SCALABLE PERCEPTUAL METRIC FOR EVALUATING AUDIO IMPAIRMENT

BY

RAHUL VANAM, B.E.

A thesis submitted to the Graduate School

in partial fulfillment of the requirements

for the degree

Master of Science in Electrical Engineering

New Mexico State University

Las Cruces, New Mexico

June 2005

"Scalable Perceptual Metric for Evaluating Audio Impairment," a thesis prepared by Rahul Vanam in partial fulfillment of the requirements for the degree, Master of Science in Electrical Engineering, has been approved and accepted by the following:

_____

Linda Lacey
Dean of the Graduate School

_____

Charles D. Creusere
Chair of the Examination Committee

_____

Date

Committee in charge:

Dr. Charles D. Creusere, Chair

Dr. Raphael J. Lyman

Dr. Phillip De Leon

Dr. L. Allen Torell

DEDICATION

Dedicated to my loving parents.

ACKNOWLEDGEMENTS

I would sincerely like to thank my advisor Dr. Charles D. Creusere for giving me an opportunity to work on an interesting problem and also for motivating me with research ideas presented in this thesis. Without his invaluable advice, help and suggestions, this thesis work would not have been possible. I would like to thank Dr. Raphael J. Lyman and Dr. Phillip De Leon for graciously agreeing to serve on my thesis committee and reviewing this work.

I thank Dr. Peter Kabal, McGill University for allowing me to include in this thesis portions of his PEAQ basic version source code that I have used in developing the PEAQ advanced version. I thank Dr. Stanley Kuo, Apple Computer Inc., and Sudeendra Maddur Gundurao, Socrates Software India Ltd., for the technical assistance I received from them during the development of the PEAQ advanced version.

I would like to thank Dr. Lyman, for giving me an opportunity to work with him in the Autoconfigurable Receiver project, which helped me finance my graduate studies. I am also grateful to him for giving me the flexibility of pursuing my thesis research in audio while being employed in his research project.

I would like to thank my friends who have been encouraging and supportive throughout my graduate studies.

Finally, I would like to express my sincere gratitude to my parents, whose constant support, understanding and encouragement have really brought me here.

VITA

June 5, 1978	Born at Bangalore, India
2000	Bachelor of Engineering (B.E), Electronics and Communication, Bangalore University, India.
2000-2003	Systems Engineer, Wipro Technologies, Bangalore, India.
2003-2005	Graduate Research Assistant, Klipsch School of Electrical and Computer Engineering, New Mexico State University, Las Cruces, New Mexico.

PUBLICATIONS

Rahul Vanam and Charles D. Creusere, "Evaluating low bitrate scalable audio quality using advanced version of PEAQ and Energy Equalization approach," *Proc. of IEEE International Conference in Audio, Speech and Signal Processing*, Vol.3, pp.189-192, Philadelphia, March 2005.

FIELD OF STUDY

Major Field: Electrical Engineering

Digital Signal Processing

ABSTRACT

SCALABLE PERCEPTUAL METRIC FOR EVALUATING AUDIO IMPAIRMENT

BY

RAHUL VANAM, B.E

Master of Science in Electrical Engineering

New Mexico State University

Las Cruces, New Mexico, 2005

Dr. Charles D. Creusere, Chair

ITU-R BS.1387-1 presents a method for objective measurement of perceived audio quality know as PEAQ (Perceptual Evaluation of Audio Quality). The PEAQ has been designed to perform optimally for evaluating high quality audio. In this thesis, we show that the PEAQ advanced version performs poorly for evaluating the quality of audio that is highly impaired when compared to the *Energy Equalization approach* (EEA). We also introduce in this work a new metric that uses six Model Output Variables (MOVs) – five MOVs from the original PEAQ advanced version and one MOV from the EEA. These MOVs are mapped to a single quality measure using an optimized single-layer neural network. This metric performs better than both EEA and PEAQ advanced version for measuring audio quality over a wide range

of impairment. Furthermore, by using the bitrate information of the encoded audio signal, the performance of the proposed approach is shown to further improve. The final result of our research is an audio quality metric capable of accurately predicting perceptual quality over a wide range of audio impairment – i.e., a scalable metric.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 INTRODUCTION

1.1 Thesis Motivation

Most audio coding algorithms have been able to achieve high compression using auditory models that approximates the characteristics of the human ear. Sound perception in the human ear varies with frequency and magnitude of the audio signal, and a compressed signal can often be made indistinguishable from the original signal by a normal listener. Therefore, such compressed audio signals cannot be evaluated for quality using methods such as signal to noise ratio or total harmonic distortion. Evaluation of such audio is done using subjective testing [1]. However, subjective tests tend to be time consuming, expensive, and difficult to reproduce. Therefore objective measurement methods have been developed such that their results closely match the subjective testing results. These methods require less evaluation time at the cost of accuracy when compared to subjective testing. The ITU initiated the standardization of objective measurement methods, combining the best features from the existing methods [2] – [7] to bring out an algorithm called the PEAQ (Perceptual Evaluation of Audio Quality). This algorithm has been put forward as Recommendation ITU-R BS.1387-1 [8]. PEAQ was optimized for measuring high quality audio and has two versions – basic and advanced – which tradeoff between accuracy and speed.

New generation of audio codecs such as those from the MPEG-4 (Motion Pictures Experts Group-4) family provides for scalability that allows portions of the encoded stream to be decoded to obtain audio signal with lower fidelity, bandwidth or selected content [1]. Thus, it allows encoding at higher bitrates and decoding at lower bitrates.

The performance of PEAQ basic version has been found to be poor for measuring low bitrate scalable audio quality. On the other hand, a method called the *Energy Equalization approach* (EEA) has been shown to perform well for measuring such high impaired audio [9]. The use of EEA parameter as an additional Model Output Variable (MOV) in the PEAQ basic version has been investigated in [10], and it has been shown to improve on the performance of PEAQ basic version when measuring high impairment audio quality. In this thesis, we examine the performance of the PEAQ advanced version for measuring quality of low and mid bitrate audio when compared to EEA. We show that the PEAQ advanced version performs poorly for these cases and that including EEA parameter as an additional MOV significantly improves its performance. Furthermore, the modified PEAQ advanced metric performs well for measuring different audio qualities.

1.2 Thesis Overview

In chapter two, the concepts of human auditory modelling are discussed as they apply to the perceptual measurement of audio quality. Chapter three introduces the different methods for objective measurement of audio quality. The basic and advanced versions of PEAQ are discussed in chapter four while chapter five presents the results from subjective testing and EEA. This chapter includes performance comparisons of the different objective measurement methods proposed in this thesis and also compares the proposed metric with a modified PEAQ basic version that includes the EEA MOV. Finally chapter 6 concludes with summary of results and possible future research in this area.

2 PSYCHOACOUSTIC CONCEPTS

2.1 Introduction

Perceptual measurement of audio quality requires a human auditory model that replicates the way in which sound is processed within the human ear and that also captures the true perception of a human listener. Perceptual models of the human ear have been popularly used in audio encoders to determine auditory thresholds that limit the allowed quantization noise to ensure no loss in audio quality. Such coding is also referred to as *perceptual transparent coding*. These psychoacoustic concepts can be extended to the analysis of audio quality.

2.2 Absolute Threshold of Hearing

The absolute threshold of hearing characterizes the amount of energy needed in a pure tone such that it can be detected by a listener in a noiseless environment [11]. A plot of the absolute threshold as a function of frequency is given in Fig 2.1. It is observed that the threshold is lowest in the region 2000-3000 Hz and increases as frequency decreases or increases from this region. This characteristic is modeled using outer and middle ear transfer function that band limit the input signal. Noise attributed to auditory nerve and blood flow is also added to the input signal [8].

2.3 Perceptual Frequency Scales

In the inner ear (cochlea), frequency-to-place transformation takes place along the basilar membrane. Depending upon the frequency of the input, large responses are generated at frequency-specific membrane positions. Therefore, each hair cell is effectively tuned to different frequency bands according to their location [11]. As a

result of frequency-to-place transformation, the cochlea can be viewed as a bank of highly overlapped bandpass filters with bandwidths increasing with frequency. The bandwidths of these cochlear filters are quantified by a function of frequency called the *critical bandwidth*. The frequency-to-place transform is nonlinear while the distribution of hair cells across the basilar membrane is linear. This leads to nonlinear frequency perception called *pitch*. In PEAQ, an approximation of the frequency-to-place transformation function called the *Bark* scale as defined by Zwicker and Feldkeller [12] is used. The Bark scale divides the frequency range from 20 Hz to 15 kHz into 24 non-overlapping frequency bands and is illustrated in Fig 2.2.

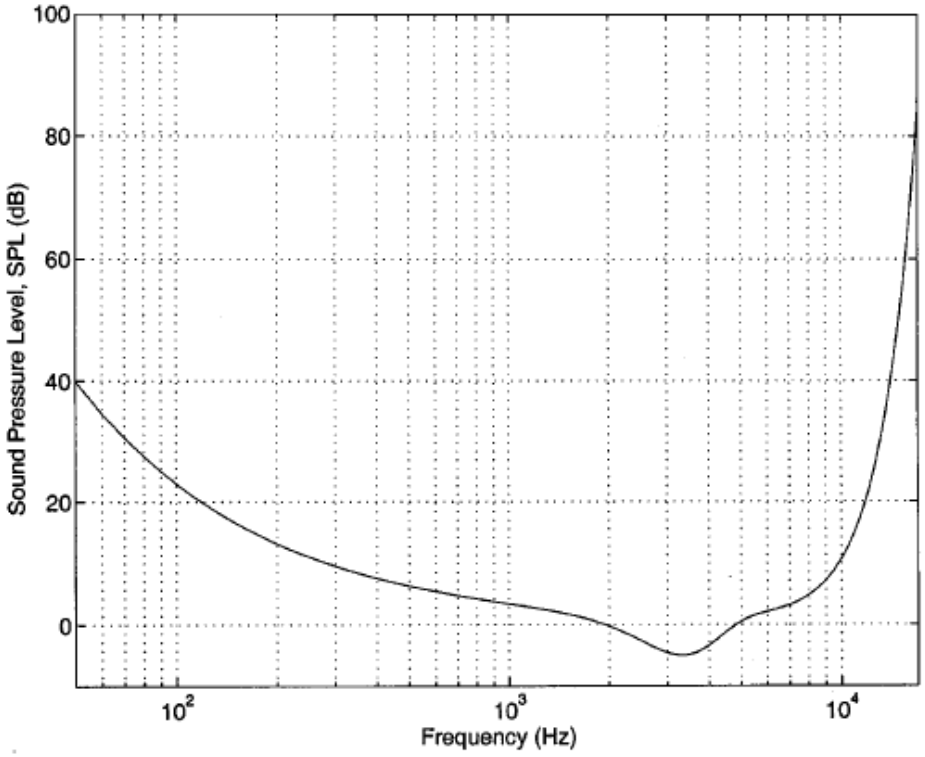


Figure 2.1. Absolute threshold of hearing [11]

2.4 Excitation

Each hair cell in the cochlea generates neural activity in response to a range of frequencies that can be described by a filter characteristic. For signals that are sinusoidal, the pattern of the neural excitation (measured in decibels) is approximately triangular in shape, and this shape is independent of the center frequency in the Bark scale [13]. The part of the excitation pattern corresponding to the positive slope is largely independent of the level of the exciting signal (about 27 dB/Bark) while the negative slope depends on the absolute sound pressure level of the signal. At higher levels, the negative slope is about -5 dB/Bark while the slope at lower levels can be a steeper -30 dB/Bark. The neural excitation patterns for different levels L of the input signal is shown in Fig 2.3. The neural excitation at a particular position on the basilar membrane results from the outputs of several overlapping filters. When these outputs are added, the resulting excitation pattern reproduces the asymmetric level dependent slopes of typical masking functions [13]. Once the input signal is terminated, the hair cells and neural processing stages need time to recover to their maximum sensitivities. The recovery time depends upon the level and duration of the signal and can be more than 100 ms. The processing of high amplitude signals is faster than processing of low amplitude signals, and therefore a strong signal can out run a weaker signal on the way between hair-cell and brain. Hence, the onset of a loud signal can mask a preceding soft signal [8].

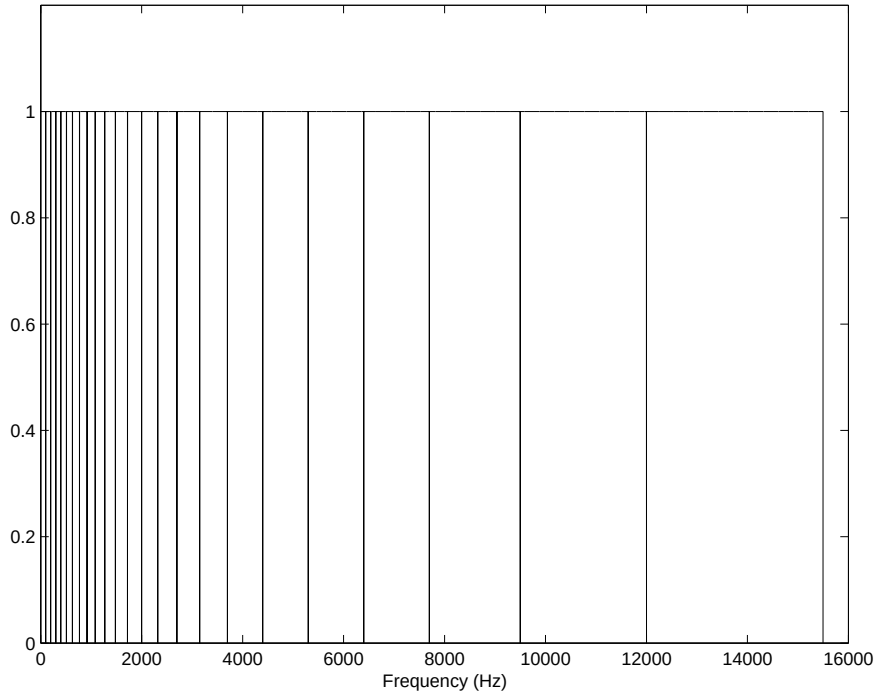


Figure 2.2. Critical bands defined by Zwicker [11]

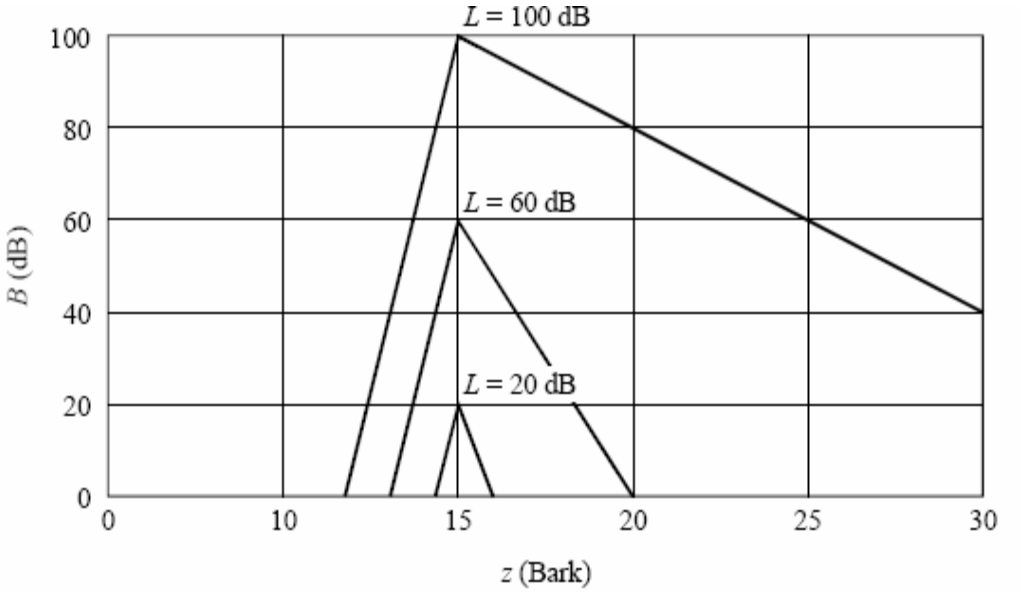


Figure 2.3. Neural excitation pattern for different levels of the input signal [8]

2.5 Discrimination between Signal Levels

A human listener has three kinds of acoustic related memory that differ by the degree of detail and the duration over which the information is present: long-term memory, short-term memory and ultra-short term memory (also known as echoic memory) [13]. Subjective testing methods such as the ITU-R BS.1116 primarily depends upon short-term and ultra-short term memory to detect difference between signals. The signal level at which probability is 50% for detecting a change in level difference is referred to as *just noticeable level difference* (JNLD). JNLD depends upon the level of the input signal. The JNLD is large for weaker signals while it is small for loud signals. For example, the JNLD is about 0.75 dB at a level of 20 dB, and 0.2 dB at a level of 80 dB [13].

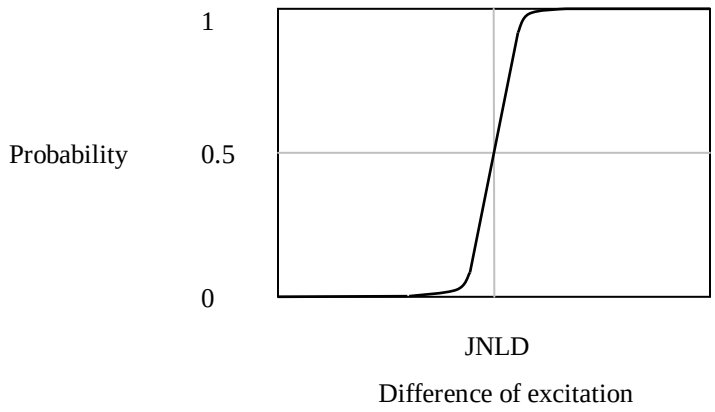


Figure 2.4. Principle of detection probability [8]

2.6 Masking

An audible signal is rendered inaudible when presented together with a perceptually louder signal. The masked signal is called the *maskee* and the masking

signal is called the *masker*. Masking can be classified into two categories: *simultaneous masking* and *non-simultaneous* or *temporal masking*.

2.6.1 Simultaneous Masking

In this scenario, the masker and maskee are present at the same time and are quasi-stationary [8]. A masker will create an excitation of sufficient strength on the basilar membrane at the critical band location to effectively block the detection of the maskee. The presence of a discrete-bandwidth masker will result in an increase in the detection threshold for frequencies above and below the masker. Simultaneous masking depends upon the structure of the masker and maskee and can be classified as *noise-masking-tone* (NMT), *tone-making-noise* (TMN), *tone-masking-tone* (TNT) and *noise-masking-noise* (NMN) [11]. In NMT, amount of masking is frequency independent. If the tone signal is 5 dB below the level of the noise signal, it becomes inaudible. For TMN, masking depends upon the frequency of the tone and is estimated by the formula (15.5 + z/Bark) dB, where z is the critical band of the masker. In addition, at high signal levels non-linear effects reduce the masked threshold near the masker [8]. This effect is also observed in TNT. Resulting masking threshold is obtained by adding the masked thresholds due to individual signals in a non-linear manner.

2.6.2 Temporal Masking

Temporal masking may occur when masker and maskee are present at different times. Shortly after the termination of the masking signal, the masked threshold is closer to the simultaneous masking threshold than to the absolute hearing threshold – i.e. it remains elevated. The decay time of the masking threshold depends upon the duration of

the masker. The decay time varies between 5 ms to more than 150 ms. This effect is called *forward* or *post masking* [13].

Weak signals that occur prior to the occurrence of a strong signal can be masked as well, and this effect is called *backward* or *pre masking.* The duration of backward masking is about 5 ms. A maskee just above the pre mask threshold is not perceived before the masker but as a change of the masker. Backward masking exhibits large deviation from listener to listener [8]. Forward and backward masking curves are illustrated in Fig 2.5.

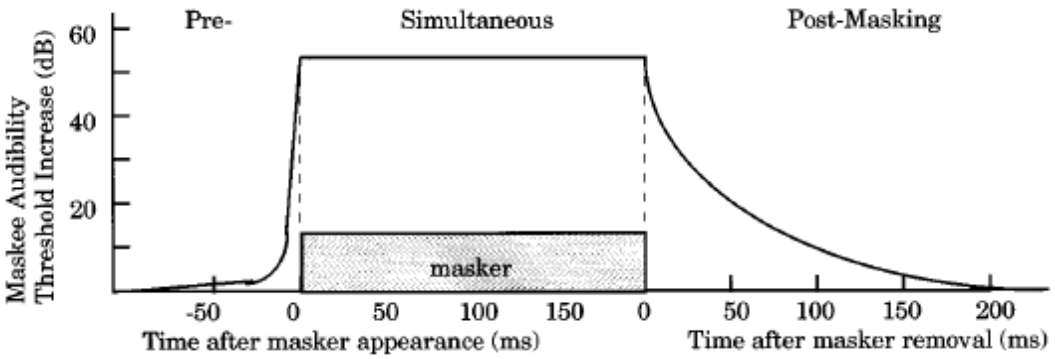


Figure 2.5. Temporal masking properties of the human ear [11].

2.7 Loudness and Partial Loudness

The perceived loudness of an audio signal depends upon its sound pressure level and duration as well as its temporal and spectral structure. The partial loudness represents the perceived loudness after the signal has been masked by a masker. In audio quality measurement, partial loudness takes into account the reduction in perceived loudness of an audible distortion due to the presence of a masker [13].

3 PERCEPTUAL MEASUREMENT CONCEPTS

3.1 Introduction

Auditory models process signals in two main stages, namely the *peripheral ear model* which translates sound field to neural excitations and the *cognitive model* that processes the neural excitations to sensations. Figure 3.1 illustrates the different stages involved in auditory processing.

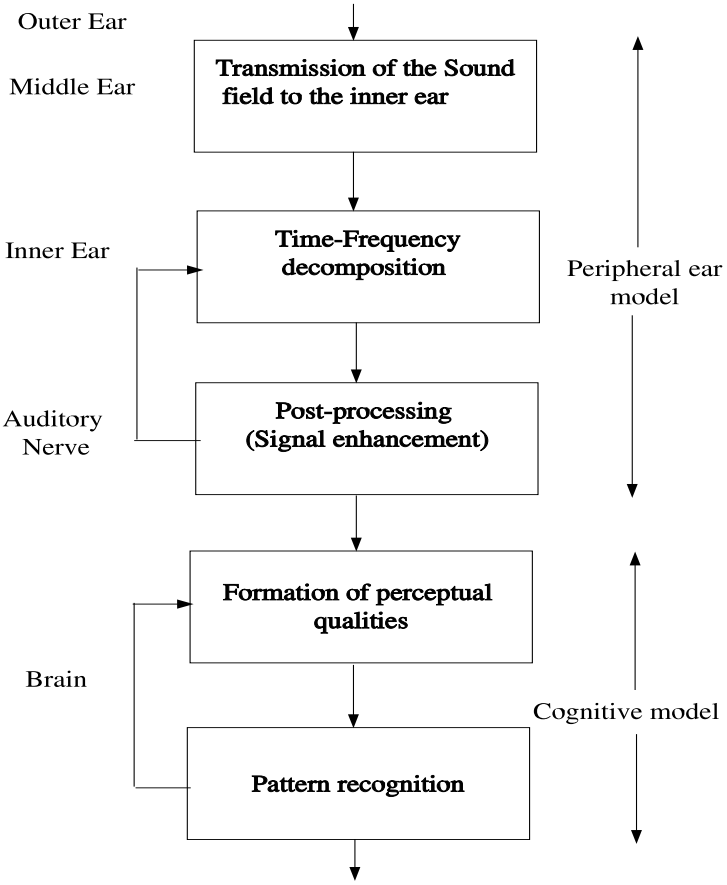


Figure 3.1. Stages of auditory processing [14]

The peripheral ear model is mainly based on the physiological structure of the auditory system and is fairly consistent for different individuals. The cognitive model

includes pattern recognition and auditory streaming, where individual biases cannot be clearly separated from the fundamental properties of auditory processing system. There are two approaches for modelling auditory processing, namely *functional models* and *heuristic models*. The functional models are based on acoustical measurements, mechanical properties of the ear, neural excitation measurements in the inner ear and measurements of neural activity in the brain [14]. The heuristic models are entirely based on listening test results.

Two main concepts for estimating perceived audio distortion are used, namely the masked threshold concept and the comparison of internal representations concept. The former is closer to the heuristic model while the later is closer to the functional model.

3.2 Masked Threshold Concept

In this approach, an error signal obtained from the difference between the original and processed signals is compared to the masked threshold of the original signal. The error is considered inaudible if it has a magnitude less than the masked threshold either in frequency or temporal domain. This concept is used in both NMR and noise loudness calculation and is illustrated in Fig 3.2

3.3 Comparison of Internal Representations

This concept was first introduced by Karjalainen in 1985 [15]. It involves the use of an auditory model to convert an input audio signal into the excitation pattern generated in the inner ear. Quality measurements are done by comparing excitation patterns of the original and processed signal. This approach is illustrated in Fig 3.3.

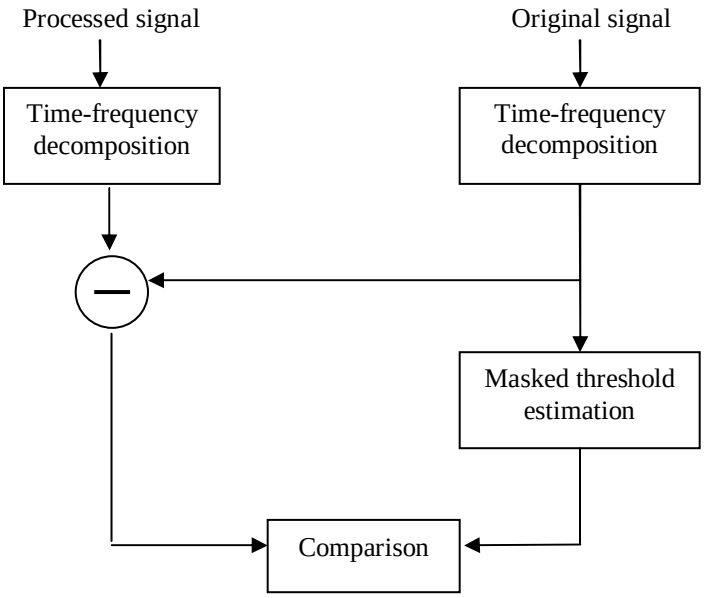


Figure 3.2. Masked threshold concept [14]

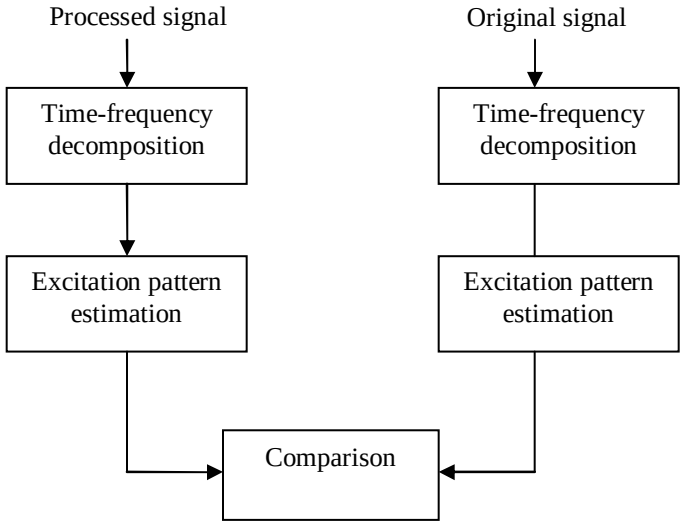


Figure 3.3. Comparison of internal representations [14]

3.4 Spectral Analysis of Errors

The spectral analysis of errors approach is more heuristic. Here, effects such as perception of fundamental frequency are more easily modelled using the linear spectra rather than the basilar membrane excitation patterns. It should be noted that this approach can be used as a supplement to the other two concepts as it provides additional information about the character of the distortions that are difficult to obtain in the time domain [14].

4 PERCEPTUAL EVALUATION OF AUDIO QUALITY (PEAQ)

4.1 Introduction

Several different objective measurement methods have been developed over the years to overcome the need for subjective testing in evaluating audio quality. Since the different methods have varying performances, the ITU initiated standardization efforts in 1994. It considered six methods and performed comparative tests to develop one of the methods as the final recommendation. These six methods included:

1. Noise to Mask Ratio (NMR).
2. Perceptual Audio Quality Measure (PAQM).
3. Perceptual Evaluation (PERCEVAL).
4. Perceptual objective measurement (POM).
5. Distortion index (DIX).
6. Toolbox.

The testing indicated that none of the methods were reliable enough to be considered as a standard method. This therefore necessitated collaboration among different model proponents to jointly develop a model that would include the best features of each method. This joint method was named PEAQ (Perceptual Evaluation of Audio Quality) and was brought out as recommendation by the ITU, i.e. ITU-R BS.1387-1 [8].

The PEAQ has different stages namely the peripheral ear model, preprocessing of excitation patterns, calculation of MOVs and a mapping from a set of MOVs to a single value representing basic audio quality of the test signal. It is illustrated in Fig 4.1. This

objective measurement method includes an FFT-based ear model as well as an ear model based on a filter bank. The MOVs are based on the masked threshold that is originated from NMR and comparison of internal representations. In addition, one MOV is instead derived from the comparison of linear spectra and not by an ear model [13]. Mapping of MOVs to a basic audio quality metric is done using an artificial neural network with one hidden layer.

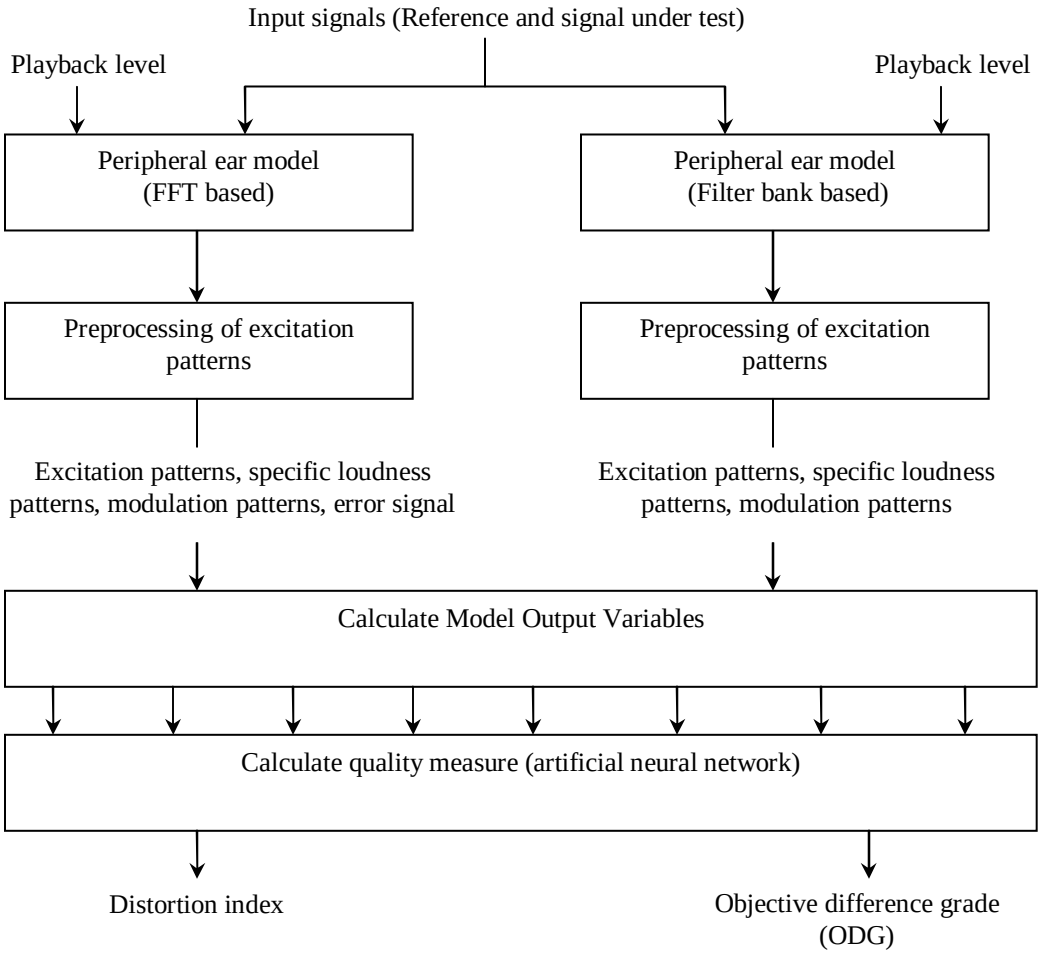


Figure 4.1. Generic block diagram of PEAQ [8]

The model consists of two versions that tradeoff between accuracy and speed. The advanced version provides highest possible accuracy with higher computational complexity while the basic version provides lower accuracy at low computational cost. The basic version uses 11 MOVs while the advanced version uses five MOVs. Although advanced version uses fewer MOVs, it is able to capture the same properties as the MOVs used by the basic version [13].

4.2 Basic Version

The basic version of the PEAQ includes only the MOVs calculated from the FFT-based ear model. It makes use of concepts such as masked threshold and comparison of internal representations. The poor temporal resolution of the FFT-based ear model is overcome by using more MOVs and increasing the spectral resolution (when compared to the advanced version) [13]. This model generates 11 MOVs that measure the changes in temporal envelope (i.e. modulation measure), noise loudness, linear distortion, relative frequency of audible distortions, noise-to-mask ratio, probability of detection of noise and harmonic structure of the error.

4.3 Advanced Version

The advanced version of the PEAQ includes MOVs that are calculated from both the filter bank-based ear model as well as the FFT-based ear model. It makes use of the masked threshold concept using the FFT-based ear model and comparison of internal representations using the filter bank-based ear model. The model generates five MOVs; those derived from the filter bank model measure the loudness of nonlinear distortions, the amount of linear distortions, and the disruption of the temporal envelope. The MOVs

derived from the FFT ear model measure noise-to-mask ratio and harmonic structure of the error [13].

4.4 Peripheral Ear Model

4.4.1. FFT-based Ear Model

The FFT based ear model was primarily developed by the *Fraunhofer Institute for Integrated Circuits* (FhG-IIS). This model takes 48 kHz time aligned reference and test signals that are divided into frames of about 0.042s (i.e. 2048 samples) with 50% overlap. Each frame is transformed to the frequency domain using a Hann window and a short-time FFT (ST-FFT), and rectified and scaled to the playback level. The spectral coefficients are then scaled by a weighting function which models the outer and middle ear frequency response [8]. The weighted spectral coefficients are then grouped into critical bands for transformation to the pitch representation. A frequency dependent offset is added to model internal noise in the auditory system. A level dependent spreading function followed by time domain spreading is applied to the spectral coefficients to model spectral auditory filtering and forward masking, respectively. The *excitation patterns* thus obtained are used to compute *specific loudness patterns* and m*ask patterns.* The *unsmeared excitation patterns* that are obtained before the final time domain spreading are used to calculate *modulation patterns.* The output of the outer and middle ear filters corresponding to both reference and test signal patterns are combined and grouped into critical bands to model the error signal. The Model Output Variables are calculated using the output of the outer and middle ear together with the excitation patterns. The block diagram illustrating the above steps is shown in Fig 4.2.

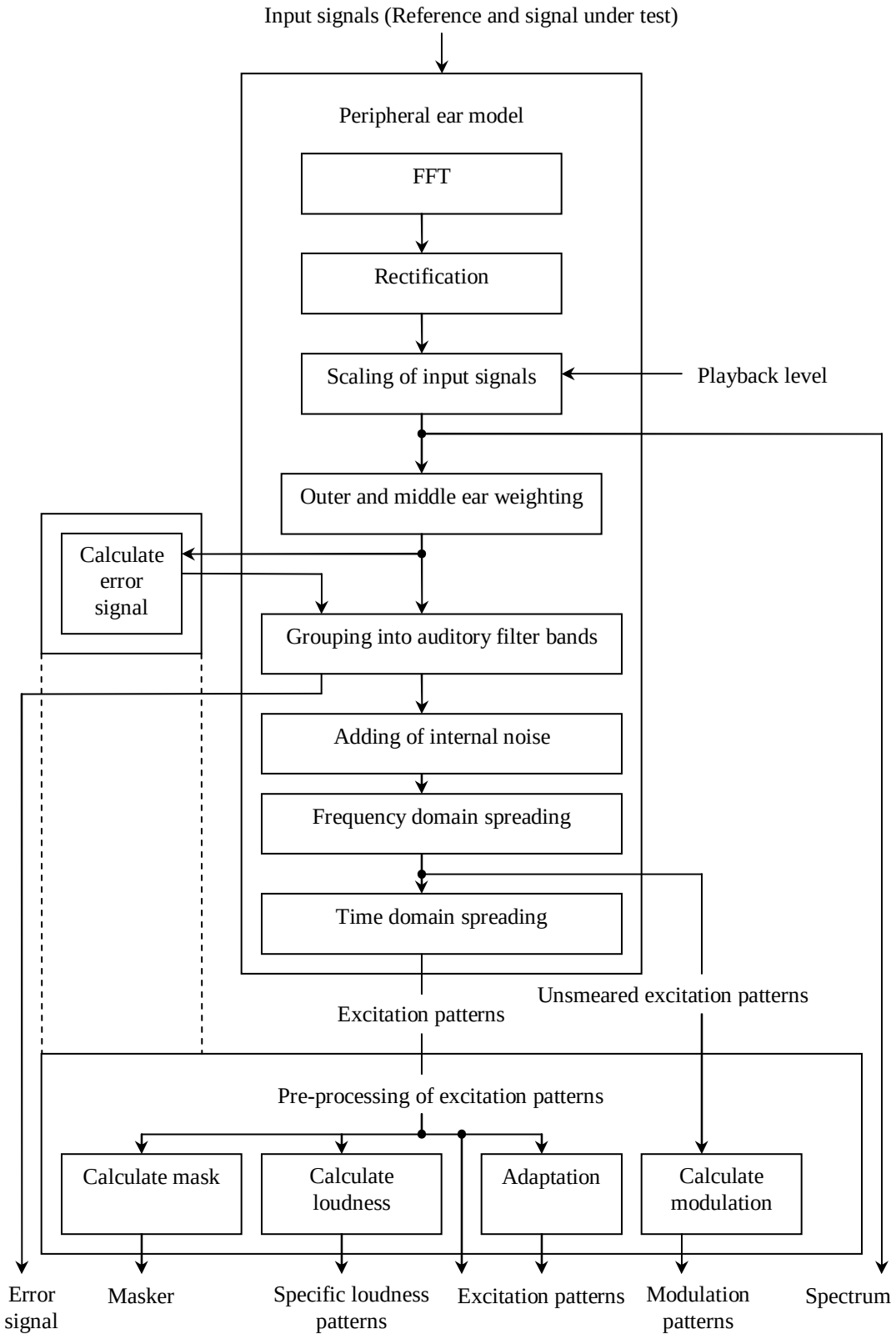


Figure 4.2. FFT-based ear model and preprocessing of excitation patterns [8]

4.4.2 Filter Bank-based Ear Model

The filter bank based peripheral ear model was primarily developed by Thiede and Kabot for their DIX metric [7]. The filter bank based ear model is illustrated in Fig 4.3 and it differs from the FFT-based ear model discussed previously in many ways. The input signal is first scaled by a playback level before rectification for reason of computational efficiency, as opposed to scaling the rectified signal as is done in the FFT-based model. Since the filter bank is sensitive to subsonics in the input signal, a DC rejection filter is applied to the scaled input signal. The filter used is a fourth order Butterworth high pass filter having a cutoff frequency of 20 Hz.

The signal is then decomposed into bandpass signals using a filter bank with 40 pairs of linear filters. Each filter pair consists of two filters having equal frequency response and their phase response differs by 90^{o}. Hence one of the filter pair represents the real part of the filtered signal and other filter represents the imaginary part. The filter bank is non-maximally decimated since there are 40 filter pairs and the output of each filter is decimated by a factor of 32. When related to the auditory pitch scale (i.e. critical band scale), the filters are equally spaced and have constant absolute bandwidth. Figure 4.4 gives the magnitude plot of the real part of the filter bank in frequency scale.

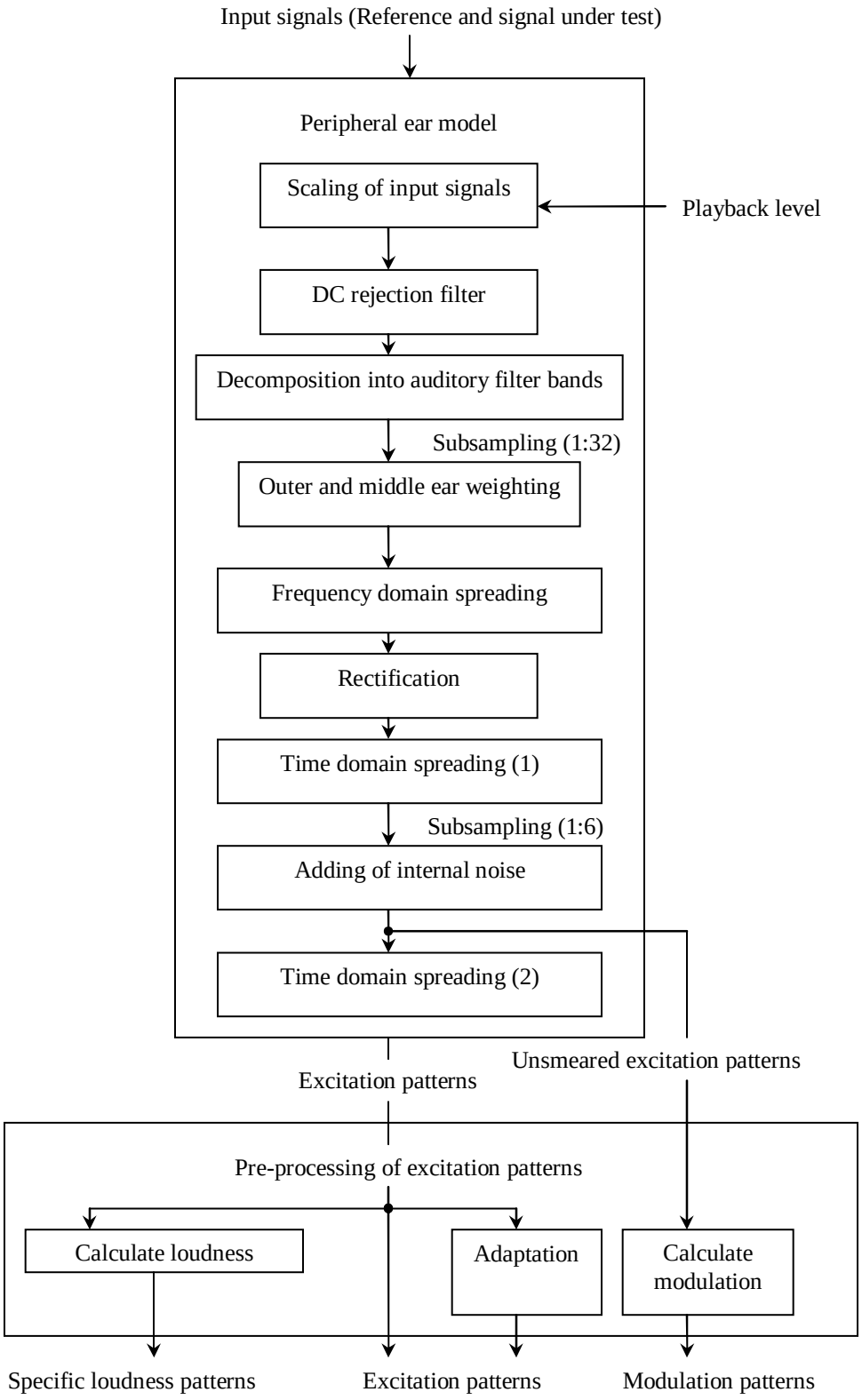


Figure 4.3. Filter bank-based ear model and preprocessing of excitation patterns [8]

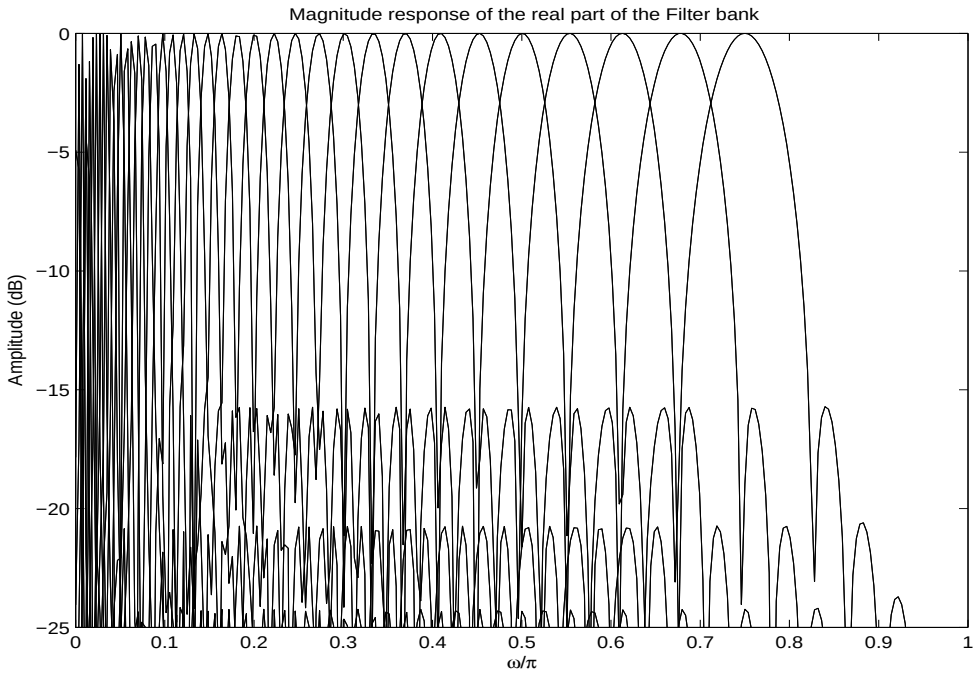


Figure 4.4. Magnitude response of the real part of the filter bank

The impulse response of the k[th] real and imaginary filter bank is given by

$$\left.\begin{aligned} h_{re}(k,n) &= \frac{4}{N[k]}.\sin^2\left(\pi.\frac{n}{N[k]}\right).\cos\left(2\pi.f_c[k].\left(n-\frac{N[k]}{2}\right).T\right) \\ h_{im}(k,n) &= \frac{4}{N[k]}.\sin^2\left(\pi.\frac{n}{N[k]}\right).\cos\left(2\pi.f_c[k].\left(n-\frac{N[k]}{2}\right).T\right) \end{aligned}\right| \; 0 \le n < N[k]$$

$$h_{re}(k,n) = h_{im}(k,n) = 0 \;\Big|\; \begin{matrix} n < 0 \\ n \ge N[k] \end{matrix} \qquad (4.1)$$

where N[k] is the length of the impulse response, $f_c[k]$ is the center frequency and T is the sampling period. The center frequencies of the filters range from 50 Hz to 18000 Hz. Each filter input is delayed by samples equal to half the difference between the length of its impulse response and the length of the filter with the longest impulse response.

A frequency dependant weighting function that models the response of the outer and middle ear is applied to the output of the filter bank. This weighting function is uniform across each subband. The frequency response of the outer and middle ear

weighting function is shown in Fig 4.5. It is observed that the response is $-\infty$ at 0 Hz and has a peak value of about 5.59 dB near 3.3 kHz. The FFT-based and filter bank based ear models use similar weighting function for outer and middle ear frequency response. However, the weights are computed for each bark band in filter bank based ear model and each frequency line for the FFT-based ear model.

Unlike the FFT-based ear model, frequency domain spreading in the filter bank model is done prior to rectification. This preserves the spectral and temporal characteristics of the filter bank [13]. The spreading function is a two sided exponential with the lower slope fixed at 31 dB/Bark and the upper slope varying between -24 and -4 dB/Bark.

After frequency domain spreading, the energy of the filter output is computed by adding the squared values of the real and imaginary part of the signal. The signal energy is then subjected to backward masking using an FIR filter having a raised squared cosine shaped impulse response. The filter has a length of 8 ms which corresponds to a backward masking duration of approximately 2 ms [13]. The smeared signal is then downsampled by a factor of 6 to improve the computation speed of the algorithm. A frequency dependent offset representing internal noise is added to the energies of each filter channel. The signal is now subjected to forward masking using a first-order IIR lowpass filter and the resulting signal is referred to as an *excitation pattern*. The time constant of this filter depends upon the center frequency of the corresponding auditory filter.

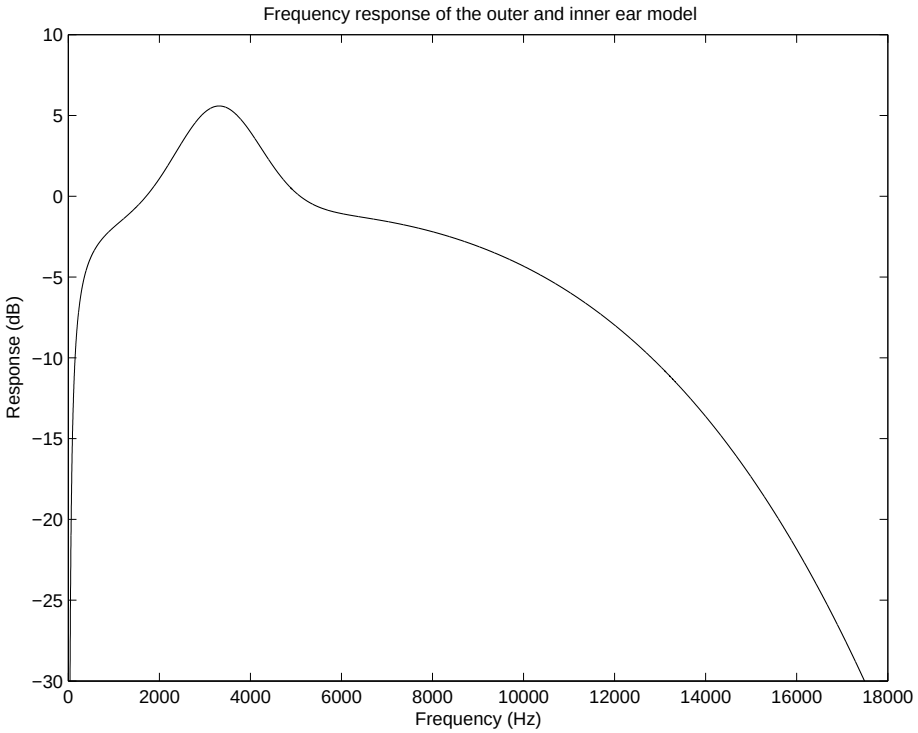


Figure 4.5. Frequency response of the outer and middle ear model

4.5 Pre-processing of Excitation Patterns

Pre-processing of excitation patterns is common to both the peripheral ear models. Objective differences such as signal delays, constant amplification or attenuation between reference and processed signal are not perceived as errors. Therefore, it is necessary to compensate for delays and level differences in the two signals prior to processing with the perceptual model [13]. Both the signals are subjected to *level* and *pattern adaptation*. In level adaptation, the source of the error is attributed to the band limitation of the processed signal. The level adaptation factor is determined from the energy ratio of the signal components within a given passband.

Linear distortion such as changes in the spectral envelope due to non-uniform frequency response of the device under test is less annoying than other audible distortions. Compensation for this distortion is achieved by pattern adaptation, where the

spectral envelopes of the original and processed signals are adapted to each other. Spectrally adapted patterns are thus obtained after the signal is processed by level and pattern adaptation.

A modulation measure is computed that takes into account the structure of the temporal envelopes, and it is calculated from the temporal derivative of the signal envelopes at each filter channel [13]. From the modulation measure of the reference and processed signals, the difference in the modulation of the temporal envelopes is computed. The overall *loudness* of the both the signals is then computed. The partial loudness measure is calculated from the specific loudness patterns, and it provides a measure of the effect of the additive non-linear distortions [13]. In the case of the FFT-based ear model, the difference between outer and middle ear magnitude spectra of the reference and test signal is also computed.

4.6 Calculation of the Model Output Variables

The Model Output Variables (MOVs) computed in the advanced version of the PEAQ are listed in Table 4.1. Using an artificial neural network having five nodes in the hidden layer, the MOVs are mapped to a single value called *distortion index* that is related to perceived basic audio quality called the *objective difference grade* (ODG). The artificial neural network in PEAQ has been designed to generate ODG values that are close to the *subjective difference grade* (SDG) values obtained from subjective test using ITU-R BS.1116 for measuring high quality audio [8]. The SDG is calculated from the difference in subjective grades for processed and reference signal. The SDG values range from 0 to -4 as shown in Fig 4.6, where 0 indicates that the impairment in the processed

signal is imperceptible while -4 indicates that the impairment in the processed signal is very annoying. The scale for ODG values in PEAQ is the same as that for SDG values.

Table 4.1 MOVs used in PEAQ advanced version [8]

MOV	Ear model	Description
$RmsModDiff_A$	Filter bank	It is squared average of the modulation difference.
$RmsNoiseLoudAsym_A$	Filter bank	It is the weighted sum of the squared averages of the noise loudness and the loudness of the lost signal components.
$AvgLinDist_A$	Filter bank	It gives a measure of the loudness of the signal components lost during spectral adaptation of the reference and the processed signal.
Segmental NMR_B	FFT	It is the average of the local noise to mask ratio.
EHS_B	FFT	It is the measure of the largest peak in the autocorrelation spectrum of the error energy.

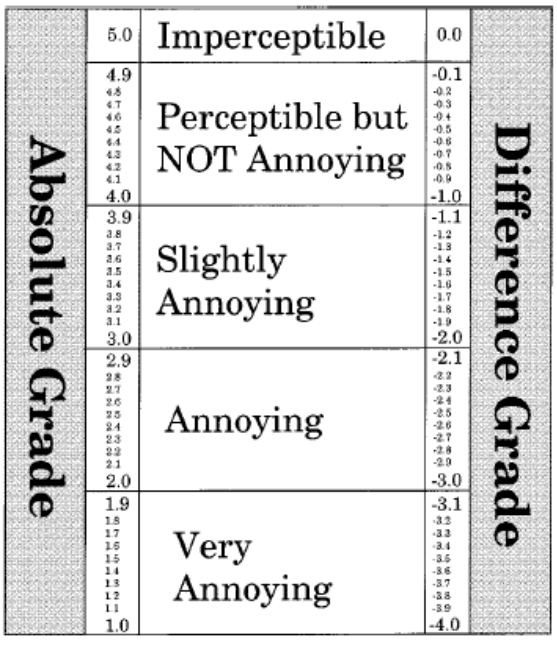


Figure 4.6. Subjective quality scales for ITU-R BS.1116 [11]

5 NEW SCALABLE PERCEPTUAL METRIC

5.1 Introduction

The perceptual metric proposed in this thesis is based on the PEAQ advanced version. In this section, we will first discuss our implementation of the PEAQ advanced version. Since we use subjective test data in developing the new metric, we will also discuss subjective test methodology considered, test audio sequences used, and subjective test results. This section also provides details about the Energy Equalization Approach together with the proposed metric. Finally, comparisons are made between the different objective measurement methods discussed in this section.

5.2 Implementation of the PEAQ Advanced Version

Since the standardization of PEAQ as Recommendation ITU-R BS.1387-1, many audio experts have tried implementing it using the algorithm published in the recommendation such that it satisfies the conformance criteria given in [8]. The recommendation itself has been found to be insufficient in providing complete details of the PEAQ algorithm, and details on the recommendation's deficiencies and the alternate approaches have been discussed in [16]. PEAQ basic version source codes by Lerch [17] and Kabal [18] are freely available on the internet. The FFT-based ear model part of our PEAQ advanced version implementation uses the source code from [18]. The frame size has not been specified for the advanced version in the recommendation. We therefore use a frame size corresponding to 0.6827s (or 32768 samples for 48 kHz sampling frequency) since it results in Distortion Index (DI) and Objective difference grade (ODG) values closest to the conformance results. We have developed the PEAQ advanced

version for 48 kHz sampling frequency since the recommendation provides an algorithm specific to this sampling frequency. For evaluating quality of audio signals having different sampling frequencies, they will have to be converted into 48 kHz sampling frequency using a sample-rate converter before being processed by the PEAQ algorithm.

Table 5.1. DI and ODG values for our implementation and ITU implementation given in [8].

Name	DI	ODG	DI (ITU)	ODG(ITU)
acodsna.wav	1.878	-0.336	1.632	-0.467
bcodtri.wav	1.942	-0.306	2.000	-0.281
ccodsax.wav	0.718	-1.156	0.567	-1.300
ecodsmg.wav	1.634	-0.465	1.594	-0.489
fcodsb1.wav	1.094	-0.833	1.039	-0.877
fcodtr1.wav	1.537	-0.523	1.555	-0.512
fcodtr2.wav	0.083	-1.792	0.162	-1.711
fcodtr3.wav	-0.327	-2.221	-0.783	-2.662
gcodcla.wav	1.537	-0.522	1.457	-0.573
hcodstr.wav	2.210	-0.194	2.232	-0.187
hcodryc.wav	2.436	-0.117	2.410	-0.187
icodsna.wav	-2.275	-3.588	-2.510	-3.664
kcodsme.wav	2.822	-0.015	2.765	-0.029
lcodhrp.wav	1.384	-0.621	1.538	-0.523
lcodpip.wav	2.179	-0.206	2.149	-0.219
mcodcla.wav	-0.054	-1.937	0.430	-1.435
ncodsfe.wav	3.180	0.052	3.163	0.050
scodclv.wav	2.144	-0.220	1.972	-0.293

ITU-R BS.1387-1 provides audio sequences for testing conformance of PEAQ implementation. The results of our PEAQ advanced version implementation for ITU test audio data has been listed in Table 5.1. Comparing the results in Table 5.1 to that of the reference implementation given in [8], we observe that the maximum absolute differences for DI and ODG are 0.484 and 0.502, respectively, corresponding to the audio sequence *mcodcla.wav*. A tolerance limit of ±0.02 is allowed for variation in DI values for conformance of an implementation by the ITU. Our implementation has a difference in

DI value which is greater than the specified tolerance limit. However, since the average difference error in DI values is a relatively small 0.1322, we chose to use this implementation in our analysis anyway.

5.3 Subjective Testing

Subjective testing is conducted to quantify performance of different objective measurement metrics for evaluating high and mid impairment audio. Hence, we consider processed audio at low and mid bitrates. Codecs from the MPEG-4 family are specifically chosen, namely non-scalable AAC (Advanced Audio Coding), non-scalable TwinVQ (Transformed-weighted interleaved vector quantization) and scalable BSAC (Bit Slice Arithmetic Coding). Using these codecs, we encode and decode seven monaural sequences for our subjective testing, of which two are from the MPEG-4 test set while the others come from various classical and popular music sources. Table 5.2 lists the audio sequences used together with its description and time duration.

Table 5.2. Sequences used in subjective testing.

File	Description	Duration (s)
1	pop/rock (Ronnie James Deo)	5
2	pop/rock (Pat Benetar)	9
3	opera (Room with a view – movie)	15
4	harpsichord (MPEG-4 test sequence)	16
5	classical (2001 : Space Odyssey – movie)	17
6	quartet (MPEG-4 test sequence)	23
7	classical (Excaliber – movie)	24

For BSAC, we encode audio signals at 64 kb/s and decode then at 16 kb/s and 32 kb/s, respectively. These sequences are indicated by labels ‘bsac16’ and ‘bsac32’,

respectively. For TwinVQ, we encode and decode at both 32 kb/s and 16 kb/s while for AAC we encode and decode at 32 kb/s. These sequences will be indicated by the labels 'tvq32', 'tvq16' and 'aac32', respectively. In addition to 16 kb/s BSAC, we use another variant of BSAC for which the original audio is lowpass filtered to 6 kHz prior to encoding at 64 kb/s and decoding at 16 kb/s. These audio sequences are labeled as 'filtered'.

The Comparison Category Rating (CCR) approach [19] is used for subjective testing since high impairment audio signals are being evaluated and it also provides for direct comparison between any two codecs. In this approach, an assessor is presented with two sequences and asked to rate their quality relative to one another. Table 5.3 gives the scale used in CCR approach for measuring quality of the second sequence with respect to the first sequence. The subjective test includes 21 assessors, each of whom make a total of 20 comparative evaluations. These tests have been performed with assessors sitting alone in a room listening through headphones to a pre-recorded, vocally annotated audio file being played back from a computer. The test subjects are given full control over the volume, but was not allowed to alter (stop, repeat, etc) the playback of the test sequence [9].

Table 5.4 gives the results of the subjective test averaged over all subjects and all input sequences. The label 'total32' in the table indicates 32 kb/s results for TwinVQ and non-scalable AAC. The second column in the table indicates the mean score – i.e., how much better the first algorithm is relative to the second on a scale of -3 to 3. The order in which the two algorithms are applied during the trial is randomized for each test entry and the results have been permuted appropriately to generate these statistics. A mean

score of zero indicate that the two algorithms are about the same and a score of +3 indicates that the first is 'much better' than the second. The third column gives standard deviation estimated from the subjective data. The last column contains the 99% confidence interval (i.e., the interval that we are 99% certain contains the true mean, assuming that the underlying probability distribution is Gaussian) [9].

Table 5.3. Scale used in CCR approach.

Score	Opinion
3	Much better
2	Better
1	Slightly better
0	About the same
-1	Slightly worse
-2	Worse
-3	Much worse

Table 5.4. Results of subjective testing [9].

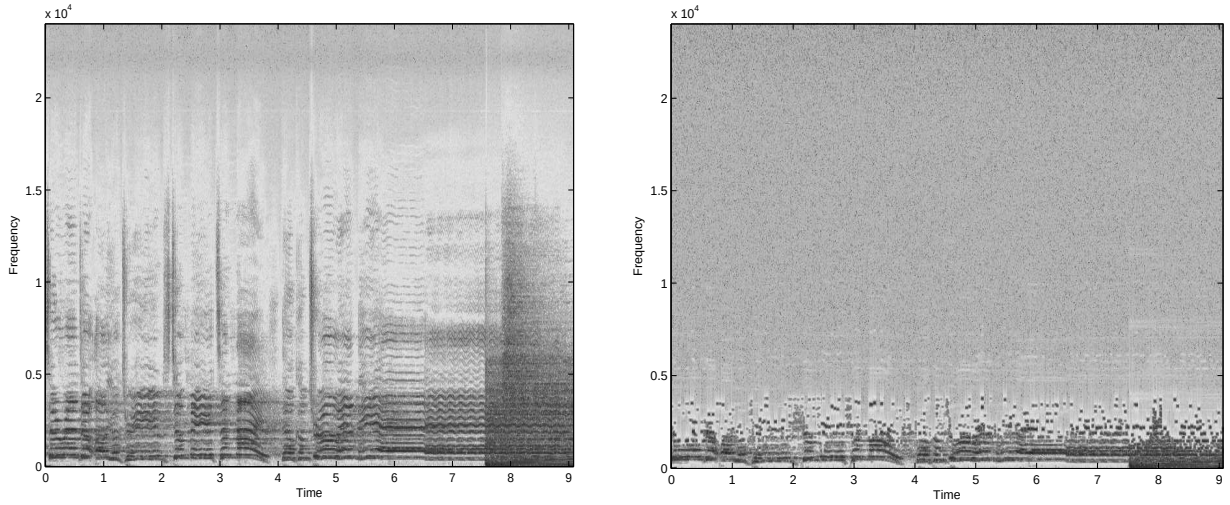
Comparison	**Mean**	**Standard Deviation**	**99% Confidence Interval**
tvq16/bsac16	2.2	0.64	+/- 0.36
filtered/bsac16	1.17	0.92	+/- 0.52
tvq32/bsac32	-0.17	0.98	+/- 0.55
aac32/bsac32	0	0.71	+/- 0.4
total32/bsac32	-0.08	0.85	+/- 0.4

The subjective test results indicate that at 16 kb/s, TwinVQ has a mean score of 2.2 when compared with BSAC. This indicates that TwinVQ provides better perceptual quality at 16 kb/s when compared to BSAC. Also, pre-filtering original audio prior to

BSAC encoding is shown to improve its perceptual quality. The performance of scalable and nonscalable algorithms is found to be nearly the same at 32 kb/s.

5.4 Energy Equalization Approach

The PEAQ has been designed to closely match the subjective results from Recommendation ITU-R BS.1116 [20] for low impairment audio data. Analysis of the performance of PEAQ basic version was made in [9] and it was shown to perform poorly for measuring low bitrate scalable audio quality. Further, a new metric called the Energy Equalization approach (EEA) was also introduced in [9]. The EEA applies time-frequency analysis to the original and processed audio signals in order to measure the quality of the processed signal. Specifically, the perceived quality of reconstructed audio appears to be degraded when isolated signal energy 'islands' are preserved in the 2 to 4 kHz region of its time-frequency plot. These islands are considered as lone time-frequency tiles that remain after those around them have dropped out due to quantization process. The EEA uses the number of isolated time-frequency islands as a measure of perceptual audio quality, grading the sequence with higher number of energy islands as much worse compared to sequence having fewer energy islands. This is done assuming that both audio signals represent the same audio sequence and that the original sequence does not contain such islands. Figure 5.1 (a) and (b) gives the time-frequency plot for the reference benetar audio sequence and its processed sequence using the BSAC codec at 16 kb/s. The isolated islands are clearly observed in the latter case.



(a) (b)

Figure 5.1. (a) Reference bene.wav and (b) its reconstructed signal obtained from the BSAC codec operating at 16 kb/s.

In EEA, the energy of the bandpass spectrogram of the reconstructed signal is first evaluated using:

$$e_k = \sum_{i=0}^{total\_time\_blocks} \sum_{j=0}^{total\_freq\_blocks} (\mathbf{rec\_spec}\ (i, j)_k)^2 \qquad (5.1)$$

where **rec_spec** is a two dimensional matrix containing the spectrogram of the reconstructed signal, *k* indexes the codec being used (BSAC, TwinVQ and pre-filtered BSAC with 16 kb/s bitrate) and *i* indexes the time with *total_time_blocks* being the total number of temporal blocks in the spectrogram, and *j* indexes frequency with *total_freq_blocks* being the total number of frequency blocks in the spectrogram. A truncation threshold T_{kn}, for each codec *k* and audio sequence *n* is applied to the spectrogram of the original signal **o_spec**, to obtain a modified spectrogram **m_spec** defined as

$$\mathbf{m\_spec}(i, j)_{T_{kn}} = \begin{cases} \mathbf{o\_spec}(i, j), & \text{if } |\mathbf{o\_spec}\ (i, j)| \geq T_{kn} \\ 0, & \text{if } |\mathbf{o\_spec}\ (i, j)| < T_{kn} \end{cases} \qquad (5.2)$$

The energy of the modified spectrum is then computed by

$$e_{T_{kn}} = \sum_{i=0}^{total\_time\_blocks} \sum_{j=0}^{total\_freq\_blocks} (\mathbf{m\_spec}(i,j)_{T_{kn}})^2 . \qquad (5.3)$$

The energy of the modified spectrogram is compared with the energy of the reconstructed spectrogram given by (5.1). Finally, an iterative optimization is performed on threshold T_{kn} such that

$$\begin{aligned} e_{T_{kn}} < e_k &\Rightarrow T_{kn} = T_{kn} - \Delta \\ e_{T_{kn}} > e_k &\Rightarrow T_{kn} = T_{kn} + \Delta \end{aligned} \qquad (5.4)$$

where Δ is the step size. For each codec *k* and audio sequence *n*, threshold T_{kn} is chosen such that $e_{T_{kn}} = e_k$.

Since the subjective comparison is differential, the difference in truncation threshold is considered, with column vector $\mathbf{a} = [(T_{k_1 1} - T_{k_2 1}), (T_{k_1 1} - T_{k_3 1}), ..., (T_{k_i n} - T_{k_m n})]^T$. The optimal predictor based on the difference in truncation threshold is determined by solving the linear equation

$$\mathbf{a}x = \mathbf{p} \qquad (5.5)$$

for a scalar *x*, where **a** is a column vector containing differential threshold data and **p** is a column vector containing the subjective test data. The vector **a** and **p** in this case are *nx1* vectors. The vector **a** is scaled to have values in the range (0,3] which corresponds to the absolute range of our subjective data. The vector **a** is also differential. The least square solution to (5.5) is given by

$$\hat{x} = (\mathbf{a}^T \mathbf{a})^{-1} \mathbf{a}^T \mathbf{p} \qquad (5.6)$$

where the solution $\hat{x}$ predicts relative quality of pairs of audio sequences.

5.5 PEAQ Advanced Version versus EEA

In this section, we perform comparisons between PEAQ advanced version, EEA and EAQUAL using 33 reconstructed audio sequences obtained from BSAC and TwinVQ codecs that operate at 16 kb/s and 32 kb/s, and pre-filtered BSAC that operates at 16 kb/s only. The test set includes all the audio sequences mentioned in Table 5.2. The 33 audio sequences form a set of 20 comparisons between audio sequences coded with different codecs.

We first discuss performance results for EEA, followed by results for advanced version and EAQUAL. The truncation threshold corresponding to each processed audio sequence is computed using the procedure discussed in Section 5.4. There may be scenario where energy of the processed signal is greater than that of the reference signal (E.g. harpsichord sequence encoded using BSAC at 64 kb/s and decoded at 32 kb/s). This can be attributed to the effect of quantization, where quantized samples have values greater than that of the original value. Whenever such a scenario is encountered the truncation threshold is chosen to be zero indicating that entire original signal energy is required to approximate the energy in the processed signal.

Vectors **a** and **p**, each of size *20x1* are constructed, with former containing difference in truncation threshold values and latter containing subjective test data. The least squares predictor $\hat{x}$ is obtained using (5.6) and its plot is shown in Fig 5.2. It is observed that slope of the predictor is close to one. Furthermore, this predictor has a correlation coefficient of 0.831 and MSE of 0.276. Correlation coefficient is a measure of the degree of linear relationship present between two variables [21]. For variables having positive relationship, the correlation coefficient is 1.0 for perfect linear

relationship and 0.0 for no linear relationship. The correlation coefficient of two variables X and Y is defined as the ratio of the covariance of X and Y to the product of the standard deviation of X and Y [21] and is given by

$$r = \frac{\sum (X - \bar{X})(Y - \bar{Y})}{\sqrt{\sum (X - \bar{X})^2}\sqrt{\sum (Y - \bar{Y})^2}}. \tag{5.7}$$

To test for the robustness of the predictor, we successively eliminate each of the 20 test cases from (5.5) and compute the predictor $\hat{x}$ using (5.6) for the modified system. The optimal predictor computed is then applied to the actual data that was eliminated in the design process to obtain a predicted objective quality measure. Squared error between predicted value and subjective data are computed and it is shown in Fig. 5.3 along with the corresponding slope variation for different holdout case. It is observed that the variation of $\hat{x}$ is in the range [0, 0.03] and the range of the squared error is limited to [0.008, 0.9263].

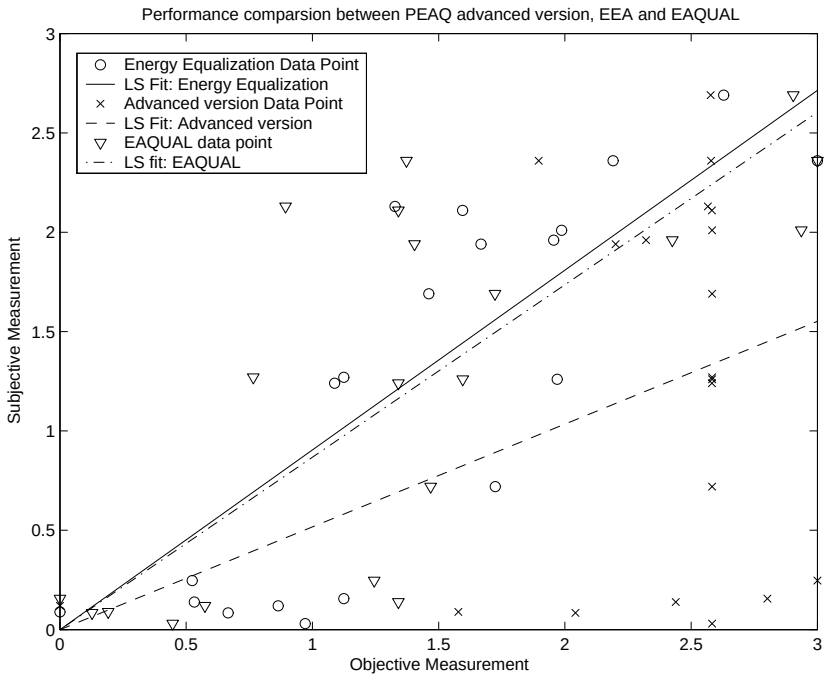


Figure 5.2. Least squares fit of objective measure versus subjective measure

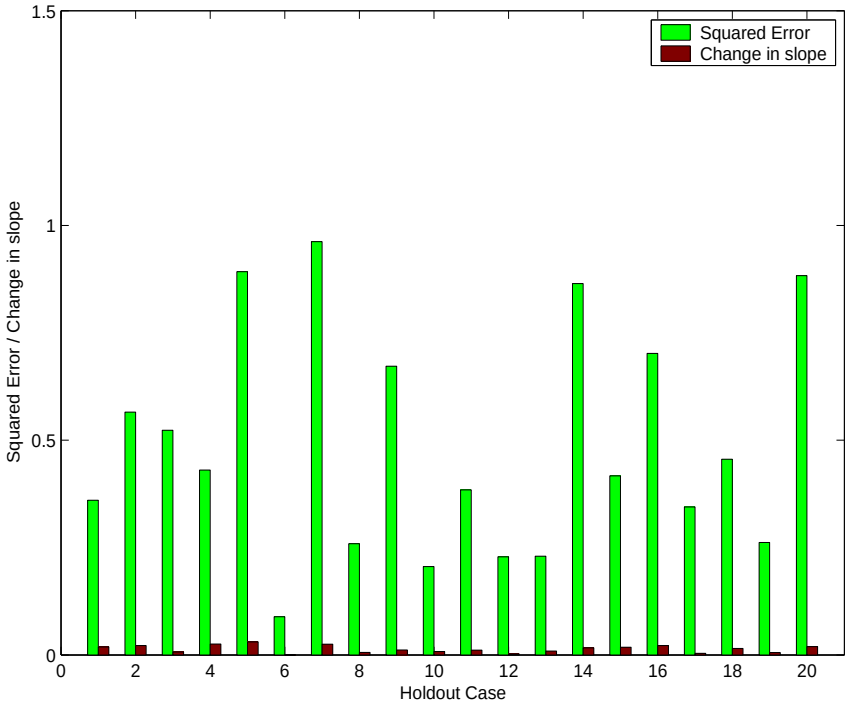


Figure 5.3. Squared error and slope variation for EEA

The ODG values for the above mentioned sequences are computed using PEAQ advanced version. A vector containing difference in ODG values corresponding to pairs of reconstructed audio sequences i.e., $\boldsymbol{a}$ = $[(ODG_{k_1}1 - ODG_{k_2}1)$, $(ODG_{k_1}1 - ODG_{k_3}1)$,..., $(\mathrm{ODG}_{\mathrm{k_i n}} - ODG_{k_m}m)]^{\mathrm{T}}$ is constructed. $\mathrm{ODG}_{\mathrm{k_i n}}$ is the ODG value corresponding to the audio sequence n processed by codec k_i. The predictor $\hat{x}$ for the linear system (5.5) with redefined $\boldsymbol{a}$ is determined using (5.6). This predictor has a slope of 0.51735 with correlation coefficient of 0.25264 and MSE of 0.803. We perform similar analysis to EAQUAL, and it results in a predictor whose slope is 0.868 with correlation coefficient of 0.7434 and MSE of 0.7118. Figure 5.2 also illustrate the results from both PEAQ advanced version and EAQUAL. Since EEA has a correlation coefficient closest to one among the objective measurement methods discussed in this section, it therefore has the

closest relationship between subjective and objective measurement values. A similar analysis has been done using 16 kb/s audio data in [22].

5.6 New Perceptual Audio Quality Metric

The EEA has been shown to be superior to PEAQ basic and advanced versions for measuring low bitrate scalable audio quality. In this section, we propose different methods to improve the performance of PEAQ advanced version for measuring low and mid-quality audio. These include the use of single layer neural network, the use of the EEA truncation threshold as an additional MOV, and the selection of weights for the single layer neural network based on encoded bitrate information. We show that including all of the above mentioned parameters results in a metric that performs well for measuring low, mid and high quality audio.

5.6.1 PEAQ with Single Layer Neural Network

The original PEAQ algorithm uses a multilayer neural network that has been designed to operate optimally for measuring high quality audio. Here, we redesign the neural network of PEAQ advanced version using a single layer neural network. The training data for the design of this neural network includes the 33 reconstructed audio sequences mentioned in the previous section belonging to the low and mid quality audio category.

A vector **m** of size *5x1* is constructed, containing differences in MOVs corresponding to the pair of audio sequences being compared. The difference in MOVs is mapped to a single quality measure by

$$\mathbf{m}^T\mathbf{w} = d \tag{5.8}$$

where **w** is a *5x1* vector containing weights corresponding to the MOV difference values. The weights are computed by solving the following linear equation

$$\mathbf{Aw} = \mathbf{p} \tag{5.9}$$

where A represents a *nx5* matrix $\begin{bmatrix} (MOV_{1,1} - MOV_{2,1}),..,(MOV_{1,5} - MOV_{2,5}) \\ \vdots \quad \vdots \quad \vdots \\ (MOV_{n-1,1} - MOV_{n,1}),...,(MOV_{n-1,5} - MOV_{n,5}) \end{bmatrix}$

containing the difference in MOVs, **p** is a *nx1* vector containing subjective test data and **w** is a *nx1* weight vector. We will consider data from 20 audio comparisons (i.e. $n = 20$). The least square solution for the weights is given by

$$\hat{w} = (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T \mathbf{p}. \tag{5.10}$$

Once the weights are computed, the ODG values for all the audio sequences are found using (5.8), where vector **m** now contains the actual MOV values and *d* contains the resulting ODG value. Using these ODG values, we form a vector containing differences in ODG values corresponding to the audio sequences that have been compared and solve (5.5) to determine the optimal predictor. The plot of the predictor for PEAQ advanced version with redesigned neural network is shown in Fig 5.4.

From Fig 5.4, the slope of the optimal predictor for modified PEAQ advanced version and EAQUAL are found to be nearly the same. The modified metric has a correlation coefficient of 0.8568 that is close to that for EEA alone. Table 5.5 compares the modified and original PEAQ advanced version, EEA and EAQUAL. From the Table 5.5, we clearly see that by incorporating a single layer neural network to the PEAQ advanced version has improved its performance greatly for measuring mid and low quality audio.

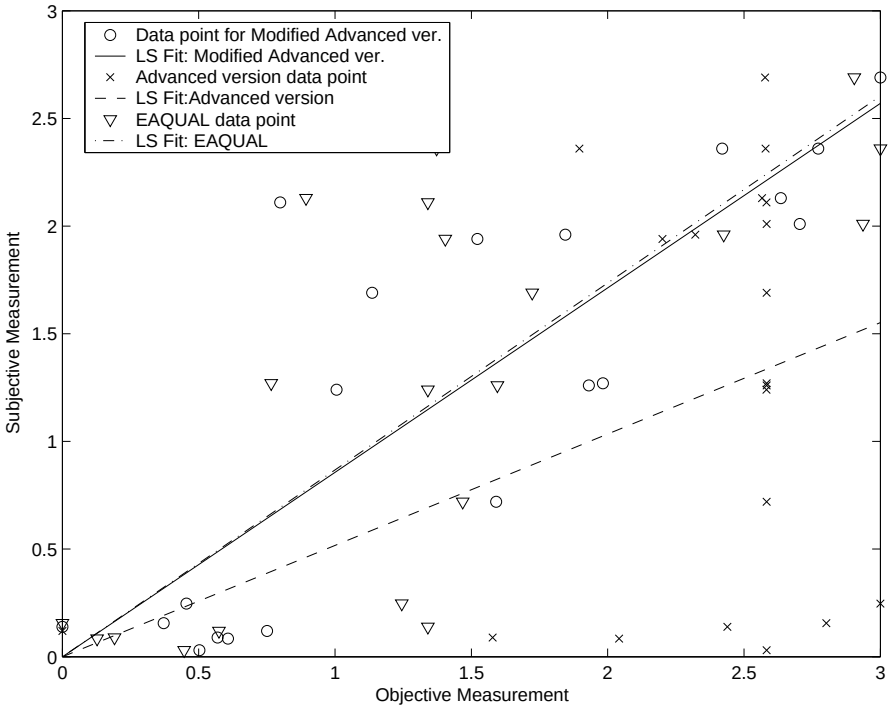


Figure 5.4. Least squares fit of objective measure vs. subjective measure for PEAQ advanced version with single layer neural network.

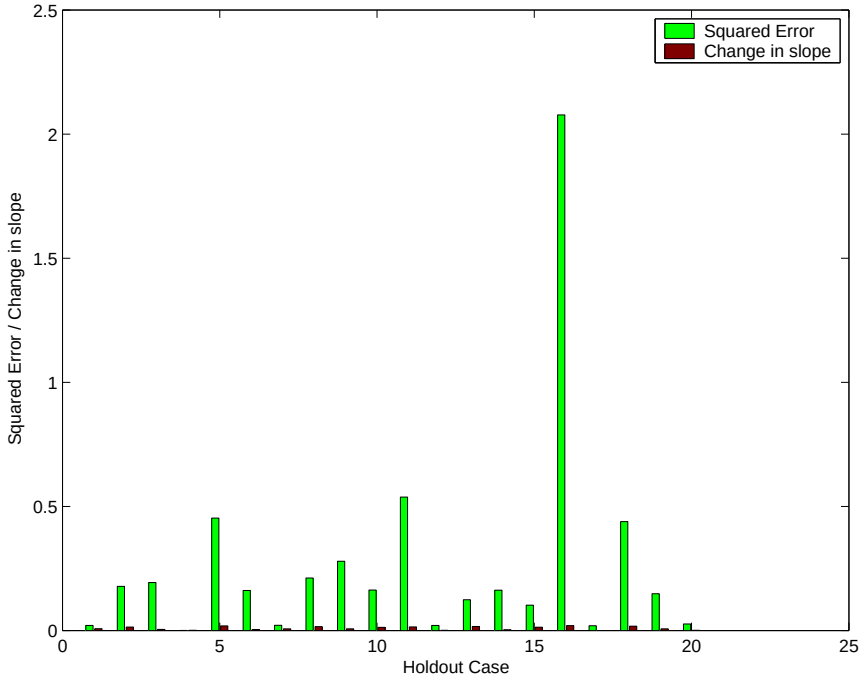


Figure 5.5. Squared error and slope variation for each holdout case for PEAQ advanced version with single layer neural network.

Table 5.5. Comparison between EEA, EAQUAL, and PEAQ advanced version with and without single layer neural network.

Objective metrics	Correlation Coefficient	Slope of its LS fit	MSE
PEAQ advanced version with single layer neural network	0.8385	0.8568	0.2524
EEA	0.831	≈1	0.276
PEAQ advanced version	0.2526	0.5174	0.8033
EAQUAL (PEAQ basic version)	0.7434	0.8680	0.7118

The proposed approach is tested for robustness using the method described in Section 5.5. From Fig 5.5, we observe that the change in slope is small for all holdout cases while the squared error is less than 0.7 in all except one holdout case. This indicates that the predictor is relatively robust with respect to changes in the training set. The robustness may be improved by including larger number of audio sequences belonging to different categories in our training set.

5.6.2 PEAQ Advanced Version with Single Layer Neural Network and EEA MOV

In Section 5.5, slope of the predictor for EEA is observed to be close to one, indicating that difference in truncation threshold can be used to directly evaluate relative quality of two codecs. This is the motivation for using truncation threshold as an additional MOV in the PEAQ advanced version. The six MOVs that include standard five MOVs from PEAQ advanced version and truncation threshold as the sixth MOV are mapped to a single audio quality measure using a single layer neural network designed from the 33 audio sequences mentioned in the previous section. The approach followed in this section is similar to that of previous section except that there are six MOVs instead

of five. Figure 5.6 gives a plot for the least squares fit for PEAQ advanced version as well as the modified advanced metric with the single layer neural network, both with and without the additional truncation threshold MOV. Comparisons of these approaches made in Table 5.6 show clearly that the proposed metric has superior performance with a higher correlation coefficient value and a smaller MSE.

The test for robustness is performed for the predictor using the approach described in Section 5.5. Figure 5.7 gives the change in slope and squared error for each holdout case for the proposed metric and it is observed that the maximum squared error is reduced by 50% when compared to that in previous case. Clearly this indicates that including EEA as an additional MOV has improved the robustness of the predictor.

Table 5.6. Performance comparison between modified PEAQ advanced version with and without EEA MOV and the original PEAQ advanced metric.

Objective metrics	Correlation Coefficient	Slope of its LS fit	MSE
PEAQ advanced version with single layer neural network and EEA MOV	0.8973	0.8461	0.1713
PEAQ advanced version with single layer neural network	0.8385	0.8568	0.2527
PEAQ advanced version	0.2526	0.5174	0.8033

5.6.3 PEAQ with EEA MOV and Single Layer Neural Network with Bitrate Specific Weights

Impairments in processed audio data are often found to increase with decreasing encoder bitrates. If the bitrate is known *a priori*, it can be used to improve the accuracy of measuring the audio quality. In this section, we propose a single layer neural network whose weights are designed specifically for the 32 kb/s and 16 kb/s cases using training data corresponding to each case. The neural network is designed using the approach

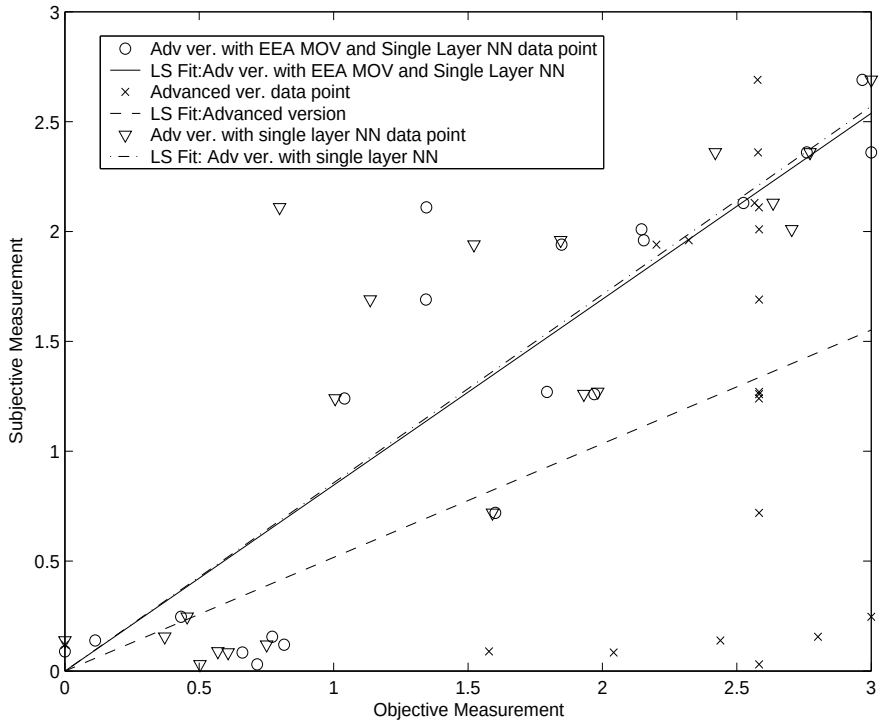


Figure.5.6 Least squares fit of objective measure vs. subjective measure for PEAQ advanced version with EEA MOV and single layer neural network.

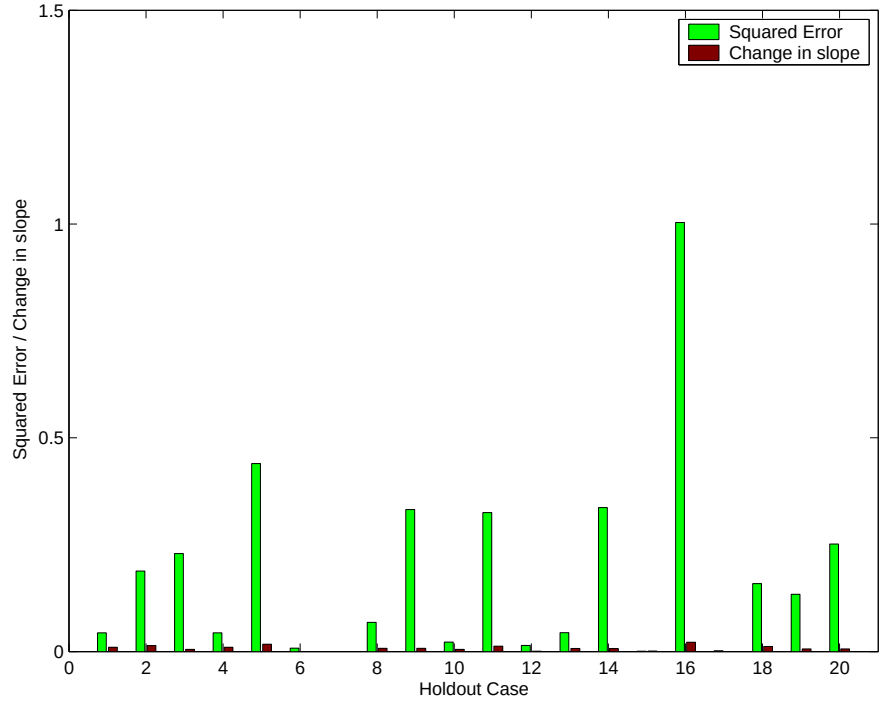


Figure 5.7. Squared error and slope variation for each holdout case for PEAQ advanced version with EEA MOV and single layer neural network.

discussed in Section 5.6.2. Prediction results using the proposed approach are shown in Fig 5.8. This approach is observed to have a correlation coefficient of 0.9155 and an MSE of 0.1537 versus 0.89726 and 0.1713, respectively, as obtained in the previous approach. The value of correlation coefficient for the proposed approach is closer to one thereby indicating that bitrate information improves the accuracy of audio quality measurements.

The test for robustness is done according to earlier sections and the maximum squared error for holdout case is less than 0.9 as shown in Fig 5.9. The change in slope is small indicating that least squares fit remains unaffected by perturbations in the training set. Clearly, we observe that bitrate information improves robustness of the proposed metric.

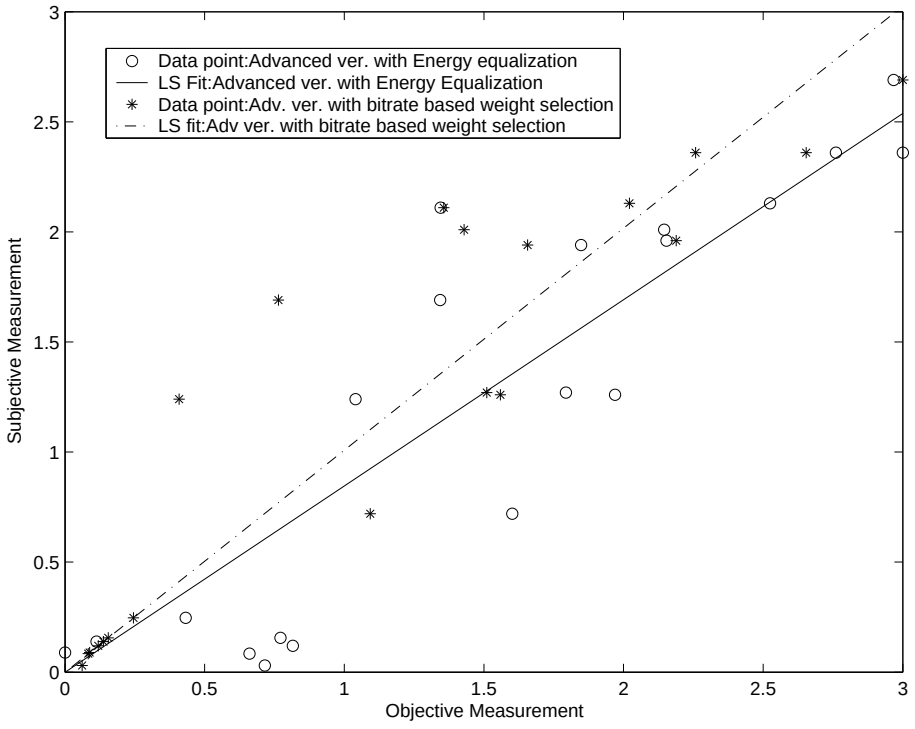


Figure 5.8. LS fit of objective measure versus subjective measure for PEAQ advanced version with EEA MOV and single layer neural network with bitrate specific weights

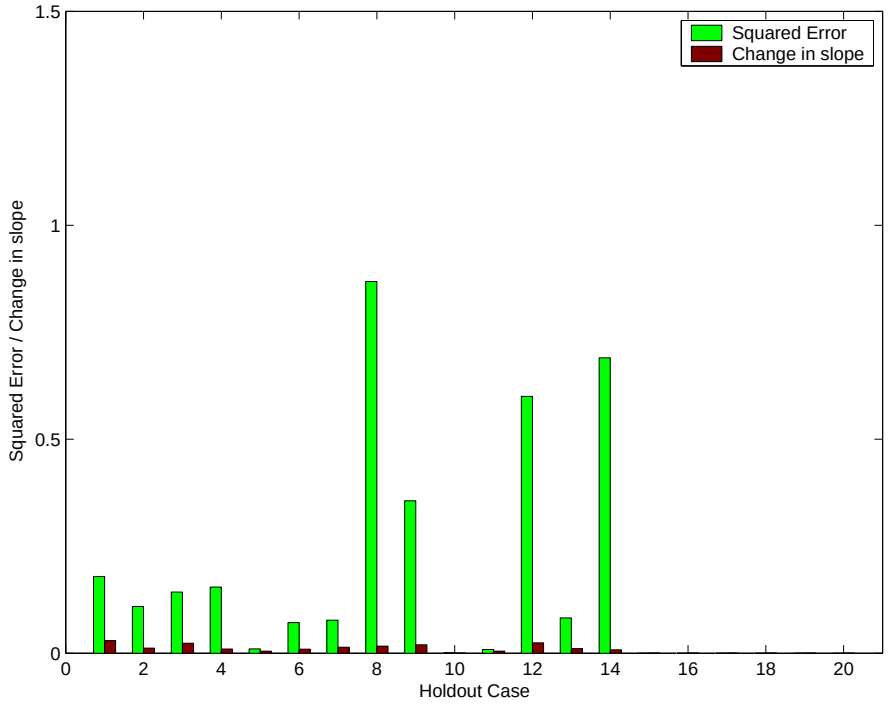


Figure 5.9. Squared error and slope variation for each holdout case for PEAQ advanced version with EEA MOV and single layer neural network with bitrate specific weights.

Similar analysis is performed using audio sequences used for conformance of the PEAQ. Specifically, we choose audio sequences whose ODG values are close to zero, indicating low impairment. We treat the ODG values of these sequences from the original PEAQ advanced version as the true subjective test values, assuming that PEAQ performs optimally for measuring high quality audio. These data points together with its least squares fit are shown in Fig 5.10. We obtain a correlation coefficient of 0.9832, an MSE of 0.0674 and a slope equal to 1.1044 using the proposed approach for high quality audio sequences. In doing the robustness test, we find that the maximum squared error is less than 0.23 as shown in Fig 5.11. Therefore, by using a single layer neural network with bitrate specific weights and including truncation threshold as an additional MOV in PEAQ advanced version, its performance is shown to improve significantly for

measuring low and mid bitrate audio, and yet it still performs well for measuring high quality audio.

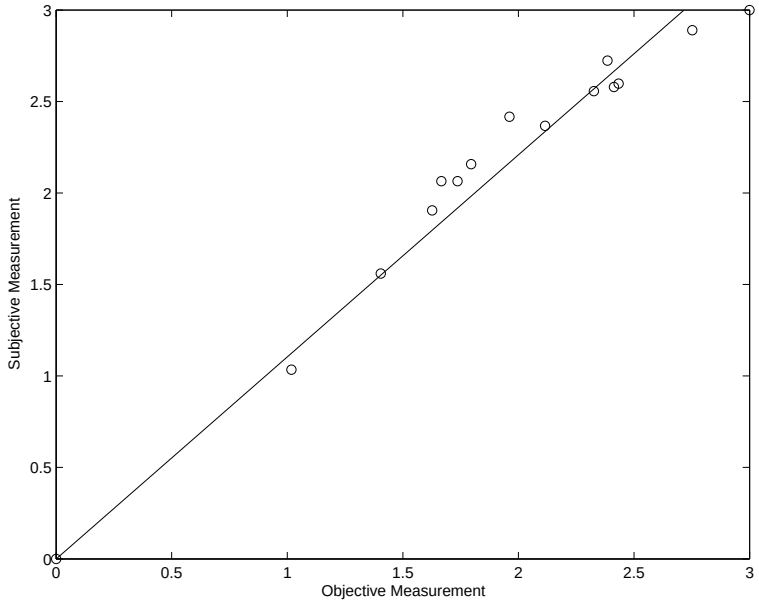


Figure 5.10. Least squares fit of objective measure vs. subjective measure for PEAQ advanced version with EEA MOV and single layer neural network with bitrate specific weights.

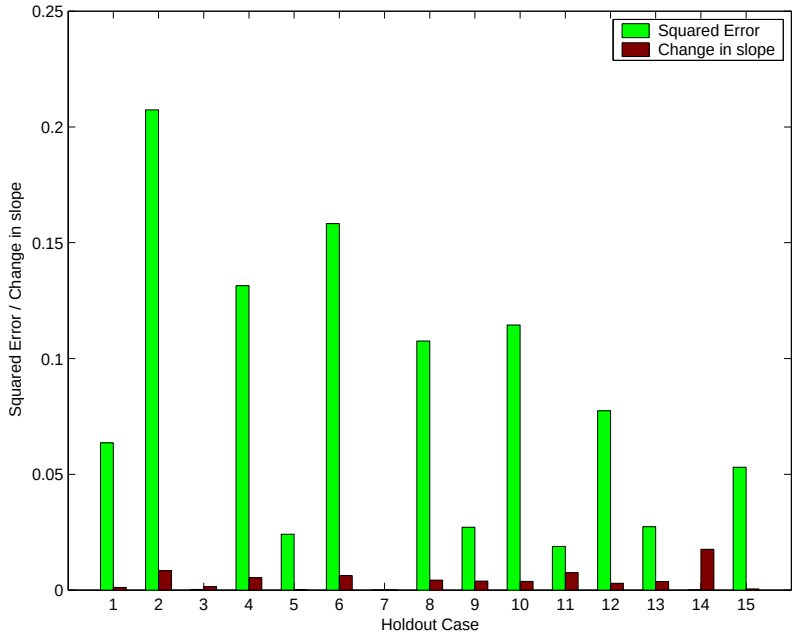


Figure 5.11. Squared error and slope variation for each holdout case for PEAQ advanced version with EEA MOV and single layer neural network with bitrate specific weights.

5.7 Comparison between Modified PEAQ Basic and Advanced Version

Sections 5.6.1 - 5.6.3 discusses different approaches for improving the performance of the PEAQ advanced version for measuring mid and low quality audio. Similar analysis has been previously done for PEAQ basic version [10]. Comparisons between modified basic and advanced versions are made with respect to MSE, slope of the least squares fit and robustness. Table 5.7, 5.8 and 5.9 compares modified basic and advanced versions for approaches discussed in Sections 5.6.2 and 5.6.3 respectively. From the tables, we observe that the modified basic version has lower MSE and lower squared error values for holdout case when compared to the modified advanced version. However, variation in slope corresponding to perturbation in training set is larger in basic version when compared to the advanced version.

Table 5.7. Performance of the basic and advanced version modified for single layer neural network and EEA MOV

Parameter	Modified basic version	Modified advanced version
MSE	0.0927	0.1713
Slope	0.8742	0.8461
Max Squared error for holdout case	0.5646	1.003
Max variation in slope for holdout case	0.3035	< 0.025

Table 5.8. Performance of the modified basic and advanced versions that uses bitrate specific weights for the neural network

Parameter	Modified basic version	Modified advanced version
MSE	0.0177	0.1538
Slope	0.7945	1.0085
Max Squared error for holdout case	0.4070	< 0.9
Max variation in slope for holdout case	0.2718	< 0.03

Table 5.9. Performance of the modified basic and advanced versions for PEAQ conformance test audio sequences

Parameter	Modified basic version	Modified advanced version
MSE	0.0051	0.0674
Slope	0.9998	1.1044
Max Squared error for holdout case	0.0163	0.2074
Max variation in slope for holdout case	0.4652	0.0176

6 CONCLUSION

The PEAQ was designed primarily for evaluating audio codecs that operate in the perceptual transparent domain. More recent audio compression algorithms allow coding at lower bitrates with reconstructed audio having mid to high levels of impairments. The PEAQ advanced version has been shown to be inaccurate in measuring such audio quality. In this thesis, the PEAQ advanced version has been enhanced to measure different audio qualities reliably by including an optimized single layer neural network whose weights are selected based on the bitrate of the encoded audio sequence and using an additional truncation threshold MOV. The performance of this modified PEAQ advanced version is shown to be superior compared to EEA and has comparable performance to the enhanced PEAQ basic version in [10].

The use of more complex neural networks is considered in our future research as it may provide further improvements to our proposed metric. Since ITU-R BS.1534-1 [23] (also known as MUSHRA – Multi Stimulus test with Hidden Reference and Anchor) provides a method for subjective assessment of low and mid quality audio and is more recent than the CCR approach, we plan to follow this recommendation for obtaining subjective test data in our future research.

APPENDIX

Usage notes on the software developed in this thesis

The scalable objective metric developed in this thesis determines relative audio quality between two processed audio sequences. The main function that performs this task is `audio_eval`(). The usage for this function can be obtained by typing the following command on the MATLAB command prompt:

```
>> help audio_eval

   AUDIO_EVAL evaluate relative audio quality

   AUDIO_EVAL(REF_FILENAME, TEST_FILENAME_1, TEST_FILENAME_2, BITRATE_TYPE)
   evaluates the relative audio quality between TEST_FILENAME_1 and
   TEST_FILENAME_2, which are the two processed audio files with
   impairments.  The REF_FILENAME is the original audio file.  The program
   outputs the relative objective quality measure OBJ_QUALITY.  The
   OBJ_QUALITY is in the range '0' to '3'. '0' indicates that the audio
   sequences have the same audio quality, '3' indicates that TEST_FILENAME_2
   is much much better when compared to TEST_FILENAME_1.
   The user can specify an optional BITRATE_TYPE, that is one of the
   following:
       'low' - This implies that the processed audio files has been obtained
       from a low bitrate audio encoded file.  E.g. for low bitrate is 16 kbps
       mono.

       'mid' - This implies that the processed audio files has been obtained
       from a mid bitrate audio encoded file.  E.g. for mid bitrate is 32 kbps
       mono.

       'high' - This implies that the TEST_FILENAME has been obtained from
       a high bitrate audio encoded file.  E.g. for high bitrate is 64 kbps
       mono.

       NOTE: When no option is provided the weights corresponding to the
       mid-bitrates are chosen for the single layer neural network.
```

The `audio_eval()` first checks the sample rates of the input audio sequences and then resamples them to 48 kHz if necessary. It then computes the truncation threshold of each of the processed audio sequence using `trunk_thresh()`. Each pair of resampled reference and processed audio sequences is time-aligned using `audio_align()` followed by computation of MOVs of the PEAQ advanced version using `PEAQ_adv_ver()`. The table containing weights for the single layer neural network for low and mid-bitrate cases have been obtained from `adv_snn_eea_wts()`, and

weights corresponding to high bitrate audio has been obtained from `adv_snn_eea_wts_itu()` respectively. Based upon the user option on bitrate, the weights for the single layer neural network are selected and they weight the difference between the MOVs and truncation threshold values corresponding to the two processed audio signals. If the bitrate is unspecified, the program will select mid-bitrate weights. The relative objective quality measure is the output of the single layer neural network.

The PEAQ advanced version alone consists of 41 '.m' files with `PEAQ_adv_ver()` being the main function. This function returns DI, ODG and MOV values. An example of its usage is given below.

```
>>[DI,ODG,MOV]=PEAQ_adv_ver(ref_filename,test_filename)
```

The input reference and test audio sequences can be time aligned using `audio_align()`. This function generates two time aligned wav files *sco.wav* and *scc.wav* corresponding to the reference and test audio files. An example for the usage of `audio_align` is shown below.

```
>>audio_align(ref_filename,test_filename)
```

The truncation threshold corresponding to a processed audio signal can be determined using `trunk_thresh()` with reference and test audio files as its input. An example of its usage is given below.

```
>>truncation_value=trunc_thresh(ref_filename,test_filename)
```

Scalable Perceptual Metric[1]

audio_eval.m

```
function obj_quality = audio_eval(ref_filename, test_filename_1, test_filename_2, ...
    bitrate_type)
% AUDIO_EVAL evaluate relative audio quality
%
%   AUDIO_EVAL(REF_FILENAME, TEST_FILENAME_1, TEST_FILENAME_2, BITRATE_TYPE)
%   evaluates the relative audio quality between TEST_FILENAME_1 and
%   TEST_FILENAME_2, which are the two processed audio files with
%   impairments.  The REF_FILENAME is the original audio file.  The program
%   outputs the relative objective quality measure OBJ_QUALITY.  The user can
%   specify an optional BITRATE_TYPE, that is one of the following:
%       'low' - This implies that the processed audio files has been obtained
%       from a low bitrate audio encoded file.  E.g. for low bitrate is 16 kbps
%       mono.
%
%       'mid' - This implies that the processed audio files has been obtained
%       from a mid bitrate audio encoded file.  E.g. for mid bitrate is 32 kbps
%       mono.
%
%       'high' - This implies that the TEST_FILENAME has been obtained from
%       a high bitrate audio encoded file.  E.g. for high bitrate is 64 kbps
%       mono.
%
%       NOTE: When no option is provided the weights corresponding to the
%       mid-bitrates are chosen for the single layer neural network.
%
%   Author: Rahul Vanam.

if (nargin < 3)
    disp(' The function needs atleast three input arguments');
    return;
end

addpath('../PEAQ_adv_ver');

% Required sampling rate
SAMPL_RATE = 48000;

% Constant for audio align function
N = 30000;

% tables for Single Layer Neural network weights
% Weights for low bit rates
weights_low = [0.0072 0.0047 -0.9480 -0.8192 -0.0005 0.1803];

% Weights for mid bit rates
weights_mid = [-0.0183 0.6585 -0.9363 -1.4590 0.1956 -0.3890];

% Weights for high bit rates
weights_high = [-0.0045 -0.6142 -0.0160 -0.2836 -0.0666 0.0157];

% Check the sampling frequency of the ref and test audio files
[x, fs_x, nbits_x] = wavread(ref_filename);
[y_1, fs_y_1, nbits_y_1] = wavread(test_filename_1);
[y_2, fs_y_2, nbits_y_2] = wavread(test_filename_2);
% PEAQ advanced version implementation is optimal for 48 kHz sampling frequency
% Sample rate converter
if (fs_x ~= SAMPL_RATE)
    x_out = resample(x, SAMPL_RATE, fs_x);
    wavwrite(x_out, SAMPL_RATE, nbits_x, 'ref.wav');
    clear x_out;
    ref_filename = 'ref.wav';
```

[1] This code incorporates portions (with permission) of the PEAQ implementation by P. Kabal at McGill University [16].

```
end
clear x;

if (fs_y_1 ~= SAMPL_RATE)
    y_out = resample(y_1, SAMPL_RATE, fs_y_1);
    wavwrite(y_out, SAMPL_RATE, nbits_y_1, 'test_1.wav');
    clear y_out;
    test_filename_1 = 'test_1.wav';
end
clear y_1;

if (fs_y_2 ~= SAMPL_RATE)
    y_out = resample(y_2, SAMPL_RATE, fs_y_2);
    wavwrite(y_out, SAMPL_RATE, nbits_y_2, 'test_2.wav');
    clear y_out;
    test_filename_2 = 'test_2.wav';
end
clear y_2;

% Compute the trucation threshold
trunc_value_1 = trunc_thresh(ref_filename, test_filename_1);
trunc_value_2 = trunc_thresh(ref_filename, test_filename_2);

%--------------------------------------------------
% Determine the MOVs from the PEAQ advanced version
%--------------------------------------------------
% Time align the input and output audio sequences
audio_align (ref_filename, test_filename_1, N);

% sco and scc are time-aligned files
% Determine the MOVs of the PEAQ advanced version
[DI_1, ODG_1, MOVB_1] = PEAQ_adv_ver ('sco.wav', 'scc.wav');

% Time align the input and output audio sequences
audio_align (ref_filename, test_filename_2, N);

% sco and scc are time-aligned files
% Determine the MOVs of the PEAQ advanced version
[DI_2, ODG_2, MOVB_2] = PEAQ_adv_ver('sco.wav', 'scc.wav');

%-------------------------------------------------------------------------
% For low, mid and high bitrates, the weights specific to the bitrates shall be
% used together with PEAQ MOVs to determine audio quality.  When bitrate is
% unknown or not provided by the user, weights corresponding to mid-bitrate
% is used.
%-------------------------------------------------------------------------
if (nargin == 4)
    if(strcmp(bitrate_type, 'low'))
        mov_diff = MOVB_1 - MOVB_2;
        obj_quality = [mov_diff (trunc_value_1 - trunc_value_2)] * weights_low';
    elseif (strcmp(bitrate_type, 'mid'))
        mov_diff = MOVB_1 - MOVB_2;
        obj_quality = [mov_diff (trunc_value_1 - trunc_value_2)] * weights_mid';
    else (strcmp(bitrate_type, 'high'))
        obj_quality = ([MOVB_1 trunc_value_1] - [MOVB_2 trunc_value_2]) ...
            * weights_high';
    end
else
    % No bitrate option: Use mid bitrate
     mov_diff = MOVB_1 - MOVB_2;
     obj_quality = [mov_diff (trunc_value_1 - trunc_value_2)] * weights_mid';
end

% delete the temporary files created
delete('sco.wav');
delete('scc.wav');
if (fs_x ~= SAMPL_RATE)
    delete('ref.wav');
end
if (fs_y_1 ~= SAMPL_RATE)
    delete('test_1.wav');
```

```
end
if (fs_y_2 ~= SAMPL_RATE)
    delete('test_2.wav');
end
%--------------------------------------------------------------------------
% End of File
%--------------------------------------------------------------------------
```

PEAQ Advanced Version Implementation

PEAQ_adv_ver.m

```
function [DI, ODG, MOVB] = ...
    PEAQ_adv_ver (Fref, Ftest, StartS, EndS)
% Perceptual evaluation of audio quality.

% - StartS shifts the frames, so that the first frame starts at that sample.
%   This is a two element array, one element for each input file. If StartS is
%   a scalar, it applies to both files.
% - EndS marks the end of data. The processing stops with the last frame that
%   contains that sample. This is a two element array, one element for each
%   input file.  If EndS is as scalar, it applies to both files.
%
%   Note : This file is the main function for the PEAQ advanced version.
%   This code incorporates portions (with permission) of the PEAQ
%   implementation by P. Kabal at McGill University [16]

% Globals (to save on copying in/out of functions)
global MOVC PQopt
global data_struct
global SAMPL_FREQ

% Analysis parameters
NF = 2048;
Nadv = NF / 2;
Version = 'Advanced';
NUM_FILTERS = 40;

% Options
PQopt.ClipMOV = 0;
PQopt.PCinit = 0;
PQopt.PDfactor = 1;
PQopt.Ni = 1;
PQopt.DelayOverlap = 1;
PQopt.DataBounds = 1;
PQopt.EndMin = NF / 2;

addpath ('CB', 'MOV', 'Misc', 'Patt');

if (nargin < 3)
    StartS = [0, 0];
end
if (nargin < 4)
    EndS = [];
end

%profile on

% Get the number of samples and channels for each file
WAV(1) = PQwavFilePar (Fref);
WAV(2) = PQwavFilePar (Ftest);

% Reconcile file differences
PQ_CheckWAV (WAV);
if (WAV(1).Nframe ~= WAV(2).Nframe)
    disp ('>>> Number of samples differ: using the minimum');
end

% Sampling frequency
```

```
SAMPL_FREQ = WAV(1).Fs;

% Data boundaries
Nchan = WAV(1).Nchan;

% Frame limit is used in Advanced Version for FB based ear model
[Frame_limit, StartS, Fstart, Fend] = ...
    PQ_Bounds (WAV, Nchan, StartS, EndS, PQopt);

% Number of PEAQ frames
Np = Fend - Fstart + 1;
if (PQopt.Ni < 0)
    PQopt.Ni = ceil (Np / abs(PQopt.Ni));
end

% Initialize the MOV structure
MOVC = PQ_InitMOVC (Nchan, Np);

% Initialize the filter memory
Nc = PQCB (Version);
for (j = 0:Nchan-1)
    Fmem(j+1) = PQinitFMem (Nc, PQopt.PCinit);
end

%--------------------------------------------------------------------
% FFT based ear model
%--------------------------------------------------------------------
is = 0;
for (i = -Fstart:Np-1)

    % Read a frame of data
    xR = PQgetData (WAV(1), StartS(1) + is, NF);    % Reference file
    xT = PQgetData (WAV(2), StartS(2) + is, NF);    % Test file
    is = is + Nadv;

    % Process a frame
    for (j = 0:Nchan-1)
        [MOVI(j+1), Fmem(j+1)] = PQeval (xR(j+1,:), xT(j+1,:), Fmem(j+1));
    end

    if (i >= 0)
        % Move the MOV precursors into a new structure
        PQframeMOV (i, MOVI,Version);   % Output is in global MOVC
    end
end
clear xR;
clear xT;
clear Fmem;

%--------------------------------------------------------------------
% Time average of the MOV values
%--------------------------------------------------------------------
if (PQopt.DelayOverlap)
    Nwup = Fstart;
else
    Nwup = 0;
end
[MOVB(3), MOVB(4)] = PQavgMOVB (MOVC, Nchan, Nwup,'FFT','Advanced');

%--------------------------------------------------------------------
% Filter Bank based ear model
%--------------------------------------------------------------------
% Setting of the playback level
L_MAX = 92; % in dB (SPL)
scale_fac = 10 ^(L_MAX/20) / 32767;

temp_length = Frame_limit(2);
ref_data =  PQgetData (WAV(1), StartS(1), temp_length);  % Reference file
test_data = PQgetData (WAV(2), StartS(2), temp_length);  % Test file

% start index should be minimum of 1
```

```
Frame_limit(1) = max(1, Frame_limit(1));

% Scaling the data and starting the data from the Data boundary
ref_data = ref_data(:,Frame_limit(1):end) * scale_fac;
test_data = test_data(:,Frame_limit(1):end) * scale_fac;

% Initiate the persistant memory used in the algorithm
data_struct = init_data_struct(Nchan);

if (SAMPL_FREQ == 48000)
       % Filter coefficient for DC reject filter-1
       num_1 = [1 -2 1];
       den_1 = [1 -1.99517 0.995174];

       % Filter coefficients for DC reject filter-2
       num_2 = num_1;
       den_2 = [1 -1.99799 0.997998];
end

% Although modifications have been made for input sampling rate of 44.1
% kHz, the resulting DI and ODG values are not close to that for 48 kHz.
% Therefore, we require that all input audio sequences with sampling
% rates other than 48 kHz be converted to 48 kHz using a sample rate
% converter.

if (SAMPL_FREQ == 44100)
       % Filter coefficient for DC reject filter-1
       num_1 = [1 -2 1];
       den_1 = [1 -1.9947 0.99475];

       % Filter coefficients for DC reject filter-2
       num_2 = num_1;
       den_2 = [1 -1.9978 0.99782];
end

% Determine the coefficients of the filter bank
[h_real, h_imag] = filterbank_coeff;

for i = 1:Nchan
    % Apply the DC rejection filter
    filter_out_1(i,:) = filter(num_1,den_1,ref_data(i,:));
    filter_out_ref(i,:) = filter(num_2,den_2,filter_out_1(i,:));

    filter_out_1(i,:) = filter(num_1,den_1,test_data(i,:));
    filter_out_test(i,:) = filter(num_2,den_2,filter_out_1(i,:));

       % Decompose the signal into auditory filter bands
       out_real_ref(i,:,:) = filter_bank(filter_out_ref(i,:), h_real);
    out_imag_ref(i,:,:) = filter_bank(filter_out_ref(i,:), h_imag);

       out_real_test(i,:,:) = filter_bank(filter_out_test(i,:), h_real);
    out_imag_test(i,:,:) = filter_bank(filter_out_test(i,:), h_imag);
end

%--------------------------------------------------
% Frame size = frame length * 32;
% The frame lenght is frame size down sampled by 32
%--------------------------------------------------
FRAME_LENGTH = 1024;

size_out = size(out_real_ref);
Fstart = 0;
Fend = floor(size_out(3)/FRAME_LENGTH)-1;
Np = Fend - Fstart + 1;

N_thresh = 0.1;
start_index_LT = size_out(3); % start index for loudness threshold

is = 1;
count(1:Nchan) = 0;
count_2(1:Nchan) = 0;
```

```
count_3(1:Nchan) = 0;
count_4(1:Nchan) = 0;
length_N(1:Nchan) = 0;

for (chan = 1:Nchan)
    disp(['chan = ' num2str(chan)]);
    data_start = 1;
    data_end = FRAME_LENGTH;
    for (frames = -Fstart:Np-1)
        %-----------------------------------------------------------------------
               % Pass the signal to process according to peripheral ear model
        %-----------------------------------------------------------------------
        disp(frames);
        real_data(:,:) = out_real_ref(chan,:,data_start:data_end);
        imag_data(:,:) = out_imag_ref(chan,:,data_start:data_end);

               [E_ref, E2_ref,data_struct.Eref_forward_mask(chan,:)] = ...
            peripheral_model(real_data, imag_data, ...
            data_struct.Eref_forward_mask(chan,:));

        real_data(:,:) = out_real_test(chan,:,data_start:data_end);
        imag_data(:,:) = out_imag_test(chan,:,data_start:data_end);

               [E_test, E2_test, data_struct.Etest_forward_mask(chan,:)] = ...
            peripheral_model(real_data, imag_data, ...
            data_struct.Etest_forward_mask(chan,:));
        clear real_data;
        clear imag_data;

         data_start = data_end + 1;
         data_end = FRAME_LENGTH + data_end;

               [mod_ref, mod_test, E_loud_ref, E_P_ref, E_P_test,N_ref(chan,:), ...
                N_test(chan,:)] = pre_process(E_ref,E_test,E2_ref,E2_test,chan);

               %-----------------------------------------------------------------------
               % Computation of the Model Output Variable (MOV)
               %-----------------------------------------------------------------------
               % Determine change in modulation
               [ModDiff, TempWt] = ...
            mod_diff(mod_ref,mod_test, E_loud_ref,Frame_limit(1));

        modDiff(chan,count(chan)+1:count(chan) + length(ModDiff)) = ModDiff;
        tempWt (chan,count(chan)+1:count(chan) + length(ModDiff)) = TempWt;
        count(chan) = count(chan) + length(ModDiff);
        clear ModDiff;
        clear TempWt;

        % constants corresponding to the MOV RmsNoiseLoud_A
        alpha = 2.5;
        ThreshFac = 0.3;
        NL_min = 0.1;

        % computation of the spectral average of the momentary noise loudness
               temp = noise_loudness(E_P_ref, E_P_test, mod_test, ...
            mod_ref,  alpha, ThreshFac, NL_min);
        RmsNoiseLoud_A(chan,(count_2(chan)+1):count_2(chan) + length(temp)) = temp;
        count_2(chan) = count_2(chan) + length(temp);
        clear temp;

        % constant corresponding to RmsMissingComponents_A
        alpha = 1.5;
        ThreshFac = 0.15;
        NL_min = 0;
        temp = noise_loudness(E_P_test, E_P_ref, ...
            mod_ref, mod_test, alpha, ThreshFac, NL_min);

        RmsMissingComponents_A(chan,(count_3(chan)+1):count_3(chan) + ...
            length(temp)) = temp;
        count_3(chan) = count_3(chan) + length(temp);
        clear temp;
```

```
        % constants corresponding to AvgLinDist_A
        alpha = 1.5;
        ThreshFac = 0.15;
        NL_min = 0;
        temp = noise_loudness(E_P_ref, E_ref, mod_ref, ...
            mod_ref, alpha, ThreshFac, NL_min);
        AvgLinDist_A(chan,(count_4(chan)+1):count_4(chan) + ...
            length(temp)) = temp;
        count_4(chan) = count_4(chan) + length(temp);
        clear temp;

               % Apply Loudness threshold
               % Determine the sample at which the overall loudness of both test and
               % reference signal exceeds 0.1 sone for either left or right channel
        index_thresh = ((N_ref(chan,:) > N_thresh) & (N_test(chan,:) > N_thresh));
        start_index_thresh = find(index_thresh) + length_N(chan);
        if (start_index_thresh ~= 0)
            if start_index_thresh(1) <= start_index_LT
                start_index_LT = start_index_thresh(1);
            end
        end
        length_N(chan) = length_N(chan) + length(N_ref(chan,:));
       end % for (frames = -Fstart:Np-1)
end

%-----------------------------------------------------------
% Delayed averaging of 0.5s applied
% delay in samples = Sampling frequency/192 * 0.5 s;
% Overlap of the Data boundary and delay avg is considered
%-----------------------------------------------------------
delay_samples = round(SAMPL_FREQ/192 * 0.5);
delay_avg = max(0, delay_samples - Frame_limit(1));
for chan = 1:Nchan
       % Apply the delay average before temporal averaging
       modDiff_val(chan,:) = modDiff(chan,delay_avg + 1:end);
       tempWt_val(chan,:) = tempWt(chan,delay_avg + 1:end);
       % Calculate the squared average of the modulation difference
       RmsModDiff_A(chan) = sqrt(NUM_FILTERS) .* ...
        sqrt(sum((tempWt_val(chan,:) .^ 2) .* ...
        (modDiff_val(chan,:) .^2))/sum(tempWt_val(chan,:) .^ 2));
end

% 50ms in samples
delay_samples_50ms = round(SAMPL_FREQ/192 * 50/1000);
%-------------------------------------------------------------------------
% For parameters that require both Delay averaging and loudness threshold to be
% applied. See "Perceptual Audio Quality assessment using Non-Linear Filter
% bank", Thilo Thiede,1999, pp.102
%-------------------------------------------------------------------------
max_delay_samples = max(delay_avg, start_index_LT + delay_samples_50ms);

% Delay due to Loudness threshold only
LT_delay = start_index_LT + delay_samples_50ms

for chan = 1:Nchan
    RmsNoiseLoud_A_avg(chan,:) = RmsNoiseLoud_A(chan,max_delay_samples:end);

    % Although ITU-R BS.1387 doesnot specify this, it is pressumed because
    % RmsNoiseLoudAysm_A is determined from this and the above parameter.
    RmsMissingComponents_A_avg(chan,:) = ...
        RmsMissingComponents_A(chan,LT_delay:end);

    RmsNoiseLoud_A_value(chan) = ...
        sqrt(1/length(RmsNoiseLoud_A_avg(chan,:)) * ...
        sum(RmsNoiseLoud_A_avg(chan,:) .^ 2));

    RmsMissingComponents_A_value(chan) = ...
        sqrt(1/length(RmsMissingComponents_A_avg(chan,:)) * ...
        sum(RmsMissingComponents_A_avg(chan,:) .^ 2));
```

```
    MOVC.RMSNoiseLoudAsym_A(chan) = RmsNoiseLoud_A_value(chan) + ...
        0.5 * RmsMissingComponents_A_value(chan);

    AvgLinDist_A_avg(chan,:) = ...
        AvgLinDist_A(chan,LT_delay:end); % Doubt regarding the delay?

    MOVC.AvgLinDist_A_value(chan) = 1/length(AvgLinDist_A_avg(chan,:)) * ...
       sum(AvgLinDist_A_avg(chan,:));
end

%--------------------------------------------------------------------------
% Averaging of the MOV corresponding to left and right channel
%--------------------------------------------------------------------------
MOVB(1) = sum(RmsModDiff_A)/(Nchan);
MOVB(2) = sum(sum(MOVC.RMSNoiseLoudAsym_A))/(Nchan);
MOVB(5) = sum(sum(MOVC.AvgLinDist_A_value))/(Nchan);

% Neural net
[DI, ODG] = PQnNet (MOVB);

% Summary printout
PQprtMOV (MOVB, ODG);

%----------
function PQ_CheckWAV (WAV)
% Check the file parameters

if (WAV(1).Nchan ~= WAV(2).Nchan)
    error ('>>> Number of channels differ');
end
if (WAV(1).Nchan > 2)
    error ('>>> Too many input channels');
end
if (WAV(1).Nframe ~= WAV(2).Nframe)
    disp ('>>> Number of samples differ');
end
if (WAV(1).Fs ~= WAV(2).Fs)
    error ('>>> Sampling frequencies differ');
end

%----------
function [Frame_limit, StartS, Fstart, Fend] = ...
    PQ_Bounds (WAV, Nchan, StartS, EndS, PQopt)

PQ_NF = 2048;
PQ_NADV = (PQ_NF / 2);

if (isempty (StartS))
    StartS(1) = 0;
    StartS(2) = 0;
elseif (length (StartS) == 1)
    StartS(2) = StartS(1);
end
Ns = WAV(1).Nframe;

% Data boundaries (determined from the reference file)
if (PQopt.DataBounds)
    Lim = PQdataBoundary (WAV(1), Nchan, StartS(1), Ns);
    fprintf ('PEAQ Data Boundaries: %ld (%.3f s) - %ld (%.3f s)\n', ...
             Lim(1), Lim(1)/WAV(1).Fs, Lim(2), Lim(2)/WAV(1).Fs);
else
    Lim = [Starts(1), StartS(1) + Ns - 1];
end

Frame_limit = Lim;

% Start frame number
Fstart = floor ((Lim(1) - StartS(1)) / PQ_NADV);

% End frame number
Fend = floor ((Lim(2) - StartS(1) + 1 - PQopt.EndMin) / PQ_NADV);
```

```
%----------
function MOVC = PQ_InitMOVC (Nchan, Np)
MOVC.MDiff.Mt1B = zeros (Nchan, Np);
MOVC.MDiff.Mt2B = zeros (Nchan, Np);
MOVC.MDiff.Wt   = zeros (Nchan, Np);
MOVC.NMR.NMRseg = zeros (Nchan, Np);
MOVC.EHS.EHS = zeros (Nchan, Np);

% Advanced version Filter bank model
MOVC.RmsModDiff_A = zeros(Nchan, Np);
MOVC.RMSNoiseLoudAsym_A = zeros(Nchan);
MOVC.AvgLinDist_A_value = zeros(Nchan, Np);
%------------------------------------------------------------------------
% End of File
%------------------------------------------------------------------------
```

modulation.m

```
function [mod_value, E] = modulation(E2,chan,testvec_type);
% modulation.m
%
%   This function determines the modulation of the envelope at each filter
%   output using the unsmeared excitation pattern
%
%   Author: Rahul Vanam

global data_struct;
global SAMPL_FREQ; % Sampling frequency

% Number of filter pairs in the filter bank
NUM_FILTERS = 40;

% inline the function containing the center frequencies of the Filters in
% the filter bank
filter_bank_table;

% time constants
T_100 = 0.05;
T0 = 0.008;

step_size = 192;

% simplified loudness value
E = zeros(size(E2));

% Modulation of the envelope
mod_value = zeros(size(E2));

size_input = size(E2);
in_length = size_input(2);

for i = 1:NUM_FILTERS
    % initialized value of the filter history is zero
    if strcmp(testvec_type, 'ref')
        prev_value_der = data_struct.Eder_ref(chan,i);
        prev_value_E2 = data_struct.E2_ref(chan,i);
        prev_value_E = data_struct.Ebar_ref(chan,i);
    else
        prev_value_der = data_struct.Eder_test(chan,i);
        prev_value_E2 = data_struct.E2_test(chan,i);
        prev_value_E = data_struct.Ebar_test(chan,i);
    end

    T = T0 + 100/fc(i) * (T_100 - T0);
    a = exp(-step_size/(SAMPL_FREQ * T));
    for j = 1:in_length
        % Absolute value of the derivative of the simplified loudness value
        E_der = prev_value_der * a + (1-a) * SAMPL_FREQ/step_size * ...
            abs(E2(i,j)^0.3 - prev_value_E2 ^0.3);
```

```
        E(i,j) = a * prev_value_E + (1-a) * E2(i,j)^0.3;

        prev_value_der = E_der;
        prev_value_E2 = E2(i,j);
        prev_value_E = E(i,j);
        mod_value(i,j) = E_der/(1 + E(i,j)/0.3);
    end
    if strcmp(testvec_type, 'ref')
        data_struct.Eder_ref(chan,i)= prev_value_der;
        data_struct.E2_ref(chan,i) = prev_value_E2;
        data_struct.Ebar_ref(chan,i) = prev_value_E;
    else
        data_struct.Eder_test(chan,i) = prev_value_der;
        data_struct.E2_test(chan,i) = prev_value_E2;
        data_struct.Ebar_test(chan,i) = prev_value_E;
    end
end
%--------------------------------------------------------------------------
% End of File
%--------------------------------------------------------------------------
```

mod_diff.m

```
function [modDiff,temp_wt] = ...
    mod_diff(mod_ref,mod_test, E_loud_ref,start_sample);
% mod_diff.m
%
%   This function determines the change in modulation using the modulation
%   of the envelope at each filter output of the reference and test signal
%
%   Author: Rahul Vanam

% Sampling frequency
global SAMPL_FREQ;

% Number of Filters in the Filter bank
NUM_FILTERS = 40;

% inline the function containing the center frequencies of the Filters in
% the filter bank
filter_bank_table;

offset = 1;

% local modulation difference
mod_diff_local = zeros(size(mod_ref));

% momentary modulation difference
modDiff = zeros(1,length(mod_diff_local));

% Level dependent weighting factor
temp_wt = zeros(1,length(mod_diff_local));

mod_diff_local = abs(mod_test - mod_ref)./(offset + mod_ref);

mod_diff_length = size(mod_diff_local);
mod_diff_length = mod_diff_length(2);

i = [1:mod_diff_length];
E_thresh = power(10,(0.4 * 0.364 * (fc(:)/1000).^-0.8));
modDiff(i) = 100/NUM_FILTERS * sum(mod_diff_local(:,i));

for j = 1:mod_diff_length
    temp_wt(j) = sum(E_loud_ref(1:end,j)./(E_loud_ref(1:end,j) + ...
        E_thresh(1:end) .^ 0.3));
end
%--------------------------------------------------------------------------
% End of File
%--------------------------------------------------------------------------
```

level_pattern_adapt.m

```
function [E_P_ref, E_P_test] = level_pattern_adapt(E_ref, E_test, chan);
% level_pattern_adapt.m
%
%   This function adapts the reference and test excitation pattern to each
%   other.
%
%   Author: Rahul Vanam

global data_struct;
global SAMPL_FREQ; % Sampling Frequency

% Number of Filters in the Filter bank
NUM_FILTERS = 40;

% inline the function containing the center frequencies of the Filters in
% the filter bank
filter_bank_table;

% time constants in seconds
T_min = 0.008;
T_100 = 0.05;

% constants specific to the Filter Bank model
step_size = 192;
z = 40;
M = 3;

% Smoothen the excitation pattern using a first order IIR filter
P_ref = zeros(size(E_ref));
P_test = zeros(size(E_test));

input_length = size(E_ref);
input_length = input_length(2);

for i = 1:NUM_FILTERS
    % time constant
    T = T_min + 100/fc(i) * (T_100 - T_min);
    a = exp(-step_size/(SAMPL_FREQ * T));
    b = 1 - a;
    P_ref_old = data_struct.Pref(chan,i);
    P_test_old = data_struct.Ptest(chan,i);
    for j = 1:input_length
        P_ref(i,j) = a * P_ref_old + b * E_ref(i,j);
        P_test(i,j) = a * P_test_old + b * E_test(i,j);
        P_ref_old = P_ref(i,j);
        P_test_old = P_test(i,j);
    end
    data_struct.Pref(chan,i) = P_ref_old;
    data_struct.Ptest(chan,i) = P_test_old;
end

%-------------------------------------------------------------------------
% Level Adaptation
%-------------------------------------------------------------------------
% Momentary correction factor
E_L_ref = E_ref;
E_L_test = E_test;
for i = 1:input_length
    num_sum = sum(sqrt(P_test(1:end,i) .* P_ref(1:end,i)));
    den_sum = sum(P_test(:,i));
    lev_corr = (num_sum/den_sum)^2;
    if lev_corr > 1
        E_L_ref(:,i) = E_ref(:,i)/lev_corr;
    else
        E_L_test(:,i) = E_test(:,i) * lev_corr;
    end
end

%-------------------------------------------------------------------------
```

```
% Pattern Adaptation
%-------------------------------------------------------------------------
R_test = zeros(size(E_L_test));
R_ref  = zeros(size(E_L_ref));
for i = 1:NUM_FILTERS
    % time constant
    T = T_min + 100/fc(i) * (T_100 - T_min);
    a = exp(-step_size/(SAMPL_FREQ * T));

    for j = 1:input_length
        num_sum = 0;
        den_sum = 0;
        for k = 1:j
            num_sum = num_sum + a ^ (k-1) * E_L_test(i,j-(k-1)) * ...
                E_L_ref(i,j-(k-1));
            den_sum = den_sum + a ^ (k-1) * E_L_ref(i,j-(k-1)) * ...
                E_L_ref(i,j-(k-1));
        end
        if(den_sum == 0)
            if (num_sum > 0)
                R_test(i,j) = 0;
                R_ref(i,j) = 1;
            end
            if ((num_sum == 0) & (i > 1))
                R_test(i,j) = R_test(i-1,j);
                R_ref(i,j) = R_ref(i-1,j);
            end
            if ((num_sum == 0) & (i == 1))
                R_test(i,j) = 1;
                R_ref(i,j) = 1;
            end
        else
            R = num_sum/den_sum;
            if R < 1
                R_test(i,j) = 1;
                R_ref(i,j) = R;
            else
                R_test(i,j) = 1/R;
                R_ref(i,j) = 1;
            end
        end
    end
end

% Spectrally adapted patterns
E_P_ref = zeros(size(E_L_ref));
E_P_test = zeros(size(E_L_test));

for i = 1:NUM_FILTERS
    % time constant
    T = T_min + 100/fc(i) * (T_100 - T_min);
    a = exp(-step_size/(SAMPL_FREQ * T));

    prev_test_value = data_struct.pattcorr_test(chan,i);
    prev_ref_value = data_struct.pattcorr_ref(chan,i);
    for j = 1 : input_length
        M1 = (M-1)/2;
        M2 = M1;
        M1 = min(i-1,M1);
        M2 = min(M2, z-i);
        M_new = M1 + M2 + 1;
        sum_test(i,j) = sum(R_test(i-M1 : i+M2,j));
        sum_ref(i,j) = sum(R_ref(i-M1 : i+M2,j));
        patt_corr_test(i,j) = a * prev_test_value + (1-a)/M_new * sum_test(i,j);
        patt_corr_ref(i,j) = a * prev_ref_value + (1-a)/M_new * sum_ref(i,j);
        prev_test_value = patt_corr_test(i,j);
        prev_ref_value = patt_corr_ref(i,j);
        E_P_ref(i,j) = E_L_ref(i,j) * prev_ref_value;
        E_P_test(i,j) = E_L_test(i,j) * prev_test_value;
    end
    data_struct.pattcorr_test(chan,i) = prev_test_value;
```

```
    data_struct.pattcorr_ref(chan,i) = prev_ref_value;
end
%-------------------------------------------------------------------------
% End of File
%-------------------------------------------------------------------------
```

init_data_struct.m

```
function data_struct = init_data_struct(Nchan);
% init_data_struct.m
%   This function initializes the members of the data structure which are
%   used as persistant memory between function calls
%
%   Author: Rahul Vanam

global data_struct;

NUM_FILTERS = 40;

% History buffer of the filter used in Time domain smearing
data_struct.Eref_forward_mask  = zeros(Nchan,NUM_FILTERS);
data_struct.Etest_forward_mask = zeros(Nchan,NUM_FILTERS);

% History buffer of the filter used in Preprocessing stage for smoothing
% the energies in each filter channel
data_struct.Pref = zeros(Nchan,NUM_FILTERS);
data_struct.Ptest = zeros(Nchan,NUM_FILTERS);

% History buffer of the filter used in Pattern correction
% The pattern correction factor is initialized to 1. See "An Examination
% and interpretation of the ITU-R BS.1387: PEAQ", P.Kabal, 2002, pp.25.
data_struct.pattcorr_ref = ones(Nchan,NUM_FILTERS);
data_struct.pattcorr_test = ones(Nchan,NUM_FILTERS);

% History buffers of the filters used in Modulation
data_struct.E2_ref = zeros(Nchan,NUM_FILTERS);
data_struct.E2_test = zeros(Nchan,NUM_FILTERS);

data_struct.Eder_ref = zeros(Nchan,NUM_FILTERS);
data_struct.Eder_test = zeros(Nchan,NUM_FILTERS);

data_struct.Ebar_ref = zeros(Nchan,NUM_FILTERS);
data_struct.Ebar_test = zeros(Nchan,NUM_FILTERS);
%-------------------------------------------------------------------------
% End of File
%-------------------------------------------------------------------------
```

freq_spread.m

```
function [out_real, out_imag] = freq_spread(in_real, in_imag)
% freq_spread.m
%
%   This function applies frequency spreading to the output of the filter
%   bank. Upward spreading is first applied followed by downward spreading.
%
%   Author: Rahul Vanam

% Sampling frequency
global SAMPL_FREQ;

% Number of filter pairs in the filter bank
NUM_FILTERS = 40;

% inline the function containing the center frequencies of the Filters in
% the filter bank
filter_bank_table;

% Compute center frequencies in Bark scale
z_high = 7 * asinh(fc(40)/650);
z_low = 7 * asinh(fc(1)/650);
```

```
%------------------------------------------------------------------------
% Level dependent upward spreading
% z(40) - z(1)/39 is the distance in bark between two adjacent filter bands
%------------------------------------------------------------------------
dist = power(0.1, (z_high-z_low)/(39 * 20));

% Coefficients for temporal smoothing
a = exp(-32/(SAMPL_FREQ*0.1));
b = 1-a;

out_real = in_real;
out_imag = in_imag;
size_in_real = size(in_real);
input_length = size_in_real(2);
cu = zeros(1,NUM_FILTERS);

%------------------------------------------------------------------------
% This portion of the algorithm is not clearly given in the ITU-R 1387-1
% pp 52.  See 'Examination and interpretation of ITU-R BS.1387: Perceptual
% Evaluation of Audio Quality', P.Kabal,2002,pp 17.
%------------------------------------------------------------------------
for n = 1:input_length
        for i = 1:NUM_FILTERS
       % Calculate the level dependent slope
       L = 10 * log10(max(in_real(i,n) .^ 2 + in_imag(i,n) .^ 2, 10^-6));
       s = max(4, 24 + 230/fc(i) - 0.2 * L);

       % Calculate spread fraction and smooth it over time
       % P.Kabal's document varries with the ITU-R psuedo-code, However
       % this equation follows the ITU-R psuedo-code
       cu(i) = a * power(dist,s) + b * cu(i);

       % spreading of band 'i'
       d1 = in_real(i,n);
       d2 = in_imag(i,n);

       for j = (i+1):NUM_FILTERS
           d1 = d1 * cu(i);
           d2 = d2 * cu(i);
           out_real(j,n) = out_real(j,n) + d1;
           out_imag(j,n) = out_imag(j,n) + d2;
       end
        end % for i = 1:NUM_FILTERS

    % Downward spreading
    c1 = power(dist,31);
    d1 = 0;
    d2 = 0;
    for i = NUM_FILTERS:1
        % spreading of band 'i'
        d1 = d1 * c1 + out_real(i,n);
        d2 = d2 * c1 + out_imag(i,n);
        out_real(i,n) = d1;
        out_imag(i,n) = d2;
    end
end % for n = 1:input_length
%------------------------------------------------------------------------
% End of File
%------------------------------------------------------------------------
```

forward_mask.m

```
function [outData,E_forward_mask] = forward_mask(inData, E_forward_mask)
% forward_mask.m
%
%   This function applies forward masking to the unsmeared excitation
%   pattern to obtained the Excitation pattern
%
%   Author: Rahul Vanam

% Sampling Frequency
```

```
global SAMPL_FREQ;

% Number of Filters in the Filter bank
NUM_FILTERS = 40;

% inline the function containing the center frequencies of the Filters in
% the filter bank
filter_bank_table;

% Time constants in seconds
T_min = 0.004;
T_100 = 0.020;

% output excitation pattern
outData = zeros(size(inData));

input_length = size(inData);
input_length = input_length(2);

for i = 1:NUM_FILTERS
    T = T_min + 100/fc(i) * (T_100 - T_min);
    a = exp(-192/(SAMPL_FREQ * T));

    % Previous output
    prev_output = E_forward_mask(i);

    for j = 1:input_length
        outData(i,j) = a * prev_output + (1-a) * inData(i,j);
        prev_output = outData(i,j);
    end
    E_forward_mask(i) = prev_output;
end
%--------------------------------------------------------------------------
% End of File
%--------------------------------------------------------------------------
```

filterbank_coeff.m

```
function [h_real, h_imag] = filterbank_coeff();
% filterbank_coeff.m
%   This function generates and returns the Real and imaginary Filter bank
%   coefficients
%
%   Author : Rahul Vanam

global SAMPL_FREQ; % Sampling frequency
NUM_FILTERS = 40;
MAX_NUM_COEFF = 1456;
TIME_PERIOD = 1/SAMPL_FREQ; % Sampling period
DOWN_SAMPL_FACTOR = 32;

 % Inlined tables required by the Filter bank
 filter_bank_table;

 h_real = zeros(NUM_FILTERS, MAX_NUM_COEFF);
 h_imag = zeros(NUM_FILTERS, MAX_NUM_COEFF);

 % Generate the Filter coefficients
for i = 1:NUM_FILTERS
    for j = 1:N(i)
        h_real(i,j) = 4/N(i) * (sin(pi * j/N(i)))^2 * ...
            cos(2 * pi * fc(i) * (j-(N(i)/2)) * TIME_PERIOD);
        h_imag(i,j) = 4 /N(i) * sin(pi * j/N(i))^2 * ...
            sin(2 * pi * fc(i) * (j-(N(i)/2)) * TIME_PERIOD);
    end
end
%--------------------------------------------------------------------------
% End of File
%--------------------------------------------------------------------------
```

filter_bank_table.m

```
% filter_bank_table.m
%
%   This file gives the tables containing the center frequency and length of
%   the impulse response/samples
%
% Author : Rahul Vanam

% Center frequency
fc = [50.00,116.19,183.57,252.82,324.64,399.79,479.01,563.11,652.97,...
      749.48,853.65,966.52,1089.25,1223.10,1369.43,1529.73,1705.64,...
      1898.95,2111.64,2345.88,2604.05,2888.79,3203.01,3549.90,...
      3933.02,4356.27,4823.97,5340.88,5912.30,6544.03,7242.54,...
      8014.95,8869.13,9813.82,10858.63,12014.24,13292.44,14706.26,...
      16270.13,18000.02];

% Length of impulse response/samples
N = [1456,1438,1406,1362,1308,1244,1176,1104,1030,956,884,814,748,686,...
    626,570,520,472,430,390,354,320,290,262,238,214,194,176,158,144,...
    130,118,106,96,86,78,70,64,58,52];

D = [1,10,26,48,75,107,141,177,214,251,287,322,355,386,416,444,469,493,...
    514,534,552,569,584,598,610,622,632,641,650,657,664,670,676,681,686,...
    690,694,697,700,703];
%--------------------------------------------------------------------------
% End of File
%--------------------------------------------------------------------------
```

filter_bank

```
function [out_down_sampl] = filter_bank(inData, filter_coeff)
% filter_bank.m
%
%   This function spilts the input signal into auditory filter bands
%
%   Author: Rahul Vanam

NUM_FILTERS = 40;
MAX_NUM_COEFF = 1456;
DOWN_SAMPL_FACTOR = 32;

% Inlined tables required by the Filter bank
filter_bank_table;

% Output of real and imaginary filters

% Every input data is delayed by number of samples equal to difference
% of the impulse response length with the length of the longest impulse
% response plus one
out_data = zeros(NUM_FILTERS, (MAX_NUM_COEFF + length(inData)));

% Down sampled data
out_down_sampl = zeros(NUM_FILTERS, ceil((MAX_NUM_COEFF + length(inData))/...
    DOWN_SAMPL_FACTOR));

for i = 1:NUM_FILTERS
     filt_out = fftfilt(filter_coeff(i,:), ...
         [inData zeros(1,length(filter_coeff(i,:))-1)],4096);
     out_data(i,1:length(filt_out)) = filt_out;
end

for i = 1:NUM_FILTERS
    out_data(i,:) = circshift(out_data(i,:),[1,D(i)]);
    out_down_sampl(i,1:end) = out_data(i,1:DOWN_SAMPL_FACTOR:end);
end
%--------------------------------------------------------------------------
% End of File
%--------------------------------------------------------------------------
```

backward_mask.m

```
function outData = backward_mask(inData);
% backward_mask.m
%
%   This function applies the backward masking to the rectified output
%   followed by down-sampling by a factor of 6
%
%   Author: Rahul Vanam

% Number of Filters in the Filter bank
NUM_FILTERS = 40;

% Raised Cosine FIR filter
raised_cos_filt = zeros(1,12);

for i = 1:12
   raised_cos_filt(i) = cos(pi*(i - 1 - 5)/12)^2;
end

inData_size = size(inData);

filter_output = zeros(1, length(raised_cos_filt)+ inData_size(2) - 1);
outData = zeros(NUM_FILTERS, ...
    ceil((length(raised_cos_filt)+ inData_size(2) - 1)/6));
for i = 1:NUM_FILTERS
    filter_output = conv(raised_cos_filt, inData(i,:));

    % downsample by factor of 6
    outData(i,:) = filter_output(1:6:end);
end

% Multiplying by calibration factor
outData = 0.9761/6 * outData;
%-------------------------------------------------------------------------
% End of File
%-------------------------------------------------------------------------
```

overall_loudness.m

```
function N_total = overall_loudness(E);
% overall_loudness.m
%
%   This function determines the overall loudness of the given signal using
%   the unsmeared excitation pattern
%
%   Author: Rahul Vanam

% Number of filter pairs in the filter bank
NUM_FILTERS = 40;

% inline the function containing the center frequencies of the Filters in
% the filter bank
filter_bank_table;

% constant
const = 1.26539;
z = 40;

N = zeros(size(E));
size_E = size(E);
N_total = zeros(1,size_E(2));

for i = 1:NUM_FILTERS
    E_thresh = power(10, 0.364 * (fc(i)/1000) ^ -0.8);
    s = power(10, 0.1 *(-2-2.05*atan(fc(i)/4000) - ...
        0.75 * atan((fc(i)/1600)^2)));
    N(i,:) = const * (E_thresh/(10^4 * s))^ 0.23 * ...
        (((1-s + (s * E(i,:)/E_thresh)).^ 0.23) - 1);
end
```

```
i = [1:size_E(2)];
N_total(i) = 24/z * sum(max(N(:,i),0));
%--------------------------------------------------------------------------
% End of File
%--------------------------------------------------------------------------
```

out_mid_ear_filt.m

```
function [out_real, out_imag] = out_mid_ear_filt(in_real, in_imag)
% out_mid_ear_filt.m
%
%   This functions models the outer and middle ear characteristic and
%   applies a weight to the output of each filter bank
%
%   Author: Rahul Vanam

% Number of filter pairs in the Filter bank
NUM_FILTERS = 40;

% inline the function containing the center frequencies of the filters of
% the filter bank
filter_bank_table;

for i = 1:NUM_FILTERS
    freq_khz = fc(i)/10^3;

    % Compute the weights in dB
    weight = -0.6 * 3.64 * (freq_khz)^ -0.8 + ...
        6.5 * exp(-0.6*((freq_khz-3.3)^2)) - 10^-3 * freq_khz^3.6;

    weight = 10 ^ (weight/20);
    in_real(i,:) = in_real(i,:) * weight;
    in_imag(i,:) = in_imag(i,:) * weight;
end

out_real = in_real;
out_imag = in_imag;
%--------------------------------------------------------------------------
% End of File
%--------------------------------------------------------------------------
```

noise_loudness.m

```
function NL_avg = ...
    noise_loudness(E_P_ref, E_P_test, mod_test, mod_ref, alpha, ThreshFac, ...
    NL_min);
% noise_loudness.m
%
%   This function determines the partial loudness of the noise in the
%   presence of the masker
%
%   Author: Rahul Vanam

% constant
S0 = 1;

% Number of filter pairs in the filter bank
NUM_FILTERS = 40;

% inline the function containing the center frequencies of the Filters in
% the filter bank
filter_bank_table;

s_test = ThreshFac * mod_test + S0;
s_ref  = ThreshFac * mod_ref  + S0;

% partial noise loudness
NL = zeros(size(E_P_ref));

% parameter that determines amount of masking
```

```
beta = exp(-alpha * (E_P_test - E_P_ref)./E_P_ref);

%-------------------------------------------------------------------------
% The Zwicker's loudness formula is split into two parts, one depending on
% the properties of the masker alone and one depending on the masker and
% maskee together.
%-------------------------------------------------------------------------
for i = 1:NUM_FILTERS
    E_thresh = power(10, (0.4 * 0.364 * (fc(i)/1000)^-0.8));
    NL(i,:) = (E_thresh ./ s_test(i,:)) .^ 0.23 .* ...
        ((1 + (max(s_test(i,:) .* E_P_test(i,:) - s_ref(i,:) .* ...
        E_P_ref(i,:), 0))./(E_thresh + s_ref(i,1:end) .* E_P_ref(i,1:end) ...
        .* beta(i,1:end))).^ 0.23 - 1);
end

%-------------------------------------------------------------------------
% Spectral averaging of the momentary noise loudness values. See
% "An Examination and interpretation of ITU-R BS.1387: PEAQ", P.Kabal,
% 2002. pp.36
% NOTE: The momentary noise loudness can be considered to be similar in
% definition to momentary modulation difference. See Equation(64),
% ITU-R BS.1387-1
%-------------------------------------------------------------------------
size_NL = size(NL);
i = [1:size_NL(2)];

index = (NL >= NL_min);
NL = NL .* index;

NL_avg = 24/NUM_FILTERS * sum(max(NL((1:end),i),0));
index = (NL_avg < NL_min);
NL_avg(index) = 0;
%-------------------------------------------------------------------------
% End of File
%-------------------------------------------------------------------------
```

PQgroupCB.m

```
function Eb = PQgroupCB (X2, Ver)
% Group a DFT energy vector into critical bands
% X2 - Squared-magnitude vector (DFT bins)
% Eb - Excitation vector (fractional critical bands)
%
% P. Kabal, McGill University, Copyright (C) 2004
% P. Kabal $Revision: 1.2 $  $Date: 2004/02/05 04:25:46 $

global SAMPL_FREQ;
persistent Nc kl ku Ul Uu Version

Emin = 1e-12;

if (~ strcmp (Ver, Version))
    Version = Ver;
    % Set up the DFT bin to critical band mapping
    NF = 2048;
    Fs = SAMPL_FREQ;
    [Nc, kl, ku, Ul, Uu] = PQ_CBMapping (NF, Fs, Ver);
end

% Allocate storage
Eb = zeros (1, Nc);

% Compute the excitation in each band
for (i = 0:Nc-1)
    Ea = Ul(i+1) * X2(kl(i+1)+1);       % First bin
    for (k = (kl(i+1)+1):(ku(i+1)-1))
        Ea = Ea + X2(k+1);              % Middle bins
    end
    Ea = Ea + Uu(i+1) * X2(ku(i+1)+1);  % Last bin
    Eb(i+1) = max(Ea, Emin);
end
```

```
%---------------------------------------
function [Nc, kl, ku, Ul, Uu] = PQ_CBMapping (NF, Fs, Version)

[Nc, fc, fl, fu] = PQCB (Version);

% Fill in the DFT bin to critical band mappings
% Rahul: This computation is different from that given in 2.1.5.1.
% EAQUAL also gives an option to use standard pseudocode(FFTModel.cpp).
% Worth observing it.

df = Fs / NF;
for (i = 0:Nc-1)
    fli = fl(i+1);
    fui = fu(i+1);
    for (k = 0:NF/2)
        if ((k+0.5)*df > fli)
            kl(i+1) = k;        % First bin in band i
            Ul(i+1) = (min(fui, (k+0.5)*df) ...
                       - max(fli, (k-0.5)*df)) / df;
            break;
        end
    end
    for (k = NF/2:-1:0)
        if ((k-0.5)*df < fui)
            ku(i+1) = k;        % Last bin in band i
            if (kl(i+1) == ku(i+1))
                Uu(i+1) = 0;       % Single bin in band
            else
                Uu(i+1) = (min(fui, (k+0.5)*df) ...
                           - max(fli, (k-0.5)*df)) / df;
            end
            break;
        end
    end
end
%---------------------------------------------------------------------------
% End of File
%---------------------------------------------------------------------------
```

PQgetData.m

```
function x = PQgetData (WAV, i, N)
% Get data from internal buffer or file
% i - file position
% N - number of samples
% x - output data (scaled to the range -32768 to +32767)
%
% Only two files can be "active" at a time.
% N = 0 resets the buffer
%
% P. Kabal, McGill University, Copyright (C) 2003
%
% Note: Code modified for memory allocation size for PEAQ advanced version.

persistent Buff

iB = WAV.iB + 1;
if (N == 0)
    Buff(iB).N = 20 * 1024;     % Fixed size
    Buff(iB).x = PQ_ReadWAV (WAV, i, Buff(iB).N);
    Buff(iB).i = i;
end

if (N > Buff(iB).N)
    % The memory size has been modified for PEAQ advanced version
    Buff(iB).N = N;
end

% Check if requested data is not already in the buffer
is = i - Buff(iB).i;
if (is < 0 | is + N - 1 > Buff(iB).N - 1)
```

```
    Buff(iB).x = PQ_ReadWAV (WAV, i, Buff(iB).N);
    Buff(iB).i = i;
end

% Copy the data
Nchan = WAV.Nchan;
is = i - Buff(iB).i;
x = Buff(iB).x(1:Nchan,is+1:is+N-1+1);

%------
function x = PQ_ReadWAV (WAV, i, N)
% This function considers the data to extended with zeros before and
% after the data in the file. If the starting offset i is negative,
% zeros are filled in before the data starts at offset 0. If the request
% extends beyond the end of data in the file, zeros are appended.

Amax = 32768;
Nchan = WAV.Nchan;

x = zeros (Nchan, N);

Nz = 0;
if (i < 0)
    Nz = min (-i, N);
    i = i + Nz;
end

Ns = min (N - Nz, WAV.Nframe - i);
if (i >= 0 & Ns > 0)
    x(1:Nchan,Nz+1:Nz+Ns-1+1) = Amax * (wavread (WAV.Fname, [i+1 i+Ns-1+1]))';
end
%--------------------------------------------------------------------------
% End of File
%--------------------------------------------------------------------------
```

PQframeMOV.m

```
function PQframeMOV (i, MOVI,Version)
% Copy instantaneous MOV values to a new structure
% The output struct MOVC is a global.
% For most MOV's, they are just copied to the output structure.
% The exception is for the probability of detection, where the
% MOV's measure the maximum frequency-by-frequecy between channels.
%
% P. Kabal, McGill University, Copyright (C) 2003
% P. Kabal $Revision: 1.1 $  $Date: 2003/12/07 13:34:46 $

global MOVC

[Nchan,Nc] = size (MOVC.MDiff.Mt1B);
if (strcmp(Version,'Basic'))
        for (j = 1:Nchan)

        % Modulation differences
        MOVC.MDiff.Mt1B(j,i+1) = MOVI(j).MDiff.Mt1B;
        MOVC.MDiff.Mt2B(j,i+1) = MOVI(j).MDiff.Mt2B;
        MOVC.MDiff.Wt(j,i+1)   = MOVI(j).MDiff.Wt;

        % Noise loudness
        MOVC.NLoud.NL(j,i+1) = MOVI(j).NLoud.NL;

        % Total loudness
        MOVC.Loud.NRef(j,i+1)  = MOVI(j).Loud.NRef;
        MOVC.Loud.NTest(j,i+1) = MOVI(j).Loud.NTest;

        % Bandwidth
        MOVC.BW.BWRef(j,i+1) = MOVI(j).BW.BWRef;
        MOVC.BW.BWTest(j,i+1) = MOVI(j).BW.BWTest;

        % Noise-to-mask ratio
        MOVC.NMR.NMRavg(j,i+1) = MOVI(j).NMR.NMRavg;
```

```
        MOVC.NMR.NMRmax(j,i+1) = MOVI(j).NMR.NMRmax;

        % Added an element to the structure for segmental NMR
        MOVC.NMR.NMRseg(j,i+1) = MOVI(j).NMR.NMRseg;

        % Error harmonic structure
        MOVC.EHS.EHS(j,i+1) = MOVI(j).EHS.EHS;
       end

       % Probability of detection (collapse frequency bands)
       [MOVC.PD.Pc(i+1), MOVC.PD.Qc(i+1)] = PQ_ChanPD (MOVI);
else
    for (j = 1:Nchan)
            % Added an element to the structure for segmental NMR
            MOVC.NMR.NMRseg(j,i+1) = MOVI(j).NMR.NMRseg;

            % Error harmonic structure
            MOVC.EHS.EHS(j,i+1) = MOVI(j).EHS.EHS;
    end
end
%---------------------------------------
function [Pc, Qc] = PQ_ChanPD (MOVI)

Nc = length (MOVI(1).PD.p);
Nchan = length (MOVI);

Pr = 1;
Qc = 0;
if (Nchan > 1)
    for (m = 0:Nc-1)
        pbin = max (MOVI(1).PD.p(m+1), MOVI(2).PD.p(m+1));
        qbin = max (MOVI(1).PD.q(m+1), MOVI(2).PD.q(m+1));
        Pr = Pr * (1 - pbin);
        Qc = Qc + qbin;
    end
else
    for (m = 0:Nc-1)
        Pr = Pr * (1 - MOVI.PD.p(m+1));
        Qc = Qc + MOVI.PD.q(m+1);
    end
end

Pc = 1 - Pr;
%---------------------------------------------------------------------------
% End of File
%---------------------------------------------------------------------------
```

PQeval.m

```
function [MOVI, Fmem] = PQeval (xR, xT, Fmem)
% PEAQ - Process one frame with the FFT model
% P. Kabal, McGill University, Copyright (C) 2003
%
% Note: This code includes modifications specific to the PEAQ advanced
% version.

NF = 2048;
Version = 'Advanced';

% Windowed DFT
% Magnitude square of the FFT output is the output of this function
X2(1,:) = PQDFTFrame (xR);
X2(2,:) = PQDFTFrame (xT);

% Critical band grouping and frequency spreading
[EbN, Es] = PQ_excitCB (X2);

% Time domain smoothing => "Excitation patterns"
[Ehs(1,:), Fmem.TDS.Ef(1,:)] = PQ_timeSpread (Es(1,:), Fmem.TDS.Ef(1,:),...
    Version);
[Ehs(2,:), Fmem.TDS.Ef(2,:)] = PQ_timeSpread (Es(2,:), Fmem.TDS.Ef(2,:),...
```

```
    Version);

% Noise-to-mask ratios
MOVI.NMR = PQmovNMRB (EbN, Ehs(1,:),Version);

% Error harmonic structure
MOVI.EHS.EHS = PQmovEHS (xR, xT, X2);

%--------------------
function [EbN, Es] = PQ_excitCB (X2)

persistent W2 EIN
global SAMPL_FREQ;

NF = 2048;
Version = 'Advanced';
if (isempty (W2))
    Fs = SAMPL_FREQ;
    f = linspace (0, Fs/2, NF/2+1);

    % Outer ear weighting factor
    W2 = PQWOME (f);

    [Nc, fc] = PQCB (Version);
    EIN = PQIntNoise (fc);
end

% Allocate storage
XwN2 = zeros (1, NF/2+1);

% Outer and middle ear filtering
Xw2(1,:) = W2 .* X2(1,1:NF/2+1);
Xw2(2,:) = W2 .* X2(2,1:NF/2+1);

% Form the difference magnitude signal
% equation (12)
for (k = 0:NF/2)
    XwN2(k+1) = (Xw2(1,k+1) - 2 * sqrt (Xw2(1,k+1) * Xw2(2,k+1)) ...
                + Xw2(2,k+1));
end

% Group into partial critical bands
% This function returns the pitch mapped energies
Eb(1,:) = PQgroupCB (Xw2(1,:), Version);
Eb(2,:) = PQgroupCB (Xw2(2,:), Version);
EbN     = PQgroupCB (XwN2, Version);

% Add the internal noise term => "Pitch patterns"
E(1,:) = Eb(1,:) + EIN;
E(2,:) = Eb(2,:) + EIN;

% Critical band spreading => "Unsmeared excitation patterns"
Es(1,:) = PQspreadCB (E(1,:), Version);
Es(2,:) = PQspreadCB (E(2,:), Version);

%--------------------
function [Ehs, Ef] = PQ_timeSpread (Es, Ef, Ver)

% Sampling frequency
global SAMPL_FREQ;

persistent Nc a b

if (isempty (Nc))
    [Nc, fc] = PQCB (Ver); % Ver = 'Basic' in orig code
    Fs = SAMPL_FREQ;
    NF = 2048;
    Nadv = NF / 2;
    Fss = Fs / Nadv;
    t100 = 0.030;
    tmin = 0.008;
```

```
    [a, b] = PQtConst (t100, tmin, fc, Fss);
end

% Allocate storage
Ehs = zeros (1, Nc);

% Time domain smoothing
for (m = 0:Nc-1)
    Ef(m+1) = a(m+1) * Ef(m+1) + b(m+1) * Es(m+1);
    Ehs(m+1) = max(Ef(m+1), Es(m+1));
end
%------------------------------------------------------------------------
% End of File
%------------------------------------------------------------------------
```

PQDFTFrame.m

```
function X2 = PQDFTFrame (x)
% Calculate the DFT of a frame of data (NF values), returning the
% squared-magnitude DFT vector (NF/2 + 1 values)
% P. Kabal, McGill University, Copyright (C) 2003

% P. Kabal $Revision: 1.1 $  $Date: 2003/12/07 13:32:57 $

% Sampling frequency
global SAMPL_FREQ;

persistent hw

NF = 2048;      % Frame size (samples)

if (isempty (hw))
    Amax = 32768;
    fc = 1019.5;
    Fs = SAMPL_FREQ;
    Lp = 92;
    % Set up the window (including all gains)
    GL = PQ_GL (NF, Amax, fc/Fs, Lp);
    hw = GL * PQHannWin (NF);
end

% Window the data
xw = hw .* x;

% DFT (output is real followed by imaginary)
X = PQRFFT (xw, NF, 1);

% Squared magnitude
X2 = PQRFFTMSq (X, NF);

%---------------------------------------
function GL = PQ_GL (NF, Amax, fcN, Lp)
% Scaled Hann window, including loudness scaling

% Calculate the gain for the Hann Window
%  - level Lp (SPL) corresponds to a sine with normalized frequency
%    fcN and a peak value of Amax

W = NF - 1;
gp = PQ_gp (fcN, NF, W);
GL = 10^(Lp / 20) / (gp * Amax/4 * W);

%----------
function gp = PQ_gp (fcN, NF, W)
% Calculate the peak factor. The signal is a sinusoid windowed with
% a Hann window. The sinusoid frequency falls between DFT bins. The
% peak of the frequency response (on a continuous frequency scale) falls
% between DFT bins. The largest DFT bin value is the peak factor times
% the peak of the continuous response.
%  fcN - Normalized sinusoid frequency (0-1)
%  NF  - Frame (DFT) length samples
```

```
%  NW  - Window length samples

% Distance to the nearest DFT bin
df = 1 / NF;
k = floor (fcN / df);
dfN = min ((k+1) * df - fcN, fcN - k * df);

dfW = dfN * W;
gp = sin(pi * dfW) / (pi * dfW * (1 - dfW^2));
%-------------------------------------------------------------------------
% End of File
%-------------------------------------------------------------------------
```

PQdataBoundary.m

```
function Lim = PQdataBoundary (WAV, Nchan, StartS, Ns)
% Search for the data boundaries in a file
% StartS - starting sample frame
% Ns     - Number of sample frames

% P. Kabal $Revision: 1.1 $  $Date: 2003/12/07 13:34:10 $
% P. Kabal, McGill University, Copyright (C) 2003

PQ_L = 5;
Amax = 32768;
NBUFF = 2048;
PQ_ATHR = 200 * (Amax / 32768);

% Search from the beginning of the file
Lim(1) = -1;
is = StartS;
EndS = StartS + Ns - 1;
while (is <= EndS)
    Nf = min (EndS - is + 1, NBUFF);
    x = PQgetData (WAV, is, Nf);
    for (k = 0:Nchan-1)
        Lim(1) = max (Lim(1), PQ_DataStart (x(k+1,:), Nf, PQ_L, PQ_ATHR));
    end
    if (Lim(1) >= 0)
        Lim(1) = Lim(1) + is;
        break
    end
    is = is + NBUFF - (PQ_L-1);
end

% Search from the end of the file
% This loop is written as if it is going in a forward direction
%  - When the "forward" position is i, the "backward" position is
%    EndS - (i - StartS + 1) + 1
Lim(2) = -1;
is = StartS;
while (is <= EndS)
    Nf = min (EndS - is + 1, NBUFF);
    ie = is + Nf - 1;                   % Forward limits [is, ie]
    js = EndS - (ie - StartS + 1) + 1;  % Backward limits [js, js+Nf-1]
    x = PQgetData (WAV, js, Nf);
    for (k = 0:Nchan-1)
        Lim(2) = max (Lim(2), PQ_DataEnd (x(k+1,:), Nf, PQ_L, PQ_ATHR));
    end
    if (Lim(2) >= 0)
        Lim(2) = Lim(2) + js;
        break
    end
    is = is + NBUFF - (PQ_L-1);
end

% Sanity checks
if (~ ((Lim(1) >= 0 & Lim(2) >= 0) | (Lim(1) < 0 & Lim(2) < 0)))
    error ('>>> PQdataBoundary: limits have difference signs');
end
if (~(Lim(1) <= Lim(2)))
```

```
    error ('>>> PQdataBoundary: Lim(1) > Lim(2)');
end

if (Lim(1) < 0)
    Lim(1) = 0;
    Lim(2) = 0;
end

%----------
function ib = PQ_DataStart (x, N, L, Thr)

ib = -1;
s = 0;
M = min (N, L);
for (i = 0:M-1)
    s = s + abs (x(i+1));
end
if (s > Thr)
    ib = 0;
    return
end

for (i = 1:N-L)                                         % i is the first sample
    s = s + (abs (x(i+L-1+1)) - abs (x(i-1+1)));    % L samples apart
    if (s > Thr)
        ib = i;
        return
    end
end

%----------
function ie = PQ_DataEnd (x, N, L, Thr)

ie = -1;
s = 0;
M = min (N, L);
for (i = N-M:N-1)
    s = s + abs (x(i+1));
end
if (s > Thr)
    ie = N-1;
    return
end

for (i = N-2:-1:L-1)                                    % i is the last sample
    s = s + (abs (x(i-L+1+1)) - abs (x(i+1+1)));    % L samples apart
    if (s > Thr)
        ie = i;
        return
    end
end
%--------------------------------------------------------------------------
% End of File
%--------------------------------------------------------------------------
```

PQCB.m

```
function [Nc, fc, fl, fu, dz] = PQCB (Version)
% Critical band parameters for the FFT model
% Nc - number of frequency bands
%
% This code incorporates portions (with permission) of the PEAQ
% implementation by P. Kabal at McGill University [16]
%
% Note : This file contains tables specific to the advanced version in
% addition to the tables for the basic version

B = inline ('7 * asinh (f / 650)');
fL = 80;
fU = 18000;
```

```
% Critical bands - set up the tables
if (strcmp (Version, 'Basic'))
    dz = 1/4;
elseif (strcmp (Version, 'Advanced'))
    dz = 1/2;
else
    error ('PQCB: Invalid version');
end

zL = B(fL);
zU = B(fU);
Nc = ceil((zU - zL) / dz);
zl = zL + (0:Nc-1) * dz;
zu = min (zL + (1:Nc) * dz, zU);
zc = 0.5 * (zl + zu);

if (strcmp (Version, 'Basic'))
    fl = [  80.000,   103.445,   127.023,   150.762,   174.694, ...
           198.849,   223.257,   247.950,   272.959,   298.317, ...
           324.055,   350.207,   376.805,   403.884,   431.478, ...
           459.622,   488.353,   517.707,   547.721,   578.434, ...
           609.885,   642.114,   675.161,   709.071,   743.884, ...
           779.647,   816.404,   854.203,   893.091,   933.119, ...
           974.336,  1016.797,  1060.555,  1105.666,  1152.187, ...
          1200.178,  1249.700,  1300.816,  1353.592,  1408.094, ...
          1464.392,  1522.559,  1582.668,  1644.795,  1709.021, ...
          1775.427,  1844.098,  1915.121,  1988.587,  2064.590, ...
          2143.227,  2224.597,  2308.806,  2395.959,  2486.169, ...
          2579.551,  2676.223,  2776.309,  2879.937,  2987.238, ...
          3098.350,  3213.415,  3332.579,  3455.993,  3583.817, ...
          3716.212,  3853.817,  3995.399,  4142.547,  4294.979, ...
          4452.890,  4616.482,  4785.962,  4961.548,  5143.463, ...
          5331.939,  5527.217,  5729.545,  5939.183,  6156.396, ...
          6381.463,  6614.671,  6856.316,  7106.708,  7366.166, ...
          7635.020,  7913.614,  8202.302,  8501.454,  8811.450, ...
          9132.688,  9465.574,  9810.536, 10168.013, 10538.460, ...
         10922.351, 11320.175, 11732.438, 12159.670, 12602.412, ...
         13061.229, 13536.710, 14029.458, 14540.103, 15069.295, ...
         15617.710, 16186.049, 16775.035, 17385.420 ];
    fc = [  91.708,   115.216,   138.870,   162.702,   186.742, ...
           211.019,   235.566,   260.413,   285.593,   311.136, ...
           337.077,   363.448,   390.282,   417.614,   445.479, ...
           473.912,   502.950,   532.629,   562.988,   594.065, ...
           625.899,   658.533,   692.006,   726.362,   761.644, ...
           797.898,   835.170,   873.508,   912.959,   953.576, ...
           995.408,  1038.511,  1082.938,  1128.746,  1175.995, ...
          1224.744,  1275.055,  1326.992,  1380.623,  1436.014, ...
          1493.237,  1552.366,  1613.474,  1676.641,  1741.946, ...
          1809.474,  1879.310,  1951.543,  2026.266,  2103.573, ...
          2183.564,  2266.340,  2352.008,  2440.675,  2532.456, ...
          2627.468,  2725.832,  2827.672,  2933.120,  3042.309, ...
          3155.379,  3272.475,  3393.745,  3519.344,  3649.432, ...
          3784.176,  3923.748,  4068.324,  4218.090,  4373.237, ...
          4533.963,  4700.473,  4872.978,  5051.700,  5236.866, ...
          5428.712,  5627.484,  5833.434,  6046.825,  6267.931, ...
          6497.031,  6734.420,  6980.399,  7235.284,  7499.397, ...
          7773.077,  8056.673,  8350.547,  8655.072,  8970.639, ...
          9297.648,  9636.520,  9987.683, 10351.586, 10728.695, ...
         11119.490, 11524.470, 11944.149, 12379.066, 12829.775, ...
         13294.850, 13780.887, 14282.503, 14802.338, 15341.057, ...
         15899.345, 16477.914, 17077.504, 17690.045 ];
    fu = [ 103.445,   127.023,   150.762,   174.694,   198.849, ...
           223.257,   247.950,   272.959,   298.317,   324.055, ...
           350.207,   376.805,   403.884,   431.478,   459.622, ...
           488.353,   517.707,   547.721,   578.434,   609.885, ...
           642.114,   675.161,   709.071,   743.884,   779.647, ...
           816.404,   854.203,   893.091,   933.113,   974.336, ...
          1016.797,  1060.555,  1105.666,  1152.187,  1200.178, ...
          1249.700,  1300.816,  1353.592,  1408.094,  1464.392, ...
          1522.559,  1582.668,  1644.795,  1709.021,  1775.427, ...
          1844.098,  1915.121,  1988.587,  2064.590,  2143.227, ...
```

```
          2224.597,  2308.806,  2395.959,  2486.169,  2579.551, ...
          2676.223,  2776.309,  2879.937,  2987.238,  3098.350, ...
          3213.415,  3332.579,  3455.993,  3583.817,  3716.212, ...
          3853.348,  3995.399,  4142.547,  4294.979,  4452.890, ...
          4643.482,  4785.962,  4961.548,  5143.463,  5331.939, ...
          5527.217,  5729.545,  5939.183,  6156.396,  6381.463, ...
          6614.671,  6856.316,  7106.708,  7366.166,  7635.020, ...
          7913.614,  8202.302,  8501.454,  8811.450,  9132.688, ...
          9465.574,  9810.536, 10168.013, 10538.460, 10922.351, ...
         11320.175, 11732.438, 12159.670, 12602.412, 13061.229, ...
         13536.710, 14029.458, 14540.103, 15069.295, 15617.710, ...
         16186.049, 16775.035, 17385.420, 18000.000 ];
end

% Advanced version
if (strcmp (Version, 'Advanced'))
    fl = [80, 127.023,174.694,223.257,272.959,324.055,376.805,431.478,...
        488.353,547.721,609.885,675.161,743.884,816.404,893.091,...
        974.336,1060.555,1152.187,1249.7,1353.592,1464.392,...
        1582.668,1709.021,1844.098,1988.587,2143.227,2308.806,...
        2486.169,2676.223,2879.937,3098.35,3332.579,3583.817,...
        3853.348,4142.547,4452.89,4785.962,5143.463,5527.217,...
        5939.183,6381.463,6856.316,7366.166,7913.614,8501.454,...
        9132.688,9810.536,10538.46,11320.175,12159.67,13061.229,...
        14029.458,15069.295,16186.049,17385.42];

    fc = [103.445,150.762,198.849,247.95,298.317,350.207,403.884,...
      459.622,517.707,578.434,642.114,709.071,779.647,854.203,...
      933.119,1016.797,1105.666,1200.178,1300.816,1408.094,1522.559,...
      1644.795,1775.427,1915.121,2064.59,2224.597,2395.959,2579.551,...
      2776.309,2987.238,3213.415,3455.993,3716.212,3995.399,4294.979,...
      4616.482,4961.548,5331.939,5729.545,6156.396,6614.671,7106.708,...
      7635.02,8202.302,8811.45,9465.574,10168.013,10922.351,11732.438,...
      12602.412,13536.71,14540.103,15617.71,16775.035,17690.045];

    fu = [127.023,174.694,223.257,272.959,324.055,376.805,431.478,488.353,...
        547.721,609.885,675.161,743.884,816.404,893.091,974.336,...
        1060.555,1152.187,1249.7,1353.592,1464.392,1582.668,1709.021,...
        1844.098,1988.587,2143.227,2308.806,2486.169,2676.223,2879.937,...
        3098.35,3332.579,3583.817,3853.348,4142.547,4452.89,4785.962,...
        5143.463,5527.217,5939.183,6381.463,6856.316,7366.166,...
        7913.614,8501.454,9132.688,9810.536,10538.46,11320.175,12159.67,...
        13061.229,14029.458,15069.295,16186.049,17385.42,18000];
end
%--------------------------------------------------------------------------
% End of File
%--------------------------------------------------------------------------
```

PQavgMOVB.m

```
function [MOV_1, MOV_2] = PQavgMOVB (MOVC, Nchan, Nwup,Mod,Ver)
% Time average MOV precursors
% P. Kabal, McGill University, Copyright (C) 2003
%
% P. Kabal $Revision: 1.1 $  $Date: 2003/12/07 13:34:46 $


% Sampling frequency
global SAMPL_FREQ;

Fs = SAMPL_FREQ;
NF = 2048;
if (strcmp (Mod, 'FFT'))
    Nadv = NF / 2;
else
    Nadv = 192;
end
Fss = Fs / Nadv;
tdel = 0.5;
tex = 0.050;
```

```
if (strcmp (Ver, 'Basic'))
    % BandwidthRefB, BandwidthTestB
    [MOV(0+1), MOV(1+1)] = PQ_avgBW (MOVC.BW);

    % Total NMRB, RelDistFramesB
    [MOV(2+1), MOV(10+1)] = PQ_avgNMRB (MOVC.NMR);

    % WinModDiff1B, AvgModDiff1B, AvgModDiff2B
    N500ms = ceil (tdel * Fss);
    Ndel = max (0, N500ms - Nwup); % Diff is between frame number

    [MOV(3+1), MOV(6+1), MOV(7+1)] = PQ_avgModDiffB (Ndel, MOVC.MDiff);

    % RmsNoiseLoudB
        N50ms = ceil (tex * Fss);
        Nloud = PQloudTest (MOVC.Loud);
        Ndel = max (Nloud + N50ms, Ndel);

    MOV(8+1) = PQ_avgNLoudB (Ndel, MOVC.NLoud);

    % ADBB, MFPDB
    [MOV(4+1), MOV(9+1)] = PQ_avgPD (MOVC.PD);

    % EHSB
    MOV(5+1) = PQ_avgEHS (MOVC.EHS);
else
    % NMRseg
    MOV_1 = PQ_avgNMRseg (MOVC.NMR);
    % EHSB
    MOV_2 = PQ_avgEHS (MOVC.EHS);
end
%----------------------------------------
function EHSB = PQ_avgEHS (EHS)

[Nchan, Np] = size (EHS.EHS);

s = 0;
for (j = 0:Nchan-1)
    s = s + PQ_LinPosAvg (EHS.EHS(j+1,:));
end
EHSB = 1000 * s / Nchan;


%----------------------------------------
function [ADBB, MFPDB] = PQ_avgPD (PD)

global PQopt

c0 = 0.9;
if (isempty (PQopt))
    c1 = 1;
else
    c1 = PQopt.PDfactor;
end

N = length (PD.Pc);
Phc = 0;
Pcmax = 0;
Qsum = 0;
nd = 0;
for (i = 0:N-1)
    Phc = c0 * Phc + (1 - c0) * PD.Pc(i+1);
    Pcmax = max (Pcmax * c1, Phc);

    if (PD.Pc(i+1) > 0.5)
        nd = nd + 1;
        Qsum = Qsum + PD.Qc(i+1);
    end
end

if (nd == 0)
```

```
    ADBB = 0;
elseif (Qsum > 0)
    ADBB = log10 (Qsum / nd);
else
    ADBB = -0.5;
end

MFPDB = Pcmax;

%---------------------------------------
function [TotalNMRB, RelDistFramesB] = PQ_avgNMRB (NMR)

[Nchan, Np] = size (NMR.NMRavg);
Thr = 10^(1.5 / 10);

s = 0;
for (j = 0:Nchan-1)
    s = s + 10 * log10 (PQ_LinAvg (NMR.NMRavg(j+1,:)));
end
TotalNMRB = s / Nchan;

s = 0;
for (j = 0:Nchan-1)
    s = s + PQ_FractThr (Thr, NMR.NMRmax(j+1,:));
end
RelDistFramesB = s / Nchan;

%---------------------------------------
function [NMRseg] = PQ_avgNMRseg (NMR)

[Nchan, Np] = size (NMR.NMRseg);

s = 0;
for (j = 0:Nchan-1)
    s = s + PQ_LinAvg (NMR.NMRseg(j+1,:));
end
NMRseg = s / Nchan;

%---------------------------------------
function [BandwidthRefB, BandwidthTestB] = PQ_avgBW (BW)

[Nchan, Np] = size (BW.BWRef);

sR = 0;
sT = 0;
for (j = 0:Nchan-1)
    sR = sR + PQ_LinPosAvg (BW.BWRef(j+1,:));
    sT = sT + PQ_LinPosAvg (BW.BWTest(j+1,:));
end
BandwidthRefB  = sR / Nchan;
BandwidthTestB = sT / Nchan;

%---------------------------------------
function    [WinModDiff1B,    AvgModDiff1B,    AvgModDiff2B]    =    PQ_avgModDiffB    (Ndel,
MDiff,Mod,Version)

% Sampling frequency
global SAMPL_FREQ;

Fs = SAMPL_FREQ;
if (strcmp (Mod, 'FFT'))
    NF = 2048;
    Nadv = NF / 2;
else
    Nadv = 192;
end
Fss = Fs / Nadv;
tavg = 0.1;

[Nchan, Np] = size (MDiff.Mt1B);
```

```
% Sliding window average - delayed average
L = floor (tavg * Fss);     % 100 ms sliding window length
s = 0;
for (j = 0:Nchan-1)
    s = s + PQ_WinAvg (L, MDiff.Mt1B(j+1,Ndel+1:Np-1+1));
end
WinModDiff1B = s / Nchan;

% Weighted linear average - delayed average
s = 0;
for (j = 0:Nchan-1)
    s = s + PQ_WtAvg (MDiff.Mt1B(j+1,Ndel+1:Np-1+1), MDiff.Wt(j+1,Ndel+1:Np-1+1));
end
AvgModDiff1B = s / Nchan;

% Weighted linear average - delayed average
s = 0;
for (j = 0:Nchan-1)
    s = s + PQ_WtAvg (MDiff.Mt2B(j+1,Ndel+1:Np-1+1), MDiff.Wt(j+1,Ndel+1:Np-1+1));
end
AvgModDiff2B = s / Nchan;

%----------------------------------------
function RmsNoiseLoudB = PQ_avgNLoudB (Ndel, NLoud)

[Nchan, Np] = size (NLoud.NL);

% RMS average - delayed average and loudness threshold
s = 0;
for (j = 0:Nchan-1)
    s = s + PQ_RMSAvg (NLoud.NL(j+1,Ndel+1:Np-1+1));
end
RmsNoiseLoudB = s / Nchan;

%----------------------------------
% Average values values, omitting values which are negative
function s = PQ_LinPosAvg (x)

N = length(x);

Nv = 0;
s = 0;
for (i = 0:N-1)
    if (x(i+1) >= 0)
        s = s + x(i+1);
        Nv = Nv + 1;
    end
end

if (Nv > 0)
    s = s / Nv;
end

%----------
% Fraction of values above a threshold
function Fd = PQ_FractThr (Thr, x)

N = length (x);

Nv = 0;
for (i = 0:N-1)
    if (x(i+1) > Thr)
        Nv = Nv + 1;
    end
end

if (N > 0)
    Fd = Nv / N;
else
    Fd = 0;
end
```

```
%-----------
% Sliding window (L samples) average
function s = PQ_WinAvg (L, x)

N = length (x);

s = 0;
for (i = L-1:N-1)
    t = 0;
    for (m = 0:L-1)
        t = t + sqrt (x(i-m+1));
    end
    s = s + (t / L)^4;
end

if (N >= L)
    s = sqrt (s / (N - L + 1));
end

%----------
% Weighted average
function s = PQ_WtAvg (x, W)

N = length (x);

s = 0;
sW = 0;
for (i = 0:N-1)
    s = s + W(i+1) * x(i+1);
    sW = sW + W(i+1);
end

if (N > 0)
    s = s / sW;
end

%----------
% Linear average
function LinAvg = PQ_LinAvg (x)

N = length (x);
s = 0;
for (i = 0:N-1)
    s = s + x(i+1);
end

LinAvg = s / N;

%----------
% Square root of average of squared values
function RMSAvg = PQ_RMSAvg (x)

N = length (x);
s = 0;
for (i = 0:N-1)
    s = s + x(i+1)^2;
end

if (N > 0)
    RMSAvg = sqrt(s / N);
else
    RMSAvg = 0;
end
%--------------------------------------------------------------------------
% End of File
%--------------------------------------------------------------------------
```

peripheral_model.m

```
function [E, E2, E_forward_mask] = peripheral_model(filter_real,filter_imag,...
    E_forward_mask);
% peripheral_model.m
%
%   This function applies the peripheral ear processing to the input data
%
% Author: Rahul Vanam

% Number of Filters in the Filter bank
NUM_FILTERS = 40;

% inline the filter bank table to include list of center frequencies of
% each filter
filter_bank_table;

% Outer ear and middle ear filtering
[filter_out_real, filter_out_imag] = out_mid_ear_filt(filter_real,filter_imag);

% Frequency domain spreading
[pattern_real, pattern_imag] = freq_spread(filter_out_real,filter_out_imag);

clear filter_out_real;
clear filter_out_imag;
clear filter_out_2;

% Rectification
E0 = zeros(size(pattern_real));
for i = 1:NUM_FILTERS
    E0(i,:) = pattern_real(i,:) .^ 2 + pattern_imag(i,:) .^ 2;
end

clear pattern_real;
clear pattern_imag;

% Backward masking (Time domain smearing -1)
E1 = backward_mask(E0);

% Adding internal noise
% unsmeared excitation pattern
E2 = zeros(size(E1));
for i = 1:NUM_FILTERS
    E_thresh = power(10, (0.4 * 0.364 * (fc(i)/10^3) ^ (-0.8)));
    E2(i,:) = E1(i,:) + E_thresh;
end

% Forward masking (Time domain smearing -2)
[E, E_forward_mask] = forward_mask(E2,E_forward_mask);
%------------------------------------------------------------------------
% End of File
%------------------------------------------------------------------------
```

pre_process.m

```
function [mod_ref,mod_test,E_loud_ref,E_P_ref,E_P_test,N_ref,N_test] = ...
    pre_process(E_ref, E_test, E2_ref, E2_test,chan);
% pre_process.m
%
%   This function is used to pre-process the excitation pattern
%
%   Author: Rahul Vanam

%------------------------------------------------------------------------
% Level and Pattern adaptation is applied to adapt the reference and the
% test signal to each other for possible difference in levels
% (linear distortion) and possible absence of data in one of the signals
% (non-linear distortion).
%------------------------------------------------------------------------
[E_P_ref, E_P_test] = level_pattern_adapt(E_ref, E_test, chan);
```

```
% Modulation of the envelope at each filter output
[mod_ref, E_loud_ref] = modulation(E2_ref,chan,'ref');
[mod_test, E_loud_test] = modulation(E2_test,chan,'test');

% Computation of overall loudness
N_ref = overall_loudness(E_ref);
N_test = overall_loudness(E_test);
%--------------------------------------------------------------------------
% End of File
%--------------------------------------------------------------------------
```

PQWOME.m

```
function W2 = PQWOME (f)
% Generate the weighting for the outer & middle ear filtering
% Note: The output is a magnitude-squared vector
% P. Kabal, McGill University, Copyright (C) 2003
%
% P. Kabal $Revision: 1.1 $  $Date: 2003/12/07 13:34:11 $

N = length (f);
for (k = 0:N-1)
    fkHz = f(k+1) / 1000;
    AdB = -2.184 * fkHz^(-0.8) + 6.5 * exp(-0.6 * (fkHz - 3.3)^2) ...
          - 0.001 * fkHz^(3.6);
    W2(k+1) = 10^(AdB / 10);
end
%--------------------------------------------------------------------------
% End of File
%--------------------------------------------------------------------------
```

PQwavFilePar.m

```
function WAV = PQwavFilePar (File)
% Print a WAVE file header, pick up the file parameters
% P. Kabal, McGill University, Copyright (C) 2003

% P. Kabal $Revision: 1.1 $  $Date: 2003/12/07 13:34:11 $

persistent iB

if (isempty (iB))
    iB = 0;
else
    iB = mod (iB + 1, 2);   % Only two files can be "active" at a time
end

[size WAV.Fs Nbit] = wavread (File, 'size');
WAV.Fname = File;
WAV.Nframe = size(1);
WAV.Nchan = size(2);
WAV.iB = iB;   % Buffer number

% Initialize the buffer
PQgetData (WAV, 0, 0);

fprintf (' WAVE file: %s\n', File);
if (WAV.Nchan == 1)
    fprintf ('   Number of samples : %d (%.4g s)\n', WAV.Nframe, WAV.Nframe / WAV.Fs);
else
    fprintf ('   Number of frames  : %d (%.4g s)\n', WAV.Nframe, WAV.Nframe / WAV.Fs);
end
fprintf ('   Sampling frequency: %g\n', WAV.Fs);
fprintf ('   Number of channels: %d (%d-bit integer)\n', WAV.Nchan, Nbit);
%--------------------------------------------------------------------------
% End of File
%--------------------------------------------------------------------------
```

PQtConst.m

```
function [a, b] = PQtConst (t100, tmin, f , Fs)
% Calculate the difference equation parameters. The time
% constant of the difference equation depends on the center
% frequencies.
% P. Kabal, McGill University, Copyright (C) 2003
%
% P. Kabal $Revision: 1.1 $  $Date: 2003/12/07 13:34:11 $

N = length (f);
for (m = 0:N-1)
    t = tmin + (100 / f(m+1)) * (t100 - tmin);
    a(m+1) = exp (-1 / (Fs * t));
    b(m+1) = (1 - a(m+1));
end
%------------------------------------------------------------------------
% End of File
%------------------------------------------------------------------------
```

PQspreadCB.m

```
function Es = PQspreadCB (E, Ver)
% Spread an excitation vector (pitch pattern) - FFT model
% Both E and Es are powers
% P. Kabal, McGill University, Copyright (C) 2003

% P. Kabal $Revision: 1.1 $  $Date: 2003/12/07 13:32:58 $

persistent Bs Version

if (~ strcmp (Ver, Version))
    Version = Ver;
    Nc = length (E);
    Bs = PQ_SpreadCB (ones(1,Nc), ones(1,Nc), Version);
end

Es = PQ_SpreadCB (E, Bs, Version);

%-------------------------
function Es = PQ_SpreadCB (E, Bs, Ver);

persistent Nc dz fc aL aUC Version

% Power law for addition of spreading
e = 0.4;

if (~ strcmp (Ver, Version))
    Version  = Ver;
    [Nc, fc, fl, fu, dz] = PQCB (Version);
end

% Allocate storage
aUCEe = zeros (1, Nc);
Ene = zeros (1, Nc);
Es = zeros (1, Nc);

% Calculate energy dependent terms
aL = 10^(-2.7 * dz);
for (m = 0:Nc-1)
    aUC = 10^((-2.4 - 23 / fc(m+1)) * dz);
    aUCE = aUC * E(m+1)^(0.2 * dz);
    gIL = (1 - aL^(m+1)) / (1 - aL);
    gIU = (1 - aUCE^(Nc-m)) / (1 - aUCE);
    En = E(m+1) / (gIL + gIU - 1);
    aUCEe(m+1) = aUCE^e;
    Ene(m+1) = En^e;
end

% Lower spreading
Es(Nc-1+1) = Ene(Nc-1+1);
```

```
aLe = aL^e;
for (m = Nc-2:-1:0)
    Es(m+1) = aLe * Es(m+1+1) + Ene(m+1);
end

% Upper spreading i > m
for (m = 0:Nc-2)
    r = Ene(m+1);
    a = aUCEe(m+1);
    for (i = m+1:Nc-1)
       r = r * a;
       Es(i+1) = Es(i+1) + r;
    end
end

for (i = 0:Nc-1)
    Es(i+1) = (Es(i+1))^(1/e) / Bs(i+1);
end
%------------------------------------------------------------------------
% End of File
%------------------------------------------------------------------------
```

PQRFFTMSq.m

```
function X2 = PQRFFTMSq (X, N)
% Calculate the magnitude squared frequency response from the
% DFT values corresponding to a real signal (assumes N is even)
% P. Kabal, McGill University, Copyright (C) 2003
%
% P. Kabal $Revision: 1.1 $  $Date: 2003/12/07 13:34:11 $

X2 = zeros (1, N/2+1);

X2(0+1) = X(0+1)^2;
for (k = 1:N/2-1)
    X2(k+1) = X(k+1)^2 + X(N/2+k+1)^2;
end
X2(N/2+1) = X(N/2+1)^2;
%------------------------------------------------------------------------
% End of File
%------------------------------------------------------------------------
```

PQRFFT.m

```
function X = PQRFFT (x, N, ifn)
% Calculate the DFT of a real N-point sequence or the inverse
% DFT corresponding to a real N-point sequence.
% ifn > 0, forward transform
%          input x(n)  - N real values
%          output X(k) - The first N/2+1 points are the real
%            parts of the transform, the next N/2-1 points
%            are the imaginary parts of the transform. However
%            the imaginary part for the first point and the
%            middle point which are known to be zero are not
%            stored.
% ifn < 0, inverse transform
%          input X(k) - The first N/2+1 points are the real
%            parts of the transform, the next N/2-1 points
%            are the imaginary parts of the transform. However
%            the imaginary part for the first point and the
%            middle point which are known to be zero are not
%            stored.
%          output x(n) - N real values
% P. Kabal, McGill University, Copyright (C) 2003
%
% P. Kabal $Revision: 1.1 $  $Date: 2003/12/07 13:34:11 $

if (ifn > 0)
    X = fft (x, N);
    XR = real(X(0+1:N/2+1));
    XI = imag(X(1+1:N/2-1+1));
```

```
    X = [XR XI];
else
    xR = [x(0+1:N/2+1)];
    xI = [0 x(N/2+1+1:N-1+1) 0];
    x = complex ([xR xR(N/2-1+1:-1:1+1)], [xI -xI(N/2-1+1:-1:1+1)]);
    X = real (ifft (x, N));
end
%--------------------------------------------------------------------------
% End of File
%--------------------------------------------------------------------------
```

PQprtMOV.m

```
function PQprtMOV (MOV, ODG)
% Print MOV values (PEAQ Basic version)
% P. Kabal, McGill University, Copyright (C) 2003

% P. Kabal $Revision: 1.1 $  $Date: 2003/12/07 13:34:47 $

N = length (MOV);
PQ_NMOV_B = 11;
PQ_NMOV_A = 5;

fprintf ('Model Output Variables:\n');
if (N == PQ_NMOV_B)
    fprintf ('   BandwidthRefB: %g\n', MOV(1));
    fprintf ('  BandwidthTestB: %g\n', MOV(2));
    fprintf ('      Total NMRB: %g\n', MOV(3));
    fprintf ('    WinModDiff1B: %g\n', MOV(4));
    fprintf ('            ADBB: %g\n', MOV(5));
    fprintf ('            EHSB: %g\n', MOV(6));
    fprintf ('    AvgModDiff1B: %g\n', MOV(7));
    fprintf ('    AvgModDiff2B: %g\n', MOV(8));
    fprintf ('   RmsNoiseLoudB: %g\n', MOV(9));
    fprintf ('           MFPDB: %g\n', MOV(10));
    fprintf ('  RelDistFramesB: %g\n', MOV(11));
elseif (N == PQ_NMOV_A)
    fprintf ('        RmsModDiffA: %g\n', MOV(1));
    fprintf ('  RmsNoiseLoudAsymA: %g\n', MOV(2));
    fprintf ('     Segmental NMRB: %g\n', MOV(3));
    fprintf ('              EHSB: %g\n', MOV(4));
    fprintf ('        AvgLinDistA: %g\n', MOV(5));
else
    error ('Invalid number of MOVs');
end

fprintf ('Objective Difference Grade: %.3f\n', ODG);

return;
%--------------------------------------------------------------------------
% End of File
%--------------------------------------------------------------------------
```

PQnNetB.m

```
function [DI, ODG] = PQnNetB (MOV)
% Neural net to get the final ODG
% P. Kabal, McGill University, Copyright (C) 2003

% P. Kabal $Revision: 1.1 $  $Date: 2003/12/07 13:27:44 $

persistent amin amax wx wxb wy wyb bmin bmax I J CLIPMOV
global PQopt

if (isempty (amin))
    I = length (MOV);
    if (I == 11)
        [amin, amax, wx, wxb, wy, wyb, bmin, bmax] = NNetPar ('Basic');
    else
        [amin, amax, wx, wxb, wy, wyb, bmin, bmax] = NNetPar ('Advanced');
    end
```

```
    [I, J] = size (wx);
end

sigmoid = inline ('1 / (1 + exp(-x))');

% Scale the MOV's
Nclip = 0;
MOVx = zeros (1, I);
for (i = 0:I-1)
    MOVx(i+1) = (MOV(i+1) - amin(i+1)) / (amax(i+1) - amin(i+1));
    if (~ isempty (PQopt) & PQopt.ClipMOV ~= 0)
        if (MOVx(i+1) < 0)
            MOVx(i+1) = 0;
            Nclip = Nclip + 1;
        elseif (MOVx(i+1) > 1)
            MOVx(i+1) = 1;
            Nclip = Nclip + 1;
        end
    end
end
if (Nclip > 0)
    fprintf ('>>> %d MOVs clipped\n', Nclip);
end

% Neural network
DI = wyb;
for (j = 0:J-1)
    arg = wxb(j+1);
    for (i = 0:I-1)
        arg = arg + wx(i+1,j+1) * MOVx(i+1);
    end
    DI = DI + wy(j+1) * sigmoid (arg);
end
% Rahul
disp('di value :');
disp(DI);

ODG = bmin + (bmax - bmin) * sigmoid (DI);

function [amin, amax, wx, wxb, wy, wyb, bmin, bmax] = NNetPar (Version)

if (strcmp (Version, 'Basic'))
    amin = ...
        [393.916656,  361.965332,  -24.045116,    1.110661,  -0.206623, ...
           0.074318,    1.113683,    0.950345,    0.029985,   0.000101, ...
           0];
    amax = ...
        [921,         881.131226,   16.212030,  107.137772,   2.886017, ...
          13.933351,   63.257874, 1145.018555,   14.819740,   1,        ...
           1];
    wx = ...
        [ [ -0.502657,  0.436333,   1.219602 ];
          [  4.307481,  3.246017,   1.123743 ];
          [  4.984241, -2.211189,  -0.192096 ];
          [  0.051056, -1.762424,   4.331315 ];
          [  2.321580,  1.789971,  -0.754560 ];
          [ -5.303901, -3.452257, -10.814982 ];
          [  2.730991, -6.111805,   1.519223 ];
          [  0.624950, -1.331523,  -5.955151 ];
          [  3.102889,  0.871260,  -5.922878 ];
          [ -1.051468, -0.939882,  -0.142913 ];
          [ -1.804679, -0.503610,  -0.620456 ] ];
    wxb = ...
        [ -2.518254,  0.654841, -2.207228 ];
    wy = ...
        [ -3.817048,  4.107138,  4.629582 ];
    wyb = -0.307594;
    bmin = -3.98;
    bmax = 0.22;
else
    amin = ...
```

```
        [   13.298751,  0.041073,  -25.018791,  0.061560,   0.024523 ];
    amax = ...
        [ 2166.5,      13.24326,    13.46708,  10.226771,  14.224874 ];
    wx = ...
        [ [  21.211773, -39.913052, -1.382553, -14.545348,  -0.320899 ];
          [  -8.981803,  19.956049,  0.935389,  -1.686586,  -3.238586 ];
          [   1.633830,  -2.877505, -7.442935,   5.606502,  -1.783120 ];
          [   6.103821,  19.587435, -0.240284,   1.088213,  -0.511314 ];
          [  11.556344,   3.892028,  9.720441,  -3.287205, -11.031250 ] ];
    wxb = ...
        [   1.330890,   2.686103,    2.096598, -1.327851,   3.087055 ];
    wy = ...
        [ -4.696996,   -3.289959,    7.004782,  6.651897,   4.009144 ];
    wyb = -1.360308;
    bmin = -3.98;
    bmax = 0.22;
end
%-------------------------------------------------------------------------
% End of File
%-------------------------------------------------------------------------
```

PQmovPD.m

```
function PD = PQmovPD (Ehs)
% Probability of detection
% P. Kabal, McGill University, Copyright (C) 2003

% P. Kabal $Revision: 1.1 $  $Date: 2003/12/07 13:34:47 $

Nc = length (Ehs);

% Allocate storage
PD.p = zeros (1, Nc);
PD.q = zeros (1, Nc);

persistent c g d1 d2 bP bM

if (isempty (c))
    c = [-0.198719 0.0550197 -0.00102438 5.05622e-6 9.01033e-11];
    d1 = 5.95072;
    d2 = 6.39468;
    g = 1.71332;
    bP = 4;
    bM = 6;
end

for (m = 0:Nc-1)
    EdBR = 10 * log10 (Ehs(1,m+1));
    EdBT = 10 * log10 (Ehs(2,m+1));
    edB = EdBR - EdBT;
    if (edB > 0)
        L = 0.3 * EdBR + 0.7 * EdBT;
        b = bP;
    else
        L = EdBT;
        b = bM;
    end
    if (L > 0)
        s = d1 * (d2 / L)^g ...
            + c(1) + L * (c(2) + L * (c(3) + L * (c(4) + L * c(5))));
    else
        s = 1e30;
    end
    PD.p(m+1) = 1 - 0.5^((edB / s)^b);        % Detection probability
    PD.q(m+1) = abs (fix(edB)) / s;           % Steps above threshold
end
%-------------------------------------------------------------------------
% End of File
%-------------------------------------------------------------------------
```

PQmovNMRB.m

```
function NMR = PQmovNMRB (EbN, Ehs,Ver)
% Noise-to-mask ratio - Basic version
% NMR(1) average NMR
% NMR(2) max NMR
% P. Kabal, McGill University, Copyright (C) 2003

% P. Kabal $Revision: 1.1 $  $Date: 2003/12/07 13:34:47 $

persistent Nc gm
Ver = 'Advanced';

if (isempty (Nc))
    [Nc, fc, fl, fu, dz] = PQCB (Ver);
    gm = PQ_MaskOffset (dz, Nc);
end

NMR.NMRmax = 0;
s = 0;
for (m = 0:Nc-1)
    NMRm = EbN(m+1) / (gm(m+1) * Ehs(m+1));
    s = s + NMRm;
    if (NMRm > NMR.NMRmax)
        NMR.NMRmax = NMRm;
    end
end
NMR.NMRavg = s / Nc;

% Rahul: NMR_local
NMR_local = 10 * log10(NMR.NMRavg);
NMR.NMRseg = sum(NMR_local)/length(NMR_local);
%---------------------------------------
function gm = PQ_MaskOffset (dz, Nc)

% Rahul: Computation of equation (26)
for (m = 0:Nc-1)
    if (m <= 12 / dz)
        mdB = 3;
    else
        mdB = 0.25 * m * dz;
    end
    gm(m+1) = 10^(-mdB / 10);
end
%------------------------------------------------------------------------
% End of File
%------------------------------------------------------------------------
```

PQmovNLoudB.m

```
function NL = PQmovNLoudB (M, EP)
% Noise Loudness
% P. Kabal, McGill University, Copyright (C) 2003

% P. Kabal $Revision: 1.1 $  $Date: 2003/12/07 13:34:47 $

persistent Nc Et

if (isempty (Nc))
    [Nc, fc] = PQCB ('Basic');
    Et = PQIntNoise (fc);
end

% Parameters
alpha = 1.5;
TF0 = 0.15;
S0 = 0.5;
NLmin = 0;
e = 0.23;

s = 0;
```

```
for (m = 0:Nc-1)
    sref  = TF0 * M(1,m+1) + S0;
    stest = TF0 * M(2,m+1) + S0;
    beta = exp (-alpha * (EP(2,m+1) - EP(1,m+1)) / EP(1,m+1));
    a = max (stest * EP(2,m+1) - sref * EP(1,m+1), 0);
    b = Et(m+1) + sref * EP(1,m+1) * beta;
    s = s + (Et(m+1) / stest)^e * ((1 + a / b)^e - 1);
end

NL = (24 / Nc) * s;
if (NL < NLmin)
    NL = 0;
end
%--------------------------------------------------------------------------
% End of File
%--------------------------------------------------------------------------
```

PQmovModDiffB.m

```
function MDiff = PQmovModDiffB (M, ERavg,Mod,Version)
% Modulation difference related MOV precursors (Basic version)
% P. Kabal, McGill University Copyright (C) 2003

% P. Kabal $Revision: 1.1 $  $Date: 2003/12/07 13:34:46 $

persistent Nc Ete

if (isempty (Nc))
    e = 0.3;
    if (strcmp (Mod, 'FFT'))
        [Nc, fc] = PQCB (Version);
    else % Filter_bank
        [Nc, fc] = PQFB;
    end
    Et = PQIntNoise (fc);
    for (m = 0:Nc-1)
        Ete(m+1) = Et(m+1)^e;
    end
end

if (strcmp (Mod, 'FFT'))
       % Parameters
       negWt2B = 0.1;
       offset1B = 1.0;
       offset2B = 0.01;
       levWt = 100;
else
    offset = 1;
    levWt_FB = 1;
end

s1B = 0;
s2B = 0;
Wt = 0;
s1 = 0;
for (m = 0:Nc-1)
    if (M(1,m+1) > M(2,m+1))
        num1B = M(1,m+1) - M(2,m+1);
        num2B = negWt2B * num1B;
    else
        num1B = M(2,m+1) - M(1,m+1);
        num2B = num1B;
    end
    if (strcmp (Mod, 'FFT'))
        MD1B = num1B / (offset1B + M(1,m+1));
        MD2B = num2B / (offset2B + M(1,m+1));
        s1B = s1B + MD1B;
        s2B = s2B + MD2B;
        Wt = Wt + ERavg(m+1) / (ERavg(m+1) + levWt * Ete(m+1));
    else % Filter bank
        MD = num1B / (offset + M(1,m+1));
```

```
        s1 = s1 + MD;
        Wt = Wt + ERavg(m+1) / (ERavg(m+1) + levWt_FB * Ete(m+1));
    end
end

if (strcmp (Mod, 'FFT'))
    MDiff.Mt1B = (100 / Nc) * s1B;
    MDiff.Mt2B = (100 / Nc) * s2B;
    MDiff.Wt = Wt;
else
    MDiff.Mt = (100 / Nc) * s1;
    MDiff.Wt = Wt;
end
%--------------------------------------------------------------------------
% End of File
%--------------------------------------------------------------------------
```

PQmovEHS.m

```
function EHS = PQmovEHS (xR, xT, X2)
% Calculate the EHS MOV values
%
% P. Kabal, McGill University Copyright (C) 2004

% P. Kabal $Revision: 1.2 $  $Date: 2004/02/05 04:26:19 $

global SAMPL_FREQ; % Sampling frequency

persistent NF Nadv NL M Hw

if (isempty (NL))
    NF = 2048;
    Nadv = NF / 2;
    Fs = SAMPL_FREQ;
    Fmax = 9000;
    NL = 2^(PQ_log2(NF * Fmax / Fs));
    M = NL;
    Hw = (1 / M) * sqrt(8 / 3) * PQHannWin (M);
end

EnThr = 8000;
kmax = NL + M - 1;

EnRef  = xR(Nadv+1:NF-1+1) * xR(Nadv+1:NF-1+1)';
EnTest = xT(Nadv+1:NF-1+1) * xT(Nadv+1:NF-1+1)';

% Set the return value to be negative for small energy frames
if (EnRef < EnThr & EnTest < EnThr)
    EHS = -1;
    return;
end

% Allocate storage
D = zeros (1, kmax);

% Differences of log values
for (k = 0:kmax-1)
    % RV: If the denominator is zero the value of D is set to zero. (?)
    if (X2(1,k+1) ~= 0)
        D(k+1) = log (X2(2,k+1) / X2(1,k+1));
    else
        D(k+1) = 0;
    end
end

% Correlation computation
C = PQ_Corr (D, NL, M);

% Normalize the correlations
Cn = PQ_NCorr (C, D, NL, M);
Cnm = (1 / NL) * sum (Cn(1:NL));
```

```
% Window the correlation
Cw = Hw .* (Cn - Cnm);

% DFT
cp = PQRFFT (Cw, NL, 1);

% Squared magnitude
c2 = PQRFFTMSq (cp, NL);

% Search for a peak after a valley
EHS = PQ_FindPeak (c2, NL/2+1);

%---------------------------------------
function log2 = PQ_log2 (a)

log2 = 0;
m = 1;
while (m < a)
    log2 = log2 + 1;
    m = 2 * m;
end
log2 = log2 - 1;

%----------
function C = PQ_Corr (D, NL, M)
% Correlation calculation
% Calculate the correlation indirectly
NFFT = 2 * NL;
D0 = [D(1:M) zeros(1,NFFT-M)];
D1 = [D(1:M+NL-1) zeros(1,NFFT-(M+NL-1))];

% DFTs of the zero-padded sequences
d0 = PQRFFT (D0, NFFT, 1);
d1 = PQRFFT (D1, NFFT, 1);

% Multiply (complex) sequences
dx(0+1) = d0(0+1) * d1(0+1);
for (n = 1:NFFT/2-1)
    m = NFFT/2 + n;
    dx(n+1) = d0(n+1) * d1(n+1) + d0(m+1) * d1(m+1);
    dx(m+1) = d0(n+1) * d1(m+1) - d0(m+1) * d1(n+1);
end
dx(NFFT/2+1) = d0(NFFT/2+1) * d1(NFFT/2+1);

% Inverse DFT
Cx = PQRFFT (dx, NFFT, -1);
C = Cx(1:NL);

%----------
function Cn = PQ_NCorr (C, D, NL, M)
% Normalize the correlation

Cn = zeros (1, NL);

s0 = C(0+1);
sj = s0;
Cn(0+1) = 1;
for (i = 1:NL-1)
    sj = sj + (D(i+M-1+1)^2 - D(i-1+1)^2);
    d = s0 * sj;
    if (d <= 0)
        Cn(i+1) = 1;
    else
        Cn(i+1) = C(i+1) / sqrt (d);
    end
end

%----------
function EHS = PQ_FindPeak (c2, N)
% Search for a peak after a valley
```

```
cprev = c2(0+1);
cmax = 0;
for (n = 1:N-1)
    if (c2(n+1) > cprev)    % Rising from a valley
        if (c2(n+1) > cmax)
            cmax = c2(n+1);
        end
    end
end
EHS = cmax;
%-------------------------------------------------------------------------
% End of File
%-------------------------------------------------------------------------
```

PQloud.m

```
function Ntot = PQloud (Ehs, Ver, Mod)
% Calculate the loudness
% P. Kabal, McGill University Copyright (C) 2003

% P. Kabal $Revision: 1.1 $  $Date: 2003/12/07 13:35:09 $

e = 0.23;

persistent Nc s Et Ets Version Model

if (~strcmp (Ver, Version) | ~strcmp (Mod, Model))
    Version = Ver;
    Model = Mod;
    if (strcmp (Model, 'FFT'))
        [Nc, fc] = PQCB (Version);
        c = 1.07664;
    else % 'Filter_bank'
        [Nc, fc] = PQFB;
        c = 1.26539;
    end
    E0 = 1e4;
    Et = PQ_enThresh (fc);
    s = PQ_exIndex (fc);
    for (m = 0:Nc-1)
        Ets(m+1) = c * (Et(m+1) / (s(m+1) * E0))^e;
    end
end

sN = 0;
for (m = 0:Nc-1)
    Nm = Ets(m+1) * ((1 - s(m+1) + s(m+1) * Ehs(m+1) / Et(m+1))^e - 1);
    sN = sN + max(Nm, 0);
end
Ntot = (24 / Nc) * sN;

%====================
function s = PQ_exIndex (f)
% Excitation index

N = length (f);
for (m = 0:N-1)
    sdB = -2 - 2.05 * atan(f(m+1) / 4000) - 0.75 * atan((f(m+1) / 1600)^2);
    s(m+1) = 10^(sdB / 10);
end

%--------------------
function Et = PQ_enThresh (f)
% Excitation threshold

N = length (f);
for (m = 0:N-1)
    EtdB = 3.64 * (f(m+1) / 1000)^(-0.8);
    Et(m+1) = 10^(EtdB / 10);
end
```

```
%------------------------------------------------------------------------
% End of File
%------------------------------------------------------------------------
```

PQIntNoise.m

```
function EIN = PQIntNoise (f)
% Generate the internal noise energy vector
% P. Kabal, McGill University, Copyright (C) 2003
%
% P. Kabal $Revision: 1.1 $  $Date: 2003/12/07 13:34:10 $

N = length (f);
for (m = 0:N-1)
    INdB = 1.456 * (f(m+1) / 1000)^(-0.8);
    EIN(m+1) = 10^(INdB / 10);
end
%------------------------------------------------------------------------
% End of File
%------------------------------------------------------------------------
```

PQinitFMem.m

```
function Fmem = PQinitFMem (Nc, PCinit)
% Initialize the filter memories
% P. Kabal, McGill University, Copyright (C) 2003

% P. Kabal $Revision: 1.1 $  $Date: 2003/12/07 13:34:10 $

Fmem.TDS.Ef(1:2,1:Nc) = 0;
Fmem.Adap.P(1:2,1:Nc) = 0;
Fmem.Adap.Rn(1:Nc) = 0;
Fmem.Adap.Rd(1:Nc) = 0;
Fmem.Adap.PC(1:2,1:Nc) = PCinit;
Fmem.Env.Ese(1:2,1:Nc) = 0;
Fmem.Env.DE(1:2,1:Nc) = 0;
Fmem.Env.Eavg(1:2,1:Nc) = 0;
%------------------------------------------------------------------------
% End of File
%------------------------------------------------------------------------
```

PQHannWin.m

```
function hw = PQHannWin (NF)
% Hann window
% P. Kabal, McGill University, Copyright (C) 2003

% P. Kabal $Revision: 1.1 $  $Date: 2003/12/07 13:34:10 $

hw = zeros (1, NF);

for (n = 0:NF-1)
    hw(n+1) = 0.5 * (1 - cos(2 * pi * n / (NF-1)));
end
%------------------------------------------------------------------------
% End of File
%------------------------------------------------------------------------
```

Program for time alignment of audio sequences

audio_align.m

```
function s = audio_align(fin1, fin2,N)
%  AUDIO_ALIGN time aligns two audio files
%
%   AUDIO_ALIGN(FILENAME1, FILENAME2, N) takes in two WAV files and tries
%   to precisely time align them.  N is the size of the correlation used
%   and is optional.  This function generates two output files sco.wav and
%   scc.wav which are the time aligned files corresponding to FILENAME1 and
%   FILENAME2 respectively
%
%   Author: Rahul Vanam

if (nargin == 2)
    N = 30000;
end

[x, Fs] = wavread(fin1);
y = wavread(fin2);

xt = x(30001:N+30000);
yt = y(30001:N+30000);

C = xcorr(xt,yt);

[z,dex]=max(C');

% s corresponds to shift of 2nd argument relative to first
s = N - dex;

%  Shift 2nd sequence to properly align

if (s < 0)
   z = zeros(-s,1);
   scomp = x(-s+1:max(size(x)));
   min_data_len = min(length(scomp),length(y));
   sc2 = y(1:min_data_len);
   scomp = scomp(1:min_data_len);
else
   sc2 = y(s+1:max(size(y)));
   scomp = x(1:length(sc2));
end

wavwrite(scomp,Fs,16,'sco.wav'); % File #1
wavwrite(sc2,Fs,16,'scc.wav');   % File #2
%--------------------------------------------------------------------------
% End of File
%--------------------------------------------------------------------------
```

Program for computing truncation threshold

trunc_thresh.m

```
function T_final = trunc_thresh(ori,cmp)
% TRUNC_THRESH that computes the truncation threshold of a pair of audio
%  sequences
%
%   T = TRUNC_THRESH(ORI,CMP) takes in the strings for the original and
%   compressed/decompressed files.  Searches over quantization threshold
%   levels to find best match 'T'.
%
%   Author: Rahul Vanam

[A_in, Fs] = wavread(ori);
[B_in, Fs] = wavread(cmp);
num_chan = min(size(A_in));
T_final = 0;
```

```
for i = 1:num_chan
    A = A_in(:,num_chan);
    B = B_in(:,num_chan);
        size_B = size(B);
        size_A = size(A);
        if (size_A(1) > size_B(1))
         A = A(1:size_B(1),1:size_B(2));
        else
         B = B(1:size_A(1),1:size_A(2));
        end

        [b,f,t] = specgram(A,1024,Fs,1024,512);
        tfo = abs(b);
        clear b
        tfc = abs(specgram(B,1024,Fs,1024,512));
        if (mean(mean(tfc.^2)) > mean(mean(tfo.^2)))
         T = 0;
         continue;
        end
        [M,N] = size(tfc);

        ee = mean(mean(tfo.^2));
        step = 1;
        dir = 1;
        T = 10;
        tff = tfo .* (tfo>T);
        e = mean(mean(tff.^2));
        ce = mean(mean(tfc.^2));
        toll = 0.001;
        cnt = 0;
        k=0;
        while ((abs(e-ce)>toll)&(cnt < 10)&(T > 0))
         k=k+1;
         oe = abs(e-ce);
         T = T + dir*step;
         tff = tfo .* (tfo>T);
         e = mean(mean(tff.^2));
         if (abs(e-ce) > oe)
             cnt = cnt+1;
             dir = -dir;
             step = step/2;
         end
        end
    T_final = T_final + T;
end
T_final = T_final/num_chan;
%-------------------------------------------------------------------------
% End of File
%-------------------------------------------------------------------------
```

Program that generates results given in Section 5.5

ee_vs_adv.m

```
function ee_vs_adv
% EE_VS_ADV plots for performance of PEAQ advanced version and EEA
%
%   This function plots performance of Energy equalization vs. Advanced
%   version for both 16 kb/s and 32 kb/s audio data.
%   Figs 5.2 and 5.3 in my MS thesis are generated using this program.
%
%   Author: Rahul Vanam

NUM_AUDIO_SEQ = 33; % Number of audio sequences
NUM_DIFF = 20; % Number of comparisons

%-----------------------------------------------------------
% ODG values of Advanced version only
%-----------------------------------------------------------
```

```
itu_odg = [ -3.611806;-3.611806;-3.6092;-3.603015;-3.35903;-3.611806;
            -3.611806;-3.610111;-3.599571;-3.146489;-3.611748;-3.611805;
            -3.604118;-3.582936;-2.418593;-3.61174;-3.611806;-3.493951;
            -3.266166;-3.454095;-3.611806;-3.611806;-3.611766;-3.611603;
            -3.611805;-3.302208;-3.362171;-3.297297;-3.611381;-3.611792;
            -3.439588;-3.239787;-3.141195;];

% Obtain the plots for Advanced version only
itu_odg = itu_odg';
count = 1;
itu(1) = itu_odg(count+1)-itu_odg(count+2);
itu(2) = itu_odg(count+1)-itu_odg(count);
itu(3) = itu_odg(count+3)-itu_odg(count+4);
count = count+5;
itu(4) = itu_odg(count+1)-itu_odg(count+2);
itu(5) = itu_odg(count+1)-itu_odg(count);
itu(6) = itu_odg(count+3)-itu_odg(count+4);
count = count+5;
itu(7) = itu_odg(count+1)-itu_odg(count+2);
itu(8) = itu_odg(count+1)-itu_odg(count);
itu(9) = itu_odg(count+3)-itu_odg(count+4);
count = count+5;
itu(10) = itu_odg(count+1)-itu_odg(count+2);
itu(11) = itu_odg(count+1)-itu_odg(count);
itu(12) = itu_odg(count+3)-itu_odg(count+4);
count = count+5;
itu(13) = itu_odg(count+1)-itu_odg(count+2);
itu(14) = itu_odg(count+1)-itu_odg(count);
count = count+3;
itu(15) = itu_odg(count+1)-itu_odg(count+2);
itu(16) = itu_odg(count+1)-itu_odg(count);
itu(17) = itu_odg(count+3)-itu_odg(count+4);
count = count+5;
itu(18) = itu_odg(count+1)-itu_odg(count+2);
itu(19) = itu_odg(count+1)-itu_odg(count);
itu(20) = itu_odg(count+4)-itu_odg(count+3);

itu = itu';
mx = max(itu);
mn = min(itu);
r = mx - mn;

itu = 3*(itu - mn)/r;

 v = [10.453125 18.96875 5.375 3.84375 2.3125 16.125 ...
      22.75 6.296875 8.992188 2.328125 2.625 4.625 1.0625...
      0 0 6.0625 10.65625 2.25 2.625 0 11.648438 12.46875...
      3.820312 4.59375 10.21875 0 2.5625 0 6.5625 8.28125...
      2.09375 0 2];

 x = zeros(NUM_DIFF,1);

x(1) = v(2) - v(3);
x(2) = v(2) - v(1);
x(3) = v(5) - v(4);

x(4) = v(7) - v(8);
x(5) = v(7) - v(6);
x(6) = v(10) - v(9);

x(7) = v(12) - v(13);
x(8) = v(12) - v(11);
x(9) = v(15) - v(14);

x(10) = v(17) - v(18);
x(11) = v(17) - v(16);
x(12) = v(20) - v(19);

x(13) = v(22) - v(23);
x(14) = v(22) - v(21);
```

```
x(15) = v(25) - v(26);
x(16) = v(25) - v(24);
x(17) = v(28) - v(27);

x(18) = v(30) - v(31);
x(19) = v(30) - v(29);
x(20) = v(33) - v(32);

x = 3 * (x - min(x))/(max(x)-min(x));
% Subjective data
p = [2.69;1.26;0.084; 2.36;0.72;0.089; 2.13;1.27;0.12; 1.96;1.69;0.247;...
     2.01;0.03;2.36;2.11;0.1389; 1.94;1.24;0.156];

figure(1)
hold off
plot(x,p,'ko')

%  Plot ITU metric comparison

hold on
m = inv(x'*x)*x'*p;
plot([0,3],[0,3*m],'k');
plot(itu,p,'kx');

m_itu = inv(itu'*itu)*itu'*p;
plot([0,3],[0,3*m_itu],'k--');
hold on

disp('MSE complete ');
err_mine = mean((m*x-p).^2);
err_itu = mean((m_itu*itu-p).^2);

% Correlation coefficient for Advanced version
mean_subj = mean(p);
mean_obj = mean(itu);
ss_x = sum((itu - mean_obj).^2);
ss_y = sum((p - mean_subj).^2);
ss_x_y = sum((itu - mean_obj) .* (p - mean_subj));
corr_coeff_adv = sqrt((ss_x_y ^ 2)/(ss_x * ss_y));

% Correlation coefficient for EEA
mean_obj = mean(x);
ss_x = sum((x - mean_obj).^2);
ss_y = sum((p - mean_subj).^2);
ss_x_y = sum((x - mean_obj) .* (p - mean_subj));
corr_coeff_EEA = sqrt((ss_x_y ^ 2)/(ss_x * ss_y));


%******************************
% Plot for EAQUAL 1.3 ver
%******************************
eaqual_odg = ...
[-3.28 -2.91 -3.69 -3.53 -3.44 -3.16 -2.83 -3.64 -3.54 -3.47...
 -3.7  -3.59 -3.74 -3.58 -3.63 -3.41 -3    -3.63 -3.35 -3.61...
 -2.8  -2.79 -3.58 -3.46 -3.17 -3.47 -3.11 -3.4  -3.51 -3.22...
 -3.53 -2.93 -3.06];

eaqual_odg = eaqual_odg';
count = 1;
itu(1) = eaqual_odg(count+1)-eaqual_odg(count+2);
itu(2) = eaqual_odg(count+1)-eaqual_odg(count);
itu(3) = eaqual_odg(count+3)-eaqual_odg(count+4);
count = count+5;
itu(4) = eaqual_odg(count+1)-eaqual_odg(count+2);
itu(5) = eaqual_odg(count+1)-eaqual_odg(count);
itu(6) = eaqual_odg(count+3)-eaqual_odg(count+4);
count = count+5;
itu(7) = eaqual_odg(count+1)-eaqual_odg(count+2);
itu(8) = eaqual_odg(count+1)-eaqual_odg(count);
itu(9) = eaqual_odg(count+3)-eaqual_odg(count+4);
count = count+5;
```

```
itu(10) = eaqual_odg(count+1)-eaqual_odg(count+2);
itu(11) = eaqual_odg(count+1)-eaqual_odg(count);
itu(12) = eaqual_odg(count+3)-eaqual_odg(count+4);
count = count+5;
itu(13) = eaqual_odg(count+1)-eaqual_odg(count+2);
itu(14) = eaqual_odg(count+1)-eaqual_odg(count);
count = count+3;
itu(15) = eaqual_odg(count+1)-eaqual_odg(count+2);
itu(16) = eaqual_odg(count+1)-eaqual_odg(count);
itu(17) = eaqual_odg(count+3)-eaqual_odg(count+4);
count = count+5;
itu(18) = eaqual_odg(count+1)-eaqual_odg(count+2);
itu(19) = eaqual_odg(count+1)-eaqual_odg(count);
itu(20) = eaqual_odg(count+4)-eaqual_odg(count+3);

mx = max(itu);
mn = min(itu);
r = mx - mn;
itu = 3*(itu - mn)/r;

m_eaqual = inv(itu'*itu)*itu'*p;
plot(itu,p,'kv');
plot([0,3],[0,3*m_eaqual],'k-.');
hold
err_eaqual = mean((m_itu*itu-p).^2);

% Correlation coefficient for EAQUAL
mean_obj = mean(itu);
ss_x = sum((itu - mean_obj).^2);
ss_y = sum((p - mean_subj).^2);
ss_x_y = sum((itu - mean_obj) .* (p - mean_subj));
corr_coeff_EAQUAL = sqrt((ss_x_y ^ 2)/(ss_x * ss_y));

xlabel('Objective Measurement');
ylabel('Subjective Measurement');
legend('Energy Equalization Data Point','LS Fit: Energy Equalization',...
         'Advanced version Data Point',...
         'LS Fit: Advanced version', 'EAQUAL data point', 'LS fit: EAQUAL');
title ('Performance comparsion between PEAQ advanced version, EEA and EAQUAL');

%  Eliminate 1 element from LS design and test on that element
figure(2)
m_org = m;
for k=1:NUM_DIFF
    xx = x(1:k-1);
    xx = [xx;x(k+1:NUM_DIFF)];
    pp = p(1:k-1);
    pp = [pp;p(k+1:NUM_DIFF)];
    m = inv(xx'*xx)*xx'*pp;
    mm(k) = m;
    e(k) = (m*x(k)-p(k))^2;
    err_m(k) = abs(m - m_org);
end

bar([sqrt(e);err_m]',1.0);axis([0 21 0 1.5]);
xlabel('Holdout Case');
ylabel('Squared Error / Change in slope');
legend('Squared Error','Change in slope');

disp(' ');
disp('For EEA:');
disp(['MSE error = ' num2str(err_mine)]);
disp(['Slope = ' num2str(m)]);
disp(['Correlation coeff = ' num2str(corr_coeff_EEA)]);
disp(' ');
disp('For Advanced version:');
disp(['MSE error = ' num2str(err_itu)]);
disp(['Slope = ' num2str(m_itu)]);
disp(['Correlation coeff = ' num2str(corr_coeff_adv)]);
disp(' ');
disp('For EAQUAL:');
```

```
disp(['MSE error = ' num2str(err_eaqual)]);
disp(['Slope = ' num2str(m_eaqual)]);
disp(['Correlation coeff = ' num2str(corr_coeff_EAQUAL)]);
%-------------------------------------------------------------------------
% End of File
%-------------------------------------------------------------------------
```

Program that generates results given in Section 5.6.1

adv_ver_snn.m

```
function adv_ver_snn();
%   ADV_VER_SNN obtains the performance plots for PEAQ advanced version
%   with and without single layer neural network and EAQUAL.
%   The Figs 5.4 and 5.5 in my MS thesis are generated from this program.
%
%   Author: Rahul Vanam

NUM_AUDIO_SEQ = 33; % Number of audio sequences

% Number of Parameters
NUM_PARAMS = 5;

NUM_DIFF = 20;
% List of MOVs and threshold
% Each row contains mov(1), mov(2), mov(3), mov(4), and mov(5) for a given
% audio sequence.
mov_thresh = [...
% bene
312.201952     13.444291      -2.100854      0.640256       82.51397;
387.084584     27.888306      -1.243569      0.293467       155.634242     ;
255.798149     7.955216       -2.733709      1.54068    26.738128;
214.141566     4.750552       -3.851137      0.942133       23.5106     ;
196.094142     5.886441       -4.446344      0.452587       8.021638       ;

% excal
286.948618     14.427171      -2.052589      0.364698       109.970014     ;
331.904468     27.217696      -1.412073      0.162426       171.367953     ;
231.157788     5.950419       -3.295602      1.077169       29.849736      ;
205.179441     3.840642       -3.850038      0.676051       22.742348      ;
185.698156     3.760124       -3.963828      0.507612       5.68031 ;

% harp
325.554053     4.807679       -12.246382     0.478101       36.449511      ;
522.292935     10.064722      -11.903431     0.404348       48.975179      ;
364.188558     4.54666         -13.527951    2.597106       14.46788       ;
258.983529     2.962384       -14.387216     0.98804       11.744567       ;
299.591867     3.261983       -16.837725     0.976588       2.115965       ;

% quar
283.12814      6.461973       -3.886465      0.228834       41.335546      ;
323.270815     14.128753      -3.535392      0.273528       63.39099       ;
245.588396     4.965395       -4.144699      1.125329       11.22936       ;
208.683577     2.419586       -5.465347      0.706649       8.357933       ;
190.618924     3.557353       -3.856769      0.607329       3.310849       ;

% rjd
340.300604     26.384268      -2.133353      0.56323    183.068982 ;
336.484433     26.09954       -2.017671      0.48978    197.642083 ;
234.813329     7.645815       -3.815062      1.52951    44.894462  ;

% room
309.323581     16.166394      -0.797471      0.372193       32.338262      ;
347.655842     17.087178      -0.907721      0.392172       56.932229      ;
236.119288     4.08659        -2.482293      1.726887       9.055847       ;
231.122976     5.25112        -2.782951      0.440768       7.896606       ;
174.196812     2.718054       -3.299009      0.469364       3.023645       ;

% spo
```

```
264.390255     5.826258       -2.527669      0.343346       35.068831      ;
292.95655      11.908721      -2.350284      0.290935       45.613783      ;
237.509336     6.08298       -2.949692       1.167405       9.805064;
193.294018     3.695994       -4.298955      0.754623       4.904118       ;
181.849984     3.805695       -4.373074      0.897837       7.354752       ;];

% [Mov(1:5), diff in threshold
count = 1;
MOV_diff(1,:) = mov_thresh(count+1,:) - mov_thresh(count+2,:); % bsac vs.tvq 16
MOV_diff(2,:) = mov_thresh(count+1,:) - mov_thresh(count,:); % bsac vs.16f
MOV_diff(3,:) = mov_thresh(count+3,:) - mov_thresh(count+4,:); % bsac vs.tvq 32
count = count+5;
MOV_diff(4,:) = mov_thresh(count+1,:) - mov_thresh(count+2,:); % bsac vs.tvq 16
MOV_diff(5,:) = mov_thresh(count+1,:) - mov_thresh(count,:); % bsac vs.16f
MOV_diff(6,:) = mov_thresh(count+3,:) - mov_thresh(count+4,:); % bsac vs.tvq 32
count = count+5;
MOV_diff(7,:) = mov_thresh(count+1,:) - mov_thresh(count+2,:); % bsac vs.tvq 16
MOV_diff(8,:) = mov_thresh(count+1,:) - mov_thresh(count,:); % bsac vs.16f
MOV_diff(9,:) = mov_thresh(count+3,:) - mov_thresh(count+4,:); % bsac vs.tvq 32
count = count+5;
MOV_diff(10,:) = mov_thresh(count+1,:) - mov_thresh(count+2,:); % bsac vs.tvq 16
MOV_diff(11,:) = mov_thresh(count+1,:) - mov_thresh(count,:); % bsac vs.16f
MOV_diff(12,:) = mov_thresh(count+3,:) - mov_thresh(count+4,:); % bsac vs.tvq 32
count = count+5;
MOV_diff(13,:) = mov_thresh(count+1,:) - mov_thresh(count+2,:); % bsac vs.tvq 16
MOV_diff(14,:) = mov_thresh(count+1,:) - mov_thresh(count,:); % bsac vs.16f
count = count+3;
MOV_diff(15,:) = mov_thresh(count+1,:) - mov_thresh(count+2,:); % bsac vs.tvq 16
MOV_diff(16,:) = mov_thresh(count+1,:) - mov_thresh(count,:); % bsac vs.16f
MOV_diff(17,:) = mov_thresh(count+3,:) - mov_thresh(count+4,:); % bsac vs.tvq 32
count = count+5;
MOV_diff(18,:) = mov_thresh(count+1,:) - mov_thresh(count+2,:); % bsac vs.tvq 16
MOV_diff(19,:) = mov_thresh(count+1,:) - mov_thresh(count,:); % bsac vs.16f
MOV_diff(20,:) = mov_thresh(count+4,:) - mov_thresh(count+3,:);% bsac vs.bsaco 32

% Subjective data
p = [2.69;1.26;0.084; 2.36;0.72;0.089; 2.13;1.27;0.12; 1.96;1.69;0.247;...
     2.01;0.03;2.36;2.11;0.1389; 1.94;1.24;0.156];

weights = inv(MOV_diff'*MOV_diff)*MOV_diff'*p;

for ii = 1:NUM_AUDIO_SEQ
    ODG(ii) = mov_thresh(ii,:) * weights;
end

odg_diff = zeros(NUM_DIFF,1);
odg_diff(1) = ODG(2) - ODG(3);
odg_diff(2) = ODG(2) - ODG(1);
odg_diff(3) = ODG(5) - ODG(4);

odg_diff(4) = ODG(7) - ODG(8);
odg_diff(5) = ODG(7) - ODG(6);
odg_diff(6) = ODG(9) - ODG(10);

odg_diff(7) = ODG(12) - ODG(13);
odg_diff(8) = ODG(12) - ODG(11);
odg_diff(9) = ODG(15) - ODG(14);

odg_diff(10) = ODG(17) - ODG(18);
odg_diff(11) = ODG(17) - ODG(16);
odg_diff(12) = ODG(20) - ODG(19);

odg_diff(13) = ODG(22) - ODG(23);
odg_diff(14) = ODG(22) - ODG(21);

odg_diff(15) = ODG(25) - ODG(26);
odg_diff(16) = ODG(25) - ODG(24);
odg_diff(17) = ODG(28) - ODG(27);

odg_diff(18) = ODG(30) - ODG(31);
odg_diff(19) = ODG(30) - ODG(29);
```

```
odg_diff(20) = ODG(33) - ODG(32);

max_odg_diff = max(odg_diff);
min_odg_diff = min(odg_diff);

odg_diff = 3 * (odg_diff - min_odg_diff)/(max_odg_diff - min_odg_diff);

% compute am = p, where a = odg_diff value and p is the subjective
% difference value
m = inv(odg_diff'*odg_diff)*odg_diff'*p;

hold on
plot(odg_diff,p,'ko');

err_mine = mean((m*odg_diff-p).^2)
plot([0,3],[0,3*m],'k');
axis([0 3 0 3]);

mean_subj = mean(p);
mean_obj = mean(odg_diff);
ss_x = sum((odg_diff - mean_obj).^2);
ss_y = sum((p - mean_subj).^2);
ss_x_y = sum((odg_diff - mean_obj) .* (p - mean_subj));
corr_coeff_adv_snn = sqrt((ss_x_y ^ 2)/(ss_x * ss_y));

disp('For Advanced version with Single layer N.N:');
disp(['Slope = ' num2str(m)]);
disp(['MSE = ' num2str(err_mine)]);
disp(['Correlation coefficient = ' num2str(corr_coeff_adv_snn)]);
disp('------------------------------------------------------------');

%-----------------------------------------------------
% ODG values of Advanced version only
%-----------------------------------------------------
itu_odg = [...
-3.611806;-3.611806;-3.6092;-3.603015;-3.35903;-3.611806;-3.611806;...
-3.610111;-3.599571;-3.146489;-3.611748;-3.611805;-3.604118;-3.582936;...
-2.418593;-3.61174;-3.611806;-3.493951;-3.266166;-3.454095;-3.611806;...
-3.611806;-3.611766;-3.611603;-3.611805;-3.302208;-3.362171;-3.297297;...
-3.611381;-3.611792;-3.439588;-3.239787;-3.141195;];

% Obtain the plots for Advanced version only
itu_odg = itu_odg';
count = 1;
itu(1) = itu_odg(count+1)-itu_odg(count+2);
itu(2) = itu_odg(count+1)-itu_odg(count);
itu(3) = itu_odg(count+3)-itu_odg(count+4);
count = count+5;
itu(4) = itu_odg(count+1)-itu_odg(count+2);
itu(5) = itu_odg(count+1)-itu_odg(count);
itu(6) = itu_odg(count+3)-itu_odg(count+4);
count = count+5;
itu(7) = itu_odg(count+1)-itu_odg(count+2);
itu(8) = itu_odg(count+1)-itu_odg(count);
itu(9) = itu_odg(count+3)-itu_odg(count+4);
count = count+5;
itu(10) = itu_odg(count+1)-itu_odg(count+2);
itu(11) = itu_odg(count+1)-itu_odg(count);
itu(12) = itu_odg(count+3)-itu_odg(count+4);
count = count+5;
itu(13) = itu_odg(count+1)-itu_odg(count+2);
itu(14) = itu_odg(count+1)-itu_odg(count);
count = count+3;
itu(15) = itu_odg(count+1)-itu_odg(count+2);
itu(16) = itu_odg(count+1)-itu_odg(count);
itu(17) = itu_odg(count+3)-itu_odg(count+4);
count = count+5;
itu(18) = itu_odg(count+1)-itu_odg(count+2);
itu(19) = itu_odg(count+1)-itu_odg(count);
itu(20) = itu_odg(count+4)-itu_odg(count+3);
```

```
itu = itu';
mx = max(itu);
mn = min(itu);
r = mx - mn;

itu = 3*(itu - mn)/r;
m_itu = inv(itu'*itu)*itu'*p;
err_itu = mean((m_itu*itu-p).^2);

mean_subj = mean(p);
mean_obj = mean(itu);
ss_x = sum((itu - mean_obj).^2);
ss_y = sum((p - mean_subj).^2);
ss_x_y = sum((itu - mean_obj) .* (p - mean_subj));
corr_coeff_adv = sqrt((ss_x_y ^ 2)/(ss_x * ss_y));

disp('For Advanced version :');
disp(['Slope = ' num2str(m_itu)]);
disp(['MSE = ' num2str(err_itu)]);
disp(['Correlation coefficient = ' num2str(corr_coeff_adv)]);
disp('------------------------------------------------------------');

plot(itu,p,'kx');
plot([0,3],[0,3*m_itu],'k--');

%*****************************
% Plot for EAQUAL 1.3 ver
%*****************************
eaqual_odg = ...
[-3.28 -2.91 -3.69 -3.53 -3.44 -3.16 -2.83 -3.64 -3.54 -3.47 -3.7...
 -3.59 -3.74 -3.58 -3.63 -3.41 -3    -3.63 -3.35 -3.61 -2.8  -2.79...
 -3.58 -3.46 -3.17 -3.47 -3.11 -3.4  -3.51 -3.22 -3.53 -2.93 -3.06];

eaqual_odg = eaqual_odg';
count = 1;
itu(1) = eaqual_odg(count+1)-eaqual_odg(count+2);
itu(2) = eaqual_odg(count+1)-eaqual_odg(count);
itu(3) = eaqual_odg(count+3)-eaqual_odg(count+4);
count = count+5;
itu(4) = eaqual_odg(count+1)-eaqual_odg(count+2);
itu(5) = eaqual_odg(count+1)-eaqual_odg(count);
itu(6) = eaqual_odg(count+3)-eaqual_odg(count+4);
count = count+5;
itu(7) = eaqual_odg(count+1)-eaqual_odg(count+2);
itu(8) = eaqual_odg(count+1)-eaqual_odg(count);
itu(9) = eaqual_odg(count+3)-eaqual_odg(count+4);
count = count+5;
itu(10) = eaqual_odg(count+1)-eaqual_odg(count+2);
itu(11) = eaqual_odg(count+1)-eaqual_odg(count);
itu(12) = eaqual_odg(count+3)-eaqual_odg(count+4);
count = count+5;
itu(13) = eaqual_odg(count+1)-eaqual_odg(count+2);
itu(14) = eaqual_odg(count+1)-eaqual_odg(count);
count = count+3;
itu(15) = eaqual_odg(count+1)-eaqual_odg(count+2);
itu(16) = eaqual_odg(count+1)-eaqual_odg(count);
itu(17) = eaqual_odg(count+3)-eaqual_odg(count+4);
count = count+5;
itu(18) = eaqual_odg(count+1)-eaqual_odg(count+2);
itu(19) = eaqual_odg(count+1)-eaqual_odg(count);
itu(20) = eaqual_odg(count+4)-eaqual_odg(count+3);

mx = max(itu);
mn = min(itu);
r = mx - mn;
itu = 3*(itu - mn)/r;

m_eaqual = inv(itu'*itu)*itu'*p;
plot(itu,p,'kv');
plot([0,3],[0,3*m_eaqual],'k-.');
err_eaqual = mean((m_itu*itu-p).^2);
```

```
% Correlation coefficient for EAQUAL
mean_obj = mean(itu);
ss_x = sum((itu - mean_obj).^2);
ss_y = sum((p - mean_subj).^2);
ss_x_y = sum((itu - mean_obj) .* (p - mean_subj));
corr_coeff_EAQUAL = sqrt((ss_x_y ^ 2)/(ss_x * ss_y));

xlabel('Objective Measurement');
ylabel('Subjective Measurement');
legend('Data point for Modified Advanced ver.',...
    'LS Fit: Modified Advanced ver.',...
    'Advanced version data point','LS Fit:Advanced version',...
    'EAQUAL data point', 'LS Fit: EAQUAL');

disp('For EAQUAL:');
disp(['Slope = ' num2str(m_eaqual)]);
disp(['MSE error = ' num2str(err_eaqual)]);
disp(['Correlation coeff = ' num2str(corr_coeff_EAQUAL)]);
disp('------------------------------------------------------------');

%  Eliminate 1 element from LS design and test on that element
m_org = m;
for k=1:NUM_DIFF
     xx = odg_diff(1:k-1);
     xx = [xx;odg_diff(k+1:NUM_DIFF)];
     pp = p(1:k-1);
     pp = [pp;p(k+1:NUM_DIFF)];
     m = inv(xx'*xx)*xx'*pp;
     mm(k) = m;
     obj_diff = m*odg_diff(k)';
     e(k) = (obj_diff - p(k))^2;
     err_m(k) = abs(m - m_org);
 end

figure(2);bar([(e)' err_m']);
xlabel('Holdout Case');
ylabel('Squared Error / Change in slope');
legend('Squared Error', 'Change in slope');
return;
%--------------------------------------------------------------------------
% End of File
%--------------------------------------------------------------------------
```

Program that generates results given in Section 5.6.2

`adv_snn_eea.m`

```
function adv_snn_eea();
%   ADV_SNN_EEA generates plots for PEAQ advanced version with single
%   layer neural network and EEA MOV together with plots for other
%   objective metrics proposed in the thesis.
%   Figs 5.6 and 5.7 in my thesis are generated by this program.
%
%   Author: Rahul Vanam

% Number of audio sequences
NUM_AUDIO_SEQ = 33;

% Number of Parameters
NUM_PARAMS = 6;

NUM_DIFF = 20;

% List of MOVs and threshold
% Each row contains mov(1), mov(2), mov(3), mov(4), and mov(5) for a given
% audio sequence.
mov_thresh = [...
% bene
```

```
312.201952     13.444291      -2.100854      0.640256       82.51397       10.453125;
387.084584     27.888306      -1.243569      0.293467       155.634242     18.96875;
255.798149     7.955216       -2.733709      1.54068        26.738128      5.375;
214.141566     4.750552       -3.851137      0.942133       23.5106     3.84375;
196.094142     5.886441       -4.446344      0.452587       8.021638       2.3125;

% excal
286.948618     14.427171      -2.052589      0.364698       109.970014     16.125;
331.904468     27.217696      -1.412073      0.162426       171.367953     22.75;
231.157788     5.950419       -3.295602      1.077169       29.849736      6.296875;
205.179441     3.840642       -3.850038      0.676051       22.742348      8.992188;
185.698156     3.760124       -3.963828      0.507612       5.68031        2.328125;

% harp
325.554053     4.807679       -12.246382     0.478101       36.449511      2.625;
522.292935     10.064722      -11.903431     0.404348       48.975179      4.625;
364.188558     4.54666        -13.527951     2.597106       14.46788       1.0625;
258.983529     2.962384       -14.387216     0.98804        11.744567      0;
299.591867     3.261983       -16.837725     0.976588       2.115965       0;

% quar
283.12814      6.461973       -3.886465      0.228834       41.335546      6.0625;
323.270815     14.128753      -3.535392      0.273528       63.39099       10.65625;
245.588396     4.965395       -4.144699      1.125329       11.22936       2.25;
208.683577     2.419586       -5.465347      0.706649       8.357933       2.625;
190.618924     3.557353       -3.856769      0.607329       3.310849       0;

% rjd
340.300604     26.384268      -2.133353      0.56323     183.068982 11.648438;
336.484433     26.09954       -2.017671      0.48978     197.642083 12.46875;
234.813329     7.645815       -3.815062      1.52951     44.894462  3.820312;

% room
309.323581     16.166394      -0.797471      0.372193       32.338262      4.59375;
347.655842     17.087178      -0.907721      0.392172       56.932229      10.21875;
236.119288     4.08659        -2.482293      1.726887       9.055847       0;
231.122976     5.25112        -2.782951      0.440768       7.896606       2.5625;
174.196812     2.718054       -3.299009      0.469364       3.023645       0;

% spo
264.390255     5.826258       -2.527669      0.343346       35.068831      6.5625;
292.95655      11.908721      -2.350284      0.290935       45.613783      8.28125;
237.509336     6.08298        -2.949692      1.167405       9.805064       2.09375;
193.294018     3.695994       -4.298955      0.754623       4.904118       0;
181.849984     3.805695       -4.373074      0.897837       7.354752       2;];

% [Mov(1:5), diff in threshold
count = 1;
MOV_diff(1,:) = mov_thresh(count+1,:) - mov_thresh(count+2,:); % bsac vs.tvq 16
MOV_diff(2,:) = mov_thresh(count+1,:) - mov_thresh(count,:); % bsac vs.16f
MOV_diff(3,:) = mov_thresh(count+3,:) - mov_thresh(count+4,:); % bsac vs.tvq 32
count = count+5;
MOV_diff(4,:) = mov_thresh(count+1,:) - mov_thresh(count+2,:); % bsac vs.tvq 16
MOV_diff(5,:) = mov_thresh(count+1,:) - mov_thresh(count,:); % bsac vs.16f
MOV_diff(6,:) = mov_thresh(count+3,:) - mov_thresh(count+4,:); % bsac vs.tvq 32
count = count+5;
MOV_diff(7,:) = mov_thresh(count+1,:) - mov_thresh(count+2,:); % bsac vs.tvq 16
MOV_diff(8,:) = mov_thresh(count+1,:) - mov_thresh(count,:); % bsac vs.16f
MOV_diff(9,:) = mov_thresh(count+3,:) - mov_thresh(count+4,:); % bsac vs.tvq 32
count = count+5;
MOV_diff(10,:) = mov_thresh(count+1,:) - mov_thresh(count+2,:); % bsac vs.tvq 16
MOV_diff(11,:) = mov_thresh(count+1,:) - mov_thresh(count,:); % bsac vs.16f
MOV_diff(12,:) = mov_thresh(count+3,:) - mov_thresh(count+4,:); % bsac vs.tvq 32
count = count+5;
MOV_diff(13,:) = mov_thresh(count+1,:) - mov_thresh(count+2,:); % bsac vs.tvq 16
MOV_diff(14,:) = mov_thresh(count+1,:) - mov_thresh(count,:); % bsac vs.16f
count = count+3;
MOV_diff(15,:) = mov_thresh(count+1,:) - mov_thresh(count+2,:); % bsac vs.tvq 16
MOV_diff(16,:) = mov_thresh(count+1,:) - mov_thresh(count,:); % bsac vs.16f
MOV_diff(17,:) = mov_thresh(count+3,:) - mov_thresh(count+4,:); % bsac vs.tvq 32
count = count+5;
```

```
MOV_diff(18,:) = mov_thresh(count+1,:) - mov_thresh(count+2,:); % bsac vs.tvq 16
MOV_diff(19,:) = mov_thresh(count+1,:) - mov_thresh(count,:); % bsac vs.16f
MOV_diff(20,:) = mov_thresh(count+4,:) - mov_thresh(count+3,:);% bsac vs.bsaco 32

% Subjective data
p = [2.69;1.26;0.084; 2.36;0.72;0.089; 2.13;1.27;0.12; 1.96;1.69;0.247;...
 2.01;0.03;2.36;2.11;0.1389; 1.94;1.24;0.156];

weights = inv(MOV_diff'*MOV_diff)*MOV_diff'*p;

for ii = 1:NUM_AUDIO_SEQ
    ODG(ii) = mov_thresh(ii,:) * weights;
end

odg_diff = zeros(NUM_DIFF,1);
odg_diff(1) = ODG(2) - ODG(3);
odg_diff(2) = ODG(2) - ODG(1);
odg_diff(3) = ODG(5) - ODG(4);

odg_diff(4) = ODG(7) - ODG(8);
odg_diff(5) = ODG(7) - ODG(6);
odg_diff(6) = ODG(10) - ODG(9);

odg_diff(7) = ODG(12) - ODG(13);
odg_diff(8) = ODG(12) - ODG(11);
odg_diff(9) = ODG(15) - ODG(14);

odg_diff(10) = ODG(17) - ODG(18);
odg_diff(11) = ODG(17) - ODG(16);
odg_diff(12) = ODG(20) - ODG(19);

odg_diff(13) = ODG(22) - ODG(23);
odg_diff(14) = ODG(22) - ODG(21);

odg_diff(15) = ODG(25) - ODG(26);
odg_diff(16) = ODG(25) - ODG(24);
odg_diff(17) = ODG(28) - ODG(27);

odg_diff(18) = ODG(30) - ODG(31);
odg_diff(19) = ODG(30) - ODG(29);
odg_diff(20) = ODG(33) - ODG(32);

max_odg_diff = max(odg_diff);
min_odg_diff = min(odg_diff);

odg_diff = 3 * (odg_diff - min_odg_diff)/(max_odg_diff - min_odg_diff);

% compute am = p, where a = odg_diff value and p is the subjective
% difference value
m = inv(odg_diff'*odg_diff)*odg_diff'*p;

plot(odg_diff,p,'ko');
hold on
err_mine = mean((m*odg_diff-p).^2);
plot([0,3],[0,3*m],'k');
axis([0 3 0 3]);

mean_subj = mean(p);
mean_obj = mean(odg_diff);
ss_x = sum((odg_diff - mean_obj).^2);
ss_y = sum((p - mean_subj).^2);
ss_x_y = sum((odg_diff - mean_obj) .* (p - mean_subj));
corr_coeff_adv_eea = sqrt((ss_x_y ^ 2)/(ss_x * ss_y));

disp('For Advanced version with EEA and Sigle layer N.N:');
disp(['Slope = ' num2str(m)]);
disp(['MSE = ' num2str(err_mine)]);
disp(['Correlation coefficient = ' num2str(corr_coeff_adv_eea)]);
disp('---------------------------------------------------------');

%--------------------------------------------------------
```

```
% ODG values of Advanced version only
%-------------------------------------------------------
itu_odg = [...
-3.611806;-3.611806;-3.6092;-3.603015;-3.35903;-3.611806;-3.611806;
-3.610111;-3.599571;-3.146489;-3.611748;-3.611805;-3.604118;-3.582936;
-2.418593;-3.61174;-3.611806;-3.493951;-3.266166;-3.454095;-3.611806;
-3.611806;-3.611766;-3.611603;-3.611805;-3.302208;-3.362171;-3.297297;
-3.611381;-3.611792;-3.439588;-3.239787;-3.141195;];

% Obtain the plots for Advanced version only
itu_odg = itu_odg';
count = 1;
itu(1) = itu_odg(count+1)-itu_odg(count+2);
itu(2) = itu_odg(count+1)-itu_odg(count);
itu(3) = itu_odg(count+3)-itu_odg(count+4);
count = count+5;
itu(4) = itu_odg(count+1)-itu_odg(count+2);
itu(5) = itu_odg(count+1)-itu_odg(count);
itu(6) = itu_odg(count+3)-itu_odg(count+4);
count = count+5;
itu(7) = itu_odg(count+1)-itu_odg(count+2);
itu(8) = itu_odg(count+1)-itu_odg(count);
itu(9) = itu_odg(count+3)-itu_odg(count+4);
count = count+5;
itu(10) = itu_odg(count+1)-itu_odg(count+2);
itu(11) = itu_odg(count+1)-itu_odg(count);
itu(12) = itu_odg(count+3)-itu_odg(count+4);
count = count+5;
itu(13) = itu_odg(count+1)-itu_odg(count+2);
itu(14) = itu_odg(count+1)-itu_odg(count);
count = count+3;
itu(15) = itu_odg(count+1)-itu_odg(count+2);
itu(16) = itu_odg(count+1)-itu_odg(count);
itu(17) = itu_odg(count+3)-itu_odg(count+4);
count = count+5;
itu(18) = itu_odg(count+1)-itu_odg(count+2);
itu(19) = itu_odg(count+1)-itu_odg(count);
itu(20) = itu_odg(count+4)-itu_odg(count+3);

itu = itu';
mx = max(itu);
mn = min(itu);
r = mx - mn;

itu = 3*(itu - mn)/r;
m_itu = inv(itu'*itu)*itu'*p;

err_itu = mean((m_itu*itu-p).^2);

plot(itu,p,'kx');
plot([0,3],[0,3*m_itu],'k--');

mean_subj = mean(p);
mean_obj = mean(itu);
ss_x = sum((itu - mean_obj).^2);
ss_y = sum((p - mean_subj).^2);
ss_x_y = sum((itu - mean_obj) .* (p - mean_subj));
corr_coeff_adv = sqrt((ss_x_y ^ 2)/(ss_x * ss_y));

disp('For Advanced version :');
disp(['Slope = ' num2str(m_itu)]);
disp(['MSE = ' num2str(err_itu)]);
disp(['Correlation coefficient = ' num2str(corr_coeff_adv)]);
disp('------------------------------------------------------------');

%---------------------------------------
% advanced ver with single layer N.N
%---------------------------------------
mov_thresh = [...
% bene
312.201952     13.444291      -2.100854      0.640256       82.51397;
```

```
387.084584     27.888306      -1.243569      0.293467       155.634242     ;
255.798149     7.955216       -2.733709      1.54068       26.738128;
214.141566     4.750552       -3.851137      0.942133       23.5106    ;
196.094142     5.886441       -4.446344      0.452587       8.021638       ;

% excal
286.948618     14.427171      -2.052589      0.364698       109.970014     ;
331.904468     27.217696      -1.412073      0.162426       171.367953     ;
231.157788     5.950419       -3.295602      1.077169       29.849736      ;
205.179441     3.840642       -3.850038      0.676051       22.742348      ;
185.698156     3.760124       -3.963828      0.507612       5.68031 ;

% harp
325.554053     4.807679       -12.246382     0.478101       36.449511      ;
522.292935     10.064722      -11.903431     0.404348       48.975179      ;
364.188558     4.54666       -13.527951      2.597106       14.46788       ;
258.983529     2.962384       -14.387216     0.98804       11.744567       ;
299.591867     3.261983       -16.837725     0.976588       2.115965       ;

% quar
283.12814      6.461973       -3.886465      0.228834       41.335546      ;
323.270815     14.128753      -3.535392      0.273528       63.39099       ;
245.588396     4.965395       -4.144699      1.125329       11.22936       ;
208.683577     2.419586       -5.465347      0.706649       8.357933       ;
190.618924     3.557353       -3.856769      0.607329       3.310849       ;

% rjd
340.300604     26.384268      -2.133353      0.56323    183.068982 ;
336.484433     26.09954       -2.017671      0.48978    197.642083 ;
234.813329     7.645815       -3.815062      1.52951    44.894462  ;
% room
309.323581     16.166394      -0.797471      0.372193       32.338262      ;
347.655842     17.087178      -0.907721      0.392172       56.932229      ;
236.119288     4.08659       -2.482293       1.726887       9.055847       ;
231.122976     5.25112       -2.782951       0.440768       7.896606       ;
174.196812     2.718054       -3.299009      0.469364       3.023645       ;

% spo
264.390255     5.826258       -2.527669      0.343346       35.068831      ;
292.95655      11.908721      -2.350284      0.290935       45.613783      ;
237.509336     6.08298         -2.949692     1.167405       9.805064;
193.294018     3.695994       -4.298955      0.754623       4.904118       ;
181.849984     3.805695       -4.373074      0.897837       7.354752       ;];

clear MOV_diff
% [Mov(1:5), diff in threshold
count = 1;
MOV_diff(1,:) = mov_thresh(count+1,:) - mov_thresh(count+2,:); % bsac vs.tvq 16
MOV_diff(2,:) = mov_thresh(count+1,:) - mov_thresh(count,:); % bsac vs.16f
MOV_diff(3,:) = mov_thresh(count+3,:) - mov_thresh(count+4,:); % bsac vs.tvq 32
count = count+5;
MOV_diff(4,:) = mov_thresh(count+1,:) - mov_thresh(count+2,:); % bsac vs.tvq 16
MOV_diff(5,:) = mov_thresh(count+1,:) - mov_thresh(count,:); % bsac vs.16f
MOV_diff(6,:) = mov_thresh(count+3,:) - mov_thresh(count+4,:); % bsac vs.tvq 32
count = count+5;
MOV_diff(7,:) = mov_thresh(count+1,:) - mov_thresh(count+2,:); % bsac vs.tvq 16
MOV_diff(8,:) = mov_thresh(count+1,:) - mov_thresh(count,:); % bsac vs.16f
MOV_diff(9,:) = mov_thresh(count+3,:) - mov_thresh(count+4,:); % bsac vs.tvq 32
count = count+5;
MOV_diff(10,:) = mov_thresh(count+1,:) - mov_thresh(count+2,:); % bsac vs.tvq 16
MOV_diff(11,:) = mov_thresh(count+1,:) - mov_thresh(count,:); % bsac vs.16f
MOV_diff(12,:) = mov_thresh(count+3,:) - mov_thresh(count+4,:); % bsac vs.tvq 32
count = count+5;
MOV_diff(13,:) = mov_thresh(count+1,:) - mov_thresh(count+2,:); % bsac vs.tvq 16
MOV_diff(14,:) = mov_thresh(count+1,:) - mov_thresh(count,:); % bsac vs.16f
count = count+3;
MOV_diff(15,:) = mov_thresh(count+1,:) - mov_thresh(count+2,:); % bsac vs.tvq 16
MOV_diff(16,:) = mov_thresh(count+1,:) - mov_thresh(count,:); % bsac vs.16f
MOV_diff(17,:) = mov_thresh(count+3,:) - mov_thresh(count+4,:); % bsac vs.tvq 32
count = count+5;
MOV_diff(18,:) = mov_thresh(count+1,:) - mov_thresh(count+2,:); % bsac vs.tvq 16
```

```
MOV_diff(19,:) = mov_thresh(count+1,:) - mov_thresh(count,:); % bsac vs.16f
MOV_diff(20,:) = mov_thresh(count+4,:) - mov_thresh(count+3,:); % bsac vs.bsaco 32

% Subjective data
p = [2.69;1.26;0.084; 2.36;0.72;0.089; 2.13;1.27;0.12; 1.96;1.69;0.247; 2.01;...
0.03;2.36;2.11;0.1389; 1.94;1.24;0.156];

weights = inv(MOV_diff'*MOV_diff)*MOV_diff'*p;

for ii = 1:NUM_AUDIO_SEQ
    ODG(ii) = mov_thresh(ii,:) * weights;
end

odg_diff_snn = zeros(NUM_DIFF,1);
odg_diff_snn(1) = ODG(2) - ODG(3);
odg_diff_snn(2) = ODG(2) - ODG(1);
odg_diff_snn(3) = ODG(5) - ODG(4);

odg_diff_snn(4) = ODG(7) - ODG(8);
odg_diff_snn(5) = ODG(7) - ODG(6);
odg_diff_snn(6) = ODG(9) - ODG(10);

odg_diff_snn(7) = ODG(12) - ODG(13);
odg_diff_snn(8) = ODG(12) - ODG(11);
odg_diff_snn(9) = ODG(15) - ODG(14);

odg_diff_snn(10) = ODG(17) - ODG(18);
odg_diff_snn(11) = ODG(17) - ODG(16);
odg_diff_snn(12) = ODG(20) - ODG(19);

odg_diff_snn(13) = ODG(22) - ODG(23);
odg_diff_snn(14) = ODG(22) - ODG(21);

odg_diff_snn(15) = ODG(25) - ODG(26);
odg_diff_snn(16) = ODG(25) - ODG(24);
odg_diff_snn(17) = ODG(28) - ODG(27);

odg_diff_snn(18) = ODG(30) - ODG(31);
odg_diff_snn(19) = ODG(30) - ODG(29);
odg_diff_snn(20) = ODG(33) - ODG(32);

max_odg_diff_snn = max(odg_diff_snn);
min_odg_diff_snn = min(odg_diff_snn);

odg_diff_snn = 3 * (odg_diff_snn - min_odg_diff_snn)/...
(max_odg_diff_snn - min_odg_diff_snn);

% compute am = p, where a = odg_diff value and p is the subjective
% difference value
m_snn = inv(odg_diff_snn'*odg_diff_snn)*odg_diff_snn'*p;

plot(odg_diff_snn,p,'kv');
hold on

err_snn = mean((m*odg_diff_snn-p).^2);
plot([0,3],[0,3*m_snn],'k-.');
axis([0 3 0 3]);

mean_subj = mean(p);
mean_obj = mean(odg_diff_snn);
ss_x = sum((odg_diff_snn - mean_obj).^2);
ss_y = sum((p - mean_subj).^2);
ss_x_y = sum((odg_diff_snn - mean_obj) .* (p - mean_subj));
corr_coeff_adv_snn = sqrt((ss_x_y ^ 2)/(ss_x * ss_y));

disp('For Advanced version with Single layer N.N:');
disp(['Slope = ' num2str(m_snn)]);
disp(['MSE = ' num2str(err_snn)]);
disp(['Correlation coefficient = ' num2str(corr_coeff_adv_snn)]);
disp('---------------------------------------------------------');
```

```
%-----------------------------------------------------------------
xlabel('Objective Measurement');
ylabel('Subjective Measurement');
legend('Adv ver. with EEA MOV and Single Layer NN data point',...
    'LS Fit:Adv ver. with EEA MOV and Single Layer NN',...
    'Advanced ver. data point','LS Fit:Advanced version',...
    'Adv ver. with single layer NN data point', ...
    'LS Fit: Adv ver. with single layer NN');

m_orig = m;
%  Eliminate 1 element from LS design and test on that element
for k=1:NUM_DIFF
     xx = odg_diff(1:k-1);
     xx = [xx;odg_diff(k+1:NUM_DIFF)];
     pp = p(1:k-1);
     pp = [pp;p(k+1:NUM_DIFF)];
     m = inv(xx'*xx)*xx'*pp;
     mm(k) = m;
     obj_diff = m*odg_diff(k)';
     e(k) = (obj_diff - p(k))^2;
     err_m(k) = abs(m - m_orig);
 end
figure(2);bar([e;err_m]');
axis([0 21 0 1.5]);
xlabel('Holdout Case');
ylabel('Squared Error / Change in slope');
legend('Squared Error','Change in slope');
return;
%-------------------------------------------------------------------------------
% End of File
%-------------------------------------------------------------------------------
```

Program that generates results given in Section 5.6.3

adv_snn_eea_wts.m

```
function adv_snn_eea_wts();
%   ADV_SNN_EEA_WTS plots for PEAQ advanced version with EEA MOV and
%   single layer neural network with weights specific to bitrates (mid
%   and low bitrates only).  Figs 5.8 and 5.9 in my MS thesis are
%   generated by this program.
%
%   Author: Rahul Vanam

% Number of audio sequences
NUM_AUDIO_SEQ = 33;

% Number of Parameters
NUM_PARAMS = 6;

NUM_DIFF = 20;

% List of MOVs and threshold
% Each row contains mov(1), mov(2), mov(3), mov(4), and mov(5) for a given
% audio sequence.
mov_thresh = [...
% bene
312.201952     13.444291      -2.100854      0.640256       82.51397       10.453125;
387.084584     27.888306      -1.243569      0.293467       155.634242     18.96875;
255.798149     7.955216       -2.733709      1.54068        26.738128      5.375;
214.141566     4.750552       -3.851137      0.942133       23.5106    3.84375;
196.094142     5.886441       -4.446344      0.452587       8.021638       2.3125;

% excal
286.948618     14.427171      -2.052589      0.364698       109.970014     16.125;
331.904468     27.217696      -1.412073      0.162426       171.367953     22.75;
231.157788     5.950419       -3.295602      1.077169       29.849736      6.296875;
205.179441     3.840642       -3.850038      0.676051       22.742348      8.992188;
185.698156     3.760124       -3.963828      0.507612       5.68031    2.328125;
```

```
% harp
325.554053     4.807679       -12.246382     0.478101       36.449511      2.625;
522.292935     10.064722      -11.903431     0.404348       48.975179      4.625;
364.188558     4.54666       -13.527951     2.597106       14.46788       1.0625;
258.983529     2.962384       -14.387216     0.98804       11.744567      0;
299.591867     3.261983       -16.837725     0.976588       2.115965       0;

% quar
283.12814      6.461973       -3.886465      0.228834       41.335546      6.0625;
323.270815     14.128753      -3.535392      0.273528       63.39099       10.65625;
245.588396     4.965395       -4.144699      1.125329       11.22936       2.25;
208.683577     2.419586       -5.465347      0.706649       8.357933       2.625;
190.618924     3.557353       -3.856769      0.607329       3.310849       0;

% rjd
340.300604     26.384268      -2.133353      0.56323    183.068982 11.648438;
336.484433     26.09954       -2.017671      0.48978    197.642083 12.46875;
234.813329     7.645815       -3.815062      1.52951    44.894462  3.820312;

% room
309.323581     16.166394      -0.797471      0.372193       32.338262      4.59375;
347.655842     17.087178      -0.907721      0.392172       56.932229      10.21875;
236.119288     4.08659       -2.482293      1.726887       9.055847       0;
231.122976     5.25112       -2.782951      0.440768       7.896606       2.5625;
174.196812     2.718054       -3.299009      0.469364       3.023645       0;

% spo
264.390255     5.826258       -2.527669      0.343346       35.068831      6.5625;
292.95655      11.908721      -2.350284      0.290935       45.613783      8.28125;
237.509336     6.08298       -2.949692      1.167405       9.805064       2.09375;
193.294018     3.695994       -4.298955      0.754623       4.904118       0;
181.849984     3.805695       -4.373074      0.897837       7.354752       2;];

% [Mov(1:5), diff in threshold
count = 1;
MOV_diff(1,:) = mov_thresh(count+1,:) - mov_thresh(count+2,:); % bsac vs.tvq 16
MOV_diff(2,:) = mov_thresh(count+1,:) - mov_thresh(count,:); % bsac vs.16f
MOV_diff(3,:) = mov_thresh(count+3,:) - mov_thresh(count+4,:); % bsac vs.tvq 32
count = count+5;
MOV_diff(4,:) = mov_thresh(count+1,:) - mov_thresh(count+2,:); % bsac vs.tvq 16
MOV_diff(5,:) = mov_thresh(count+1,:) - mov_thresh(count,:); % bsac vs.16f
MOV_diff(6,:) = mov_thresh(count+3,:) - mov_thresh(count+4,:); % bsac vs.tvq 32
count = count+5;
MOV_diff(7,:) = mov_thresh(count+1,:) - mov_thresh(count+2,:); % bsac vs.tvq 16
MOV_diff(8,:) = mov_thresh(count+1,:) - mov_thresh(count,:); % bsac vs.16f
MOV_diff(9,:) = mov_thresh(count+3,:) - mov_thresh(count+4,:); % bsac vs.tvq 32
count = count+5;
MOV_diff(10,:) = mov_thresh(count+1,:) - mov_thresh(count+2,:); % bsac vs.tvq 16
MOV_diff(11,:) = mov_thresh(count+1,:) - mov_thresh(count,:); % bsac vs.16f
MOV_diff(12,:) = mov_thresh(count+3,:) - mov_thresh(count+4,:); % bsac vs.tvq 32
count = count+5;
MOV_diff(13,:) = mov_thresh(count+1,:) - mov_thresh(count+2,:); % bsac vs.tvq 16
MOV_diff(14,:) = mov_thresh(count+1,:) - mov_thresh(count,:); % bsac vs.16f
count = count+3;
MOV_diff(15,:) = mov_thresh(count+1,:) - mov_thresh(count+2,:); % bsac vs.tvq 16
MOV_diff(16,:) = mov_thresh(count+1,:) - mov_thresh(count,:); % bsac vs.16f
MOV_diff(17,:) = mov_thresh(count+3,:) - mov_thresh(count+4,:); % bsac vs.tvq 32
count = count+5;
MOV_diff(18,:) = mov_thresh(count+1,:) - mov_thresh(count+2,:); % bsac vs.tvq 16
MOV_diff(19,:) = mov_thresh(count+1,:) - mov_thresh(count,:); % bsac vs.16f
MOV_diff(20,:) = mov_thresh(count+4,:) - mov_thresh(count+3,:); % bsac vs.bsaco 32

% Subjective data
p = [2.69;1.26;0.084; 2.36;0.72;0.089; 2.13;1.27;0.12; 1.96;1.69;0.247; 2.01;...
0.03;2.36;2.11;0.1389; 1.94;1.24;0.156];

weights = inv(MOV_diff'*MOV_diff)*MOV_diff'*p;

for ii = 1:NUM_AUDIO_SEQ
    ODG(ii) = mov_thresh(ii,:) * weights;
end
```

```
odg_diff = zeros(NUM_DIFF,1);
odg_diff(1) = ODG(2) - ODG(3);
odg_diff(2) = ODG(2) - ODG(1);
odg_diff(3) = ODG(5) - ODG(4);

odg_diff(4) = ODG(7) - ODG(8);
odg_diff(5) = ODG(7) - ODG(6);
odg_diff(6) = ODG(10) - ODG(9);

odg_diff(7) = ODG(12) - ODG(13);
odg_diff(8) = ODG(12) - ODG(11);
odg_diff(9) = ODG(15) - ODG(14);

odg_diff(10) = ODG(17) - ODG(18);
odg_diff(11) = ODG(17) - ODG(16);
odg_diff(12) = ODG(20) - ODG(19);

odg_diff(13) = ODG(22) - ODG(23);
odg_diff(14) = ODG(22) - ODG(21);

odg_diff(15) = ODG(25) - ODG(26);
odg_diff(16) = ODG(25) - ODG(24);
odg_diff(17) = ODG(28) - ODG(27);

odg_diff(18) = ODG(30) - ODG(31);
odg_diff(19) = ODG(30) - ODG(29);
odg_diff(20) = ODG(33) - ODG(32);

max_odg_diff = max(odg_diff);
min_odg_diff = min(odg_diff);

odg_diff = 3 * (odg_diff - min_odg_diff)/(max_odg_diff - min_odg_diff);

% compute am = p, where a = odg_diff value and p is the subjective
% difference value
m = inv(odg_diff'*odg_diff)*odg_diff'*p;

disp(['predictor for modified Advanced version = ' num2str(m)]);

plot(odg_diff,p,'ko');
hold
err_mine = mean((m*odg_diff-p).^2);
plot([0,3],[0,3*m],'k');
axis([0 3 0 3]);
hold on

%--------------------------------------------------
% selecting weights based on bitrate of coded audio
%--------------------------------------------------
%--------------
% 16 kbps data
%--------------
NUM_AUDIO_SEQ_16 = 21;
NUM_DIFF_16 = 14;
mov_thresh_16 = [...
    % bene
        312.201952     13.444291      -2.100854      0.640256       82.51397
        10.453125;
        387.084584     27.888306      -1.243569      0.293467       155.634242
        18.96875;
        255.798149     7.955216       -2.733709      1.54068    26.738128  5.375;

        % excal
        286.948618     14.427171      -2.052589      0.364698       109.970014     16.125;
        331.904468     27.217696      -1.412073      0.162426       171.367953     22.75;
        231.157788     5.950419       -3.295602      1.077169       29.849736
        6.296875;

        % harp
        325.554053     4.807679       -12.246382     0.478101       36.449511      2.625;
```

```
        522.292935     10.064722      -11.903431     0.404348       48.975179      4.625;
        364.188558     4.54666         -13.527951    2.597106       14.46788       1.0625;

        % quar
        283.12814      6.461973       -3.886465      0.228834       41.335546      6.0625;
        323.270815     14.128753      -3.535392      0.273528       63.39099     10.65625;
        245.588396     4.965395       -4.144699      1.125329       11.22936       2.25;

        % rjd
        340.300604     26.384268      -2.133353      0.56323    183.068982 11.648438;
        336.484433     26.09954       -2.017671      0.48978    197.642083 12.46875;
        234.813329     7.645815       -3.815062      1.52951    44.894462  3.820312;

        % room
        309.323581     16.166394      -0.797471      0.372193       32.338262 4.59375;
        347.655842     17.087178      -0.907721      0.392172       56.932229 10.21875;
        236.119288     4.08659        -2.482293      1.726887       9.055847       0;

        % spo
        264.390255     5.826258       -2.527669      0.343346       35.068831 6.5625;
        292.95655      11.908721      -2.350284      0.290935       45.613783 8.28125;
        237.509336     6.08298        -2.949692      1.167405       9.805064  2.09375;];

% [Mov(1:5), diff in threshold
count = 1;
for i = 1:2:14
    % bsac vs.tvq 16
        MOV_diff_16(i,:) = mov_thresh_16(count+1,:) - mov_thresh_16(count+2,:);

    % bsac vs.16f
        MOV_diff_16(i+1,:) = mov_thresh_16(count+1,:) - mov_thresh_16(count,:);
    count = count+3;
end

% Subjective data
p_16 = [2.69;1.26;2.36;0.72;2.13;1.27;1.96;1.69;2.01;0.03;2.36;2.11;1.94;1.24];

weights_16 = inv(MOV_diff_16'*MOV_diff_16)*MOV_diff_16'*p_16;

for ii = 1:NUM_AUDIO_SEQ_16
    ODG_16(ii) = mov_thresh_16(ii,:) * weights_16;
end

odg_diff_16 = zeros(NUM_DIFF_16,1);
count = 1;
for i = 1:2:14
    odg_diff_16(i) = ODG_16(count+1) - ODG_16(count+2);
    odg_diff_16(i+1) = ODG_16(count+1) - ODG_16(count);
    count = count+3;
end

max_odg_diff = max(odg_diff_16);
min_odg_diff = min(odg_diff_16);

% 32 kbps data
NUM_AUDIO_SEQ_32 = 12;
NUM_DIFF_32 = 6;
mov_thresh_32 = [...
% bene
214.141566     4.750552       -3.851137      0.942133       23.5106     3.84375;
196.094142     5.886441       -4.446344      0.452587       8.021638    2.3125;

% excal
205.179441     3.840642       -3.850038      0.676051       22.742348      8.992188;
185.698156     3.760124       -3.963828      0.507612       5.68031       2.328125;

% harp
258.983529     2.962384       -14.387216     0.98804        11.744567      0;
299.591867     3.261983       -16.837725     0.976588       2.115965       0;

% quar
```

```
208.683577     2.419586       -5.465347      0.706649       8.357933       2.625;
190.618924     3.557353       -3.856769      0.607329       3.310849       0;

% room
231.122976     5.25112       -2.782951       0.440768       7.896606       2.5625;
174.196812     2.718054       -3.299009      0.469364       3.023645       0;

% spo
193.294018     3.695994       -4.298955      0.754623       4.904118       0;
181.849984     3.805695       -4.373074      0.897837       7.354752       2;];

% [Mov(1:5), diff in threshold
count = 1;
for i = 1:6
    % bsac vs.tvq 32
    MOV_diff_32(i,:) = mov_thresh_32(count,:) - mov_thresh_32(count+1,:);
    count = count+2;
end

% Subjective data
p_32 = [0.084; 0.089; 0.12; 0.247; 0.1389; 0.156];

weights_32 = inv(MOV_diff_32'*MOV_diff_32)*MOV_diff_32'*p_32;

for ii = 1:NUM_AUDIO_SEQ_32
    ODG_32(ii) = mov_thresh_32(ii,:) * weights_32;
end

odg_diff_32 = zeros(NUM_DIFF_32,1);

count = 1;
for i = 1:NUM_DIFF_32
    odg_diff_32(i) = ODG_32(count) - ODG_32(count+1);
    count = count + 2;
end

max_odg_diff = max(odg_diff_32);

p_join = [p_16' p_32']';
odg_diff = [odg_diff_16' odg_diff_32']';
odg_diff = 3 * odg_diff/max(odg_diff);
m = inv(odg_diff'*odg_diff)*odg_diff'*p_join;

mean_subj = mean(p_join);
mean_obj = mean(odg_diff);
ss_x = sum((odg_diff - mean_obj).^2);
ss_y = sum((p_join - mean_subj).^2);
ss_x_y = sum((odg_diff - mean_obj) .* (p_join - mean_subj));
corr_coeff = sqrt((ss_x_y ^ 2)/(ss_x * ss_y));

plot(odg_diff,p_join,'k*');
err_mine = mean((m*odg_diff-p_join).^2)

disp('For the Advanced version using weights based on bitrate');
disp(['predictor  = ' num2str(m)]);
disp(['Correlation coeff = ' num2str(corr_coeff)]);
disp(['MSE = ' num2str(err_mine)]);

plot([0,3],[0,3*m],'k-.');
axis([0 3 0 3]);
hold on

xlabel('Objective Measurement');
ylabel('Subjective Measurement');
legend('Data point:Advanced ver. with Energy equalization',...
    'LS Fit:Advanced ver. with Energy Equalization',...
    'Data point:Adv. ver. with bitrate based weight selection', ...
    'LS fit:Adv ver. with bitrate based weight selection');
hold off;
org_m = m;
NUM_DIFF = 20;
```

```
for k=1:20
     xx = odg_diff(1:k-1);
     xx = [xx;odg_diff(k+1:NUM_DIFF)];
     pp = p_join(1:k-1);
     pp = [pp;p_join(k+1:NUM_DIFF)];
     m = inv(xx'*xx)*xx'*pp;
     mm(k) = m;
     obj_diff = m*odg_diff(k)';
     e(k) = (obj_diff - p_join(k))^2;
     err_m(k) = abs(m - org_m);
 end
figure(2);bar([e;err_m]', 1.0);axis([0 21 0 1.5])
xlabel('Holdout Case');
ylabel('Squared Error / Change in slope');
legend('Squared Error','Change in slope');
return;
%--------------------------------------------------------------------------
% End of File
%--------------------------------------------------------------------------
```

adv_snn_eea_wts_itu.m

```
function adv_snn_eea_wts_itu();
%   ADV_SNN_EEA_WTS_ITU generates plots for PEAQ advanced version
%   with EEA MOV and single layer neural network trained for high
%   bitrates.  Figs 5.10 and 5.11 in my thesis are generated using
%   this program.
%
%   Author: Rahul Vanam

% Number of audio sequences
NUM_AUDIO_SEQ = 15;

% Number of Parameters
NUM_PARAMS = 5;

% List of MOVs and threshold
% Each row contains mov(1), mov(2), mov(3), mov(4), mov(5) and truncation
% threshold for a given audio sequence.  The audio sequences considered
% here are the conformance test audio sequences provided by the ITU.
mov_thresh = [...
99.332416      0.051394       -11.742054     0.233797       0.021695   0;
82.507233      0.441752       -16.618177     0.284909       0.08785    0.5;
144.050082     0.689523       -8.32637       0.471927       0.108362      1.5;
60.052657      0.644161       -12.14474      0.277308       0.14347       0;
125.822274     0.422028       -10.900552     0.47684        0.358602      0;
79.875626      0.317628       -16.667137     1.262408       0.066054      0.5;
158.969516     1.061878       -13.841776     1.930945       0.131035      0;
121.71475      0.268286       -15.407618     0.48256        0.065945      0.5;
43.012114      0.238412       -12.878937     0.096335       0.237441      1.4375;
37.673624      0.369128       -15.18286      0.101507       0.247168      0.875;
28.739782      0.095772       -18.131671     0.154794       0.150301      0;
80.166531      0.331611       -13.932265     1.059736       0.382365      0;
59.754871      0.101696       -14.40901      0.208584       0.11605       0;
13.668545      0.048443       -21.976061     0.175343       0.072323      0;
56.044516      0.273849       -14.895369     0.212146       0.063084      1.5;];

% Subjective data
p = [-0.336849; -0.306408; -1.156657; -0.465648; -0.833398; -0.523128;...
     -1.792714; -0.522816; -0.194927; -0.117682; -0.015752; -0.621111;
     -0.206613; 0.052329; -0.220352;];

weights = inv(mov_thresh'*mov_thresh)*mov_thresh'*p;
for ii = 1:NUM_AUDIO_SEQ
    ODG(ii) = mov_thresh(ii,:) * weights;
end

ODG = 3 * (ODG - min(ODG))/(max(ODG) - min(ODG));
p = 3 * (p - min(p))/(max(p)-min(p));
plot(ODG,p','ko');
```

```
hold on
err_mine = mean((ODG - p').^2);
m = inv(ODG*ODG')*ODG*p;
plot([0,3],[0,3*m],'k');
axis([0 3 0 3]);
hold off

mean_subj = mean(p);
mean_obj = mean(ODG);
ss_x = sum((ODG - mean_obj).^2);
ss_y = sum((p - mean_subj).^2);
ss_x_y = sum((ODG' - mean_obj) .* (p - mean_subj));
corr_coeff = sqrt((ss_x_y ^ 2)/(ss_x * ss_y));

disp('For ITU:');
disp(['Slope = ' num2str(m)]);
disp(['MSE error = ' num2str(err_mine)]);
disp(['Correlation coeff = ' num2str(corr_coeff)]);

xlabel('Objective Measurement');
ylabel('Subjective Measurement');
m_org = m;
%  Eliminate 1 element from LS design and test on that element
for k=1:NUM_AUDIO_SEQ
     xx = ODG(1:k-1);
     xx = [xx ODG(k+1:NUM_AUDIO_SEQ)];
     pp = p(1:k-1)';
     pp = [pp p(k+1:NUM_AUDIO_SEQ)'];
     m = inv(xx*xx')*xx*pp';
     mm(k) = m;
     obj_diff = m*ODG(k)';
     e(k) = (ODG(k) - p(k))^2;
     err_m(k) = abs(m_org - m);
 end
figure(2);bar([e;err_m]', 1.0);
xlabel('Holdout Case');
ylabel('Squared Error / Change in slope');
legend('Squared Error','Change in slope');
var_slope = var(mm);
disp(['Variance in the slope = ' num2str(var_slope)]);
return;
%--------------------------------------------------------------------------------
% End of File
%--------------------------------------------------------------------------------
```

REFERENCES

[1] M. Kahrs and K. Brandenburg, "Applications of Digital Signal Processing to audio and acoustics," Kluwer academic publishers, Boston, 1998.

[2] K. Brandenburg, "Evaluation of quality of audio encoding at low bit rates," *82nd AES Convention of the Audio Engineering Society*, preprint 2433, London, UK, February 1987.

[3] J. G. Beerends and J. A. Stemerdink, "A perceptual audio quality measure based on psychoacoustic sound representation," *J. Audio Eng. Soc.*, Vol. 40, pp.963-978, December 1992.

[4] B. Paillard, P. Mabilleau, S. Morissette and J. Soumagne, "PERCEVAL: Perceptual evaluation of the quality of audio signals," *J. Audio Eng. Soc.*, Vol.40, pp. 21-31, Jan./Feb 1992.

[5] C. Colomes, M. Lever, J. B. Rault, Y.F. Dehery, "A perceptual model applied to audio bit-rate reduction," *J. Audio Eng. Soc.*, Vol.43, p. 233-240.

[6] T. Sporer, "Objective audio signal evaluation – applied psychoacoustics for modeling the perceived quality of digital audio," *103rd AES-Convention*, preprint 4512. New York, USA.

[7] T. Thiede and E. Kabot, "A new perceptual quality measure for the bitrate reduced audio", *100th Convention of the Audio Engineering Society*, preprint 4280, Copenhagen, Denmark, 1996.

[8] *Method for objective measurements of perceived audio quality*, Recommendation ITU-R BS.1387-1, Geneva, Switzerland, 1998-2001.

[9] C. D. Creusere, "Quantifying perceptual distortion in scalably compressed MPEG audio," *37th Asilomar Conference on Signals, Systems and Computers*, Monterey, U.S.A, pp.265-269, November 2003.

[10] K. Kallakuri, "An improved perceptual quality metric for highly to moderately impaired audio," M.S. thesis, Electrical and Computer Engineering, New Mexico State University, Las Cruces, USA, 2004.

[11] T. Painter and A. Spanias, "Perceptual coding of digital audio," *Proceedings Of the IEEE*, Vol.88, No.4, April 2000.

[12] E. Zwicker and R. Feldtkeller, *Das Ohr als Nachrichtenempfänger*, Stuttgart, Hirzel Verlag, Germany, 1967.

[13] T. Theide et al, "PEAQ – the ITU standard for objective measurement of audio quality," *J.Audio Eng. Soc.*, Vol. 48, No. 1/2, January/February 2000.

[14] T. Theide, "Perceptual audio quality assessment using non-linear filter bank," PhD thesis, Technical University of Berlin, Berlin, 1999.

[15] J. Karjalainen, "A new auditory model for evaluation of sound quality of audio systems," *Proc. of IEEE International Conference in Audio, Speech and Signal Processing*, pp. 608-11, Tampa Bay, FL, March 1985.

[16] P. Kabal, "An examination and interpretation of ITU-R BS.1387: Perceptual Evaluation of Audio Quality," Technical Report, Electrical and Computer engineering, McGill University, Montreal, Canada, 2002.

[17] Source code for EAQUAL by A. Lerch can be downloaded from the website: http://www.mp3-tech.org/programmer/misc.html

[18] PEAQ software by P. Kabal, Source code can be downloaded from the website http://www.tsp.ece.mcgill.ca/MMSP/Documents/Software/index.html

[19] *Subjective performance assessment of telephone-band and wide-bandwidth digital codecs*, Recommendation ITU-R P.830, Geneva, Switzerland, 1996.

[20] *Methods for subjective assessment of small impairments in audio systems including multichannel sound systems*, Recommendation ITU-R BS.1116, Geneva, Switzerland, 1994.

[21] A. L. Edwards, *An introduction to linear regression and correlation*, W. H. Freeman, New York, 1984.

[22] R. Vanam and C. D. Creusere, "Evaluating low bitrate scalable audio quality using advanced version of PEAQ and Energy Equalization approach," *Proc. of IEEE International Conference in Audio, Speech and Signal Processing*, Vol.3, pp.189-192, Philadelphia, March 2005.

[23] *Method for the subjective assessment of intermediate quality level of coding systems*, Recommendation ITU-R BS.1534-1, Geneva, Switzerland, 2001-2003.