Sound Quality User-defined Cursor Reading Control -Tonality Metric

Zhong Zhang (s001071)
Merina Shrestha (s001078)

IMM, DTU
Brüel & Kjær

10/03/2003

Abstract

Sound Quality as a relatively new product parameter has become an important competition item among the car manufactures. Sound Quality metrics is the same as Sound Quality parameters, which can reflect most of the psychoacoustic properties of the human perception of sound. There are some standardized metrics, such as Stationary Loudness, Tone to Noise Ratio, Prominence ratio, but most metrics are not standardized. They all have the advantage that they conclude with a single number on the characteristic properties of the sound.

Tonality is one of the Sound Quality metrics that is not yet standardized, but it is an important parameter affecting Sound Quality because it is proportional to the human perception of tonal contents in the sound.

Terhardt's Tonality, which was proposed in 1892, is an algorithm for extraction of pitch and pitch salience from complex tonal signals. It is widely accepted that Terhardt's method is the foundation for Tonality metric definition.

Studying Terhardt's algorithm is the start point of this thesis project. A MATLAB model according to Terhardt's algorithm is made. AVC++ implementation is also made in order to make it ready for Brüel & Kjær Sound Quality Application to use. Furthermore, both MATLAB and VC++ programs have been tested, and validated for further research work.

An overall study of Stationary Loudness and Aures's Model of Tonality is also provided by our thesis. They broaden our knowledge in psychoacoustics research field.

Preface

This master thesis has been carried out under the cooperation between Informatics and Mathematical Modelling Department (IMM), Technical University of Denmark (DTU), and Brüel & Kjær Sound and Vibration A/S, from Sep 9, 2002 to Mar 10, 2003.

March 10, 2003

Zhong Zhang s001071

Merina Shrestha s001078

Acknowledgements

We would like to express our sincere thanks to our thesis supervisors, Jens Thyge Kristensen and Henrik Haslev, who are on behalf of IMM, DTU and B&K respectively, for their nice guidance and suggestions during our thesis period.

We would like to thank Svend Lysemose, Poul Ladegaard, Tommy Schack, Howard Mealor, Jacob Juhl Christensen and all colleagues in Sound Quality group in B&K for providing us such a convenient and friendly work environment.

We would like to thank Torben Poulsen, Aaron Hastings, Wolfgang Ellermeier for providing us the helpful information.

Abbreviations

ANSI	American National Standards Institute
B&K	Brüel & Kjær
CPB	Constant Percentage Bands
DFD	Data Flow Diagram
DIN	German Standard
FFT	Fast Fourier Transform
ISO	The International Organization for Standardization
SPL	Sound Pressure Level
SQ	Sound Quality

Contents

List of Figures

List of Table

Chapter 1 Introduction

This chapter introduces our thesis, and includes what it is about, how it carried out and how it is organized in the following chapters.

1.1 Problem definition

The B&K PULSE sound quality software, a program that records, analyses and edits sounds, that later on can be played back to a panel of listeners, is an application that is used by product designers to compare, evaluate and change the sound the product makes.

In addition to the standard analysis tools, it has a built-in user-defined cursor reading functionality that allows the customers to make their own analysis functions or "metrics" as they are often called. That functionality is implemented by supporting with a specific interface definition for ActiveX controls. Data can be retrieved from the SQ program, analysed in the control and the result sent back and shown as a single number in a display in SQ.

In the project carried out by Joseph Emmanuel Ammuah [1.1], a number of user – defined Cursor Controls were developed. The focus was on structuring of the code for the controls, so it would be easy for SQ customers to develop their own controls. Therefore the metrics were chosen not to be too complex.

One metric, however, is very much in demand, namely the "Tonality". Tonality, as a calculation, was proposed in 1982 by Ernst Terhardt, Gerhard Stoll and Manfred Seewan from the Technical University of Munich. The value for Tonality is proportional to the human perception of tonal contents in the sound. Knowledge about the properties of the human auditory system is built into the method making it very complex. But also, the method is based on the capabilities of the analysis equipment available at that time. Therefore, the whole calculation procedure should be revised, so present technology is utilised.

Based on the original article and the present tools and techniques a User – defined cursor reading control for the Tonality metric must be implemented and documented. Visual C++ is the preferred development environment.

The project requires a basic knowledge of Sound Quality. The tools for the project are,

- MATLAB
- Visual C++
- ATL
- Object modelling tool or data flow design tools

1.2 Our Difficulties and Strategies

This is our master thesis project, and both of us are master students in computer system engineering. Through our course projects at DTU, we learned some

knowledge in both software engineering and system design, and also got some practical experience with programming and group work. But when we came to this project, we realized that our knowledge we learned about software in DTU was only a basic requirement for us to be able to fulfil a project like this in a real company. In addition, a deep understanding of software development in the real world and some basic knowledge used in Sound Quality field is needed in order to fulfil the task. We must say that we do not really feel that we have enough basic knowledge in the acoustics field before we start this project. So in the beginning of this project we have put lots of considerations on how to step-wise gain relevant knowledge, how to make plans in each stage and how to work together efficiently.

The different aspects or difficulties, which we need to work on in order to fulfil the task through this project period, were highlighted as follows:

- Frequency Analysis: We have the fundamental knowledge of advanced mathematics, but we have never touched the field of frequency analysis before we start this project. So we decided that we definitely need some time to reach an understanding of frequency analysis and more relevant knowledge in the field of signal analysis and processing.

- Virtual Pitch theory: It is the very basic knowledge in order to understand the 2 articles which we got as an inspiration for this project which relates very much with virtual pitch theory in the psychoacoustics field. So we thought we definitely needed some time to read and understand the virtual pitch theory from the psychoacoustics point of view. Some help from the experts will also be necessary and helpful for us.

- MATLAB Programming: After reading through the two original articles and relevant discussions with other developer in Sound Quality group, we were ready to design the workflow of the Tonality algorithm in order to implement it in a programming language. From Joseph's final thesis, we got some idea about how VC++ or VB used as development tools in B&K SQ Application and what is the general procedure to develop a user-defined cursor reading control for B&K SQ Application. On the other hand, we feel that comparing to the several metrics that Joseph has implemented, our Tonality metric has some specialty which is that Tonality Algorithm is not a standard, it is a very general calculation procedure for complex tones and there are lots of mathematic formulas involved. Furthermore, we want to seek the possibilities to extend this algorithm into some specific applied field such as noise control. So we need to make a model in a very flexible way in order to let it have the possibility of changing some settings or parameters in the model, which will be very much helpful for further research work. In addition, VC++ programming is also quite new for us even though we have C programming and other language programming experience. MATLAB is more function oriented programming language and is one of the best tools designed for mathematic modelling purpose, which is coincident to the original algorithm in a better way than object oriented programming languages. So we finally decide to make a MATLAB model instead of going to VC++ programming directly. One can understand that making Tonality MATLAB model is an in-between procedure between understanding the algorithm and the VC++ programming for Tonality implementation.

- VC++ development tools: We do not often use VC++ in our study period in DTU, so we need to be familiar with the development tools and the language. It is nice for us to have Joseph's thesis in hand, which provides us some practical instruction on using the ATL to make COM programming. ATL makes the COM programming simpler and easier, and programmers need not bother too much about the concepts behind COM, which is probably why Joseph did not mention much about COM in his report. On the other hand, we found that Joseph did not really present much detail about how COM Components being used in developing new user-defined cursor reading controls. Therefore, we decided to acquaint ourselves with some major concepts inside COM before starting ATL COM programming and later reach a better understand on the approach that B&K SQ Application used in general user-defined cursor reading control development. So in this report we will not repeat the same content that Joseph has presented in his report, but will mention some COM concepts, which are closely related to our project. We think they are very helpful to understand the technology behind VC++ COM programming.

- The revision of the method: Ideally, it will be very nice that we could present a new way or revision of the Tonality method. But when we really want deep into this subject, we found out that the current situation of Tonality is quite complicated. Firstly, it is not yet standardised, and the original algorithm is actually generally related with complex tones, but not for specific applied field. That means if we need to use it in noise control fields for example, or some other aspects, we need to consider more about the specific signal characteristics, which might influence the revision of the method in different directions. Secondly, different Acoustic companies have different implementations in their SQ Application, which might be specific to their customer requirement and their previous research work, which postpone the standardising work. Thirdly, we only have 6 months to work on this project, and knowledge in several fields is needed for us to understand. For a new metric definition in Sound Quality at least some subjective evaluations need to be carried out. This is both very much time-consuming and expensive. On the other hand, lacking of psychoacoustics knowledge prevent us to be able to present a method to extract a single number for Tonality metric. So we decided to do some research work on the relevant field in order to reach a better understanding of psychoacoustics.

- Why study Loudness and Aures's model of Tonality: We must say that Tonality algorithm is very complicated because it involves a lot of psychoacoustics knowledge. After we got the MATLAB and VC++ code done, we have gain the basic understand of the algorithm, which is more from a programmer point of view but less from psychoacoustics point of view. At this stage, it is very difficult for us to suggest a way to get a single number for Tonality metric based on spectral pitch pattern and virtual pitch pattern. While Aures's model of tonality simplifies the tonality procedure and introduces Loudness into Tonality. So we decide to study Loudness in order to get some fresh idea and solid understand of some psychoacoustics concepts.

The strategies that we applied in the whole project period in order to complete this project in an efficient way, are as follows.

- The whole project period is divided into several stages. We made a detailed plan in the beginning of each stage and wrote relevant progress documents in the end of each stage in order to summarize the work done and also keep a good track for the final thesis content in the last stage.
- We tried to balance the practical and theoretical work to make sure that the whole project has considerations from both sides.
- Using all the available possible resources from the company because it is a project integrating software development technology and acoustic knowledge in many aspects, many problems can be solved in a nice and easy way if you are able to exchange your ideas with people working in different field.
- Understanding the specific knowledge to the extent that we need for using it will have higher priority than to deeply understand it, which sometimes is very time-consuming and you easily get stuck.
- Co-operation and individual work can be easily and efficiently carried out if both of the project partners reach the same understanding of the whole project in a correct way, therefore, sharing knowledge is the right and wise thing to do for both of us in order to complete this project perfect.

1.3 Working Process

Our work has progressed as illustrated in the figure below when read in a bottom-up manner. Firstly, we build our knowledge from the basic concepts of sound, frequency analysis and psychoacoustics, which are related in a way that lower level is necessary

to be understood before you come to the higher level. Secondly, we investigate the Tonality method and make a MATLAB model afterwards. Thirdly, we have two paths: one is to be familiar with the SQ Application and the development tools, implement Tonality metric in VC++ and testing afterwards, the other is to study Loudness, Aures's Model of Tonality, and seek a way of improving the Tonality model. Finally, we write our thesis based on the work we have done in this project.

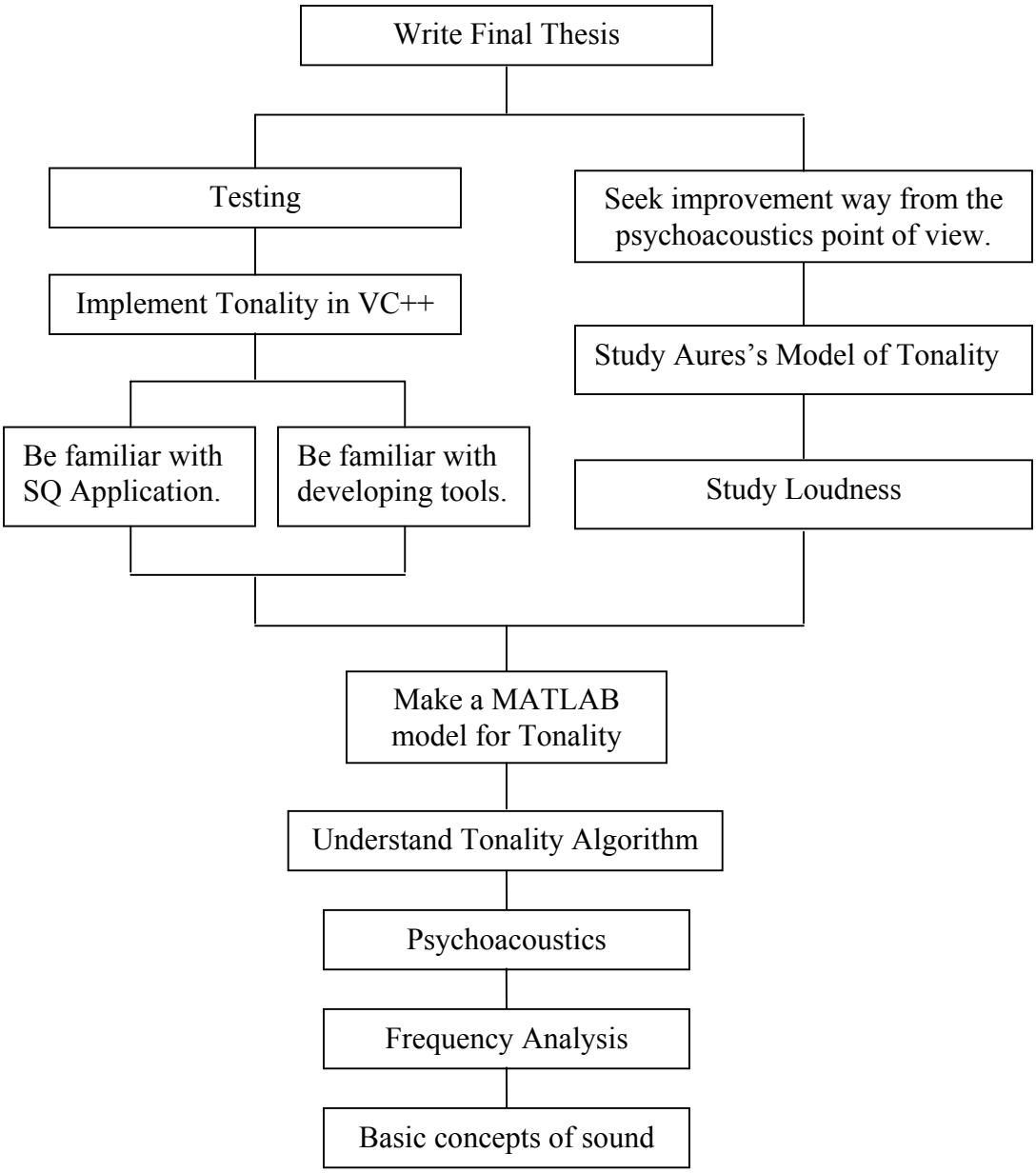

Figure 1.1 General structure of this project

1.4 Thesis Organization

The remainder of this thesis is organized as follows:

Chapter 2 provides the specific background knowledge about Sound Quality, Psychoacoustics and Sound Quality metrics, which are necessary to know in order to understand this thesis.

Chapter 3 reviews the original Tonality Algorithm presented in 1982 by Terhardt, E., Stoll,G., and Seewann, M.

Chapter 4 describes how we built Tonality Model in MATLAB according to Terhardt's method.

Chapter 5 describes the technical and practical information in the progress of the implementation of Tonality metric for B&K SQ Application in VC++.

Chapter 6 shows how we test our MATLAB and VC++ programs.

Chapter 7 includes some theoretic study in Stationary Loudness and Aures's Model of Tonality.

Chapter 8 summarizes our work and indicates future implementation paths.

Additionally, source code, test data and relevant materials of this project are available in Appendix.

Chapter 2 Specific Background Knowledge for this Thesis

According to the specific area of this project, we would like to present the background knowledge before we go into our thesis work in detail. This chapter covers the basic

concepts of Sound Quality and psychoacoustics, the metrics involved in Sound Quality field, the feature of Tonality metric, and some basic concepts. Most of the content in this chapter is taken from the relevant B&K Sound Quality Lecture Notes and it can be studied as an infrastructure of the whole thesis.

2.1 Sound Quality

2.1.1 What is Sound Quality?

The sound of a product is now a product parameter that needs the same attention as its physical design, horsepower, color, weight, price etc. It all started in the automotive industry more than 10 years ago and it is still in that area that the most advanced Sound Quality developments take place. One reason is the very heavy competition between car manufacturers and the fact that most cars are of high quality and have the same performance in relation to what they are supposed to do. Sound Quality as a relatively new product parameter has then become an important item to compete on. If your car has better Sound Quality than the competition you are closer to win the sale. In recent years the focus on Sound Quality has spread to almost all other industries producing products that makes noise. The household appliance industries are good examples. Sound Quality as a product parameter is most developed in USA, Europe and Japan. In other countries is expected to grow rapidly as more and more products become not sellable unless their Sound Quality parameters has been attended to. As product sound is directly communicating with the users senses – the ears – the knowledge of how we perceive sound has got increased focus. This discipline is called Psychoacoustics and is important in the education of design and development engineers.

2.1.2 Why improve the Quality of Sound?

The noise from a product is a part of the communication between the product and its user. Therefore it has to be changed into sound that is pleasing to the user and give him all the information of the function and life time of the product he needs - no more no less. The pleasing aspect of product sound is perceived subjectively and depends on the individual user. This situation leaves the designer of the product in a very challenging position. He has to optimise the product sound to the target customer group that has a taste that is not uniform and will change over time as fashion. Pleasing sounds get worn and need replacement by new exciting sounds. He has to skip old design tools and learn new ones. For example the widely used A-weighting is fine for noise but useless for sound. Three different vacuum cleaners may have the same A-weighted noise level but can have very different sounds. No noise may be a target for a noise control, but in relation to Sound Quality no sound is unacceptable. Products should always signal proper operation to the user as well as a warning signal when the electric drill is overloaded or the car is hitting rough road surface an the driver should reduce speed.

2.1.3 Work with Sound Quality

Working with Sound Quality is an iterative process. Often you start with prototypes of a product, which has to be optimised in Sound Quality. You make recordings - preferably using a Head and Torso - of the sound from your prototypes and you may also have competitor products included in the test. Then you get the first evaluation from a listening test with a jury representing the final users of the product. If your prototype wins the listening test and is perfect you have finished the job.

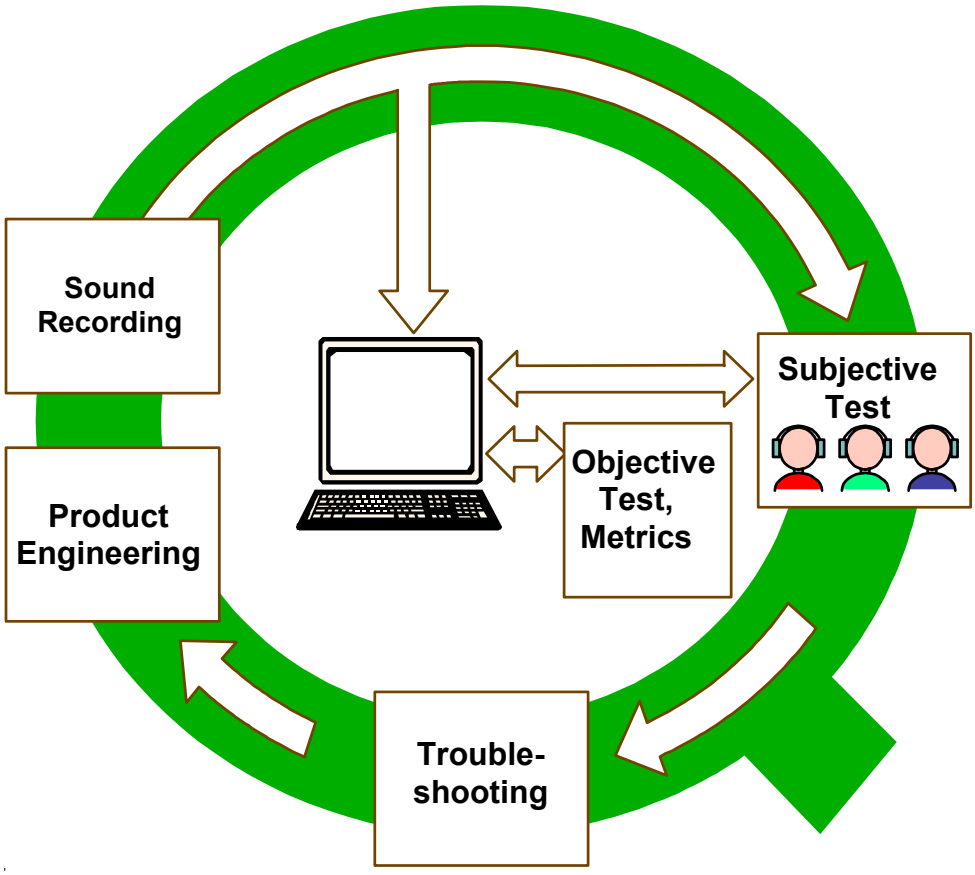

Figure 2.1 an iterative process showing the general steps of Sound Quality Analysis

If your prototype fails you can direct the sounds to the Sound Quality program for detailed analysis. In that you may find some spectral components, which you expect responsible for the poor sound. With the edit function in the program you can then simulate a removal of the unwanted components. If a new listening test approve the modification the next step is to do some trouble shooting to identify where the unwanted components come from. Then some product engineering is needed to modify the prototype. Then a new sound recording and listening test is needed.

If your prototype still fails you must go to the analysis again and try other edits to modify the sound. In order to qualify your progress a number of objective tests - Metrics - are available. They give a single number to characterise specific properties of the sound for example how rough the sound is. If you know that e.g. an increase in the value corresponds to improved Sound Quality you can use this metric to optimise the simulations of product changes and save time consuming listening tests.

2.1.4 Optimisation of Sound Quality Analysis

In optimising product Sound Quality you must never forget that the human being is the final judge on how well you succeed. That is, it has to pass the subjective listening tests flawlessly. Listening tests are both time consuming, costly and off line in relation to the Sound Quality editing and simulation process. Therefore the objective metrics based on psychoacoustics research are very attractive as a complement to the subjective tests. They are cheap, fast and on line, but only a real substitute to listening tests if they can give matching results.

The big challenge is to design a set of metrics e.g. as a combination of several metrics with individual weighting, that as a single number can give precise and reliable correlation to the subjective tests and preferences.

Most manufacturers working with Sound Quality deal seriously with this problem. They regard the results as company secrets, so they seldom publish their findings. On the other hand the results are often so product specific that they hardly are of any direct use for other manufacturers.

It is believed that a metric never will be good enough to replace the subjective test completely - they will remain complementary partners.

Another use of metrics is: As sub suppliers now also have to meet Sound Quality requirements, they are obliged to perform QC (Quality Control) on their products. Naturally, they can't rely on subjective tests for that purpose, but have to develop a good correlating metric.

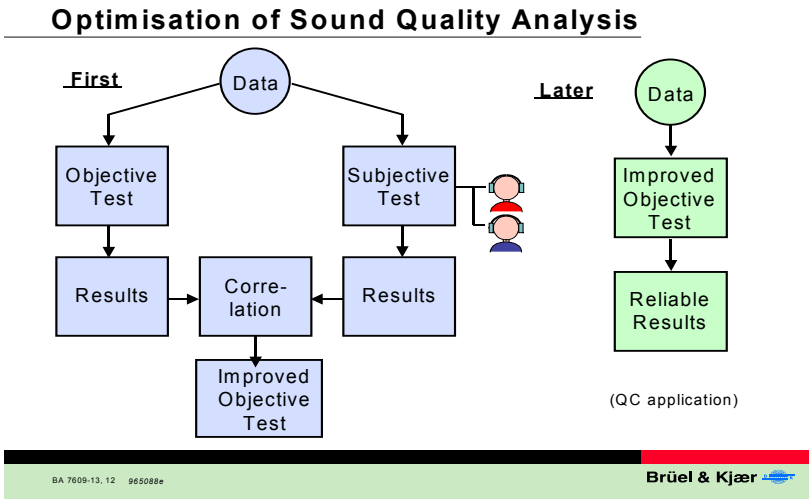

Figure 2.2 Optimisation of Sound Quality Analysis

2.2 Psychoacoustics

The ability of our hearing system to receive information is determined not only by the qualitative relation between sound and impression, but also by the quantitative relation between acoustical stimuli and hearing sensations. With the advent of new digital audio techniques, the science of the hearing system as a receiver of acoustical

information, i.e. the science of psychoacoustics, has gained additional importance. In the years from 1952 to 1967, the research group on hearing phenomena at the Institute of Telecommunications in Stuttgart made important contributions to the quantitative correlation of acoustical stimuli and hearing sensations, i.e. to psychoacoustics. Since 1967, research groups at the Institute of Electroacoustics in Munich have continued to make progress in this field. The correlation between acoustical stimuli and hearing sensations is investigated both by acquiring sets of experimental data and by models which simulate the measured facts in an understandable way.

2.2.1 Stimuli and Sensations

The most important physical magnitude for psychoacoustics is the time function of sound pressure. The stimulus can be described by physical means in terms of sound pressure level, frequency, duration and so on. The physical magnitudes mentioned are correlated with the psychophysical magnitudes loudness, pitch, and subjective duration, which are called hearing sensations. However, it should be mentioned that the pitch of a pure tone depends not only on its frequency, but also to some extent on its level. Nonetheless, the main correlation of the hearing sensation pitch is the stimulus quantity frequency. Physical stimuli only lead to hearing sensations if their physical magnitudes lie within the range relevant for the hearing organ. For example, frequencies below 20Hz and above about 20kHz do not lead to a hearing sensation whatever their stimulus magnitude. Just as we can describe a stimulus by separate physical characteristics, we can also consider several hearing sensations separately. For instance, we can state, “the tone with the higher pitch was louder than the tone with the lower pitch”. This means that we can attend separately to the hearing sensation “loudness” on one hand and “pitch” on the other. A major goal of psychoacoustics is to arrive at sensation magnitudes analogous to stimulus magnitudes. For example, we can state that a 1-kHz tone with 20mPa sound pressure produces a loudness of 4sone in terms of hearing sensation. The unit “sone” is used for the hearing sensation loudness in just the same way as the unit “Pa” is used for the sound pressure. It is most important not to mix up stimulus magnitudes such as “Pa” or “dB” and sensation magnitudes such as “sone”. [2.1]

2.2.2 Hearing Area

The hearing area is a plane in which audible sounds can be displayed. In its normal form, the hearing area is plotted with frequency on a logarithmic scale as the abscissa, and sound pressure level in dB on a linear scale as the ordinate. This means that two logarithmic scales are used because the level is related to the logarithm of sound pressure. The critical-band rate may also be used as the abscissa. This scale is more equivalent to features of our hearing system than frequency.

The usual display of the human hearing area is shown in Figure 2.3. On the right, the ordinate scales are sound intensity in Watt per square meter (W/m^2) and sound pressure in Pascal (Pa). Sound pressure level is given for a free-field condition relative to $2\times10^{-5}\,Pa$. Sound intensity level is plotted relative to $10^{-12}\,W/m^2$.

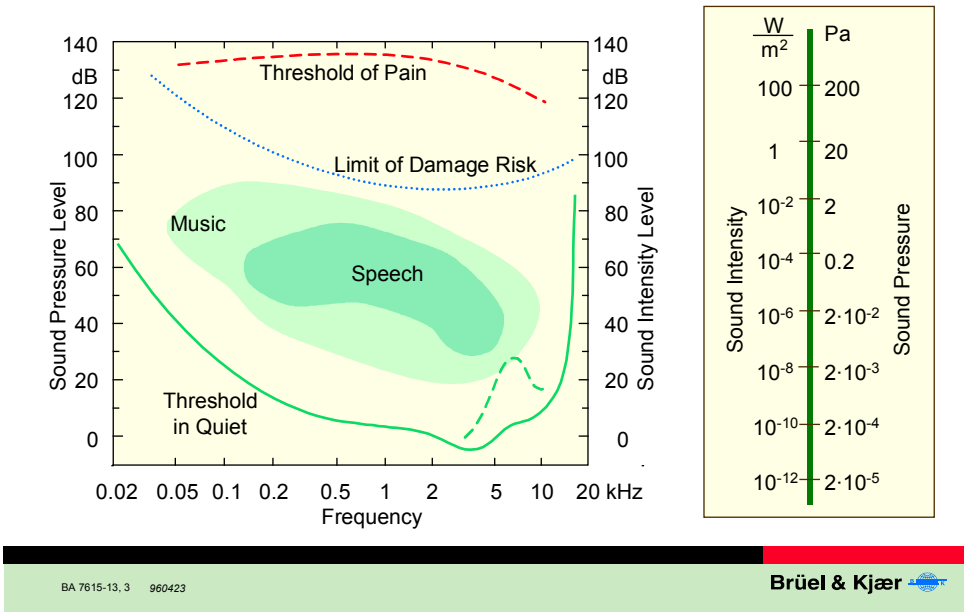

Figure 2.3 Hearing areas between threshold in quiet and threshold of pain.

This display of the auditory field illustrates the limits of the human auditory system. The solid line denotes, as a lower limit, the threshold in quiet for a pure tone to be just audible. The upper dashed line represents the threshold of pain. However if the Limit of Damage Risk is exceeded for a longer time, permanent hearing loss may occur. This could lead to an increase in the threshold of hearing as illustrated by the dashed curve in the lower right-hand corner. Normal speech and music have levels in the shaded areas, while higher levels require electronic amplification. Human hearing is extremely sensitive. An acoustic power intensity of only 1 mW per square metre may already exceed the limit of damage risk.

2.2.3 Equal Loudness Contours and A-weighting

The hearing sensation of loudness represents a dominant feature for Sound Quality evaluation. The solid curves in Figure 2.4 are called “equal loudness contours”. They demonstrate that the hearing system is most sensitive for frequencies around 4 kHz and shows reduced sensitivity at lower and higher frequencies. In particular at low frequencies the equal loudness contours are not shifted in parallel, but show a level dependence. The contours are labelled in phon. A 60phon contour represents the level in dB needed to give equal sensation of signal loudness versus frequency. At 1 kHz the level in dB and phon have the same value. Another measure of loudness is sone. It has a reference in a 1 kHz level of 40 phon or 40 dB which is equal to 1 sone. A doubling of the sone value represents a doubling of the perceived loudness of a sound. It takes an increase in level from 40 phon to 50 phon to reach 2 sone. And another increase in level from 50 phon to 60 phon will give 4 sone. In short, it is necessary to increase the loudness value by 10 phon to give the sensation of a doubling of the loudness.

The dashed curve in the graph shows the well-known A-weighting. For very low sounds there is a good agreement with the 20 phon curve. At higher levels, e.g., 80 phon - typical for everyday sounds - it underestimates the loudness of their low frequency components.

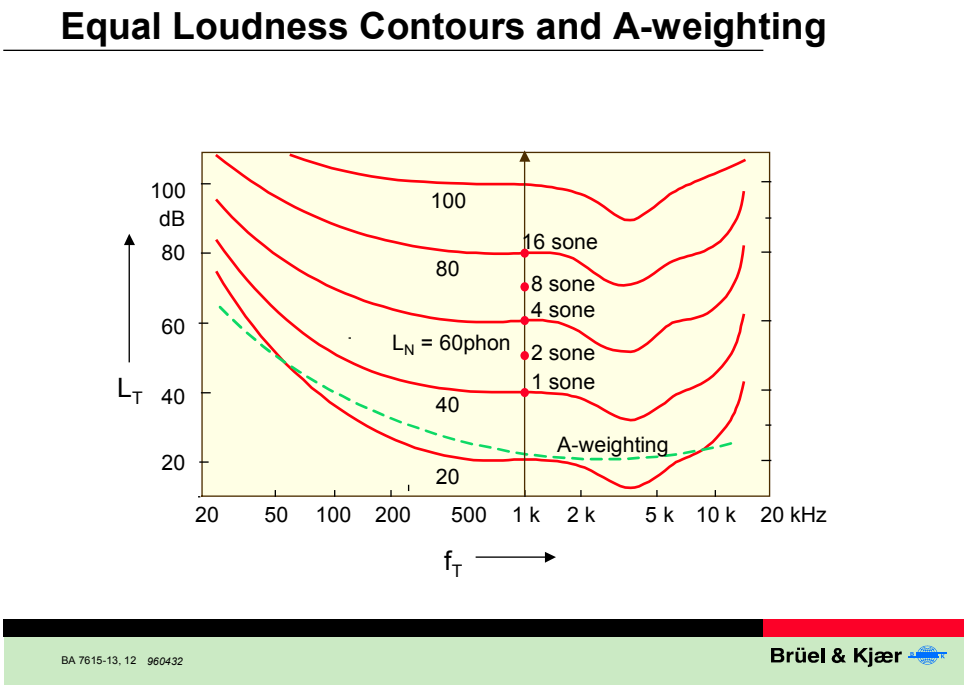

Figure 2.4 equal Loudness Contours and A-weighting

2.2.4 Masking

Masking plays a very important role in everyday life. For a conversation on the pavements of a quiet street, for example, little speech power is necessary for the speakers to understand each other. However, if a loud truck passes by, our conversation is severely disturbed: by keeping the speech power constant, our partner can no longer hear us. There are two ways of overcoming this phenomenon of masking. We can either wait until the truck passed and then continue our conversation, or we can raise our voice to produce more speech power and greater loudness. Our partner then can hear the speech sound again. Similar effects take place in most pieces of music. One instrument may be masked by another if one of them produces high levels while the other remains faint. If the loud instrument pauses, the faint one becomes audible again. These are typical examples of simultaneous masking. To measure the effect of masking quantitatively, the masked threshold is usually determined. The masked threshold is the sound pressure level of a test sound (usually a sinusoidal test tone), necessary to be just audible in the presence of a masker. Masked threshold, in all but a very few special cases, always lies above threshold in quiet; it is identical with threshold in quiet when the frequencies of the masker and the test sound are very different.

If the masker is increased steadily, there is a continuous transition between an audible (unmasked) test tone and one that is totally masked. This means that besides total masking, partial masking also occurs. Partial masking reduces the loudness of a test tone but does not mask the test tone completely. This effect often takes place in conversations. Partial masking is related to a reduction in loudness.

Masking effects can be measured not only when masker and test sound are presented simultaneously, but also when they are not simultaneous. In the latter case, the test sound has to be a short burst or sound impulse, which can be presented before the masker stimulus is switched on. The masking effect produced under these conditions is called pre-stimulus masking, shorted to "premasking" (the expression "backward masking" is also used). This effect is not very strong, but if the test sound is presented after the masker is switched off, then quite pronounced effects occur. Because the test sound is presented after the termination of the masker, the effect is called post-stimulus masking, shorted to "postmasking" (the expression "forward masking" is also used), as shown in Figure 2.5.

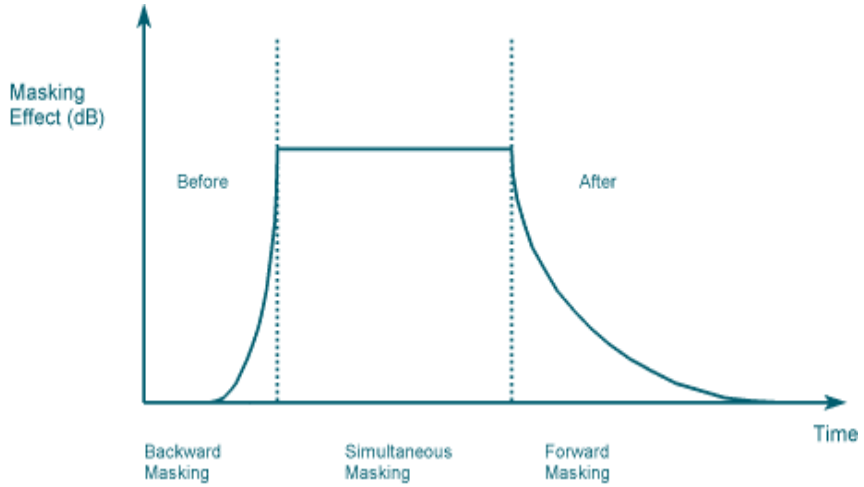

Figure 2.5 per (backward) and post- (forward) masking

Masking represents one of the most basic effects in psychoacoustics. This is normally determined as the audibility of pure tones in the presence of masking sounds. Different kinds of noises are commonly used in psychoacoustics as masker noises when investigating masking patterns.

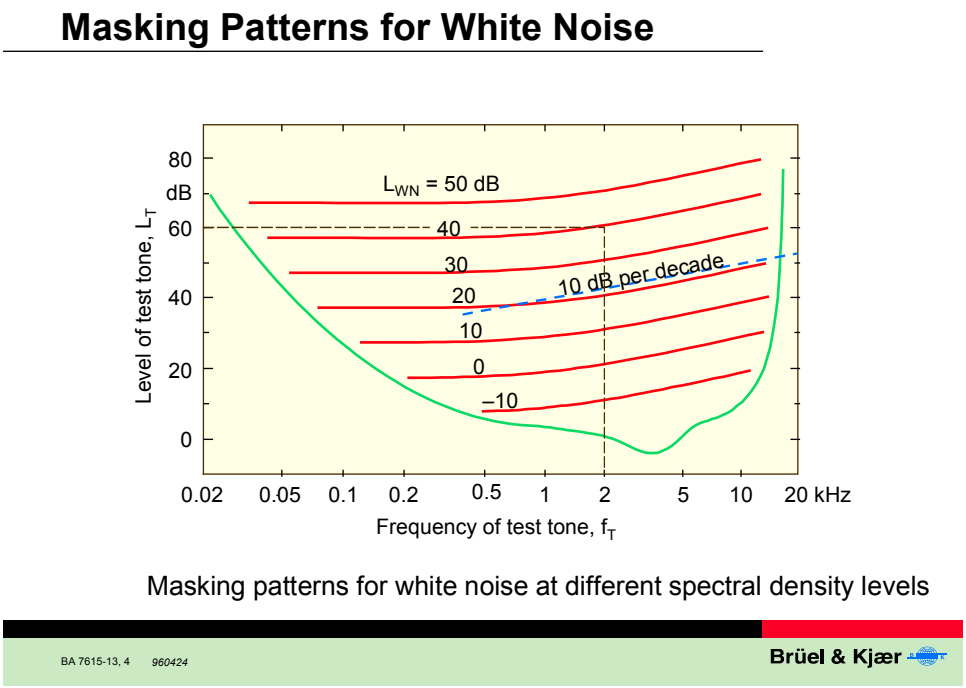

Figure 2.6 Masking patterns for white noise

This figure gives an example with white noise as a masker. The level of the just audible sound is given as a function of frequency. The lowest curve represents the threshold in quiet of the audibility of test tones without masker. The other curves represent masking patterns of white noise at different spectral density levels. If, for example, the level of a test tone L_T at 2 kHz is 60 dB or below, it will be masked if the white noise has a level L_{WN} of 40 dB. With increasing masking level, the masking patterns of white noise are shifted in parallel towards higher test tone level. Up to a test tone frequency of about 500 Hz, the masking patterns are horizontal, at higher frequencies an increase with a slope of about 10 dB per decade shows up. Since white noise has a spectral density level independent of frequency, the shape of the masking pattern is somewhat unexpected. However, it can be explained on the basis of critical bands described later in this lecture.

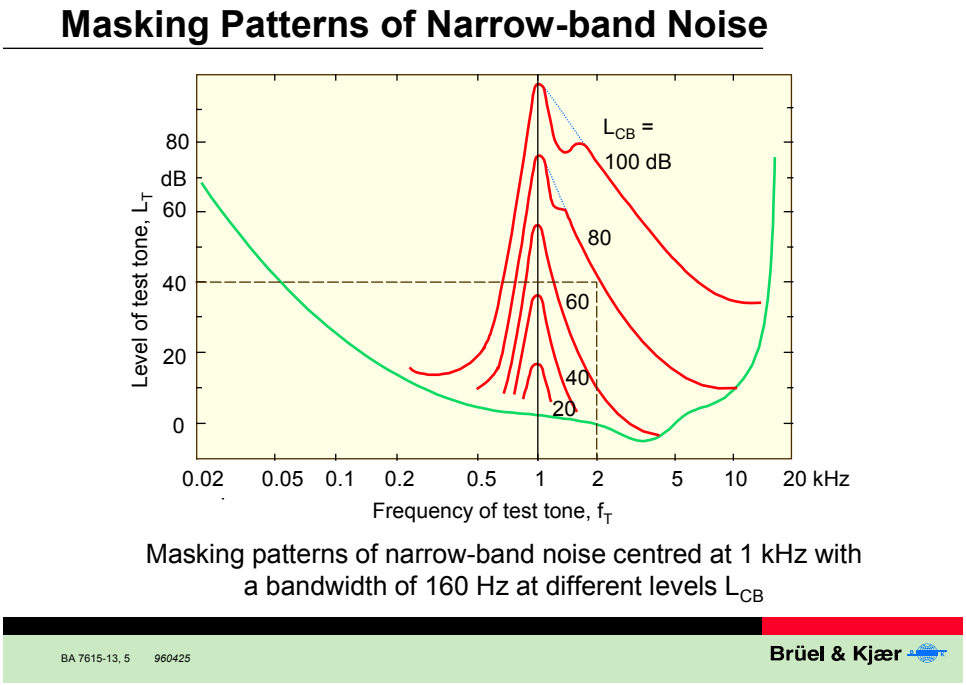

Figure 2.7 Masking patterns of narrow-band noise

This figure shows the masking patterns of a narrow-band noise centred at 1 kHz with a bandwidth of 160 Hz. The lowest curve represents the threshold in quiet. The other curves illustrate masking patterns for different levels of the narrow-band noise. For example, a test tone f_T at 2 kHz with a level L_T of 40 dB and below is masked if the noise level L_{CB} is above 80 dB. At low levels of the narrow band masker, the masking pattern has a symmetrical shape. However, when increasing the masker level above 40 dB, the lower level is shifted in parallel, whereas the upper slope gets flatter and flatter. This effect is called the “non-linear upward spread of masking”.

2.2.5 Critical Bands

Band is the frequencies, which are within two definite limits, the middle of which is called the centre frequency. The concept of critical bands is a basic feature of psychoacoustics. It is based on the assumption that the sound is analysed in the human hearing system by a bank of filters. In the following figure the bandwidth of these filters (critical bandwidth) is shown as a function of frequency - the solid line. The dashed lines illustrate useful approximations:

- At frequencies up to 500 Hz the bandwidth is constant at 100 Hz
- At higher frequencies the bandwidth is relative - about 20%

That means that at frequencies above 500 Hz, the critical bands can be compared with 1/3 octave-band filters, which have a relative bandwidth of 23%.

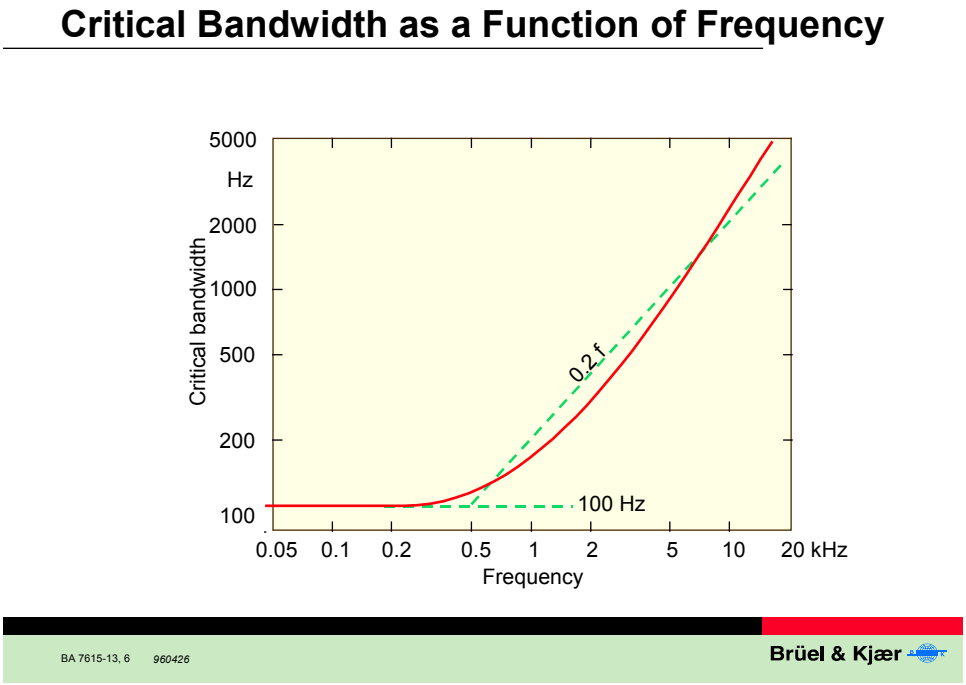

Figure 2.8 Critical bandwidth as a function of frequency

2.2.6 Bark Scale

The Bark scale is a frequency scale based on critical bands.

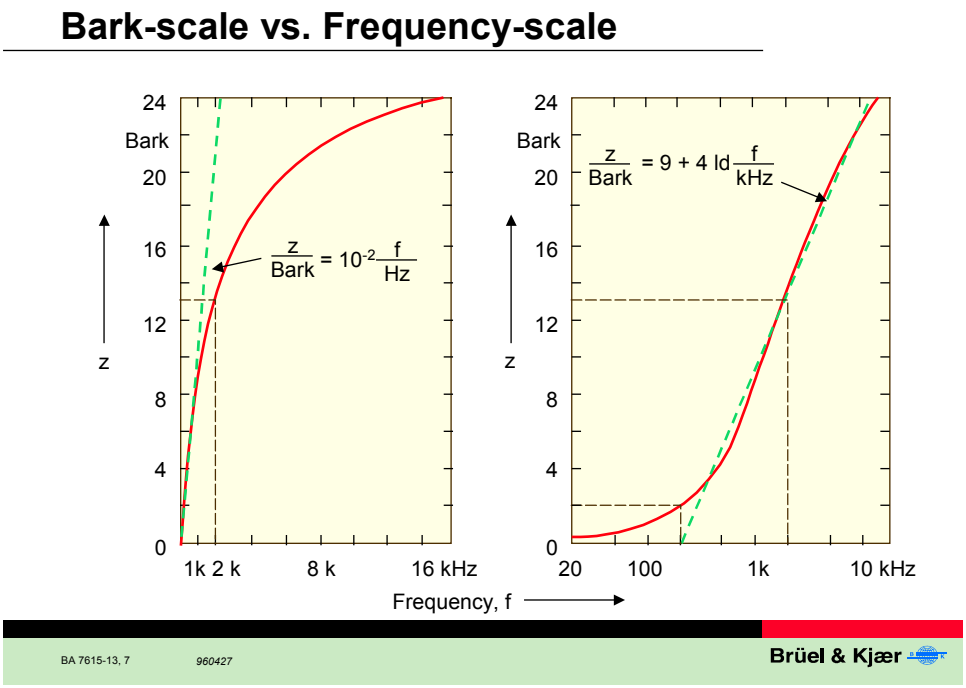

Figure 2.9 Bark scale vs frequency scale

In this figure we have compared the Bark scale with the frequency scale. In the left panel, the frequency scale is linear, in the right panel it is logarithmic. The solid

curves describe the relation between the Bark scale and the frequency scale. The dashed curves show the useful approximations. These are valid up to 500 Hz left panel, and above 500 Hz right panel. Examples of relation between Bark and frequency values:

- A frequency of 200 Hz corresponds to 2 Bark
- A frequency of 2 kHz corresponds to 13 Bark
- Bark band 1 covers the frequency range from 0 - 100 Hz
- Bark band 24 covers the frequency range from 12000 - 15500 Hz

The name “Bark” is chosen in honour of the late famous acoustician Barkhausen from Dresden.

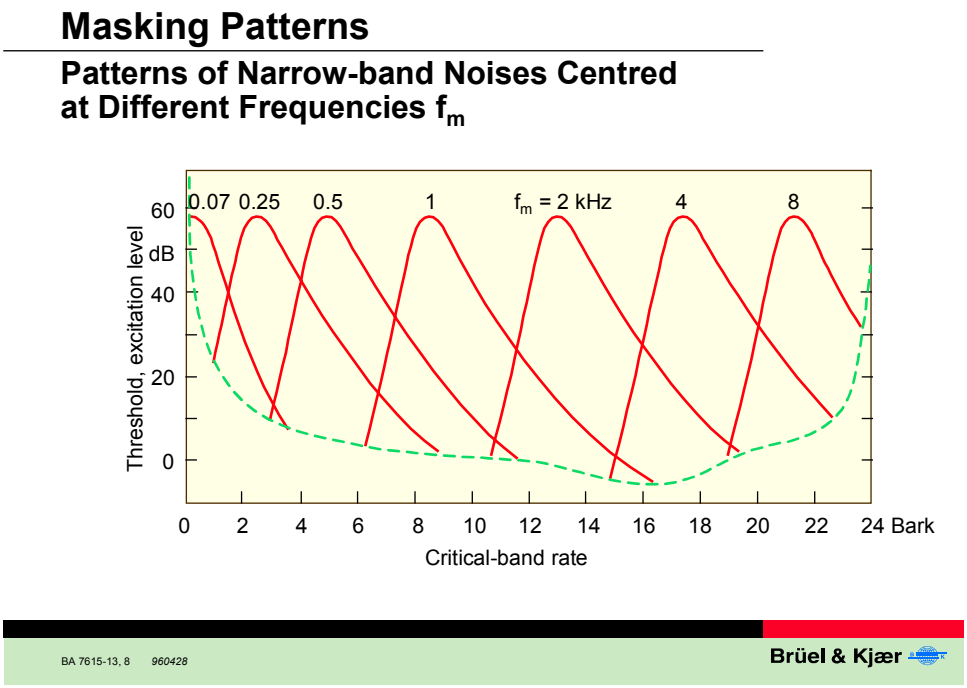

Figure 2.10 Masking patterns of narrow band noises centred at different frequencies

Here, one of the many advantages of the Bark scale is shown. The masking patterns of narrow-band noises 1 Bark wide, centred at different frequencies are plotted as solid curves. The dashed curve is the threshold in quiet. Plotted on the Bark scale they all have the same shape independent of the frequency and can be regarded as filter characteristics installed in the human hearing system.

2.2.7 Model of Virtual Pitch

A sophisticated model of virtual pitch has been elaborated by Terhardt. In general, the model is based on the fact that the first six to eight harmonics of a complex tone can be perceived as separate spectral pitches. These spectral pitches form the elements from which virtual pitch is extracted by a type of “Gestalt” recognition phenomenon. A visual analogue for the model of virtual pitch is illustrated in Figure 2.11. The word

"pitch" displayed on the top is produced by thin border lines as an analogue of a complex tone containing all the relevant harmonics. On the bottom of Figure 2.11, the letters are only indicated by parts of their borders analogue of a complex tone from which some of the basic features, the lower harmonics for example, have been removed. The two parts of Figure 2.11 are meant to illustrate the "philosophy" of the virtual pitch concept: from an incomplete set of basic features (incomplete border lines or incomplete spectral pitches) a complete image (the word "pitch" or the virtual pitch) is readily deduced by a mechanism of "Gestalt"[1] recognition. [2.1]

Figure 2.11 Visual analogue of the model of virtual pitch

The model of virtual pitch can be illustrated using Figure 2.12, which for didactical reasons, includes some simplifications. The influence of pitch shifts, for instance, is neglected at this stage. In the upper part of Figure 2.12, a complex tone with a fundamental frequency of 200Hz and from which the first two harmonics have been removed is displayed schematically. Both the harmonic number and the frequency of the spectral components are given. In the first stage, spectral pitches are derived (neglecting pitch shifts) from the spectral components, and a spectral weighting with a maximum around 600Hz is applied. Next, subharmonics are calculated for each spectral pitch present. Finally, the coincidence of the subharmonics of each spectral pitch is evaluated.

[1] Gestalt theory is used in pattern recognition.

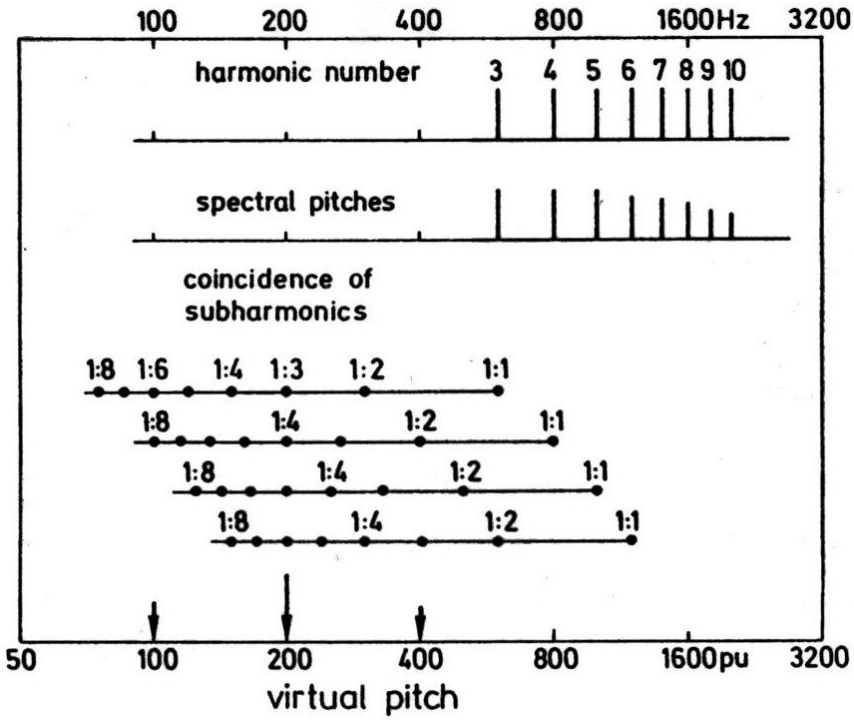

Figure 2.12 Illustration of the model of virtual pitch based on the coincidence of subharmonics, derived from the spectral pitches corresponding to the spectral lines of the complex tone.

For example, (again neglecting pitch shifts, that means the spectral pitch is equal to the frequency in value) the spectral component at 600 Hz is first transformed into a spectral at 600 pitch units (pu). Starting from this value, the first eight subharmonics which occur at 300pu, 200pu, 150pu, 120 pu, 100 pu, 85.7 pu, and 75 pu are calculated. In Figure 3.3, each of these subharmonics is indicated by a dot, and the corresponding ratio is given in numbers. The same procedure is performed with the next spectral component at 800Hz. In this case, we start from a spectral pitch at 800pu, get the first subharmonic at 400pu, the next at 266.7 pu, the next at 200pu, and so on. The same procedure is then applied for the spectral pitches at 1000pu and at 1200pu. In this way, an array of "yardsticks" containing dots representing the respective subharmonics is obtained. From this array, virtual pitch is deduced as follows: a scanning mechanism simply counts the number of dots that are contained in a narrow "pitch window", which is shifted like a cursor from left to right. At 200 pu in Figure 3.3, four dots are found in the window. A large number of coincident subharmonics indicates a strong virtual pitch and therefore this spot is marked by a long arrow on the virtual pitch scale. Near 100pu and 400pu two dots are found in the window, therefore two small arrows are plotted at the corresponding locations. The largest number of coincidences of subharmonics occurs near 200pu and the virtual pitch of the complex tone is calculated to be 200pu as indicated by the long arrow. However, near 100pu and 400pu, candidates for the calculated virtual pitch also occur, but with less weight. This means that the complex tone produces a virtual pitch corresponding to 200pu with some octave ambiguities (100 and 400pu) in both directions. Such octave ambiguities are often found in experiments on virtual pitch. In this case, however, a pure tone with a frequency a little below 200Hz will be matched to the pitch of the complex tone with the spectrum shown at the top of Figure 3.3. [2.1]

2.3 Sound Quality Metrics

The most important Parameters or Metrics used in Sound Quality are those based on Zwicker Loudness calculations. They are reflecting most of the psychoacoustics properties of the human perception of sound. They have the advantage that they put a single figure on characteristic properties of the sound. Three of the important ones are mentioned here:

- Fluctuation Strength is a measure of low frequency - around 4 Hz - frequency and amplitude modulation in the time sample and is based on a non-stationary loudness calculation.

- Roughness is similar to Fluctuation Strength apart from measuring the modulation around 70 Hz.

- Sharpness is a measure of the amount of high frequency content in the signals frequency spectrum. It can be calculated based on both a stationary and a non-stationary loudness calculation.

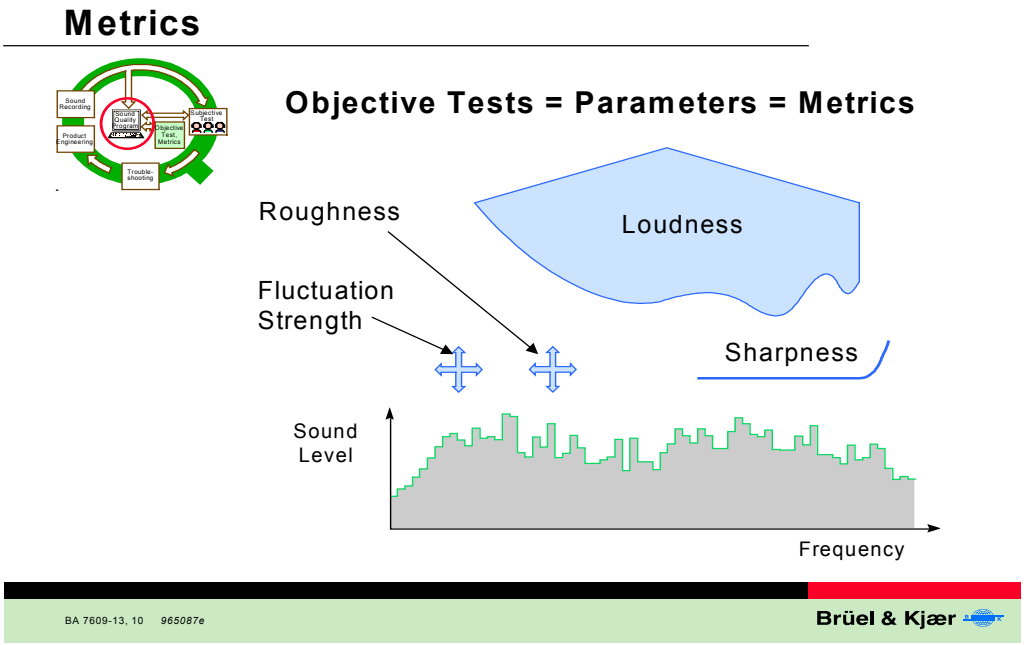

Figure 2.13 Metrics

Only Zwicker Loudness calculations for stationary signals are standardised. And most real life signals are non-stationary. Although there exists descriptions and formulas for Roughness, Fluctuation strength and Sharpness they are not very precise. That means that implementations of these Metrics from different manufacturers of Sound Quality analysis equipment will vary and can give different results. Efforts regarding standardisation are going on both in ANSI and DIN.

In addition, some other objective measurements – Metrics, are often used in Sound Quality evaluations.

- Pleasantness and Annoyance are combination Metrics based on a weighted sum of Zwicker Loudness, Fluctuation Strength, Roughness and Sharpness.

- Tone to Noise Ratio is a measure describing the amount of pure tones in the signal

- Prominence ratio is a description of the amount of noise in a critical band in relation to the noise in the adjacent bands.

- Tonality or Pitch is a measure of how strong the sensation of "frequency" is in a complex signal.

- Speech Interference Level, Articulation Index and Speech Transmission index are all measures related to the quality of a speech transmission channel. They also find uses in some Sound Quality applications.

- Kurtosis is a measure of impulsiveness of the time signal. Basically it sums up all time samples level differences from the signals mean value and raised to the power of 4 and then normalised. The method exaggerates the impulses in the sound and a high kurtosis value normally reflects poor Sound Quality.

2.4 Tonality Metric

Tonality is an important parameter affecting Sound Quality. The perceived tonal character of sounds plays an important role both in Sound Quality research and in the study of annoyance through noise. Tonality is not standardized yet so different companies have their own version of implementation. Tonality calculation is based on pitch-extraction algorithm described by Terhardt. The algorithm provides two pitch patterns: the spectral-pitch pattern and the virtual-pitch pattern, each of which consists of pitch values and pitch weights. For more information about the algorithm, please refer to Chapter 3.

2.5 Terms and Definitions

Before going into detail of the following sections, it is worth noting some of the basic concepts used in the Sound Quality field.

- **Attenuation**

Reduction in magnitude of a physical quantity such as sound, either by electronic means or by a physical barrier, including various absorptive materials.

- **Amplitude**

The instantaneous magnitude of an oscillating quantity such as sound pressure. The peak amplitude is the maximum value.

- **Band**

Frequencies, which are within two definite limits, the middle of which is called the centre frequency.

- **Bandwidth**

The difference between the highest and lowest frequencies of a band, sometimes expressed in standard sizes, such as octave, half-octave, third-octave.

- **Central frequency**

The frequency in the middle of a Band of frequencies, by which the band is identified together with the Bandwidth.

- **Constant percentage Bands (CPB)**

CPB means constant percentage bandwidth. It is a way of displaying data in octave form – in Sound Quality's case 1/3-octave spacing.

- **Decibel (dB)**

The decibel is not an absolute unit of measurement. It is a ratio between a measured quantity and an agreed reference level. The dB scale is logarithmic and uses the hearing threshold of 20 μ Pa as the reference level. This is defined as 0 dB.

The advantage of using dB's is that it converts the linear scale with large and unwieldy numbers into a much more manageable scale from 0 dB at the threshold of hearing (20 μ Pa) to 130 dB at the threshold of pain.

- **Diffuse field**

The sound is assumed to reach the listener's ears from all directions at the same intensity. This condition is approximated in an ordinary room. The method is applicable to all types of spectra and also is based on the assumption that the sound is steady rather than intermittent.

- **Fast Fourier Transform (FFT)**

The Fast Fourier Transform is an algorithm or calculation procedure for obtaining the Discrete Fourier Transform (DFT) with a greatly reduced number of arithmetic operations compared with a direct evaluation. Since its first publication in 1965 it has revolutionized the field of signal analysis, and it is still probably the most important single analysis technique available.

- **Filter**

An electrical device used to affect certain parts of the spectrum of a sound, by causing the attenuation of certain frequency bands, while allowing other bands to pass unattenuated. Some common types of filters are:

- *high-pass filters* (which attenuate low frequencies below the cut-off frequency);
- *low-pass filters* (which attenuate high frequencies above the cut-off frequency);
- *band-pass filters* (which combine both high-pass and low-pass functions as in the figure 7.2 below);
- *band-reject filters* (which perform the opposite function of the band-pass type);
- *octave, half-octave, third-octave, tenth-octave filters* (which pass a controllable amount of the spectrum in each band);
- *shelving filters* (which boost or attenuate all frequencies above or below the shelf point);
- *resonant or Formant filters* (with variable centre frequency and Q^2).

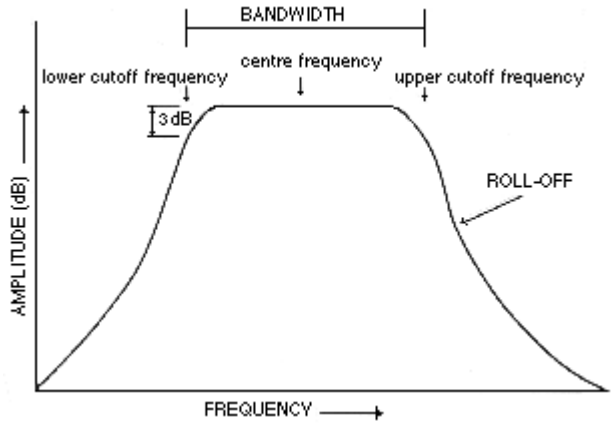

Figure 2.14 Generalized response characteristic of a band-pass filter.

- **Free (Frontal) field**

The sound is assumed to reach the listener's ears only from the direction straight ahead of person, in the open air or in a non-reflecting environment. In all enclosures, frontal sound is approximated when a small source is operating close to and directly ahead of the listener.

- **Fundamental Frequency**

If a sound is a complex of many tones of various frequency, amplitude and phase, repeating together in a basic cycle of definite frequency, the fundamental frequency is the lowest frequency of this complex.

[2] 1/N octave filters have a constant relative bandwidth, which means that the Q factor of the filters are the same.

- **Harmonic**

A harmonic tone consists of a sum of pure tones (each one called a partial), whose frequencies are in integer ratios of 1, 2, 3, ... Partials related in this way are called the harmonics of the tone.

- **Hertz**

The unit of frequency measurement, representing cycles per second.

- **Loudness**

Subjective impression of the intensity of a sound.

- **Masking**

The process by which threshold of audibility of one sound is raised by the presence of another (masking) sound.

- **Narrow band noise**

Sound classed as noise, which has its energy distributed over a relatively small section of the audible range.

- **Octave**

An octave is a doubling or halving of frequency. 20Hz-40Hz is often considered the bottom octave.

- **One Octave bands**

Frequency ranges in which the upper limit of each band is twice the lower limit and Octave bands are identified by their geometric mean frequency, or centre frequency.

- **One-third octave bands**

Frequency ranges where each octave is divided into one-third octaves with the upper frequency limit being 2* (1.26) times the lower frequency. Identified by the geometric mean frequency of each band.

- **Pascal, Pa**

A unit of pressure corresponding to a force of 1 newton acting uniformly upon an area of 1 square metre. Hence $1Pa = 1N/m^2$.

- **Phon**

A unit used to describe the loudness level of a given sound or noise of loudness level of a tone.

- **Pitch**

Pitch is a subjective term for the perceived frequency of a tone. Pitch of pure tones depends not only on frequency, but also on other parameters such as sound pressure level. Pitch shifts are used to identify the level influence of pitch perception. Complex tones can be regarded as the sum of several pure tones. The pitch of complex tones can be assessed by pitch matches with pure tones.

- **Sample**

The signals we use in the real world, such as our voices, are called "analog" signals. To process these signals in computers, we need to convert the signals to "digital" form. While an analog signal is continuous in both time and amplitude, a digital signal is discrete in both time and amplitude. To convert a signal from continuous time to discrete time, a process called sampling is used. The value of the signal is measured at certain intervals in time. Each measurement[3] is referred to as a sample. [3]

- **Sound Pressure Level**

Sound is defined as any pressure variation that the ear can detect ranging from the weakest sounds to sound levels that can damage hearing. When a sound source vibrates, it sets up pressure variations in the surrounding air.

The sound pressure level, L_p, expressed in dB's, of a sound or noise is given by

$$L_p = 20\log(p / p_0)dB$$

where:

p is the measured value in Pa, and

p_0 is a standardised reference level of $20\,\mu$ Pa--the threshold of hearing.

- **Sound Power**

Sound power is the energy emitted by a sound source per unit time. The symbol for sound power is **W** and its unit is the **watt**. (Named after the Scottish mechanical engineer James Watt, 1736-1819, of steam engine fame.)

- **Sound Intensity**

Sound intensity, at a point in the surrounding medium, is the power passing through a unit area. Its symbol is **I** and its unit, **watts/m^2**.

$$I = W / S$$

where:

[3] We can also understand the measurement as a collection of time-amplitude values.

W is the sound power in watts
and S is the surface area in m^2

- **Sound Intensity Level**

In plane traveling waves, sound pressure level and sound intensity level are related by

$$L = 20\log(p / p_0)\,\text{dB} = 10\log(I / I_0)\,\text{dB}$$

The reference value I_0 is defined as 10^{-12} **watts/m^2**.

Note that the sound intensity level and the sound pressure level are approximately numerically equal. This means you can use the value of sound pressure level instead of the value of noise intensity level in calculation.

- **Sound**

Sound is vibration disturbance, exciting hearing mechanisms, transmitted in a predictable manner determined by the medium through, which it propagates. To be audible the disturbance must fall within the frequency range 20Hz to 20,000Hz.

- **Sone**

A linear unit of loudness. The ration of the Loudness of a sound to that of a 1 kHz tone 40 dB above the threshold of hearing.

- **Spectrum**

Spectrum is the frequency content of a sound or audio signal, often displayed as a graphic representation of amplitude (or intensity level) against frequency. The spectrum of a sound may be determined by a sound analyser or by Fourier analysis and is distributed over the audible range (20 to 20,000 Hz). It is the distribution of the energy of a signal with frequency so it is also called ppower spectrum or averaged spectrum.

- **Specific loudness (sone/bark)**

The specific loudness is the loudness per critical band of a certain sound. If the sound does not have a low level, it produces a specific loudness in other critical bands than those corresponding to the physical sound.

- **Sub-harmonic**

An integer submultiple or fraction of a fundamental frequency and Subharmonic series consists of pitches related to the fundamental by ratios: 1/2, 1/3, 1/4, 1/5, 1/6, etc.

- **White noise**

Broadband noise having constant energy per unit of frequency.

- **Zwicker Loudness**

A model of loudness that provides a method of rating subjective loudness in a relatively objective way. It uses third octave measurements of sound pressure levels. It involves plotting these octave bands and finding the area under the curve to determine loudness.

Chapter 3 Review Tonality Algorithm

In this chapter, we will review the Algorithm for Tonality, which was generally for complex tone and presented by Terhardt, E., Stoll, G., and Seewann, M. in 1982 [3.1]. It is the theoretical basis of Tonality metric definition. We focus on the general procedure and relevant formulae, but not on details of all the psychoacoustics concepts. The readers who interested in more details, please refer to the relevant Psychoacoustics books.

The algorithm is based on the virtual-pitch theory. The purpose of the algorithm is to provide two pitch patterns: the spectral-pitch pattern and the virtual-pitch pattern, each of which consists of pitch (height) values (using "pu" as the unit, an abbreviation for "pitch unit") and pitch weights. The whole pitch percept is then described by a combination of the two patterns. The spectral-pitch pattern (SP pattern) is obtained by a special process of spectral analysis, which is based on established psychoacoustics principles such as masking and spectral dominance [Figure 3.1 (a) to (d)]. The SP pattern describes the pitch percepts directly associated to the tonal spectrum components. In addition, the SP pattern is the basis of the virtual pitch pattern (VP pattern). The latter is deduced from the former by a process of subharmonic coincidence assessment [Figure 3.1 (d) to (e)].

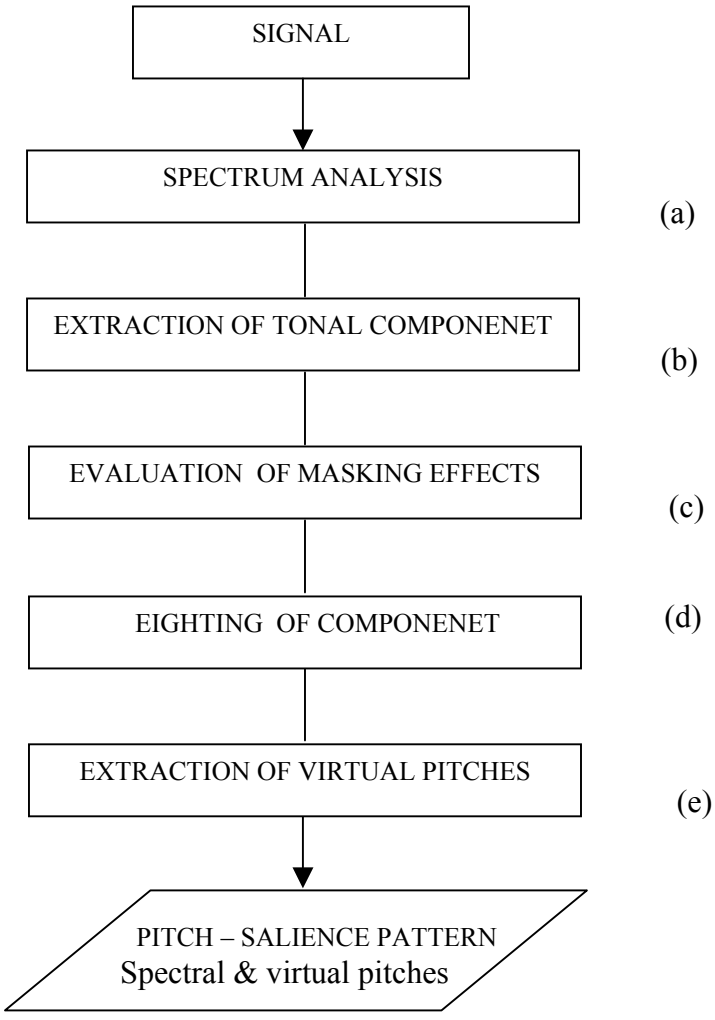

Figure 3.1 Survey on the pitch-evaluation procedure.

The steps of the algorithm given by Figure 3.1 will be covered in the following sections of this chapter.

3.1 Spectrum Analysis

The conventional digital Fourier analysis (FFT) is used for the spectrum analysis. Frequency region is about 20Hz to 5kHz which is relevant to the aurally region. The power spectrum is represented by 400 samples; thus the sample spacing is 12.5 Hz.

From a programmer point of view, we can say that a sample is a pair of frequency and SPL value. Moreover, A spectrum is a list of samples.

3.2 Extraction of Tonal Components

As a first step, candidates for tonal components are detected by scanning the spectral samples and testing if

$$L_{i-1} < L_i \geq L_{i+1} \qquad (1)$$ [4]

where:

- L_i is the relative SPL of the ith spectrum sample,
- L_{i-1} is the relative SPL of the next lower sample,
- L_{i+1} is the relative SPL of the next higher sample.

If a candidate was found by this criterion, it is further tested whether each of the conditions

$$L_i - L_{i+j} \geq 7\,dB \;;\; j = -3,-2,+2,+3, \qquad (2)$$

is fulfilled. If this is the case, it is assumed that the considered group of seven spectral samples (i.e., i-3; i-2; …i+3) represents a tonal component. The differential threshold value of 7 dB in Eq.(2) was determined empirically by analyzing several complex sounds.

The local maximum samples in a spectrum are those which are satisfied by Eq.(1) and Eq.(2). But the tonal components are a little different from local maximum samples because we calculate the centre frequency in an approximate way as follows.

A good approximation to the precise component frequency f_c is obtained by the interpolation formula.

$$f_c = f_i + 0.46\ (Hz\ /\ dB\ )(L_{i+1} - L_{i-1}) \qquad (3)$$

where.

- f_i is the frequency of ith spectrum sample;

[4] Eq.(1) means $L_{i-1} < L_i$ and $L_{i+1} \leq L_i$

- L_{i+1} and L_{i-1} is from the Eq.(1).

You can see from Eq.(3) that instead of using the frequency of the local maximum sample directly, the neighboring SPL values L_{i+1} and L_{i-1} are taken into account for calculating the frequency of tonal component.

In any case, the input to the next processing stage comprises the following parameters: The number N of tonal components, the frequency f_c, and the SPL L_c ($=L_i$) of every component.

3.3 Evaluation of Masking Effects

Mutual masking of spectral components has two relevant effects. The first is that components may be inaudible and thus irrelevant for the pitch percept, while others may be more or less reduced in their individual audibility. The second effect is that the individual spectral pitches of aurally relevant components will be shifted more or less substantially, due to mutual partial masking. These two effects are evaluated quantitatively as follows.

3.3.1 Sound-pressure Level Excess

The extent to which a tonal component is "aurally relevant" is described by the so-called sound-pressure level excess (SPL excess) LX.

Formally, the SPL excess of the μth tonal component, $LX_\mu (1 \le \mu \le N)$ is specified by

$$LX_\mu = L_\mu - 10 \log_{10} \left[\left(\sum_{\substack{v=1 \\ v \neq \mu}}^{N} 10^{L_{Ev}(f_\mu)/(20\,dB)} \right)^2 + I_{N\mu} + 10^{L_{TH}(f_\mu)/(10\,dB)} \right] dB \qquad (4)$$

- L_μ is the component's SPL as determined in the previous extraction procedure.
- $L_{Ev}(f_\mu)$ is the excitation level which is produced at the frequency f_μ by the vth tonal component (notice that v is incremented from 1 to N, skipping μ).
- $I_{N\mu}$ is the noise intensity present in the critical band around the considered tonal component, is used to take account of the additional masking effect of "noise" spectral samples. It will be specified below.
- $L_{TH}(f_\mu)$ is the hearing threshold at the frequency f_μ.

The excitation level $L_{Ev}(f_\mu)$ is specified by

$$L_{Ev}(f_\mu) = L_v - s(z_v - z_\mu) \qquad (5)$$

- L_v is the SPL of the vth tonal component.

- z represents the critical-band rate. Thus z_μ and z_v are the critical-band rates of the μth and vth components, respectively. The relationship between frequency and critical-band rate is with high accuracy provided by

$$z = \{13 \arctan [0.76 (f / kHz)] + 3.5 \arctan(f / 7.5\,kHz)^2\}\, Bark \quad (6)$$

- Equation (5) represents the triangular shape of the excitation level critical-band rate pattern, where s depicts the steepness of the slopes. The parameter s is specified by

$$s = 27\ dB / Bark, \quad \text{if } f_\mu \le f_v; \quad (7a)$$

$$s = [-24 - (0.23\ kHz / f_v) + (0.2\, L_v / dB)]\, dB / Bark, \quad \text{if } f_\mu > f_v. \quad (7b)$$

The noise intensity $I_{N\mu}$ is obtained by adding the sound intensities of those spectrum samples, which correspond to the particular critical-band rate interval extending from ($z_\mu - 0.5$ Bark) to ($z_\mu + 0.5$ Bark), skipping the five central samples of every tonal component which eventually has been detected in that critical band (i.e., the samples $i-2,\ i-1,\ i,\ i+1,\ and\ i+2$ where i indicates the locally maximal sample defined in Sec. 3.2).

The threshold of hearing, which in Eq. (4) is represented by L_{TH}, is adequately specified by the formula

$$L_{TH}(f_\mu) = \{3.64 (f_\mu / kHz)^{-0.8} - 6.5 \exp[-0.6 (f_\mu / kHz - 3.3)^2] + 10^{-3} (f_\mu / kHz)^4\}\, dB \quad (8)$$

When the masking evaluation enabled by Eqs. (4) to (8) is carried out for all the tonal components(i.e., $\mu = 1$ to N), the so-called SPL excess pattern is obtained. Positive values of LX indicate aural relevance, while tonal components whose LX is zero or negative are considered as irrelevant with respect to any type of pitch percept. The SPL excess pattern is considered as being representative of the tonal aspects of the stimulus.

3.3.2 Pitch Shifts

Due to interaction of simultaneous spectral components in the auditory system, the spectral pitches are not exactly identical to the pitches of isolated tones with the same frequency.

The individual spectral pitch of the μth component is specified by

$$H_\mu = (f_\mu / Hz)(1 + v_\mu)\, pu, \quad (9)$$

- H_μ is the spectral pitch of the μth component, it is measured in principle on a scale proportional to frequency, the unit is "pitch units," pu.
- ν_μ is the induced pitch shifts (a quantity of maximally a few percent) specified by the formula

$$\nu_\mu = 2.10^{-4}\left(\frac{L_\mu}{dB} - 60\right)\left(\frac{f_\mu}{KHz} - 2\right) + 1.5 \cdot 10^{-2} \exp\left(\frac{-LX'_\mu}{20\ dB}\right)$$
$$\left(3 - \ln\frac{f_\mu}{KHz}\right) + 3.10^{-2} \exp\left(\frac{-LX''_\mu}{20\ dB}\right)\left(0.36 + \ln\frac{f_\mu}{KHz}\right) \qquad (10)$$

  - L_μ and f_μ are the SPL and frequency of the considered component, respectively.
  - LX'_μ And LX''_μ are specific representations of the SPL excess, pertaining to the interference of the considered component with the lower (LX'_μ) and the higher
    (LX''_μ) ones. These specific SPL excess values are given by

$$LX'_\mu = L_\mu - 20 \log_{10} \sum_{v=1}^{\mu-1} 10^{L_{Ev}(f_\mu / 20\ dB)} dB\ ; \qquad (11a)$$

$$LX''_\mu = L_\mu - 20 \log_{10} \sum_{v=\mu+1}^{N} 10^{L_{Ev}(f_\mu / 20\ dB)} dB\ ; \qquad (11b)$$

The calculation of ν_μ is only required for those tonal components whose SPL excess is greater than 0. The pitch measure given by Eqs. (9) to (11) is called *true spectral pitch*. It is numerically identical to the frequency of a single pure tone at 60 dB SPL which was matched in pitch to that spectral component whose pitch is to be depicted. If the pitch shifts described by Eqs. (10) and (11) are ignored (which can be done in many cases), *nominal spectral pitch* is obtained; it is numerically identical to the considered component's frequency.

The data provided by the evaluation of masking algorithm to further processing consist of (1) a number R of relevant tonal components ($R \leq N$), (2) the SPL excess, and (3) the spectral pitch of each of these components.

From a programmer's point of view, we can say that the SPL excesses is a list of frequency and SPL excess pairs, and similar to the spectral pitches which is a list of frequency and spectral pitch pairs.

3.4 Weighting of Components

The extend to which each tonal component contributes to the entire tonal percept is considered to depend on (1) the SPL excess and (2) the frequency of the component. The weight of a particular spectral pitch is specified by

$$WS_{\mu} = \left[1 - \exp\left(\frac{-LX_{\mu}}{15\ dB}\right)\right]\left[1 + 0.07\left(\frac{f_{\mu}}{0.7\ KHz} - \frac{0.7\ KHz}{f_{\mu}}\right)^{2}\right]^{-1/2},$$

$$\text{if } LX_{\mu} \geq 0; \tag{12a}$$

$$WS_{\mu} = 0, \quad \text{if} \quad LX_{\mu} < 0 \tag{12b}$$

The pattern of spectral-pitch weights *WS* versus spectral pitch represented by equivalent frequency is the ultimate basis. This pattern is called the *spectrum-pitch pattern* (SP pattern). It is thought to represent the relative salience of competing simultaneous spectral pitches. The data relevant for the formation of virtual pitch are provided by the SP pattern. This process is described in the following section.

3.5 Evaluation of Virtual Pitch

Following the principle of virtual-pitch theory, every virtual-pitch candidate is specified as a subharmonic of one of the relevant spectral pitches. If pitch shifts of spectral components and stretch of subharmonic intervals are ignored, the virtual pitch represented by the *mth* subharmonic of the *ith* relevant component is

$$H_{im} = m^{-1}(f_i / Hz)\,pu \tag{13}$$

It is called nominal virtual pitch. But in many cases it is of interest to evaluate also the effects of pitch shifts and subharmonic interval stretch. The true virtual pitch is to be calculated by the formula

$$\begin{aligned} H_{im} &= m^{-1}(f_i / Hz)(1 + v_i - sign(m-1)10^{-3}\{18 + 2.5m \\ &-(50-7m)(f_i / KHz)m^{-1} + 0.1\left[m^{-1}(f_i / KHz)\right]^{-2}\})\,pu \end{aligned} \tag{14}$$

The eventual pitch shift of the *ith* component is taken into account (v_i), and the subharmonic interval stretch is represented by the terms which follow the $sign(m-1)$ function.

Each of the relevant components ($i = 1...R$) provides M potential virtual pitches, where M is the maximal subharmonic number taken into consideration, i.e., $m = 1...M$. In principle, it requires testing of near coincidence of every subharmonic of every relevant component. However, by an adequate strategy it can be accomplished that the most significant virtual pitches will be extracted early in the testing process such that testing can be stopped at an appropriate stage with

sufficiently high probability that all significant pitches have been extracted. The procedure following such a strategy is shown in Figure 3.2.

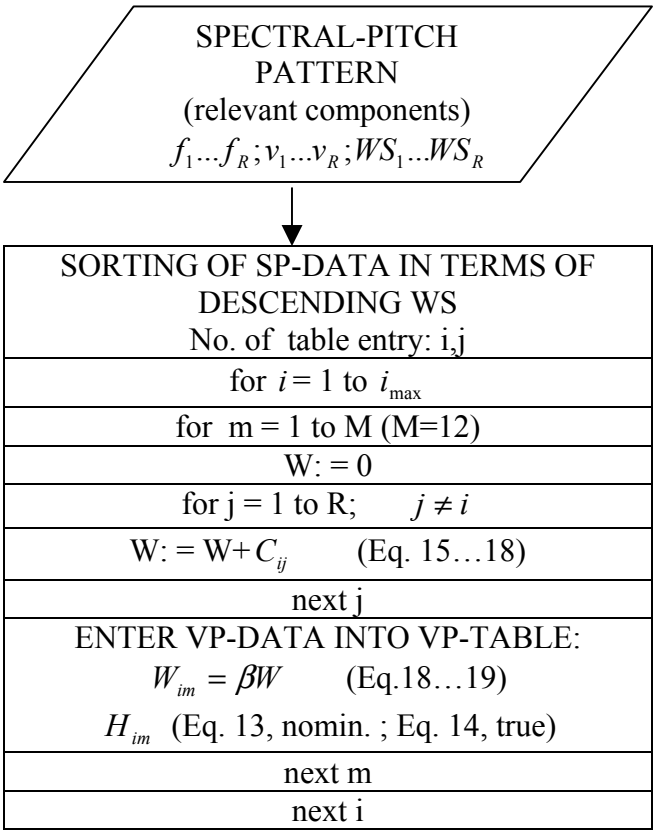

Figure 3.2 Algorithm of the extraction of virtual pitch from the spectral-pitch pattern

First the spectral-pitch data are sorted in terms of descending spectral-pitch weight WS, such that the most prominent spectral pitch is represented on top of a table which contains the frequencies, pitch shifts, and spectral-pitch weights.

It was empirically found that it is sufficient to take only those components as "i components" into account whose SP weight is at lease 70% of the largest one. Thus a certain i_{max} is specified such that $WS(i_{max}) \geq 0.7WS(i=1)$, and i is incremented only from 1 to i_{max}. By this strategy, computation time is considerably reduced while the probability of missing important virtual pitches is very low.

M is typically equal to 12.

R is the number of the relevant tonal components.

The coincidence coefficient C_{ij} is specified as follows

$$C_{ij} = (WS_i WS_j / mn)^{1/2}(1-\gamma/\delta), \text{if } \gamma \leq \delta, \quad (15a)$$

$$C_{ij} = 0, \text{ if } \gamma > \delta \text{ and/or } n > 20 \quad (15b)$$

- WS_i and WS_j are the spectral-pitch weights of the *ith* and *jth* relevant components respectively;

- m and n are the respective subharmonic numbers. While m is set by the algorithm, n must for each coincidence test be determined from the formula

$$n = Int(mf_j / f_i + 0.5), \tag{16}$$

- γ is the coefficient which represents the degree of inharmonicity of the tested component pair.

$$\gamma = \left|(nf_i / f_j) - 1\right| \tag{17}$$

- δ is the width of the coincidence interval which is specified by a constant, 0.08.

The weight W_{im} which is assigned to the *mth* subharmonic of the *ith* relevant component is essentially the sum of the coincidence coefficients provided by the tested component pairs:

$$W_{im} = \beta \sum_{\substack{j=1 \\ j \neq i}}^{R} C_{ij} \tag{18}$$

- β is a factor. It is called the "fundamental frequency weight." It has a low-pass characteristic specified by

$$\beta = \left[1 + (H_{im} / 800\,pu)^4\right]^{-1} \tag{19}$$

The evaluation of virtual pitch leads to the *virtual-pitch pattern* (VP pattern), which contains a list of frequency, virtual pitch and virtual pitch weight triples.

The algorithm described here implies that in an actual listening situation, neither of the two pitch modes is completely suppressed; rather, the whole pitch percept is described as a competition between spectral and virtual pitches. These two types of pitch are extracted from the signal and represented by the spectral-pitch pattern and the virtual-pitch pattern.

Chapter 4 Tonality Model in MATLAB

This chapter introduces MATLAB, M-file, shows the Data Flow of Tonality model, explains how the Data Flow is related with the algorithm and the MATLAB program, and finally presents some results from model test.

4.1 What is MATLAB?

The name MATLAB stands for matrix laboratory. It is a high-performance language for technical computing. It integrates computation, visualization, and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation. Typical uses include:

- Math and computation
- Algorithm development
- Modelling, simulation, and prototyping
- Data analysis, exploration, and visualization
- Scientific and engineering graphics
- Application development, including graphical user interface building

MATLAB is an interactive system whose basic data element is an array that does not require dimensioning. This allows you to solve many technical computing problems, especially those with matrix and vector formulations, in a fraction of the time it would take to write a program in a scalar noninteractive language such as C or Fortran.

The name MATLAB stands for matrix laboratory. MATLAB was originally written to provide easy access to matrix software developed by the LINPACK and EISPACK projects. Today, MATLAB uses software developed by the LAPACK and ARPACK projects, which together represent the state-of-the-art in software for matrix computation.

MATLAB has evolved over a period of years with input from many users. In university environments, it is the standard instructional tool for introductory and advanced courses in mathematics, engineering, and science. In industry, MATLAB is the tool of choice for high-productivity research, development, and analysis.

4.2 What is M-file

MATLAB provides a full programming language that enables you to write a series of MATLAB statements into a file and then execute them with a single command. You write your program in an ordinary text file, giving the file a name of filename.m. The term you use for filename becomes the new command that MATLAB associates with the program. The file extension of .m makes this a MATLAB M-file.

There are two kinds of M-files in MATLAB, Script M-files and Function M-files. Script M-files do not accept input arguments or return output arguments, while Function M-files can accept input arguments and return output arguments. Script M-files operate on data in the workspace, and are useful for automating a series of steps you need to perform many times, while in Function M-files, internal variables are

local to the function by default, and Function M-files are useful for extending the MATLAB language for your application. An M-file might contain one main function and zero, one or more sub functions, and the M-file name must be the same as the main function name. [4.1]

4.3 Data Flow of Tonality Model

The Data Flow for Tonality model is shown in Figure 4.1. It is based on the pitch-evaluation procedure we presented in Figure 3.1 in Chapter 3 and considered how to implement the algorithm in MATLAB programming language.

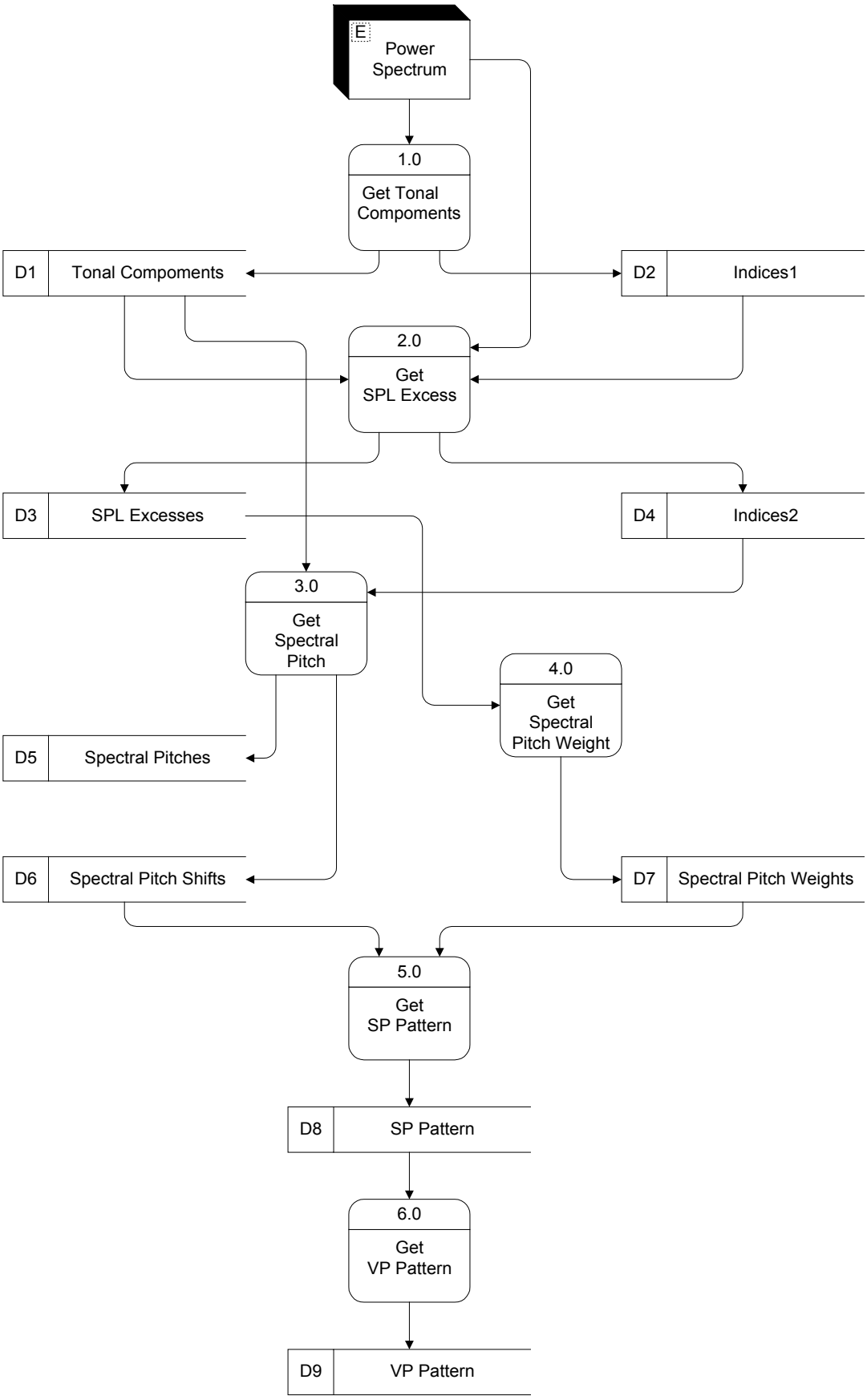

Figure 4.1 DFD for Tonality Model.

Power spectrum is the input of the whole model. We get power spectrum from B&K Sound Quality Type 7698 and save it into Excel, then import to MATLAB as a two-column matrix:

Frequency (Hz)	SPL (dB)

Functions 1.0 – 6.0 are main functions in our MATLAB model. Datastores D1 – D9 are the input or output of the relevant functions, and they are represented in MATLAB as matrices. A matrix is a rectangular array of numbers.

Functions 1.0-5.0 are the calculation procedure for Spectral Pitch Pattern. Function 6.0 is for Virtual Pitch Pattern.

Function 1.0 implements the procedure of Section 3.2, which extracts the tonal components from power spectrum. Relevant M-file is get_tonal_comp.m in Appendix A.

Function 2.0 implements the procedure of Section 3.3.1, which calculate the sound-pressure level excess. Relevant M-file is get_spl_excess.m in Appendix A.

Function 3.0 implements the procedure of Section 3.3.2. It calculates the true spectral pitch by evaluating the spectral pitch shift of each relevant tonal component, whose sound-pressure level excess is larger than zero. The relevant M-file is get_spectrum_pitch.m in Appendix A.

Function 4.0 implements the procedure of Section 3.4, which calculates the weighting of components whose sound-pressure level excess is larger than zero. The relevant M-file is get_sp_weight.m in Appendix A.

Function 5.0 combines all the necessary information for Spectral Pitch Pattern. The relevant M-file is get_SP_Pattern.m in Appendix A.

Function 6.0 is related to Section 3.5, which evaluates the true virtual pitch and get Virtual Pitch Pattern. The relevant M-file is get_VP_Pattern.m in Appendix A.

D1 contains Tonal Components represented in MATLAB as a two-column matrix:

Frequency (Hz)	SPL (dB)

D2 (Indices1) is the Power Spectrum row indices used to remember locations of the local maximal samples. Local maximal samples satisfy Eq. (1) and Eq. (2) in Chapter 3. It is represented in MATLAB as a one-column matrix:

Row Indices

D3 contains SPL Excesses information represented in MATLAB as a two-column matrix:

Frequency (Hz)	SPL Excess (dB)

D4 (Indices2) is the Tonal Components row indices, whose SPL Excess is larger than zero. It is represented in MATLAB as a one-column matrix:

Row Indices

Design of D2 and D4 is for remembering the interesting locations for programming purpose.

D5 contains Spectral Pitches information represented in MATLAB as a two-column matrix:

Frequency (Hz)	Spectral Pitch (pu)

D6 contains Spectral Pitch Shifts information represented in MATLAB as a two-column matrix:

Frequency (Hz)	Spectral Pitch Shift

D7 contains Spectral Pitch Weights information represented in MATLAB as a two-column matrix:

Frequency (Hz)	Spectral Pitch Weight

D8 contains SP Pattern information represented in MATLAB as a three-column matrix:

Frequency (Hz)	Spectral Pitch Shift	Spectral Pitch Weight

D9 contains VP Pattern information represented in MATLAB as a three-column matrix:

Frequency (Hz)	Virtual Pitch (pu)	Virtual Pitch Weight

4.4 Model Simplification and Modification

The different threshold value of 7 dB in Eq. (2) in Chapter 3 was determined empirically by analysing several complex sounds. But when we test our MATLAB model with some natural speech vowels, we found that the 7 dB is so big that most of the cases, there is no tonal components or very few tonal components can be extracted. But in order to test the other pitch-evaluation procedure we must have some tonal components data. So we change it to 3 dB just for programming and testing purpose.

According to the algorithm of virtual pitch evaluation, which is shown in Figure 3. 4 in Chapter 3, we will get a big number of virtual pitches, which is 12 times the number of spectral pitches. But we say that only the five most prominent virtual pitches, which have the higher weight values, are our concern.

4.5 The Relations of Different M-files in Tonality MATLAB Model

Functions 1.0-6.0 in Figure 4.1 are implemented as M-files. Each of them has main function and some of them have sub functions. Besides all these main functions and sub functions, we also have some auxiliary functions, which are shared by different main functions or sub functions.

Figure 4.2 and Figure 4.3 only show the most related functions in our MATLAB model. Each rectangular means one M-file. The first function name is the main function name, which should be the same as the M-file name. The following function(s) are sub function(s), which are defined in the same M-file. If there is an arrow, from function A to function B, then it means that function A uses function B in its definition. The purpose of presenting functions in this way is to show the layers of different functions.

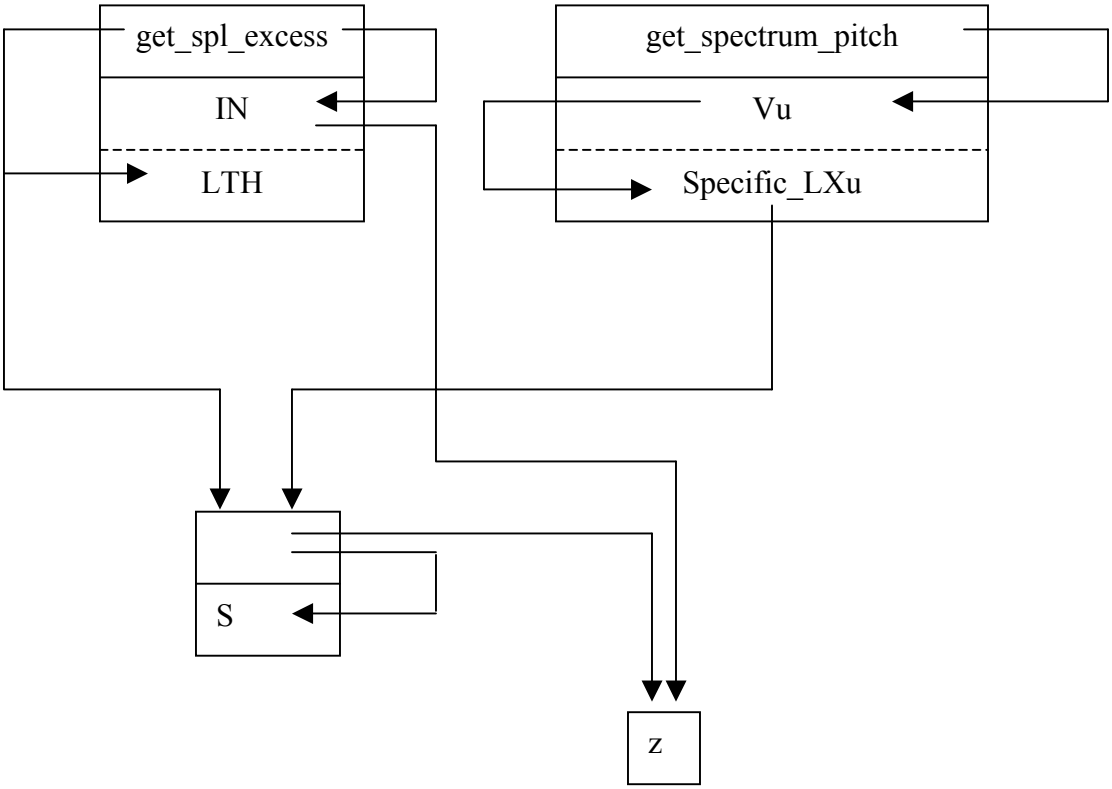

Figure 4.2 Relations of main, sub and auxiliary functions in different M-files.

Figure 4.2 shows the functions and sub functions relationship among four M-files, namely, get_spl_excess.m, get_spectrum_pitch.m, Lev.m and z.m. Function get_spl_excess is the main function to calculate the SPL excesses of tonal components. IN is a sub function to calculate Noise Intensity, present in the critical band around the considered tonal component. LTH is a sub function to calculate the hearing threshold at a certain frequency. Function get_spectrum_pitch is the main function to calculate the spectral pitches for the relevant tonal components whose SPL excess is larger than zero. Vu calculates the induced pitch shifts. Specific_LXu calculates the Specific representations of the SPL excess. Function Lev and z are two

auxiliary functions. They are defined in different M-files. Lev calculates the excitation level, which produced at a certain frequency by one tonal component. S calculates the steepness of the slopes in the excitation level critical-band rate patern. Function S is a sub function in M-file Lev.m. Function z calculates the critical-band rate (Bark) of a certain frequency. Funciton Lev is shared by get_spl_excess and Specific_LXu. Function z is shared by Lev and IN.

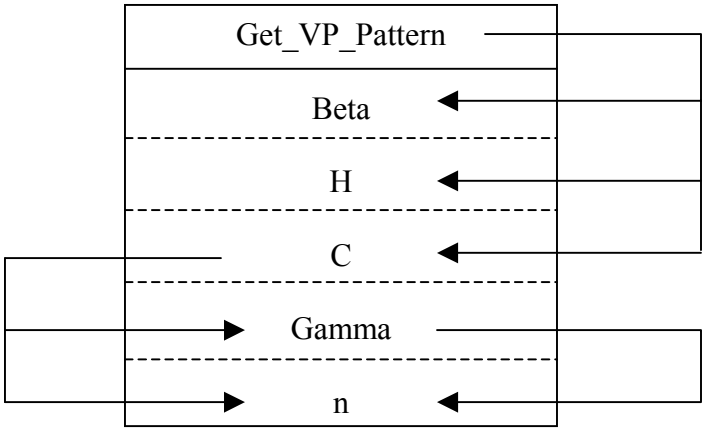

Figure 4.3 Relations of main and sub functions in M-file: get_VP_Pattern.m

Figure 4.3 shows the relations of the main function and different sub functions in M-file: get_VP_Pattern.m. Sub function Beta calculates a factor, called the 'fundamental frequency weight'. H calculates the true virtual pitch. C calculates the coincidence coefficient. Gamma calculates the inharmonicity coefficient. Sub function n is related with Eq.16 in Chapter 3.

4.6 A Function M-file Example

Let us take M-file: get_tonal_comp.m as an example and see how M-file looks like in MATLAB and how to use it in MATLAB Command window.

```
function [y,z] = Get_tonal_comp(x)
%Extraction of tonal components.
%Input:  power spectrum
%Output: tonal components.

%Check whether the input matrix validate.
[m,n] = size(x);
if (~(n == 2) | (m == 1 & n == 1))
  error('Input must be a two column vector!')
end

k = 1;
for i = 4:(length(x)-3)

        %Look for local maximal samples.
        if x(i,2)>=x(i+1,2) & x(i,2)>x(i-1,2) & x(i,2)-x(i-3,2)>=3 & x(i,2)-x(i-2,2)>=3 &
        x(i,2)-x(i+2,2)>=3 & x(i,2)-x(i+3,2)>=3
```

```
                %Calculate and save the frequency and the SPL values for tonal components.
                y(k,1)=x(i,1)+0.46*(x(i+1,2)-x(i-1,2))
                y(k,2)=x(i,2)

                %Check whether you want to remember the power spectrum indices of the
                %local maximal samples.
                if (nargout == 2)
                        z(k,1)=i
                end
                k=k+1;
        end
end
```

Input x is the power spectrum which contains 401 samples. Output y is the tonal components, and output z is the row indices of the local maximal samples.

The if statement:

```
if x(i,2)>=x(i+1,2) & x(i,2)>x(i-1,2) & x(i,2)-x(i-3,2)>=7 & x(i,2)-x(i-2,2)>=7 &
x(i,2)-x(i+2,2)>=7 & x(i,2)-x(i+3,2)>=7
```

is related to Eq.(1) and Eq.(2) in Chapter 3.

The statement:

```
y(k,1)=x(i,1)+0.46*(x(i+1,2)-x(i-1,2))
```

is related to Eq.(3) in Chapter 3.

If we type the following command in MATLAB Command Window,

```
[tonal_components, indices1] = get_tonal_comp(power_spectrum)
```

The input: power_spectrum contains the same data as Test 1 in Appendix C. We will get the following results,

```
tonal_components =

  1.0e+003 *

    0.3879    0.0581
    0.7660    0.0332
    0.9891    0.0265
    1.5282    0.0272
    1.8718    0.0397
    2.0781    0.0467
    2.5091    0.0417
    2.6367    0.0439
    2.8638    0.0372
    3.0911    0.0350
    3.3482    0.0406
    3.4672    0.0391
```

```
   3.5950    0.0350
   3.8641    0.0268


indices1 =

    37
    72
    93
   143
   175
   194
   234
   246
   267
   288
   312
   323
   335
   360
```

The data in variable tonal_components is a list of frequency and SPL values. They will be used in the later SPL Excess and Spectral Pitch calculations, and also being used to generate the plot for Tonal Components. The data in variable indices1 is the row indices of the local maximal samples. The relations between local maximal sample and tonal components can be found in Section 3.2. More information about the other MATLAB functions can be found in Appendix A.

4.7 How to Show the Computational Results?

After we have the matrices for power spectrum, tonal components, SPL excesses, spectral pitches pattern and virtual pitch pattern, we want to present them in one figure and list them as different plots. We wrote a Script M-file, namely “show.m” which use MATLAB functions “subplot” and some auxiliary functions defined by us, such as “show_spectrum” and “show_stem”.

```
function show(ps)
%show all the figures for the model

%show the power spectrum in first plot
%'show_spectrum' is a function we defined using Matlab's 'plot' function
subplot(5,1,1); show_spectrum(ps)

%get tonal components
[tc1,idx1]=Get_tonal_comp(ps)

%show the tonal components in second plot
%'show_stem' is a function we defined using Matlab's 'stem' function, y is between 1
%to 5.
subplot(5,1,2); show_stem(tc1)
ylabel('SPL(dB)')
```

```
%get the SPL excess
[tc2,idx2]=Get_spl_excess(ps,tc1,idx1)

%show the SPL excess value in third plot
subplot(5,1,3); show_stem(tc2)
ylabel('SPL EXCESS')

%calculate spectral pitch, pitch shift, spectral weight
[spec_pitch,pitch_shift]=Get_spectrum_pitch(tc1,idx2)
spec_weight=Get_sp_weight(tc2)

%show WS in fourth plot
subplot(5,1,4); show_stem(spec_weight)
ylim([0 1])
ylabel('WS')

%calculate virtual pitch
vpp_input=Get_SP_Pattern(pitch_shift,spec_weight)
vpp_output=Get_VP_Pattern(vpp_input)

%show Virtual Pitch weight in fifth plot
subplot(5,1,5); stem(vpp_output(:,1)/1000,vpp_output(:,3),'fill')
xlim([0 5])
xlabel('Frequency(KHz)')
ylabel('WP')
```

The auxiliary function show_spectrum.m is as follows.

```
function show_spectrum(x)
% show spectrum

plot(x(:,1)/1000,x(:,2))
ylabel('SPL(dB)')
xlim([0 5])
```

The auxiliary function show_stem.m is as follows.

```
function show_stem(x)
% show stem

stem(x(:,1)/1000,x(:,2),'fill')
xlim([0 5])
```

4.8 Model Testing

Let us see some of the testing result from our Tonality MATLAB model, which shows the plots of power spectrum, tonal components, relevant tonal components (SPL excess), spectral pitch pattern, and virtual pitch pattern.

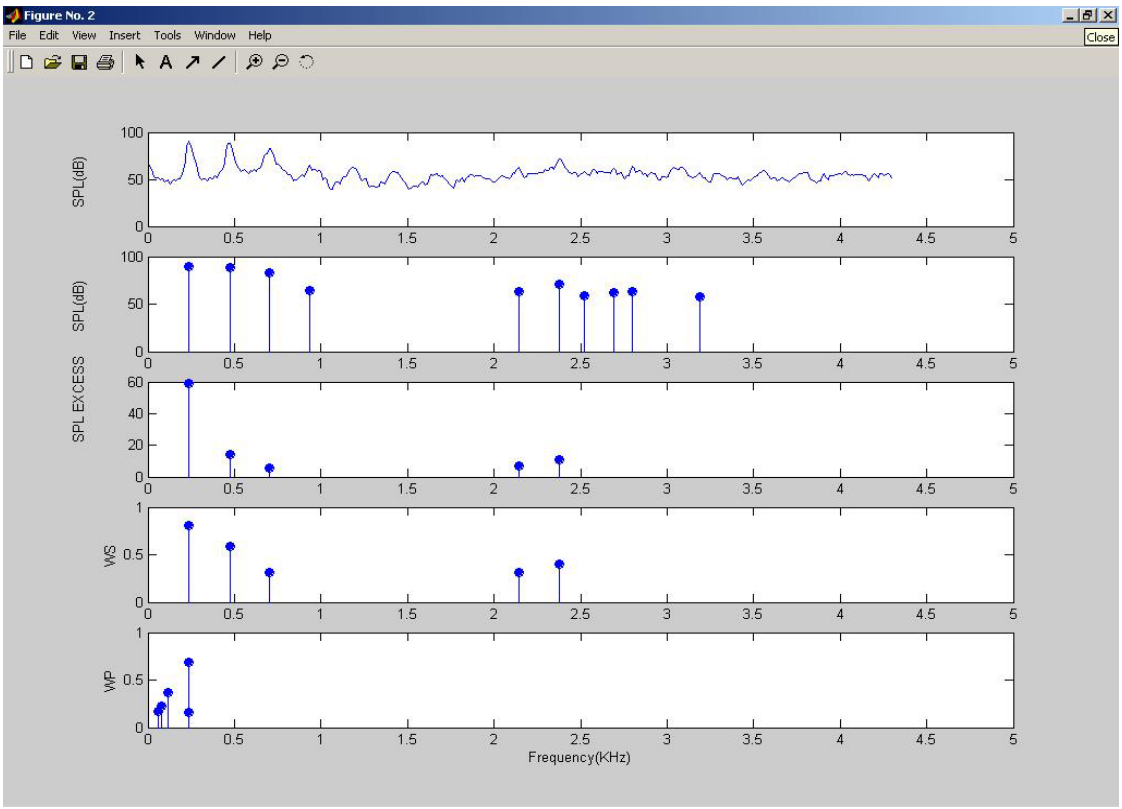

Figure 4.4 Pitch analysis of a natural speech vowel /a/

Figure 4.4 shows the pitch analysis of a natural speech vowel /a/. It contains 5 plots. The first one is the plot of power spectrum. The second one is the tonal components extracted from power spectrum. We can see that not all of the peaks in power spectrum can be extracted as tonal components. Only ten of them, which satisfy the requirements in the algorithm, can be extracted. The third one is the relevant tonal components whose SPL Excess is larger than zero. We can see that from ten tonal components only five of them the SPL excess is larger than zero. The fourth one is the spectral pitch pattern, represented by SP weight (WS). SP weights shows the extend to which each relevant tonal component contributes to the entire tonal percept. The fifth one is the virtual pitch pattern, represented by VP weight (WP). Both the fourth and fifth plots are final results from our MATLAB model. The abscissas of all the plots are for frequency. The ordinates of the first three plots are for SPL value, while the ordinates of the other two are for pitch weight value. Only the five most prominent virtual pitches are shown in order to simplify the plot. We can see that the most prominent of all pitches is the virtual pitch, which corresponds to the fundamental frequency.

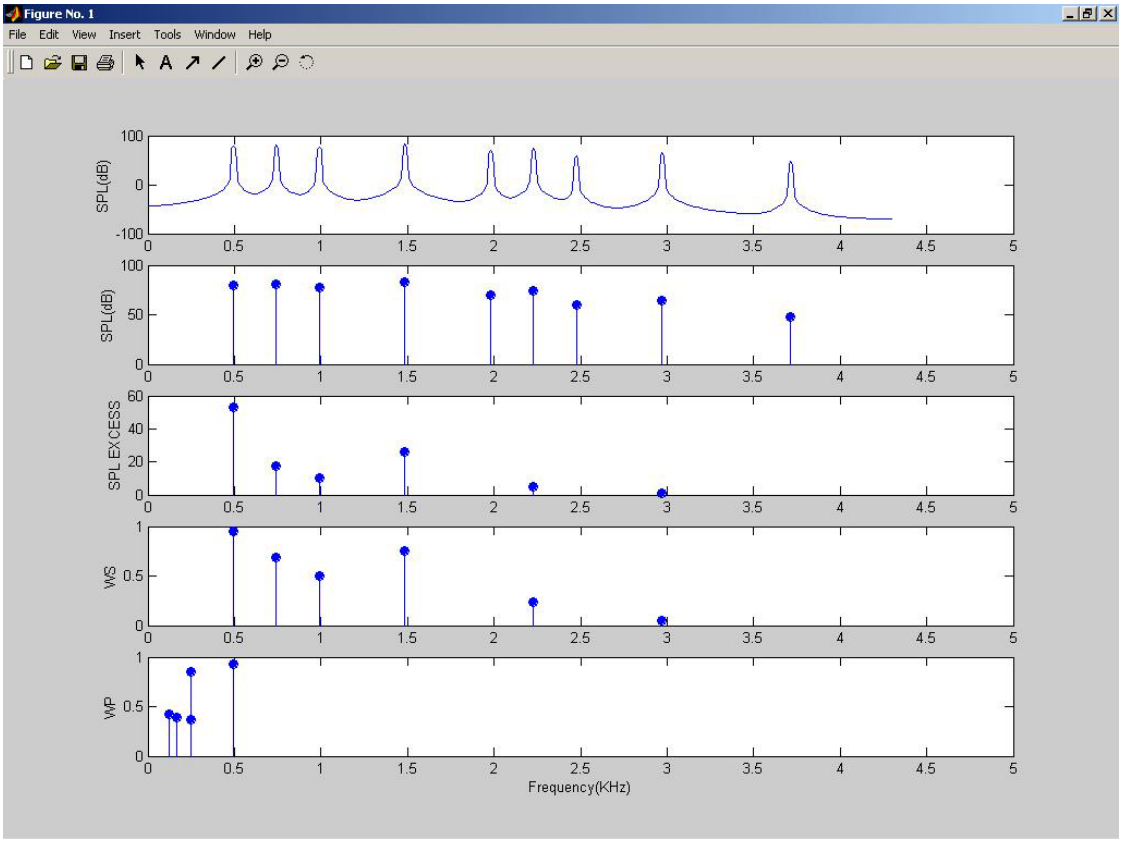

Figure 4.5 Pitch analysis of an artificial complex tone produced in Brüel & Kjær Sound Quality Type 7698

Figure 4.5 shows the pitch analysis of an artificial complex tone produced in B&K Sound Quality Type 7698. It contains 5 plots. The first one is the plot of power spectrum from which we can easily see the nine peaks. The second one is the tonal components extracted from power spectrum. We can see that all the peaks can be extracted as tonal components. The third one is the relevant tonal components whose SPL Excess is larger than zero. We can see that from nine tonal components only six of them the SPL excess is large than zero. So we only get six relevant tonal components. The fourth one is the spectral pitch pattern, represented by SP weight (WS). SP weights shows the extend to which each relevant tonal component contributes to the entire tonal percept. The fifth one is the virtual pitch pattern, represented by VP weight (WP). Both the fourth and fifth plots are final results from our MATLAB model. The abscissas of all the plots are for frequency. The ordinates of the first three plots are for SPL value, while the ordinates of the other two are for pitch weight value. Only the five most prominent virtual pitches are shown in order to simplify the plot. We can see that the most prominent of all pitches is the virtual pitch, which corresponds to the fundamental frequency.

Chapter 5 Implementation of Tonality Metric for SQ Application in VC++

After we made Tonality MATLAB model, we are in the stage to implement the same algorithm in VC++ for SQ application. We need to follow the program structure used in SQ application in order to make Tonality metric work, and that is to define COM components for Tonality metric. Section 5.1 contains the relevant principles about how COM component being used in SQ application, Section 5.2 contains the explanation of the relevant VC++ implementation.

5.1 How COM Component being used in our Project?

This part of our project work is based on Joseph Emmanuel Amuah's final thesis "Sound Quality User-defined Cursor Reading Controls", which gave us a very good basic understanding of B&K Sound Quality Program and a very good starting point for using ATL to make the specific control for Tonality in VC++.

"ATL is the Active Template Library, a set of template-based C++ classes with which you can easily create small, fast Component Object Model (COM) objects."

5.1.1 What is a Template Library?

A template is somewhat like a macro. As with a macro, invoking a template causes it to expand (with appropriate parameter substitution) to code you have written. However, a template goes further than this to allow the creation of new classes based on types that you pass as parameters. These new classes implement type-safe ways of performing the operation expressed in your template code.

Template libraries such as ATL differ from traditional C++ class libraries in that they are typically supplied only as source code (or as source code with a little, supporting run time) and are not inherently or necessarily hierarchical in nature. Rather than deriving from a class to get the functionality you desire, you instantiate a class from a template. [5.1]

5.1.2 More about COM Programming

First of all, a clear understand of COM is very important.

"COM is a specification and a set of services that allow you to create modular, object-oriented, customisable and upgradeable, distributed applications using a number of programming languages."

COM is not about any particular type of application. It's not about controls (that's ActiveX); it's not about compound documents (that's OLE); it's not about data access (that's OLE DB and ADO); and it's not about games and graphics (that's DirectX). But COM is the object model that underlies all these technologies.

COM programming splits roughly into two parts: **server** and **client.** The former is generally regarded as being the more difficult, and involves the actual creation of COM components. Assisting with this process is the purpose for which ATL is primarily intended. Client programming, on the other hand, is comparatively simple: it involves writing code that makes use of the components that you or a third party have created. [5.2]

Furthermore, COM components can be implemented in two types of code modules: DLLs and EXEs. In addition, COM components can run on machines remote to the client. There are many technologies you need to know in order to understand COM precisely, but here we focus on the understanding of the relevant technology we need to use in our project.

5.1.3 What is Dynamic Link Libraries (DLL)?

Dynamic link libraries (DLLs) are a very powerful feature of the Microsoft® Windows® operating system. In the MS-DOS environment, all a program's object modules were statically linked during the build process. Windows allows dynamic linking, which means that specially constructed libraries can be loaded and linked at runtime. Multiple applications can share dynamic link libraries (DLLs), which saves memory and disk space. Dynamic linking increases program modularity because you can compile and test DLLs separately.

5.1.4 Using COM in SQ Application

The current SQ application requires that any Cursor Reading Control must be implemented as a COM component, and this COM component must be implemented in a DLL (Dynamic link library). We can see from Figure 5.1 that, this DLL, which contains the specific control, is an in-process server of SQ application.

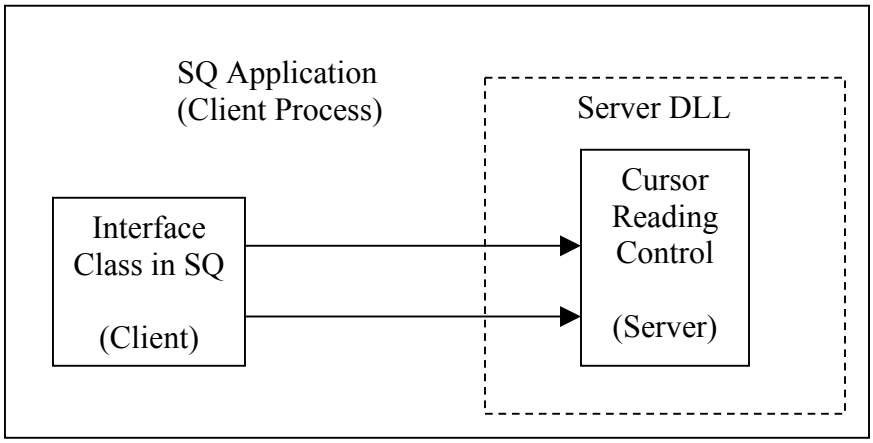

Figure 5.1 Communications Between SQ Application and Cursor Reading Control

Now let us go into more detail of how the Cursor Control Class exposes its functionality to the host SQ application and how the SQ Interface Class communicates with the control. The Interface Class is the only way that the SQ application can communicate with the Cursor Control Class. You can see from Figure

5.2 that it is similar to Fig. 6.0 in Joseph's report except that the functions on the interface are shown in the order in which they are called.

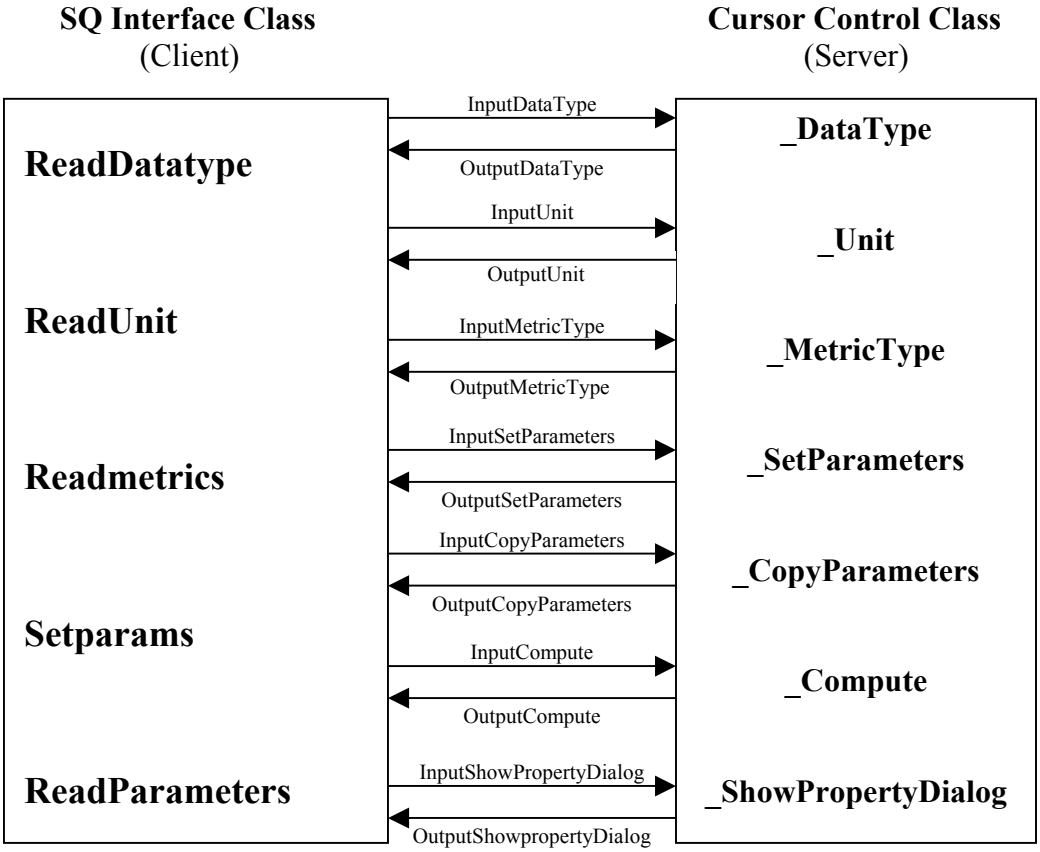

Figure 5.2 Communications between SQ Interface Class and Cursor Control Class

The arrows in Figure 5.2 are two directions, which indicate that communication goes in both directions, but in fact that is the conceptual understanding. While in the real VC++ programming, we achieve the communications by means of pointers. Let's see how one of the cursor control methods, **_Compute**, tries to fulfil the communications with SQ application.

STDMETHODIMP CSQC_TonalityCur::_Compute(double *Lin, double *Rin, double *mLin, double *mRin, double *Lout, double *Rout)

When we call this function, we need to transfer six pointers to real variables. The first four are for the input, and the last two are for the output. It is because the pointer we used, which insure the last two real variables will be set to new values after calling this method.

Generally, all the user-defined cursor reading controls in SQ Application are implemented as COM components, which ensure that they can expose their functionality to the host SQ application by means of interfaces. In COM, the only way to manipulate the data associated with an object is through interfaces on the object.

"An interface is the way in which an object can expose its functionality to the outside world. In COM, an interface is a table of pointers (like a C++ vtable) to functions

implemented by the object. The table represents the interface, and the functions to which it points are the methods of that interface. An object can expose as many interfaces as it chooses. Each interface is based on the fundamental COM interface, IUnknown." [5.1]

COM uses the word interface in a different sense from that typically used in C++ programming. A C++ interface, such as we used in "SQ Interface Class", refers to all of the functions that a class supports and that clients of an object can call to interact with it. A COM interface refers to a predefined group of related functions that a COM class implements, but a specific interface does not necessarily represent all the functions that the class supports. When the term "interface" is used in the following part of this report, it refers to an implementation in code of a COM binary-compliant interface that is associated with an object.

There are three kinds of interfaces, early binding interface, late binding interface and dual interface. Binding describes how clients access properties and methods of a server. If your client can detect at compile time what object a property or method belongs to, it can resolve the reference to the object at compile time. The compiled executable contains only the code to invoke the object's properties, methods, and events. This is called early binding. Another situation is if your client was written before your COM object, it can't get any information of the server at compile time. So it must query the COM object at run time to find the properties, methods, and events supported by COM object. This is called late binding. Dual interface means that functions and parameters are supported both through custom definition and through COM interface IDispatch. Dual interfaces permit COM objects to support both early binding and late binding. Dual interface combines speed and flexibility. For this reason, dual interfaces are recommended whenever possible. Dual interface is being used when writing COM components for the SQ application.

For our tonality control, we only need to define one dual interface based on the COM interface IDispatch, which is inherent from the fundamental COM Interface, IUnknown.

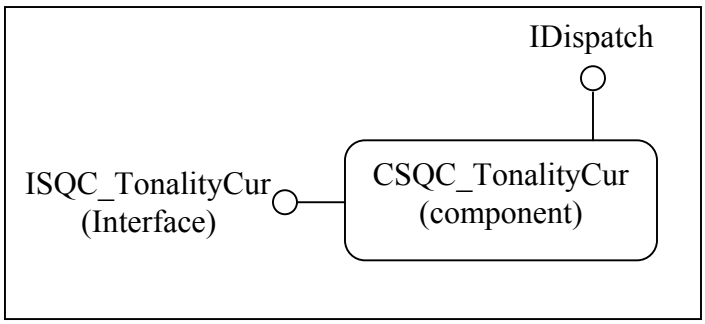

Figure 5.3 COM Diagrams for Tonality Cursor Control Class

Actually no specific code needs to be written for interface definition because all the definition can be easily done by means of ATL.

5.1.5 More about COM Interface

In the following part of this section, we will first introduce the notions, Interface Definition and Interface Implementation. Second we will discuss the difference between these two concepts. Last we will introduce The Interface Definition Language (IDL), which is used to write interface definition, and furthermore we present the concept of type library, which is best thought of as a binary version of an Interface Definition Language (IDL) file.

- **Interface Definition and Interface Implementation**

COM makes a fundamental distinction between interface definitions and their implementations. An *interface* is actually a contract that consists of a group of related function *prototypes* whose usage is defined but whose implementation is not. These function prototypes are equivalent to pure virtual base classes in C++ programming. An interface definition specifies the interface's member functions, called *methods,* their return types, the number and types of their parameters, and what they must do. There is no implementation associated with an interface. [5.1]

An *interface implementation* is the code a programmer supplies to carry out the actions specified in an interface definition. Implementations of many of the interfaces a programmer can use in an object-based application are included in the COM libraries. However, programmers are free to ignore these implementations and write their own. An interface implementation is to be associated with an object when an instance of that object is created, and the implementation provides the services that the object offers. [5.1]

The interface definition would not specify how the functions are to be implemented in code. In implementing the interface, different programmers might implement it in different ways such as using different data structures and different data flows and control flows. Regardless of a particular implementation of the interface methods, the in-memory representation of a pointer to an interface, and therefore its use by a client, is completely determined by the interface definition.

- **How to Understand Interface**

Simple objects support only a single interface. More complicated objects, such as embeddable objects, typically support several interfaces. Clients have access to a COM object only through a pointer to one of its interfaces, which, in turn, allows the client to call any of the methods that make up that interface. These methods determine how a client can use the object's data. [5.1]

Interfaces define a contract between an object and its clients. The contract specifies the methods that must be associated with each interface and what the behaviour of each of the methods must be in terms of input and output. The contract generally does not define *how* to implement the methods in an interface. Another important aspect of the contract is that if an object supports an interface, it must support all of that interface's methods in some way. Not all of the methods in an implementation need to do something—if an object does not support the function implied by a method, its

implementation may be a simple return or perhaps the return of a meaningful error message—but the methods must exist. [5.1]

- **The Interface Definition Language**

Although ATL makes COM programming more simple in a way that you do not need to define the interfaces by your own code. Instead, you just follow some instructions and set-up some dialogs, then ATL will help you generate the code for interface definition. But understanding of how an unambiguous definition of the interfaces is implemented will be very helpful for you to understand the technology behind ATL COM programming. Interface definition is achieved by means of The Interface Definition Language (IDL).

The Interface Definition Language (IDL) is used to provide complete definitions of COM interfaces, and it allows us to associate the methods of an interface with its IID (Interface Identifier), to specify details of the interface in a form that can be machine-processed to produce marshalling code, and to instruct clients and servers on how they should allocate and deallocate memory. [5.2]

- **Type Library**

COM is a binary standard for component implementation and integration, and therefore the language used to build or call COM objects is irrelevant. Since our component is already built on COM, you might wonder why anything special is required to make it accessible to developers using other languages. Well, to make a component more accessible to developers working in other languages, you need to create a type library. [5.1]

A type library is best thought of as a binary (or compiled) version of an Interface Definition Language (IDL) file. It contains a binary description of the interfaces exposed by a component, defining the methods along with their parameters and return types. Many environments support type libraries: Visual Basic, Visual J++, and Microsoft Visual C++® all understand type libraries; so do Delphi, Microsoft Visual FoxPro®, and Microsoft Transaction Server. Rumor says that the next version of Microsoft (Visual) Macro Assembler will support COM via type libraries. [5.1]

5.2 Tonality Implementation in VC++

Before starting to write VC++ code, there are several things that we need to consider. One is what kinds of Data Structures should be used to store the data that we manipulate in each steps of the algorithm; the second is the data flow of our program based on the data structure we use. The first problem is solved by using the Vector Container Type, which is part of C++ Standard Template Library (STL), and the second problem is solved by using the analysis of our previous work, please refer to Figure 4.1 DFD for Tonality Model. In the following part of this section, we will highlight some special points, which proved to be very helpful for making the Tonality Control in VC++.

5.2.1 Why Choose Vector Container Type?

For the purposes of making Tonality Control, We firstly think about using Array. In C++, An array is a collection of objects of a single data type. Each element is accessed by its position in the array, which is called indexing or subscripting. If the number of elements to be stored is known at compile time, an array can be used. Apparently this is not suitable for our purpose, because in each procedure the result of the calculation is generated dynamically, the number of elements to be stored is unknown and likely to vary widely. So using dynamically allocated storage for each object turns out to be a better way, and sequence container types are considered. A sequence container holds an ordered collection of elements of a single type. The sequence container types in VC++ includes, the vector, the deque, and the list, which all grow dynamically.

The vector class provides an alternative representation to the built-in array, and its usage is highly recommended by experienced programmers. The vector class is part of the standard library introduced with Standard C++.

There are two very different forms of using a vector, the array idiom and the STL idiom. The array idiom is analogous to defining a built-in array. In the STL idiom, a vector is used very differently. Rather than define it with a given size, we define an empty vector. Rather than using index and assign to an element, we instead insert an element into the vector. There are some STL functions in Visual C++ that we can use to operate the vector. The following list contains some of them, which are frequently used in our code.

- vector::begin - Returns an iterator to start traversal of the vector.
- vector::end - Returns an iterator for the last element of the vector.
- vector::push_back - Appends (inserts) an element to the end of a vector, allocating memory for it if necessary.
- vector::iterator - Traverses the vector. An iterator is a standard library class that represents the functionality of a pointer. One can dereference the iterator and access the actual object addressed.
- vector::size - Returns number of elements in the vector.
- vector::erase - Deletes elements from a vector (single & range).

The second form of vector, the STL idiom, is chosen due to the dynamic nature of the problem. Let's see an example about how to define power spectrum as a vector type in C++ code. The considerations are, power spectrum of a signal can be presented as a list of frequency and SPL pairs. In C++, the pair class is part of the standard library, and allows us to associate two values of either the same or different types within a single object. So this is defined in, POWER_SPECTRUM_ITEM, which contains the pair of two double values. In fact these two double values refer to frequency and SPL respectively. Then we define a vector type, POWER_SPECTRUM, which contains POWER_SPECTRUM_ITEM. The code is as follows,

```
#include <vector>
using namespace std ;
typedef struct pair<double, double> POWER_SPECTRUM_ITEM ;
typedef vector<POWER_SPECTRUM_ITEM> POWER_SPECTRUM;
```

We have also defined some other vector types, such as SPECTRUM_PITCH, SPECTRUM_PITCH_SHIFT, SPECTRUM_PITCH_WEIGHT, SP_PATTERN, VP_PATTERN. For more information about the other vector type definitions, please refer to the header file, **SQC_TonalityCur.h** in Appendix B.

Especially, we want to mention the definition of the following vector for the purpose of remembering the indices as shown in Figure 4.1.

```
typedef vector<POWER_SPECTRUM::iterator> INDICES;
```

We define the INDICES as a vector type contains the iterator of POWER_SPECTRUM. So actually INDICES is a vector type contains pointers, which can be used to dereference and access the actual object addressed.

5.2.2 Using Pointers

Dynamic memory allocation and pointer manipulation is a fundamental aspect of real-world programming in C++. It is very efficient to transfer the pointer to a vector instead of the whole vector when you call some sub functions from the main function because it provides a way that can avoid transferring the whole variable, but only a pointer to that variable instead. It becomes more efficient when the variable needs big storage space.

Let's look at an example of the **Get_Tonal_Comp** function. This procedure is used after we call **Get_Power_Spectrum**. We need to transfer three pointers as parameters, which are referred to three real variables. After calling this function, the real variables' value is changed because the pointers we use in this function definition. The first variable to this function call is the input, while the other two variables are the output because we assigned new values to them.

```
Void CSQC_TonalityCur::Calc::Get_Tonal_Comp
                        (POWER_SPECTRUM *pPS, POWER_SPECTRUM *pTC, INDICES *p_idcs1)
{
        POWER_SPECTRUM::iterator i;
        POWER_SPECTRUM_ITEM tc_item;
        for( i = pPS->begin()+3; i != pPS->end()-3 ; i++)
        {
            if ((*i).second >= (*(i+1)).second && (*i).second > (*(i-1)).second && (*i).second –
               (*(i-3)).second >=3 && (*i).second - (*(i-2)).second >= 3 && (*i).second –
               (*(i+2)).second >=3 && (*i).second - (*(i+3)).second >= 3)
                  {
                     tc_item.first = (*i).first+0.46 * ((*(i+1)).second - (*(i-1)).second);
                     tc_item.second =(*i).second;
                     pTC->push_back(tc_item);
                     p_idcs1->push_back(i);
                  }
        }
}
```

More explanation should be given for the last two lines in for loop. The first line *pTC->push_back(tc_item)* means to save the result into the vector by means of push_back method which is provided by the vector container type in Standard Type

Library(STL) in Visual C++. The second line *p_idcs1->push_back(i)* means to save the iterator of the current items which you just saved. An iterator can also be understood as a pointer to the current element or item in the vector in our usage. While precisely speaking, an iterator provides a general method of successively accessing each element within any of the sequential or associative container types. For more information about iterator of vector container types, STL or pointers, please refer to reference [5.3].

On the other hand, using pointers might cause some problems in some situations. For example, if you have a variable, which needs a large space to store the relevant information, and more than two pointers refer to it in your application. Let us assume a situation as follows: you have such two procedures in your main program, which use these two pointers respectively. Then what happened when both of the procedures trying to access the same variable? Yes, it will cause some security problems--make the data in the variable unsafe. So how can we solve it and avoid this kind of security problems?

Actually the ultimate problem of this question is also one of the most critical issues of the design of libraries and long running programs, that is memory management. A library writer doesn't in general know if a library will be part of an application where memory is scarce, where a memory leak would be a serious problem, or where memory-management overhead could be a serious liability.

The fundamental question about memory management can be stated in this way: If f() passes or returns a pointer to an object to g(), who is responsible for the object's destruction? A secondary question must also be answered: When can it be safely destroyed? In particular, these questions are critical to designers and users of container classes, such as lists, arrays, and associative arrays. From the point of view of a library provider, the ideal answers are "the system" and "whenever nobody is using the object any longer." A system doing this is usually said to be garbage collecting, and the part of the system that determines that nobody uses a given object and destroys it is called the garbage collector. [5.4]

Unfortunately, garbage collection implies overheads in runtime and space, service interruptions, special supportive hardware, trouble with linking to program fragments written in other languages, or system complexity. This, many users cannot afford. Consequently, although garbage collecting implementations of C++ exist, most C++ programs cannot rely on garbage collection and must devise strategies for managing objects on the free store without help from the system. [5.4]

Garbage Collection can be seen as a way of simulating an infinite memory in a limited memory. For more information about Garbage Collection, please refer to [5.3] and [5.4].

This kind of security problem can be solved in C++ programming by using Exception Handling in both procedures to protect the same variable, in each procedure, use dynamic memory allocation and deallocation for the pointer. Exception handling is a mechanism that allows two separately developed program components to communicate when a program anomaly, called an *exception*, is encountered during the execution of the program. It is similar idea as critical region in Concurrent

Programming. Exception handling provides a way of transferring control and information from a point in the execution of a program to an *exception handler* associated with a point previously passed by the execution. A handler will be invoked only by a *throw-expression* invoked in code executed in the handler's *try-block* or in functions called from the handler's *try-block*. For more information about *exception handler, throw-expression* and *try-block,* please refer to [5.4]. Dynamic memory allocation and deallocation in C++ means that objects are allocated by a run-time library, which invoked during program execution. Contrariwise, for static memory allocation, the objects are allocated by the compiler as it processes the program source code. Whenever using the dynamic memory allocation, one needs to remember to delete the dynamically allocated memory. Otherwise it ends up with a *memory leak.* A memory leak is a chunk of dynamically allocated memory that we no longer have a pointer to, and thus we cannot return it to the operating system for later reuse.

Actually, we simply solve this security problem in our C++ program by making sure that no pointers access the same variable at the same time. We think it is the easiest and the most convenient way to solve the problem and we do not need to bother too much about the usage of Exception Handling, which is a little bit complicated for us to try when we have strict project time limitation.

5.2.3 Using The Generic Algorithms

In the VP Pattern calculation procedure in Tonality algorithm, we need to sort the SP Pattern in terms of descending WS (Spectrum Weight). Many sorting algorithms exist, such as Bubble Sort, Heap Sort, Insertion Sort, Merge Sort, Quick Sort and so on. But the most efficient way to solve this problem is not to write our own algorithm, but to use the Generic Algorithms in VC++. Generic Algorithm is also part of Standard Template Library (STL).

To find the minimum or maximum value among the elements of a vector, we must instead invoke one of the generic algorithms: "algorithms" because they implement common operations such as min(), max(), find(), and sort(); "generic" because they operate across multiple container types—not only the vector and list types, for example, but also the built-in array type. The container is bound to the generic algorithm operating on it by an iterator pair marking the elements to traverse. [5.3]

Each generic algorithm is implemented independently of the individual container types. Because the type dependence of the algorithm has been removed, a single template instance can work across all the container types as well as the built-in array type. [5.3]

For more information about the mechanism of Generic Algorithms, Please refer to reference [5.3] and [5.1]. Let's look at an example from my VC++ code to show how we use it when we need to solve our specific problem.

The structure and type definition defined in the header file.

```
struct VP_PATTERN_ITEM
{
        double frequency;
        double vp_pitch;
```

```
        double vp_weight;
        bool operator<(const VP_PATTERN_ITEM &vp_pattern_item) const
        {
                return vp_weight<vp_pattern_item.vp_weight;
        }
};

typedef vector <VP_PATTERN_ITEM> VP_PATTERN;
```

Now Sorting the vector becomes so easy by means of the Generic Algorithms.

```
//Sort in terms of descending WS
void  CSQC_TonalityCur::Calc::Sort_VP_Pattern(VP_PATTERN *pVP_Pattern)
{
        //sorts increasingly
        sort(pVP_Pattern->begin(), pVP_Pattern->end());

        //change the increasing order into descending order
        reverse(pVP_Pattern->begin(), pVP_Pattern->end());
}
```

5.2.4 Object-based Programming

Object-based programming means the definition and use of C++ class facility to define new types that can be manipulated as easily as built-in types. By creating new types to describe the problem domain, C++ allows the programmer to write applications that are easier to understand. Facilities for encapsulating the data and functions supporting the implementation of the new type can simplify subsequent maintenance and evolution of our application dramatically. [5.3]

For a specific implementation, the decision on how to use class depends on the specific algorithm. For our program, the data flow of different functions is very clear, and the input and output of each function do not require very complicated data structure, and we can not really extract the base class and derived class so that we do not need to use inherence. But from the upper level of the algorithm, it will be nice if we encapsulate the Tonality metric calculation in one class, and make some privileges of different data members and member functions. Another advantage of using class in our implementation is that it can be instantiated according the user's usage of channel(s). We will present the idea and the usage of class in the following part.

The considerations we have taken into account for our project is we need to figure out a way that should be the best and not that complicated, and specifically to solve our problem within the project period. So it put us in some situations that we must choose something that we think it is better for our project. Such as we choose the Vector Container Type to use instead of using the other sequence container types, such as the deque or the list, it is mainly because it is highly suggested by some experienced programmer and we also feel it is exactly fit our specific problem. Moreover, why we want to mention about Object-Based Programming in this report is also because some of the considerations during the project period. In the first version of our VC++ program, we did not use class at all, which does not mean no class in the whole ATL COM programming, but means that no class considerations in our own codes. The drawback of our first version of VC++ program is as follows, First one should know

that SQ application provide single channel or dual channel measurement, through the one of the Interface Methods **_Compute**, one could tell which is the case. Our problem is we must consider the maximum channels that a user could use is two, so that we declare twice for all the variables for both left and right channels, which might cause some waster of memory allocation, if you only need one channel. Because the variable declarations are done at compile time (not for the vector type, because vector encapsulate the dynamic memory allocation characteristics inside the Vector class), but any way, it is not convenient and waster of space, so that leads us to think about using some concepts in Object-based programming. Encapsulating all the calculation procedures in one class, let's say it is called **Calc**, is a good idea, because we can make it in a way that it declare all the relevant variables as protected data members, declare the upper level functions as protected member functions, and declare the other lower level functions as private member functions, furthermore, the only public member function, which can be accessed by the interface method, is **Get_Tonality** function. It uses all the necessary functions to get Tonality calculation result. For more information about the class **Calc** definition, please refer to **SQC_TonalityCur.h** for the class declaration, and **SQC_TonalityCur.cpp** for the class definition in the source code in Appendix B.

Now Let us move into another topic on how we use class **Calc** in interface method, **_Compute**. There are two channels available in a SQ project. The user can use either single channel or dual channels for a certain measurement. By encapsulating all Tonality calculations in a class called **Calc**, and in the interface method **_Compute** of Tonality Cursor Reading Control, class **CSQC_TonalityCur**, the program automatically detect which channel(s) the user is currently using, then initialise the class **Calc**, and call its member function **Get_Tonality** to complete the calculations. **Get_Tonality** is the only public member function in **Calc** class, the reason is that we want to encapsulate all the data members and other member functions in the class, and the only way to access the class is through member function **Get_Tonality.** The following code is the implementation for the interface method **_Compute.**

```
STDMETHODIMP CSQC_TonalityCur::_Compute(double *Lin, double *Rin, double *mLin, double
*mRin, double *Lout, double *Rout)
{
	AFX_MANAGE_STATE(AfxGetStaticModuleState())

	// TODO: Add your implementation code here
	if (Lin != NULL)
	{
		Calc* Left_Calc = new Calc;
		Left_Calc->Get_Tonality(Lin);
		delete Left_Calc;
	}
	if (Rin != NULL)
	{
		Calc* Right_Calc = new Calc;
		Right_Calc->Get_Tonality(Rin);
		delete Right_Calc;
	}
	return S_OK;
}
```

For more information about class **CSQC_TonalityCur**, Please refer to the header file **SQC_TonalityCur.h** and the definition of the class in file **SQC_TonalityCur.cpp** in Appendix B.

Chapter 6 Testing of MATLAB and VC++ Program

MATLAB and VC++ code for Tonality are based on the same Data Flow (Figure 4.1), so they have the same main functions and the similar computational results from each function. The way we test them is to validate the difference of computational results from each main function is small enough.

Power Spectrum data, the result of FFT analysis, is the input data for both MATLAB Model and VC++ implementation. So that it will be perfect if we can input the same power spectrum data into both of them. But these two input data are a little bit different because the way we get them is not the same.

For MATLAB, we get the Power Spectrum data from B&K SQ Application, export into EXCEL, and then transfer them to MATLAB. The procedure is limited by the applications we use so that it is not possible for us to change the precision of the input data.

For VC++, we calculate Power Spectrum in the following function,

```
void CSQC_TonalityCur::Calc::Get_Power_Spectrum(double *Din, POWER_SPECTRUM *pPS)
{
	POWER_SPECTRUM_ITEM ps_item;
	//ps_item: power spectrum item, it is a pair of frequency and dB value.

	const No_Of_SAMPLES = 401;
	for (int i = 0; i < No_Of_SAMPLES; i++)
	{
		POWER_SPECTRUM_ITEM* ps_item = new POWER_SPECTRUM_ITEM;

		//calculate frequency
		ps_item.first=(i*11025.)/(400.*2.56);

		//get dB value
		ps_item.second=20. * log10(Din[i]) + 94.   ;

		//save the pair of frequency and dB value to vector.
		pPS->push_back(ps_item);
	}
}
```

Din is the Input Data from left or right channel of B&K SQ Application, which is represented as double type array. Later, we calculate the frequency and dB values for each sample.

The following table shows an example of different precisions between the Power Spectrum data to MATLAB and VC++. They come from the same signal from B&K SQ application.

MATLAB		VC++	
Frequency (Hz)	SPL (dB)	Frequency (Hz)	SPL (dB)
0.0000	56.2787	0.00000000000	56.299313566358919
10.7666	54.3696	10.7666015625	54.390231318086556
21.5332	47.1922	21.5332031250	47.212799468536033
32.2998	42.9453	32.2998046875	42.965851238165897
43.0664	38.7051	43.0664062500	38.725731869053199
53.8330	34.7855	53.8330078125	34.806115168988541
64.5996	30.6750	64.5996093750	30.695610691174878
75.3662	30.9614	75.3662109375	30.981955980834996
86.1328	34.4380	86.1328125000	34.458624572923227
96.8994	47.6942	96.8994140625	47.714788494471001
…	…	…	…

Table 6.1 Different precisions for the Power Spectrum data in MATLAB and VC++

Even through both MATLAB and VC++ perform all computations in double precision after they get the input data, but we must accept the fact that the original input data into these two implementations have a little bit difference, and the difference is restricted by the applications and programming languages we use.

On the other hand, we think this little difference can be neglected and the two programs can be validated in a way of showing the consistency between them.

The following two tests performed on our MATLAB model and VC++ implementation. Power Spectrum of Test1 and Test2 can be found in Appendix C. Because the two programs implement the same algorithm and follow the same data flow, so that comparable results can be shown according to the Data Stores in DFD of Figure 4.1.

Due to the different programming approach, D2 and D4 have no comparability. So we did not consider them.

The calculations of value difference do not consider the frequency values because they are almost the same. We calculate the value difference of Tonal Components in the following way,

(SPL2-SPL1)/SPL2

SPL1: The SPL value of Tonal Components in MATLAB.
SPL2: The SPL value of Tonal Components in VC++.

The similar calculations of value difference are carried out for all the other comparable computational results.

- **Test 1**

The difference of Power Spectrum data is maximal 0.19%, we think this can be neglected. The Difference of Tonal Components data is between 0.03% and 0.08%. We think they are small enough to be ignored. The difference of SPL Excesses has two bigger values, that is 5.30% in frequency 989.05 Hz and 2.91% in frequency

1528.19 Hz. We think this difference is acceptable because the complexity of the calculation procedure for SPL Excess. The difference of Spectral Pitches has only one value not equal to 0, which is 0.32% in frequency 387.89 Hz. We think it is small enough to be omitted. The Spectral Pitch Shifts have no difference. The difference of Spectral Pitch Weights has two bigger values at the same frequencies as SPL Excesses, which is 4.26% in frequency 989.05 Hz, and 2.27% in frequency 1528.19 Hz. This is because the calculation of Spectral Pitch Weights depend on the SPL Excesses, so the difference is some how inherited from the SPL Excesses. We think this is also acceptable. SP Pattern actually just combines the Spectral Pitch Shifts and Spectral Pitch Weights values, and the difference for SP Pattern has the same values as what appears in the difference of Spectral Pitch Shifts and Spectral Pitch Weights. The difference of VP Pattern contains two columns, one is for the virtual pitch value difference, which is maximal 0.34% and the other is for virtual pitch weight value difference, which is 1.33%, and we think these differences can be neglected. So Test1 can be used to conform the agreement between MATLAB and VC++ program.

	MATLAB	VC++	Difference
Power Spectrum	Appendix C	Appendix C	Max. = 0.19%
Tonal Components	387.89 58.13	387.89 58.15	0.03%
	765.95 33.20	765.95 33.22	0.06%
	989.05 26.51	989.05 26.53	0.08%
	1528.19 27.16	1528.19 27.18	0.07%
	1871.83 39.69	1871.82 39.71	0.05%
	2078.14 46.70	2078.14 46.72	0.04%
	2509.13 41.67	2509.12 41.69	0.05%
	2636.70 43.89	2636.69 43.91	0.05%
	2863.81 37.18	2863.81 37.20	0.05%
	3091.05 35.01	3091.05 35.03	0.06%
	3348.17 40.62	3348.18 40.65	0.07%
	3467.18 39.10	3467.18 39.12	0.05%
	3595.04 35.01	3595.05 35.03	0.06%
	3864.10 26.76	3864.10 26.78	0.07%
SPL Excesses	387.89 39.02	387.89 39.29	0.69%
	765.95 12.86	765.95 12.95	0.69%
	989.05 9.11	989.05 9.62	5.30%
	1528.19 9.67	1528.19 9.96	2.91%
	1871.83 10.35	1871.82 10.37	0.19%
	2078.14 16.20	2078.14 16.23	0.18%
	2509.13 1.52	2509.12 1.52	0.00%
	2636.70 3.04	2636.69 3.04	0.00%
Spectral Pitches	387.89 388.10	387.89 389.35	0.32%
	765.95 788.11	765.95 788.12	0.00%
	989.05 1014.15	989.05 1014.16	0.00%
	1528.19 1549.35	1528.19 1549.34	0.00%
	1871.83 1914.71	1871.82 1914.71	0.00%
	2078.14 2119.70	2078.14 2119.70	0.00%
	2509.13 2624.00	2509.12 2624.01	0.00%
	2636.70 2737.91	2636.69 2737.92	0.00%
Spectral Pitch Shifts	387.89 0.00	387.89 0.00	0.00%
	765.95 0.03	765.95 0.03	0.00%
	989.05 0.03	989.05 0.03	0.00%
	1528.19 0.01	1528.19 0.01	0.00%
	1871.83 0.02	1871.82 0.02	0.00%
	2078.14 0.02	2078.14 0.02	0.00%
	2509.13 0.05	2509.12 0.05	0.00%
	2636.70 0.04	2636.69 0.04	0.00%
Spectral Pitch Weights	387.89 0.88	387.89 0.88	0.00%

	765.95	0.58		765.95	0.58	0.00%	
	989.05	0.45		989.05	0.47	4.26%	
	1528.19	0.43		1528.19	0.44	2.27%	
	1871.83	0.43		1871.82	0.43	0.00%	
	2078.14	0.54		2078.14	0.54	0.00%	
	2509.13	0.07		2509.12	0.07	0.00%	
	2636.70	0.13		2636.69	0.13	0.00%	
SP Pattern	387.89	0.00	0.88	387.89	0.00	0.88	0.00% 0.00%
	765.95	0.03	0.58	765.95	0.03	0.58	0.00% 0.00%
	989.05	0.03	0.45	989.05	0.03	0.47	0.00% 4.26%
	1528.19	0.01	0.43	1528.19	0.01	0.44	0.00% 2.27%
	1871.83	0.02	0.43	1871.82	0.02	0.43	0.00% 0.00%
	2078.14	0.02	0.54	2078.14	0.02	0.54	0.00% 0.00%
	2509.13	0.05	0.07	2509.12	0.05	0.07	0.00% 0.00%
	2636.70	0.04	0.13	2636.69	0.04	0.13	0.00% 0.00%
VP Pattern	77.58	74.06	0.19	77.58	74.31	0.19	0.34% 0.00%
	96.97	93.49	0.30	96.97	93.80	0.30	0.33% 0.00%
	129.30	125.78	0.49	129.30	126.20	0.49	0.33% 0.00%
	193.951	190.43	0.74	193.95	191.05	0.75	0.32% 1.33%
	387.89	388.10	0.91	387.89	389.35	0.92	0.32% 1.09%

Table 6.2 Computational Results from Test1

- **Test 2**

The maximal difference of Tonal components is 0.02%.
The maximal difference of SPL Excesses is 0.79%.
The maximal difference Spectral Pitches is 0.07%.
The difference of Spectral Pitch Shifts is always 0.00%.
The difference of Spectral Pitch Weights is always 0.00%.
SP Pattern is the combination of Spectral Pitch Shifts and Spectral Pitch Weights.
The difference of VP Pattern contains two columns, one is for the virtual pitch value difference, which has the maximal value 0.07%, and the other is for virtual pitch weight value difference, which is always 0.

We think all the above differences can be neglected. So Test1 can also be used to conform the agreement between MATLAB and VC++ program.

	MATLAB	VC++	Difference
Power Spectrum	Appendix C	Appendix C	Max. = 0.07%
Tonal Components	387.59 87.87	387.59 87.89	0.02%
	807.49 90.96	807.49 90.98	0.02%
	1410.41 90.97	1410.41 90.99	0.02%
	2196.38 87.87	2196.38 87.89	0.02%
SPL Excesses	387.59 67.83	387.59 68.37	0.79%
	807.49 27.40	807.49 27.39	-0.04%
	1410.41 21.61	1410.41 21.59	-0.09%
	2196.38 13.66	2196.38 13.65	-0.07%
Spectral Pitches	387.59 384.03	387.59 384.31	0.07%
	807.49 811.45	807.49 811.45	0.00%
	1410.41 1424.82	1410.41 1424.84	0.00%
	2196.38 2235.61	2196.38 2236.57	0.04%
Spectral Pitch Shifts	387.59 -0.01	387.59 -0.01	0.00%
	807.49 0.00	807.49 0.00	0.00%
	1410.41 0.01	1410.41 0.01	0.00%
	2196.38 0.02	2196.38 0.02	0.00%

Spectral Pitch Weights	387.59	0.94		387.59	0.94	0.00%	
	807.49	0.84		807.49	0.84	0.00%	
	1410.41	0.71		1410.41	0.71	0.00%	
	2196.38	0.48		2196.38	0.48	0.00%	
SP Pattern	387.59	-0.01	0.94	387.59	-0.01	0.94	0.00%	0.00%
	807.49	0.00	0.84	807.49	0.00	0.84	0.00%	0.00%
	1410.41	0.01	0.71	1410.41	0.01	0.71	0.00%	0.00%
	2196.38	0.02	0.48	2196.38	0.02	0.48	0.00%	0.00%
VP Pattern	129.20	124.43	0.33	129.20	124.52	0.33	0.07%	0.00%
	193.80	188.40	0.36	193.80	188.54	0.36	0.07%	0.00%
	201.49	195.94	0.31	201.49	195.94	0.31	0.00%	0.00%
	201.87	197.61	0.37	201.87	197.61	0.37	0.00%	0.00%
	387.59	384.03	0.37	387.59	384.31	0.37	0.07%	0.00%

Table 6.3 Computational Results from Test2

From Test 1 and Test 2, we can see that the relevant results for both MATLAB and VC++ code agree with each other very well, even though they have some different Power Spectrum as input data. So that the two programs are validated.

Chapter 7 Further Study

This chapter introduces Loudness and Aures's Model of Tonality. The Psychoacoustic concepts, which are used in Loudness such as masking patterns, critical bands, bark scale, we have introduced in chapter 2. The Stationary Loudness model is built as a MATLAB library and tested with a set of reference signals.

7.1 Loudness

Loudness is a description of how human beings perceives the physical sound pressure level of a sound. A computational model for loudness must take some of the main properties of our hearing into account. The most important ones are masking, frequency selectivity, threshold in quiet and on accounting for frontal (free) sound field and diffuse sound field (Zwicker, 1981).

7.1.1 Stationary Loudness and Non-stationary Loudness

There are two types of loudness – stationary loudness for stationary signals and non-stationary loudness for non-stationary signals.

- A stationary signal is a signal that is perceived by a human as not to change over time. For example: noise from waterfall or a tone at constant level. The stationary loudness is described in ISO 532 and DIN 45632, and DIN45632 is based on the BASIC computer program contained in the German standard DIN 45631,1991.

- Non-Stationary signals are signals that are perceived to change over time. For example: transient signals, like a hammer hitting a nail, and modulating signals, like an alternating ambulance horn. Currently there is no standard for the calculation of non-stationary loudness. According to Poul Ladegaard, Brüel & Kjær, a standard for non-stationary Loudness is not finished yet. An International committee is working on to standardize it. It is expect to be standardized within half a year.

7.1.2 Method A and Method B for Stationary Loudness

There are two methods for calculating Stationary Loudness, according to the international Standard ISO 532.

- The first Method A, referred to as Stevens method, utilizes physical measurement obtained from spectrum analysis in terms of octave bands.

- The second Method B, referred to as Zwicker's method, utilizes spectrum analysis in terms of one-third octave bands.

Method B, which is for Stationary Loudness, is the widely used method in Sound Quality field and also more complex than the Stevens method A. Method B provides

results corresponding well with subjective tests. In our project, we only consider Method B.

7.1.3 Zwicker's Stationary Loudness Model

Figure 7.1 illustrates the fundamentals in the Zwicker's stationary loudness model.

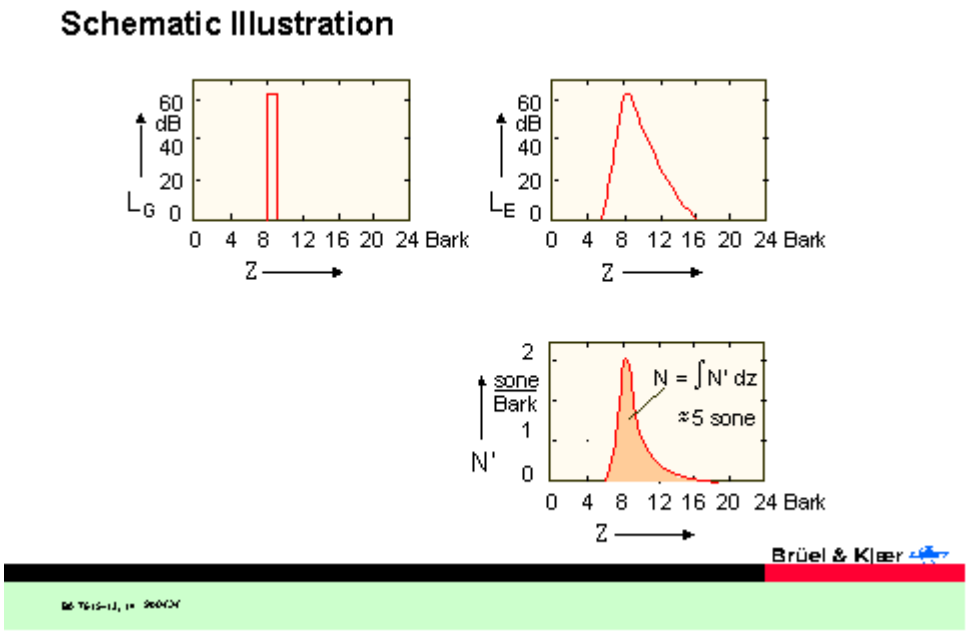

Figure 7.1 Zwicker's Stationary Loudness model

The left panel shows what a narrow band of noise, centred around 1 kHz corresponding to 8.5 Bark, will look like on an ordinary analyser. The equal sound pressure level of the critical band wide noises is 64 dB. The upper right panel shows how it is perceived by the human including its masking pattern. The lower right panel shows the specific loudness pattern. That is, the loudness value per critical Bark band measured in sone. The masking pattern is transformed into Loudness by use of equal loudness contours. The most important feature of this Zwicker loudness model is that the area under the specific loudness curve N' (shaded area) is directly proportional to the perceived loudness. This direct relationship is the great advantage of loudness patterns in comparison to alternative spectral representations like FFT spectra or 1/3 octave-band spectra. [7.2]

7.1.4 The Standard (ISO R 532) of Stationary Loudness

The procedure for standard Stationary Loudness is based on five empirical relations or concepts, according to the Stationary Loudness standard [7.1]:

- The widest frequency bands in which the loudness level depends only on the sound pressure level (critical bands) (in German *Frequenzgruppen*).
- A rule relating the total loudness of a sound to the contributions from the critical bands (*Frequenzgruppen*).

- A relation between the part of the loudness appropriate to each band and the centre frequency of that band.
- The difference between the equal-loudness contours for frontal (free) sound and for a diffuse sound field.
- A loudness function relating loudness in sones to loudness level in phons, which is identical to that given in ISO/R 131.

Figure 7.2 shows the general procedure for calculating the Stationary Loudness according to Method B. We will give a brief description of each step.

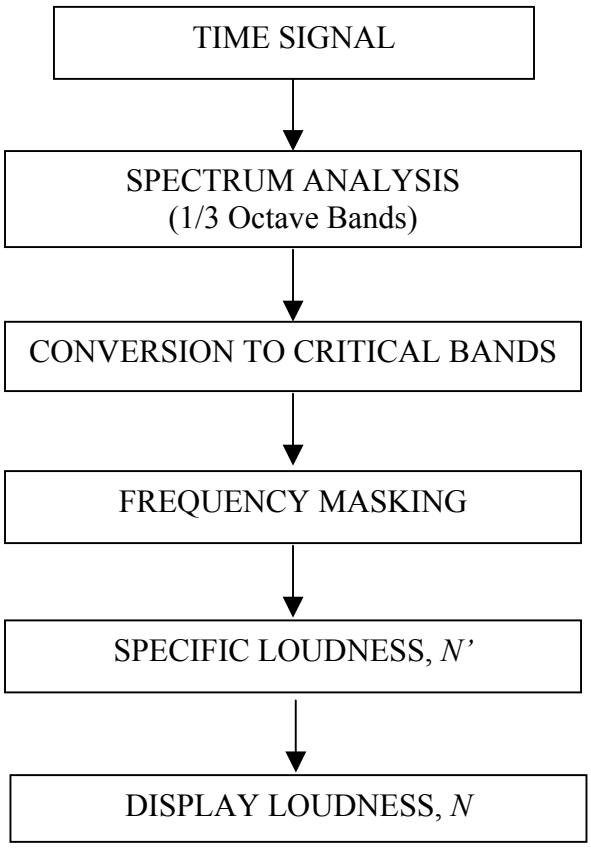

Figure 7.2 General procedures for calculating the Stationary Loudness according to Method B

- **Time signal**

Any signals you want to analyse for calculate Stationary Loudness.

- **Spectrum Analysis**

28 1/3 octave filters Spectrum Analysis covering the frequencies audible by the human being.

- **Conversion to critical bands**

Above 500 Hz, critical bands are approximately one-third octave in width. Since the lower one-third octave filters are narrower than 100 Hz, the bands below 500 Hz are combined to approximate critical bands. The 28 1/3 Octave bands are combined to the critical bands in the following way. In the lowest frequency region, the one-third octave bands with centre frequencies from 25 to 80 Hz are combined to form the first band. The one-third octave bands from 100 to 160 Hz are combined to form the next band. The 200 and 250 Hz bands are combined. The bands from 315 to 12500 Hz are used individually, since their bandwidths approximate critical bands. The DIN 45631 standard describes how to obtain values for critical banded based on one-third octave bands.

- **Frequency masking**

After conversion from 28 1/3 octave bands into 20 Critical bands a correction is applied depending whether the environment is a free or diffuse field. The SPL values are then converted into loudness values and frequency masking is applied. The frequency masking is only for the masking of higher critical bands by lower critical bands. The masking of lower critical bands is considered to be negligible.

- **Specific Loudness, N'**

Each critical band can be displayed as a specific loudness spectrum, *N'*.

- **Display Loudness, N**

We display all bands of specific loudness summed into a single number as stationary loudness (Total Loudness), *N*.

Figure7.3 shows how the Stationary Loudness spectrum displays in B&K SQ Application. Total Loudness is the sum of Specific Loudness. The curved line represents frequency masking. The Stationary Loudness is the area under the curve.

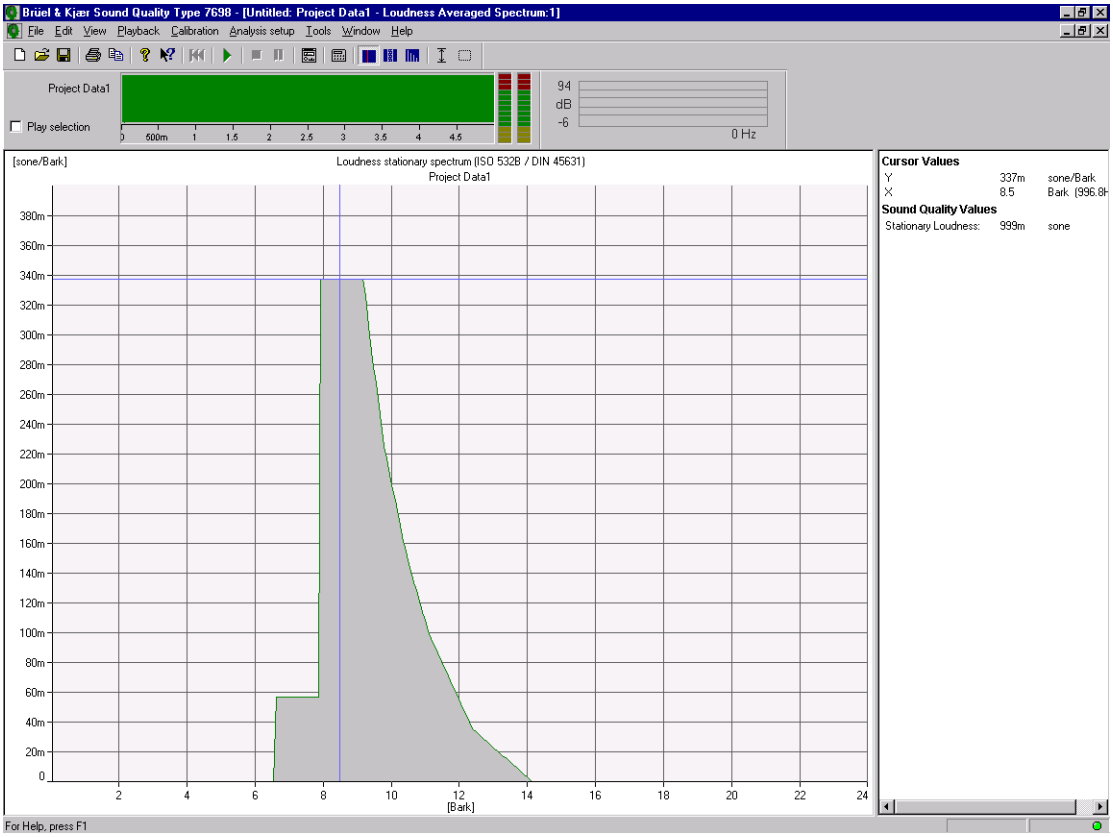

Figure7.3 The loudness spectrum displays in B&K SQ Application.

7.1.5 Testing of MATLAB Loudness Program

We got a preliminary MATLAB program from Torben Poulsen, Ørsted•DTU, Acoustic Technology, which implement the calculation procedure for Stationary Loudness according to DIN 45631. We modified some of errors, which was in MATLAB program and it has been tested and verified by comparing its output with B&K Sound Quality results. The source code is provided in Appendix D.

The Stationary Loudness program in B&K Sound Quality Application has been validated by Round Robin Tests. So by comparing the results from MATLAB and B&K Sound Quality Application, the MATLAB program can be validated if the computational results are similar to some extend.

In order to test MATLAB program, a conventional digital CPB analysis is required for the spectrum analysis. Frequency region is about 20Hz to 20 kHz. In order to transfer the signal from B&K Sound Quality Application to MATLAB, we export the spectrum data of CPB analysis from B&K Sound Quality Application into Microsoft Excess and then import to MATLAB.

The test signals are different pure tones with different frequency and SPL value. The descriptions of test signal types are shown in Table 7.1. The CPB analysis data for MATLAB tests are provided in Appendix E.

The following table contains the test results for Stationary Loudness from B&K Sound Quality type 7698 and MATLAB.

No	Signal type	Loudness in Theory (≈)	Loudness (unit: sone)					
			B&K SQ		MATLAB		Difference	
			Free (F1)	Diffuse (D1)	Free (F2)	Diffuse (D2)	|F2-F1|/F1	|D2-D1|/D1
1	Pure Tone at 1kHz of 40 dB	1	0,99	1,26	1,0	1,3	1.01%	3.17%
2	Pure Tone at 2kHz of 80 dB	19	19,4	17,9	19,3	17,9	0.52%	0.00%
3	Pure Tone at 4kHz of 80 dB	24	24,6	22,8	24,0	22,5	2.44%	1.32%
4	Pure Tone at 8kHz of 80 dB	13	13,1	16,5	12,6	15,8	3.82%	4.24%

Table 7.1 Test Result from the Stationary Loudness

Table 7.1 shows that the maximal difference of Stationary Loudness result in free field is 3.82%, and the maximal difference of Stationary Loudness result in diffuse field is 4.24%. The difference is acceptable, because we think it is courses by the different programming languages and the fact that the input data to MATLAB is slightly different from the Sound Quality Application.

Besides the calculation results for Stationary Loudness, MATLAB program can also show some spectrums. Figure 7.4 shows the spectrum when we made the first test.

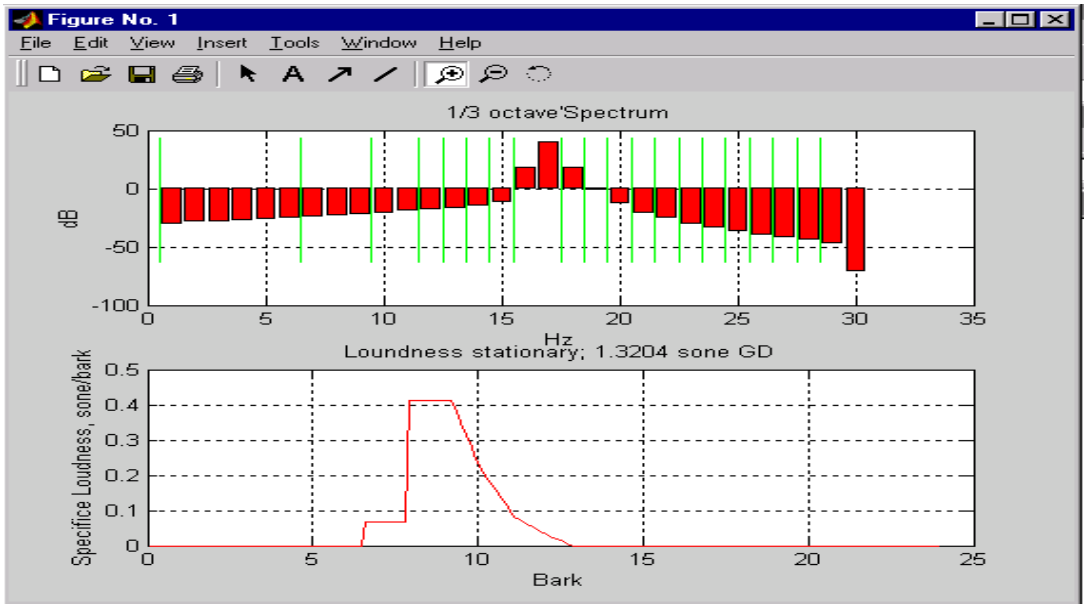

Figure 7.4 Stationary Loudness Test Spectrum for a pure tone at 1 kHz 40 dB in MATLAB

The upper plot shows 28 1/3 octave filters Spectrum Analysis. The vertical lines shows how critical bands are achieved by combination of 1/3 Octave bands. The lower plot shows total Loudness by summing of Specific Loudness. The curved line represents frequency masking. The Stationary Loudness is the area under the curve. The rest of the test figures are listed in Appendix F.

7.1.6 The Available Programs for the Stationary Loudness

Noise measurement techniques based on features of the human hearing system are nowadays widely used. In particular with respect to the hearing sensation loudness, measurement systems from different manufactures are on the market. Although the basics of loudness calculations are laid down in a computer program published in German standard DIN 45631, different manufacturers may use various implementations. Therefore, it is of interest to compare for identical sounds the indications of several loudness analysis systems. Different manufacturers stand for different implementations like CISC versus RISC processors or FFT versus 1/3 octave band analysis. The below table provides an overview of the companies involved, the name of their system as well as the version of the stationary loudness algorithm used.

Company	System	Version
Akustik Technologie Gottingen	Si++	3.4
Bruel & Kjær	7698	1.21
Head acoustics	BAS	4.40
Muller-BBM	PAK	4.0
Neutrik Cortex Instruments	CF 90	30.38
S.A.S. Systems	SASS Win	1.0

Table 7.2 The available programs for stationary loudness

Sound Quality Program Type 7698 is a software package that runs under Microsoft® Windows NT®. It uses optional software BZ5625 to calculate Zwicker Loudness. Sound quality software in B&K was developed in VC++.

7.2 Aures's Model of Tonality

Aures's model of tonality is a relatively new method. It is based on Terhardt's method, but differs in such a way that it takes into account the frequency, bandwidth and level of all tonal components as well as the effect of noise. It generates a number for Tonality based on Tonal Weighting and Tonal Loudness Weighting. This method uses the same calculation for SPL excess as Terhardt's method and introduces Loudness calculation into the Tonality calculation.

Aures's model of Tonality is shown in Figure 8.1. After spectrum analysis, the calculations fall into two paths. One is for the tonal component based calculation procedure, which involves the calculation of SPL excess for each tonal component and the Tonal Weighting function. The other is the calculation of Total Loudness of the whole spectrum and the Loudness of the noise spectrum and finally gets the Tonal Loudness Weighting. Noise spectrum means the spectrum, which remove the tonal or narrow band components.

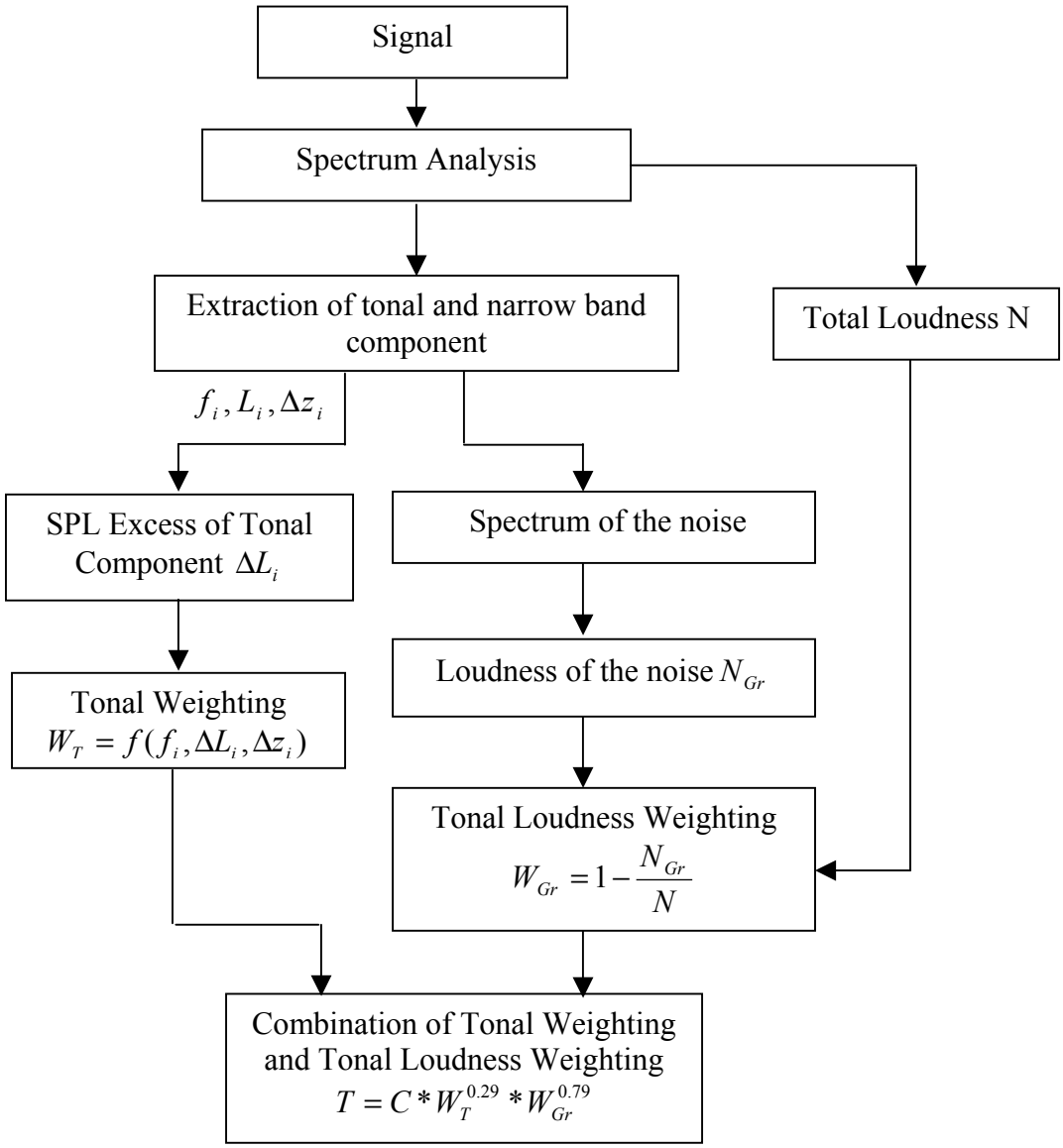

Figure 8.1 Aures's Model of Tonality

The tonal weighting is calculated for all components with positive SPL excess levels.

$$w_T = \sqrt{\sum^{n}\left[w_1'(\Delta z_i)w_2'(f_i)w_3'(\Delta L_i)\right]} \tag{20}$$

where

Δz is the bandwidth of the component in Bark.
f_i is the central frequency of tonal component in Hz.
ΔLi is the SPL excess of the tonal component.

$$w'_n = w_n^{1/0.29} \text{ , for n =1,2,3.} \tag{21}$$

The weighting functions w_1, w_2, w_3 are calculated as follows.

$$w_1(\Delta z_i) = \frac{0.13}{\Delta z + 0.13} \tag{22}$$

$$w_2(f_i) = \left(\frac{1}{\sqrt{1 + 0.2(\frac{f}{700} + \frac{700}{f})}} \right)^{0.29} \tag{23}$$

$$w_3(\Delta L_i) = \left(1 - \exp^{-\frac{\Delta l}{15}} \right)^{0.29} \tag{24}$$

Aures's model of Tonality considers the level dependence of human hearing system by using Loudness. It also shows a way of how to extract a single number for Tonality. But compared to the Tonality algorithm we presented in Chapter 3, Aures's model of Tonality is relatively simple and does not consider the virtual pitch perception. More information about Aures's model of Tonality, please refer to [7.7] and [7.8].

Chapter 8 Conclusions and Further work

The work carried out in this thesis can be described as four major parts. The first part is the study of Terhardt's Tonality algorithm and the simulation of the algorithm by using MATLAB. The second part is the study of ATL COM programming in order to make a metric for B&K SQ Application, and the implementation of Terhardt's Tonality algorithm in VC++. The third part is the testing of MATLAB and VC++ code by showing the consistency of the computational results. The fourth part is the study of Stationary Loudness and Aures's model of Tonality.

Simulating the algorithm in MATLAB helped us to understand the calculation procedure in detail, and proved that the algorithm could be implemented according to our Data Flow analysis. The experience of VC++ programming for Tonality metric gave us an opportunity to learn how COM technology being used in B&K Sound Quality Application, and how the same algorithm can be implemented using different programming languages. Testing results showed that the computational results agree with each other, so that our VC++ code is ready to be used in B&K Sound Quality Application for further research or development work. The study of Stationary Loudness and Aures's model of Tonality made us understand how psychoacoustics concepts are being used in Loudness and Tonality and brought us some fresh ideas for further discussion.

We think the following work can be considered and carried out based on our thesis project.

- Extract a single number and make subjective test for Tonality

A single number for Tonality means it could statistically show the proportional relationship between Tonality metric and the tonal perception of human hearing system for a signal, which has the tonal attribute. The Tonality algorithm used in this thesis described that in an actual listening situation, the whole pitch percept is described as a competition between spectral and virtual pitches. These two types of pitches are extracted from the signal and represented by the spectral-pitch pattern and the virtual-pitch pattern. The evaluation of the competition between spectral and virtual pitches becomes the important issues in order to extract a single number for Tonality. We are sure our MATLAB model can help to evaluate the competition, but further work need to be done in order to find a way to extract a single number for Tonality. For example, by finding a weighting function, which can be applied on both spectral-pattern and virtual- pitch pattern, the single number for Tonality can be extracted. This should be correlated with results from subjective test.

- Improve Tonality by using parts of the Loudness Model

Mutual masking of spectral components has been evaluated in the Tonality algorithm through the calculation procedures for sound-pressure level excess and pitch shifts. The Stationary Loudness is an ISO Standard, which evaluate the masking effects based on Critical Bands and finally get a single number for Loudness from the specific Loudness. The single number we get from Loudness contains the overall information of how human being perceives the physical sound pressure level of a

sound. So if we can find a right way to introduce Loudness into Tonality, the calculation of Tonality will have more precision, better correlation with subjective results and more standardized evaluation of masking effects.

Aures's model is an example of trying to combine Tonality and Loudness. But we think they use Tonality in a very simple way, which is not enough to evaluate the tonal aspects of a sound. Aures's model can be built based on our Tonality MATLAB model and the Loudness MATLAB Model for further research work.

- Tonality for Non-stationary signal

Terhardt's Tonality algorithm, which we discussed in Chapter 3, is a pitch-evaluation procedure for any complex tone. The algorithm can work well with stationary signals, but for non-stationary signals it might have some drawbacks. For example, it will evaluate a strong tonal component as a non-tonal component if the frequency of the tone changes with time. The reason is the spectrum from FFT analysis of a non-stationary signal might merge some discrete tonal contents which can be seen in time domain, and show them as non-tonal contents in frequency domain.

By assuming that the time signal can be composed by a sequence of short duration signal, we can divide the non-stationary signal into some small time blocks. Each of them can be understood as a stationary signal when the time duration is small enough. Next, evaluate each of them by using the Tonality algorithm. In the end, you will get a spectral pitch pattern and virtual pitch pattern for each short duration signal, which you can show together in a contour or waterfall graph with both frequency and time axis. Inspecting the graph more tones may show up and contribute to the overall Tonality compared to just analysing a single spectrum.

- Improvement of Tonality VC++ code

The Tonality metric implementation in VC++ follows the general routines of making a user-defined cursor reading control. It is the same as how B&K SQ customers make their own metrics for SQ Application. The main idea of this program structure is to simplify the connection between the user-defined metric and the main SQ program. B&K also has some metrics developed inside SQ program, which all provide the function of showing the spectrums and contours. These functions are very hard to implement by using ATL COM programming. But by embedding Tonality as a part of main SQ program, more functions can be provided for the customers, such as showing the spectrum-pitch pattern and virtual-pitch pattern in a spectrum. We suggest that Tonality should be implemented as a part of SQ main program if one day it becomes part of the commercial software.

Bibliography

[1.1] Joseph Emmanuel Amuah, "Sound Quality User-defined Cursor Reading Controls", Master Thesis, IMM, DTU, Lyngby, Denmark. June 2002

[2.1] E. Zwicker, H. Fastel: Psychoacoustics, Facts and Models, Springer-Verlag Berlin Heidelberg 1999.
[2.2] B&KA/S, Lecture notes 'Basic Concepts of Sound'
[2.3] B&KA/S, Lecture notes 'Basic Frequency Analysis of Sound'
[2.4] B&KA/S, Lecture notes 'Sound Quality Equalisation and Calibration'
[2.5] B&KA/S, Guided Tour - System Calibration, Sound Recording and Playback
[2.6] B&KA/S, Lecture notes 'Psychoacoustics – a qualitative description'.
[2.7] B&KA/S, Lecture notes 'Introduction to Sound Quality'.
[2.8] http://www2.egr.uh.edu/~glover/applets/Sampling/Sampling.html (1/10/2002).
[2.9] http://www.ihear.com/Pitch/paradoxical.html (1/10/2002).
[2.10] http://acoustics.auc.dk/research/SQRU/index.php Sound Quality Research Unit (15/10/2002)
[2.11] http://www.sfu.ca/sonic-studio/handbook/index.html Handbook For Acoustic Ecology. (20/10/2002)

[3.1] Terhardt, E., Stoll, G., and Seewann, M.(1982). "An algorithm for extraction of pitch and pitch salience from complex tonal signals," J. Acoust. Soc. Am. 71, 679-688.
[3.2] Terhardt, E., Stoll, G., and Seewann, M.(1982). "Pitch of complex signals according to virtual-pitch theory: Test, examples, and predictions," J. Acoust. Soc. Am. 71, 671-678.
[3.3] International Standard ISO 532 Acoustics-Method for calculating loudness level.

[4.1] MATLAB Help

[5.1] Richard Grimes, *Beginning ATL 3 COM Programming*, Wrox Press, Birmingham, UK, 1999.
[5.2] Stanley B. Lippman, Josee Lajoie, *C++ Primer,* Addison-Wesley, Canada, 1999.
[5.3] The Microsoft Developer Network (MSDN) Library, October 2001.
[5.4] David J.Kruglinski, *Inside Visual C++ ,* Microsoft Press, Washington, 1993.
[5.5] Bjarne Stroustrup, The C++ Programming Language, Second Edition, AT&T Bell Laboratories, Murray Hill, New Jersey, 1992.

[7.1] E.Zwicker and H.Fastl, Psychoacoustics. Springer Verlag, 2nd ed.,1999
[7.2] B&Ksoftware, Sound Quality type 7698
[7.3] May, P.G., "Assessment of Refrigerator Sound Quality", Purdue University, 1998.
[7.4] International standard ISO 131,
[7.5] International standard ISO 532, Acoustics –Method for calculation loudness level, 1st ed., 1975
[7.6] Brüel & Kjær, "Psychoacoustics", Lecture, BA 7615-13.
[7.7] W.Aures, Procedure for Calculating the Sensory Euphony of Arbitrary Sound Signals, *Acusitca* 59, 130-141, 1985.

[7.8]Aaron Hastings and Patricia Davies, An examination of Aures's model of Tonality, InterNoise 2002.

Appendix A MATLAB Source Code for Tonality

Appendix B VC++ Source Code for Tonality

Appendix C Power Spectrum Data for Test1 and Test2

Test1: Maximal SPL Difference = 0.19%
Test2: Maximal SPL Difference = 0.07%

Power Spectrum of Test 1					Power Spectrum of Test 2				
MATLAB		VC++		(SPL2-SPL1)/SPL2	MATLAB		VC++		(SPL4-SPL3)/SPL4
Freq. (Hz)	SPL1 (dB)	Freq. (Hz)	SPL2 (dB)		Freq. (Hz)	SPL3 (dB)	Freq. (Hz)	SPL4 (dB)	
0.00	56.28	0.00	56.30	0.04%	0.00	51.49	0.00	51.51	0.04%
10.77	54.37	10.77	54.39	0.04%	10.77	50.42	10.77	50.44	0.04%
21.53	47.19	21.53	47.21	0.04%	21.53	48.97	21.53	48.99	0.04%
32.30	42.95	32.30	42.97	0.05%	32.30	48.93	32.30	48.95	0.04%
43.07	38.71	43.07	38.73	0.05%	43.07	49.76	43.07	49.78	0.04%
53.83	34.79	53.83	34.81	0.06%	53.83	50.26	53.83	50.28	0.04%
64.60	30.68	64.60	30.70	0.07%	64.60	50.35	64.60	50.37	0.04%
75.37	30.96	75.37	30.98	0.06%	75.37	49.10	75.37	49.12	0.04%
86.13	34.44	86.13	34.46	0.06%	86.13	49.36	86.13	49.38	0.04%
96.90	47.69	96.90	47.71	0.04%	96.90	49.34	96.90	49.36	0.04%
107.67	56.08	107.67	56.10	0.04%	107.67	48.93	107.67	48.95	0.04%
118.43	56.83	118.43	56.85	0.04%	118.43	48.87	118.43	48.89	0.04%
129.20	56.25	129.20	56.27	0.04%	129.20	47.47	129.20	47.49	0.04%
139.97	55.49	139.97	55.51	0.04%	139.97	49.59	139.97	49.62	0.06%
150.73	54.61	150.73	54.63	0.04%	150.73	49.35	150.73	49.37	0.04%
161.50	51.36	161.50	51.38	0.04%	161.50	49.32	161.50	49.34	0.04%
172.27	42.02	172.27	42.04	0.05%	172.27	49.03	172.27	49.05	0.04%
183.03	34.99	183.03	35.01	0.06%	183.03	49.29	183.03	49.31	0.04%
193.80	35.80	193.80	35.82	0.06%	193.80	49.05	193.80	49.07	0.04%
204.57	45.26	204.57	45.28	0.04%	204.57	49.15	204.57	49.17	0.04%
215.33	53.91	215.33	53.93	0.04%	215.33	48.91	215.33	48.93	0.04%
226.10	54.98	226.10	55.00	0.04%	226.10	48.93	226.10	48.95	0.04%
236.87	53.77	236.87	53.79	0.04%	236.87	48.96	236.87	48.98	0.04%
247.63	55.36	247.63	55.38	0.04%	247.63	49.18	247.63	49.20	0.04%
258.40	57.31	258.40	57.33	0.03%	258.40	48.72	258.40	48.75	0.06%
269.17	55.94	269.17	55.96	0.04%	269.17	48.87	269.17	48.89	0.04%
279.93	51.85	279.93	51.87	0.04%	279.93	49.27	279.93	49.29	0.04%
290.70	49.88	290.70	49.90	0.04%	290.70	49.24	290.70	49.27	0.06%
301.47	48.31	301.46	48.33	0.04%	301.47	49.75	301.46	49.77	0.04%
312.23	47.14	312.23	47.16	0.04%	312.23	49.16	312.23	49.19	0.06%
323.00	52.91	323.00	52.93	0.04%	323.00	48.94	323.00	48.96	0.04%
333.76	55.15	333.76	55.17	0.04%	333.76	49.27	333.76	49.29	0.04%
344.53	53.04	344.53	53.06	0.04%	344.53	49.28	344.53	49.30	0.04%
355.30	53.03	355.30	53.05	0.04%	355.30	48.80	355.30	48.82	0.04%
366.06	54.09	366.06	54.11	0.04%	366.06	49.09	366.06	49.11	0.04%
376.83	56.31	376.83	56.33	0.04%	376.83	81.86	376.83	81.88	0.02%
387.60	58.13	387.60	58.15	0.03%	387.60	87.87	387.60	87.89	0.02%
398.36	56.95	398.36	56.97	0.04%	398.36	81.85	398.36	81.87	0.02%
409.13	53.23	409.13	53.25	0.04%	409.13	49.39	409.13	49.42	0.06%
419.90	51.32	419.90	51.34	0.04%	419.90	48.69	419.90	48.71	0.04%
430.66	54.41	430.66	54.43	0.04%	430.66	49.33	430.66	49.35	0.04%
441.43	54.67	441.43	54.69	0.04%	441.43	49.55	441.43	49.57	0.04%
452.20	50.90	452.20	50.92	0.04%	452.20	49.15	452.20	49.17	0.04%
462.96	48.38	462.96	48.40	0.04%	462.96	49.41	462.96	49.43	0.04%
473.73	46.08	473.73	46.10	0.04%	473.73	49.72	473.73	49.74	0.04%
484.50	46.07	484.50	46.09	0.04%	484.50	48.84	484.50	48.87	0.06%
495.26	45.04	495.26	45.06	0.04%	495.26	48.71	495.26	48.73	0.04%

506.03	42.25	506.03	42.27	0.05%	506.03	48.61	506.03	48.63	0.04%
516.80	40.51	516.80	40.53	0.05%	516.80	48.62	516.80	48.64	0.04%
527.56	39.39	527.56	39.41	0.05%	527.56	48.79	527.56	48.81	0.04%
538.33	39.47	538.33	39.49	0.05%	538.33	49.00	538.33	49.02	0.04%
549.10	41.97	549.10	41.99	0.05%	549.10	49.93	549.10	49.95	0.04%
559.86	39.32	559.86	39.34	0.05%	559.86	49.58	559.86	49.60	0.04%
570.63	35.92	570.63	35.94	0.06%	570.63	49.23	570.63	49.25	0.04%
581.40	35.76	581.40	35.78	0.06%	581.40	49.61	581.40	49.63	0.04%
592.16	35.02	592.16	35.04	0.06%	592.16	49.83	592.16	49.85	0.04%
602.93	33.09	602.93	33.11	0.06%	602.93	49.86	602.93	49.88	0.04%
613.70	32.97	613.70	32.99	0.06%	613.70	50.23	613.70	50.26	0.06%
624.46	33.56	624.46	33.58	0.06%	624.46	49.50	624.46	49.52	0.04%
635.23	33.92	635.23	33.94	0.06%	635.23	48.36	635.23	48.38	0.04%
646.00	36.16	646.00	36.19	0.08%	646.00	49.53	646.00	49.55	0.04%
656.76	36.28	656.76	36.30	0.06%	656.76	49.88	656.76	49.90	0.04%
667.53	33.01	667.53	33.03	0.06%	667.53	49.27	667.53	49.29	0.04%
678.30	29.53	678.30	29.55	0.07%	678.30	49.38	678.30	49.40	0.04%
689.06	26.61	689.06	26.64	0.11%	689.06	49.55	689.06	49.57	0.04%
699.83	26.21	699.83	26.23	0.08%	699.83	49.39	699.83	49.41	0.04%
710.60	26.77	710.60	26.79	0.07%	710.60	49.10	710.60	49.12	0.04%
721.36	26.69	721.36	26.71	0.07%	721.36	49.97	721.36	50.00	0.06%
732.13	25.71	732.13	25.73	0.08%	732.13	49.61	732.13	49.63	0.04%
742.90	24.87	742.90	24.89	0.08%	742.90	48.45	742.90	48.47	0.04%
753.66	28.68	753.66	28.70	0.07%	753.66	49.13	753.66	49.15	0.04%
764.43	33.20	764.43	33.22	0.06%	764.43	48.86	764.43	48.88	0.04%
775.20	31.98	775.20	32.01	0.09%	775.20	48.34	775.20	48.36	0.04%
785.96	29.53	785.96	29.55	0.07%	785.96	48.34	785.96	48.36	0.04%
796.73	28.79	796.73	28.81	0.07%	796.73	84.97	796.73	84.99	0.02%
807.50	25.52	807.50	25.54	0.08%	807.50	90.96	807.50	90.98	0.02%
818.26	24.34	818.26	24.36	0.08%	818.26	84.96	818.26	84.98	0.02%
829.03	25.68	829.03	25.70	0.08%	829.03	49.82	829.03	49.84	0.04%
839.80	24.06	839.79	24.08	0.08%	839.80	48.84	839.79	48.86	0.04%
850.56	21.94	850.56	21.96	0.09%	850.56	48.51	850.56	48.53	0.04%
861.33	22.57	861.33	22.59	0.09%	861.33	49.25	861.33	49.27	0.04%
872.10	25.21	872.09	25.24	0.12%	872.10	49.57	872.09	49.59	0.04%
882.86	24.33	882.86	24.35	0.08%	882.86	48.80	882.86	48.82	0.04%
893.63	22.55	893.63	22.57	0.09%	893.63	48.80	893.63	48.82	0.04%
904.39	22.04	904.39	22.06	0.09%	904.39	48.45	904.39	48.47	0.04%
915.16	22.72	915.16	22.74	0.09%	915.16	48.17	915.16	48.19	0.04%
925.93	21.64	925.93	21.66	0.09%	925.93	48.66	925.93	48.68	0.04%
936.69	20.04	936.69	20.06	0.10%	936.69	49.15	936.69	49.17	0.04%
947.46	19.73	947.46	19.75	0.10%	947.46	48.50	947.46	48.52	0.04%
958.23	20.96	958.23	20.98	0.10%	958.23	49.09	958.23	49.11	0.04%
968.99	21.95	968.99	21.97	0.09%	968.99	49.70	968.99	49.72	0.04%
979.76	25.52	979.76	25.54	0.08%	979.76	50.09	979.76	50.11	0.04%
990.53	26.51	990.53	26.53	0.08%	990.53	49.21	990.53	49.23	0.04%
1001.29	22.31	1001.29	22.33	0.09%	1001.29	48.95	1001.29	48.97	0.04%
1012.06	20.94	1012.06	20.96	0.10%	1012.06	49.62	1012.06	49.64	0.04%
1022.83	20.05	1022.83	20.07	0.10%	1022.83	49.61	1022.83	49.63	0.04%
1033.59	22.58	1033.59	22.60	0.09%	1033.59	49.20	1033.59	49.22	0.04%
1044.36	22.70	1044.36	22.72	0.09%	1044.36	50.23	1044.36	50.25	0.04%
1055.13	19.50	1055.13	19.52	0.10%	1055.13	49.13	1055.13	49.16	0.06%
1065.89	19.91	1065.89	19.93	0.10%	1065.89	48.97	1065.89	48.99	0.04%
1076.66	22.77	1076.66	22.79	0.09%	1076.66	49.44	1076.66	49.46	0.04%
1087.43	22.72	1087.43	22.74	0.09%	1087.43	50.00	1087.43	50.02	0.04%
1098.19	22.43	1098.19	22.45	0.09%	1098.19	49.14	1098.19	49.16	0.04%
1108.96	20.85	1108.96	20.87	0.10%	1108.96	48.68	1108.96	48.70	0.04%
1119.73	20.57	1119.73	20.59	0.10%	1119.73	49.49	1119.73	49.51	0.04%
1130.49	19.32	1130.49	19.34	0.10%	1130.49	49.69	1130.49	49.71	0.04%
1141.26	18.68	1141.26	18.70	0.11%	1141.26	49.12	1141.26	49.14	0.04%

1152.03	19.10	1152.03	19.12	0.10%	1152.03	49.06	1152.03	49.08	0.04%
1162.79	21.58	1162.79	21.60	0.09%	1162.79	48.62	1162.79	48.64	0.04%
1173.56	20.80	1173.56	20.82	0.10%	1173.56	48.96	1173.56	48.98	0.04%
1184.33	20.60	1184.33	20.62	0.10%	1184.33	49.17	1184.33	49.19	0.04%
1195.09	23.33	1195.09	23.35	0.09%	1195.09	49.22	1195.09	49.24	0.04%
1205.86	24.91	1205.86	24.93	0.08%	1205.86	49.29	1205.86	49.31	0.04%
1216.63	25.30	1216.63	25.32	0.08%	1216.63	49.46	1216.63	49.48	0.04%
1227.39	23.82	1227.39	23.84	0.08%	1227.39	48.94	1227.39	48.96	0.04%
1238.16	19.21	1238.16	19.23	0.10%	1238.16	48.70	1238.16	48.72	0.04%
1248.93	18.59	1248.93	18.61	0.11%	1248.93	48.84	1248.93	48.86	0.04%
1259.69	17.33	1259.69	17.35	0.12%	1259.69	49.43	1259.69	49.45	0.04%
1270.46	19.60	1270.46	19.62	0.10%	1270.46	49.95	1270.46	49.97	0.04%
1281.23	21.23	1281.23	21.25	0.09%	1281.23	49.65	1281.23	49.67	0.04%
1291.99	21.51	1291.99	21.54	0.14%	1291.99	49.33	1291.99	49.35	0.04%
1302.76	22.18	1302.76	22.20	0.09%	1302.76	50.26	1302.76	50.28	0.04%
1313.53	21.59	1313.53	21.61	0.09%	1313.53	49.30	1313.53	49.32	0.04%
1324.29	20.09	1324.29	20.11	0.10%	1324.29	49.34	1324.29	49.36	0.04%
1335.06	20.38	1335.06	20.40	0.10%	1335.06	49.08	1335.06	49.10	0.04%
1345.83	20.45	1345.83	20.47	0.10%	1345.83	48.79	1345.83	48.81	0.04%
1356.59	19.54	1356.59	19.56	0.10%	1356.59	48.84	1356.59	48.86	0.04%
1367.36	17.82	1367.36	17.84	0.11%	1367.36	48.93	1367.36	48.95	0.04%
1378.13	16.24	1378.13	16.26	0.12%	1378.13	48.23	1378.13	48.25	0.04%
1388.89	18.23	1388.89	18.25	0.11%	1388.89	48.58	1388.89	48.60	0.04%
1399.66	18.91	1399.66	18.93	0.11%	1399.66	84.97	1399.66	84.99	0.02%
1410.42	23.24	1410.42	23.26	0.09%	1410.42	90.97	1410.42	90.99	0.02%
1421.19	24.04	1421.19	24.06	0.08%	1421.19	84.94	1421.19	84.96	0.02%
1431.96	23.11	1431.96	23.13	0.09%	1431.96	48.91	1431.96	48.94	0.06%
1442.72	22.49	1442.72	22.51	0.09%	1442.72	49.20	1442.72	49.22	0.04%
1453.49	24.38	1453.49	24.40	0.08%	1453.49	49.50	1453.49	49.52	0.04%
1464.26	24.46	1464.26	24.49	0.12%	1464.26	49.16	1464.26	49.18	0.04%
1475.02	23.97	1475.02	23.99	0.08%	1475.02	49.05	1475.02	49.07	0.04%
1485.79	21.65	1485.79	21.67	0.09%	1485.79	49.03	1485.79	49.05	0.04%
1496.56	20.05	1496.56	20.07	0.10%	1496.56	49.88	1496.56	49.90	0.04%
1507.32	19.56	1507.32	19.58	0.10%	1507.32	49.46	1507.32	49.49	0.06%
1518.09	25.57	1518.09	25.59	0.08%	1518.09	48.89	1518.09	48.91	0.04%
1528.86	27.16	1528.86	27.18	0.07%	1528.86	49.23	1528.86	49.25	0.04%
1539.62	24.12	1539.62	24.14	0.08%	1539.62	49.72	1539.62	49.74	0.04%
1550.39	23.79	1550.39	23.81	0.08%	1550.39	50.32	1550.39	50.34	0.04%
1561.16	23.55	1561.16	23.57	0.08%	1561.16	49.60	1561.16	49.62	0.04%
1571.92	24.45	1571.92	24.47	0.08%	1571.92	48.55	1571.92	48.57	0.04%
1582.69	25.49	1582.69	25.51	0.08%	1582.69	48.78	1582.69	48.80	0.04%
1593.46	27.21	1593.46	27.23	0.07%	1593.46	48.86	1593.46	48.88	0.04%
1604.22	25.57	1604.22	25.60	0.12%	1604.22	50.17	1604.22	50.19	0.04%
1614.99	23.45	1614.99	23.47	0.09%	1614.99	49.58	1614.99	49.60	0.04%
1625.76	26.43	1625.76	26.45	0.08%	1625.76	48.63	1625.76	48.65	0.04%
1636.52	28.37	1636.52	28.39	0.07%	1636.52	48.81	1636.52	48.83	0.04%
1647.29	25.76	1647.29	25.78	0.08%	1647.29	49.35	1647.29	49.37	0.04%
1658.06	24.92	1658.06	24.94	0.08%	1658.06	49.59	1658.06	49.61	0.04%
1668.82	27.50	1668.82	27.52	0.07%	1668.82	48.73	1668.82	48.75	0.04%
1679.59	25.97	1679.59	25.99	0.08%	1679.59	49.22	1679.59	49.24	0.04%
1690.36	25.02	1690.36	25.04	0.08%	1690.36	49.81	1690.36	49.83	0.04%
1701.12	29.02	1701.12	29.04	0.07%	1701.12	49.43	1701.12	49.45	0.04%
1711.89	30.64	1711.89	30.66	0.07%	1711.89	48.97	1711.89	48.99	0.04%
1722.66	28.23	1722.66	28.25	0.07%	1722.66	49.19	1722.66	49.21	0.04%
1733.42	28.54	1733.42	28.56	0.07%	1733.42	48.49	1733.42	48.51	0.04%
1744.19	33.55	1744.19	33.57	0.06%	1744.19	49.56	1744.19	49.58	0.04%
1754.96	32.50	1754.96	32.52	0.06%	1754.96	49.72	1754.96	49.74	0.04%
1765.72	25.36	1765.72	25.38	0.08%	1765.72	48.94	1765.72	48.96	0.04%
1776.49	27.82	1776.49	27.84	0.07%	1776.49	48.46	1776.49	48.48	0.04%
1787.26	28.57	1787.26	28.59	0.07%	1787.26	49.43	1787.26	49.45	0.04%

1798.02	28.09	1798.02	28.11	0.07%	1798.02	49.37	1798.02	49.39	0.04%
1808.79	28.62	1808.79	28.64	0.07%	1808.79	48.25	1808.79	48.27	0.04%
1819.56	28.94	1819.56	28.96	0.07%	1819.56	48.67	1819.56	48.69	0.04%
1830.32	32.67	1830.32	32.69	0.06%	1830.32	49.51	1830.32	49.53	0.04%
1841.09	33.98	1841.09	34.00	0.06%	1841.09	49.89	1841.09	49.91	0.04%
1851.86	35.26	1851.86	35.28	0.06%	1851.86	50.07	1851.86	50.09	0.04%
1862.62	39.14	1862.62	39.16	0.05%	1862.62	49.93	1862.62	49.95	0.04%
1873.39	39.69	1873.39	39.71	0.05%	1873.39	49.07	1873.39	49.09	0.04%
1884.16	35.74	1884.16	35.76	0.06%	1884.16	49.06	1884.16	49.08	0.04%
1894.92	34.80	1894.92	34.82	0.06%	1894.92	49.12	1894.92	49.14	0.04%
1905.69	35.70	1905.69	35.72	0.06%	1905.69	48.07	1905.69	48.09	0.04%
1916.46	33.71	1916.46	33.73	0.06%	1916.46	48.68	1916.46	48.70	0.04%
1927.22	35.74	1927.22	35.76	0.06%	1927.22	49.97	1927.22	49.99	0.04%
1937.99	37.44	1937.99	37.46	0.05%	1937.99	49.61	1937.99	49.63	0.04%
1948.75	38.73	1948.75	38.75	0.05%	1948.75	48.70	1948.75	48.72	0.04%
1959.52	42.91	1959.52	42.93	0.05%	1959.52	48.85	1959.5	48.87	0.04%
1970.29	42.00	1970.29	42.02	0.05%	1970.29	48.43	1970.29	48.45	0.04%
1981.05	42.86	1981.05	42.88	0.05%	1981.05	48.57	1981.05	48.59	0.04%
1991.82	45.36	1991.82	45.38	0.04%	1991.82	48.87	1991.82	48.89	0.04%
2002.59	44.50	2002.59	44.52	0.04%	2002.59	49.45	2002.59	49.47	0.04%
2013.35	44.21	2013.35	44.24	0.07%	2013.35	48.84	2013.35	48.86	0.04%
2024.12	42.42	2024.12	42.45	0.07%	2024.12	49.49	2024.12	49.51	0.04%
2034.89	39.37	2034.89	39.39	0.05%	2034.89	49.52	2034.89	49.54	0.04%
2045.65	42.30	2045.65	42.32	0.05%	2045.65	48.94	2045.65	48.96	0.04%
2056.42	42.27	2056.42	42.29	0.05%	2056.42	49.22	2056.42	49.24	0.04%
2067.19	43.55	2067.19	43.57	0.05%	2067.19	50.00	2067.19	50.02	0.04%
2077.95	46.70	2077.95	46.72	0.04%	2077.95	49.54	2077.95	49.56	0.04%
2088.72	43.96	2088.72	43.98	0.05%	2088.72	48.62	2088.72	48.64	0.04%
2099.49	42.98	2099.49	43.00	0.05%	2099.49	48.54	2099.49	48.56	0.04%
2110.25	43.06	2110.25	43.08	0.05%	2110.25	49.38	2110.25	49.40	0.04%
2121.02	40.26	2121.02	40.28	0.05%	2121.02	49.95	2121.02	49.97	0.04%
2131.79	41.32	2131.79	41.35	0.07%	2131.79	49.35	2131.79	49.37	0.04%
2142.55	39.98	2142.55	40.00	0.05%	2142.55	49.51	2142.55	49.53	0.04%
2153.32	38.81	2153.32	38.83	0.05%	2153.32	49.50	2153.32	49.52	0.04%
2164.09	38.79	2164.09	38.81	0.05%	2164.09	48.89	2164.09	48.91	0.04%
2174.85	38.58	2174.85	38.60	0.05%	2174.85	49.52	2174.85	49.54	0.04%
2185.62	40.40	2185.62	40.42	0.05%	2185.62	81.88	2185.62	81.90	0.02%
2196.39	36.57	2196.39	36.59	0.05%	2196.39	87.87	2196.39	87.89	0.02%
2207.15	33.42	2207.15	33.44	0.06%	2207.15	81.85	2207.15	81.87	0.02%
2217.92	34.65	2217.92	34.67	0.06%	2217.92	49.79	2217.92	49.81	0.04%
2228.69	33.42	2228.69	33.44	0.06%	2228.69	50.12	2228.69	50.14	0.04%
2239.45	36.50	2239.45	36.52	0.05%	2239.45	49.30	2239.45	49.32	0.04%
2250.22	35.76	2250.22	35.78	0.06%	2250.22	48.95	2250.22	48.97	0.04%
2260.99	35.21	2260.99	35.24	0.09%	2260.99	48.78	2260.99	48.80	0.04%
2271.75	34.43	2271.75	34.45	0.06%	2271.75	49.21	2271.75	49.23	0.04%
2282.52	34.75	2282.52	34.77	0.06%	2282.52	48.58	2282.52	48.60	0.04%
2293.29	37.65	2293.29	37.67	0.05%	2293.29	49.15	2293.29	49.17	0.04%
2304.05	35.32	2304.05	35.34	0.06%	2304.05	49.39	2304.05	49.41	0.04%
2314.82	32.87	2314.82	32.89	0.06%	2314.82	49.31	2314.82	49.33	0.04%
2325.59	31.44	2325.59	31.46	0.06%	2325.59	48.90	2325.59	48.92	0.04%
2336.35	30.83	2336.35	30.85	0.06%	2336.35	49.59	2336.35	49.61	0.04%
2347.12	32.16	2347.12	32.18	0.06%	2347.12	49.14	2347.12	49.16	0.04%
2357.89	31.15	2357.89	31.17	0.06%	2357.89	48.53	2357.89	48.55	0.04%
2368.65	33.59	2368.65	33.61	0.06%	2368.65	50.17	2368.65	50.19	0.04%
2379.42	37.21	2379.42	37.23	0.05%	2379.42	50.08	2379.42	50.10	0.04%
2390.19	35.30	2390.19	35.32	0.06%	2390.19	49.65	2390.19	49.67	0.04%
2400.95	35.58	2400.95	35.60	0.06%	2400.95	48.90	2400.95	48.92	0.04%
2411.72	36.03	2411.72	36.05	0.06%	2411.72	49.08	2411.72	49.10	0.04%
2422.49	35.12	2422.49	35.14	0.06%	2422.49	48.96	2422.49	48.98	0.04%
2433.25	30.96	2433.25	30.98	0.06%	2433.25	48.54	2433.25	48.56	0.04%

2444.02	30.27	2444.02	30.29	0.07%	2444.02	48.20	2444.02	48.22	0.04%
2454.79	30.50	2454.79	30.52	0.07%	2454.79	48.70	2454.79	48.72	0.04%
2465.55	30.34	2465.55	30.36	0.07%	2465.55	48.69	2465.55	48.71	0.04%
2476.32	32.98	2476.32	33.00	0.06%	2476.32	48.97	2476.32	48.99	0.04%
2487.08	37.10	2487.08	37.12	0.05%	2487.08	49.46	2487.08	49.48	0.04%
2497.85	39.27	2497.85	39.29	0.05%	2497.85	49.74	2497.85	49.76	0.04%
2508.62	41.67	2508.62	41.69	0.05%	2508.62	49.92	2508.62	49.94	0.04%
2519.38	40.37	2519.38	40.39	0.05%	2519.38	49.93	2519.38	49.95	0.04%
2530.15	36.66	2530.15	36.68	0.05%	2530.15	49.48	2530.15	49.50	0.04%
2540.92	36.76	2540.92	36.78	0.05%	2540.92	48.71	2540.92	48.73	0.04%
2551.68	34.22	2551.68	34.24	0.06%	2551.68	49.06	2551.68	49.08	0.04%
2562.45	33.38	2562.45	33.40	0.06%	2562.45	48.84	2562.45	48.86	0.04%
2573.22	33.63	2573.22	33.65	0.06%	2573.22	49.56	2573.22	49.58	0.04%
2583.98	35.32	2583.98	35.34	0.06%	2583.98	49.47	2583.98	49.49	0.04%
2594.75	35.68	2594.75	35.70	0.06%	2594.75	49.00	2594.75	49.02	0.04%
2605.52	37.37	2605.52	37.39	0.05%	2605.52	48.66	2605.52	48.68	0.04%
2616.28	38.76	2616.28	38.78	0.05%	2616.28	49.23	2616.28	49.25	0.04%
2627.05	43.02	2627.05	43.04	0.05%	2627.05	50.09	2627.05	50.11	0.04%
2637.82	43.89	2637.82	43.91	0.05%	2637.82	49.69	2637.82	49.71	0.04%
2648.58	40.58	2648.58	40.60	0.05%	2648.58	49.85	2648.58	49.87	0.04%
2659.35	39.33	2659.35	39.35	0.05%	2659.35	50.02	2659.35	50.04	0.04%
2670.12	38.30	2670.12	38.32	0.05%	2670.12	49.30	2670.12	49.32	0.04%
2680.88	38.27	2680.88	38.29	0.05%	2680.88	48.68	2680.88	48.70	0.04%
2691.65	35.01	2691.65	35.04	0.09%	2691.65	49.43	2691.65	49.45	0.04%
2702.42	38.29	2702.42	38.31	0.05%	2702.42	49.35	2702.42	49.37	0.04%
2713.18	36.68	2713.18	36.70	0.05%	2713.18	49.46	2713.18	49.48	0.04%
2723.95	37.65	2723.95	37.67	0.05%	2723.95	49.43	2723.95	49.45	0.04%
2734.72	41.93	2734.72	41.95	0.05%	2734.72	49.87	2734.72	49.89	0.04%
2745.48	41.73	2745.48	41.75	0.05%	2745.48	49.90	2745.48	49.92	0.04%
2756.25	39.91	2756.25	39.93	0.05%	2756.25	49.12	2756.25	49.14	0.04%
2767.02	36.95	2767.02	36.97	0.05%	2767.02	48.81	2767.02	48.83	0.04%
2777.78	34.96	2777.78	34.98	0.06%	2777.78	49.55	2777.78	49.57	0.04%
2788.55	35.64	2788.55	35.66	0.06%	2788.55	49.70	2788.55	49.72	0.04%
2799.32	34.07	2799.32	34.09	0.06%	2799.32	48.90	2799.32	48.92	0.04%
2810.08	33.99	2810.08	34.01	0.06%	2810.08	49.14	2810.08	49.16	0.04%
2820.85	33.34	2820.85	33.36	0.06%	2820.85	49.47	2820.85	49.49	0.04%
2831.62	31.34	2831.62	31.36	0.06%	2831.62	49.38	2831.62	49.40	0.04%
2842.38	31.90	2842.38	31.92	0.06%	2842.38	49.80	2842.38	49.82	0.04%
2853.15	35.61	2853.15	35.63	0.06%	2853.15	49.26	2853.15	49.29	0.06%
2863.92	37.18	2863.92	37.20	0.05%	2863.92	48.69	2863.92	48.71	0.04%
2874.68	35.37	2874.68	35.39	0.06%	2874.68	49.56	2874.68	49.58	0.04%
2885.45	32.85	2885.45	32.87	0.06%	2885.45	49.35	2885.45	49.37	0.04%
2896.22	31.58	2896.22	31.60	0.06%	2896.22	48.66	2896.22	48.68	0.04%
2906.98	30.91	2906.98	30.93	0.06%	2906.98	48.33	2906.98	48.35	0.04%
2917.75	32.30	2917.75	32.32	0.06%	2917.75	49.20	2917.75	49.22	0.04%
2928.52	30.02	2928.52	30.05	0.10%	2928.52	49.61	2928.52	49.63	0.04%
2939.28	29.15	2939.28	29.17	0.07%	2939.28	48.99	2939.28	49.01	0.04%
2950.05	31.55	2950.05	31.57	0.06%	2950.05	49.54	2950.05	49.56	0.04%
2960.82	31.13	2960.82	31.15	0.06%	2960.82	48.92	2960.82	48.94	0.04%
2971.58	33.12	2971.58	33.14	0.06%	2971.58	49.12	2971.58	49.14	0.04%
2982.35	34.50	2982.35	34.52	0.06%	2982.35	49.46	2982.35	49.48	0.04%
2993.12	31.41	2993.12	31.43	0.06%	2993.12	49.27	2993.12	49.29	0.04%
3003.88	28.28	3003.88	28.30	0.07%	3003.88	49.31	3003.88	49.34	0.06%
3014.65	28.12	3014.65	28.14	0.07%	3014.65	48.67	3014.65	48.69	0.04%
3025.42	29.66	3025.42	29.68	0.07%	3025.42	49.10	3025.42	49.12	0.04%
3036.18	29.21	3036.18	29.23	0.07%	3036.18	49.16	3036.18	49.18	0.04%
3046.95	26.57	3046.95	26.59	0.08%	3046.95	48.26	3046.95	48.28	0.04%
3057.71	27.80	3057.71	27.82	0.07%	3057.71	49.17	3057.71	49.19	0.04%
3068.48	28.50	3068.48	28.52	0.07%	3068.48	49.27	3068.48	49.29	0.04%
3079.25	31.56	3079.25	31.58	0.06%	3079.25	50.24	3079.25	50.26	0.04%

3090.01	35.01	3090.01	35.03	0.06%	3090.01	49.39	3090.01	49.41	0.04%
3100.78	33.82	3100.78	33.84	0.06%	3100.78	49.50	3100.78	49.52	0.04%
3111.55	31.10	3111.55	31.12	0.06%	3111.55	48.87	3111.55	48.89	0.04%
3122.31	30.54	3122.31	30.56	0.07%	3122.31	49.78	3122.31	49.80	0.04%
3133.08	27.42	3133.08	27.44	0.07%	3133.08	50.58	3133.08	50.60	0.04%
3143.85	30.14	3143.85	30.16	0.07%	3143.85	49.04	3143.85	49.06	0.04%
3154.61	30.18	3154.61	30.20	0.07%	3154.61	48.81	3154.61	48.83	0.04%
3165.38	28.03	3165.38	28.05	0.07%	3165.38	48.56	3165.38	48.58	0.04%
3176.15	30.76	3176.15	30.78	0.06%	3176.15	48.86	3176.15	48.88	0.04%
3186.91	30.54	3186.91	30.56	0.07%	3186.91	49.73	3186.91	49.75	0.04%
3197.68	33.11	3197.68	33.13	0.06%	3197.68	49.56	3197.68	49.58	0.04%
3208.45	37.18	3208.45	37.20	0.05%	3208.45	48.39	3208.45	48.41	0.04%
3219.21	36.97	3219.21	36.99	0.05%	3219.21	49.55	3219.21	49.57	0.04%
3229.98	36.73	3229.98	36.75	0.05%	3229.98	49.62	3229.98	49.64	0.04%
3240.75	35.21	3240.75	35.23	0.06%	3240.75	49.36	3240.75	49.38	0.04%
3251.51	31.99	3251.51	32.01	0.06%	3251.51	48.87	3251.51	48.89	0.04%
3262.28	30.98	3262.28	31.00	0.06%	3262.28	49.11	3262.28	49.13	0.04%
3273.05	30.82	3273.05	30.84	0.06%	3273.05	49.50	3273.05	49.52	0.04%
3283.81	31.22	3283.81	31.24	0.06%	3283.81	50.05	3283.81	50.07	0.04%
3294.58	32.39	3294.58	32.41	0.06%	3294.58	50.18	3294.58	50.20	0.04%
3305.35	35.62	3305.35	35.64	0.06%	3305.35	49.11	3305.35	49.14	0.06%
3316.11	34.34	3316.11	34.36	0.06%	3316.11	48.78	3316.11	48.80	0.04%
3326.88	36.95	3326.88	36.97	0.05%	3326.88	49.24	3326.88	49.26	0.04%
3337.65	39.55	3337.65	39.57	0.05%	3337.65	49.12	3337.65	49.14	0.04%
3348.41	40.62	3348.41	40.65	0.07%	3348.41	49.47	3348.41	49.49	0.04%
3359.18	39.04	3359.18	39.06	0.05%	3359.18	49.94	3359.18	49.96	0.04%
3369.95	34.47	3369.95	34.50	0.09%	3369.95	49.42	3369.95	49.44	0.04%
3380.71	31.23	3380.71	31.25	0.06%	3380.71	48.99	3380.71	49.01	0.04%
3391.48	30.81	3391.48	30.84	0.10%	3391.48	48.51	3391.48	48.53	0.04%
3402.25	32.62	3402.25	32.64	0.06%	3402.25	49.67	3402.25	49.69	0.04%
3413.01	34.31	3413.01	34.33	0.06%	3413.01	49.45	3413.01	49.47	0.04%
3423.78	36.38	3423.78	36.40	0.05%	3423.78	48.62	3423.78	48.64	0.04%
3434.55	35.26	3434.55	35.28	0.06%	3434.55	49.17	3434.55	49.19	0.04%
3445.31	33.78	3445.31	33.80	0.06%	3445.31	49.14	3445.31	49.16	0.04%
3456.08	36.58	3456.08	36.60	0.05%	3456.08	48.91	3456.08	48.93	0.04%
3466.85	39.10	3466.85	39.12	0.05%	3466.85	48.34	3466.85	48.36	0.04%
3477.61	37.30	3477.61	37.32	0.05%	3477.61	48.34	3477.61	48.36	0.04%
3488.38	34.24	3488.38	34.26	0.06%	3488.38	48.69	3488.38	48.71	0.04%
3499.15	32.58	3499.15	32.60	0.06%	3499.15	49.12	3499.15	49.15	0.06%
3509.91	34.23	3509.91	34.25	0.06%	3509.91	49.56	3509.91	49.58	0.04%
3520.68	33.93	3520.68	33.95	0.06%	3520.68	48.91	3520.68	48.93	0.04%
3531.45	34.17	3531.45	34.19	0.06%	3531.45	49.02	3531.45	49.04	0.04%
3542.21	34.68	3542.21	34.70	0.06%	3542.21	49.47	3542.21	49.49	0.04%
3552.98	31.89	3552.98	31.91	0.06%	3552.98	49.12	3552.98	49.14	0.04%
3563.75	29.86	3563.75	29.88	0.07%	3563.75	48.78	3563.75	48.80	0.04%
3574.51	31.85	3574.51	31.88	0.09%	3574.51	48.75	3574.51	48.77	0.04%
3585.28	33.20	3585.28	33.22	0.06%	3585.28	49.07	3585.28	49.09	0.04%
3596.04	35.01	3596.04	35.03	0.06%	3596.04	49.34	3596.04	49.36	0.04%
3606.81	31.03	3606.81	31.05	0.06%	3606.81	49.96	3606.81	49.98	0.04%
3617.58	26.03	3617.58	26.05	0.08%	3617.58	49.49	3617.58	49.51	0.04%
3628.34	29.04	3628.34	29.06	0.07%	3628.34	48.86	3628.34	48.88	0.04%
3639.11	30.11	3639.11	30.13	0.07%	3639.11	49.46	3639.11	49.48	0.04%
3649.88	30.51	3649.88	30.53	0.07%	3649.88	49.13	3649.88	49.15	0.04%
3660.64	27.26	3660.64	27.29	0.11%	3660.64	48.46	3660.64	48.48	0.04%
3671.41	27.16	3671.41	27.19	0.11%	3671.41	49.23	3671.41	49.25	0.04%
3682.18	28.78	3682.18	28.80	0.07%	3682.18	49.85	3682.18	49.88	0.06%
3692.94	27.80	3692.94	27.82	0.07%	3692.94	49.16	3692.94	49.18	0.04%
3703.71	25.92	3703.71	25.95	0.12%	3703.71	49.10	3703.71	49.12	0.04%
3714.48	26.41	3714.48	26.43	0.08%	3714.48	49.97	3714.48	49.99	0.04%
3725.24	28.35	3725.24	28.37	0.07%	3725.24	50.06	3725.24	50.08	0.04%

3736.01	27.88	3736.01	27.90	0.07%	3736.01	49.26	3736.01	49.28	0.04%
3746.78	25.04	3746.78	25.06	0.08%	3746.78	49.05	3746.78	49.07	0.04%
3757.54	23.04	3757.54	23.06	0.09%	3757.54	48.78	3757.54	48.80	0.04%
3768.31	20.15	3768.31	20.17	0.10%	3768.31	48.02	3768.31	48.04	0.04%
3779.08	20.25	3779.08	20.27	0.10%	3779.08	49.33	3779.08	49.35	0.04%
3789.84	23.53	3789.84	23.55	0.08%	3789.84	49.52	3789.84	49.54	0.04%
3800.61	25.03	3800.61	25.06	0.12%	3800.61	49.33	3800.61	49.35	0.04%
3811.38	24.03	3811.38	24.05	0.08%	3811.38	49.05	3811.38	49.07	0.04%
3822.14	23.65	3822.14	23.68	0.13%	3822.14	49.21	3822.14	49.23	0.04%
3832.91	23.40	3832.91	23.43	0.13%	3832.91	47.93	3832.91	47.95	0.04%
3843.68	22.46	3843.68	22.48	0.09%	3843.68	47.81	3843.68	47.83	0.04%
3854.44	25.47	3854.44	25.49	0.08%	3854.44	48.82	3854.44	48.84	0.04%
3865.21	26.76	3865.21	26.78	0.07%	3865.21	49.50	3865.21	49.53	0.06%
3875.98	23.06	3875.98	23.08	0.09%	3875.98	47.90	3875.98	47.93	0.06%
3886.74	20.57	3886.74	20.59	0.10%	3886.74	49.48	3886.74	49.50	0.04%
3897.51	20.29	3897.51	20.31	0.10%	3897.51	48.94	3897.51	48.96	0.04%
3908.28	19.96	3908.28	19.98	0.10%	3908.28	49.34	3908.28	49.36	0.04%
3919.04	21.92	3919.04	21.94	0.09%	3919.04	49.43	3919.04	49.45	0.04%
3929.81	22.62	3929.81	22.64	0.09%	3929.81	48.79	3929.81	48.81	0.04%
3940.58	21.45	3940.58	21.47	0.09%	3940.58	48.55	3940.58	48.57	0.04%
3951.34	19.87	3951.34	19.89	0.10%	3951.34	48.33	3951.34	48.35	0.04%
3962.11	20.68	3962.11	20.70	0.10%	3962.11	48.35	3962.11	48.37	0.04%
3972.88	19.40	3972.88	19.42	0.10%	3972.88	48.46	3972.88	48.48	0.04%
3983.64	20.07	3983.64	20.09	0.10%	3983.64	49.03	3983.64	49.05	0.04%
3994.41	22.01	3994.41	22.03	0.09%	3994.41	48.83	3994.41	48.85	0.04%
4005.18	20.30	4005.18	20.32	0.10%	4005.18	47.91	4005.18	47.93	0.04%
4015.94	20.47	4015.94	20.49	0.10%	4015.94	48.42	4015.94	48.44	0.04%
4026.71	15.78	4026.71	15.81	0.19%	4026.71	47.79	4026.71	47.82	0.06%
4037.48	17.00	4037.48	17.02	0.12%	4037.48	48.06	4037.48	48.08	0.04%
4048.24	17.65	4048.24	17.67	0.11%	4048.24	47.93	4048.24	47.95	0.04%
4059.01	18.28	4059.01	18.30	0.11%	4059.01	47.67	4059.01	47.69	0.04%
4069.78	19.28	4069.78	19.30	0.10%	4069.78	47.90	4069.78	47.92	0.04%
4080.54	18.71	4080.54	18.73	0.11%	4080.54	47.64	4080.54	47.67	0.06%
4091.31	19.33	4091.31	19.35	0.10%	4091.31	47.68	4091.31	47.70	0.04%
4102.08	14.79	4102.08	14.81	0.14%	4102.08	47.13	4102.08	47.15	0.04%
4112.84	13.20	4112.84	13.22	0.15%	4112.84	46.23	4112.84	46.25	0.04%
4123.61	17.71	4123.61	17.73	0.11%	4123.61	46.54	4123.61	46.56	0.04%
4134.38	17.14	4134.38	17.16	0.12%	4134.38	47.30	4134.38	47.33	0.06%
4145.14	14.84	4145.14	14.86	0.13%	4145.14	47.08	4145.14	47.10	0.04%
4155.91	14.24	4155.91	14.26	0.14%	4155.91	46.19	4155.91	46.22	0.06%
4166.67	15.28	4166.67	15.30	0.13%	4166.67	45.92	4166.67	45.94	0.04%
4177.44	15.45	4177.44	15.47	0.13%	4177.44	45.92	4177.44	45.94	0.04%
4188.21	14.19	4188.21	14.21	0.14%	4188.21	46.53	4188.21	46.56	0.06%
4198.97	14.23	4198.97	14.25	0.14%	4198.97	46.74	4198.97	46.76	0.04%
4209.74	12.92	4209.74	12.94	0.15%	4209.74	45.60	4209.74	45.62	0.04%
4220.51	12.82	4220.51	12.84	0.16%	4220.51	44.88	4220.51	44.90	0.04%
4231.27	11.52	4231.27	11.54	0.17%	4231.27	44.87	4231.27	44.89	0.04%
4242.04	11.35	4242.04	11.37	0.18%	4242.04	44.33	4242.04	44.35	0.05%
4252.81	13.78	4252.81	13.80	0.14%	4252.81	44.49	4252.81	44.52	0.07%
4263.57	13.15	4263.57	13.17	0.15%	4263.57	44.88	4263.57	44.90	0.04%
4274.34	12.30	4274.34	12.32	0.16%	4274.34	43.85	4274.34	43.87	0.05%
4285.11	14.10	4285.11	14.12	0.14%	4285.11	44.08	4285.11	44.10	0.05%
4295.87	11.85	4295.87	11.87	0.17%	4295.87	43.65	4295.87	43.67	0.05%
4306.64	11.69	4306.64	11.71	0.17%	4306.64	42.61	4306.64	42.63	0.05%

Appendix D MATLAB Source Code for Stationary Loudness

Appendix E Test Data for Stationary Loudness

Frequency (Hz)	1	2	3	4
25.00	-29.18	4.94	-5.97	4.87
31.50	-27.98	5.82	-5.45	5.86
40.00	-27.60	6.69	-5.03	6.86
50.00	-26.13	7.82	-3.99	7.86
3.00	-25.22	8.81	-2.92	8.86
80.00	-24.26	9.82	-1.94	9.85
100.00	-23.24	10.81	-1.03	10.90
125.00	-22.11	11.83	-0.13	11.91
160.00	-21.08	12.83	0.86	12.92
200.00	-19.93	13.88	1.85	13.97
250.00	-18.71	14.93	2.80	15.04
315.00	-17.37	15.99	3.77	16.12
400.00	-15.72	17.17	4.75	17.33
500.00	-13.65	18.38	5.69	18.58
630.00	-10.59	19.72	6.57	19.96
800.00	18.62	21.33	7.42	23.29
1000.00	39.98	23.38	8.21	40.08
1250.00	18.32	26.41	8.97	27.55
1600.00	-0.37	58.67	9.86	44.47
2000.00	-11.77	80.05	11.12	72.49
2500.00	-20.02	58.39	13.24	75.62
3150.00	-24.92	39.90	16.15	48.04
4000.00	-29.36	28.11	19.10	67.77
5000.00	-33.02	19.08	21.22	68.62
6300.00	-36.03	13.40	58.18	45.90
8000.00	-38.80	8.10	80.13	33.20
10000.00	-41.21	3.58	59.05	25.68
12500.00	-43.04	-0.07	40.22	24.39
16000.00	-46.71	-5.68	28.13	21.19
20000.00	-70.32	-22.10	18.01	-1.72

Table E1. Shows CPB analysis data obtained from different pure tones.

Appendix F Test Results of MATLAB

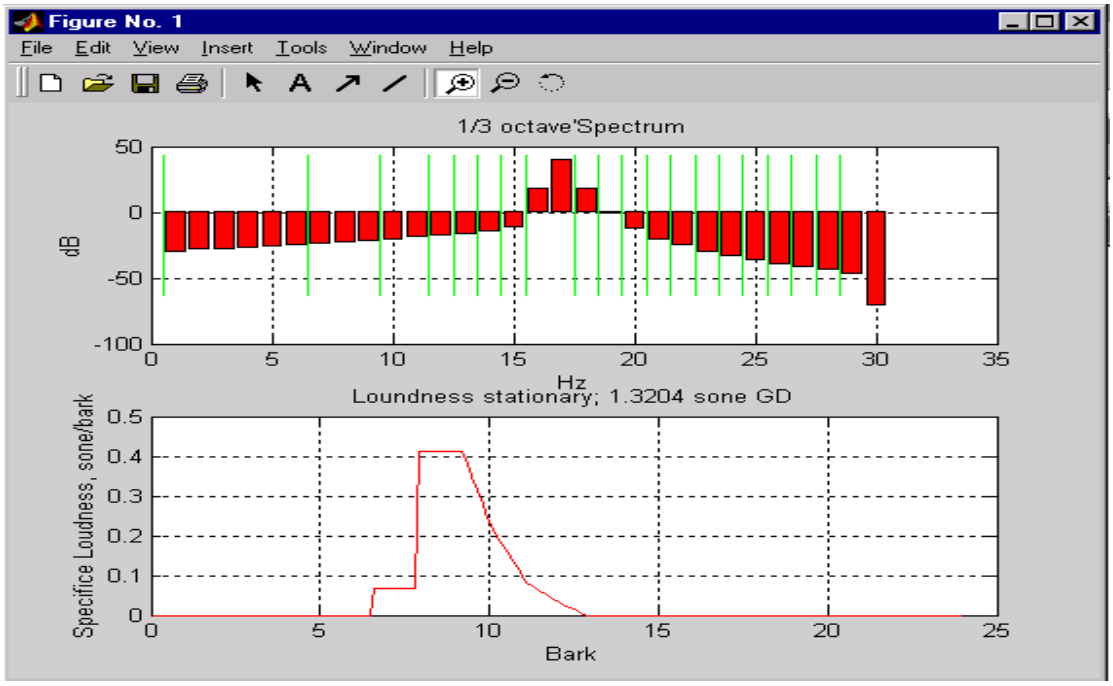

Figure F1. Shows Pure Tone 1 kHz 40 dB in diffuse field.

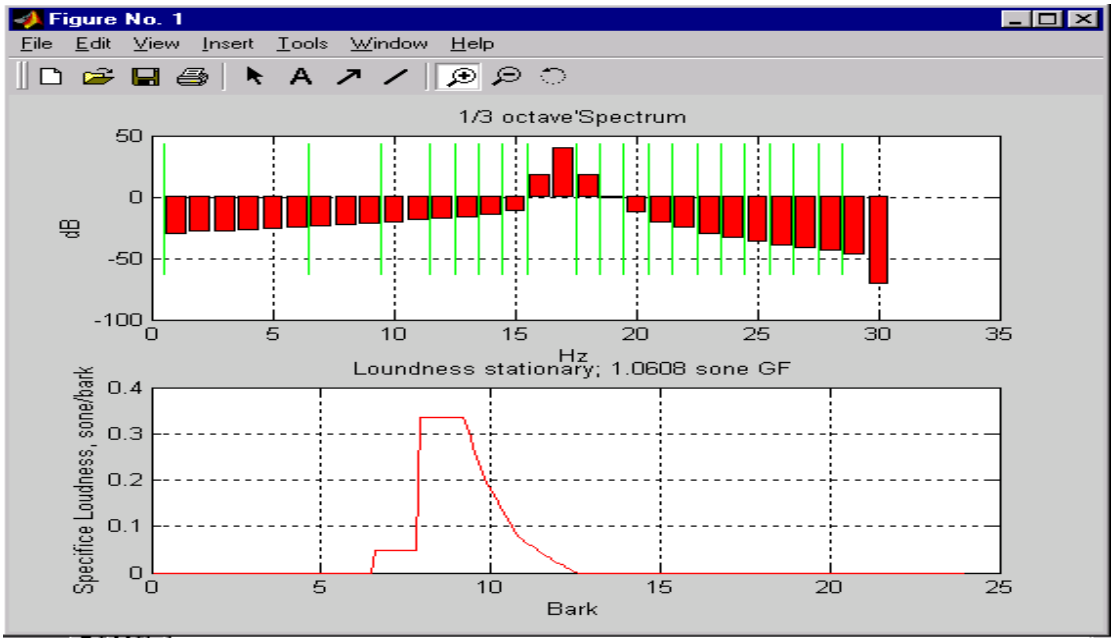

Figure F2. Shows Pure Tone 1 kHz 40 dB in free field.

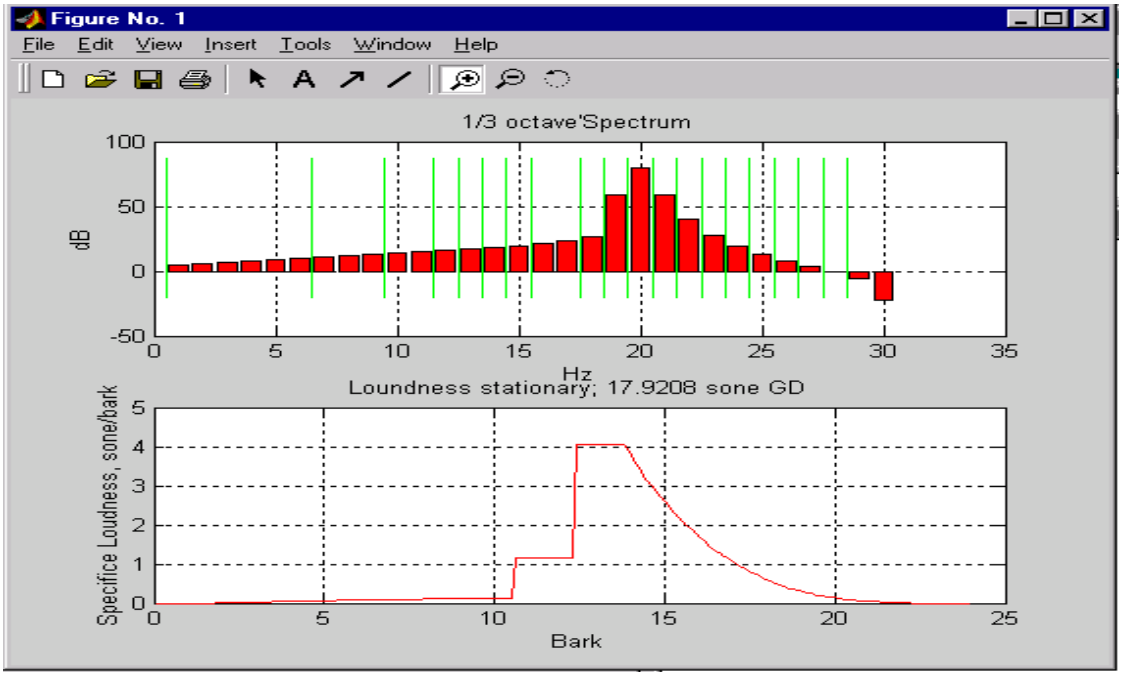

Figure F3. Shows 2 kHz 80dB Pure Tone in diffuse field

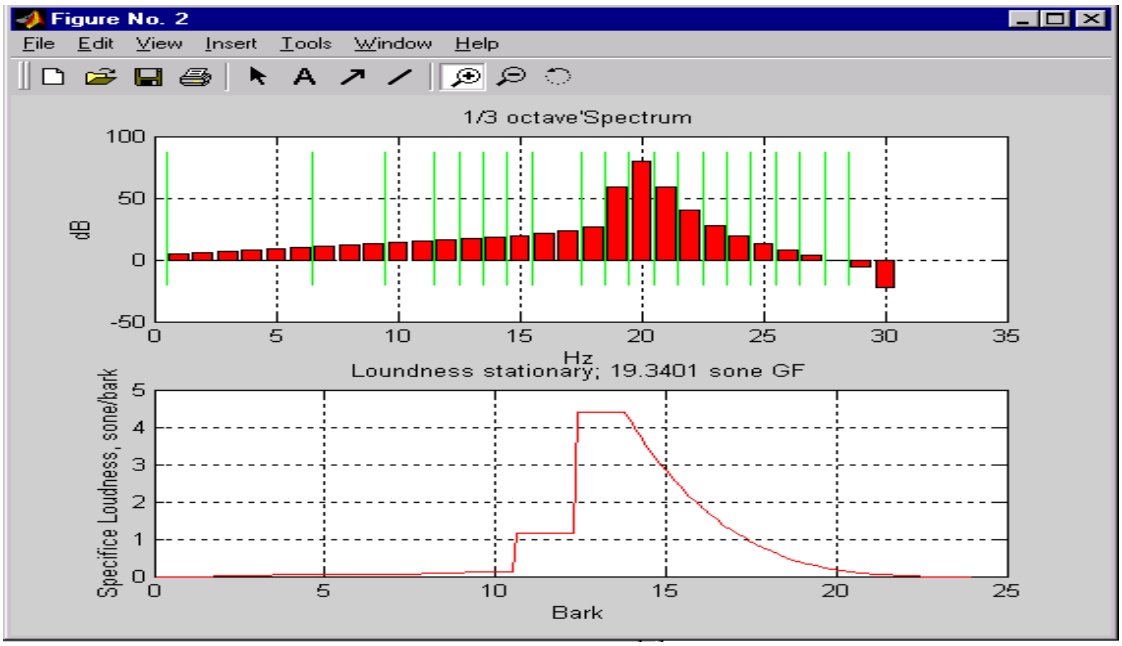

Figure F4. shows 2 kHz 80dB Pure Tone in free field

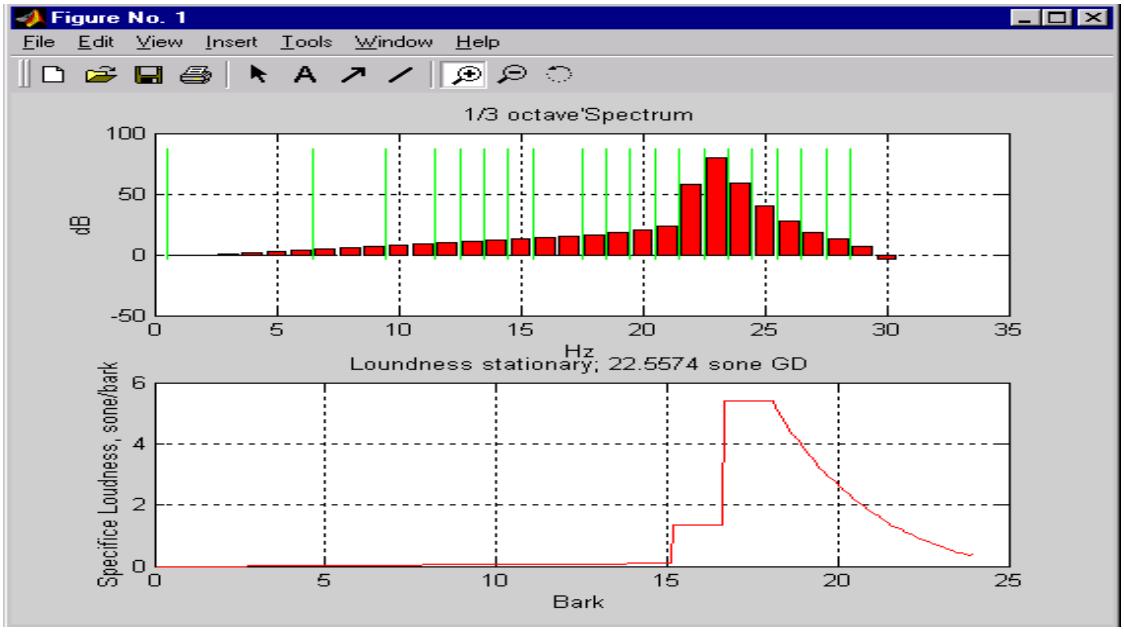

Figure F5. Shows 4 kHz 80dB Pure Tone in diffuse field

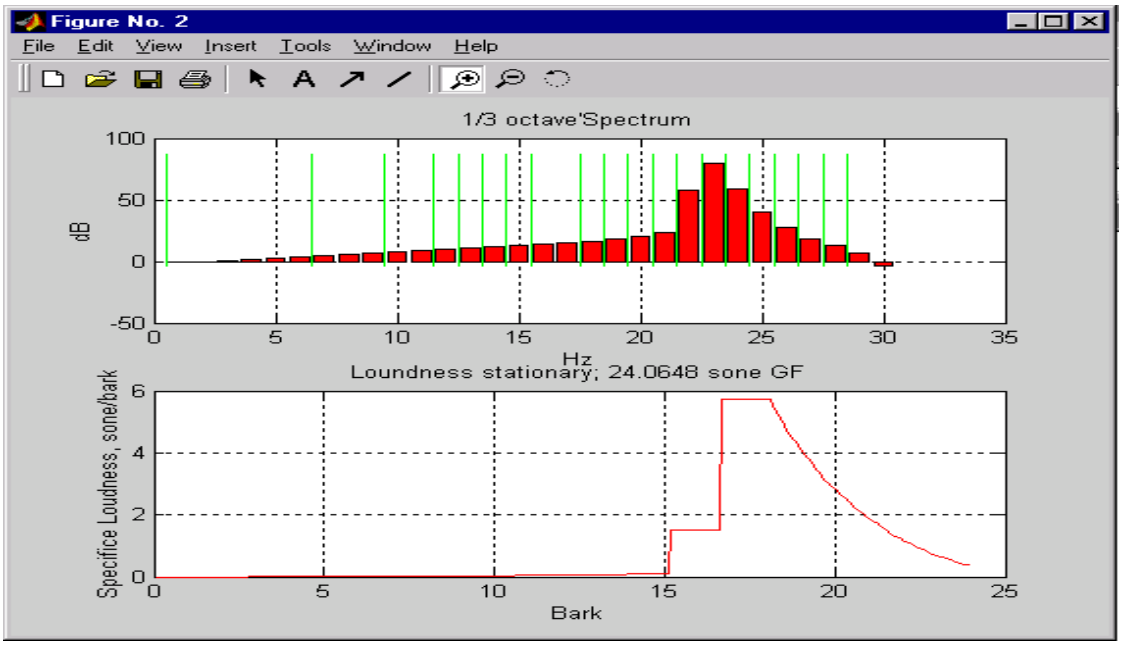

Figure F6. Shows 4 kHz 80dB Pure Tone in free field

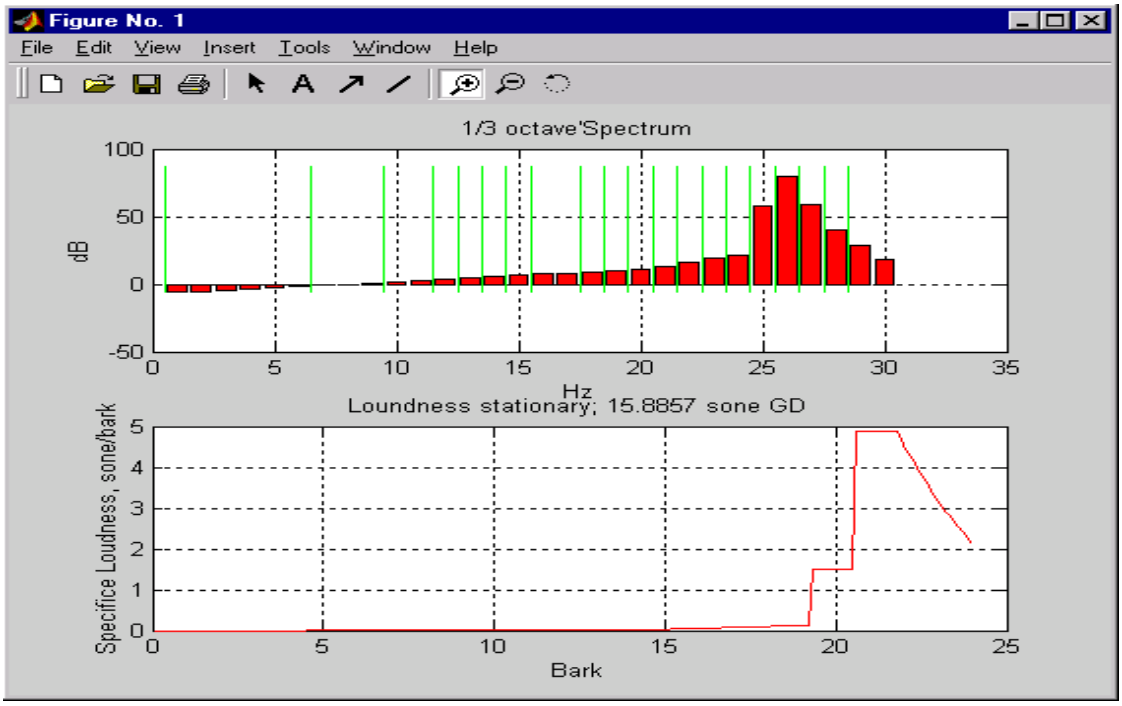

Figure F7. Shows 8 kHz 80dB Pure Tone in diffuse field

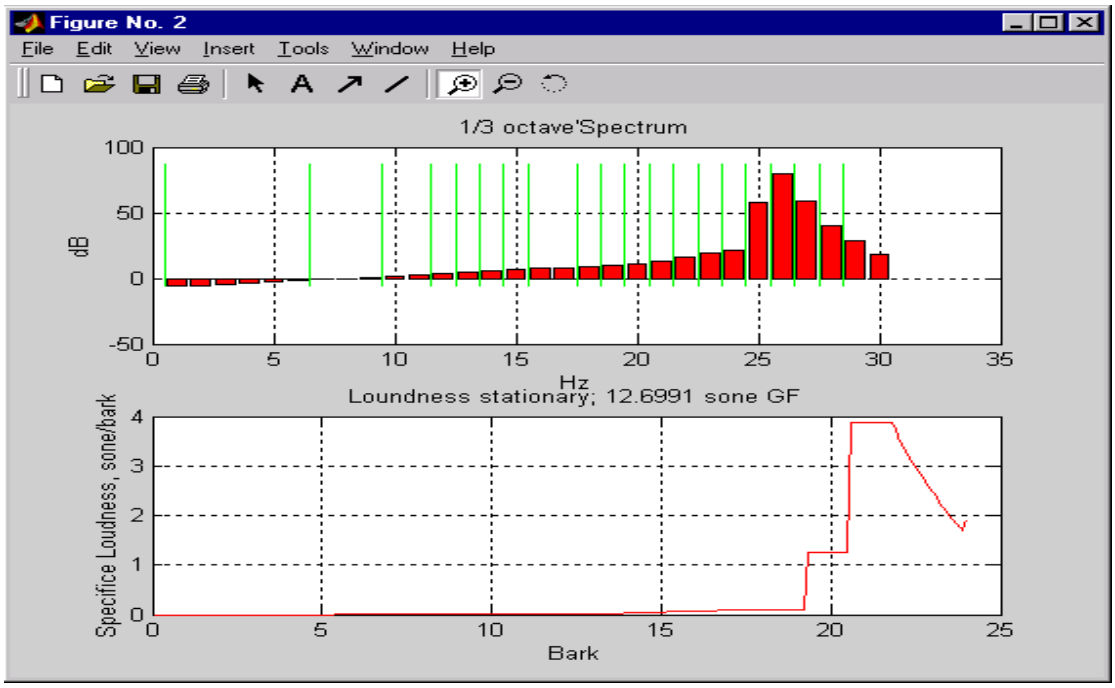

Figure F8. Shows 8 kHz 80dB Pure Tone in free field

Appendix G Algorithm for Extraction of Pitch and Pitch Salience from Complex Tonal Signals

Appendix H Pitch of Complex Signals according to Virtual-pitch Theory: Tests, examples, and predictions.

Appendix I International Standard ISO 532 Acoustics – Method for Calculating Loudness Level

Appendix J DIN 45 631 - Procedure for Calculating Loudness Level and Loudness

Appendix K An Examination of Aures's Model of Tonality

Appendix L Lecture Notes 'Introduction to Sound Quality' by B&K A/S

Appendix M Lecture Notes 'Psychoacoustics – A Qualitative Description' by B&K A/S

Filename: Tonality_Final_Thesis
Directory: E:\ThesisCD
Template: C:\Documents and Settings\fkc\Application Data\Microsoft\Templates\Normal.dot
Title: Sound Quality User-defined Cursor Reading Control
Subject:
Author: zzhong
Keywords:
Comments:
Creation Date: 09-03-2003 16:17
Change Number: 4
Last Saved On: 09-03-2003 16:45
Last Saved By: zzhong
Total Editing Time: 13 Minutes
Last Printed On: 11-03-2003 11:16
As of Last Complete Printing
Number of Pages: 110
Number of Words: 28.621 (approx.)
Number of Characters: 163.145 (approx.)