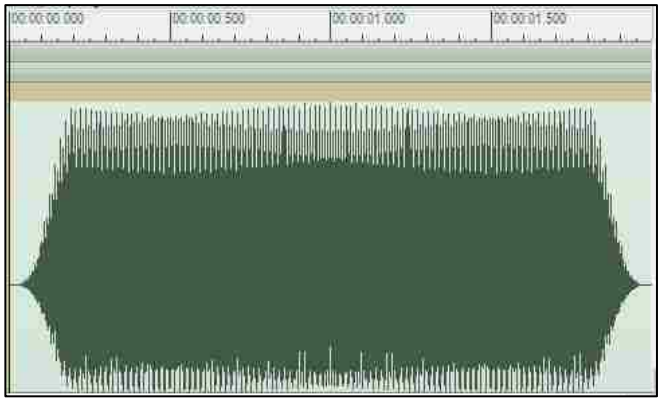


Name: Octaves Low
Fs: 65,536 samples per second
Nbits: 16 bits per sample
Length: 2.000 seconds
Description: Pure tones produced at each 1/1-octave including 63 Hz, 125 Hz, 250 Hz, 500 Hz, and 1 kHz. Tones were recorded in such a manner that they would produce the same amplitude when played over a pair of calibrated

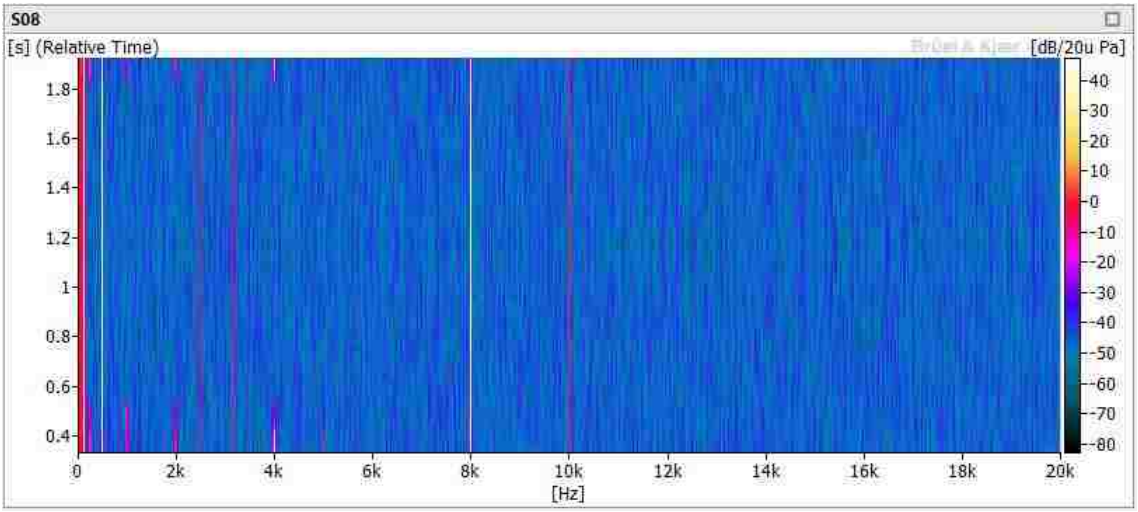


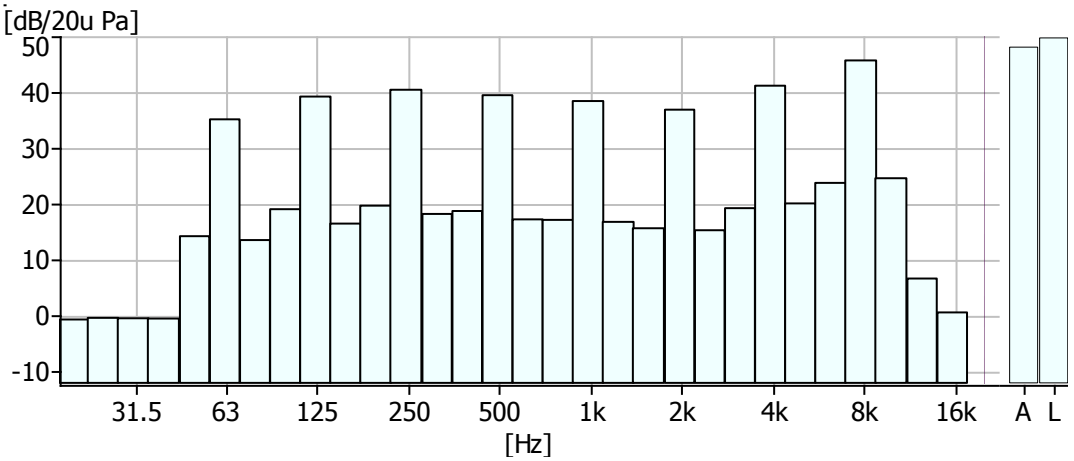


Signal 8 – Octaves (63 Hz – 8 kHz)

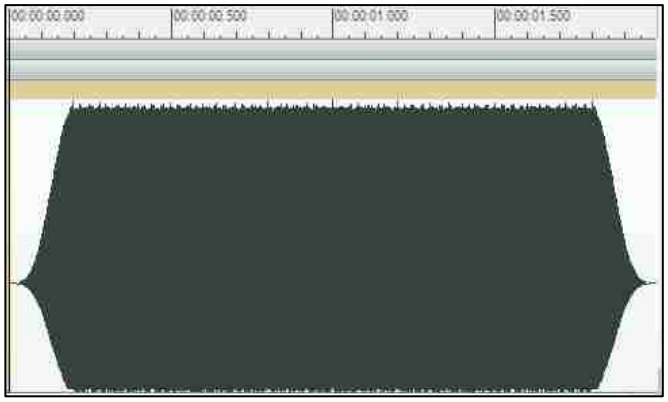


Name: Octaves
Fs: 65,536 samples per second
Nbits: 16 bits per sample
Length: 2.000 seconds
Description: Pure tones produced at each 1/1-octave including 63 Hz, 125 Hz, 250 Hz, 500 Hz, 1 kHz, 2 kHz, 4 kHz, and 8 kHz. Recorded in such a manner that they would produce the same amplitude when played over a pair

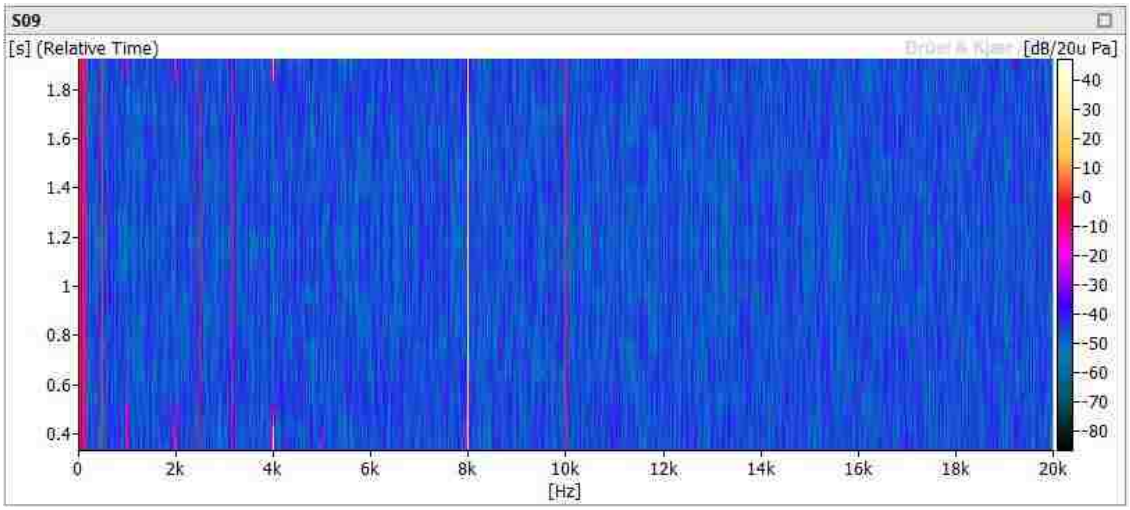


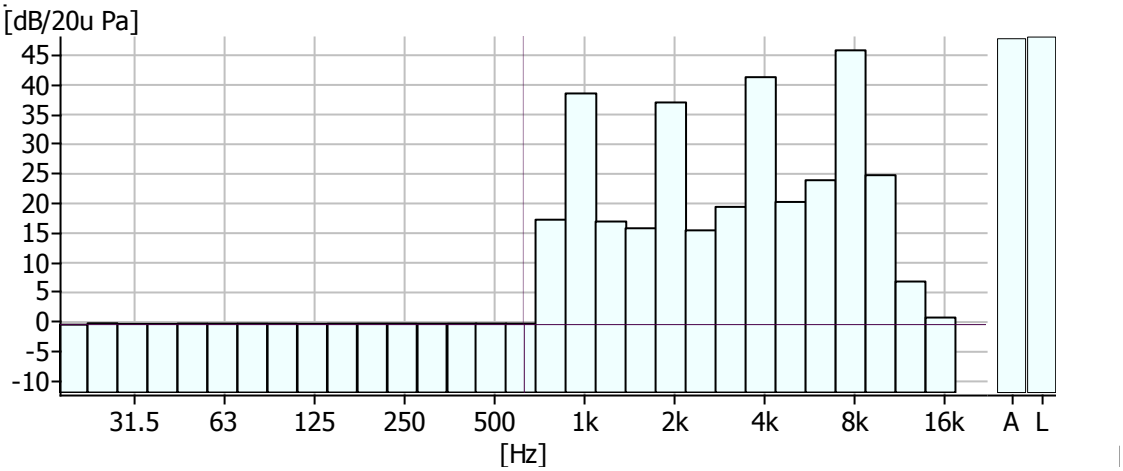


Signal 9 – Octaves High (1 kHz – 8 kHz)



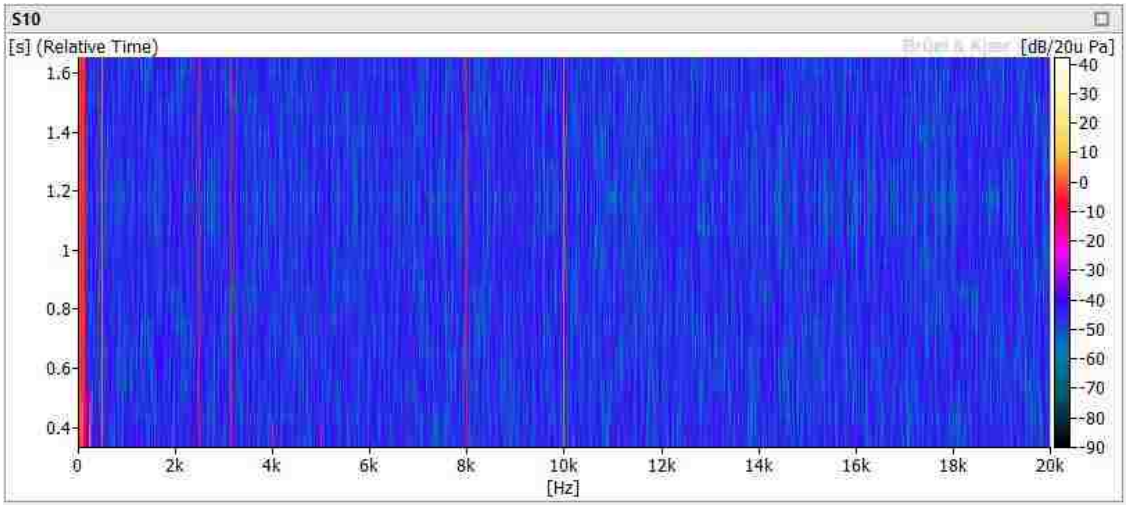
Name: Octaves High
Fs: 65,536 samples per second
Nbits: 16 bits per sample
Length: 2.000 seconds
Description: Pure tones produced at each 1/1-octave including 1 kHz, 2 kHz, 4 kHz, and 8 kHz. Tones were recorded in such a manner that they would produce the same amplitude when played over a pair of calibrated

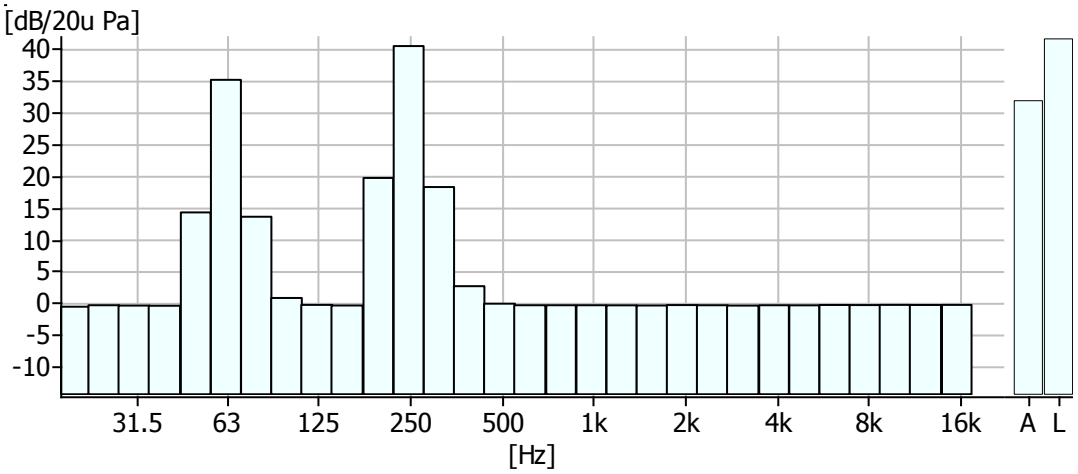




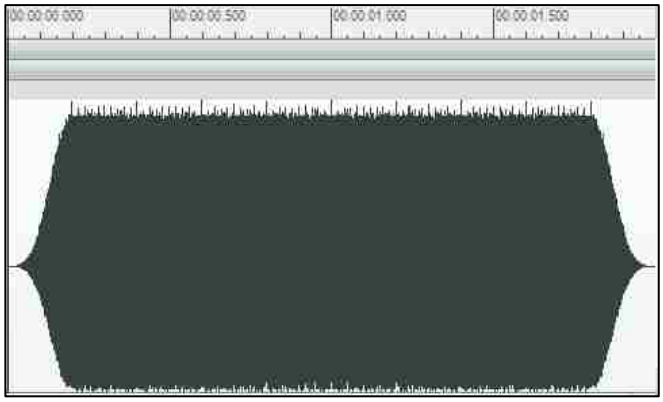
Signal 10 – Complex Tones 1 (63 Hz + 250 Hz)

Name: Complex 1
Fs: 65,536 samples per second
Nbits: 16 bits per sample
Length: 2.000 seconds
Description: Complex pair of pure tones produced at 63 Hz, and 250 Hz targeting low frequencies. Tones were recorded in such a manner that they would produce the same amplitude when played over a pair of calibrated

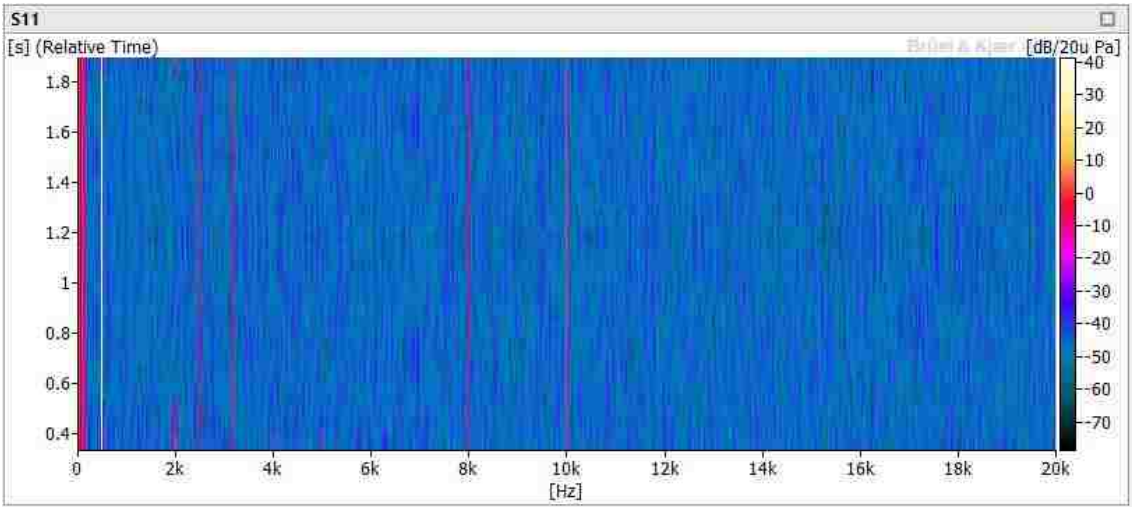


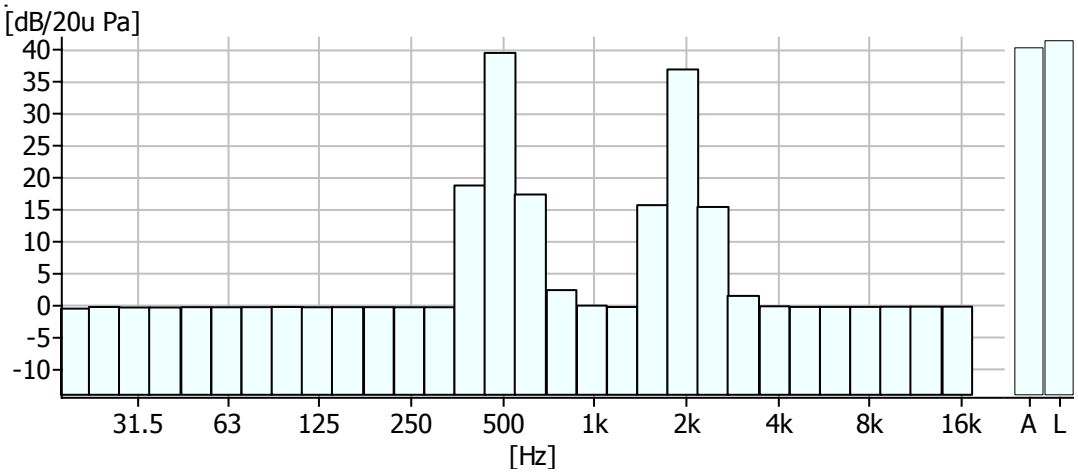


Signal 11 – Complex Tones 2 (500 Hz + 2 kHz)

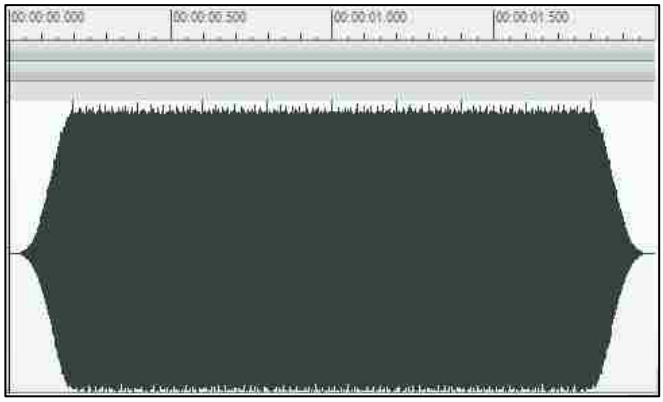


Name: Complex 2
Fs: 65,536 samples per second
Nbits: 16 bits per sample
Length: 2.000 seconds
Description: Complex pair of pure tones produced at 500 Hz, and 2 kHz targeting mid frequencies of moderate separation. Recorded in such a manner that they would produce the same amplitude when played

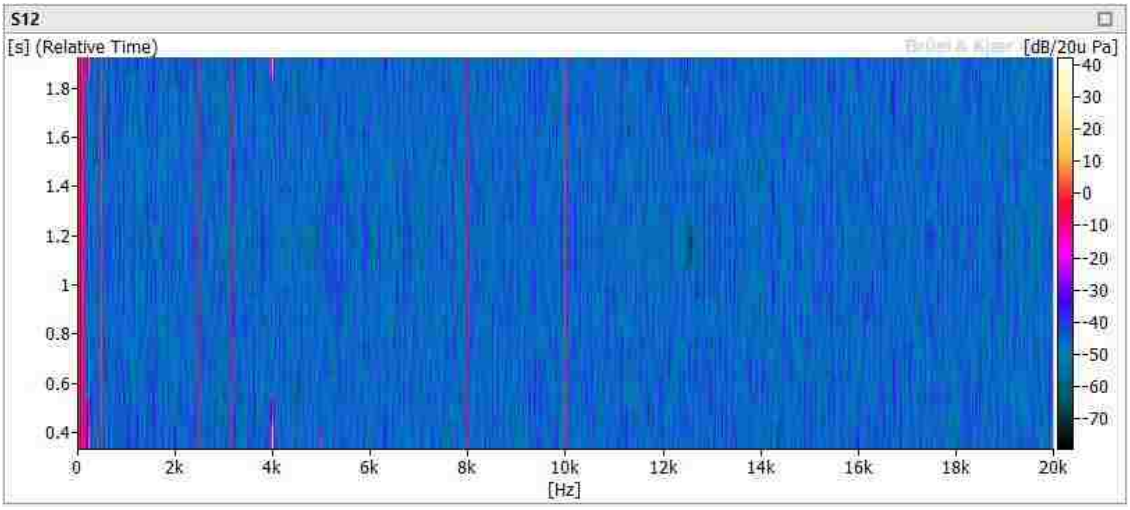


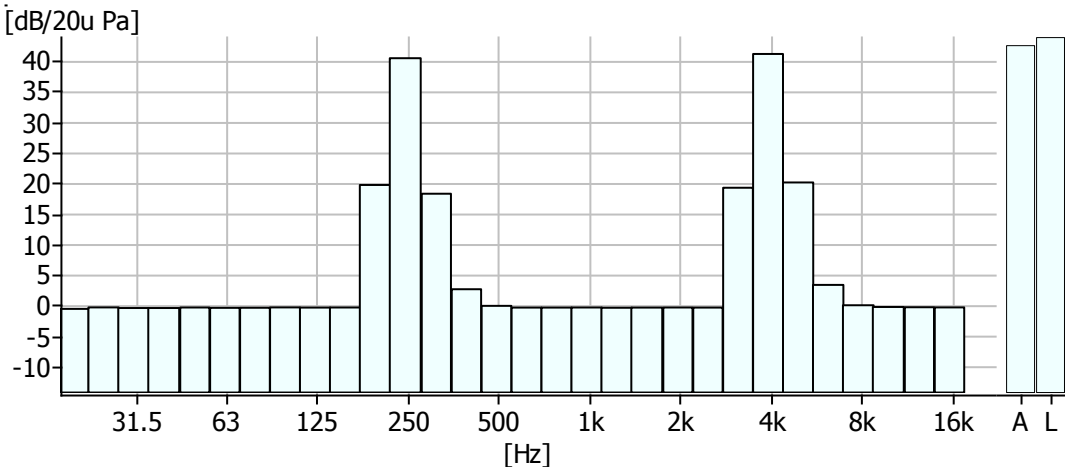


Signal 12 – Complex Tones 3 (250 Hz + 4 kHz)

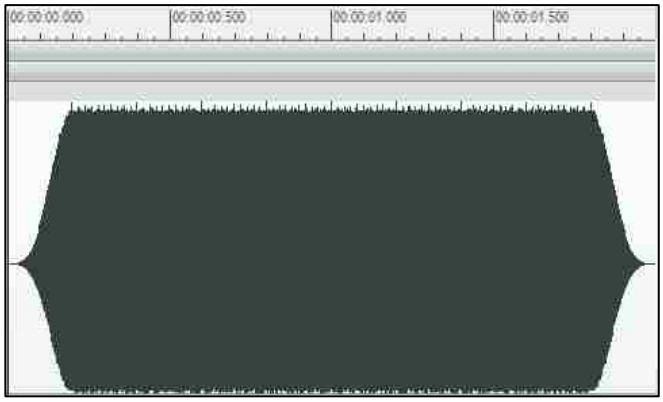


Name: Complex 3
Fs: 65,536 samples per second
Nbits: 16 bits per sample
Length: 2.000 seconds
Description: Complex pair of pure tones produced at 250 Hz, and 4 kHz targeting mid frequencies of wide separation. Recorded in such a manner that they would produce the same amplitude when played over a pair

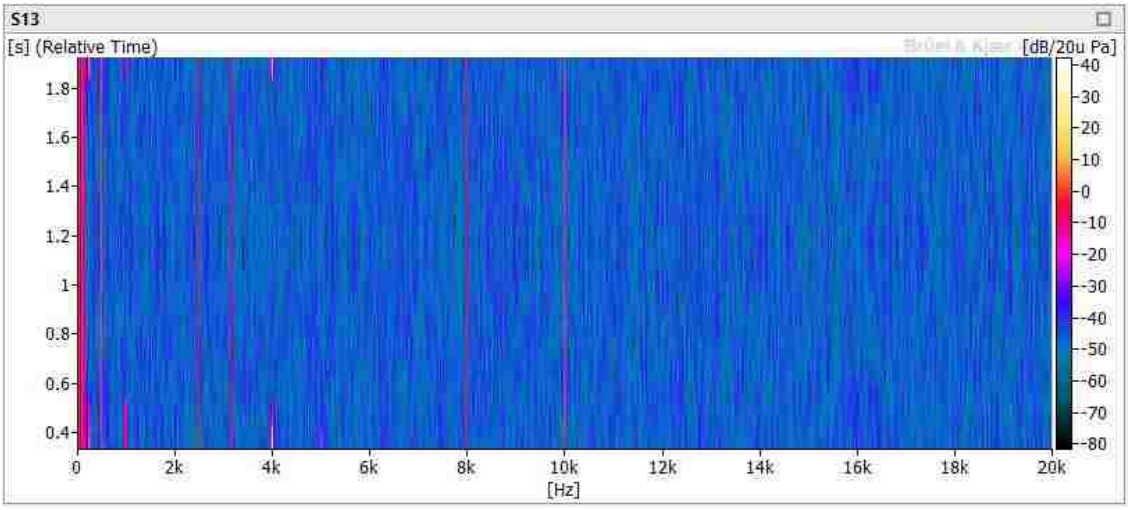


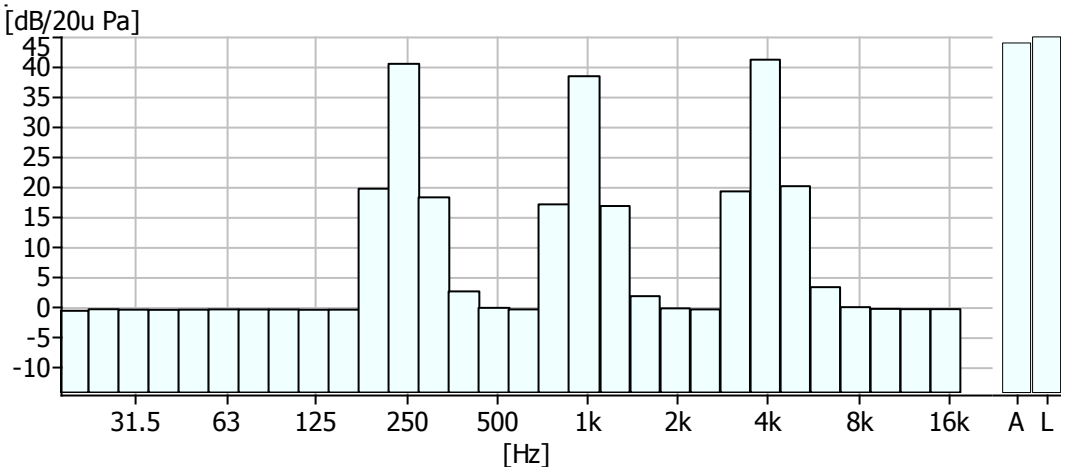


Signal 13 – Complex Tones 4 (250 Hz + 1 kHz + 4 kHz)

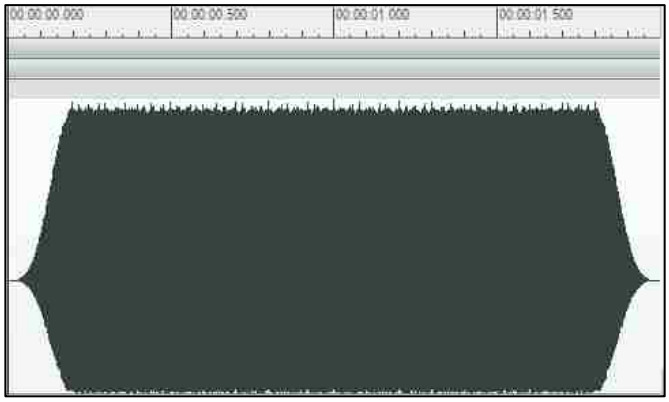


Name: Complex 4
Fs: 65,536 samples per second
Nbits: 16 bits per sample
Length: 2.000 seconds
Description: Complex triplet of pure tones produced at 250 Hz, 1kHz and 4 kHz. Tones were recorded in such a manner that they would produce the same amplitude when played over a pair of calibrated headphones.

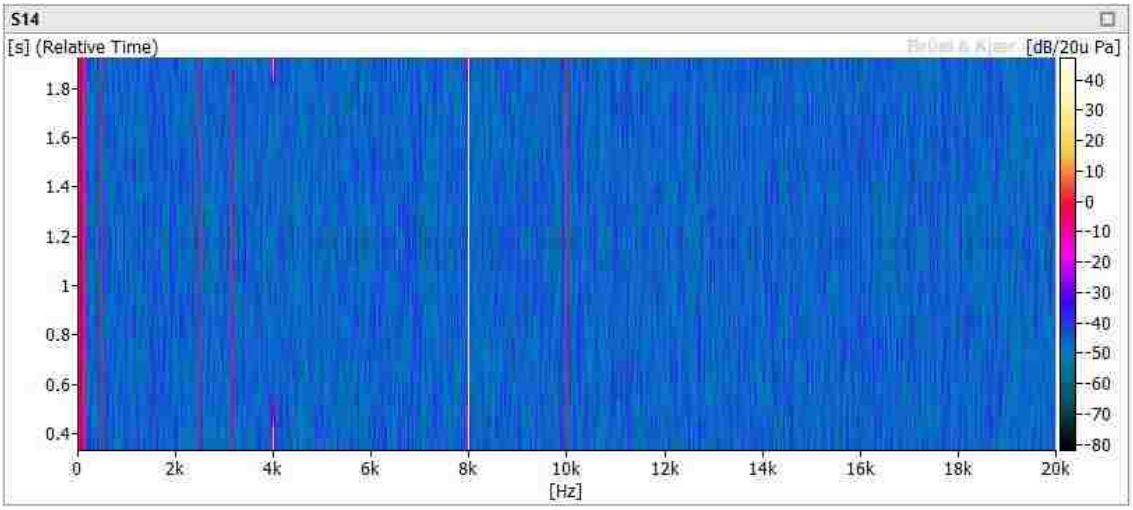


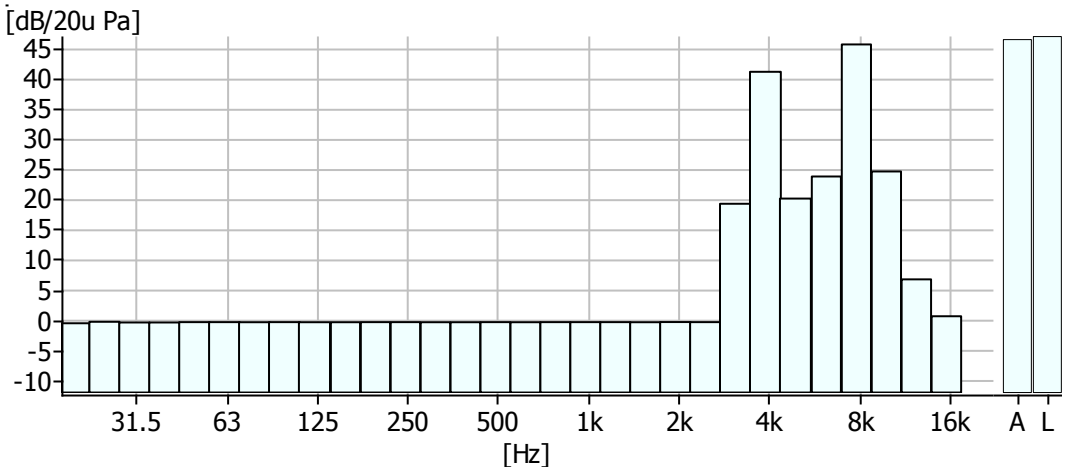


Signal 14 – Complex Tones 5 (4 kHz + 16 kHz)



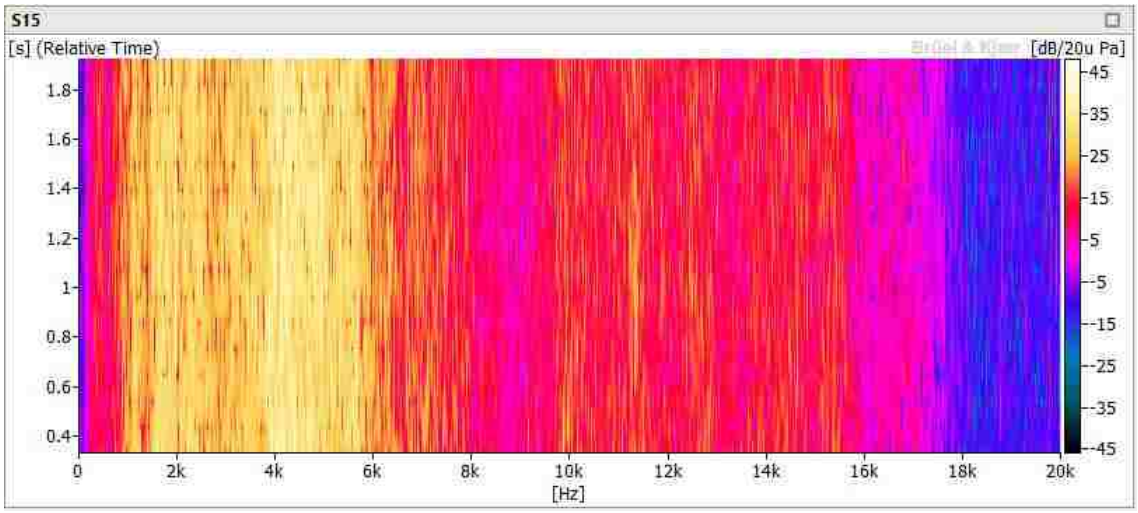
Name: Complex 5
Fs: 65,536 samples per second
Nbits: 16 bits per sample
Length: 2.000 seconds
Description: Complex pair of pure tones produced at 4 kHz, and 16 kHz targeting high frequencies. Tones were recorded in such a manner that they would produce the same amplitude when played over a pair of calibrated

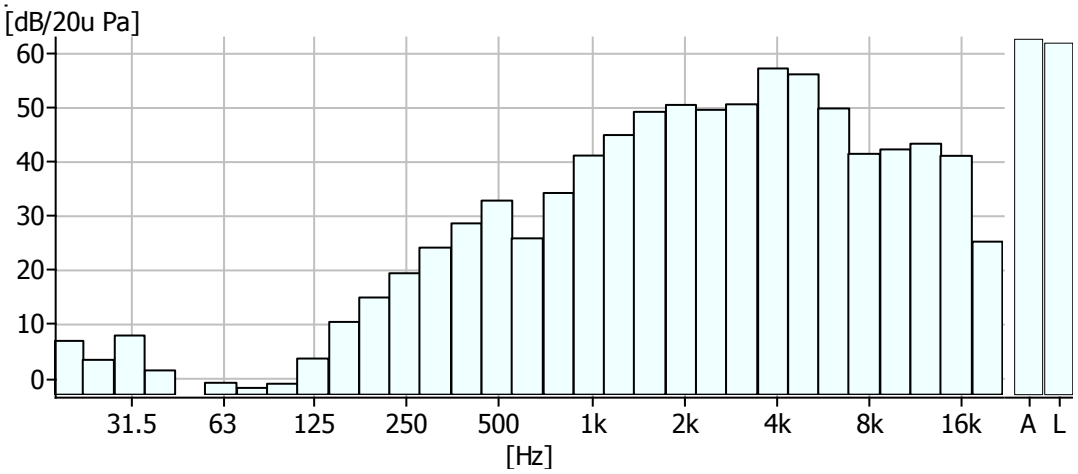




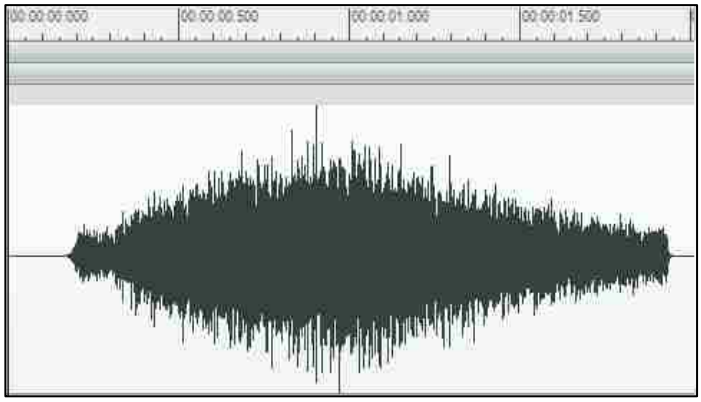
Signal 15 – Drill Noise (Constant)

Name: Drill (Constant Speed)
Fs: 65,536 samples per second
Nbits: 16 bits per sample
Length: 2.002 seconds
Description: Recorded noise signature of an electric drill, recorded in a semi-anechoic environment, 2 metres from the head and torso

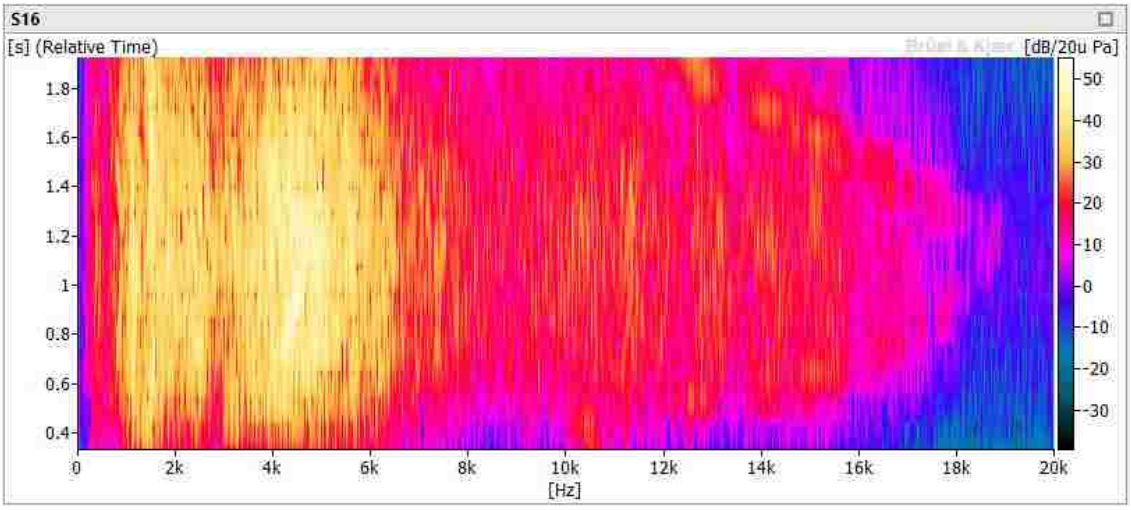


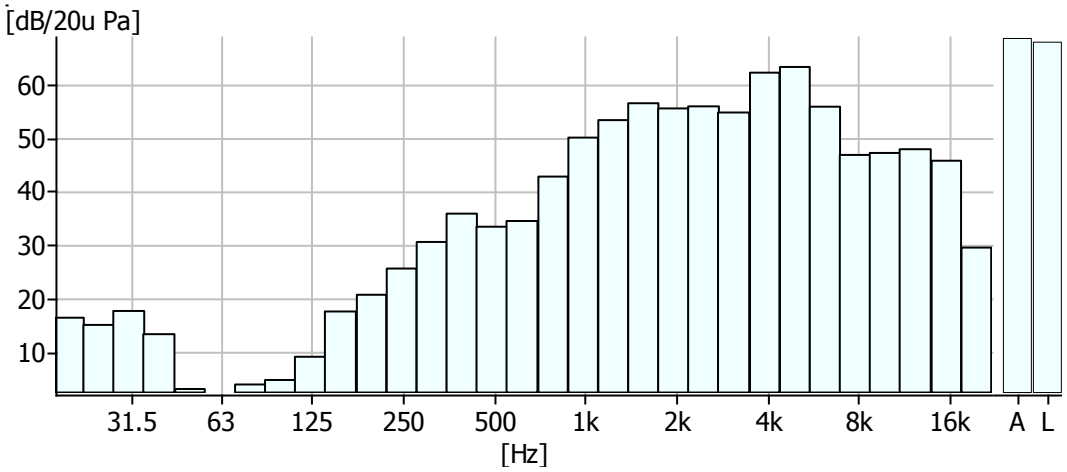


Signal 16 – Drill Noise (Pulsed)

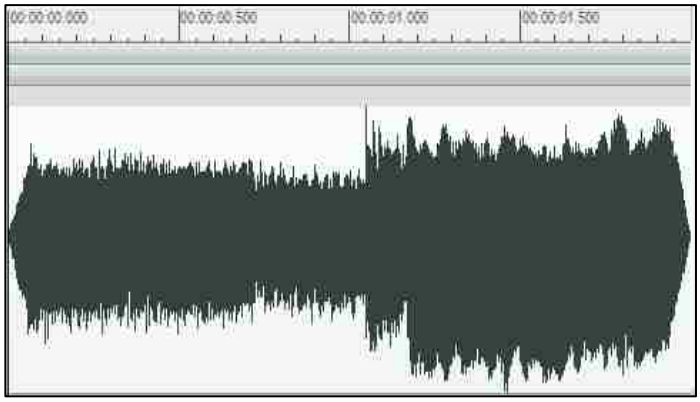


Name: Drill (Burst Ramp-up/down)
Fs: 65,536 samples per second
Nbits: 16 bits per sample
Length: 2.014 seconds
Description: Recorded noise signature of an electric drill, recorded in a semi-anechoic environment, 2 metres from the head and torso

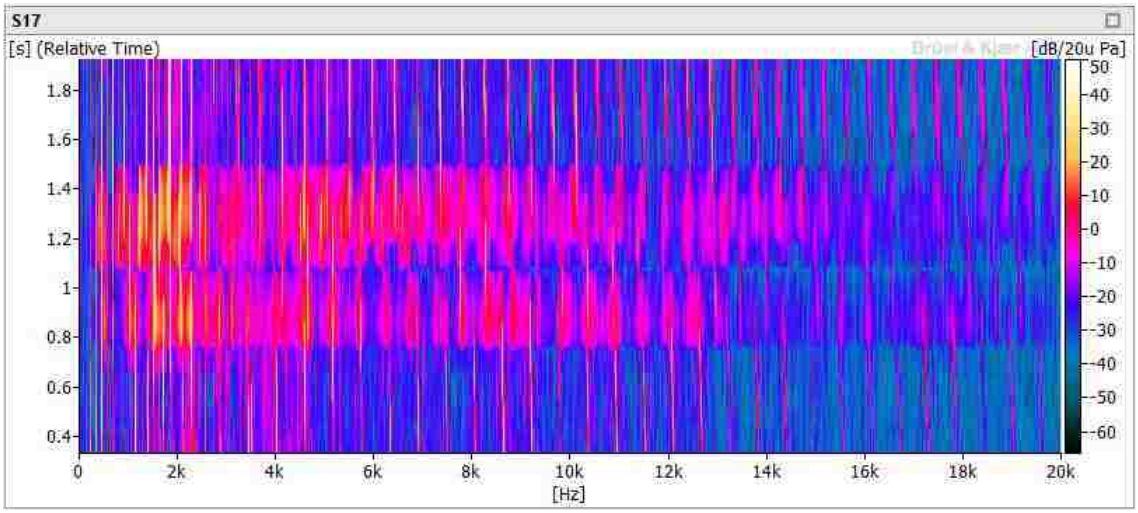


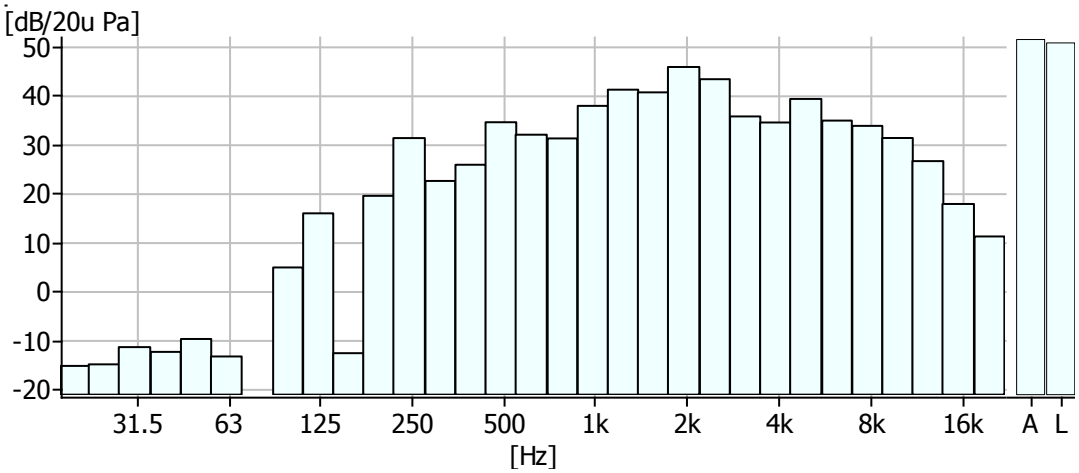


Signal 17 – Bagpipe Recording

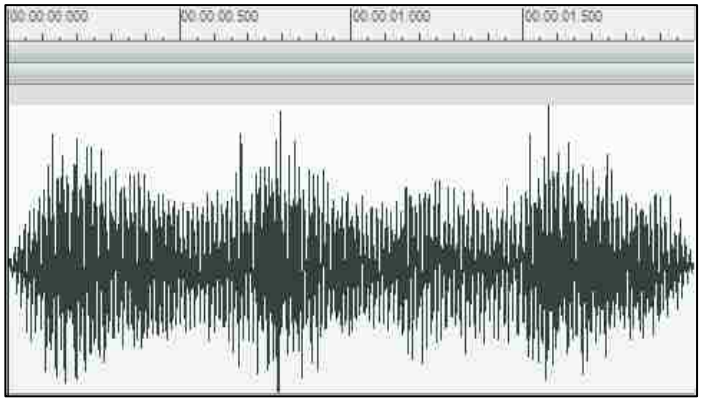


Name: Bagpipe
Fs: 44,100 samples per second
Nbits: 16 bits per sample
Length: 2.002 seconds
Description: Music segment cut from generic bagpipe soundtrack. Signal was adjusted with ramp-up/down functions to remove 'popping' sensation.

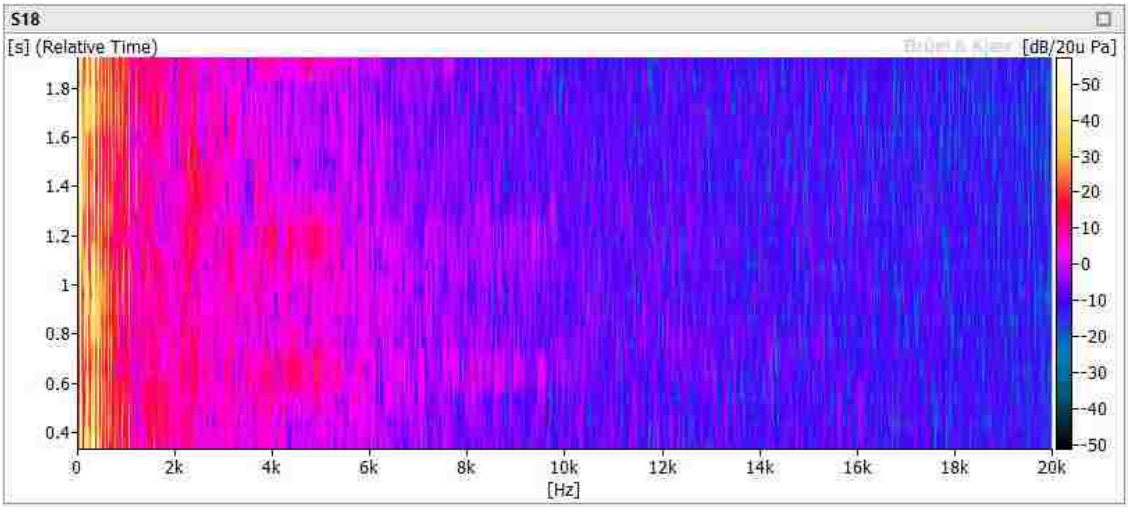


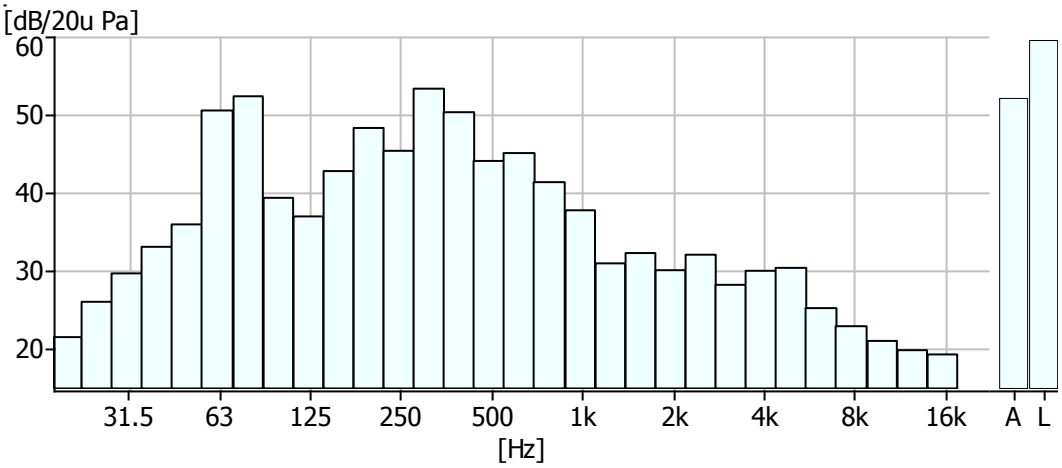


Signal 18 – Didgeridoo Recording

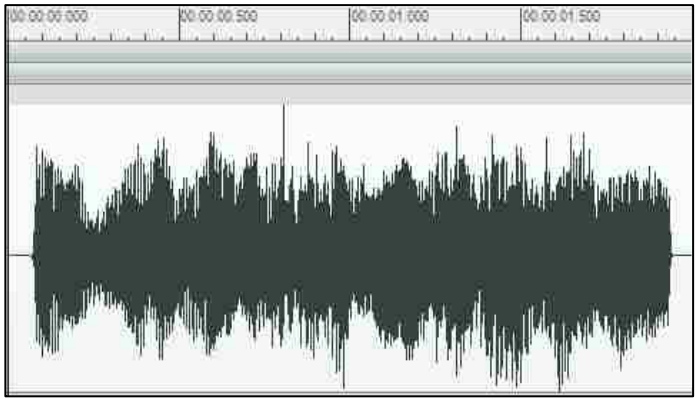


Name: Didgeridoo
Fs: 44,100 samples per second
Nbits: 24 bits per sample
Length: 2.002 seconds
Description: Music segment cut from generic didgeridoo soundtrack. Signal was adjusted with ramp-up/down functions to remove 'popping' sensation.

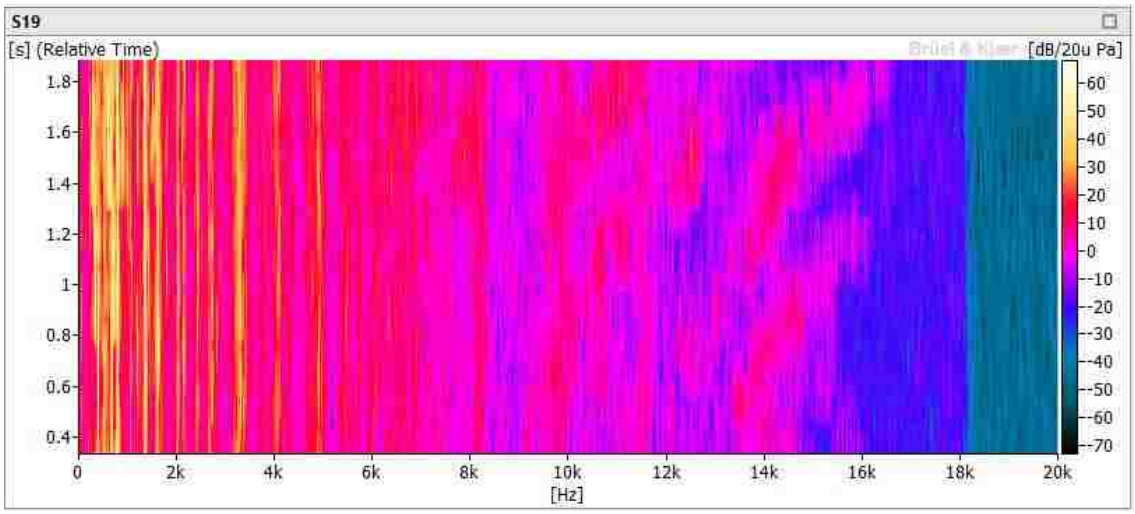


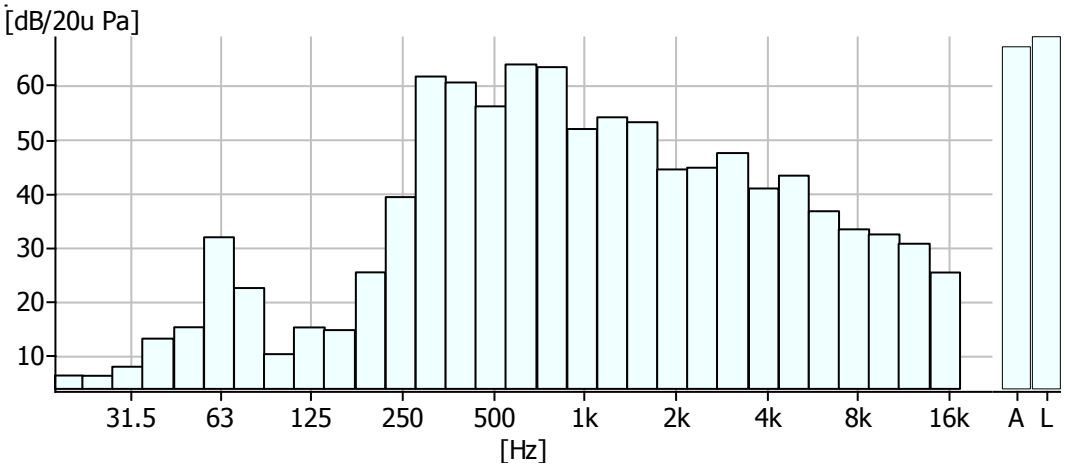


Signal 19 – Movie Intro Music

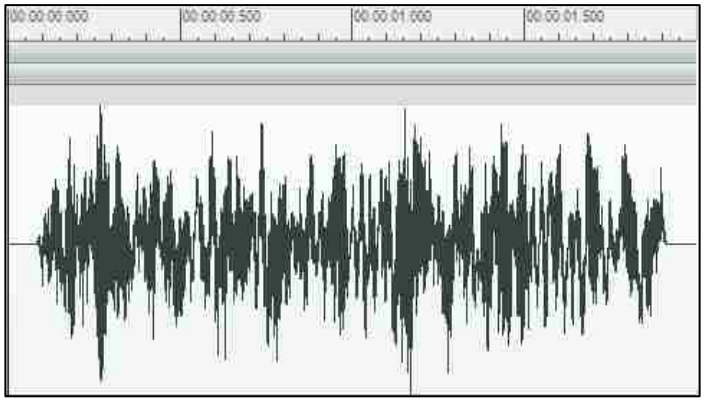


Name: Movie Intro Music
Fs: 48,000 samples per second
Nbits: 16 bits per sample
Length: 2.002 seconds
Description:
Introduction music segment cut from generic movie soundtrack. Signal was adjusted with ramp-up/down functions to remove

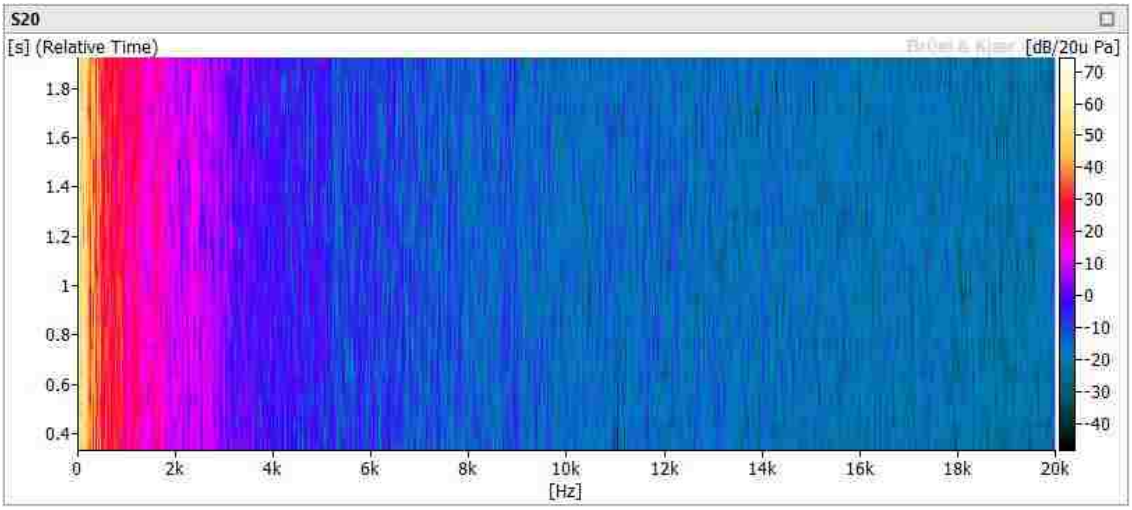


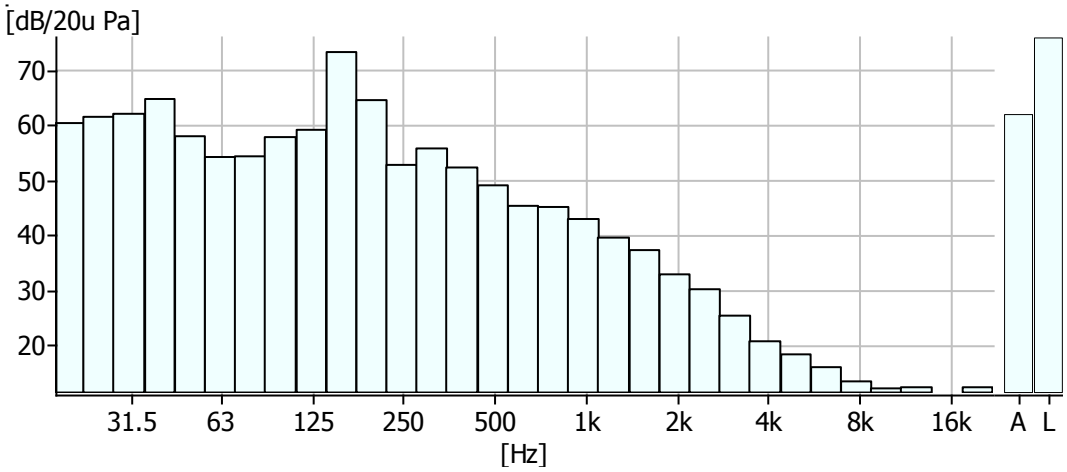


Signal 20 – Vehicle Cabin Noise

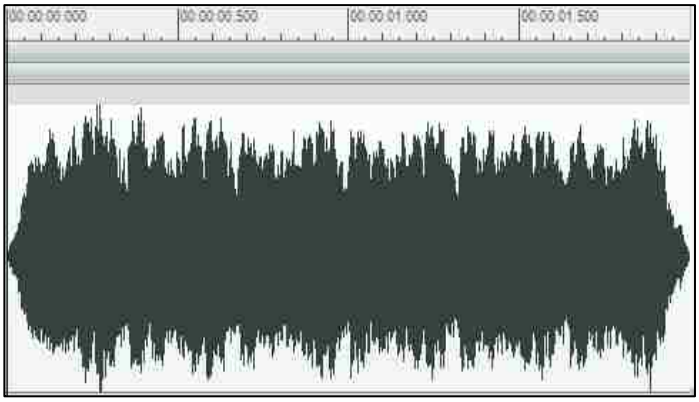


Name: Vehicle Cabin Noise
Fs: 44,100 samples per second
Nbits: 16 bits per sample
Length: 2.000 seconds
Description:

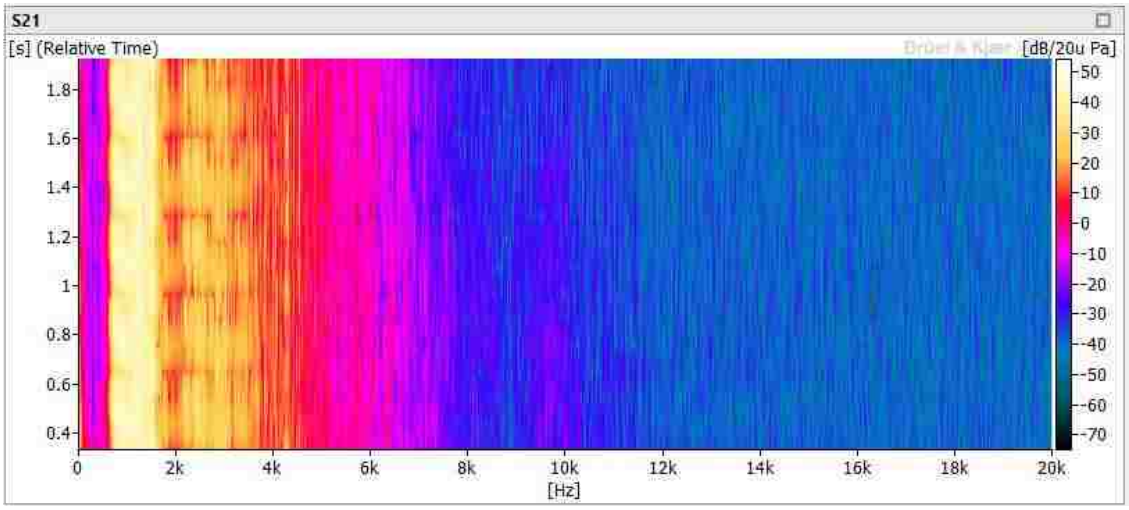


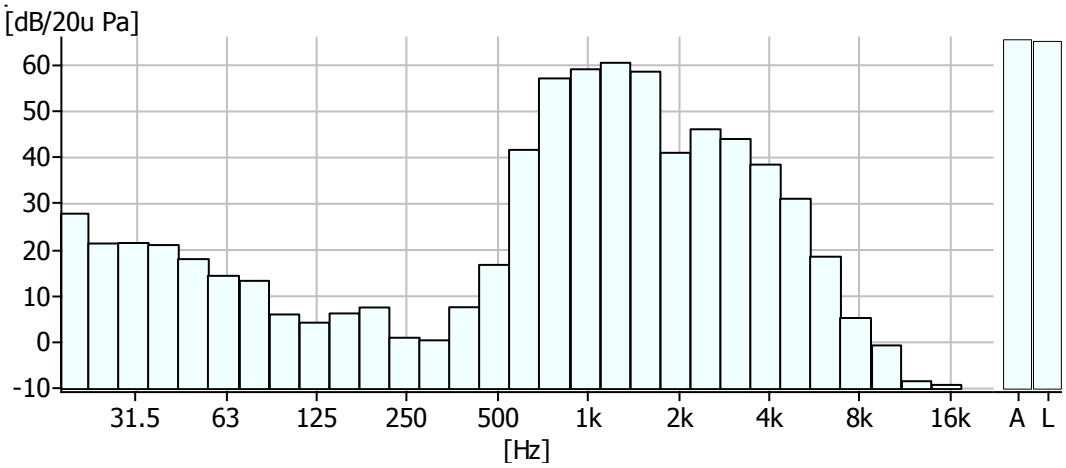


Signal 21 – Electronic Siren

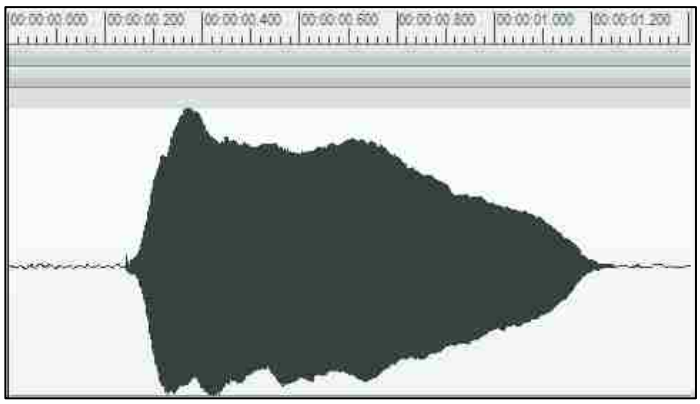


Name: Siren
Fs: 44,100 samples per second
Nbits: 16 bits per sample
Length: 2.000 seconds
Description:

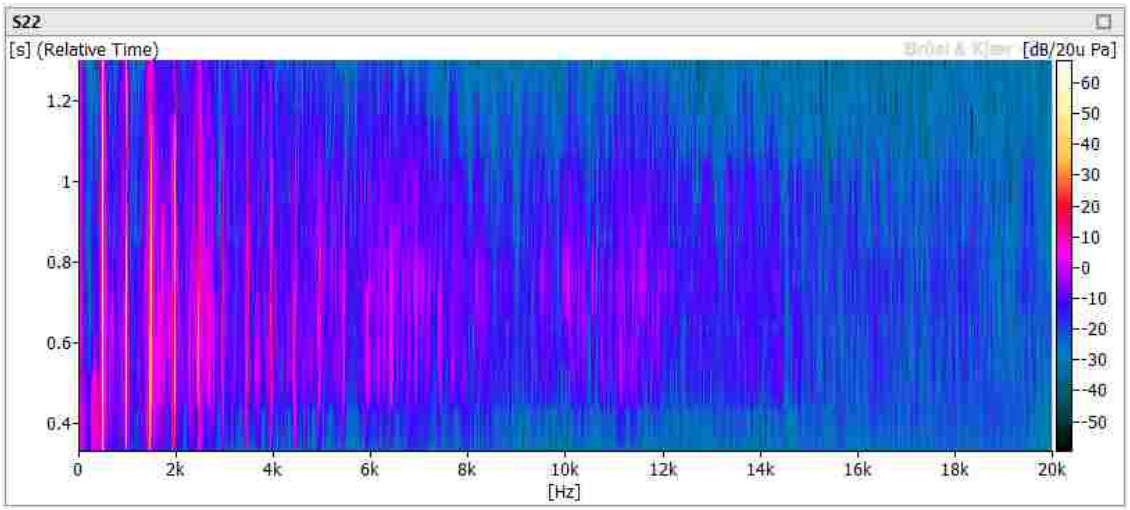


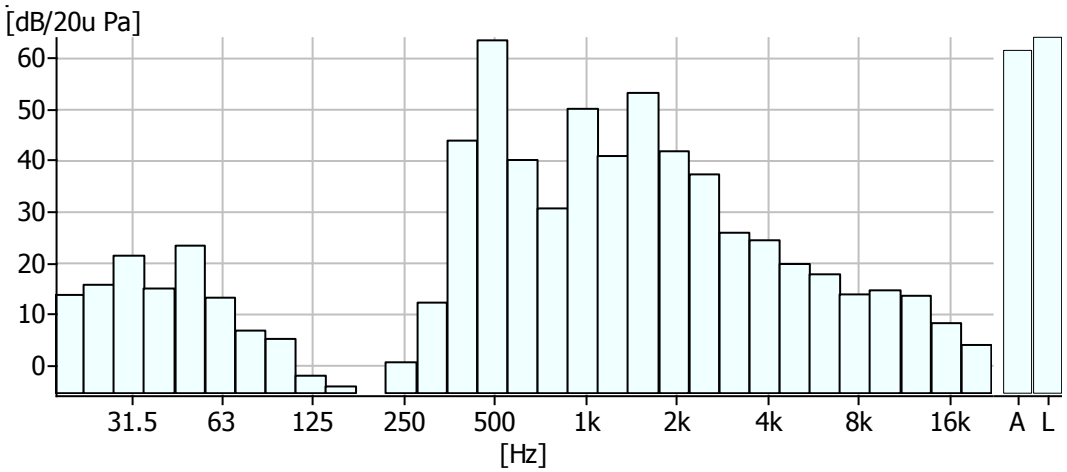


Signal 22 - Flute

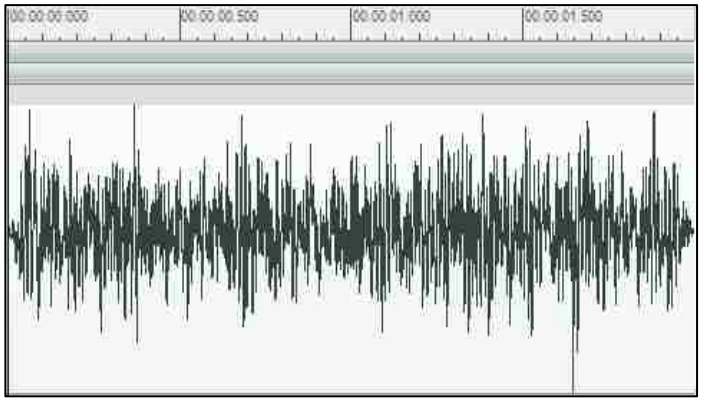


Name: Flute
Fs: 44,100 samples per second
Nbits: 16 bits per sample
Length: 1.406 seconds
Description: Flute sound segment cut from electronic source.

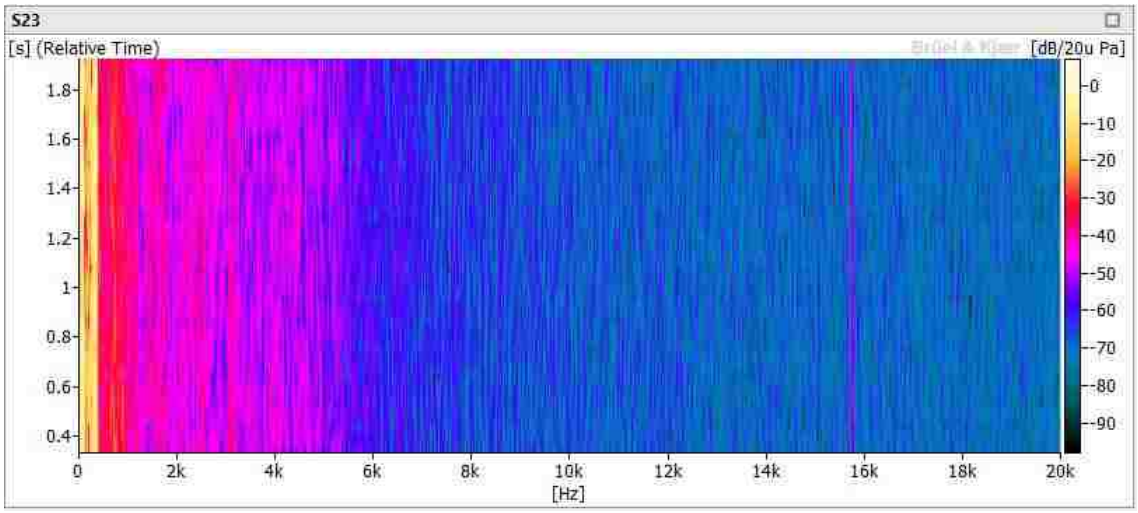


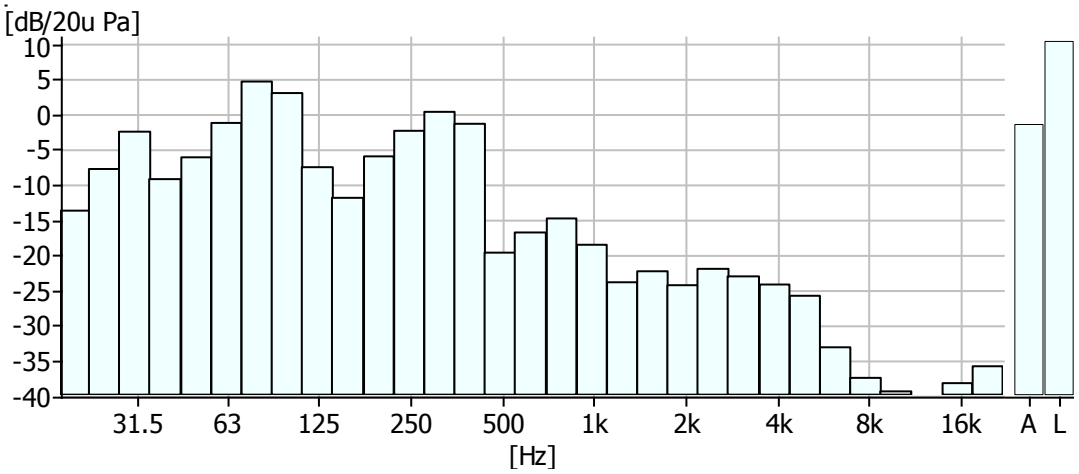


Signal 23 – Rumbler Siren

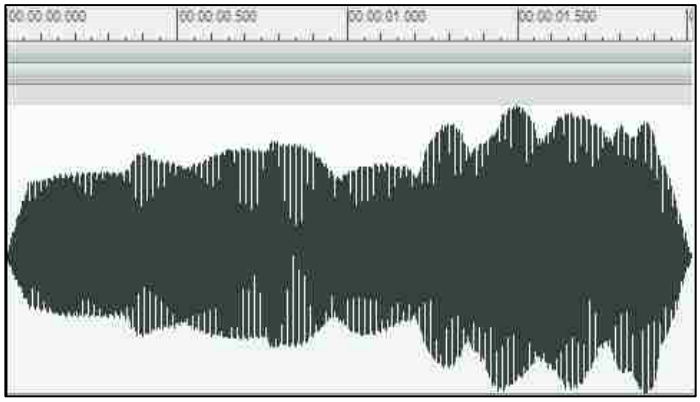


Name: Rumbler
Fs: 44,100 samples per second
Nbits: 24 bits per sample
Length: 2.002 seconds
Description:

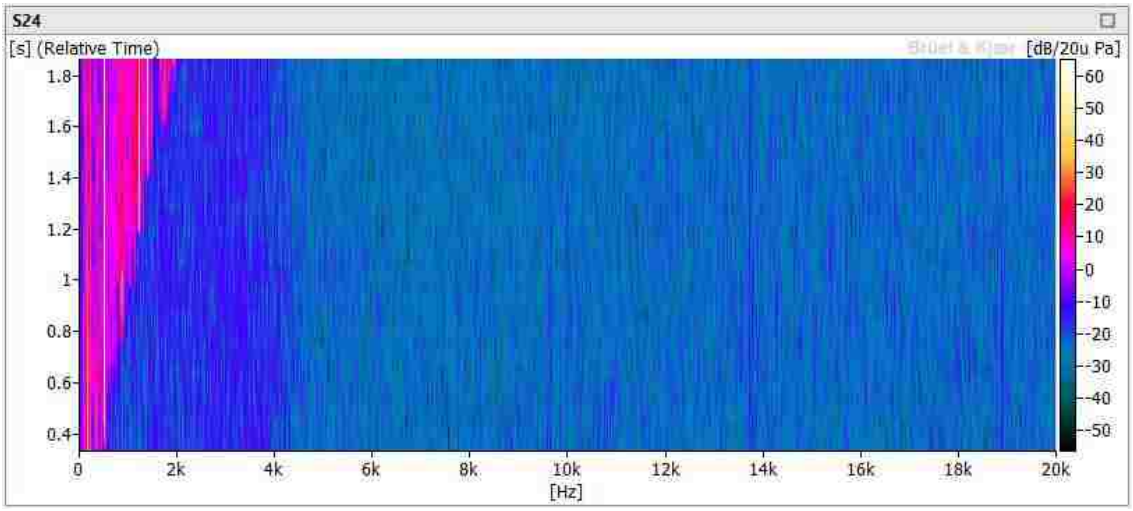


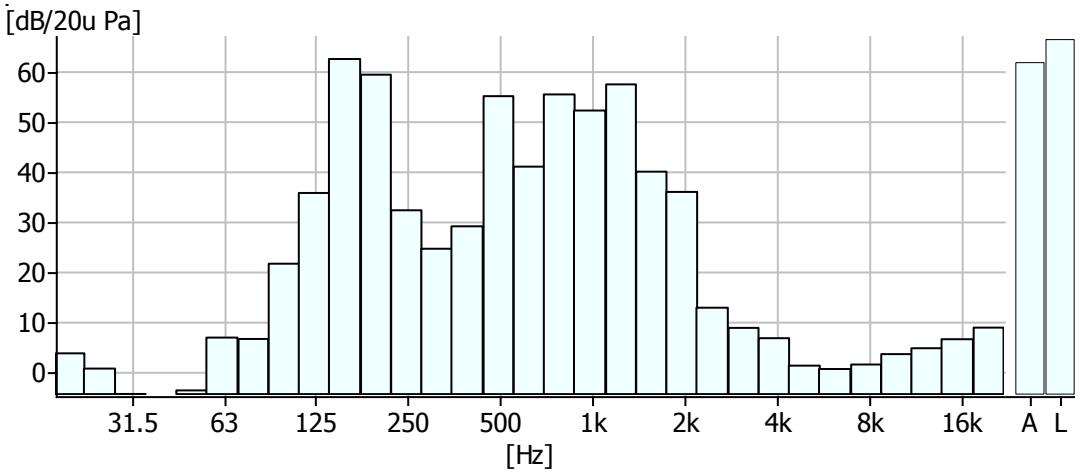


Signal 24 – Clarinet Musical Segment

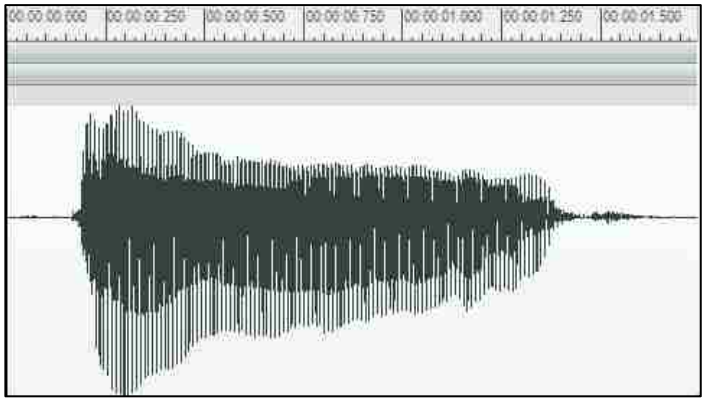


Name: Clarinet
Fs: 44,100 samples per second
Nbits: 16 bits per sample
Length: 2.015 seconds
Description: Music segment cut from generic clarinet medley. Signal was adjusted with ramp-up/down functions to remove 'popping' sensation.

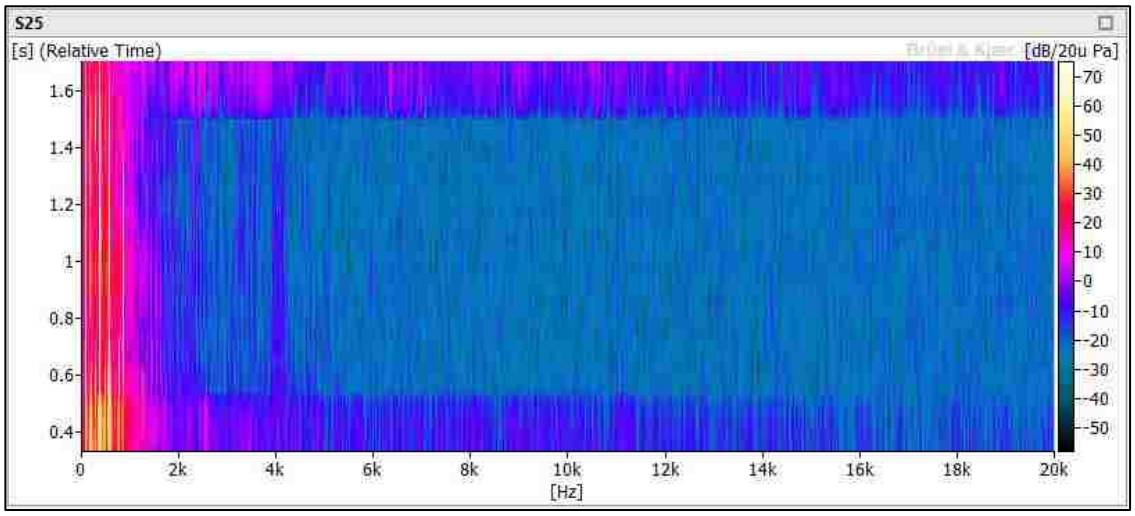


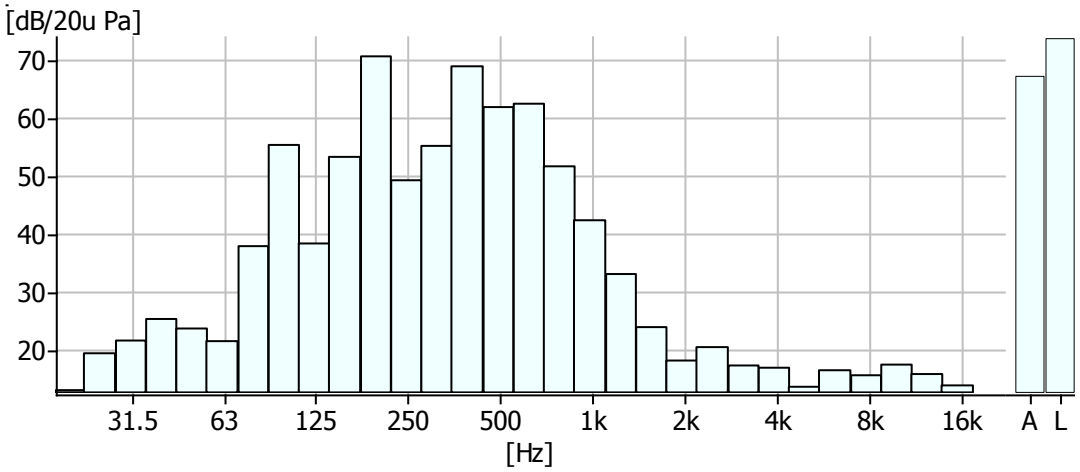


Signal 25 – Tuba Musical Segment

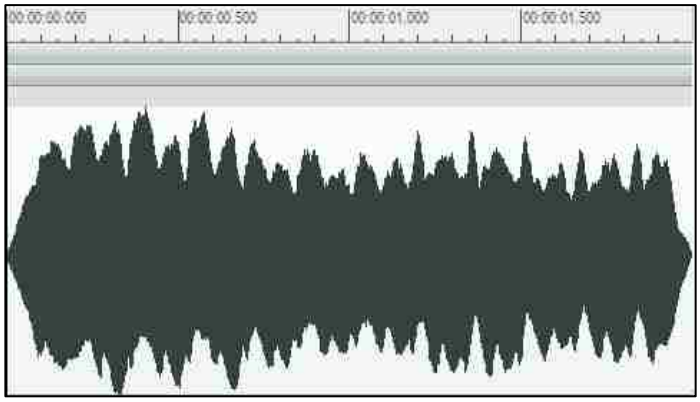


Name: Tuba
Fs: 44,100 samples per second
Nbits: 16 bits per sample
Length: 1.745 seconds
Description: Music segment cut from generic Tuba instrumental.

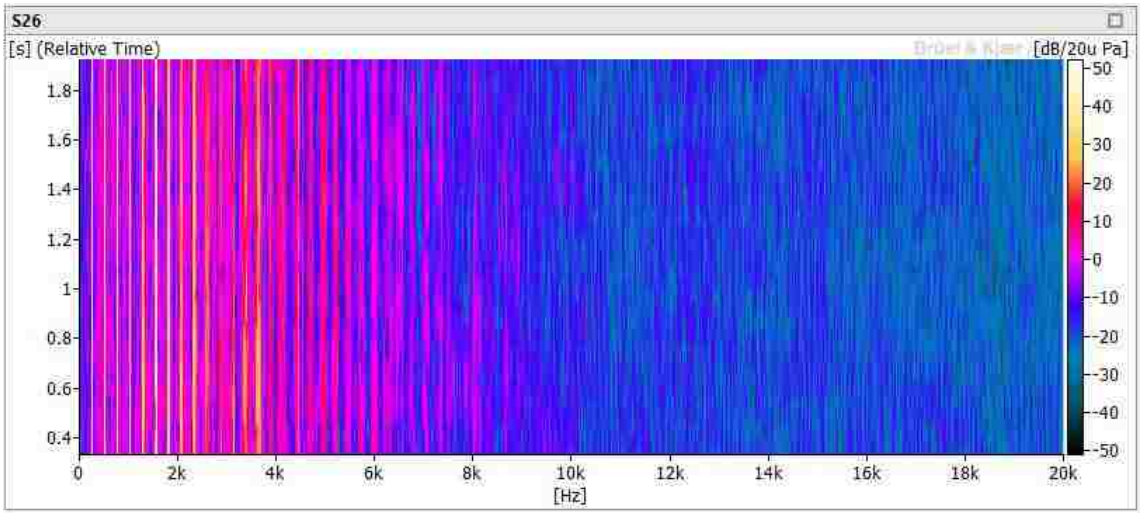


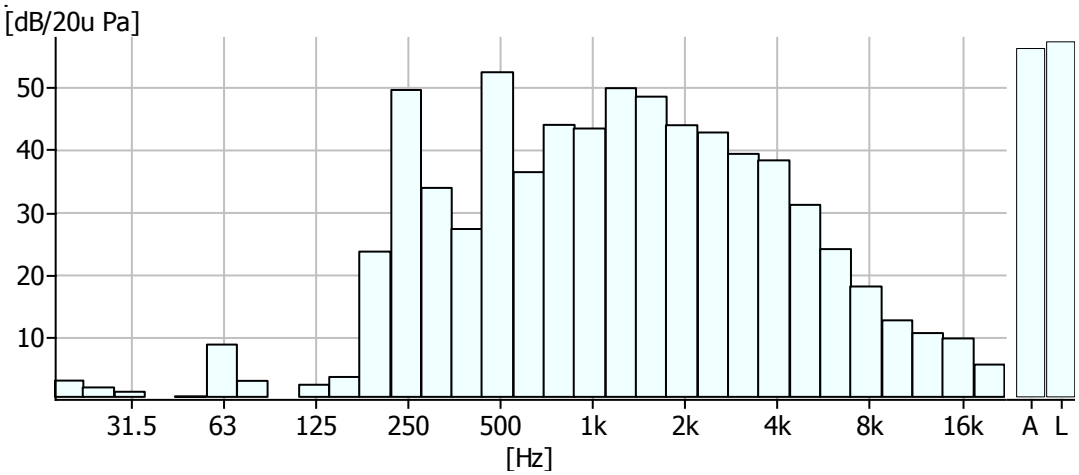


Signal 26 – Violin Musical Segment

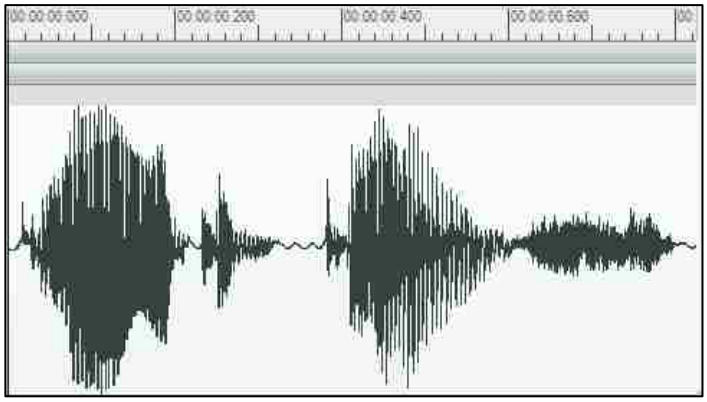


Name: Violin
Fs: 44,100 samples per second
Nbits: 16 bits per sample
Length: 2.002 seconds
Description: Music segment cut from generic violin medley. Signal was adjusted with ramp-up/down functions to remove 'popping' sensation.

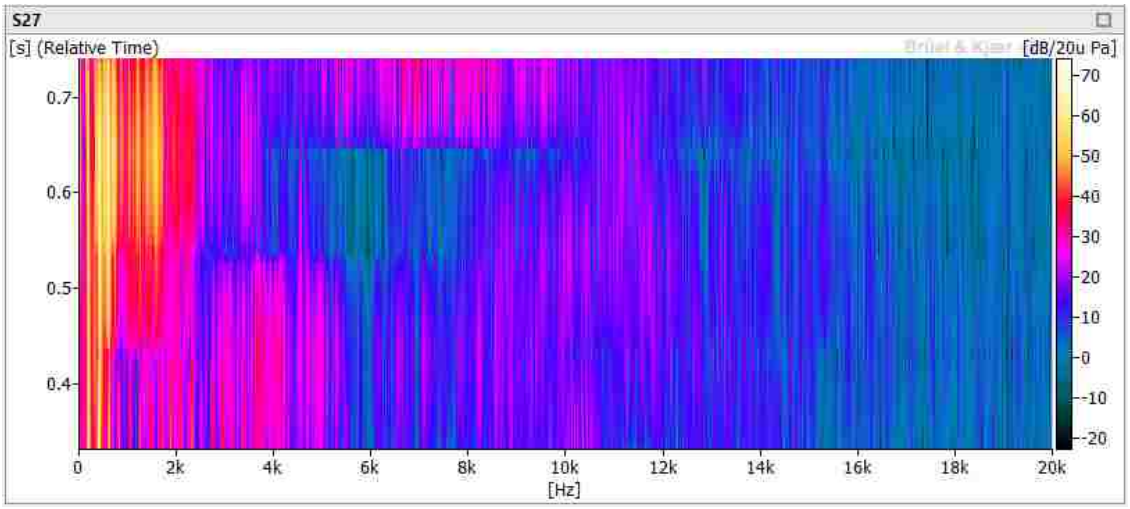


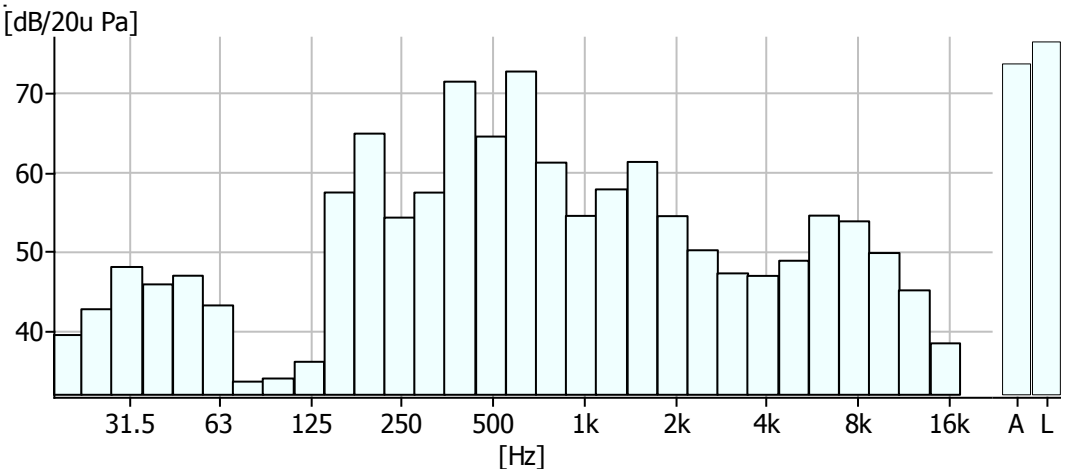


Signal 27 – Speech Segment (“Blade of Grass”)

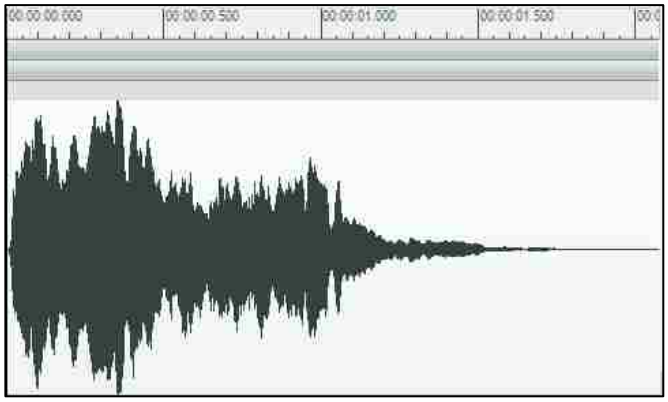


Name: Speech “Blade of Grass”
Fs: 44,100 samples per second
Nbits: 16 bits per sample
Length: 0.826 seconds
Description: Female voice recording of “Blade of Grass”

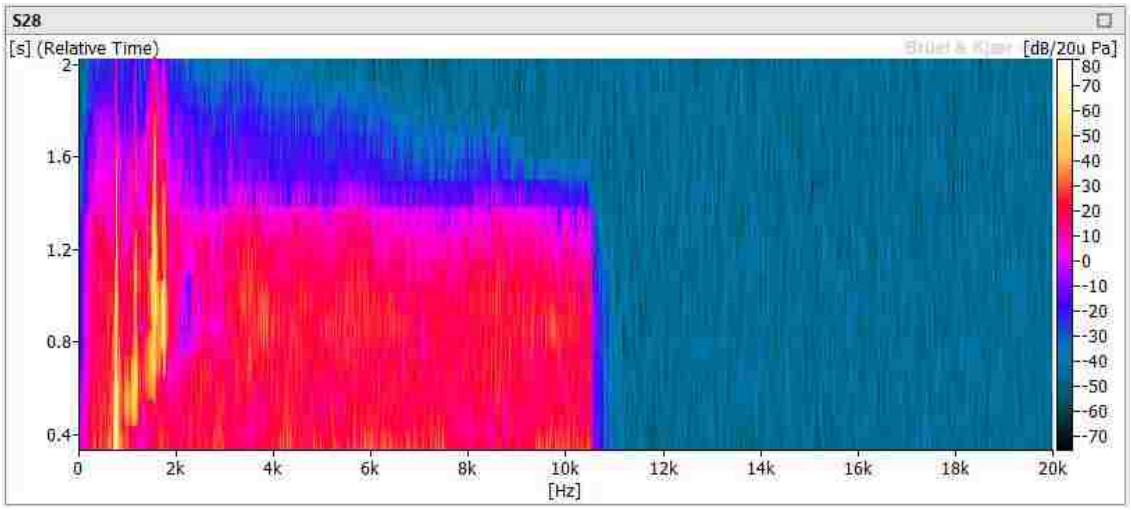


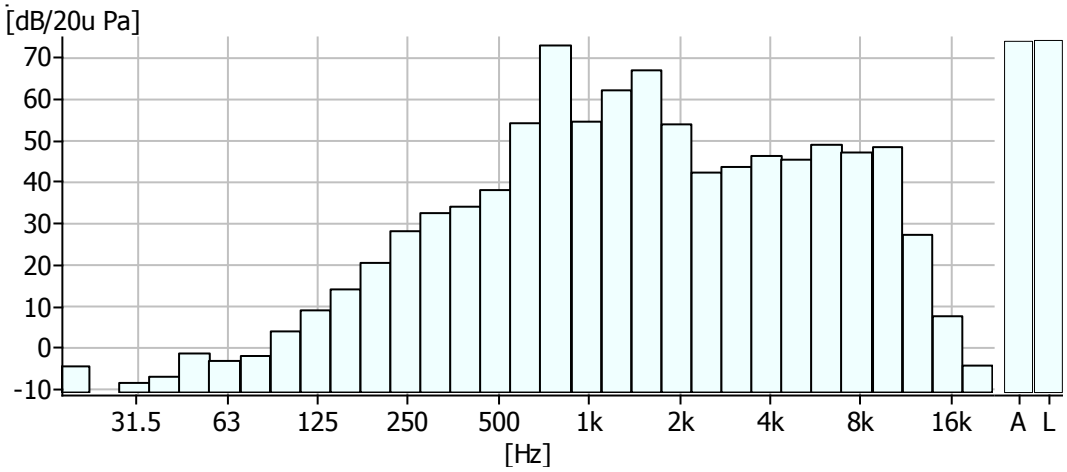


Signal 28 – Phone Notification 1 (Chiff)

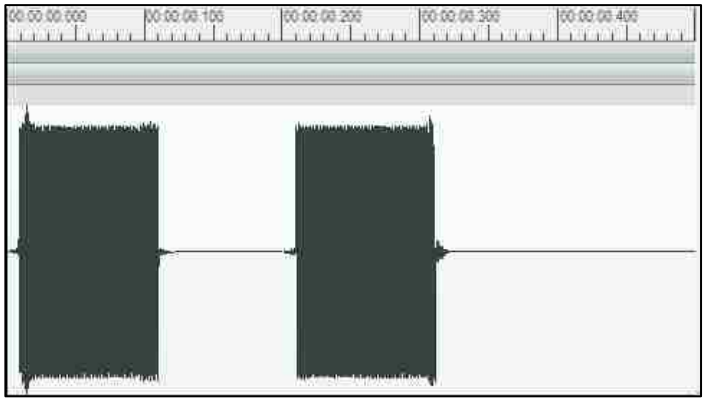


Name: Chiff Notification
Fs: 44,100 samples per second
Nbits: 16 bits per sample
Length: 2.078 seconds
Description:

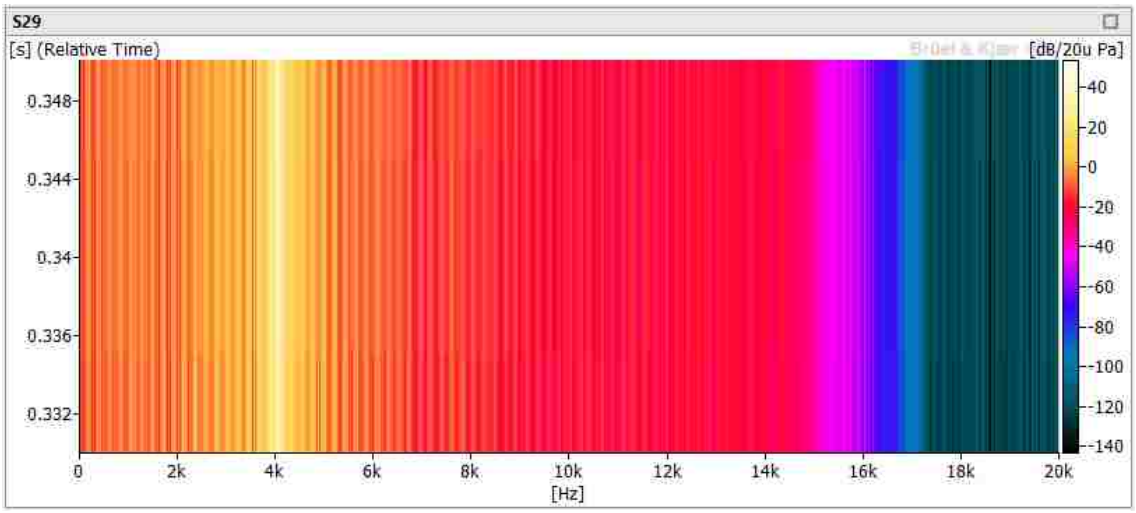


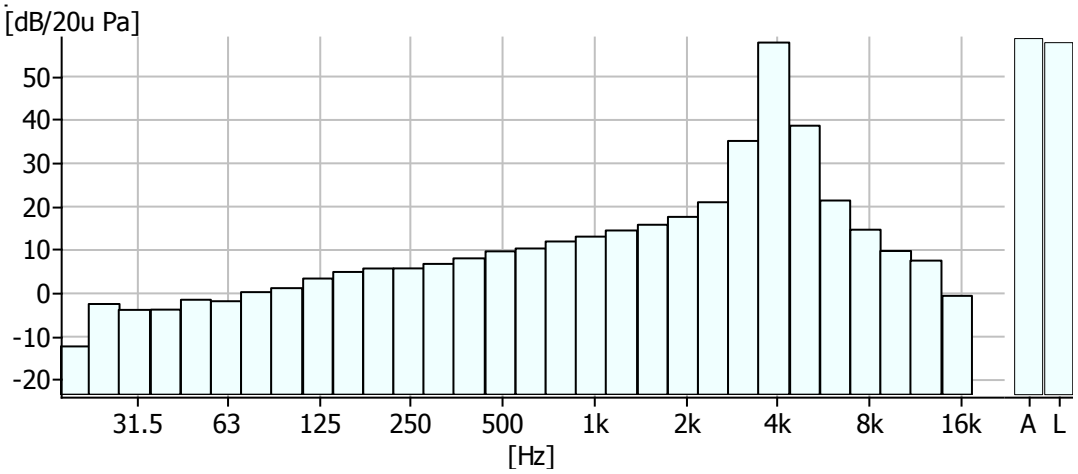


Signal 29 – Phone Notification 2 (Alarm)

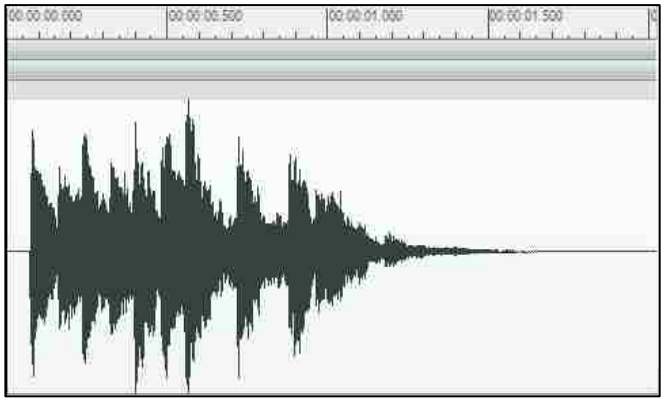


Name: Alarm Notification
Fs: 32,000 samples per second
Nbits: 16 bits per sample
Length: 0.501 seconds
Description:

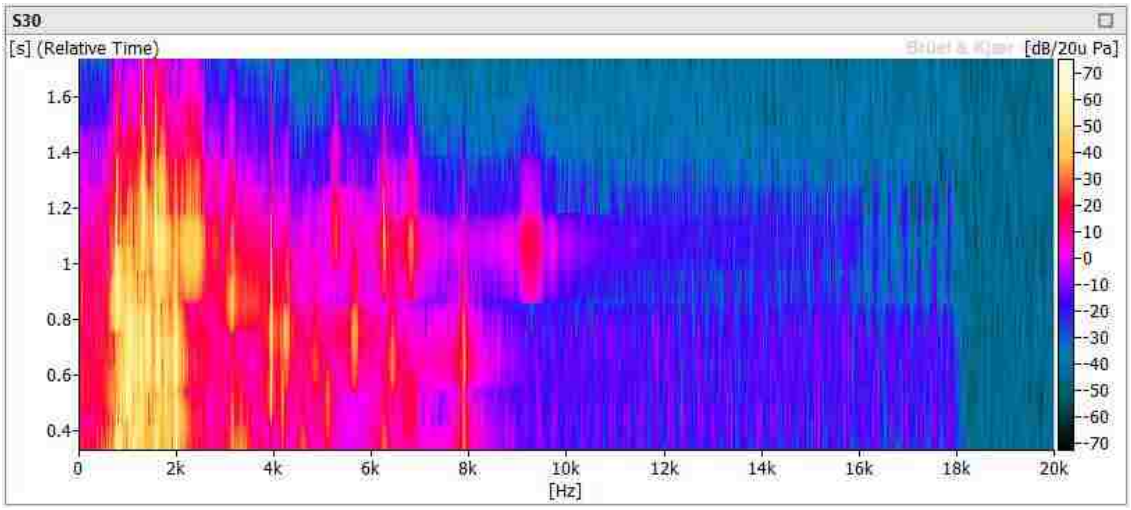


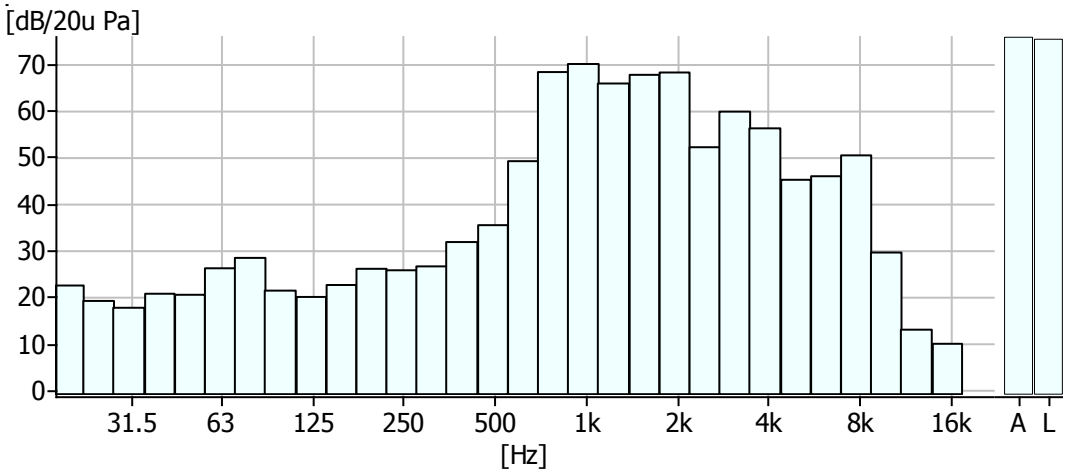


Signal 30 – Phone Notification 3 (Talkative)



Name: "Talkative" Notification
Fs: 44,100 samples per second
Nbits: 16 bits per sample
Length: 2.026 seconds
Description:





Appendix B.2: Recorded Pure Tones

Pure tones for the recorded for the verification test were subject to the limitations outlined in **Section 4.3.2** where **Figure 5.2** has been reproduced as **Figure B.1** below to summarize these limitations.

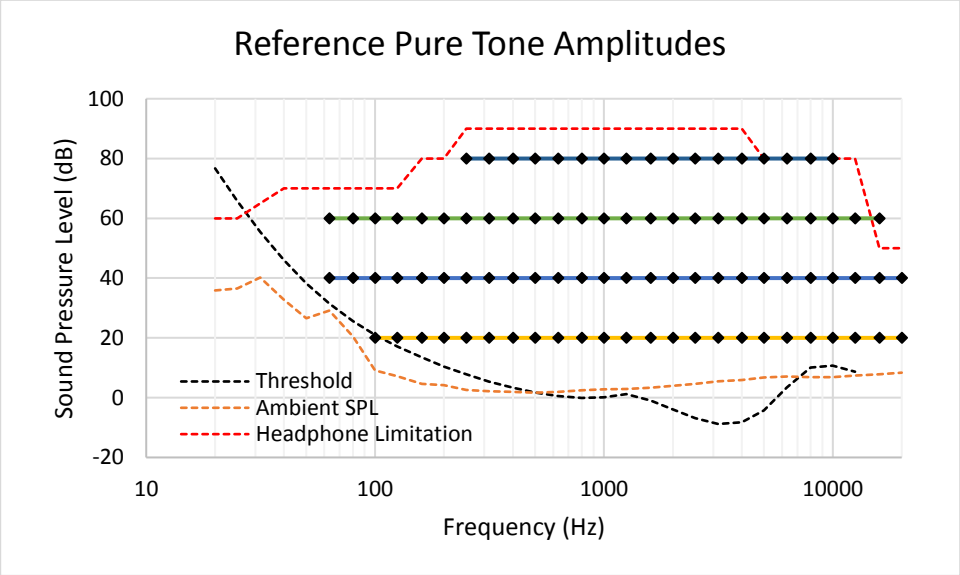


Figure B.3 - Limitations of reference pure tones (reproduced from Figure 5.2)

The detailed results from the semi-anechoic investigation are summarized in **Figures B.4** through **Figure B.6** below, with tabulated results in **Table B.1**.

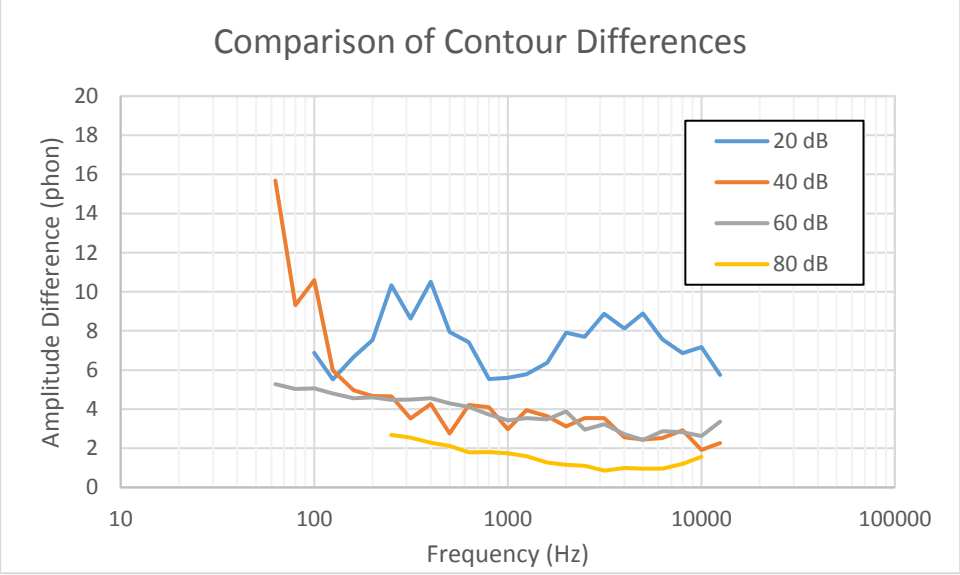


Figure B.4 – Results of pure tone verification check using semi-anechoic recordings (reproduced from Figure 5.4)

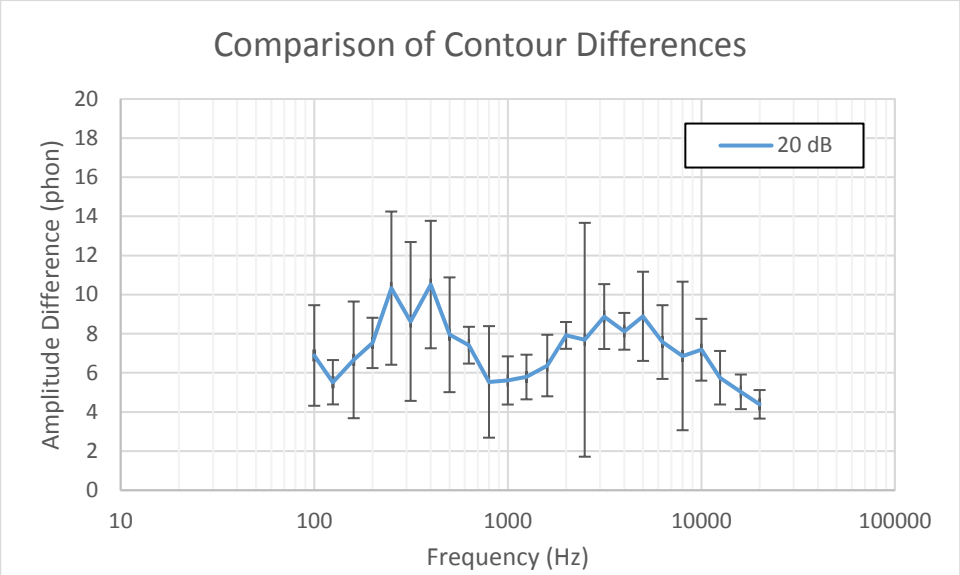


Figure B.5 - Results of 20 dB pure tone verification check with error bars.

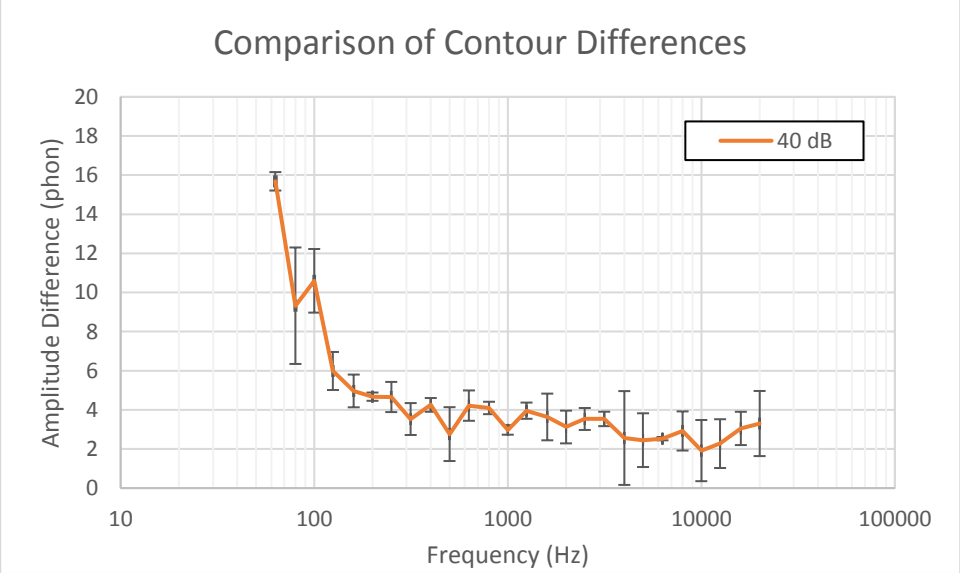


Figure B.6 - Results of 40 dB pure tone verification check with error bars.

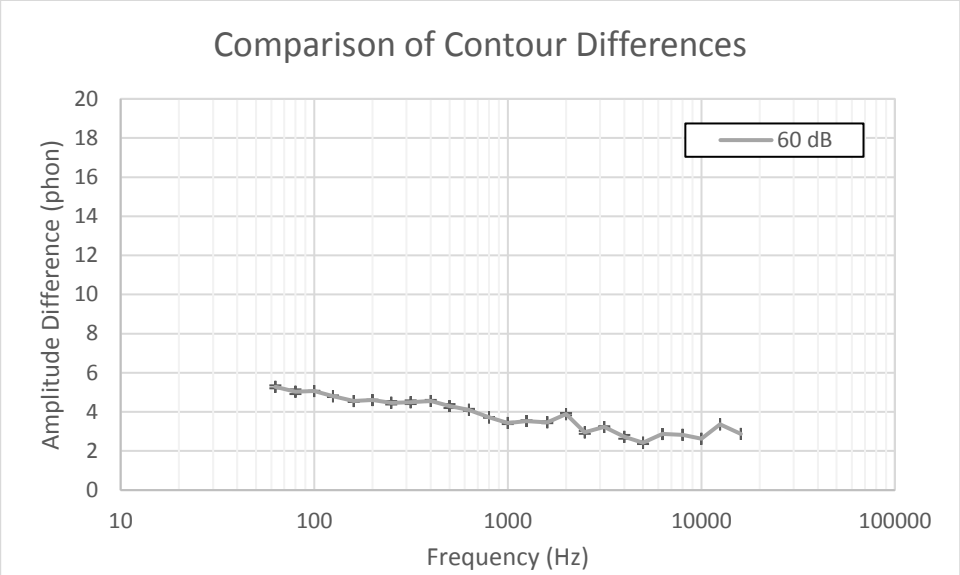


Figure B.7 - Results of 60 dB pure tone verification check with error bars.

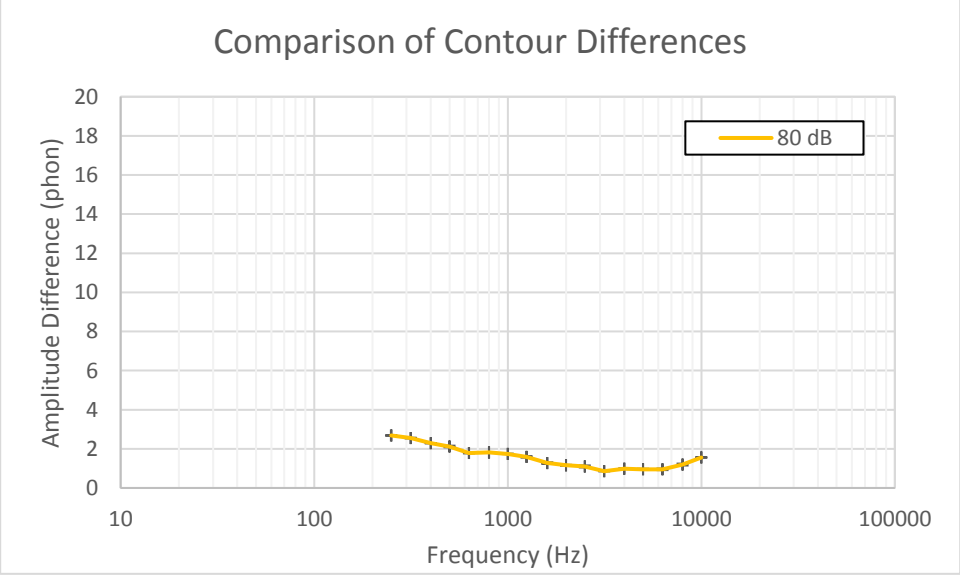


Figure B.8 - Results of 80 dB pure tone verification check with error bars.

Table B.1 – Statistical analysis of pure tone verification check with regards to recorded pure tones in semi-anechoic chamber (n=3)

	AVERAGE OF RECORDINGS				STANDARD DEVIATION			
	20 dB	40 dB	60 dB	80 dB	20 dB	40 dB	60 dB	80 dB
50								
63		15.68	5.28			0.47	0.14	
80		9.32	5.03			2.98	0.04	
100	6.89	10.60	5.06		2.57	1.63	0.04	
125	5.52	5.99	4.80		1.13	0.97	0.06	
160	6.66	4.97	4.55		2.98	0.84	0.23	
200	7.53	4.67	4.60		1.29	0.21	0.08	
250	10.33	4.66	4.47	2.68	3.92	0.77	0.12	0.02
315	8.63	3.53	4.49	2.55	4.06	0.82	0.02	0.02
400	10.51	4.25	4.56	2.29	3.26	0.35	0.04	0.01
500	7.95	2.76	4.30	2.12	2.93	1.38	0.05	0.04
630	7.41	4.22	4.11	1.79	0.94	0.78	0.02	0.02
800	5.54	4.10	3.72	1.81	2.85	0.32	0.09	0.01
1000	5.61	2.97	3.43	1.74	1.23	0.24	0.09	0.01
1250	5.79	3.96	3.53	1.59	1.14	0.42	0.06	0.04
1600	6.37	3.64	3.47	1.27	1.57	1.19	0.10	0.04
2000	7.91	3.12	3.88	1.16	0.69	0.84	0.05	0.03
2500	7.69	3.53	2.95	1.10	5.98	0.56	0.02	0.05
3150	8.88	3.54	3.23	0.86	1.66	0.37	0.06	0.01
4000	8.12	2.56	2.72	0.99	0.94	2.40	0.04	0.02
5000	8.89	2.45	2.42	0.96	2.28	1.37	0.05	0.03
6300	7.57	2.53	2.87	0.95	1.88	0.09	0.06	0.04
8000	6.86	2.92	2.83	1.20	3.80	1.00	0.08	0.05
10000	7.18	1.92	2.62	1.56	1.58	1.56	0.04	0.01
12500	5.75	2.27	3.36		1.37	1.25	0.10	
16000	5.03	3.05	2.87		0.89	0.85	0.07	
20000	4.39	3.30			0.73	1.66		
				Min.	0.69	0.09	0.02	0.01
				Max.	5.98	2.40	0.23	0.05

Appendix B Bibliography

[1] P. N. Vassilakis, "Course: 43-2310 Psychoacoustics," 2013. [Online]. Available: Acousticslab.org.

[2] K. Ortega, Director, *Hocus Pocus.* [Film]. 1993.

[3] "[Download] Android Devices Stock System Sounds," 14 May 2013. [Online]. Available: http://forums.theandroidchannel.com/topic/203-download-android-devices-stock-system-sounds/.

APPENDIX C: Curve Fit Comparison

C.1 – Sensation Level Investigation: Raw Data

The results of the sensation level experiment confirmed that the two methods of signal presentation produced nearly identical results. This was confirmed from the resulting average trends demonstrated in **Figure C.1** below (reproduced from **Figure 4.13**).

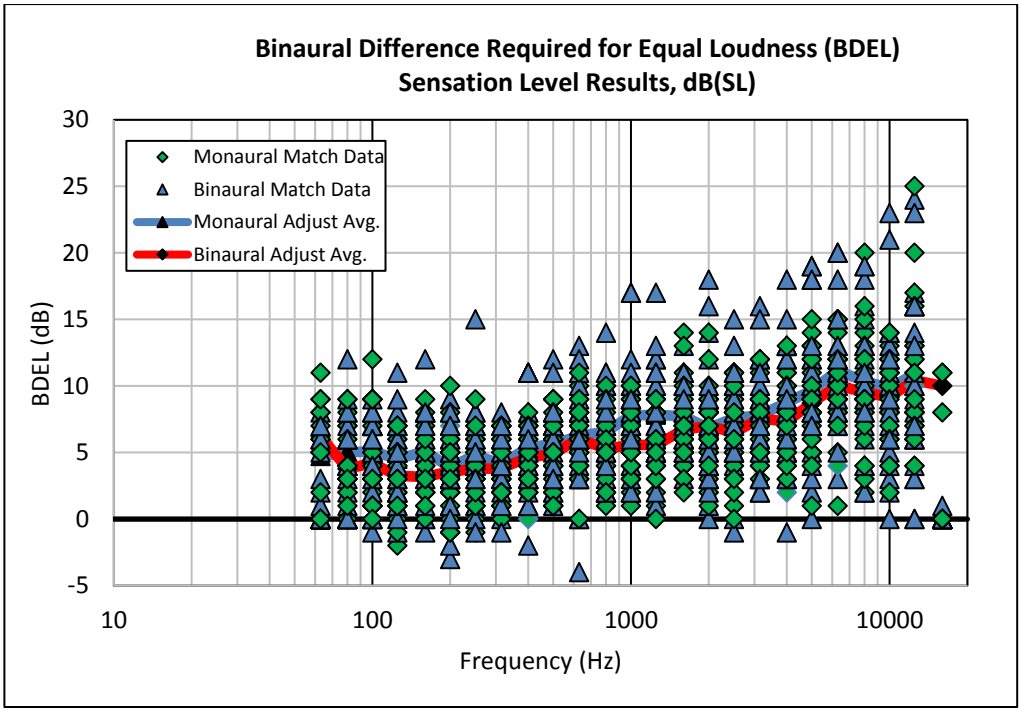


Figure C.1 – Results of sensation level experiment (reproduced from Figure 4.13).

The resulting trend appeared as a binaural summation function which increased with frequency. While curve fitting this trend several functions were investigated where results are summarized in the figures below. Each of the following trend-lines were derived using built in Microsoft Excel functions. During the comparison, disregard any significant digits greater two as the sound pressure levels collected were limited to this resolution. The conclusions from these results were summarized in **Section 5.2.1**.

C.2 – Curve Fitting Investigation

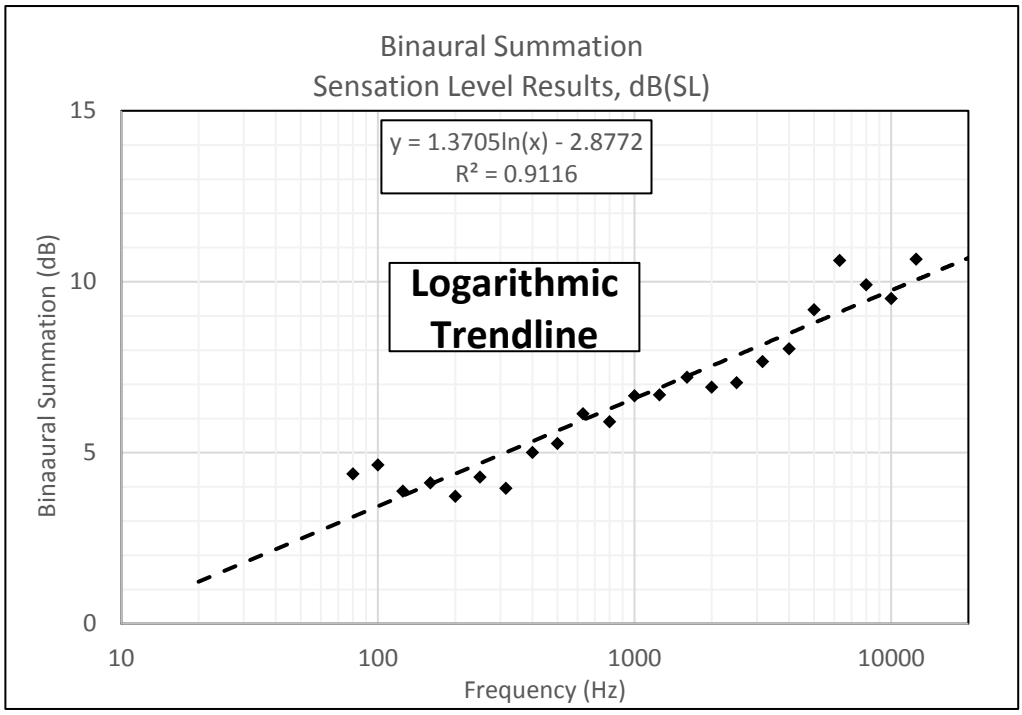


Figure C.2 - Logarithmic Trend-line

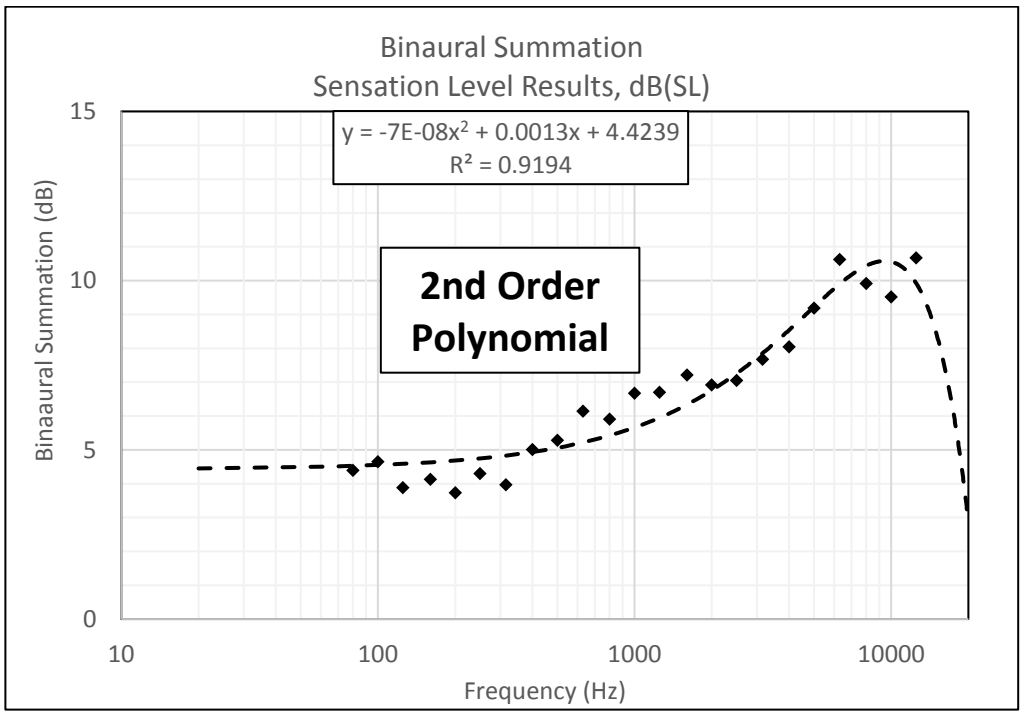


Figure C.3 - 2nd Order Polynomial

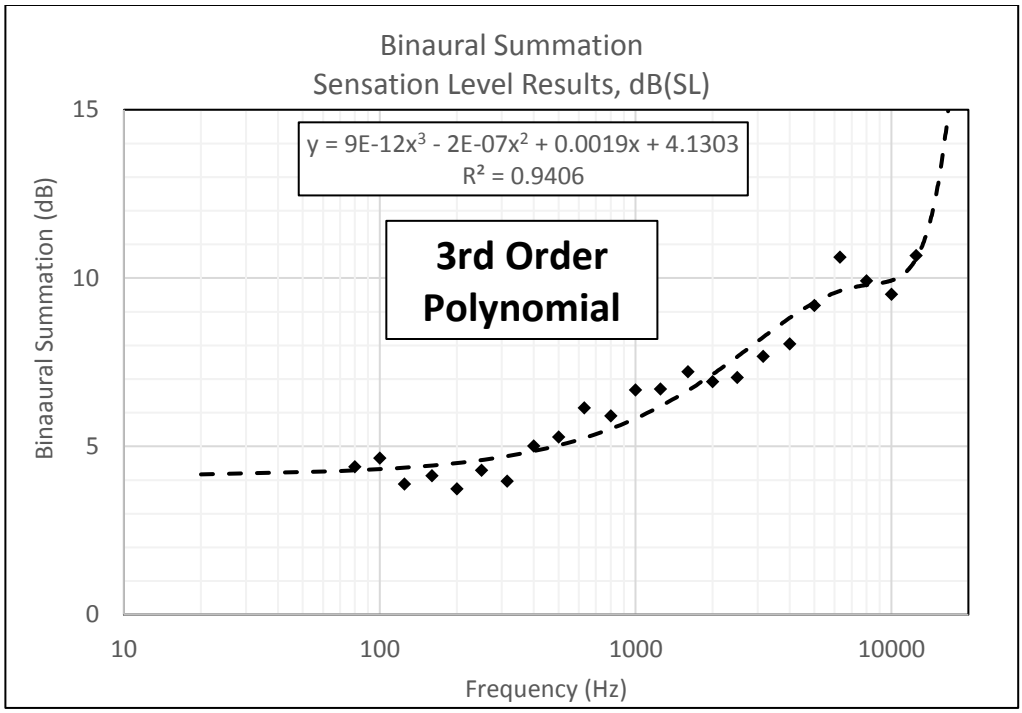


Figure C.4 - 3rd Order Polynomial

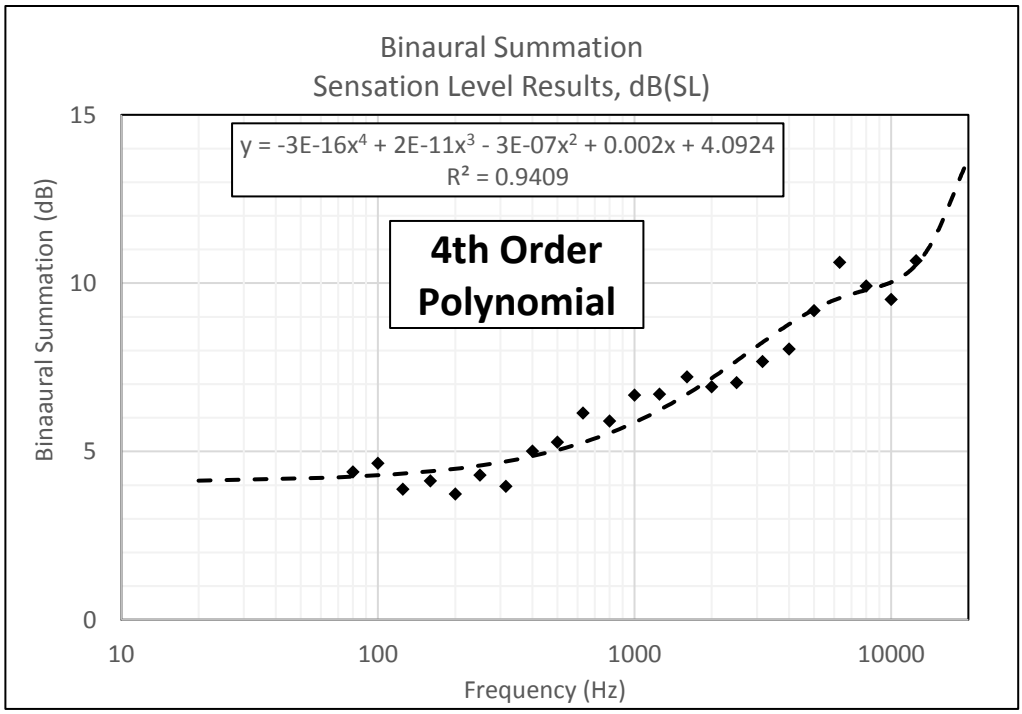


Figure C.5 - 4th Order Polynomial

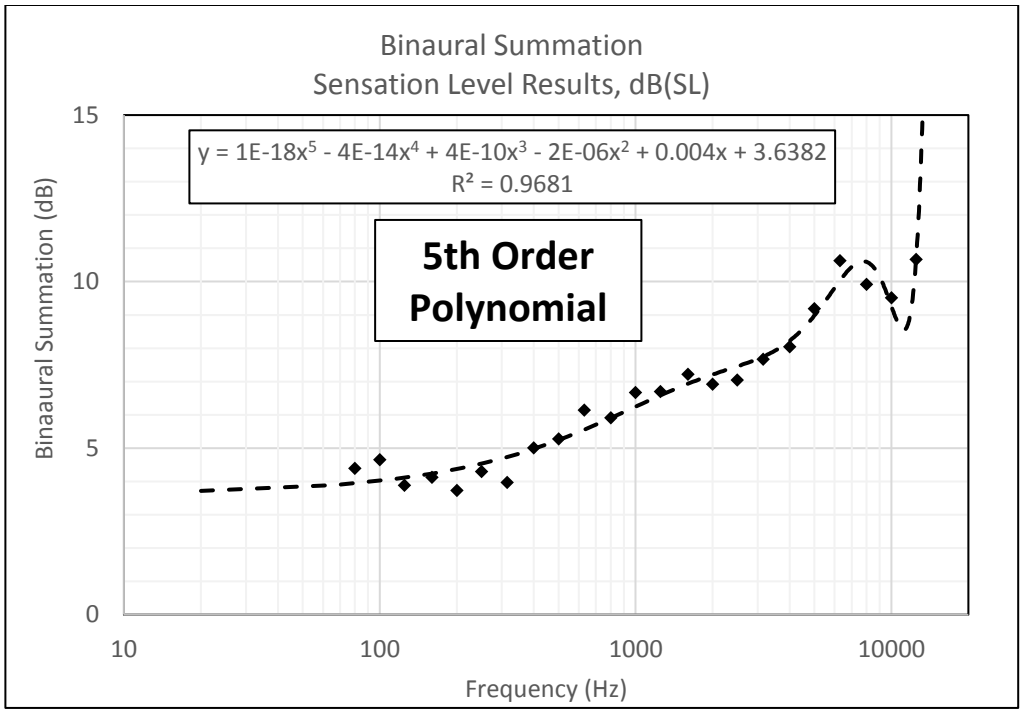


Figure C.6 - 5th Order Polynomial

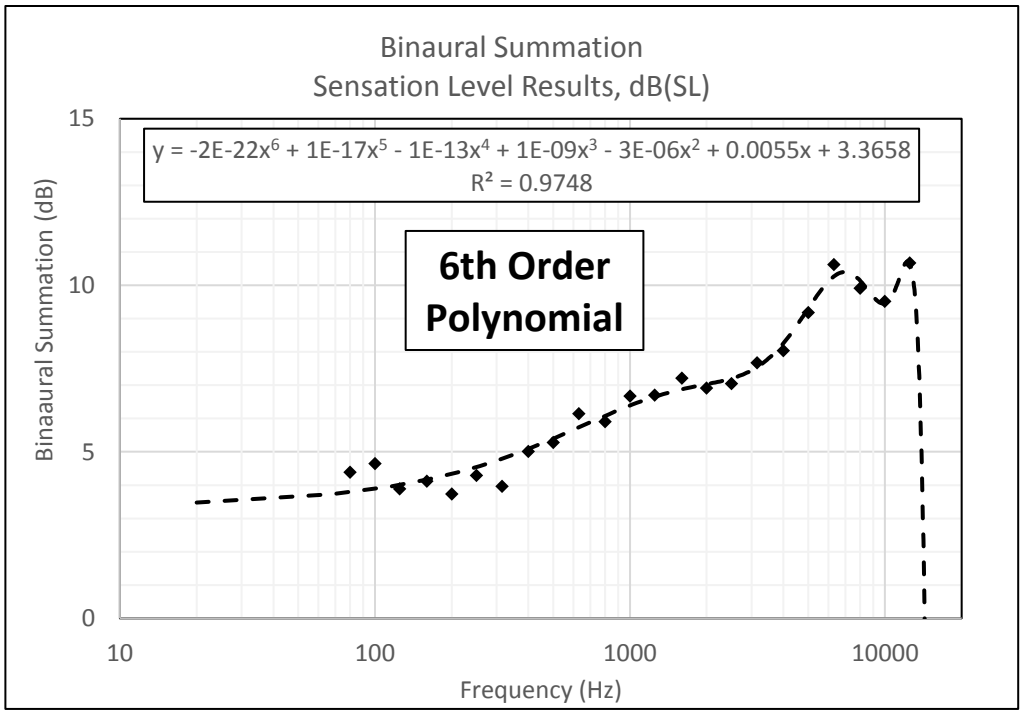


Figure C.7 - 6th Order Polynomial

APPENDIX D: Data Sensitivity Check

Appendix D: Data Sensitivity Check (MATLAB Code)

In order to verify that the population size chosen for the listening test was appropriate (ie 50 individuals), the sensitivity of the averaged result was investigated by removing random samples. To accomplish this the MATLAB Code below was written to randomly select a specified number of data sets to remove (Sens_Check), and calculate the resulting average. This process was repeated a set number of times (Num_Runs), in order to capture all of the possible combinations of averages with a smaller sample size. As a measure of sensitivity the maximum and minimum values from all of the averages was recorded and stored as maximum and minimum contours respectively.

For example, consider four measurements were taken (A, B, C, and D) where the average was calculated as E. How would the resulting average change if only elements A, B, and D were included in the average; and likewise had a different variable been removed instead? The following code answers this question and records the maximum variance in the resulting average.

D.1 MATLAB Code:

To run, an array of data results must already exist, here it has been named **"Resultant."** Resultant is an array where columns represent successive trials and rows represent the frequency spectrum in one-third-octaves from 20 Hz to 20 kHz (31 rows).

MATLAB M-FILE – Results.m

```
function [XData, MaxArray, ResAverage, MinArray, SpreadArray] = AverageCheck(XData, YData, Sens_Check, Num_Runs, Create_Figure)
%
% [XData, MaxArray, ResAverage, MinArray, SpreadArray] = AverageCheck(XData, YData, Sens_Check, Num_Runs, Create_Figure)
%
% = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
%
%   Description: Sensitivity check program which verifies that the current
%                   number of data points is sufficient where the resulting
%                   average is marginally impacted by each successive data
%                   set.
%
%     Operation: Calculates the average of the data-set once a set number
%                   of values (Sens_Check) are removed randomly. This
%                   process is repeated a specified number of times
%                   (Num_Runs), where the maximum and minimum values of the
%                   variation are collected and returned to the command
%                   window.
%
%        Author: Jeremy Charbonneau
%       Revised: 2015-06-29
%    Insitution: University of Windsor, Windsor, Ont. Canada
%
% = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
%   INPUTS
%       XData - Constant data used for all averages
%                   MUST be the same size as the YData rows.
%       YData - Array of data to test for sensitivity
%       Sens_Check  - Number of values to take out of the Average   (ie 10)
%       Num_Runs    - Number of Averages to run                     (ie 1000)
%       Plot_Figures    - Set to 1 if Figure has not been opened    (ie 0)
%
%       ie. AverageCheck(Freq, ResultantData, 5, 1000, 1)
%
% = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
%   OUTPUTS
%       XData - Constant data used for all averages, entered by USER
%       MaxArray    - Maximum variation from the averaging process.
%       ResAverage  - Average of entire Array as entered by USER
%       MinArray    - Minimum variation from the averaging process.
%       SpreadArray - Spread of variance, (ie. Max - Min)
%
% = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
%   EXAMPLE
%
%       ie. AverageCheck(Freq, ResultantData, 5, 1000, 1)
%
% = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =

     ignore_below = -5;  % Ignore all y-values below this (set to NaN)
     StatsFig = 0;       % Initiating handle for figure.

%%
clc     % Clear MATLAB Command Window

% Reading Array Size                        % Example Outputs
    ArraySize = size(YData);                % 31 52
        Number_of_Trials = ArraySize(1,2);  % Number of Trials in the Array
        NumRows = ArraySize(1,1);           % 31

    XData_Check = size(XData);              % 1 31

if XData_Check(1,2) == NumRows;


% Reading Size of X-Values (Frequency)
    Fsize = size(XData);    % 1 31

% Creating Min and Max Arrays as Placeholders
    MaxArray = zeros(ArraySize(1,1),1);
    MinArray = ones(ArraySize(1,1),1);
    MinArray = MinArray*100;   % Setting high value to be replaced

%% Set plot/average to ignore values indicated by -10 (-10 --> NaN)
    for plotall = 1:ArraySize(1,2)
        for GapVal = 1:Fsize(1,2)   % Replacing Zeros with NaN for Plotting
            x = YData(GapVal,plotall);

            if x >= MaxArray(GapVal,1);
                MaxArray(GapVal,1) = x;
            end

                x(x == -10) = NaN;
                if x <= ignore_below; x = NaN; end

                if  x <= MinArray(GapVal,1);
                    MinArray(GapVal,1) = x;
                end;

                if MaxArray(GapVal,1)<=1;
                    if isnan(x);
                        MaxArray(GapVal,1) = NaN(1);
                    end;
                end
            YData(GapVal,plotall) = x;
        end

    end

%% Random Sampling of Data
     % How would average be impacted without "x" number of data-sets

     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%         Number_of_Trials = 45;
%         Sens_Check = 10;    % "x" Value for check
```

```
 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    % Number of trials in Average_Check
        Trials_to_AVG = Number_of_Trials - Sens_Check;

    % Run this Average_Check "Num_Runs" amount of times
%         Num_Runs = 10000;
    % Creating blank array blocks to store data in
        AVG_Array = zeros(NumRows,Trials_to_AVG);
        AVG_Run_Array = zeros(NumRows,Num_Runs);

%=================================================================%
%   Average Run of "Num_Runs" Samples (1000)
%=================================================================%
    % Run "Num_Runs" Times
    for AVG_Run = 1:Num_Runs;

        % Create random order of Trials (minus sensitivity check)
        Order = randperm(Number_of_Trials,Number_of_Trials);

        % Building Random Array
        for AVG_Incl = 1:Trials_to_AVG
            Col = Order(1,AVG_Incl);
            AVG_Array(:,AVG_Incl)=YData(:,Col);
        end

        AVG_Mean = nanmean(AVG_Array.');
        AVG_Run_Array(:,AVG_Run)=AVG_Mean.';

    end

    MinArray = zeros(NumRows,1);
    MaxArray = zeros(NumRows,1);
    SpreadArray = zeros(NumRows,1);

    for ArrayStep = 1:NumRows
        MinArray(ArrayStep,1) = min(AVG_Run_Array(ArrayStep,:));
        MaxArray(ArrayStep,1) = max(AVG_Run_Array(ArrayStep,:));
        SpreadArray(ArrayStep,1) = MaxArray(ArrayStep,1) - MinArray(ArrayStep,1);
    end



        %$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
        %$$$$$$$$$$$$$$$$$ PLOT $$$$$$$$$$$$$$$$$$$
        %$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

%             StatsFig = figure('Name','Sensitivity of Data','Number','off');
        if Create_Figure == 0
            if isempty(StatsFig)
                if ishandle(StatsFig)
                    close(StatsFig)
                    StatsFig = figure('Name','Sensitivity of Data','Number','off');
                    % Leave window open
                else
                    % If does not exist, create new window
                    StatsFig = figure('Name','Sensitivity of Data','Number','off');
                end
            end
        else
                    StatsFig = figure('Name','Sensitivity of Data','Number','off');
        end

        ResAverage = nanmean(YData.');

    %=================================================================%
    %   Min and Max of Averages
    %=================================================================%
        subplot(3,1,[1,2])  % Plot takes up two spaces (makes larger)
%             semilogx(10, 0)       % Initiate Plot

            semilogx(XData, MaxArray, 'r')                      % Red

            hold                    % Hold Plot

            semilogx(XData, MinArray, 'Color', [0.2 0.6 0.0])    % Green
            semilogx(XData,ResAverage,'.k')
            hold                    % Release Plot

            ax_PlotAxesH1 = gca; %Collect info on current axes values to gca
            set(ax_PlotAxesH1,...
                'FontSize', 10); % Setting the x-labels to non-scientific
            legend('Max. of Averages','Min. of Averages','Average Used')
            %grid
            axis([10 20000 -5 25])

        %=================================================================%
        %   Adding Run Information to Plot
        %=================================================================%

            TrialsText = uicontrol('style','text', 'string','Total Number of Trials:',...
                'units','normalized','position',[0.15 0.85 0.25 0.05],...
                'BackgroundColor',[1 1 1],'HorizontalAlignment','left');
            TrialsValText = uicontrol('style','text', 'string',Number_of_Trials,...
                'units','normalized','position',[0.4 0.85 0.15 0.05],...
                'BackgroundColor',[1 1 1],'HorizontalAlignment','left');

            AVGText = uicontrol('style','text', 'string','Trials Used for Average:',...
                'units','normalized','position',[0.15 0.8125 0.25 0.05],...
                'BackgroundColor',[1 1 1],'HorizontalAlignment','left');
            AVGValText = uicontrol('style','text', 'string',Trials_to_AVG,...
                'units','normalized','position',[0.4 0.8125 0.15 0.05],...
                'BackgroundColor',[1 1 1],'HorizontalAlignment','left');

            RunsText = uicontrol('style','text', 'string','Number of Averages Run:',...
                'units','normalized','position',[0.15 0.775 0.25 0.05],...
                'BackgroundColor',[1 1 1],'HorizontalAlignment','left');
            RunsValText = uicontrol('style','text', 'string',Num_Runs,...
                'units','normalized','position',[0.4 0.775 0.15 0.05],...
                'BackgroundColor',[1 1 1],'HorizontalAlignment','left');

    %=================================================================%
    %   Data Spread Plot
    %=================================================================%
```

```
        subplot(3,1,3)
            % Hiding frequencies with minimal data (50 Hz and 16 kHz)
%                 SpreadArray(5,1) = NaN;
%                 SpreadArray(30,1) = NaN;

            % Plotting the difference between the two averages
               plot2 = semilogx(XData, SpreadArray, 'k');

            ax_PlotAxesH2 = gca; %Collect info on current axes values to gca
            set(ax_PlotAxesH2,...
                'FontSize', 10); % Setting the x-labels to non-scientific
            %grid
            axis([10 20000 0 15])

            MaxVal = max(SpreadArray);
            legend('Delta Between Max and Min','Location','north')

        %====================================================================%
        %   Adding Minimum and Maximum Data to Plot
        %====================================================================%
            MinMax = [min(SpreadArray) max(SpreadArray)];
            MaxText = uicontrol('style','text', 'string','Max:',...
                'units','normalized','position',[0.15 0.25 0.05 0.05],...
                'BackgroundColor',[1 1 1]);
            MaxValText = uicontrol('style','text', 'string',round(MinMax(1,2)*10)/10,...
                'units','normalized','position',[0.2 0.25 0.05 0.05],...
                'BackgroundColor',[1 1 1]);

            MinText = uicontrol('style','text', 'string','Min:',...
                'units','normalized','position',[0.15 0.2125 0.05 0.05],...
                'BackgroundColor',[1 1 1]);
            MinValText = uicontrol('style','text', 'string',round(MinMax(1,1)*10)/10,...
                'units','normalized','position',[0.2 0.2125 0.05 0.05],...
                'BackgroundColor',[1 1 1]);

    %================================================================%
    %   Send Data Back to Matlab
    %================================================================%

        DataShow = [XData.' SpreadArray]
        MinMax = [min(SpreadArray) max(SpreadArray)]


else
        fprintf('XData size = %d \nYData size = %d\n',XData_Check, NumRows);
        fprintf('Row length of XData and YData must match.\n')
end
end
```

With 52 data points in the Resultant array, the following results were generated for the removal of 10 data sets which were randomly selected and averaged. This process was repeated 100,000 times in order to capture all possible average combinations.

D.2 MATLAB Command Window

```
>> [XData, MaxArray, ResAverage, MinArray, SpreadArray] = AverageCheck(Freq,
ResultantData, 10, 100000, 1)
```

D.3 MATLAB Results

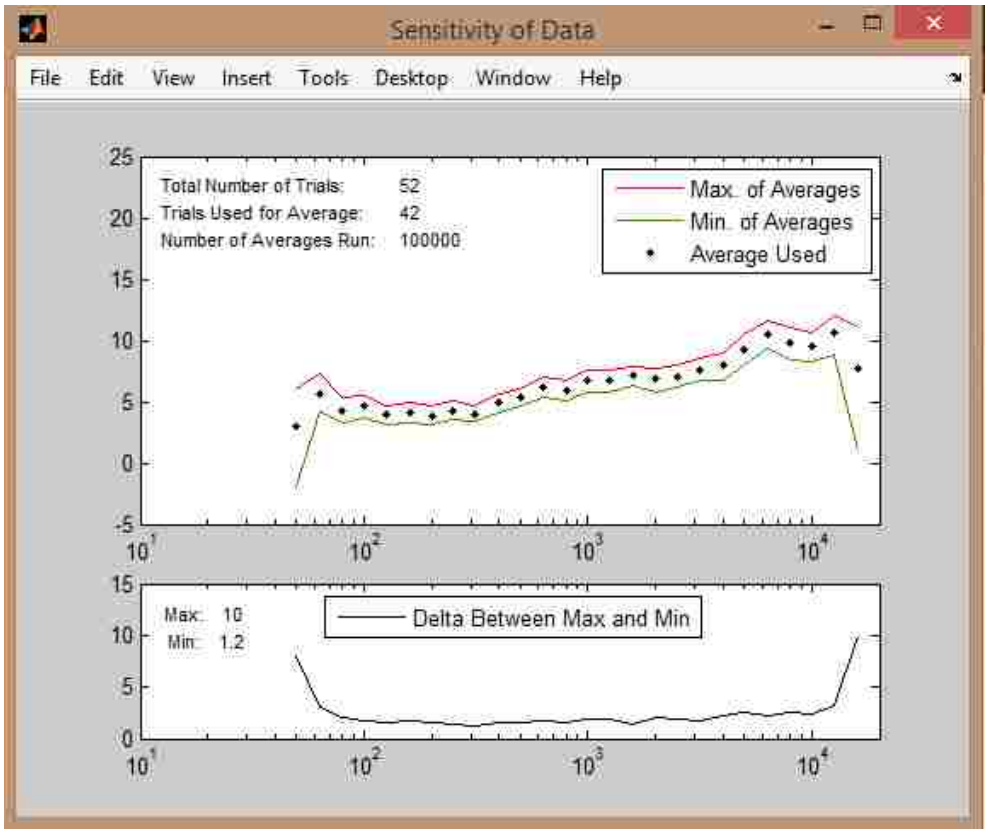


Figure D.1 – Sensitivity of Results

Table D.1 – Spread of Averaged Results

Freq.	Delta	Freq.	Delta
20	NaN	800	1.61
25	NaN	1000	1.87
31.5	NaN	1250	1.85
40	NaN	1600	1.48
50	8.00	2000	2.02
63	3.04	2500	1.83
80	2.10	3150	1.80
100	1.81	4000	2.23
125	1.55	5000	2.59
160	1.77	6300	2.23
200	1.63	8000	2.63
250	1.48	10000	2.33
315	1.22	12500	3.19
400	1.50	16000	10.00
500	1.50	20000	NaN
630	1.77		

D.4 Review of Sensitivity Check

When reviewing the resulting plots it is important to note that for the average calculation two frequencies will be removed from the data set; 50 Hz and 16 kHz. These two rows were removed as considerably less data was available at these frequencies due to testing and participant threshold limitations. In order to produce a 50 Hz signal at the desired sensation level a considerable amount of energy is required for the headphones. However, to reduce damaging the equipment limitations were set on the signal generator which prevented elevated sound pressure levels in the low frequency regions. As a result the desired sensation levels were unachievable for many of the participants who exhibited greater than normal hearing thresholds in the low frequency regions. For frequencies

greater than 12.5 kHz it was observed that only a small percentage of the population sampled could perceive a 16 kHz tone or a 20 kHz tone within the limitations set on the model. For this reason these frequencies were often not investigated and can reasonably be removed.

With the above limitations in place and the data limited between 63 Hz and 12.5 kHz the resulting average varies between 1.2 dB and 3.2 dB after removing 10 random data sets over 100,000 evaluations. The maximum values were observed at the upper and lower ends of the investigated spectrum and represent areas of hearing where audible matching is difficult for most participants. These results confirm that had the study only included 42 participants, the results would have been within reasonable uncertainty levels. The fact that the study was expanded to include 50 participants improved upon these results considerably and confirms that additional testing in this instance is unnecessary. As the sensitivity check is expanded to removing 15 data sets, the resulting averages vary between 1.7 dB and 4.1 dB indicating a greater uncertainty level for a participant pool size of 37; in this instance the largest spread was again located at the extreme frequency limits of the test.

VITA AUCTORIS

NAME:	Jeremy E. Charbonneau
PLACE OF BIRTH:	Chatham, Ontario
DATE OF BIRTH:	1985
EDUCATION:	Blenheim District High School, Blenheim, Ontario 1999-2003
	University of Windsor, Windsor, Ontario 2003-2008, B.A.Sc.
	University of Windsor, Windsor, Ontario 2008-2010, M.A.Sc.
	University of Windsor, Windsor, Ontario 2012-2015, Ph.D. Candidate

PUBLICATIONS:

Charbonneau, Jeremy; Novak, Colin; Gaspar, Robert; and Ule, Helen. (2012) Assessment of head and torso simulators using temporal loudness calculations. Inter-Noise 2012 New York, USA (August 19-22, 2012).

Charbonneau, Jeremy; Novak, Colin; Gaspar, Robert; and Ule, Helen. (2012) A-weighting the equal loudness contours. Journal of the Acoustical Society of America. Volume 131, Issue 4: 3502-3503, (MASc Work).

Novak, Colin; Ule, Helen; Charbonneau, Jeremy; and Letowski, Tomasz, (2012) Comparative study of unsteady loudness models for mechanical and real sounds. Journal of the Acoustical Society of America. Volume 131, Issue 4: 3503-3503, (PhD Work).

Charbonneau, Jeremy; Novak, Colin; and Ule, Helen. "Comparison of loudness calculation procedure results to equal loudness contours." Inter-noise 2009, Ottawa, Canada. August, 2009.

Charbonneau, Jeremy; Novak, Colin; and Ule, Helen. "Loudness Prediction Model Comparison Using The Equal Loudness Contours." CAA 2009, Niagara-on-the-Lake, Ontario. October, 2009.

Novak, Colin; Charbonneau, Jeremy; and Ule, Helen. "Comparison of Non-Stationary Loudness Results to Equal Loudness Contours." CAA 2009, Niagara-on-the-Lake, Ontario. October, 2009.