

Summer 2010

An Application of Spectral Translation and Spectral Envelope Extrapolation for High-frequency Bandwidth Extension of Generic Audio Signals

Alexei Kontsevoi

Follow this and additional works at: <https://dsc.duq.edu/etd>

Recommended Citation

Kontsevoi, A. (2010). An Application of Spectral Translation and Spectral Envelope Extrapolation for High-frequency Bandwidth Extension of Generic Audio Signals (Master's thesis, Duquesne University). Retrieved from <https://dsc.duq.edu/etd/769>

This Immediate Access is brought to you for free and open access by Duquesne Scholarship Collection. It has been accepted for inclusion in Electronic Theses and Dissertations by an authorized administrator of Duquesne Scholarship Collection. For more information, please contact phillips@duq.edu.

AN APPLICATION OF SPECTRAL TRANSLATION AND SPECTRAL ENVELOPE
EXTRAPOLATION FOR HIGH-FREQUENCY BANDWIDTH EXTENSION OF
GENERIC AUDIO SIGNALS

A Thesis

Submitted to the McAnulty College and Graduate School of Liberal Arts

Duquesne University

In partial fulfillment of the requirements for
the degree of Master of Science

By

Alexei Kontsevoi

August 2010

AN APPLICATION OF SPECTRAL TRANSLATION AND SPECTRAL ENVELOPE
EXTRAPOLATION FOR HIGH-FREQUENCY BANDWIDTH EXTENSION OF
GENERIC AUDIO SIGNALS

By
Alexei Kontsevoi

Approved May 5, 2010

Stacey Levine, Ph.D.
Associate Professor of Mathematics
Thesis Director

Carl Toews, Ph.D.
Assistant Professor of Mathematics
Committee Member

Donald Simon, Ph.D.
Associate Professor of Computer Science
Director of Graduate Studies

Jeffrey Jackson, Ph.D.
Professor of Computer Science
Chair, Department of Mathematics and
Computer Science

Christopher M. Duncan, Ph.D.
Dean, McAnulty College and
Graduate School of Liberal Arts

ABSTRACT

AN APPLICATION OF SPECTRAL TRANSLATION AND SPECTRAL ENVELOPE EXTRAPOLATION FOR HIGH-FREQUENCY BANDWIDTH EXTENSION OF GENERIC AUDIO SIGNALS

By

Alexei Kontsevoi

August 2010

Thesis supervised by Dr. Stacey Levine

The scope of this work is to introduce a conceptually simple yet effective algorithm for blind high-frequency bandwidth extension of audio signals, a means of improving perceptual quality for sound which has been previously low-pass filtered or downsampled (typically due to storage considerations). The algorithm combines an application of the modulation theorem for discrete Fourier transform to regenerate the missing high-frequency end of the signal spectrum with a linear-regression-driven approach to shape the spectral envelope for the regenerated band. The results are graphically and acoustically compared to those obtained with existing audio restoration software for a variety of input signals. The source code and Windows binaries of the resulting algorithm implementation are also included.

ACKNOWLEDGEMENT

The author would like to thank the Department of Mathematics and Computer Science faculty for patience and encouragement to render the present work as a thesis, and the following online resources for providing materials used to complete the project:

- [TheForce.net](#) and Vidizen Films for Broken Allegiance, a fan-made Star Wars-themed motion picture;
- [Panic Struck Productions](#) for Revelations, another fan-made motion picture set in the Star Wars universe;
- [MusOpen.com](#) and US Navy Band for their rendition of Tchaikovsky's Symphony No. 4 Finale;
- [Dance-Industries.com](#) and Maurice Corbach (Dutch Dance Connection) for Ocean, a free trance track with enough variation in percussion to be one of the test cases for the project;
- [Q Software Solutions](#) and Jacob Navia for LCC-Win32, a Windows C compiler free for non-commercial use;
- [Diamond Cut Productions](#) for Diamond Cut Audio Restoration Tools, the most versatile audio restoration software known to the author;
- Matteo Frigo and Steven G. Johnson for [FFTW](#), a free and open-source Fast Fourier Transform library for C and C++.
- [Xiph.Org Foundation](#) and Josh Coalson for [FLAC](#), a free and open-source lossless audio codec used to compress the audio samples included with this paper.

TABLE OF CONTENTS

Abstract	iv
Acknowledgement	v
List of Tables	viii
List of Figures	ix
1 Introduction	1
2 Background	2
2.1 Discrete Fourier Transform	2
2.2 Spectrogram	4
2.3 Spectral Averaging, Spectral Plot, and Spectral Envelope	6
3 Existing Methods	8
3.1 Equalization	9
3.2 Harmonic Excitation	9
3.3 Available Research	13
3.4 Nearest-neighbor Resampling	14
4 Implementation Details	15
4.1 Spectral Translation	16
4.2 Spectral Envelope Estimation	22
4.3 Further Improvements	29
5 Comparative Results	32
5.1 Parameters	33

5.2	Electronic Music	34
5.3	Symphonic Music	37
5.4	Speech	40
5.5	Mixed Audio	43
6	Conclusion and Potential Improvements	46
	References	49
A	Source Code and Windows Binaries	50

LIST OF TABLES

1	Parameters used for VVA and BandR processing, with BandR statistics obtained	34
2	Electronic music test signals	37
3	Symphonic music test signals	37
4	Speech test signals	40
5	Mixed audio test signals	43
6	Attached program files	50

LIST OF FIGURES

1	Example of the spectrogram	5
2	Example of the spectral plot of a wide-band audio signal	7
3	Example of the spectral plot of a low-pass filtered audio signal	8
4	Spectral plots of the band-limited signal in Figure 3, processed with a harmonic exciter	11
5	Spectral plot of the signal in Figure 3 following an application of raw spectral translation	18
6	Possible results of fitting a least-squares line through log-scale spectral line magnitudes in the short-term power spectrum	23
7	Histogram of the average log-scale DFT magnitude distribution at a single frequency line	27
8	Spectrograms of the electronic music test signal	35
9	Long-term power spectra of the electronic music test signal	36
10	Spectrograms of the symphonic music test signal	38
11	Long-term power spectra of the symphonic music test signal	39
12	Spectrograms of the speech test signal	41
13	Long-term power spectra of the speech test signal	42
14	Spectrograms of the mixed audio test signal	44
15	Long-term power spectra of the mixed audio test signal	45

1 Introduction

With the advent of digital media and the wide availability of the Internet which promotes its distribution, it is not uncommon to see the circulation of poor quality audio material. One of the most common types of quality degradation is the low-pass filtering of a given signal. Improperly chosen filters during mastering, poor frequency responses of the underlying analog equipment, and especially low encoding bit rates for lossy audio compression techniques such as various flavors of MPEG (owing to transfer rate or storage space constraints) are among the most frequent causes. In audio material that has suffered from low-pass filtering, practically all high-frequency information beyond a certain sharp cutoff (e.g. 11 kHz) is completely obliterated. The resulting audio sounds characteristically muffled. Research in this area is scarce (see [LL03], [LR04, §5], and [BN07]), and worse yet, no commercially or freely available software (to author's knowledge) exists to effectively deal with the problem. This circumstance prompted the author to develop a freely available program for dealing with such degraded audio. The scope of this paper is to summarize the existing blind high-frequency bandwidth extension methods for generic audio, and to explain and provide a working implementation of a new algorithm for dealing with this problem, which offers a perceptual improvement over widely available methods used in modern audio restoration software.

This work is structured as follows. §2 focuses on the *discrete Fourier transform*, the primary analysis and processing tool used in just about any digital signal processing software, and introduces the essential frequency analysis methods used in the remainder of the paper. §3 surveys the algorithms and implementations which already exist to deal with the problem of high-frequency bandwidth extension. §4 provides a detailed description of the proposed method, while §5 compares its effectiveness to that of other available methods. §6 provides suggestions for further work to improve the proposed algorithm. Finally, the attachments in Appendix A contain the source code and Windows binaries of the algorithm's implementation.

2 Background

2.1 Discrete Fourier Transform

Digital audio tracks, in their basic form, are stored in computers as sequences of samples representing values of the sound wave's amplitude (or, in their electrical representation, voltage) at successive and evenly spaced points in time. Such sequences are said to be in the *time domain*. The human ear, however, does not perceive the amplitudes of the signal at individual points in time; because of the way it is structured, it perceives *frequencies* (tones) as they vary with time (see [LR04, §1.4] for an introduction to psychoacoustics and further references). We would therefore like to have analysis tools and processing algorithms mimic this property of the human ear.

The Fourier and related transforms are probably the most widely used techniques to accomplish that. The discrete Fourier transform (DFT) is a function \mathcal{F} that takes an N -point complex-valued finite sequence $\{x_k\}$ (where $0 \leq k \leq N - 1$) and returns another N -point complex-valued sequence $\{X_j\}$ ($0 \leq j \leq N - 1$) obtained from $\{x_k\}$ as follows:

$$\begin{aligned} X_j = \mathcal{F}(\{x_k\})_j &= \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} x_k \left(\cos\left(\frac{2\pi j k}{N}\right) - i \sin\left(\frac{2\pi j k}{N}\right) \right) \\ &= \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} x_k e^{-\frac{2\pi i j k}{N}}. \end{aligned} \tag{1}$$

The DFT is invertible; that is, there exists an inverse transform \mathcal{F}^{-1} which takes the sequence $\{X_j\}$ and returns the original sequence $\{x_k\}$. The inverse DFT is defined as

$$\begin{aligned} x_k = \mathcal{F}^{-1}(\{X_j\})_k &= \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} X_j \left(\cos\left(\frac{2\pi k j}{N}\right) + i \sin\left(\frac{2\pi k j}{N}\right) \right) \\ &= \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} X_j e^{\frac{2\pi i k j}{N}}. \end{aligned} \tag{2}$$

Essentially, the effect of the DFT is to represent the sequence $\{x_k\}$ as a sum of periodic

waves with frequencies ranging from 0 to $N - 1$ periods per sequence; the values of $\{X_j\}$ represent the amplitudes of these periodic waves at each frequency, and are necessary to reconstruct the original signal $\{x_k\}$. There are two equivalent forms for each transform given above; the former (*rectangular*) form represents the signal as a sum of cosine and sine waves of each frequency j , where the amplitudes of cosine and sine waves are stored in $\Re X_j$ and $\Im X_j$ respectively. The latter (*polar*) form, equivalent to the former by the Euler's formula $e^{i\phi} = \cos \phi + i \sin \phi$, conceptually represents each frequency point j as having a *magnitude* $r_j = \sqrt{(\Re X_j)^2 + (\Im X_j)^2}$ and a *phase (argument)* ϕ_j (an angle, normally constrained between $-\pi$ and π or between 0 and 2π , such that $\Re X_j = r_j \cos \phi_j$ and $\Im X_j = r_j \sin \phi_j$). The magnitude r_j thus represents the amplitude of the cosine wave at each frequency point j , and the phase ϕ_j determines the initial value of that cosine wave at the time point $k = 0$. The DFT is normally computed in its rectangular form for practical reasons, but a change to polar coordinates is often done afterwards because the polar form may be more convenient for signal processing applications, including the one introduced in §4 of this paper.

An important property of the DFT is that if the sequence $\{x_k\}$ is real-valued (as is the case in audio processing), then the output sequence $\{X_j\}$ is *conjugate-symmetric* about its center, that is, for all $1 \leq j \leq \lfloor N/2 \rfloor$, $X_j^* = X_{N-j}$ (where $X_j^* \equiv \Re X_j - i \Im X_j$), and conversely, if for all $1 \leq j \leq \lfloor N/2 \rfloor$, $X_j^* = X_{N-j}$, then the inverse DFT of $\{X_j\}$ is a real-valued sequence. Firstly, this implies that for a DFT of real-valued sequences, the upper portion of $\{X_j\}$ (for $j > \lfloor N/2 \rfloor$) mirrors the lower portion, and therefore needs not be stored for practical applications (the first $\lfloor N/2 \rfloor + 1$ points of $\{X_j\}$ are sufficient to reconstruct $\{x_k\}$ via an inverse DFT). This gives rise to the so-called *real-to-complex* and *complex-to-real* DFT types, which are used extensively in signal processing applications including the one presented here, and some digital signal processing books like [Sm97] rely on this kind of DFT almost exclusively. Essentially, the DFT of N real-valued points returns $\lfloor N/2 \rfloor + 1$ complex-valued points, which are then processed; the inverse DFT is

then taken with the assumption that the upper portion of $\{X_j\}$ would have been adjusted accordingly to mirror the lower portion, resulting in a real-valued sequence. Secondly, this property implies that in a sample with a given number of time-domain points, there are only half as many frequency-domain points, and therefore for a given sampling rate, the maximum meaningful frequency that can be contained in the sequence $\{x_k\}$ is half that sampling rate (the Nyquist-Shannon theorem). For this reason most signal processing software and literature (including this work) limit the frequency axis to half the sampling rate of the source audio (the *Nyquist frequency*) in all spectral analysis tools. This is also the reason why resampling the audio to a lower sampling rate (as is often done to save storage space) introduces the very kind of problem this work is going to address, namely missing high-frequency components and the resulting muffled sound.

Finally, we need to address the normalization factor $1/\sqrt{N}$ which appears in front of the summation in the definitions of both forward and inverse Fourier transforms. Its choice is somewhat arbitrary as long as the product of the factors in (1) and (2) is $1/N$; different sources thus use different conventions [Sh95]. The factor $1/\sqrt{N}$ is convenient because with this choice of normalization, the root mean square magnitude of the source sequence $\{x_k\}$ equals the root mean square magnitude of its Fourier transform $\{X_j\}$, that is,

$$\sum_{k=0}^{N-1} |x_k|^2 = \sum_{j=0}^{N-1} |X_j|^2, \quad (3)$$

a property known as the Parseval's theorem. This allows us to store the members of $\{X_j\}$ in the same power units as those of $\{x_k\}$, the original waveform data.

2.2 Spectrogram

As evident from the above description, a sequence in the time domain can be represented in the frequency domain by calculating its DFT. Calculating the DFT of the entire audio signal, however, is both impractical and of limited use, firstly because the memory

requirements for that may be prohibitive, and secondly because by doing so, all the temporal information in the signal will be hidden in the phase data (which is not easily visualized or analyzed). Because human ear perceives frequencies as their magnitudes change with time, it is often more accurate for analysis purposes to take the DFT over small chunks of the audio signal, each of them N points long, compute the magnitudes of the transformed sequence members X_j (normally referred to as the *spectral lines* of the signal), and plot them against the time and frequency axes either as a surface, or more frequently, as a grayscale or in pseudocolor. Such a plot therefore shows the magnitudes of the cosine waves of different frequencies as they change with time; it is known as the *spectrogram*. Figure 1 shows an example of the spectrogram; the x axis is the time axis,

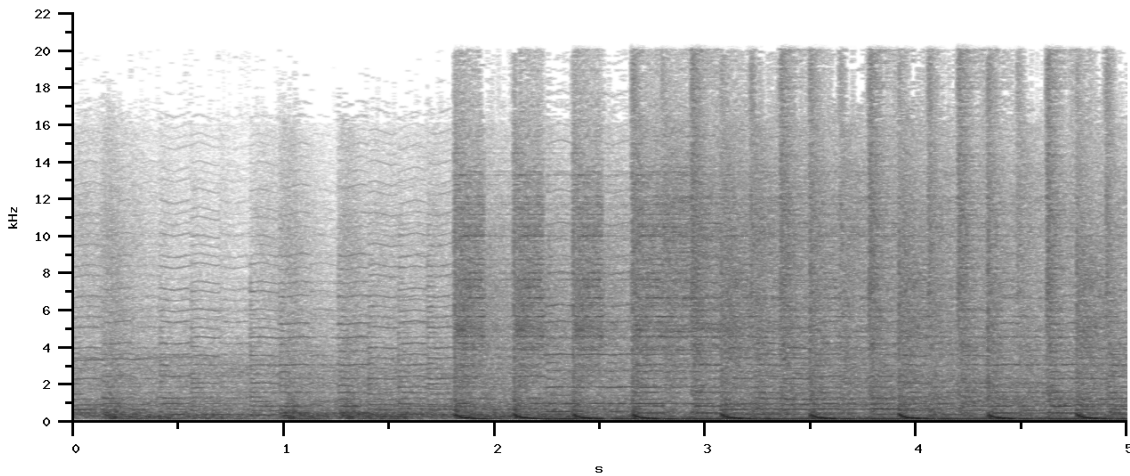


Figure 1: Example of the spectrogram

and the y axis is the frequency axis. Darker regions represent greater magnitudes of the spectral lines, while brighter regions represent lesser magnitudes (white being close to 0). Additionally, because for typical audio there is usually a lot more energy in the lower frequencies (closer to 0) than in the higher frequencies, the magnitudes are transformed to a logarithmic scale before plotting. These conventions are followed throughout this work.

2.3 Spectral Averaging, Spectral Plot, and Spectral Envelope

When assessing the general frequency versus magnitude characteristics of an audio signal, however, it often makes sense to look at the plot of spectral line magnitudes over an entire section of audio, where the x axis is the frequency axis and the y axis is the magnitude axis. Taking the DFT over the entire section of audio is often still impractical even for this purpose, because as N increases, the noisy (jagged) nature of the resulting plot, in general, remains [Sm10, §5.11]. A different approach is therefore used: the squared magnitudes of the DFT lines are averaged over multiple successive DFTs taken over small chunks (*windows*) of the source data, which can be either non-overlapping (*Bartlett method*) or overlapping by half in the time domain (the more accurate *Welch method*). As more windows are averaged, the noise is reduced in proportion to the square root of the number of windows [Sm97, §9.1]. To increase the estimation accuracy even further, a *window function* (*apodization function*) is usually applied to the signal before its DFT is taken, resulting in what is known as the *short-time Fourier transform*. Typical window functions are a variety of symmetric bell-shaped curves; they modulate (weight) the signal in such a way that the points at the edges of a window receive less weight than the points close to the center. In doing so, they reduce the *spectral leakage* associated with the Fourier analysis, which mainly arises because the Fourier transform (with its implicit treatment of any signal as periodic in both time and frequency domains) is applied to pieces of aperiodic sequences or sequences whose periodic components do not fit exactly into the chosen window size (i.e. have a non-integer number of periods per window). High spectral leakage has the effect of introducing into the spectral average non-zero magnitudes at frequencies where the actual signal may have no energy at all, thus resulting in *spectral bias* and raising the *noise floor* of the analysis (i.e. the minimum meaningful magnitude in the plot). Refer to either [Sm10] or [Sm97] for more information on this subject.

Once the data is averaged over a sufficiently large number of windows, it is plotted. The resulting plot is somewhat interchangeably referred to in different sources as the *spectral*

plot, the *power spectrum*, the *spectral density*, or the *periodogram* of the signal. Along with the spectrogram, it is one of the most frequently used tools in audio analysis. Just as with the spectrogram, the averaged squared magnitudes are normally transformed to the logarithmic *decibel* (dB) scale before plotting, using the formula $L_j = 10 \log \overline{|X_j|^2}$, or equivalently $L_j = 20 \log \sqrt{\overline{|X_j|^2}}$ (the overline denotes averaging) if the software uses linear (voltage) rather than quadratic (power) units internally. Thus, the scale of the spectral plot in the dB units is usually consistent from one application to another. However, this is not the case with the bias of the plot; different software packages bias the spectral plot differently in the dB units (or equivalently, scale it differently in the linear or quadratic units). The most straightforward approach is to leave the power spectrum in whatever units it was in after taking the DFT; when the DFT is defined as (1) and the window function integrates to 1 on its domain, these will also happen to be the units of the original waveform (see (3) above). But frequently, before being transformed to the dB scale, the averages are also normalized by the source audio's sampling rate or the Nyquist frequency (and sometimes also a factor of $\frac{1}{2\pi}$) to give average power per unit of frequency (or angular frequency), an estimate of the *spectral power density* (SPD) function. This is the case for all spectral plots presented in this work as well. A sample spectral plot, averaged over a time period of 5 seconds with 2048 samples per window, is shown in Figure 2. Note that the

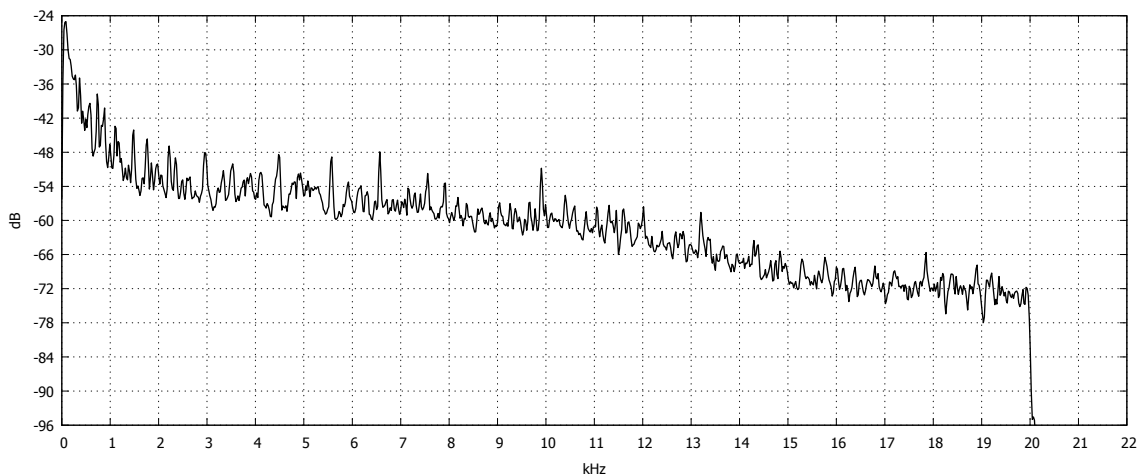


Figure 2: Example of the spectral plot of a wide-band audio signal

plot is normally truncated at a certain minimum magnitude (in this case -96 dB) because anything below that is usually spectral leakage and is not meaningful to the analysis.

When taken over a sufficiently long period of time or otherwise smoothed to eliminate any random variation from the data, the spectral plot (or rather, the curve which would be plotted) is referred to as the *spectral envelope* of the signal—the primary factor which affects the audio’s perceived *timbre*, especially in the higher-frequency hearing range [LR04, §1.4.6]. Regenerating degraded signals in such a way that their spectral envelopes mimic those of the original signals (or at least behave in a way that would be expected for such signals)—more importantly in the long term but preferably in the short term as well—is therefore important for high-frequency bandwidth extension applications.

3 Existing Methods

As a rule, the symptoms of absent high-frequency content in audio are readily visible on the signal’s long-term spectral plot. Figure 3 shows the power spectrum of the same signal

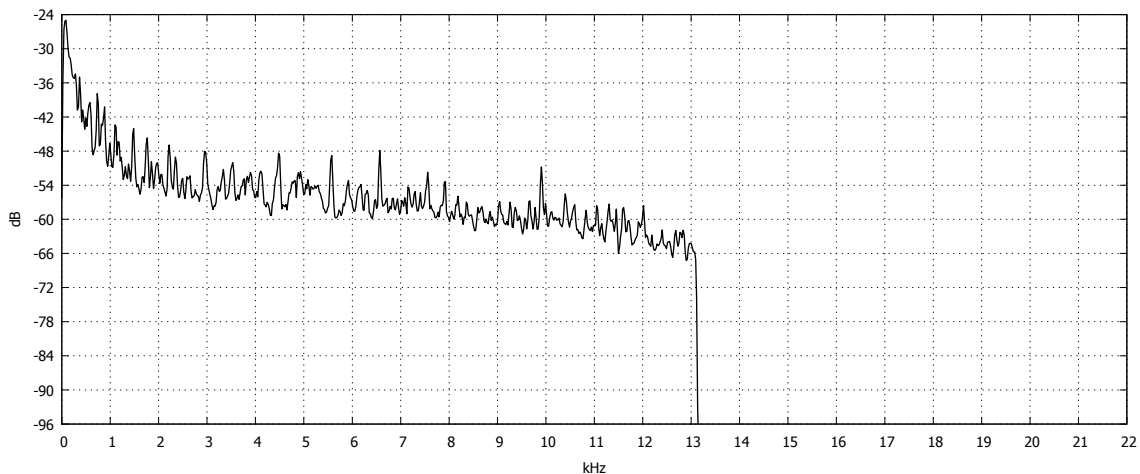


Figure 3: Example of the spectral plot of a low-pass filtered audio signal

as in Figure 2, but after it has been filtered with a brick-wall low-pass filter with a cut-off of 13 kHz (a brick-wall filter is a filter with a very steep transition, such as what might be applied during resampling to a lower sampling rate or by lossy encoders to minimize

the noise in the remainder of the spectrum). Whereas in the original signal, the spectrum extended all the way to 20 kHz, in the filtered signal there is essentially nothing beyond 13 kHz, which gives the resulting sound its dull and muffled quality. In this chapter, we survey the existing and proposed methods for dealing with the problem of missing high-frequency content.

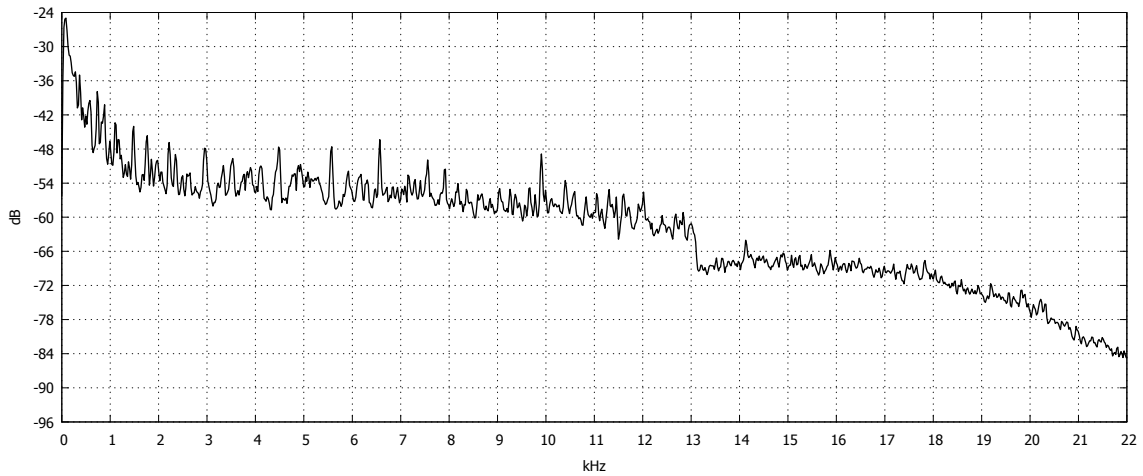
3.1 Equalization

The first thing an audio engineer might be tempted to do with most audio restoration software is to apply an *equalizer* to boost the high-frequency end of the spectrum in the degraded signal. An equalizer (named so because it was originally developed to flatten the spectral envelopes of signals) is a linear filter with a user-adjustable response curve; sometimes it is convolution-based, sometimes DFT-based (since one of the DFT properties is that circular convolution of one sequence with another in the time domain is equivalent to pointwise multiplication of the sequences' Fourier transforms in the frequency domain). Essentially it scales parts of the signal's spectral envelope based on a filter response curve drawn by the user. Some of the commercially available implementations (such as Elevation FreEq Boy or Voxengo CurveEq) even offer as much as automatic matching of the spectral envelope of one signal to a reference envelope computed from another signal. But unfortunately, in a brick-wall-filtered signal there is not much content in the missing frequency band to scale—whatever content was there is now reduced well below the noise floor. Thus attempting to scale the missing frequency band magnitudes only introduces irregular, uncorrelated or improperly correlated noise to the signal. Consequently, this approach fails outright.

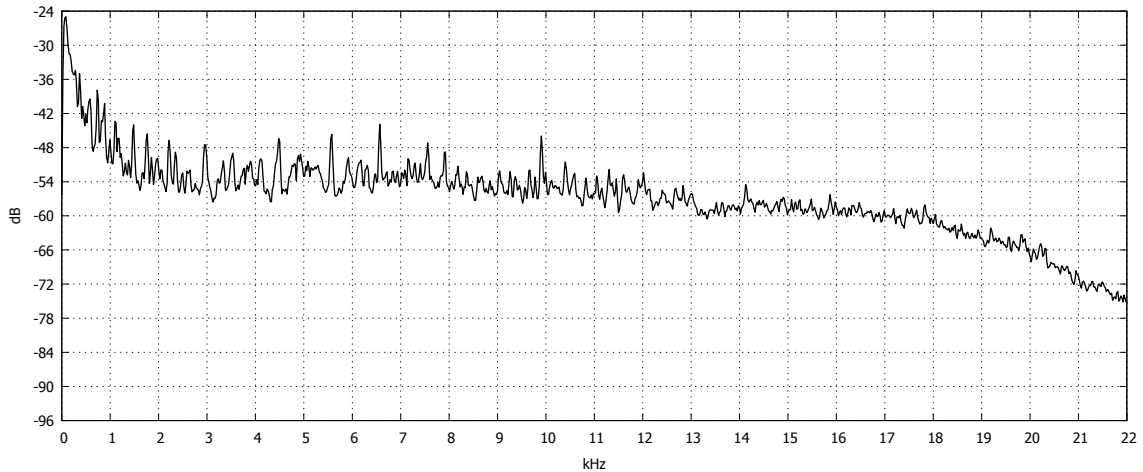
3.2 Harmonic Excitation

Another thing one might attempt to do is described in [LR04, §5.4]. This method is based around the concept of a *harmonic exciter*. Typically, it is a non-linear function, such as

$y = x^2$ or $y = |x|$ (a *full-wave rectifier*), which takes the existing band-limited signal (or its linearly filtered version) and generates *harmonics* (i.e. integer multiples) of the frequency lines still present in the band-limited signal. The output of the non-linear function is then high-pass filtered (has its lower-frequency spectral lines removed) and mixed with the band-limited signal to produce its enhanced version. To author's knowledge, this method of bandwidth extension, in one form or another, is used in all professional or semiprofessional audio restoration software where bandwidth extension is offered, including Nero Wave Editor version 5 (henceforth NWE, part of the Nero 9 package), Steinberg WaveLab version 6 (its included Spectralizer plug-in), DiamondCut Audio Restoration Tools version 7 (henceforth DC-ART), and the open-source Crystallinity post-processing plug-in available for various media players. Other surveyed software, including Sony Sound Forge version 10, did not appear to offer any bandwidth extension functionality out of the box, nor was the author able to find any free or commercially available third-party VST plug-ins which would offer any better functionality in this regard than what was already available in NWE, DC-ART, or WaveLab. In the existing implementations, the user is able to adjust the cut-off frequency (which in most software is documented to be the non-linear function *input* cut-off frequency), the density (the amount of harmonics to generate), and the usual wet/dry mix (the overall amount of effect to apply). Unfortunately, this set of controls is largely inadequate for dealing with signals with abrupt cut-offs in the power spectrum, such as the one in Figure 3. Figure 4(a) shows the spectral plot of the band-limited signal in Figure 3, processed with Steinberg's Spectralizer plug-in with moderate settings (Input: 0 dB, Gain: 100.0, 2nd: 100.0, 3rd: 0.0, Mix: 25.0, Frequency: 6000, Density: 3). From the plot we can see that while the software did regenerate the high end of the spectrum, it did a poor job at extrapolating the spectral envelope of the signal, leaving a large jump discontinuity at the original cut-off frequency. Furthermore, if we attempt more aggressive settings to reduce that discontinuity and increase the Mix parameter to 75.0 (see Figure 4(b)), we see that while it does reduce the "step" in the spectrum, the side effect is that the



(a) Moderate settings



(b) Aggressive settings

Figure 4: Spectral plots of the band-limited signal in Figure 3, processed with a harmonic exciter

frequency lines significantly *below* twice the specified harmonic exciter cut-off frequency (12 kHz) gain roughly 6 dB, that is, increase in magnitude roughly two-fold. This is a highly undesirable result, because firstly the algorithm distorts the spectral envelope of the signal in the frequency range which the user did not even request the tool to process, and secondly, this behavior indicates that the exciter is probably introducing a foreign signal into that frequency range, which means that even if we were now to apply an equalizer

to make the spectral envelope look like it should, the non-linear distortion introduced by the harmonic exciter would remain audible. Auditory examination and comparing the spectrograms of the band-limited and enhanced signals confirm this suspicion. Worse, even if the controls were flexible enough to tailor this method to this and similar test cases, by its design the algorithm would have only been capable of generating integer multiples of the existing frequencies; for example, to regenerate missing spectral lines around 14 kHz, it would have had to use the existing spectral lines around 7 kHz at best. Such a large gap makes this approach inflexible by design; while the regenerated signal may have a “natural” harmonic structure (i.e. frequencies only coming in multiples of one another, without which the regenerated frequencies might be expected to sound “off key”)—due to the structure of human ear and its fairly low sensitivity to pitch variation at high frequencies ([LR04, §1.4, §6.3.5], and also in author’s personal experience)—the correct harmonic pattern for high frequencies is not nearly as important as their proper temporal localization. And the more distant the spectral lines are from one another in the frequency domain, in general, the less correlated their magnitudes tend to be. Thus, for a signal where frequencies below 13 kHz are still readily available, it is likely that better estimates of the missing spectral line magnitudes could be obtained from the lines located a lot closer, for example around 12 kHz, than those a whole octave away. But the algorithm does not allow the user to take advantage of that.

Of all the software examined by the author, DC-ART seems to offer the best implementation of the harmonic exciter model. In DC-ART version 7, it is implemented as part of a tool called Virtual Valve Amplifier (VVA), which simulates the non-linear characteristics of vintage vacuum tube rectifiers (and other tubes as well, at the user’s discretion), allowing the user to vary parameters such as drive and operating point (thus controlling the amount and type of harmonics generated), but more importantly, also featuring a threshold control. This control only “switches on” the harmonic exciter when the source signal is louder than the specified threshold, thus allowing high-frequency

generation in louder sections without garbaging the spectrum during quiet passages (where improperly generated harmonics can sound quite unpleasant). Test samples processed with DC-ART are therefore included in §5 of this work.¹

3.3 Available Research

Attempting to re-use some of the already available research on this topic also yields only general ideas. Firstly, while a good deal of research has been done in the realm of bandwidth extension for *speech* signals because of its most obvious application, telephony (the entire [LR04, §6] is devoted to bandwidth extension for speech, with numerous references to other sources), these methods (by the admission of the chapter's author) would be unlikely to work well with arbitrary audio [LR04, §6.10] because of very specific model assumptions (not applicable to generic audio) that most of the speech bandwidth extension systems rely on when it comes to spectral envelope extrapolation. While such assumptions may be necessary in the realm of telephony, where the signals are severely band-limited (normally with 3.4 kHz filter cut-off), less severe cases of audio degradation (11–14 kHz cut-off), which the present work is tailored for, could benefit from a less powerful but generally more robust spectral envelope estimation model. However, some speech bandwidth extension techniques apart from envelope estimation (for example those described in [LR04, §6.3]) can be (and are) reliably extended in this work to handle general audio.

The idea of regenerating missing high-frequency spectral line magnitudes from lower frequencies has also gained a widespread use in audio compression. The Spectral Band Replication (SBR) has been developed as an add-on feature to popular extensible audio compression formats such as MP3 (resulting in mp3PRO) and AAC (resulting in aacPlus)

¹As this paper was being prepared, DC-ART version 8 was released which introduced a more traditional harmonic exciter (named Overtone Synthesizer) as an alternative for high-frequency restoration. This approach is similar to that implemented by NWE's Band Extrapolation and Steinberg's Spectralizer, and is subject to much the same limitations including the mandatory one-octave range. In most cases it does not outperform the VVA, which retained its exciter functionality without change in version 8.

[DL02][Ek02]. While this technique does not work blindly—it requires certain metadata to be present for envelope estimation, which is injected into the stream during encoding—and consequently only works in tandem with an SBR-enabled encoder, it is actually listening to aacPlus streams which gave the author the idea that something similar could be attempted in the realm of audio restoration as well.

Despite that, so far as high-frequency bandwidth extension for general audio is concerned, the research remains scarce, and the author found only two fairly recent works, [LL03] and [BN07]. They suggest similar and in some respects more adaptive techniques than those used in this work, but unfortunately the former was only discovered by the author relatively recently (long after the project was initially implemented), while the latter was not yet published at that time. To author’s knowledge, these techniques are yet to make their way into mainstream audio restoration software, and moreover, the author was unable to find any project or personal Web pages related to these works, or any other place where implementations of these algorithms could be obtained. Thus, no comparative analysis can presently be given for these techniques.

3.4 Nearest-neighbor Resampling

Lastly, one purely time-domain approach is worth attention here that is widely used in Microsoft Windows multimedia engine for sampling rate conversions, and also by older video games like Duke Nukem 3D. When a file is due to be upsampled to a new sampling rate which is double the original (for instance, 44.1 kHz versus 22.05 kHz), Windows multimedia engine simply copies the existing samples to the new data points, which results in a step-like appearance of the waveform. This turns out to have approximately the same effect as modulating the source signal by a cosine wave of the Nyquist frequency (or equivalently, multiplying each sample by a constant with alternating sign, and then adding the resulting signal with the original, an approach described as *spectral folding* in [LR04, §6.3.3]); because of the modulation and periodicity properties of the DFT, this

produces a reflection of the signal’s power spectrum about half the Nyquist frequency. Nearest-neighbor resampling is somewhat more adaptive than spectral folding; intuitively, the more high-frequency components there are in the original power spectrum (which mean a lot of steep oscillations in the time domain), the more of them there will be in the regenerated band (because the greater the “steps” are), while less high frequency in the original means lesser steps and therefore less undesirable noise. This approach is not without its issues—for example, it is prone to a similar integer factor limitation as the algorithm described in §3.2, and somewhat prone to problems described in [LR04, §6.3.3] as well, specifically the low-frequency content inappropriately folded into the high-frequency range and the spectral gap between the original and regenerated bands. But for its sheer simplicity, computational efficiency, and non-existent delay in streaming applications, this technique achieves spectacular results. It is ironic that more sophisticated resampling algorithms (those which *avoid* folding the spectrum) often produce perceptually worse results. This method is therefore included among comparisons in §5 below.

4 Implementation Details

As demonstrated above, all the existing implementations available to the author have generally failed to do an adequate job at dealing with missing high-frequency spectral content. The goals set before the project was started were several:

- Unlike the existing harmonic excitation and nearest-neighbor resampling methods, the final implementation must be flexible and configurable enough to allow restoration of missing spectral content at other than whole-octave intervals;
- With a proper choice of parameters by the user, the program should not modify the spectral content *below* the cut-off frequency, i.e. in the region where the spectral content is fine and no action needs to be taken;

- In general, with a proper choice of parameters the algorithm should achieve perceptually satisfactory results for most audio signals which have been low-pass-filtered with a cut-off frequency no less than 12 kHz (which was the cut-off frequency of the degraded audio which initially prompted the author to develop the software);
- The program must be publicly releasable and usable by advanced users and audio engineers for remastering purposes, not just by the author for experimentation. Unlike most commercial implementations of audio processing algorithms, which are essentially blackboxes and have only rudimentary documentation which does not really explain what the tool does and exactly how it does that, this project needs to be transparent to a competent user, which means being free (libre), open-source, well documented, and conceptually simple (at least as much as it can be while remaining effective).

In this chapter we focus on conceptually relevant details of the development process, explaining the decisions made about the processing algorithm.

4.1 Spectral Translation

The first thing to be decided was what exact information should be used to restore the high-frequency content, and how. This was answered based on the author's prior experience in the use of spectral analysis tools for audio restoration purposes. Firstly, there was an observation that at least for the kind of audio the author had to deal with, there is a very high amount of correlation between the magnitudes of spectral lines above approximately 6–8 kHz; for instance, observe the spectrogram in Figure 1 above. Secondly, some experience working with audio compression techniques (specifically with the source code of [LAME](#) MP3 encoder project and its then-current GPSYCHO psychoacoustic model) has provided insight on the concept of tonality estimation. The *tonality* of a signal (i.e. the quality of it

being composed of one or several strong spectral lines versus having a uniform noise-like distribution) is important in lossy audio compression because it determines how much noise can be introduced by the encoder without it being perceptible by a listener—the more tonal, the less noise can be introduced [Jo88][TH05]. On the other hand, GPSYCHO stops tonality estimation at about 8.9 kHz by default, assuming frequency bands above that to have a constant medium tonality, and this behavior does not significantly influence the perceptual quality of audio output by the encoder. This assumption is also confirmed by casual examination of most spectrograms and spectral plots. Thirdly, the usage of Spectral Band Replication in modern compression algorithms (the very name of the technique) suggested the idea of translating spectral lines from one location to another in the Fourier spectrum. Thus to restore the missing high-frequency band, one might attempt to simply take the DFT of the signal, define the start and end spectral lines s and f delimiting the frequency band immediately preceding the cut-off (f being the cut-off or close to the cut-off), and then copy and juxtapose that frequency band beyond the cut-off, thus extending the bandwidth of the signal from f to $f + (f - s) = 2f - s$. To the present day this remains the general principle of the program's operation. This approach is identified as *spectral translation* in [LR04, §6.3.3]; much like spectral folding (see §3.4), it is employed for speech bandwidth extension, but only in tandem with speech-specific spectral envelope estimation models.

One problem with this that can be thought of outright is that it would require an 8 kHz-wide band to fill the gap between the original 12 kHz cut-off and the highest frequency a human ear can perceive, generally accepted to be around 20 kHz, which would require s to be around 4 kHz, somewhat too low in the frequency range to be reliably temporally correlated with spectral content higher up the frequency line. However, the author's prior experience in audio restoration confirmed that one needs not go that far; normally, a cut-off of 16 kHz (and sometimes, depending on the audio, even 14 kHz) is quite adequate for the resulting audio not to be perceived as “muffled.” In fact, due to internal MP3 stream

format limitations, MP3 encoders routinely truncate or distort the spectral content above 16 kHz (when operating with the typical 44.1 kHz or 48 kHz sampling rate), even at higher bit rates. This can be easily detected by looking at spectrograms, yet remains inaudible to most people. For example, in the case of our low-pass filtered signal in Figure 3, we might choose s to be 10 kHz, and f to be 13 kHz. Copying and pasting this band beyond f would allow us to extend the signal's bandwidth to 16 kHz, as shown in Figure 5.

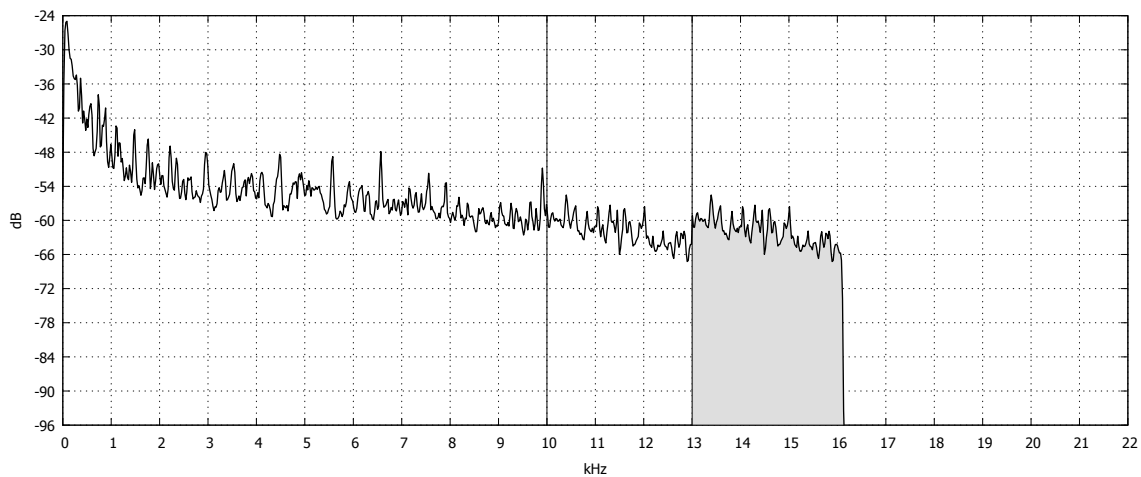


Figure 5: Spectral plot of the signal in Figure 3 following an application of raw spectral translation

To implement this approach practically, firstly the author needed to decide on the programming language to use. Since the program is aimed towards general audience rather than academics only and needs to be able to run on a typical computer without any specialized packages, C was chosen instead of popular numerical computing environments like MATLAB (or interpreted languages like Java) because of its portability, performance, and native (rather than bytecode) compilation.

Secondly, the time scale of the Fourier transform needed to be decided, so that the requirements for stream access routines could be set. As explained above in §2.2, taking the DFT of the entire signal and processing that would most likely not yield any useful results, because while manipulating the spectral content, the synchronicity of the regenerated frequency bursts with the bursts of frequencies they were regenerated *from* would likely

be lost, so even though the spectral envelope of the signal would look right in the long term, it most likely would not be correct in the short term. Thus it was immediately decided to take the DFT piecewise in the time domain, applying the algorithm to each piece successively, with a window size that can be changed by the user depending on the nature of the source audio. The default window size is chosen to be around 5 ms (given the audio's sampling rate), but rounded to the nearest power of 2 for performance reasons. For a common sampling rate of 44.1 kHz, this amounts to 256 samples (or 5.8 ms), which on one hand provides an adequate time resolution (for comparison, the shortest MP3 transform window is 192 samples (or 4.3 ms) for 44.1 kHz sampling rate), and on the other hand allows adequate frequency resolution (about 170 Hz per spectral line) for algorithm's needs as well.

Because of some prior experience in manipulating the DFT, the author was already aware that an overlap-add windowing mechanism was needed, where successive DFTs are taken over windows which overlap by half; after processing, a weighted average for each sample is computed to yield the output signal. The program has a selection of window functions to apply to the signal before it is fed to the DFT engine (and unapply after the processed signal is taken back to the time domain), including the well-known rectangular, triangular, raised cosine (Hann), and Nuttall (see [Sm10, §3] or the source code in Appendix A for definitions of various window functions). The default raised cosine in most cases does an adequate job at minimizing the audible artifacts resulting from spectral leakage.

Lastly, the specific method for transferring spectral lines had to be chosen. The *similarity (scaling) theorem* was considered, which essentially states that compressing a signal by a factor of a in the time domain results in stretching it by the same factor of a in the frequency domain, but unfortunately, the theorem is applicable to continuous time and frequency domains only and does not carry well into the discrete domains, because either a or $1/a$ is necessarily non-integer for $a \neq 1$. However, the DFT property which can still be used to justify the approach of simply copying spectral lines is known as the

modulation theorem. Modulation refers to pointwise multiplication of two sequences in the time domain. In the form which is useful to us, the modulation theorem for the DFT can be stated as follows: if $X_j = \mathcal{F}(\{x_k\})_j$ and $\{h_k\} = \cos(2\pi ak/N)$ for $0 \leq j \leq N - 1$ and $0 \leq k \leq N - 1$, then

$$\mathcal{F}(\{x_k h_k\})_j = \frac{1}{2}X_{j-a} + \frac{1}{2}X_{j+a}, \quad (4)$$

where $j - a$ and $j + a$ are to be interpreted modulo N . Essentially, this means that pointwise multiplying an arbitrary sequence $\{x_k\}$ by a cosine wave with a periods per window splits the Fourier spectrum of the signal, shifting half of it upwards and half of it downwards by a spectral lines. In our case, we are only interested in shifting the spectrum upwards, so we would be concerned about the $j - a$ shifted band overlapping with the $j + a$ shifted band. However, because in our case $a = f - s$ and there is virtually no spectral content beyond f with a proper choice of operating parameters, the shifted bands do not overlap anywhere beyond f . Because DFT also has the property of being *linear* (that is, for arbitrary N -point sequences $\{x_k\}$ and $\{y_k\}$ and arbitrary constants c and d , $\mathcal{F}(\{cx_k + dy_k\})_j = c\mathcal{F}(\{x_k\})_j + d\mathcal{F}(\{y_k\})_j$), so far our frequency domain processing can be equivalently represented in the time domain by low-pass filtering the original signal at f (if necessary), modulating the signal by a cosine carrier wave with a periods per window, scaling it by a factor of 2, high-pass filtering it at f , and then pointwise adding the low-pass filtered and high-pass filtered versions. All this should result in a well-behaved signal as the individual operations are known to be well-behaved when applied to audio; in particular, the spectral content below f is untouched, which was one of the requirements. Additionally, we use a real-to-complex and complex-to-real DFT types for forward and inverse transforms respectively, so we do not have to worry about adjusting any X_j beyond the Nyquist frequency (see §2.1); this is done implicitly by the DFT engine.

However, it was discovered that this algorithm worked as expected in only half of the cases, specifically where the amount of shift $a = f - s$ was *even*. To understand why it didn't work as expected for odd a (instead producing nasty-sounding interference patterns),

we observe that although amplitude modulation $h(y) = y \cos(2\pi at)$ is a *linear operation* with respect to y (for arbitrary signals $y_1(t)$ and $y_2(t)$ and arbitrary constants γ and δ , $\gamma h(y_1) + \delta h(y_2) = h(\gamma y_1 + \delta y_2)$ at any fixed point in time t)—which is the property that allows the windowed overlap-add method we employ to produce consistent results in the first place—it is not *time-invariant*, that is, if $y(t) = y(t - \tau)$, generally it is *not* the case that $h(y(t)) = h(y(t - \tau))$, because h is itself a function of t . Luckily, in our case it is a periodic function, so we only need to consider the specific amount of overlap for our piecewise processing to fix the problem. For odd a , we observe that the carrier cosine wave $\cos(2\pi ak/N)$ has an odd number of periods per window. Since windows overlap by half, the carrier wave enters each window in a phase opposite to that in the previous window, so failing to take that into account results in partial cancellation of the modulated signal in the overlapping regions. We therefore need a more general version of the modulation theorem than what we have in (4)—one that would allow us to vary the initial phase ϕ of the carrier wave, that is, assume $\{h_k\} = \cos(2\pi ak/N + \phi)$, and tell us what the frequency-domain representation of $\{x_k h_k\}$ would be in that case. Manipulating the Euler’s formula $e^{i\phi} = \cos \phi + i \sin \phi$ to get $\cos(\phi) = (e^{i\phi} + e^{-i\phi})/2$, we can express $\{h_k\} = (e^{i(2\pi ak/N + \phi)} + e^{-i(2\pi ak/N + \phi)})/2$, and then do simple algebra on (1) to establish

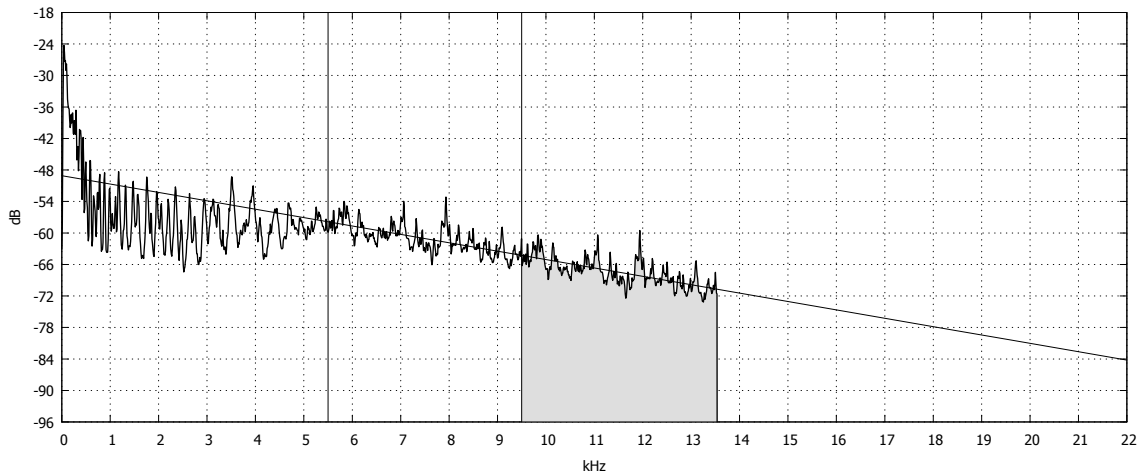
$$\mathcal{F}(\{x_k h_k\})_j = \frac{e^{i\phi}}{2} X_{j-a} + \frac{e^{-i\phi}}{2} X_{j+a}, \quad (5)$$

which indicates that, as suspected, for odd a and windows overlapping by half, we need to add π to the phases of the of the copied spectral lines (or equivalently, just flip the signs of both real and imaginary components) in every other window, which corrects the issue and allows the overlap-add method to work as expected. If in the future an option is added to overlap windows by a different amount (as is sometimes done in other signal processing applications), the amount of phase adjustment for each window will need to be reconsidered again.

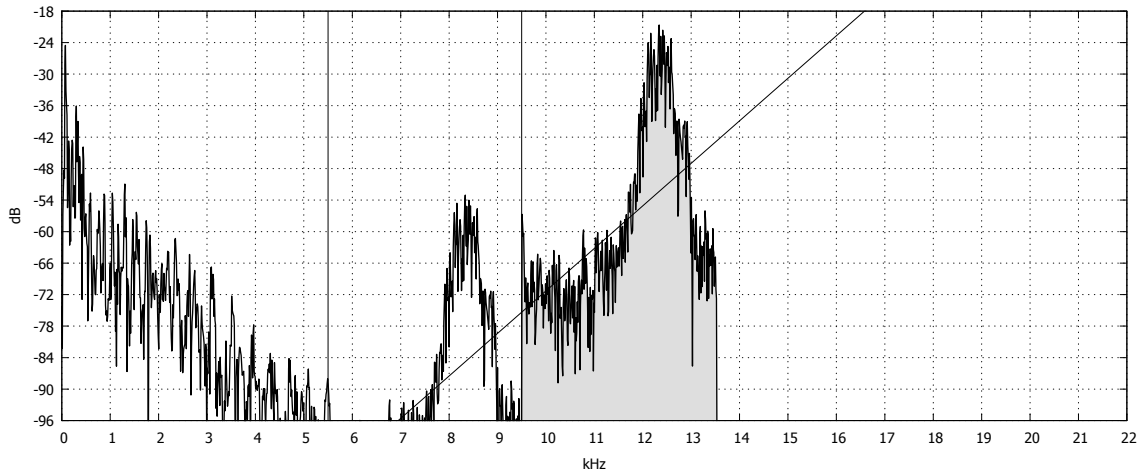
4.2 Spectral Envelope Estimation

At this point, we have an approach to restore missing spectral content from existing content. In author's experience, this blind copying of spectral content already provides decent perceptual results in many cases, especially after applying an equalizer to eliminate the jump discontinuity in the long-term power spectrum at the former cut-off frequency f (which is very visible in Figure 5, for instance). However, this is not fool-proof because even though we can even out the power spectrum in the long term by applying filters with appropriate parameters, the spectrum can still remain markedly discontinuous in the short term. Therefore, in general we would like to have a more adaptive approach for connecting the original and restored spectral bands together, preferably an algorithm that operates on a window-by-window basis and thus provides better-shaped power spectra in the short term. We start by making an observation that the power spectrum in Figure 5, if taken on the dB scale and the random noise is discounted, is overwhelmingly linear in terms of frequency in the range between s and f (which in this case correspond to 10 kHz and 13 kHz respectively when the sampling rate is taken into account). We may therefore attempt to fit a simple least-square-error regression line through that range using a logarithmic scale for spectral magnitudes (similar to the dB units used in the plot). Once the estimated slope β of that line is known, the value $\beta(f - s)$ may be added to each regenerated spectral line magnitude on the logarithmic scale (or, equivalently and more intuitively, it can be taken to the linear scale and used as a factor to scale the magnitude of each regenerated spectral line). Because initially, the original and the copied frequency bands are identical in magnitude and are placed side by side, this procedure is expected to create a smooth transition between the bands. An example of such a good fit is shown in Figure 6(a) (here, s maps to 5.5 kHz and f maps to 9.5 kHz). A logarithmic *frequency* scale has also been tried, but in general it did not yield any significant improvements, so a linear frequency scale is presently used by the program.

However, it was quickly discovered that using short-term power spectra from each



(a) The power spectrum is overwhelmingly linear in the fit range, and the resulting line fits accurately



(b) Local peaks in the power spectrum prevent a meaningful fit

Figure 6: Possible results of fitting a least-squares line through log-scale spectral line magnitudes in the short-term power spectrum

individual window was in most cases inadequate to obtain a good linear regression. Local temporal variations in the spectrum will often prevent a meaningful fit, and the resulting scale factor applied to the regenerated band will often only make spectral envelope shape worse than it used to be (see e.g. Figure 6(b)). The resulting estimates end up not only exaggerated but unstable from window to window, giving the audio an unpleasant “hoarse” characteristic.

Several measures were introduced to work around this problem. Firstly, the DFT window size can be increased to span greater time segments, thus providing more points for the fit, but while it does offer improvement, the side effects of doing so (normally in the form of “breathing,” that is, temporal smearing of sharp transient signals) are often unpleasant as well. As an alternative to that, a separate analysis pass was introduced to the program; no data is output on this pass, and it is only used for envelope estimation. In this pass, a running average of the power spectrum is computed for each window (using the data from a user-adjustable number of neighboring windows), and the slope of the regression line is determined based on this average. This stabilizes the fit significantly while largely avoiding the “breathing” artifacts associated with larger window sizes, thus yielding vastly improved perceptual quality compared to both per-window fit and blind spectral translation. Greater number of windows to average the power spectrum over tends to yield more stable but also more inert estimates of the spectral envelope slope, so that both extremes can increase discontinuities in the short-term spectra. In the absence of a user-specified parameter, the default averaging period is chosen to be approximately 0.1 s (amounting to 34 overlapping windows at 44.1 kHz sampling rate), which yields acceptable results in most cases.

Secondly, the program has an option to use a different set of points for envelope estimation than the one used for spectral translation; they both share the end line f , but the start line can be adjusted separately for each (from now on, s for spectral translation and l for envelope estimation). Extending the fit range (and sometimes contracting it) can improve the fit, especially if there is a marked non-linearity in the fit range, for example long-term highly tonal signals (where increasing the window size or the number of windows for the running average has little effect).

Thirdly, some numerical statistics for the regression line are displayed by the program following the analysis phase to assist the user with choosing appropriate parameters pertaining to the envelope slope estimation. The first statistic is the root mean square error

(RMSE) of the log-scale magnitudes in the fit range, calculated as $\sqrt{\frac{TSSE}{N_c N_w (n-2)}}$, where TSSE is the sum of squared residuals after the fit over all windows and channels of the audio file, N_c and N_w are the numbers of channels in the file and the number of DFT windows per channel respectively, and $n = f - l + 1$ is the number of points that the lines were fitted through ($n - 2$ being the number of degrees of freedom for each). Generally, the lower the RMSE, the more meaningful the fit is on the average; values less than 4.5 dB are usually good, whereas greater values can indicate either unstable estimates, or significant non-linearities present in the portion of the spectrum used for estimation, or both. In the former case, increasing the number of windows in the running average can improve quality, but if that doesn't affect RMSE much, the user should attempt to revise the range of frequencies for the envelope slope estimation (and probably for the spectral translation as well) to use wider, narrower, or otherwise more linear regions of the spectrum.

The second statistic is the p -value α_0 for the null hypothesis $\beta = 0$, or simply, the likelihood that the spectral envelope estimation employed by the program is of no use for that particular audio and that particular choice of parameters; the closer to 1, the more likely. Note that this can be either due to not having low enough variance for a significant fit (inadequate number of windows in the running average), or because there are significant non-linearities present in the fit range, or because there was a meaningful fit, but the true envelope slope is indeed 0 (as is the case for e.g. white noise, whose spectral envelope is flat by definition). α_0 is computed separately for each window (given the log-scale spectral average through which the regression line is fitted) as a two-tail Student t -test for a regression line slope (see e.g. [KK98, §5.7]):

$$\alpha_0 = 2T_{n-2} \left(- \left| \beta \sqrt{\frac{(n-2)SSX}{SSE}} \right| \right),$$

where β is the estimated slope of the regression line, $T_{n-2}(x)$ is the CDF of the Student's t -distribution with $n - 2$ degrees of freedom, SSE is the sum of squared residuals after the

fit for a particular window, and

$$\begin{aligned}
 SSX &= \sum_{j=l}^f (j - \bar{j})^2 = \sum_{j=l}^f \left(j - \frac{f+l}{2} \right)^2 = \frac{(f-l)(f-l+1)(f-l+2)}{12} \\
 &= \frac{(n-1)n(n+1)}{12}
 \end{aligned}$$

since the frequency lines j are uniformly distributed. The results are placed into an array, which at the end of the analysis is sorted, and the familiar minimum, 1st quartile, median, 3rd quartile, and maximum are displayed. These values are intended to give the user an idea whether there were *sections* of audio where the fit was meaningful or not meaningful, and whether using the envelope extrapolation feature (with the parameters provided) was justified overall.

The computational accuracy of α_0 depends greatly on how well the linear model assumptions are satisfied for a given sample. To assess that, we need to make repeated measurements of the log-scale magnitude average at some specified spectral line, and then compare the distributions of these measurements at different spectral lines. This is difficult to do with any accuracy given a highly dynamic nature of audio signals; we can, however, perform this exercise for a reasonably long sample of white noise. To that end, first we need to consider the probability distributions of log-scale magnitudes at each spectral line (which are the input to the regression). These variables are averages of magnitudes coming from multiple DFT windows. Different successive power means been tried for averaging these short-term magnitude spectra: the cubic mean M_3 , the quadratic mean M_2 , the arithmetic mean M_1 , the geometric mean M_0 , and the harmonic mean M_{-1} . M_0 , M_1 and M_2 all perform approximately even; each works better for some samples than others. M_2 (the traditional Welch periodogram) seems to do best on the average in terms of RMSE, but because the quadratic average is biased away from 0, the result is sometimes an overestimated slope for windows *in the vicinity* of a window with a large burst of high-frequency energy. On the other hand, the geometric mean M_0 is biased *towards*

0 and thus is more resilient to such scenarios. But more importantly, it is equivalent to the arithmetic mean on the logarithmic scale, which is not biased either towards or away from 0 and gives us additional security in terms of the probability distributions of the log-scale magnitudes fed to the regression model, because with the increasing number of averaged windows, these distributions are expected to be asymptotically normal by the Central Limit Theorem, regardless of what the probability distributions of each averaged term are. Consequently, the geometric mean M_0 is the one employed by the program for averaging the short-term power spectra. Simple inspection shows that the distributions of logarithmically averaged spectral line magnitudes taken on a sample of white noise are indeed close to normal (the more windows are considered for the average, the closer to normal, as is expected due to the Central Limit Theorem), and the variances of the distributions at each frequency line are roughly equal, as assumed by the linear model (of course, these variances depend on the number of windows averaged). Figure 7 shows one

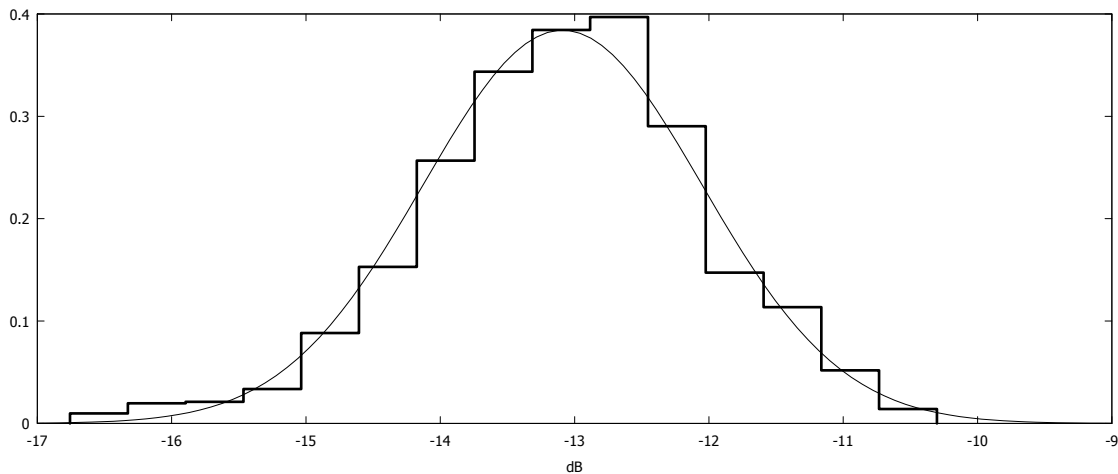


Figure 7: Histogram of the average log-scale DFT magnitude distribution at a single frequency line, taken on a sample of white noise, overlaid by the p.d.f. of a normal distribution of equal mean and variance

such distribution, obtained from a 5-second sample of white noise with a sampling rate of 44.1 kHz, a window size of 256, and 34 windows per spectral average.

The next issue to address is independence of measurements in each spectral line, and

here we have a slight problem in that the magnitudes of spectral lines are normally not entirely independent of one another due to spectral leakage. Experimentation indeed shows that the use or non-use of a window function has a pronounced effect on the values of α_0 and RMSE, however, the use of a balanced window function that significantly reduces spectral leakage at a lesser or greater expense of frequency resolution (Hann, Bartlett, Nuttall) yields reasonably consistent values regardless of which exact function is used; compared to no window function (i.e. rectangular window), the use of such function with otherwise equal parameters perceptually improves the results as well (even though it generally increases RMSE). Using a window function is therefore crucial for proper operation of the algorithm.

From this, we may conclude that the computation of α_0 and RMSE should behave reasonably well with high-frequency bands for most samples (which for high-frequency spectral content are mostly noise-like) with the default (Hann) window function and a reasonable choice of window size, starting and ending lines for the fit. However, having a statistically significant fit with e.g. $\alpha_0 = 10^{-5}$ at each window still tells us nothing about the variability of the envelope slope estimate *across* windows, but practice shows that a significant variability there usually results in bad perceptual quality regardless of how well the line fits individually through each window. To address this issue, we compute and output the third statistic, the p -value α_p for the null hypothesis $\beta = \beta_p$, where β_p is the estimated envelope slope for the *prior* window:

$$\alpha_p = 2T_{n-2} \left(- \left| (\beta - \beta_p) \sqrt{\frac{(n-2)SSX}{SSE}} \right| \right).$$

This is again output in a familiar box-and-whiskers fashion. As opposed to α_0 , it is desirable that the value of α_p be *high*, that is, it is desirable for the value of the slope estimate β for each window *not* to be significantly different from that of the prior window. The target values depend on the window size, but for the default 5 ms, generally the 3rd quartile α_0 of 0.2 or less combined with the 1st quartile α_p of 0.75 or more yields

perceptually stable spectral envelopes (according to author's experience).

Increasing the number of windows to average tends to improve all three statistics at the expense of the algorithm's temporal resolution. If the guidelines cannot be satisfied with any choice of parameters and no perceptually good results are obtained (indicating that either the spectrum is mostly flat and does not need extrapolation, or that there is a lot of highly non-linear regions), the envelope extrapolation feature can be turned off altogether as a last resort.

4.3 Further Improvements

At this point we have an implementation that takes into account both short-term and longer-term variations in the power spectrum, but a few inexpensive enhancements can still be made to improve perceptual quality of the output.

Firstly, we need to account for the possibility that a cut-off in the low-pass filtered signal may not be as abrupt as in e.g. Figure 3; a more gradual cut-off is often introduced by the filter because abrupt cut-offs tend to introduce ringing artifacts into the signal. Working with such signal would have been inconvenient, because if we choose a cut-off frequency f too low, aiming to have a good linear fit through the magnitudes, we will discard useful data beyond f (and introduce ringing artifacts at the new cut-off), whereas if we choose f too high, aiming to preserve the entire available spectrum, this may result in a poor fit; worse, the resulting signal will have a spectral gap in it. To avoid this dilemma, instead of transferring just the region between spectral lines s and f , we transfer the *entire* region above s (whatever portion of it fits under the Nyquist frequency) to the new location starting at f . If we now choose f to end in a linear region (even though there is data beyond it), the algorithm will preserve the cut-off shape in the translated spectral band while avoiding introducing a spectral gap, thus avoiding the problem. However, f should still be chosen reasonably close to the cut-off frequency; firstly, we generally want as much of the original spectral content as possible to remain in place, and secondly, if there is significant spectral

content beyond $2f - s$, to get mathematically predictable results we can no longer ignore one of the terms in the modulation theorem (5), and need to transfer the content beyond $2f - s$ *downwards* in the spectrum as well as upwards; this is not currently done by the implementation.

Secondly, as we have seen above in §4.2, the spectral envelope estimation algorithm is based on statistics and therefore is not fool-proof. If despite all the precautions it does run into a situation like in Figure 6(b) due to some significant non-linearity of the regression region, we would like a fail-safe which prevents creating a large step-like discontinuity at the junction of the original and translated spectral bands. To that end, we introduce a slight overlap of 3 spectral lines between the original content and the translated band, where the spectral content is interpolated. Depending on the window size, unfortunately this reduces the width of the translated band (and the new cut-off frequency); for the usual window size of 256 at 44.1 kHz sampling rate, the reduction is about 0.5 kHz, but in general this tweak does a good job at concealing otherwise problematic cases. Care needs to be taken when averaging spectral content; one may be tempted just to perform vector addition, but while it does determine the resulting phases in a meaningful way, the magnitude of a vector average of two vectors is not, in general, an average of their magnitudes. Therefore, after the vector addition is performed, the magnitude of the resulting spectral component is computed separately as a weighted root mean square of the magnitudes of the averaged vectors. This avoids creating a “dip” in an otherwise perfectly linear spectral envelope.

Thirdly, the phases of the spectral lines in the generated band may be adjusted. If the source (s through f) spectral lines happen to contain signals that form sharp peaks in the time domain due to a specific combination of their phases, this (often perceptually undesirable) “click” quality is carried into the translated band as well. To combat that, the program introduces a linear sweep into the phases of the translated lines, starting from one value at f and linearly increasing to another value at $2f - s$. By scattering the phases in this manner, the impulse noise embedded in the original band can be “chirped” over

time (see e.g. [Sm97, §11.6]), reducing the “rough” quality of the sound which would otherwise result. Other methods have been tried to accomplish the same as well, including multiplying the phases by a proportion of the spectral shift for each line and adding random (but constant for the duration of execution) numbers to the phase of each component; both have been rejected because the former tends to introduce destructive interference due to overlapping DFT windows, while the latter results in a non-repeatable output. Care needs to be taken when using this feature as it tends to worsen spectral leakage artifacts, especially in temporally sparse signals such as speech, so greater values may require longer windows. The amount of phase sweep is therefore adjustable; the default range is from $-\pi/2$ to $\pi/2$ and appears to work well in most cases.

Lastly, based on a suggestion from [LL03] a limiting threshold of $\beta \leq 0$ was introduced to disallow increasing spectral envelope slope. Legitimate signals with a rising envelope in the normal bandwidth extension range for this program (which is roughly 11 kHz and up) are comparatively rare, whereas increasing envelopes below that frequency (where the spectral lines are borrowed) are more common. If the envelope is blindly extended and allowed to increase in the regenerated range as well, audible artifacts often result, most typically in the form of hissing “s” sounds when applied to speech. However, instead of blindly limiting the slope (which would result in a discontinuity at the junction of the original and regenerated bands, the very issue we are trying to avoid), we sweep the scaling factor across the regenerated band, using the usual scaling factor near the start and no scaling close to the end of the band, which results in a flat envelope in that range while not introducing a discontinuity at the junction. Unfortunately this sometimes introduces a bias into the spectral envelope estimation, because random downward variations for a flat (white-noise) spectrum are allowed whereas random upward variations are not. However, with the recommended choice of parameters (such that α_p is 0.75 or more) and a 10.5 kHz to 15 kHz bandwidth extension, the amount of long-term bias at the 15 kHz end does not exceed 0.5 dB when measured on a white-noise sample, which is perceptually negligible.

Additionally, signals with a flat spectral envelope in that range are very infrequent. On the other hand, this safeguard further mitigates the undesirable consequences of algorithm's application to cases like in Figure 6(b) and saves the user a lot of hassle involved in hand-picking appropriate parameters to handle such cases (or applying external tools like equalizers or dynamic processors to the program's output to further adjust the spectral envelope).

5 Comparative Results

In this chapter we present some visual and acoustical comparisons of the algorithm's performance with that of the methods available in some existing software. Since the algorithm is claimed to be robust enough to handle most audio signals, its performance will be tested on four different kinds of audio: electronic music, symphonic music, speech, and a mixture of music, sound effects, and speech (a sample taken from a movie soundtrack). The author has been routinely using the program to restore movie soundtracks, but additional samples cannot be presented in this paper due to both space constraints and copyright restrictions.

Each test case starts out as the original wide-band signal with a sampling rate of 44.1 kHz and an effective bandwidth of about 16 kHz. The signal is then downsampled to 22.05 kHz (as is frequently done to save space), which reduces the effective bandwidth to about 10.8 kHz, and then upsampled back to 44.1 kHz for processing, which further reduces the bandwidth to about 10.5 kHz (due to the application of anti-aliasing filters during resampling). The resulting signal is processed with three different methods, believed by the author to do the best job so far: DC-ART's Virtual Valve Amplifier (§3.2) with the choice of parameters deemed optimal by the author; the nearest-neighbor resampling (§3.4); and BandR, the program implementing the algorithm described in this paper (refer to Appendix A for binaries, source code, and a detailed user manual). The audio files, including the

originals, are provided here in a lossless format so that the process can be repeated by an interested reader; they are compressed with FLAC to save space. If necessary, FLAC decoders for all popular operating systems can be obtained at <http://flac.sourceforge.net/>.

Since the only difference between 4 of the 5 samples in each of the 4 groups is the spectral content above 10.5 kHz, in order to hear any difference between them *at all*, they should either be listened on high-fidelity audio equipment at an adequate volume, or by individuals younger than 40 years of age—preferably both. Considering that high frequencies carry little energy in most audio signals compared to lower frequencies, poor frequency response of the audio reproduction equipment coupled with the age-related hearing loss—which for frequencies 8 kHz and higher tends to be very significant even for middle-aged people (24–40 dB compared to an average 20-year-old [LR04, §1.4.4])—may push any spectral content over 10.5 kHz well below the threshold of hearing. Generally, using around-ear headphones is preferable to using earphones and speakers. Using a hearing aid (if available) or appropriate equalizer/tone control settings to boost the treble (compensating the attenuation due to poor transduction systems and hearing loss) can be effective as well.

5.1 Parameters

Table 1 summarizes the parameters used for DC-ART VVA and BandR processing. Nearest-neighbor resampling does not take any parameters other than the sampling rate, which is always converted from 22.05 kHz to 44.1 kHz. For BandR parameters, only the ones which were set by the user are specified, while the rest are implementation defaults. For 44.1 kHz sampling rate the defaults are 256 (5.8 ms) for the window size, 34 (98.7 ms) for the number of windows to average power spectra over, raised cosine (Hann) for the window function, and $0.5 (-\pi/2 \dots \pi/2)$ for the phase scatter factor.

Sample		Electronic Music	Symphonic Music	Speech	Mixed
VVA Parameters	Tube Type	Exciter			
	Spectrum	Sweet			
	Drive	100			
	Mix	100			
	Operating Point ^a	-100	100		55
	Threshold	20		5	25
BandR Parameters	Translation Start	5.5 kHz			
	Translation End	10.5 kHz			
	Averaged Windows	15 (44 ms)	19 (55 ms)	23 (67 ms)	
BandR Statistics	RMSE	3.3 dB	2.5 dB	3.2 dB	3.5 dB
	3Q α_0	$2.7 \cdot 10^{-6}$	$5.4 \cdot 10^{-10}$	0.073 ^b	$1.6 \cdot 10^{-6}$
	1Q α_p	0.75 ^c			

^a This parameter does not actually control the operating point of a vacuum tube in Exciter mode, as it is not meaningful for a gridless rectifier diode which the algorithm simulates. Instead, it controls the number of harmonics generated.

^b The median and 1Q α_0 for this test case are 0.0044 and $6.1 \cdot 10^{-5}$ respectively, indicating that the envelope slope estimation is still largely useful.

^c The Averaged Windows parameter was chosen to be a minimum number which yields 1Q α_p of 0.75. With this choice of parameters, median and 3Q α_p are approximately 0.86 and 0.94 respectively for all samples.

Table 1: Parameters used for VVA and BandR processing, with BandR statistics obtained

5.2 Electronic Music

Figures 8 and 9 show the spectrograms and the long-term power spectra of the signal in the following order: original, downsampled, processed with VVA, processed with nearest-neighbor resampling, and processed with BandR. Table 2 contains all five samples used to produce the plots. VVA does a good job at plowing through the entire spectrum here, but in doing so garbages it with noise (which can be readily heard). Nearest-neighbor resampling produces a reasonably consistent spectrum (discounting the gap in the middle, which is mostly due to an anti-aliasing filter applied during downsampling to 11 kHz), but because it reflects the spectrum about half the Nyquist frequency, it ends up introducing strong noise lines around 14–16 kHz. BandR does the best job, accurately extrapolating the

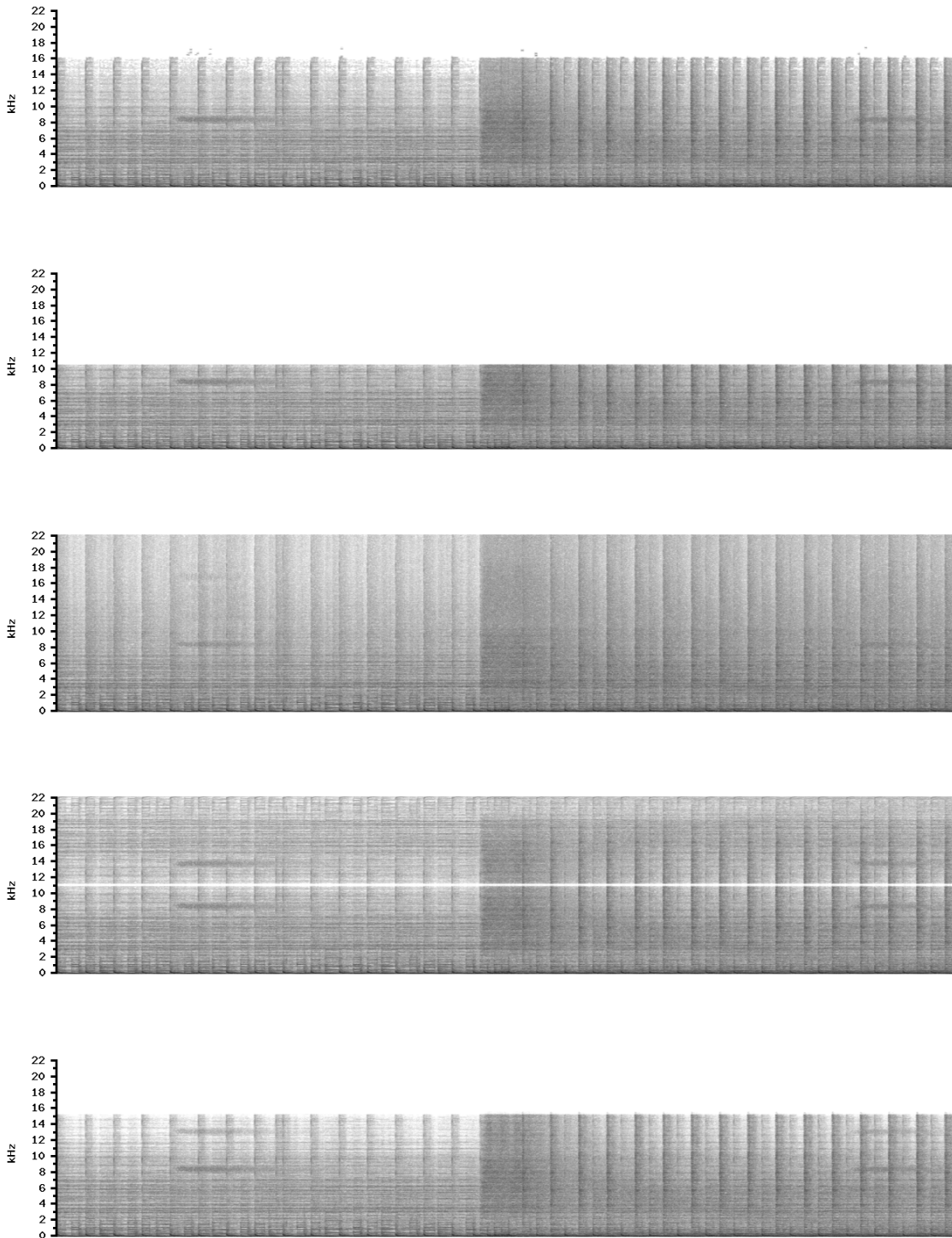


Figure 8: Spectrograms of the electronic music test signal, top to bottom: original, downsampled, processed with VVA, processed with nearest-neighbor resampling, processed with BandR

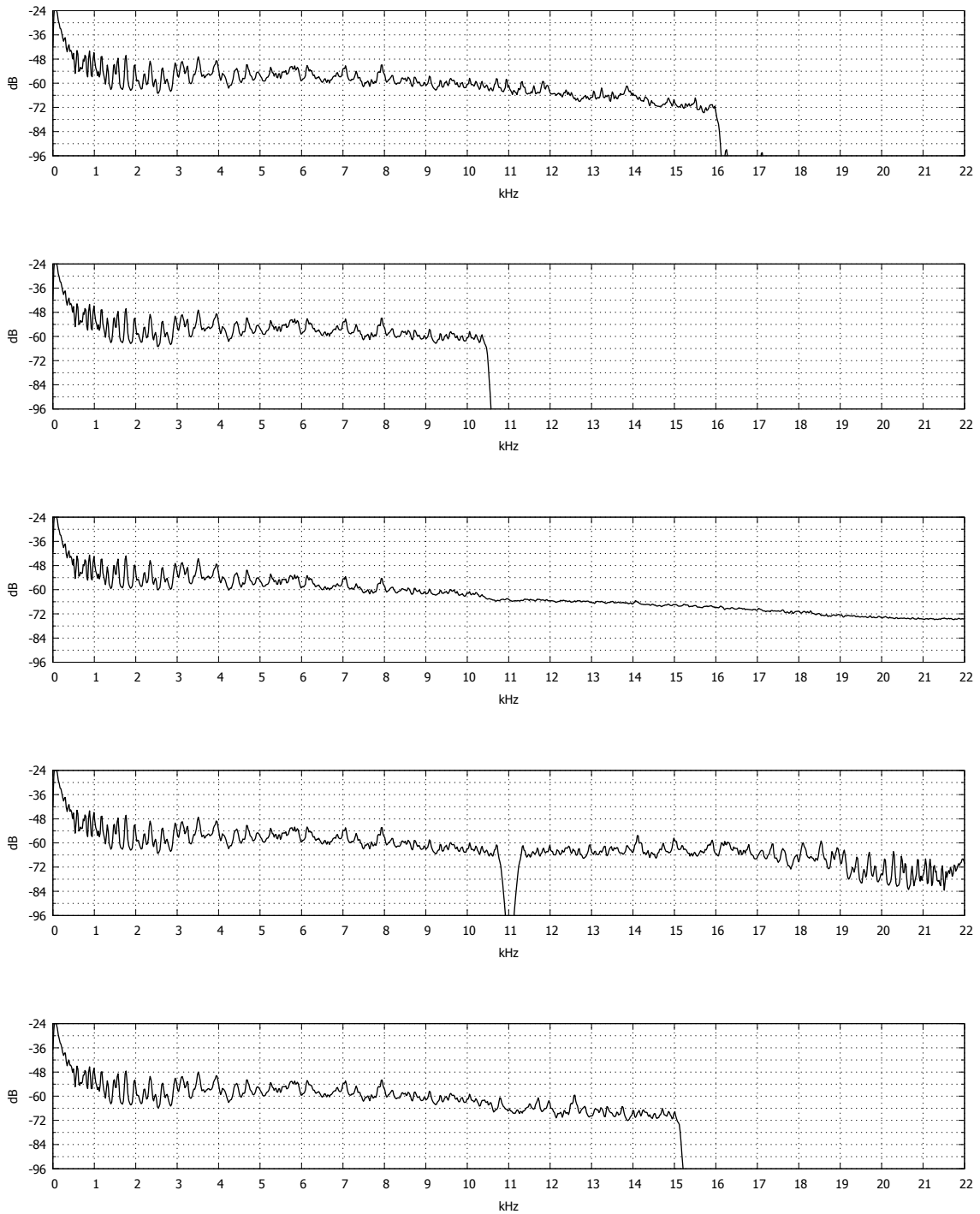


Figure 9: Long-term power spectra of the electronic music test signal, top to bottom: original, downsampled, processed with VVA, processed with nearest-neighbor resampling, processed with BandR

short-term and long-term spectral envelopes, producing a spectrum that looks and sounds closest to the original.






	music_elec_orig.flac	Original
	music_elec_ds.flac	Downsampled
	music_elec_proc_vva.flac	Processed with VVA
	music_elec_proc_nnr.flac	Processed with nearest-neighbor resampling
	music_elec_proc_bandr.flac	Processed with BandR

Table 2: Electronic music test signals

5.3 Symphonic Music






	music_symph_orig.flac	Original
	music_symph_ds.flac	Downsampled
	music_symph_proc_vva.flac	Processed with VVA
	music_symph_proc_nnr.flac	Processed with nearest-neighbor resampling
	music_symph_proc_bandr.flac	Processed with BandR

Table 3: Symphonic music test signals

Figures 10 and 11 demonstrate the spectrograms and the power spectra of a sample of symphonic music processed with various algorithms, and Table 3 contains the relevant audio samples. Once again, the VVA harmonic exciter extrapolates all the way to the Nyquist frequency at the cost of introducing distortions in the range where no processing was needed, although this sounds somewhat better than in the electronic music case because

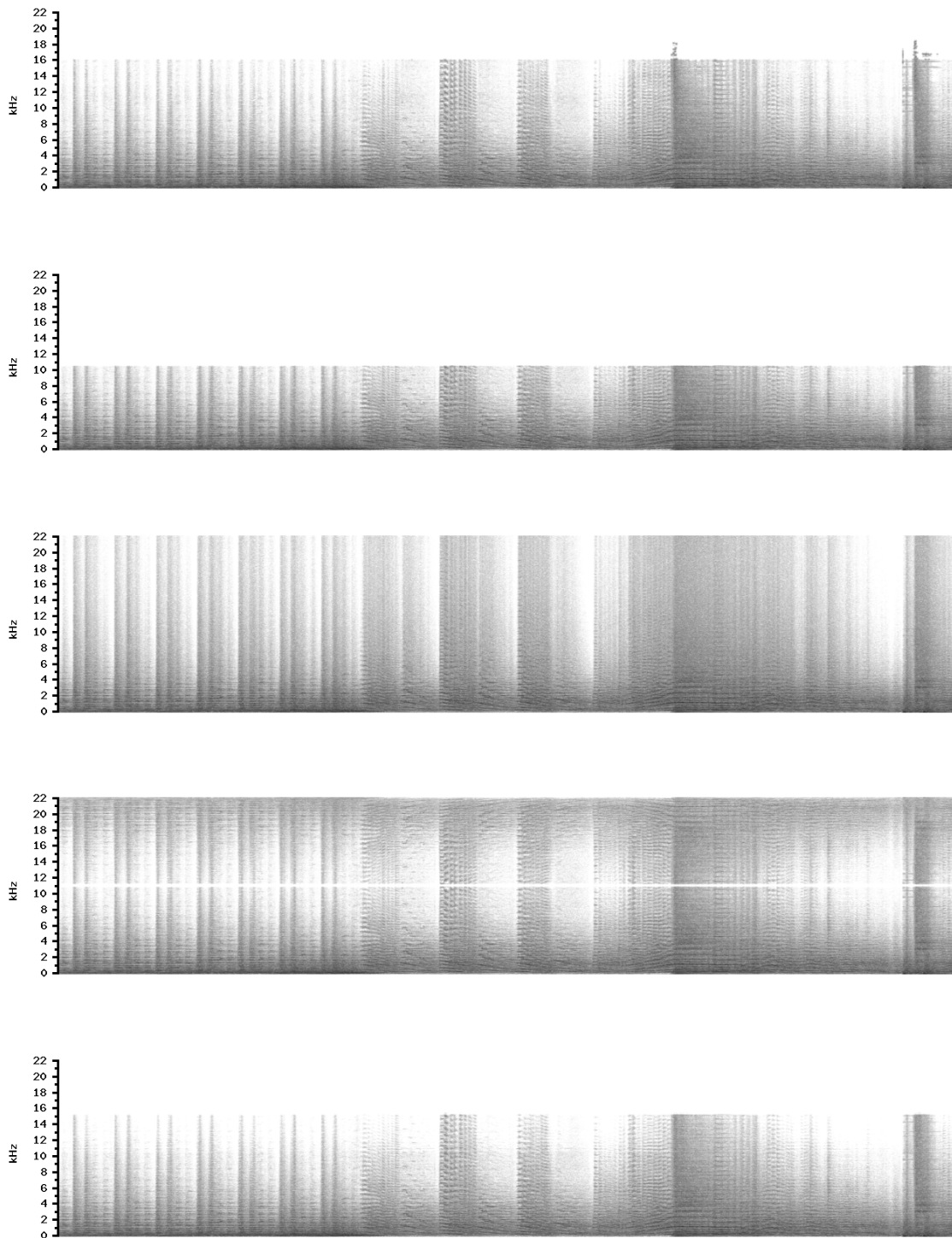


Figure 10: Spectrograms of the symphonic music test signal, top to bottom: original, downsampled, processed with VVA, processed with nearest-neighbor resampling, processed with BandR

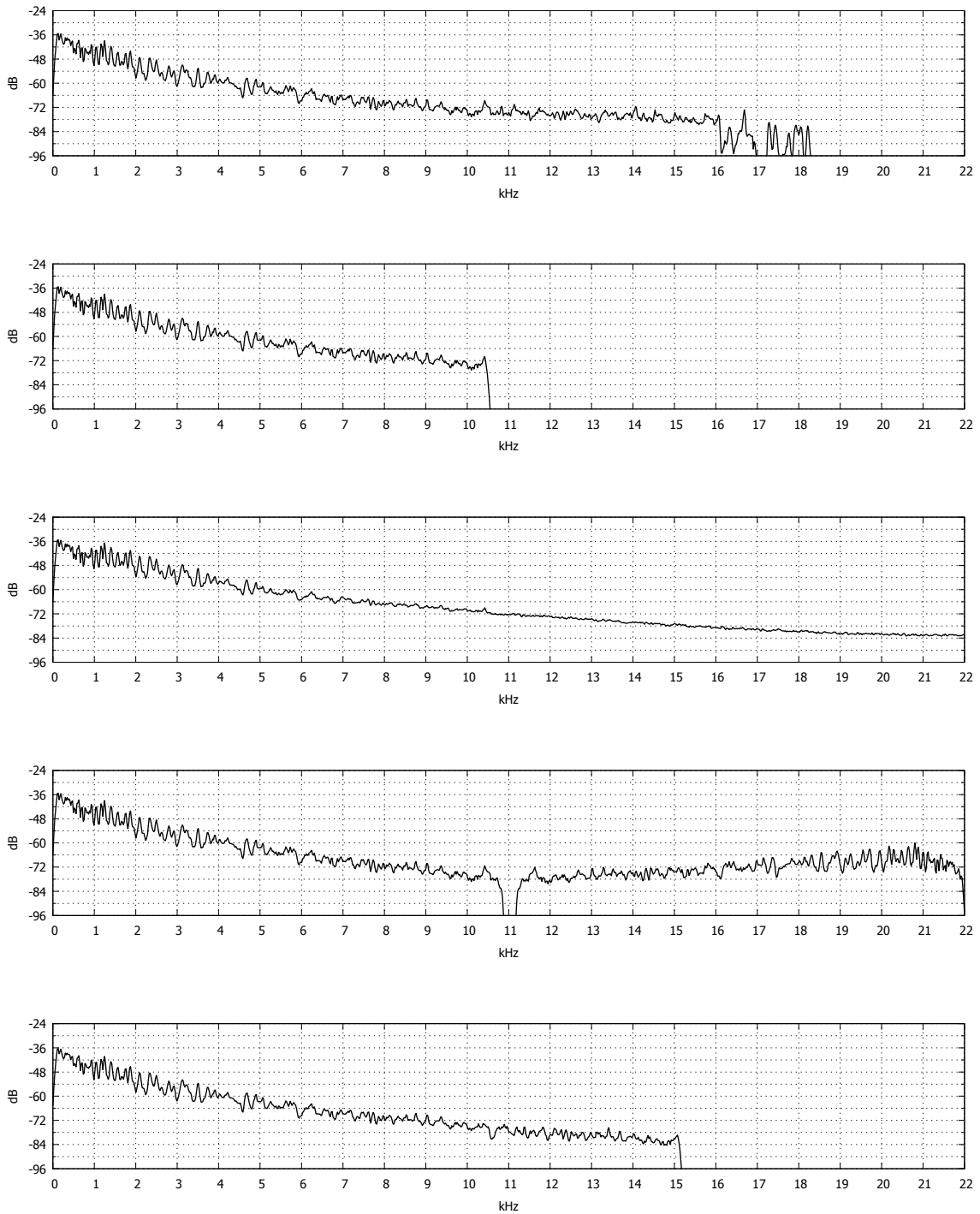


Figure 11: Long-term power spectra of the symphonic music test signal, top to bottom: original, downsampled, processed with VVA, processed with nearest-neighbor resampling, processed with BandR

the symphonic music’s high dynamic range enables effective use of the Threshold control. Nearest-neighbor resampling also sounds better this time, as it introduces no significant noise into the audible range, but its spectral envelope from 10 kHz to 15 kHz is still very distinct from the original, resulting in markedly different coloration of the sound. Once again, BandR produces an output that both looks and sounds closest to the original, albeit it underestimates the long-term spectrum in the 13.5–15 kHz range because the slope in that range of the original was not quite the same as in the range below 10.5 kHz, which was used for reconstruction.

5.4 Speech






	speech_orig.flac	Original
	speech_ds.flac	Downsampled
	speech_proc_vva.flac	Processed with VVA
	speech_proc_nnr.flac	Processed with nearest-neighbor resampling
	speech_proc_bandr.flac	Processed with BandR

Table 4: Speech test signals

Figures 12 and 13 show the spectrograms and the long-term power spectra of a speech signal processed with the compared algorithms, while Table 4 contains the samples from which the graphs were produced. The author could not find any good VVA parameters which would both avoid considerable harmonic distortion and avoid a discontinuity in the spectrum, so VVA gives the worst results here. With this sample we also observe the main drawback of the spectral translation algorithm in BandR—the generated spectral band can only be as good as the band it was generated from. As we can see, there is a spectral gap around 6.5–7 kHz in the original audio, and this gets translated into the

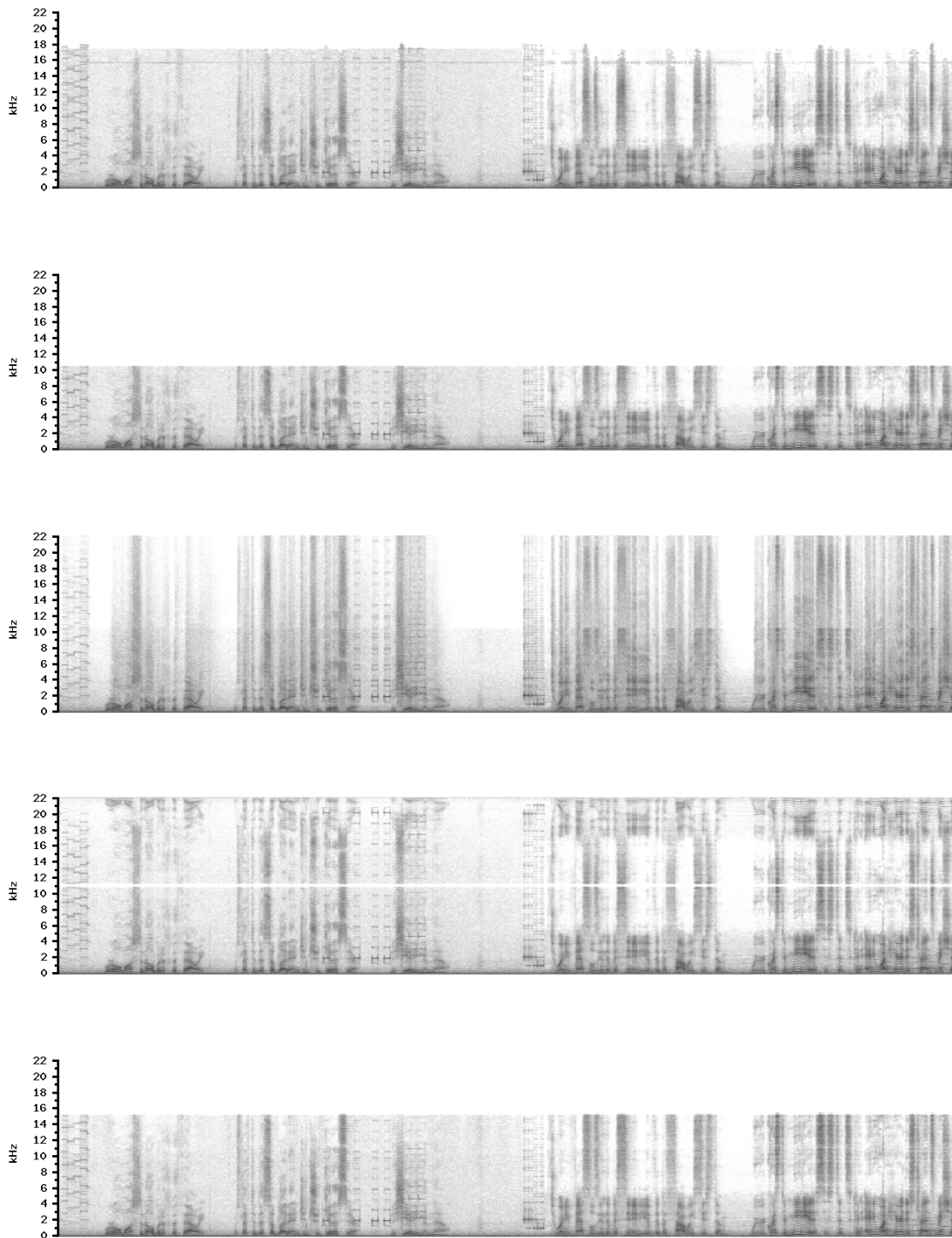


Figure 12: Spectrograms of the speech test signal, top to bottom: original, downsampled, processed with VVA, processed with nearest-neighbor resampling, processed with BandR

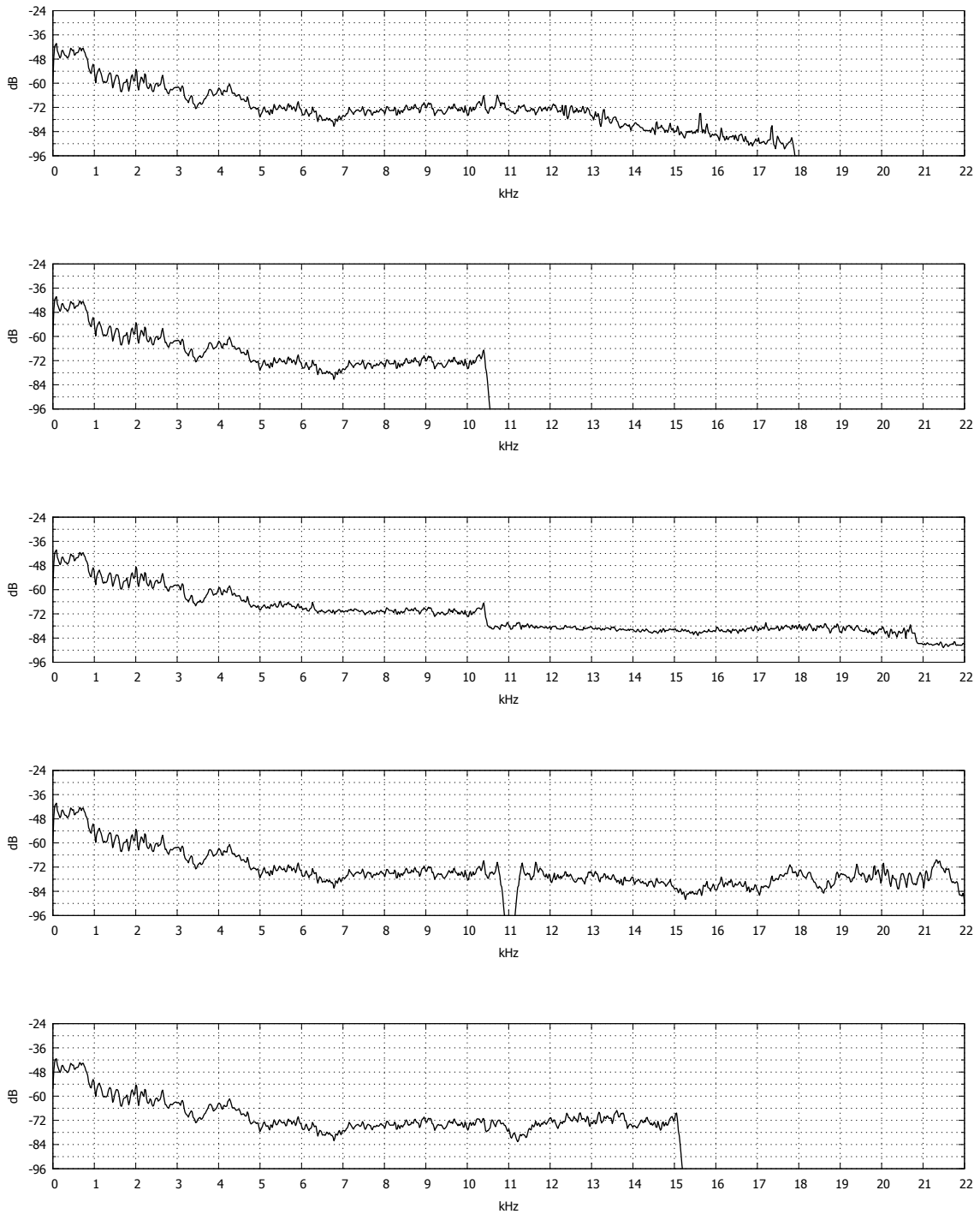


Figure 13: Long-term power spectra of the speech test signal, top to bottom: original, downsampled, processed with VVA, processed with nearest-neighbor resampling, processed with BandR

restored spectral band along with everything else. Although the algorithm again joins the bands smoothly and does a good job at flattening the short-term spectral envelope where it would have otherwise been extrapolated as rising (around the “s” sounds) and would have caused audible artifacts, it does end up somewhat overestimating the long-term envelope, and is outdone by nearest-neighbor resampling here in terms of both perceptual quality and similarity to the original (at least in the hearable range). Still, BandR produces perceptually stable results, and a listener who is unaware of the process would be unlikely to perceive any artifacts in the BandR version that they wouldn’t also hear in the original.

5.5 Mixed Audio






	mixed_orig.flac	Original
	mixed_ds.flac	Downsampled
	mixed_proc_vva.flac	Processed with VVA
	mixed_proc_nnr.flac	Processed with nearest-neighbor resampling
	mixed_proc_bandr.flac	Processed with BandR

Table 5: Mixed audio test signals

Finally, Figures 14 and 15 display the spectrograms and the cumulative power spectra of a movie sound track excerpt containing a mixture of music, sound effects, and speech with some background hustle; Table 5 contains the relevant samples. Again, with VVA the author was not able to find settings which would avoid both a discontinuity in the spectrum and a considerable distortion in the output. Nearest-neighbor resampling looks and sounds decent, but again ends up coloring the sound due to significantly overestimating the spectral envelope. BandR ends up somewhat underestimating the long-term envelope in the 10.5–13 kHz range, but for the most part it again produces the sound which graphically

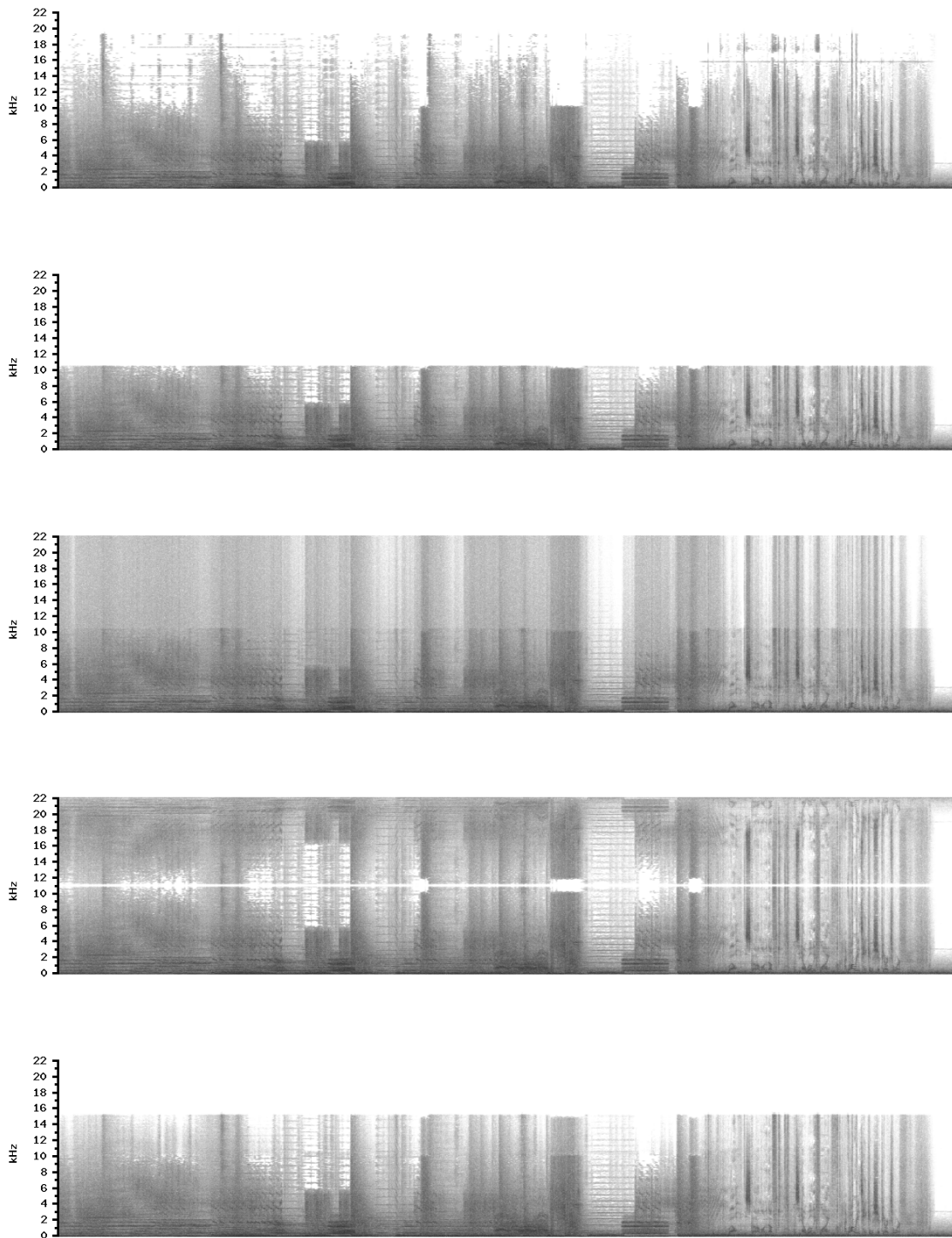


Figure 14: Spectrograms of the mixed test signal, top to bottom: original, downsampled, processed with VVA, processed with nearest-neighbor resampling, processed with BandR

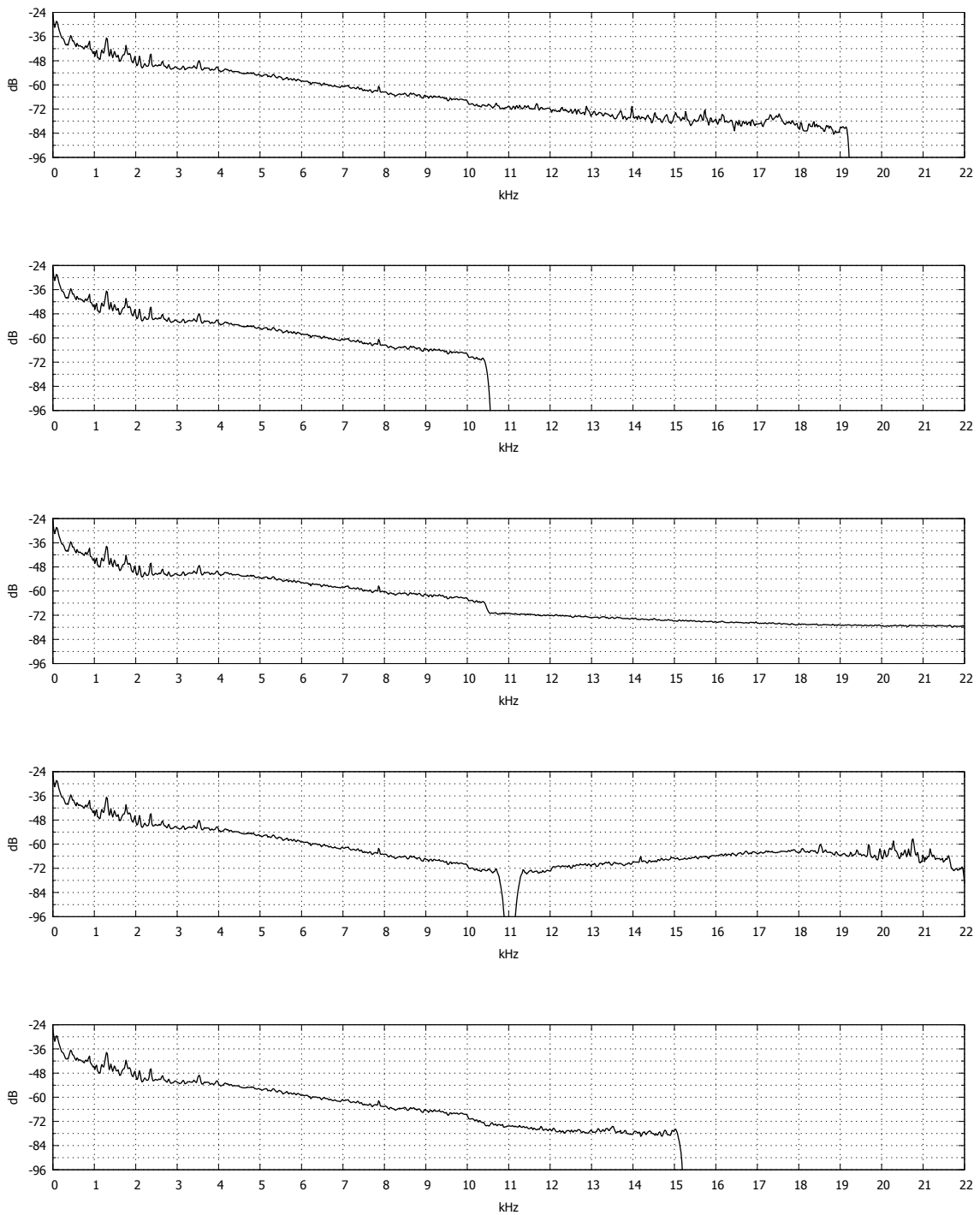


Figure 15: Long-term power spectra of the mixed test signal, top to bottom: original, downsampled, processed with VVA, processed with nearest-neighbor resampling, processed with BandR

and acoustically most closely follows the original, with the exception of three explosions which only extend up to about 10 kHz in the original, but end up being extended into higher frequencies by BandR. However, a user who has not heard the original probably would not be able to identify that as an artifact.

6 Conclusion and Potential Improvements

While the resulting algorithm for the most part is adjustable and robust enough to be adequate for author's purposes (which mostly consist of remastering poor quality music and audio tracks), the following additional enhancements could be implemented to improve the program.

- An option to replicate the selected band multiple times could be added, in order to effectively tile the missing portion of the spectrum. This could allow the program to handle more extremely low-pass filtered audio, as well as regenerate the spectrum beyond the usual 15–16 kHz range (albeit extending the bandwidth further is largely unnecessary in the overwhelming majority of cases). [LL03] suggests determining the band used for reconstruction dynamically on a per-window basis; [BN07] also suggests determining the cut-off frequency f automatically. However, doing that properly would prove challenging as the authors of the above papers do not go into the details of how they dealt with the issue of getting the overlap-add windowing method to work correctly in tandem with these techniques. In author's experience, doing just about anything with the spectral line phases that is inconsistent from window to window results in often unpredictable destructive interference patterns when the windowed sequences are taken back to the time domain and overlapped.
- Perhaps the linear fit p -value metrics α_0 and α_p could be used for automatically deciding whether to apply spectral envelope shaping on a per-window basis.

- Presently, the spectral envelope slope estimation is based on a number of windows *centered* on the target window; however, the human ear’s temporal resolution on the rising transients (*attacks*) is a lot greater than on the falling transients (*releases*), and because of this the ear notices *pre-echo* much more readily than *echo* (the “breathing” artifacts resulting from inadequate temporal resolution of the algorithm). Perhaps shortening the leading edge and extending the lagging edge of the running spectral average window range used for envelope estimation could provide higher perceptual quality.
- The implementation can probably be made a lot cleaner by using other sinusoidal transforms than the DFT, particularly the *modified discrete cosine transform* (MDCT) which is often employed in lossy audio compression. Firstly, discrete cosine transforms, unlike discrete Fourier transforms, imply an *even* (mirror image) extension of a finite sequence, rather than a *periodic* (tiling) extension. If the signal is manipulated in the frequency domain in a way that causes it to spill over from the wrong end of the window when the inverse transform is taken, discontinuities are a lot less likely to be audible that way. Secondly, the MDCT has the overlap-add concept built into it; the transform takes a $2N$ -point real-valued sequence in the time domain and returns an N -point real-valued sequence in the frequency domain; the inverse transform takes an N -point sequence and returns a $2N$ -point sequence which, when overlap-added with such inverse transforms of the preceding and following windows, yields the original signal. However, it remains to be seen how and whether the DFT properties which allow the algorithm to work, in particular the modulation theorem, can be adapted to work in the same way with the MDCT.
- Multithreading can easily be introduced to improve performance on most modern systems by simply processing each channel of the audio in a separate thread. Another screaming potential performance improvement involves combining the

envelope analysis and spectral translation passes, but that would require significant architectural changes to the program.











- The default parameters could probably use further tweaks. Command-line syntax should be improved to read parameters by name rather than by position.
- A VST or DirectX plug-in with a front-end GUI should be written based on this implementation, to allow the use of the algorithm directly from mainstream audio mastering software such as Sony Sound Forge and Steinberg WaveLab.

References

- [BN07] Chatree Budsabathon and Akinori Nishihara. *Bandwidth Extension with Hybrid Signal Extrapolation for Audio Coding*, IEICE Trans. Fundamentals, Vol. E90-A, No. 8, pp. 1564–1569. Oxford University Press, Oxford, UK, 2007.
- [DL02] Martin Dietz, Lars Liljeryd, Kristofer Kjörning, and Oliver Kunz. *Spectral Band Replication, a Novel Approach in Audio Coding*. AES 112th Convention, Munich, Germany, 2002.
- [Ek02] Per Ekstrand. *Bandwidth Extension of Audio Signals by Spectral Band Replication*. MPCA-2002 1st Proceeding, Lauen, Belgium, 2002. Available: <http://www.esat.kuleuven.be/psi/spraak/seminars/mpca/papers/ekstrand:mpca02.pdf>.
- [Jo88] James D. Johnston. *Transform Coding of Audio Signals Using Perceptual Noise Criteria*. IEEE Journal on Selected Areas in Communications, Vol. 6, No. 2, pp. 314–323. IEEE Publishing Services, Piscataway, NJ, 1988.
- [KK98] David G. Kleinbaum, Lawrence L. Kupper, Keith E. Muller, and Azhar Nizam. *Applied Regression Analysis and Other Multivariable Methods*, 3rd Edition. Duxbury Press, Pacific Grove, CA, 1998.
- [LR04] Erik Larsen and Ronald M. Aarts. *Audio Bandwidth Extension: Application of Psychoacoustics, Signal Processing and Loudspeaker Design*. John Wiley & Sons, West Sussex, UK, 2004.
- [LL03] Chi-Min Liu, Wen-Chieh Lee, and Han-Wen Hsu. *High Frequency Reconstruction for Band-limited Audio Signals*. DAFX-03 6th International Conference, London, UK, 2003. Available: <http://www.elec.qmul.ac.uk/dafx03/proceedings/pdfs/dafx59.pdf>.
- [Sh95] Hagit Shatkay. *The Fourier Transform—A Primer*. Brown University, Providence, RI, 1995. Available: <http://www.cs.brown.edu/research/ai/dynamics/tutorial/Postscript/FourierTransform.ps>.
- [Sm97] Steven W. Smith. *The Scientist and Engineer’s Guide to Digital Signal Processing*. California Technical Publishing, San Diego, CA, 1997. Available: <http://www.dspguide.com/>.
- [Sm10] Julius O. Smith III. *Spectral Audio Signal Processing*, March 2010 Draft. CCRMA, Department of Music, Stanford University, Stanford, CA, 2010. Available: <https://ccrma.stanford.edu/~jos/sasp/>.
- [TH05] Mark Taylor, Robert Hegemann, Naoki Shibata, Gabriel Bouvigne, Takehiro Tominaga, Alexander Leidinger, Rogério Brito, Mike Cheng, and Frank Klemm. *The LAME Project*, Version 3.97 beta (2005), Psychoacoustics routines (psymodel.c). Available: http://lame.cvs.sourceforge.net/viewvc/*checkout*/lame/lame/libmp3lame/psymodel.c?revision=1.142&pathrev=MAIN

A Source Code and Windows Binaries

Table 6 contains files which can be detached to get the complete source code, the Windows executable, and the detailed user manual for version 1.2 of the program. Currently, the project file is only provided for LCC-Win32, but the author believes the code is portable enough to be compiled with minimal modifications on any little-endian platform by a C99-compliant compiler. The latest version of the program is available on the author's Web site, <http://pa3pyx.dnsalias.org/>, BandR section.

	bandr.c	The bulk of the algorithm implementation
	wave.c	Access routines for manipulating WAVE files
	wave.h	Declarations for wave.c
	fftw3.h	Declarations for imports and data types from FFTW library
	bandr.prj	LCC-Win32 project definition
	fftw3.def	List of exported symbols of FFTW library, required to build the project
	gpl.txt	The GNU General Public License (version 2) governing the acceptable use of the program
	readme.txt	The operation manual supplied with the program
	bandr.exe ¹	Windows 32-bit x86 executable build of the program
	libfftw3f-3.dll ¹	The FFTW dynamically linked library (to be placed in the same directory as bandr.exe)

¹ To enable saving of file attachments with certain file types in newer versions of Adobe Reader with their overzealous security settings, Windows registry must be edited to remove these file types from the black list; for Adobe Reader version 9, the key to edit is "HKLM\Software\Policies\Adobe\Acrobat Reader\9.0\FeatureLockDown\cDefaultLaunchAttachmentPerms\tBuiltInPermList" (setting it to an empty string is sufficient).

Table 6: Attached program files