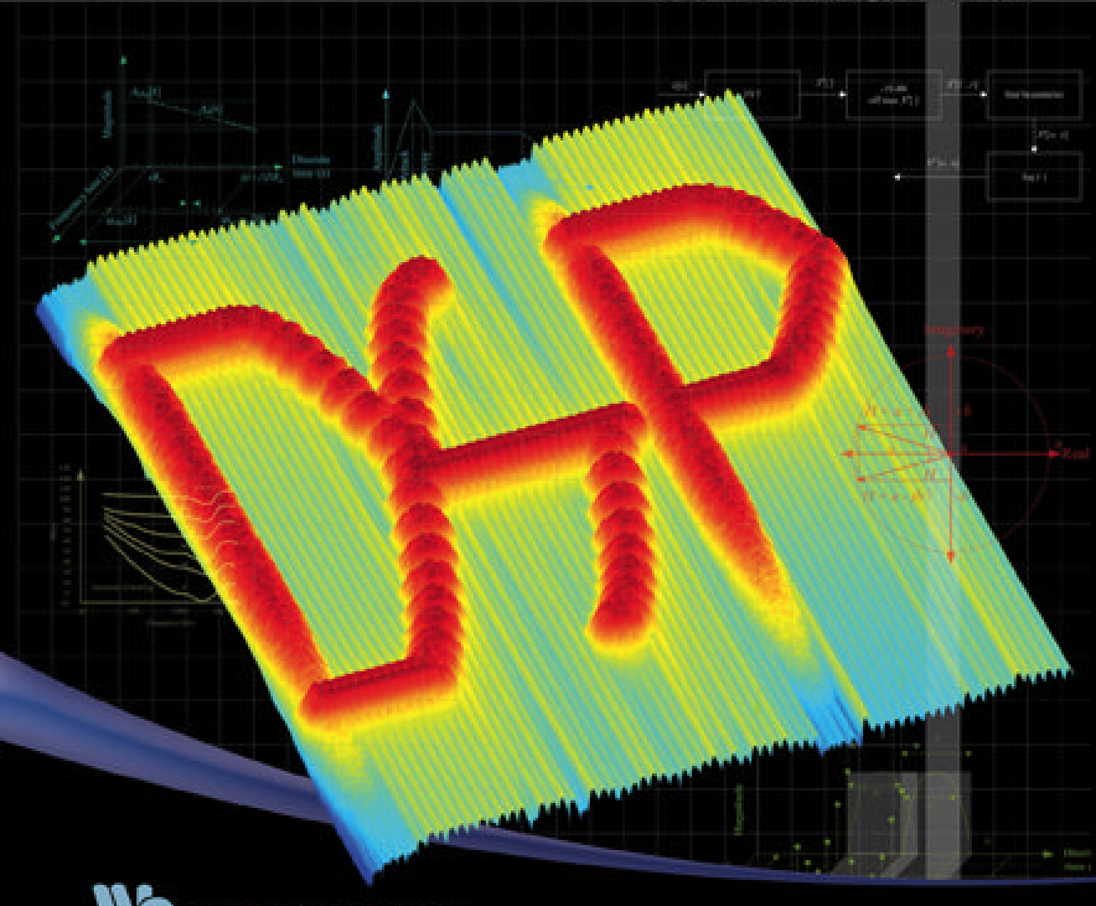
Introduction to Digital Signal Processing

Computer Musically Speaking

Tae Hong Park



World Scientific

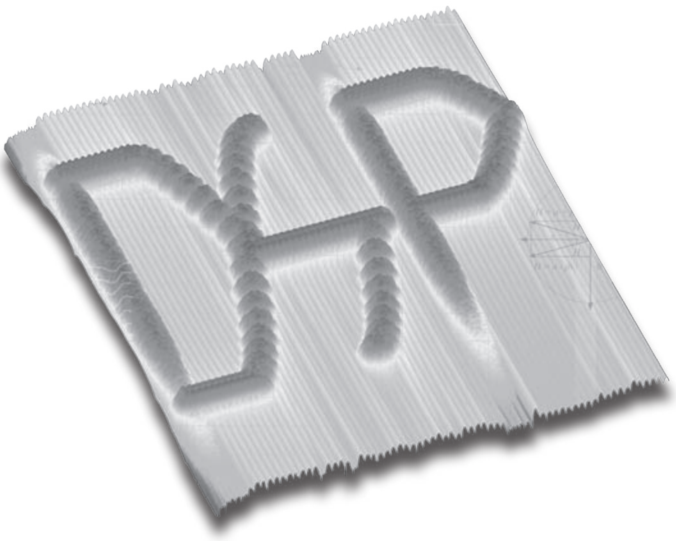
Introduction to

Digital Signal Processing

Computer Musically Speaking

Introduction to Digital Signal Processing

Computer Musically Speaking



Tae Hong Park

Tulane University, USA

World Scientific

NEW JERSEY • LONDON • SINGAPORE • BEIJING • SHANGHAI • HONG KONG • TAIPEI • CHENNAI

Published by

World Scientific Publishing Co. Pte. Ltd.

5 Toh Tuck Link, Singapore 596224

USA office: 27 Warren Street, Suite 401-402, Hackensack, NJ 07601

UK office: 57 Shelton Street, Covent Garden, London WC2H 9HE

British Library Cataloguing-in-Publication Data
A catalogue record for this book is available from the British Library.

INTRODUCTION TO DIGITAL SIGNAL PROCESSING
Computer Musically Speaking

ISBN-13 978-981-279-027-9
ISBN-10 981-279-027-6

Typeset by Stallion Press
Email: enquiries@stallionpress.com

Printed in Singapore.

PREFACE

This book is intended for the reader who is interested in learning about *digital signal processing* (DSP) with an emphasis on audio signals. It is an introductory book that covers the fundamentals in DSP, including important theories and applications related to sampling, filtering, sound synthesis algorithms and sound effects, time and frequency-domain analysis, and various topics central to the study of signal processing. The book has been designed to present DSP in a practical way, concentrating on audio and musical signals by addressing the concepts, mathematical foundations, and associated musical and sound examples to provide an appealing and, dare I say, *intuitive* way to become familiar and comfortable with the materials presented. Throughout the chapters, we try to follow the L^3 (look, listen, learn) model as much as possible — a pedagogical approach that my brother has engrained in me over many years. A bulk of the concepts and diagrams presented in this book have accompanying MATLAB® code that can be downloaded from http://music.princeton.edu/∼park/dspBook (details can be found in the appendix). During the course of the book, important theories and ideas are laid out where questions that pertain to the "why" are given special attention, especially when new materials are introduced for the first time. This approach is often rather lacking in standard engineering books, which at times tend to dwell heavily on very abstract ideas with little reference to practical situations and applications. At the same time, the book also tries to fill in some of the holes on the other end of the spectrum — the mathematical side of the coin which often is explained fleetingly without much depth in non-engineering-centric books related to signal processing. In this book, the goal is to focus on the "why" *and* the "how," with the ultimate aim to help the reader learn and understand the art of digital signal processing. As far as the reader's mathematical background is concerned, the majority of the concepts will be manageable having knowledge of algebra. The audience for this book therefore encompasses a wide range of readers, including musicians, composers, engineers, computer scientists, programmers, and undergraduate/graduate students in various disciplines.

Each chapter is structured to include the presentation of concepts, mathematical foundations, practical examples, lots of pictures/plots, and ends with a section introducing compositional and musical examples in relation to the materials covered. The book starts in the time-domain with the familiar sine tone along with the introduction of basics in acoustics and human hearing limitations. This sets up the reader in understanding why and how the computer is used in representing and manipulating analog sound by an inferior digital counterpart without compromising quality. Artifacts of sampling followed by quantization, bit resolution, and the so-called CD-quality standard pave the way to important concepts in digital audio/sound including the ADSR, windowing, RMS envelope detection, wavetable synthesis and sample rate conversion with musical examples such as *Queen's Another One Bites the Dust* and time-domain time-stretching and compression seen in William Schottstaedt's *Leviathan*.

All through the book, especially in the beginning stages, DSP is introduced to the reader in a somewhat *subconscious* manner — via concentration and presentation of fun and interesting sound synthesis and analysis examples (within the context of DSP and the mathematics involved). For example, we start off by introducing sine waves, amplitude modulation, frequency modulation concepts, and the *clapping technique* to explain the impulse response which will help us get ready for the theory and practical use of convolution in audio. By the middle of the book the reader will hopefully be armed with enough confidence to get into the nuts and bolts of signal processing essentials — diving into topics such as difference equations, frequency response, z-transforms, and filters. As usual, DSP pertinent musical examples will follow the materials being presented at the end of each chapter and Matlab® implementations of topics such as comb-filtering and its kinship to physical modeling of plucked strings is also explored. In the final chapters we will fully venture into the frequency-domain, focusing on the Fourier transform with emphasis on the DFT (discrete Fourier transform) and its applications in various areas in computer music. We will at this point also revisit concepts introduced at the beginning of the book, such as up-sampling, down-sampling, decimation, harmonic distortion/dithering, and the Nyquist theorem to reinforce our knowledge of these very fundamental and important signal processing concepts by reviewing and viewing the same theories from different angles.

As a matter of fact, the practice of revisiting previously learned materials is a reoccurring theme throughout the book. This will hopefully further help us strengthen our understanding of principles presented during the course of the book. The final chapter discusses a number of classic vocoder algorithms and finishes off the book with an overview of research topics in the field of digital audio and computer/electro-acoustic music.

ACKNOWLEDGEMENTS

I started writing this book during the Hurricane Katrina period when we had to evacuate from New Orleans for a semester — the city was flooded and devastated in the fall of 2005. My wife, our cat, and I had escaped to West Lafayette, Indiana as we knew that we would not be going home for quite a while. During that time of uncertainty, I came in contact with World Scientific Publishing Co. and the rest, as they say, is history.

Writing this book has been an interesting, fun, demanding, and a very rewarding project for me. Needless to say, I owe great gratitude and debt to many people who have helped me directly and indirectly in the completion of this book.

First and foremost, I am thankful to my father Dong Kyu Park and mother Chung Suk Yang and to my two brothers Kihong Park and Jae Hong Park. They have always been there for me and supported me unconditionally.

I am also grateful to Zhiye Li and Travis Scharr who have contributed greatly in smoking out the bugs that seem to have found solace in the many pages of this book. Special thanks to Iroro Orife, Georg Essl, and Ted Coffey who contributed so much with their insightful feedback, criticisms, and proof-reading the materials (let me know when your books need some debugging). I am sure there are still a great number bugs crawling about, some big, some small, and some nearly invisible. However, if I imagine the version of the manuscript before my friends cleaned it up, I somewhat feel more at ease. Having said that, for any comments, suggestions, and bug reports please go to http://music.princeton.edu/∼park/dspBook.

I would also like to thank my teachers Jon Appleton, Perry Cook, Charles Dodge, Paul Koonce, Paul Lansky, Larry Polansky, and Dan Trueman who have influenced, taught, and advised me during and after my time spent in graduate school.

Thanks also to Chanu Yi and the folks at World Scientific Publishing Co. — Chunguang Sun and Chelsea Chin.

Last but not least, a big thank you goes to my wife Kyoung Hyun Ahn for her endless support, encouragement, and love... OK back to work.

Tae Hong Park, May 2009
New Orleans, LA
USA

ABOUT THE BOOK COVER DESIGN

The book cover shows a spectrogram of the letters D, S, and P along with various plots found in this book. The sound was generated using filters and plotted as three-dimensional spectrogram coded in MATLAB® (bookCover.m) which can be downloadable from http://music.princeton.edu/∼park/dspBook along with other MATLAB® programs (more details can be found in the appendix).

This page intentionally left blank

CONTENTS

Chapter 1

ACOUSTICS, HEARING LIMITATIONS, AND SAMPLING

1 Introduction

We will begin our journey into the world of digital signal processing by first tapping into topics with which we are (hopefully) already familiar. We will hence briefly introduce the sine tone, followed by discussion of important fundamental concepts in acoustics, talk about issues in human hearing and various interesting human hearing limitations and work our way to the concept of sampling and digitization of analog signals. As we shall see in subsequent sections, digitization and representing analog signals such as sound waves in the digital domain, is in large part possible due to deficiencies in our hearing system — the key word here is *limitation* which is critically exploited by digital systems. Our ears, eyes, and other input sensors do not have infinite resolution. Although this may seem like a serious shortcoming for our sensory systems, it may in reality be a blessing in disguise on many levels, including on the level of information overload from our brain's perspective and computation load from the computer's perspective.

2 The Sine Tone

The sine tone is introduced in this very first chapter for a couple of reasons: it is easy to implement, hear, and see on a computer, it is a sound that is probably familiar and it is a signal that encompasses fundamental characteristics of sound including amplitude, frequency, and phase.

Let's briefly refresh our memory regarding the sine wave by looking at Fig. 2.1. We probably remember a demonstration in our physics labs (or some educational television show) dealing with oscillation patterns — a pen that can only move perpendicularly oscillating up and down while at the same time moving a sheet of paper horizontally (left) as illustrated below. After a while and at the end of the demonstration, we were rewarded with an oscillatory pattern on the piece of paper — a sine wave. The oscillatory pattern can also be represented by a circle as shown on the left side of Fig. 2.1 rotating in a counterclockwise direction with constant angular velocity — the locations on the circle being represented in radians. For example, the peak will correspond to $\pi/2$ radians with an amplitude value of 1.0. This oscillatory pattern is characterized by three fundamental parameters — amplitude, frequency, and initial phase as expressed in Eq. (2.1) and summarized in Table 2.1.

$$y(t) = A \cdot \sin(2 \cdot \pi \cdot f \cdot t + \phi) \tag{2.1}$$

The amplitude is normalized to ± 1.0 and allows all real numbers in between. The number of times this oscillatory pattern repeats itself every second is defined as Hertz (Hz) where one full cycle of the pattern is referred to as the period. The initial phase parameter ϕ merely dictates where on the

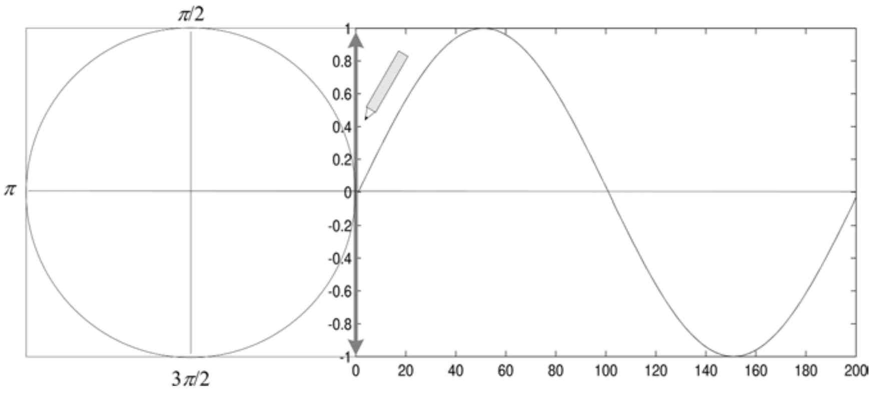


Fig. 2.1. The unit circle and sine wave.

Table 2.1. Sine oscillator parameters.

Parameter	Description
A	Amplitude (in our example -1.0 to $+1.0$)
f	Frequency in Hertz (Hz)
t	Time in seconds
ϕ	Initial phase in radians

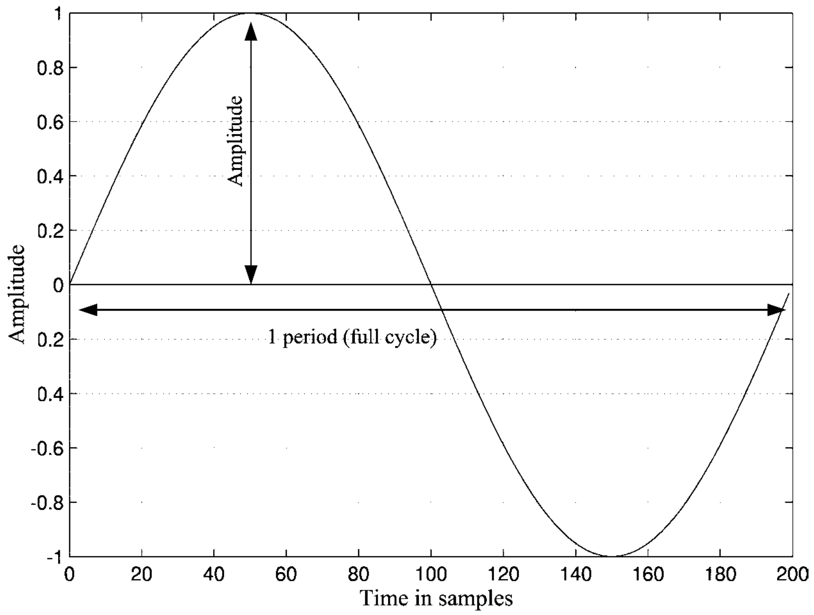


Fig. 2.2. Sine wave amplitude and a single cycle.

circle the pen starts rotating in a counterclockwise direction. For example, at $\phi = \pi/2$ radians or 90 degrees the sine will become a cosine. Figure 2.2 shows a sine wave making a full cycle.

I hope that you have MATLAB® available to run the following code as we will be using MATLAB® to show DSP examples, make sounds, and view plots. If you do not have it do not worry as coding is coding and the way it is coded will make sense especially if you have programming experience. If you do have MATLAB® you can try running the following code to hear a sine wave at $f = 440\,\text{Hz}$ (pitch equivalent of the A4 note), $A = 1$, duration = 1 second, and initial phase = 0.

```
y = sin(2*pi*440/44100*[0:44099]);
sound(y, 44100)
```

Code Example 2.1

Do not be concerned if you do not understand the above code at this time as it is meant for you to hear the sine tone and play around with — try changing some of the parameters discussed above. It will, however, become evident what those numbers mean once we cover sampling and the notion of discrete time.

3 Human Hearing and Its Limitations

Sound can be generally described via four very basic but important parameters — duration, pitch, amplitude, and timbre. All of these basic parameters are crucial in the of hearing sounds, whether be it sounds that one experiences in nature, music that blares through loudspeakers in concert halls, or when listening to a soft recording on a CD in your room on a quiet Monday evening. It so turns out that although it may not seem that obvious at first, we cannot hear everything that is around us and certain limitations prohibit us to perceive all of the subtleties that vibrate in the air. In this section, we will discuss some our hearing limitations.

3.1 *Duration*

Duration is quite straightforward and can be thought of the lifetime of a tone or sound object with units in milliseconds, seconds, minutes, hours, etc. However, even with duration, although upon initial glance it may seem overly simple on the surface, when coupled with human perception and psychoacoustics, things become very complex rather rapidly. For example, let's say we are listening to a song that is exactly 3 minutes long when heard at home alone on the stereo system in a quiet environment. Let's then assume you take the same CD and listen to the same exact track at a party from beginning to end with the exact same amplitude level. The song will probably not be perceived as having the same length, although in theory both versions are exactly the same and 3 minutes long! We will not go deeply into the psychoacoustic side of sound and will remain mostly in the realm of sound that can be measured but suffice it to say that basic topics such as duration for music and sound objects are not trivial at all.

3.2 *Pitch*

Pitch is a perceptual aspect of sound corresponding to periodic or more often quasi-periodic characteristics of sounds. A higher pitch corresponds to the perception of an increase in frequency and lower pitch a decrease in frequency and is generally measured in Hertz (Hz). Pitch and frequency (f) have a reciprocal relationship with time (t):

$$f = 1/t \tag{3.1}$$

Try singing *ahhh* or try using MATLAB® Code Example 2.1 to play various pitches by changing the frequency parameter. Of course not all musical sounds have pitch — for example the tambourine or the kick drum do not have pitch per se. Although in theory, kick drums can be tuned to a high enough resonant frequency for it to be perceivable as pitch, this is for most instances not even the desired effect especially in popular music — the timpani which is different from the kick drum is, however, tuned to specific pitches for example. In case of the kick drum scenario, imagine the poor drummer of a band having to "tune" the whole drum set (not just the kick) every time a new song is to be played at a concert due to key changes ... Pitch is commonly referred to as the fundamental frequency, but there are cases where there is no fundamental frequency and we still can perceive its pitch which does not really exist. This is referred to as the *missing fundamental* phenomenon. A healthy person will have the capability to hear frequencies from approximately 20 to 20,000 Hz. It is a common mistake to think that we can perceive pitch in the same range. This is, however, not the case. The exact upper and lower limits for pitch perception vary from person to person, but frequencies from 200 Hz to 2,000 Hz (Dodge, Jerse 1985) comprise the region of greatest acuity and sensitivity to change in frequency (there is a good reason why the piano starts at 27.50 Hz and goes up to only 4186 Hz). As the frequency is increased beyond 4,000 Hz the aspect of pitch slowly transforms into the perception of "high frequency" rather than that of pitch. Notice the distinction here — as we ascend towards the perceivable upper pitch limit we will tend to sense the presence of high frequency content rather than hear a precise pitch value. A similar yet different phenomenon occurs on the lower limit of frequency/pitch perception. When we start off a periodic pulse signal at 1 Hz and slowly increase it to around 30 Hz we will go through three perceptual frequency regions — 1) a region where we can actually count or almost count the number of pulses per second sometimes eliciting a sense of rhythm; 2) when the pulses become faster to the degree that we cannot count them anymore, at which point we start hearing these pulses as a sensation of roughness; 3) last region where the roughness slowly turns into pitch. There are interesting theories in the way we perceive pitch, the two main ones being the so-called *place theory* and the other using theories in *phase-locking* of nerve fibers. Suffice it to say that for now, humans cannot hear all pitches nor can we hear all frequencies — there are upper and lower soft limits.

3.3 *Amplitude and sound levels*

Amplitude refers to the strength of a signal. The larger the amplitude the louder it will seem and vice-versa. The unit used to represent the strength for audio signals is decibels (dB). There are various similar yet different names when it comes to addressing amplitude in audio and music in particular, including *sound intensity level* (SIL), *sound pressure level* (SPL), and *loudness.* It is not uncommon that these terms are sometimes used interchangeably, potentially causing confusion. The confusion perhaps arises because loudness like duration/time is seemingly straightforward in concept and we all seem to know how to crank up the volume dial (often buttons nowadays) on our stereo to get a louder sound. In reality, however, "what we hear is not what we hear" so-to-speak. I will briefly discuss the above terms in the following sections.

3.3.1 *Sound intensity level (SIL)*

Imagine a spherical sound source with radius r_{small} with an infinite number of infinitely small loudspeakers making up its surface, radiating equally in all directions (isotropic) in some medium such as air. Now, if we engulf this spherical sound source with another bigger sphere — the same sphere with a larger radius r_{big} sharing the same origin we can visualize the sound hitting the inner shell of the outer sphere. The power of the radiated sound source passing one square meter is measured in $watts/m^2$ and is defined as sound intensity. We also know from experience the further away we are from a sound source the softer the sound will become. Specifically, it adheres to the following inverse square relationship:

$$I = \frac{P}{4\pi r^2} \tag{3.2}$$

P is the acoustic power of the sound source and r the radius or distance from the sound source. This basically means that as the distance (r) from the sound source increases the intensity of the sound source decreases exponentially and not linearly. The sound intensity level (SIL) is defined according to Eq. (3.3).

$$dB_{SIL} = 10 \cdot \log_{10}(I_1/I_0) \tag{3.3}$$

The term decibel itself is derived from the name Alexander Graham Bell who was instrumental in developing the telephone. The *bel* is defined as:

$$bel = \log_{10}(I_1/I_0) \tag{3.4}$$

As you can see, the only difference between the *bel* and decibel in (3.3) and (3.4) is the scalar multiplier 10. The *deci* (stands for 10 in Latin) was added later largely in part to help in the usability and readability of the *bel* measurements itself.

In the above equations, I_1 refers to the intensity level of interest and I_0 a reference level called the threshold of hearing (measured as power) — the intensity threshold where we will start to hear a 1 kHz signal if we were to increase the intensity from complete silence to this threshold value (I_0). The reason a 1k Hz signal is used will become evident below when the Fletcher-Munson curve is introduced — but let's for the time being assert that we do not hear all frequencies equally well and hence a representative frequency needs to be selected. Obviously in other situations I_0 can be some other constant but in the area of audio and sound it is defined as the threshold of hearing:

$$I_0 = 10^{-12}\ Watts/m^2 \tag{3.5}$$

One very important characteristic about sound intensity level is that it is a ratio and not an absolute value. To put this into perspective, if one wants to increase the SIL to twice its value from say 10 dB_{SIL} to 20 dB_{SIL} we would need to increase the I_1/I_0 ratio so as to square it — one's gut reaction would probably be multiplying the source I_1 by 2 (let's assume I_0 is equal to 1 for simplicity) but this will yield $10 \cdot \log_{10}(2 \cdot 10) = 13.01$ dB_{SIL} which is not twice of 10 dB_{SIL}. However, if we increase the intensity 10-fold in our example (square the I_1/I_0 ratio), we increase the resulting dB_{SIL} level to 20:

$$10 \cdot \log_{10}(10 \cdot 10) = 10 \cdot \log_{10}(10^2) = 20 \cdot \log_{10}(10) = 20$$

3.3.2 *Sound pressure level (SPL)*

Another popular *dB* version often encountered in audio is SPL (sound pressure level) and its corresponding dB_{SPL} which is defined as shown below in Eq. (3.6).

$$dB_{SPL} = 20 \cdot \log(A_1/A_0) \tag{3.6}$$

In this case, the difference between SIL and SPL is namely the multiplier 20 instead of 10 and change in parameter names — amplitudes A_1 and A_0. A is in Pascals (Pa) used to measure air pressure and the threshold of hearing $A_0 = 2 \times 10^{-5}\,\text{N/m}^2$ at 1,000 Hz. The above derivation comes from the fact that $I \propto A^2$, in other words I (power, watts) is proportional to the square of the amplitude.

$$20 \log_{10} \frac{A_1}{A_0} = 10 \log_{10} \left(\frac{A_1}{A_0}\right)^2 = 10 \log_{10} \frac{A_1^2}{A_0^2} \tag{3.7}$$

Figure 3.1 depicts a diagram showing some of the typical *dB* levels. Notice that our hearing system has limitations which are essential characteristics that we will exploit for digital systems and DSP.

Some common *dB* levels you may have encountered or heard audio engineers use is 0 $dB_{SPL}(A_1 = A_0)$ and 6 $dB_{SPL}(2 \cdot A_1/A_0)$ which now we understand what is meant by them.

3.3.3 *Just noticeable difference (JND)*

Just noticeable difference (JND) is defined as the minimum change in amplitude required in order for the listener to detect a change in amplitude (there is also an equivalent for frequency, the just noticeable difference in detectable frequency). This means that human hearing systems have further limitations. The JND is usually measured using pure sine tones and actually changes depending on what frequency is presented to the subject and at what *dB* level the tone is presented. The general rule of thumb, however, is that the JND is about 1 *dB*. Figure 3.2 shows a 1 kHz tone and its JND characteristics. This is probably why we nowadays often see volume controls on stereo systems that feature discrete steps for increasing and decreasing the “volume” rather than having a continuous dial.

3.3.4 *Equal loudness curve*

We briefly mentioned that humans do not perceive all frequencies equally well. For example, if we were to listen to a 100 Hz sine tone and a 1 kHz

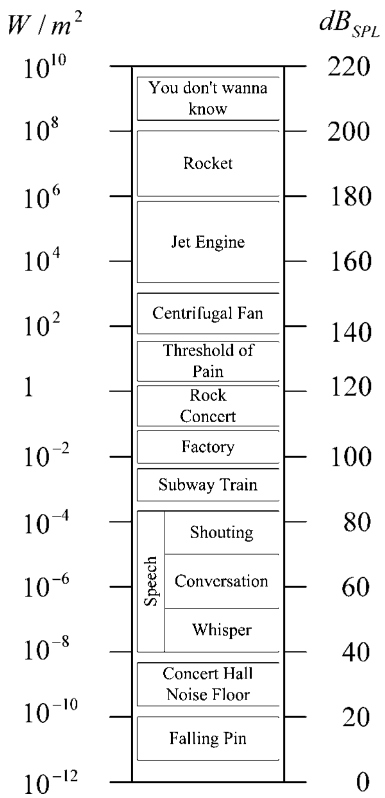


Fig. 3.1. Typical *dB* levels.

sine tone, we would not perceive them to have the same loudness even if we did not change the amplitude or the "volume" dial on your amplifier. Try using the previous MATLAB® code to play a 100 Hz sine tone vs. a 1 kHz sine tone without changing the amplitude parameter by keeping it at 1.0 for example.

This bias of perceiving the strength of a signal in our hearing system is illustrated in Fig. 3.3 known as the Fletcher-Munson curve or the equal loudness curve. The graph may at first inspection look a bit strange, so let's use a 40 loudness level as an example to try to grasp the concept behind the Fletcher-Munson curve. Let's pretend that we have two control knobs — one controlling the sine tone's frequency and one controlling the sine tone's dB_{SPL} level. If we were to start at 20 Hz and sweep the frequency gradually at a linear pace all the way up to around 10,000 Hz and not alter the dB_{SPL} level dial, we would actually perceive the tone to become louder

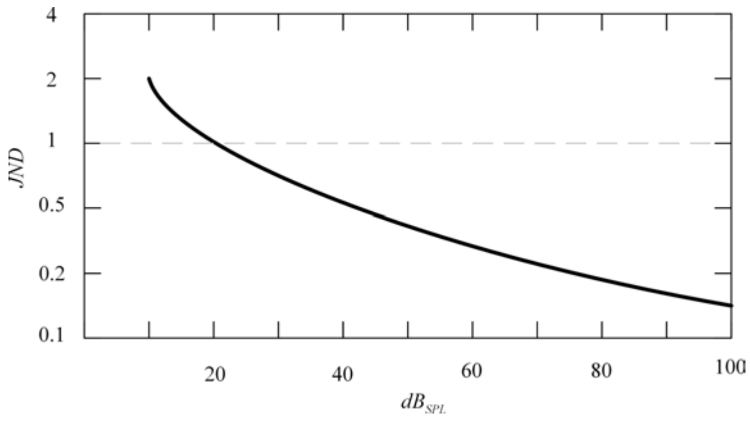


Fig. 3.2. JND for 1 kHz tone (Zwicker, Fastl 1999).

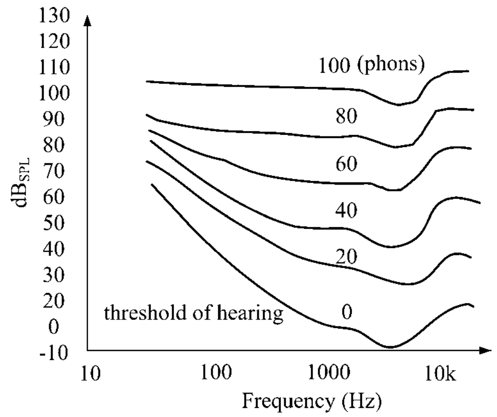


Fig. 3.3. Fletcher-Munson curve (Fletcher, Munson 1933).

up until around 3,000 Hz and then softer when reaching 10,000 Hz — even when keeping the "volume" dial constant. In order for us to perceive the same loudness level of, say, 40 throughout the sweep from 20 to 10 kHz we would need to actually increase the volume knob (dB_{SPL} level) to around 80 dB_{SPL} in the beginning part up to 20 Hz, decrease it to 40 dB_{SPL} at around 1,000 Hz, and again increase it to around 50 dB_{SPL} at approximately 10 kHz to render a perception of "equal loudness." This essentially is what the Fletcher-Munson curve outlines. This loudness level also has a special name called *phons* and as you probably already noticed, the shape of the curves change considerably with respect to phons where at the threshold of pain it is almost a straight line — at a loud rock concert all frequencies will be perceived pretty much equally. Maybe that's one of the reasons rock concerts are so loud ...

3.4 *Auditory masking*

Imagine being in a quiet room watching a tennis match on TV between Federer and Nadal at some dB level that's just perfect. For some reason or another, your roommate suddenly feels the urge to engage in some very important vacuum cleaning project. All of a sudden, the sound coming from the TV that was so crisp, clear, and just the right dB level is now overwhelmed by the mechanical sound produced by the 15-year old vacuum cleaner. You cannot hear anything — the sound from the tennis match is *masked* by the sound from the vacuum cleaner. This scenario where one sound source overwhelms another is referred to as *masking*.

There are basically two types of masking principles: *simultaneous masking* and *non-simultaneous masking*. Simultaneous masking refers to a situation when two sound events, the *masker* (vacuum cleaner) and the *maskee* (tennis game), occur at the same time instant. You can probably imagine that the threshold for hearing the maskee without the masker will be lower than when the maskee is present and vice-versa. That is, in a very quiet room without a masker, you will not need to turn up the volume on the TV too much, whereas a room with lots of noise the signal to noise ratio (SNR, see Section 5.1) will have to be improved by increasing the level of the television program. What is interesting is that this threshold of hearing of the maskee is not only a function of the maskee's and masker's intensity levels but also frequency levels. Figure 3.4 shows a general plot depicting the behavior of the masking threshold (adapted from Gelfand 2004) where the masking frequency is kept constant at f_{mask} Hz and the maskee frequency is altered along the frequency-axis (intensity levels are kept constant for both maskee and masker). What Fig. 3.4 tells us is that as the maskee (TV) frequency is shifted away from the masking frequency f_{masker}, the less of an effect the masker will have in overwhelming the maskee sound source. When the maskee frequency is equal to f_{masker}, the most noticeable masking effect takes place — you will need to really crank up the volume of your TV set if you want to hear it while the vacuum cleaner is running, especially when they are both at the same frequency range. However, if the vacuum cleaner's frequency range (f_{masker}) is much lower

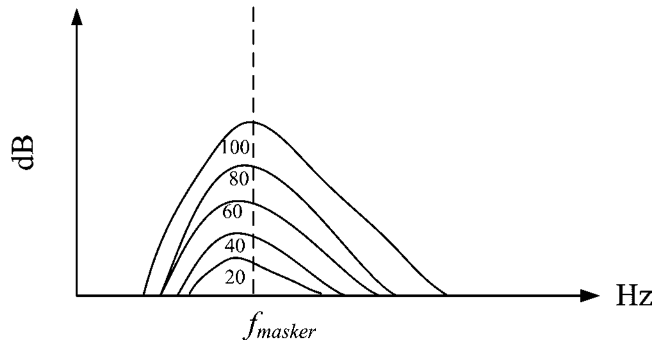


Fig. 3.4. General characteristics of masking threshold.

than the TV sound, you will not need to increase much the volume button on your remote control. For example, if the vacuum cleaner is running in another room separated by a wall (this is referred to as filtering discussed in Chapter 7) much of the high frequency content will vanish (as well as the intensity level) and your TV experience, from the sonic point of view, will not be affected much.

Unlike simultaneous masking, non-simultaneous masking (also known as *temporal masking*) refers to masking principles when the masker and maskee are out of synchrony and do not occur at the same time. Within temporal masking we also have what is referred to as *post-masking* and *pre-masking.* Post-masking intuitively makes sense if you consider the following scenario. Let's say we are crossing a busy street in the heart of Seoul minding our own business (too much) without noticing that the pedestrian light has just turned red. This guy sitting in a huge truck gets all mad and hits on his 120 dB car honk that just blows you away (without damaging your ears luckily!). Even though the honk only lasted one second, you are not able to hear anything that occurred for another 200 milliseconds or so. This is called post-masking. Pre-masking is a bit more interesting. Let's consider the same situation where you are crossing the street as before. Everything is the same including the post-masking effects, that is, sounds that will be masked out *after* the loud honk. But that's not the end of the story. It so happens that some sounds that occur *before* the masker, if they happen close enough to the start time of the maskers acoustic event (honk) will also not be heard. In other words, acoustic events that occur *prior* to the masker will also be erased without a trace — this is called pre-masking. Pre-masking is also a function of intensity and frequency of both the masker

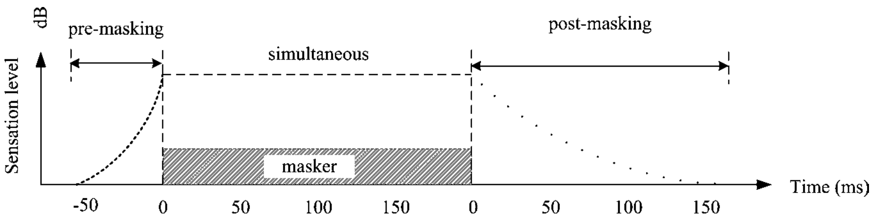


Fig. 3.5. Masking characteristics.

and maskee. A summary of the three types of masking types are shown in Fig. 3.5 (after Zwicker 1999).

A group of audio compression algorithms that exploit the limitation and psychoacoustic tendencies of our hearing system are referred to as *perceptual codecs.* In these types of codecs (coder-decoder), a great example being MP3, masking characteristics play an important role in allowing the reduction of data for representing a given audio signal, thus improving download and upload performance for our listening pleasures.

4 Sampling: The Art of Being Discrete

Up until now we have been emphasizing on various limitations in our hearing system which brings us to the concept of sampling and digital representation of analog signals. Simply put sampling is defined as a process of analog to digital conversion through devices called ADCs (analog to digital converters) whereby a continuous analog signal is encoded into a discrete and limited version of the original analog signal. Think of motion picture movies — when we watch a movie in cinemas, we perceive through our visual sensory organs continuous and smooth change in moving objects, people, and anything else that is in motion. This is, however, an illusion as in reality there are a set number of pictures known as frames (24, 25, and 30 frames per second are common) that are projected on the silver screen. Like our hearing system our eyes have a finite sampling capacity or sampling rate and information at a higher rate than this sampling rate is not perceived. There is thus generally little need to have a movie camera take 100 snap-shots per second nor is there a need to project each frame for a duration shorter than 1/24th, 1/25th or /30th of a second as we will not be able to tell the difference. That is, 50 frames or 100 frames per second or conversely 1/50th or 1/100th of a second for each frame becomes redundant and more or less unperceivable. Our perceptual organs and brains have limited ability in capturing and registering time critical events. However, if we want to slow down movies, however, that would be a totally different story as more frames per second will render a smoother motion picture experience.

The same concept applies when sampling sound. As we have learned previously, we only hear up to approximately 20 kHz. Furthermore, we cannot hear every infinite subtlety in dynamic change either. When we say that we only hear up to 20 kHz, we mean that if there is a sine wave that oscillates above 20,000 cycles per second, we would not be able to perceive it. Our limitations in hearing are actually a blessing in disguise, at least when viewed from a shear number crunching or processing perspective, whether be it a machine or our brains. In other words, the time between each sample need not be infinitely small (infinity is something problematic on computers) but in fact only need be small enough to convince our ears and brain that what we have sampled (digitized version of the sound) is equivalent to the original analog version (non-digitized version of the sound). To get an idea what this all means let's look at Fig. 4.1.

Although the plot looks very smooth and perhaps even continuous, this is not the case. The actual sampled version in reality looks more like Fig. 4.2 (here we have zoomed into half of the sine wave only). Upon closer inspection we see that it is not smooth at all and is rather made up of

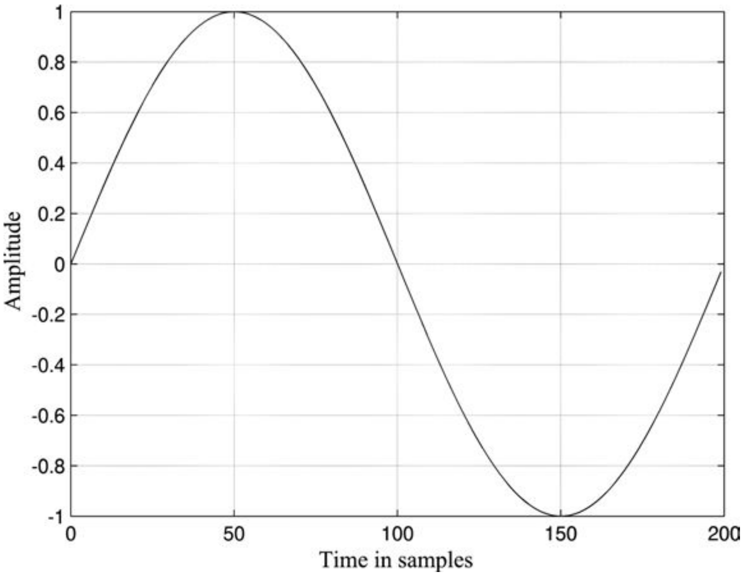


Fig. 4.1. Full 1 Hz sine wave.

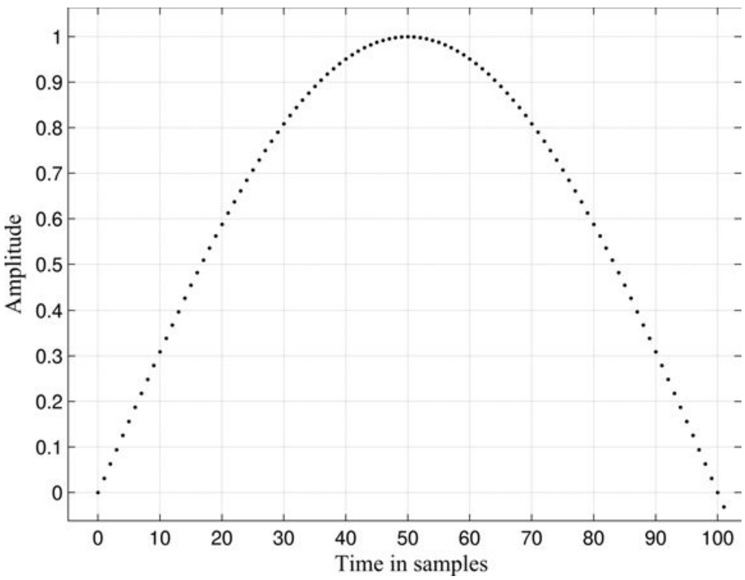


Fig. 4.2. About half of 1 Hz sine wave sample at $f_s = 200\,\text{Hz}$

discrete points outlining the sine wave. It's somewhat analogous to pixel resolution of your computer monitor. When viewed from afar (whatever afar means) the digital picture you took last year at the top of the mountain looks smooth and continuous. But viewed with your nose up against the monitor and zoomed-in, you will inevitably see the artifacts of discreetness in the form of pixel resolution.

In this example, we actually used 200 discrete samples to represent one full cycle of the sine wave where the 200 samples correspond to exactly 1 second duration. Since the sine wave makes one full resolution in one second, it must be a 1 Hz signal by definition. We say that the sampling rate, commonly denoted as f_s, for this system is $f_s = 200\,\text{Hz}$ or 200 samples per second. We can also deduce that each time unit or grid is $1/f_s = 1/200$th of a second (in this example). This time unit is referred to as the sampling period and usually is denoted as T. Hence, we have the following relationship between sampling rate f_s (Hz) and T (sec):

$$f_s = 1/T \tag{4.1}$$

or

$$T = 1/f_s \tag{4.2}$$

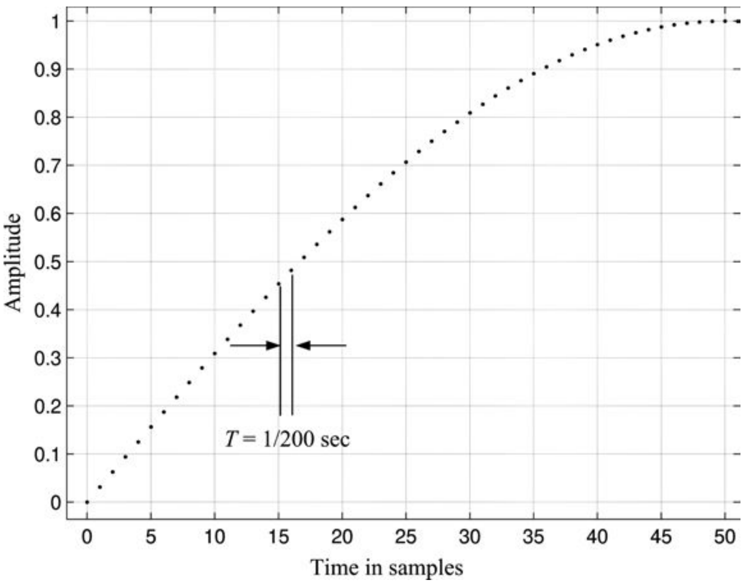


Fig. 4.3. Zoomed-in portion of sine wave and sampling period T.

This is further illustrated in Fig. 4.3 (quarter plot of the same sine wave) where we notice that the spacing between each sample is equal to T, 1/200th of a second. Figure 4.4 shows the same half sine tone we have been using but sampled at a quarter of the previous sampling frequency with $f_s = 50\,\text{Hz}$. It clearly shows that the number of samples representing the half sine wave has reduced by 1/4.

We may also at this point notice that time is now discrete — with sampling frequency of 200 Hz we have $T = 1/200 = 0.005$ seconds or 5 milliseconds (ms) which means that we cannot for example have any data at 2.5 milliseconds or any other value that is not an integer multiple of T as shown below (if we start at $t = 0$).

$$t = n \cdot T \tag{4.3}$$

Here n is the integer time index. We are now ready to express each time index or sample of the sine wave (or any digital signal for that matter) in the following manner:

$$y(t) = \sin(2 \cdot pi \cdot f \cdot t) \tag{4.4}$$

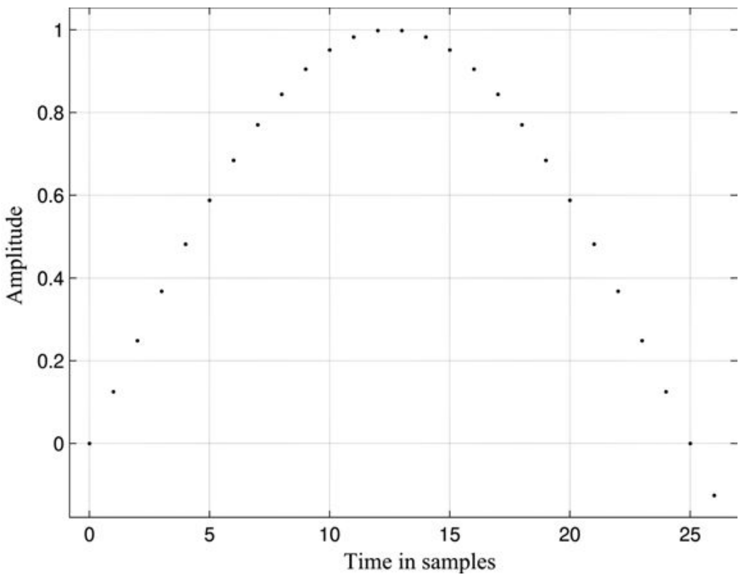


Fig. 4.4. Approximately half duration of 1 Hz sine wave sample at $f_s = 50$ Hz.

and replacing t with Eq. (4.3) we have:

$$y[n \cdot T] = \sin(2 \cdot pi \cdot f \cdot n \cdot T) \tag{4.5}$$

Again, n is again an integer number denoting the time index. For instance the 0th sample in our example in Fig. 4.5 is:

$$y[n \cdot T] = y[0 \cdot T] = y[0] = 0 \tag{4.6}$$

The 7th sample of our sine wave ($n = 6$ since we started counting from 0) is:

$$y[n \cdot T] = y[6 \cdot T] = 0.2 \tag{4.7}$$

Note that for analog signals we use the ( ) (parentheses) and for digital signals we use the [ ] (square) brackets to clearly differentiate continuous and discrete signals. The representation of each time "location" which is now discrete can be done for every sample since we know the sampling rate f_s and hence the period T. The sampling period T is customarily omitted for notational convenience and $y[n \cdot T]$ becomes $y[n]$. Going back to the MATLAB® code that we started off the chapter with we can now see how the continuous time sine wave (t) becomes the discrete time sine wave ($n \cdot T$).

In other words:

$$y(t) = \sin(2 \cdot pi \cdot f \cdot t) \tag{4.8}$$

setting $t = n \cdot T$ we get

$$y|_{t=nT}(t) = y[nT] = \sin(2 \cdot pi \cdot f \cdot n \cdot T) \tag{4.9}$$

remembering that $T = 1/f_s$ the MATLAB® code becomes

$$y[n] = \sin(2 \cdot pi \cdot f \cdot n \cdot T) = \sin\left(2 \cdot pi \cdot f \cdot \frac{n}{f_s}\right) \tag{4.10}$$

and since n is an integer value corresponding to the sample/time index, by setting $n = [0 : 44099]$ we get a 1 second sine wave (the [0:44099] notation produces sequential integer numbers from 0 to 44099):

$$y[0 : 44099] = \sin\left(2 \cdot pi \cdot \frac{f}{f_s} \cdot [0 : 44099]\right) \tag{4.11}$$

By plugging in $f = 1$ we get a 1 second, 1 Hz signal sampled at $f_s = 44,100\,\text{Hz}$.

4.1 *Sampling theorem*

In our sine wave MATLAB® example we used a sampling rate of 44,100 Hz meaning we used 44,100 samples per second to represent the sine wave. We could, however, have used different sampling rates as well — or could we have done this? The answer actually is dependent on what sine wave frequency we want to generate or more specifically what the highest sine wave frequency we would like to produce. That is, what sampling rate would be adequate in representing an analog signal (single sine wave in our example) without audible artifacts or distortion? As we have previously asserted we only hear up to 20,000 Hz and hence a good assumption at this point might be $f_s = 20,000\,\text{Hz}$ (remember that $f = 1/t$) equivalent to 1/20000th of a second or conversely, 20,000 samples per second. Although this is indeed a good guess, it turns out that we need at least twice that amount. More accurately stated, the sampling frequency must be at least twice the highest frequency component in an audio signal. This is known as the *sampling theorem* where $f_s/2$ is referred to as the *Nyquist frequency* named after Harry Nyquist. The sampling theorem is shown below in (4.12) and (4.13):

$$f_{\text{max}} < \frac{f_s}{2} \tag{4.12}$$

or

$$f_s > 2 \cdot f_{\text{max}} \tag{4.13}$$

For a 1 Hz sine wave that would mean that we only actually would require a sampling rate greater than 2 Hz or a minimum of 3 samples per second since it is a signal with only one frequency component. In our MATLAB® example where we used a 440 Hz sine wave, the minimum sampling rate required would be a number greater than $440 \cdot 2 = 880$ Hz — a minimum of any frequency greater than 880 samples per second must be selected. Now, since our hearing limitation is around 20,000 Hz we know from the sampling theorem that we would need a sampling rate greater than 40,000 Hz. It so happens that the sampling frequency used for the standard compact disc is 44,100 Hz at 16 bits per channel (2 channels exist for CD). Why 44.1 kHz? The urban legend goes something like this: the main industrial heavyweights in developing the compact disc Sony and Philips (other collaborators included CBS/Sony and Polygram), were in the midst of determining a standard for the audio CD back in the 1980s. Seemingly at some point in time during that historic period in the 80s, it came down to choosing a sampling rate between 36 kHz vs. 44.1 kHz. Many arguments for and against each sampling rate standard went back and forth, but when the arguments had ceased, the 44.1 kHz was chosen as the standard because:

Human hearing goes up to 20,000 Hz

Sampling theorem requires $f_s > 40,000$ Hz

Add extra frequency padding to 40 kHz sampling frequency

Beethoven's 9th Symphony has to fit into one CD

Interestingly enough, one criteria for choosing the current CD specifications is that Sony and Philips agreed that Beethoven's 9th *Symphony* be used as the benchmark for how long a CD should play. Thus, after conducting some research they found that the performances of *Alle Menschen werden Brüder* were between 66 and 74 minutes long. Ironically the first CD title that CBS/Sony produced was not Beethoven's 9th but Billy Joel's 52nd

Street and the rest, as they say, is history. Nowadays we often see 24 bits as a standard way to store the amplitude values and sampling rates at 96 kHz and even up to 192 kHz. There is much discussion going on regarding the necessity for 192 kHz sampling rates and many arguments seemingly are in favor and some against such necessities. We'll leave it at that since this topic itself would require another whole chapter or two at least to address the technical and non-technical issues involved.

4.2 *Aliasing*

At this point you should be asking yourself — what if I use a sampling rate that is lower than twice the maximum frequency component of the audio signal. That is, when $f_s/2 < f_{\text{max}}$? The answer to that is that you will experience a particular digital artifact called *aliasing*. The resulting digital artifact is quite interesting and presents itself as a distorted frequency component not part of the original signal. This artifact caused by *under-sampling* may sometimes be intriguing and perhaps useful in the context of musical composition, but when the objective is to acquire an accurate representation of the analog audio signal in the digital domain via sampling, this will be an undesirable artifact. We will see in more detail how aliasing occurs in Chap. 8 after learning about the frequency-domain and the Fourier transforms. For now, let's try to grasp the idea from a more intuitive time-domain approach.

Let's consider a 1 Hz sine wave sampled at $f_s = 100\,\text{Hz}$ (100 samples per second) as shown in Fig. 4.5. We know that a 1 Hz sine wave by itself sampled at $f_s = 100\,\text{Hz}$ meets the sampling theorem criteria, the Nyquist limit being at 50 Hz — any sine wave that is below 50 Hz can be unambiguously represented in the digital domain. So there should be no aliasing artifacts in this particular case ($f = 1\,\text{Hz}$). Looking at the plot we clearly see that a 1 Hz sine wave makes one full cycle in 1 second and the number of samples representing this one 1 Hz sine wave is 100 samples with a period $T = 1/100\,\text{sec}$.

Let's now increase the frequency of the sine wave to 5 Hz and keep everything else unaltered — the results are shown in Fig. 4.6. Once again we note that T and the number of samples stay unchanged at 1/100th of a second and 100 samples, but since it is a 5 Hz sine wave we will get 5 full cycles in one second rather than just one. Note also that previously we had the full 100 samples to represent one entire cycle of the 1 Hz sine wave and now we only have $100/5 = 20$ samples to represent one full 5 Hz cycle.

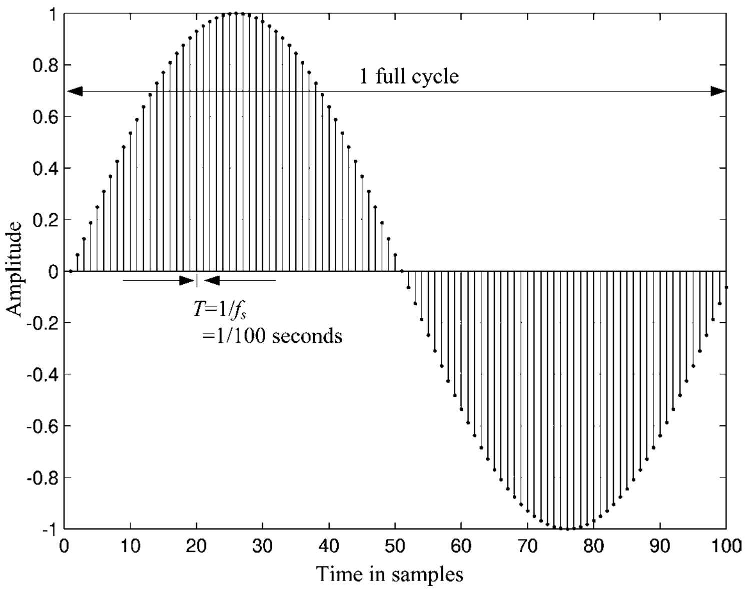


Fig. 4.5. Sine wave with $f = 1\,\text{Hz}$ and $f_s = 100\,\text{Hz}$.

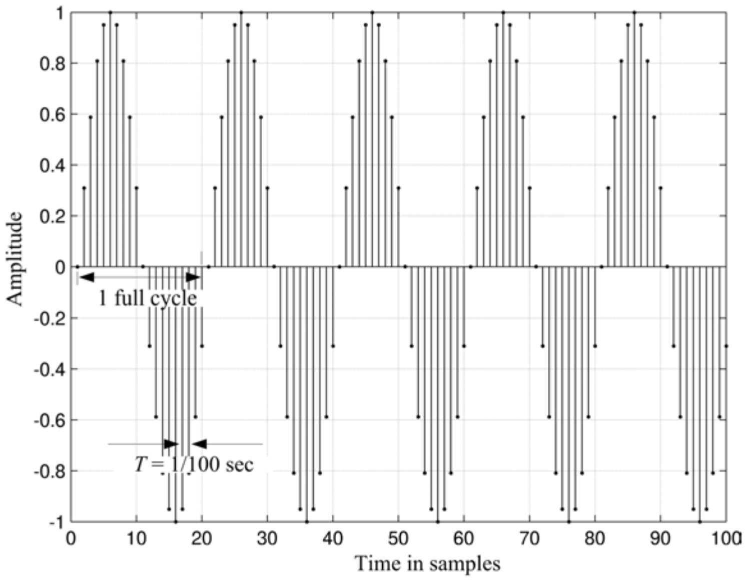


Fig. 4.6. Sine wave with $f = 5\,\text{Hz}$ and $f_s = 100\,\text{Hz}$.

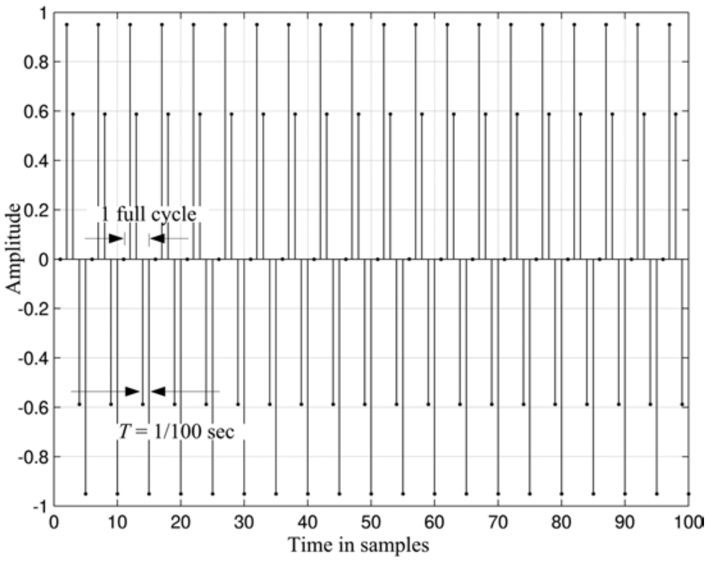


Fig. 4.7. Sine wave with $f = 20\,\text{Hz}$ and $f_s = 100\,\text{Hz}$.

Let's further increase the frequency to 20 Hz. The results are as expected as shown in Fig. 4.7. We again observe that one full cycle of the 20 Hz sine wave is further being deprived of samples to represent one full cycle — we only have a mere 5 samples for one full cycle for the 20 Hz sine tone. Although it still kind of looks like a sine wave and certainly sounds like a sine wave (try it with the MATLAB® code by changing the sampling rate and frequency) it is quickly loosing its characteristic sine tone appearance.

Now let's push the envelope a bit further and bring the frequency up to the edge of the Nyquist frequency limit of 50 Hz (in actuality the sine tone is not as pretty as seen in Fig. 4.8 but for the sake of argument let's pretend that it is). Note that in Fig. 4.8, the sine tone that is just below 50 Hz is has been reduced to a bleak sequence of plus 1.0 and minus 1.0 samples, each pair of change in polarity representing a full cycle of this *almost* 50 Hz sine tone. I suppose that one could look at one cycle and be really generous and say — I guess it still possesses the bare minimum features of a sine wave (again this is not the way it really works but let's pretend for just a bit longer and see what the voilà is at the end).

We have at this point pretty much stretched our ability to represent a sine wave with a sampling rate of 100 Hz and note that a sine wave with any higher frequency would be impossible to represent. At the edge 50 Hz we

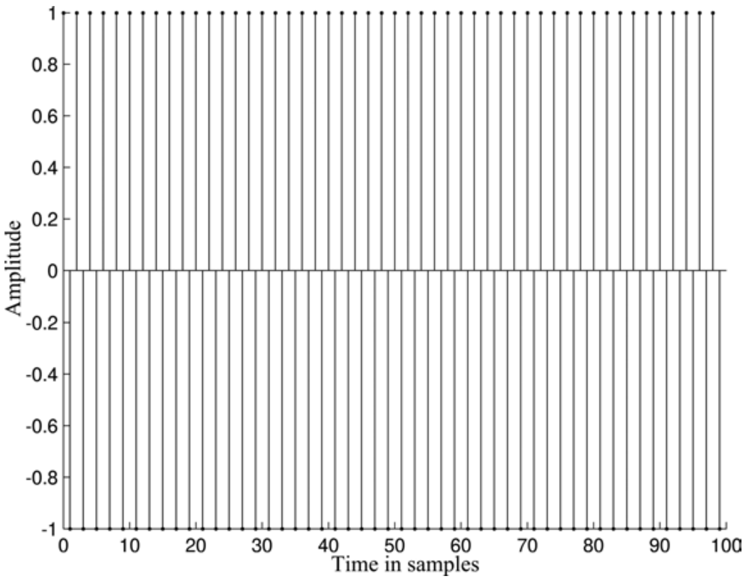


Fig. 4.8. Sine wave with f "almost" 50 Hz and $f_s = 100$ Hz.

were using two samples to *mimic* a sine wave but any frequency higher than that would mean one sample or less (if we follow the trend of decreasing number of samples with increase in frequency) per one whole sine cycle. It would not be possible to describe a sine wave with just one sample (and certainly not with 0 samples!) as a sine wave should at least have the characteristic of two oppositely peaking amplitudes. However, if we had a higher sampling rate of say 200 Hz we would be able to represent a 50 Hz or 60 Hz sine wave quite easily as we would have more samples per second to play with. This is not the case with a 100 Hz sampling rate where we've now come to a dead end. This is the pivot point where aliasing occurs. To illustrate the artifacts of aliasing let's go beyond the Nyquist limit starting with 50 Hz and see what the resulting plot actually looks like. This is shown in Fig. 4.9.

What just happened? In Fig. 4.9 we note that the 50 Hz sine is exactly the same as a 0 Hz sine, the 95 Hz sine (Fig. 4.10) is an inverted version (or phase shifted by 180^o) of the original 5 Hz sine, and the 105 sine (Fig. 4.11) is exactly the same as the 5 Hz sine (no inversion). If we disregard the inversion aspect of the sine waves in the figures, frequencies that are above the Nyquist frequency are literally aliased towards a lower frequency sine wave — this artifact that results from a process called under-sampling

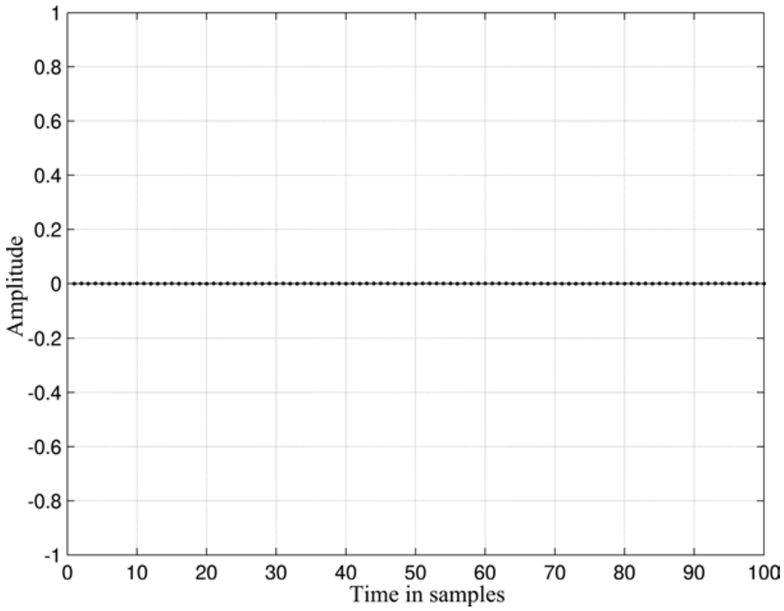


Fig. 4.9. Sine wave at $f = 50\,\text{Hz}$ and $f_s = 100\,\text{Hz}$.

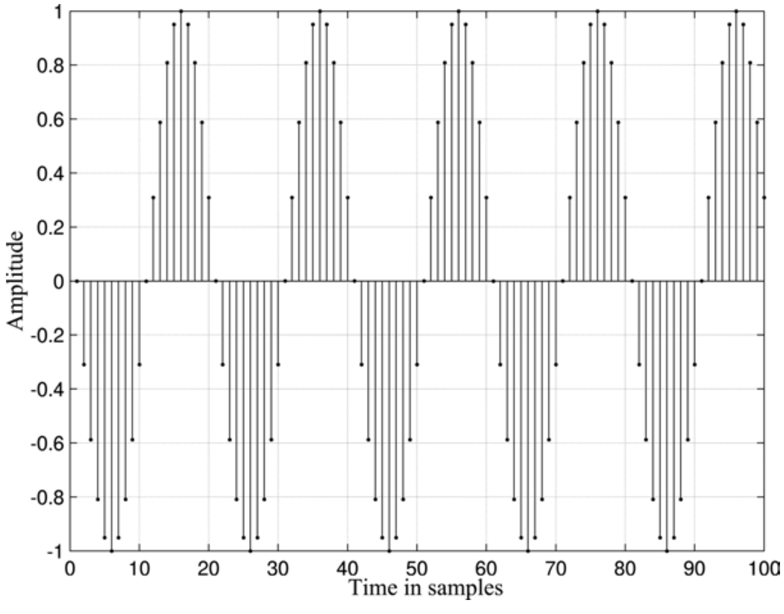


Fig. 4.10. Sine wave at $f = 95\,\text{Hz}$ and $f_s = 100\,\text{Hz}$.

is referred to as aliasing. The 95 Hz has aliased back to the 5 Hz sine (with inversion of 180° or π) as has the 105 Hz sine (without inversion). An interesting characteristic of our hearing system is that we do not hear phase differences when listening to a single sine wave and hence the 5 Hz and the 95 Hz and 105 Hz signal actually sound exactly the same to us even though there are phase shift differences. However, if a signal is comprised of a number of sine waves with different phases, the perception of the complex sound due to *constructive and deconstructive interference* is different (deconstructive and constructive interference is discussed in Chap. 4 Sec. 4). Try changing the frequency in MATLAB® to convince yourself that higher frequencies that go beyond the Nyquist limit exhibit this aliasing phenomenon to a lower frequency. The above explanation of aliasing in terms of the decreasing number of samples per period is one way to intuitively look at aliasing until we develop more powerful tools in Chaps. 6 and 8. Figure 4.12 illustrates this idea of aliasing.

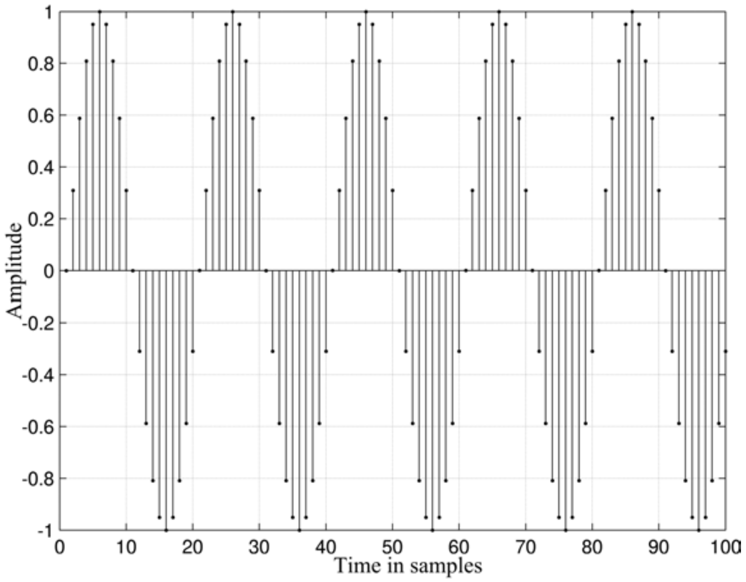


Fig. 4.11. Sine wave at $f = 105\,\text{Hz}$ and $f_s = 100\,\text{Hz}$.

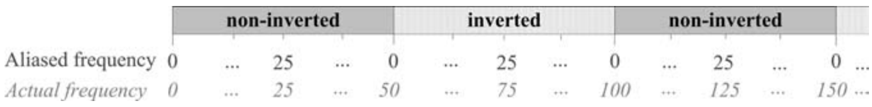

non-inverted
inverted
non-inverted
Aliased frequency 0 ... 25 ... 0 ... 25 ... 0 ... 25 ... 0 ...
Actual frequency 0 ... 25 ... 50 ... 75 ... 100 ... 125 ... 150 ...

Fig. 4.12. Aliasing pattern of a single sine wave with $f_s = 100\,\text{Hz}$.

To recap, the frequencies that are beyond the Nyquist limit will always alias downwards to a lower frequency. What makes things very interesting is that musical signals are rarely just made up of a single sine tone, but are rather made up of an infinite number of sine tones. One can imagine how that may contribute to the net cumulative aliasing result when using inappropriate sampling rates.

5 Quantization and Pulse Code Modulation (PCM)

Until now we have pretty much concentrated on the frequency component of sound and not much attention has been given to the issues pertaining to what happens to the amplitude values of each sample during the analog to digital conversion process. Here too, the topic of concern is the levels of inferiority in representing the original analog counterpart in the digital domain. The quality, resolution, and accuracy of the amplitude of sampled analog signals are determined by the bit depth impacting the quantization error. By quantization error I mean the error (ε) between the discrete digitized and analog amplitude values as seen in Eq. (5.1) where n is the sample index.

$$\varepsilon = x(t)|_{t=nT} - x[n \cdot T] \tag{5.1}$$

As previously mentioned, audio CD specifications include two channels of audio sampled at 44.1 kHz for each channel and are quantized at 16 bits equivalent to 65,536 possible discrete amplitude values. For example, if the minimum and maximum amplitude values are normalized to -1.0 and $+1.0$ a 16 bit system would mean that the range between ± 1.0 would be divided into discrete 65,536 units. On the other hand, if values fall between, say 0 and 1, like 0.4, for an integer system with 65,536 integer points either a rounding (adding 0.5 and truncating the mantissa), "floor"ing (truncating the mantissa) or "ceil"ing (using next integer value only) method is commonly used in the quantization process. The method of using equally spaced amplitude step sizes is referred to as *uniform quantization.* Quantization of the amplitude values and using a specific sample rate to store or record data is referred to as PCM (pulse code modulation). It is somewhat a confusing term as there are really no pulses in a PCM system per se, except perhaps when analyzing the encoded binary structure of the digitized amplitude values. For example 2,175 in binary 16 bit format is 0000 1000 1000 1001 as illustrated in Fig. 5.1.

In Fig. 5.2 we can see the discrete amplitude/time sine wave representing an inferior version of the original sine wave. Note that the greater the bit resolution for a digital system, the smaller the amplitude grid division for the y-axis much like the sampling rate which decreases the time grid (T) as f_s is increased. This is depicted in Fig. 5.3 where the sampling interval is kept constant while the bit resolution is decreased

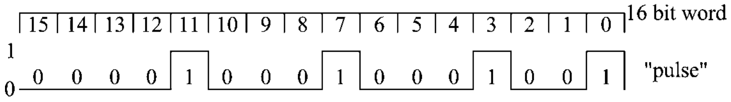


Fig. 5.1. PCM: integer value 2,175 in 16 bit word binary format.

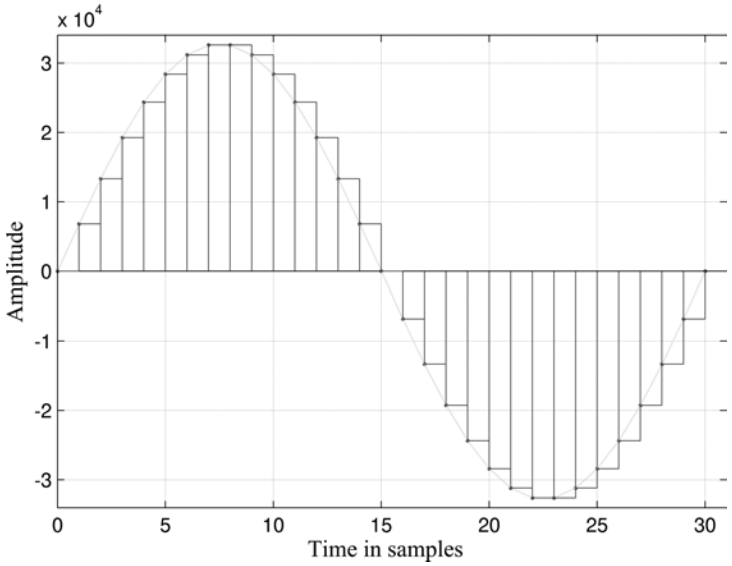


Fig. 5.2. Quantization of amplitude values.

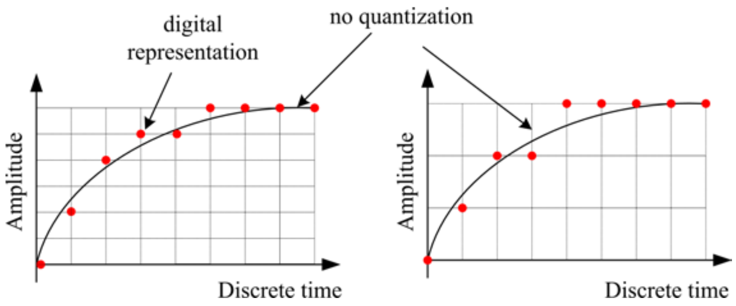


Fig. 5.3. Bit resolution and effect on quantization. Better quantization resolution (left), poorer quantization resolution (right).

causing the quantization error to increase — the left plot has a higher quantization resolution and right plot a lower one limiting the number of points that can be represented on the amplitude y-axis.

5.1 *SNR and QSNR*

A concept known as *signal-to-noise ratio* (SNR) is a standard way to determine the strength of a desired signal, such as music or speech, in relation to background noise or noise floor and is closely related to the bit resolution and quantization *signal-to-noise ratio* (QSNR) as we shall shortly see. The SNR is defined as the power ratio of a signal vs. noise and is usually measured as the average signal-to-noise ratio according to Eq. (5.2).

$$SNR = 20 \cdot \log_{10}\left(\frac{A_{signal}}{A_{noise}}\right) \tag{5.2}$$

A system with high SNR is generally desirable as it suggests that the signal is greater than the noise or conversely the noise is lower than the signal. Having a healthy SNR level is standard recording practice where the performer, say an electric guitarist, would have the guitar's volume dial at around its top position to mask out the inherent noise floor of the guitar itself, the amplifier, and microphone, as well as any other hardware device or recording space in the audio chain that is used to record the guitar sound in the studio.

When we quantize an analog signal we will inevitably get some sort of error as it is impossible to have infinite bit resolution in a digital system as seen in Eq. (5.1). The question then is how much quantization SNR, or QSNR as it is called, can we expect for a given bit? The QSNR is defined as follows where N refers to the bit resolution:

$$QSNR = N \cdot 6.02 + 1.76\,dB \tag{5.3}$$

An approximation to the QSNR can be derived by using the maximum amplitude of the desired signal (2^{N-1}) and the maximum error between the analog and digital amplitude values corresponding to 1/2 bit. The 1/2 bit error would be the error that results if the analog signal is smack between two successive integer values, say 60 and 61 at 60.5. The reason for 2^{N-1} opposed to 2^N is also quite simple. Go back to Fig. 5.2 where we used a 16 bit sine wave so that we have the full possible range from 0 to 65,535 integer values. If we want it to oscillate about the $x = 0$ axis (signed representation) we would have integer values between -32768 to $+32767$ (notice that the total possible values is 65,536 if you include the 0). Hence, if for example we are looking at a 16 bit signal the absolute (eg. $abs(-1) = 1$) maximum value would be $2^{16}/2 = 2^{16-1} = 32768$.

$$\begin{aligned} QSNR = 20 \cdot \log_{10}\left(\frac{A_{signal\_\max}}{A_{noise\_\max}}\right) &= 20 \cdot \log_{10}\left(\frac{2^{N-1}}{1/2}\right) \\ &= 20 \cdot \log_{10}(2^{N-1}) - 20 \cdot \log_{10}(1/2) \\ &= 20(N-1) \cdot \log_{10}(2) + 6.02 \\ &= 20 \cdot N \cdot \log_{10}(2) - 20 \cdot \log_{10}(2) + 6.02 \\ &= 6.02 \cdot N \end{aligned} \tag{5.4}$$

The important result that we get from QSNR is that we can approximate the dynamic range of a digital system — the QSNR basically tells us that for every bit added we get an increase of about 6 dB dynamic range. Hence, for a 16 bit digital system, which is the case for CD quality audio, we can expect a dynamic range of about $16 \cdot 6 = 96$ dB.

6 DC Component

The term DC actually comes from the electrical engineering community which stands for *direct current*. The DC, as referred to in DSP, pertains to an offset value that does not change over time. For example, if we have an input signal x and a corresponding output y, an offset or the DC would look something like Eq. (6.1), where C is a constant real number:

$$y(t) = x(t) + C \tag{6.1}$$

In our sine wave example adding an offset would be:

$$y(t) = A \cdot \sin(2 \cdot \pi \cdot f \cdot t) + C \tag{6.2}$$

This would simply result in shifting up/down the whole sine wave on the y-axis (vertically) without actually changing the shape of it. Hence, in an ideal situation, our hearing system would not be able to differentiate between a signal with a DC component and one without it. However, in digital systems as well as analog electronic systems, the DC component may cause

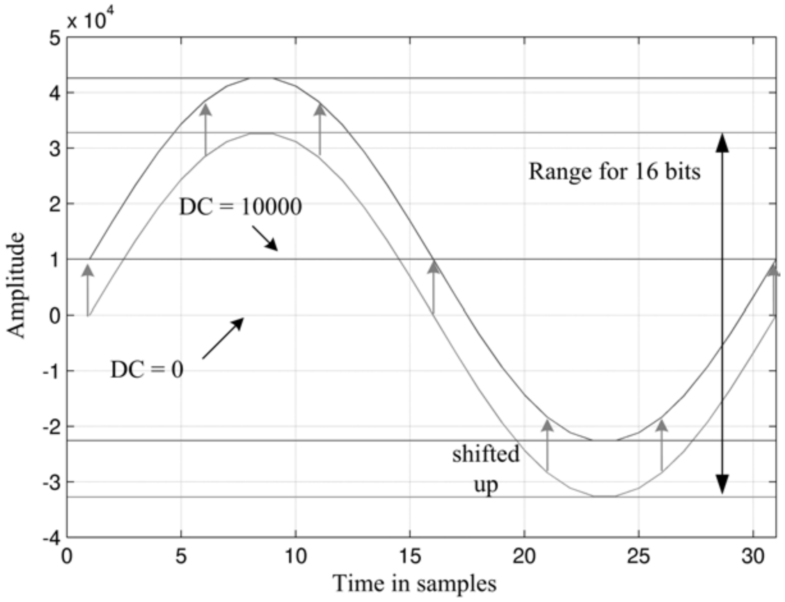


Fig. 6.1. DC offset and shifting of sine wave.

some potentially serious distortion problems. The reason is that for digital systems, only a finite number of bits are available for storing/representing data or waveforms (16 bits for CD). For example, if we add a DC constant to the sine tone in Fig. 5.2 of, say, 10,000, we get the plot as seen in Fig. 6.1.

Note that when storing/representing this waveform as a 16 bit signed integer, *clipping* (subject matter in next section) will occur at the top (in MATLAB® when saving to soundfiles amplitude values are normalized to −1.0 and +1.0). When clipping does occur due to DC offset issues, the waveform and the sound of the waveform will indeed be distorted and altered. Hence, care should be taken in removing the DC offset of a signal before sampling. We will later see how to address DC offset removal via filtering methods using a high-pass filter. An intuitive method for removing the DC component without *filtering* (although we are technically implementing a high-pass filter as will be discussed in Chap. 7) is finding the arithmetic mean (point on y-axis where it oscillates) of the waveform and removing the mean from the waveform. This is shown in Eq. (6.3).

$$y(t) = x(t) - \bar{x} \tag{6.3}$$

In MATLAB® this can be straightforwardly coded by using the built-in mean function as follows:

```
y = x - mean(x)
```

Code example 6.1

The mean is of course just the arithmetic mean — summing all numbers in vector x and dividing it by the vector size (number of elements in the vector).

7 Distortion and Square Waves

In the previous section, we briefly introduced the concept of clipping. Clipping occurs when a signal is too hot or when a DC component exists causing a signal to be clipped or cut off, in which case appropriate adjustments need to be made. Some of those adjustments include reducing the signal itself before some signal processing procedure (or before sampling) or taking out the DC offset as outlined above. In other situations, clipping is actually intentional — in music clipping is often used to alter the timbre of a sound source. Electric guitar players are probably most familiar with the term and use it to distort the guitar signal via stomp boxes, pedals, or amplifiers which exemplify the rock guitar sound. Clipping, distortion, or overdrive as it is also sometimes called in this context is simple enough:

$$y(t) = a \cdot x(t) \tag{7.1}$$

The parameter a is the gain factor which scales the input x. The following pseudo code implements a simple clipping distortion effect:

```
if (| a*x | > clippingLevel)
   y = sign(x)*clippingLevel;
else                                        (7.2)
   y = a*x;
end
```

If the input were a sine wave and we would gradually increase the gain, we would observe the characteristic (here we assume that values above the signed 16 bit limit is clipped) as seen in Fig. 7.1 where the scalar a is simply doubled at every plot starting at 1.0.

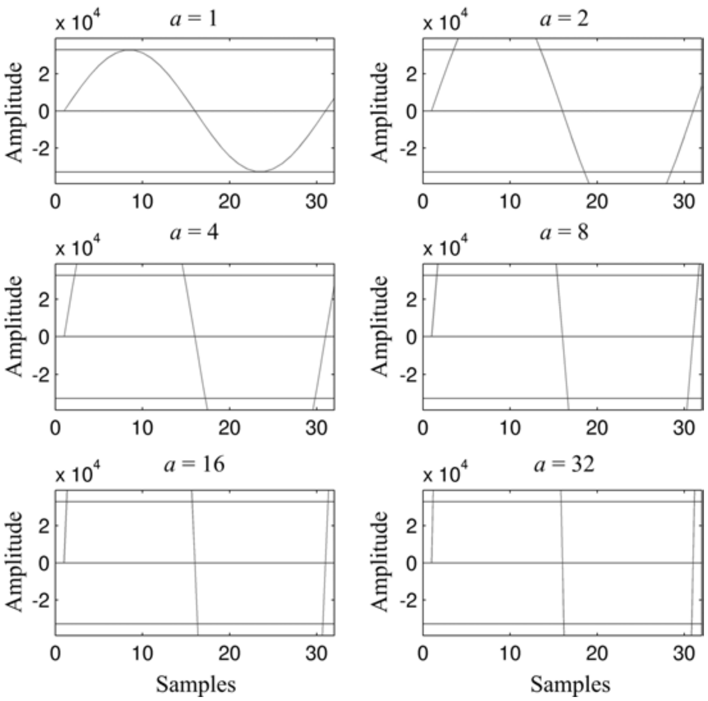


Fig. 7.1. Distortion and clipping — doubling scalar a from 1 to 32.

Nothing seemingly special here, other than the sine wave turning more and more into what resembles a *square wave*. Generally speaking, the purest tone (and probably most boring) is a single sine tone. It has an unmistakable and distinct sound quality and timbre but is not very musically intriguing, although it has had its share of usage in early electronic music, especially in Germany when experimentation with oscillators was popular with composers such as Karheinz Stockhausen. As the sine wave becomes more and more like a square wave, it also becomes more and more interesting and complex. So why is a square wave shape-like property more interesting? It is difficult to give a good explanation at this point as we have not talked about the Fourier transform and the decomposition of signals into sinusoids. However, the general rule of thumb in the complexity and richness of a sound object (least rich would be a sine wave, most rich would be white noise) is that the sharper the change in amplitude over time, the richer the sound becomes. Sine waves are very smooth and do not change much from sample to sample, whereas more complex waveforms such as the square wave change more drastically from sample to sample with noise signals being extreme examples. In our example shown in Fig. 7.1, the sine wave slowly becomes a square wave and the salient characteristic is that the resulting waveform exhibits sharper and sharper edges and hence produces a richer, more complex sound, and perhaps even a more interesting sound.

Going back to our guitar example, if the waveform in Fig. 7.1 were not a sine wave but a guitar sound, a similar distortion effect would result making the original guitar sound richer and again in a way, more interesting. It so turns out that with the clipped distortion effect odd numbered harmonics get added to the resulting signal (we will see why this is the case in Chap. 8). Hence, the clipped distortion method is also referred to as *harmonic distortion* as it adds odd harmonics to the original waveform, and depending on the modulated waveform, makes it richer, or at least more "oddly harmonic" so-to-speak.

7.1 *Dithering*

Dithering is a method where we add very small amounts (sub bit resolution — smaller than one bit) of pseudo-random noise (error ε) to the analog input signal before quantization and sampling as shown in Eq. (7.3). That may indeed sound unorthodox, as it seems counterintuitive to add noise to a signal as we want it to be as clean as possible.

$$x(t)_{dithered} = x(t)_{original} + \varepsilon \tag{7.3}$$

Have you ever listened to a CD recording on headphones and tried increasing the volume in the fadeout part of your favorite piece to see if there are any hidden messages or the like and all you got is this rather annoying buzzing, fuzzy, gritty sound? This is actually referred to as granulation noise. This is an artifact of sampling and quantization and is often times not so evident because a recording does not often play with the last 1 or 2 bits for a prolonged period of time. The listener thus may find this artifact not to be a problem at all, as it is not often heard due to the low signal level. However, to the composer or audio researcher who is analyzing, designing DSP algorithms, editing, or composing a musical work that deals with low amplitude signals, such problems are as common as finding portions of a sound where clipping occurs (hopefully not unintentionally on a commercial CD though). Dithering alleviates some of the problems associated with quantization described above. To understand how dithering works let's look at the quantization process using rounding. Figure 7.2 (top) shows

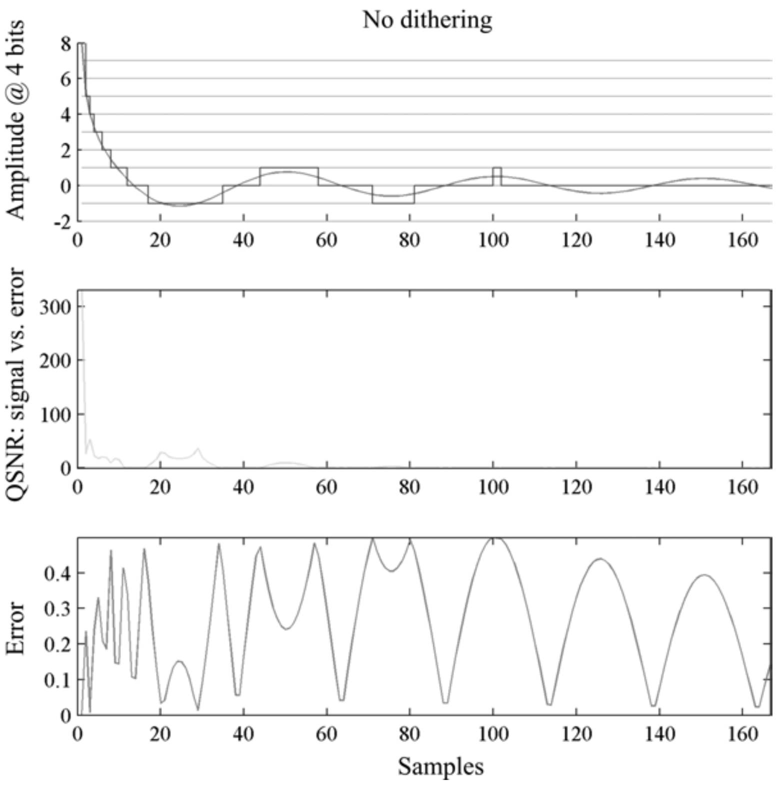


Fig. 7.2. Cosine wave after quantization (rounding) without dithering.

an exponentially decaying analog cosine wave and its quantized version with 4-bit resolution represented by the horizontal quantization grid lines.

As we can see in the middle plot, the QSNR decreases rapidly (caused by small signal to error ratio), which means that the ratio between the strength of the original signal and quantization error becomes smaller and smaller. This is the source of some potential problems — at around 20 to 110 samples, the quantized waveform becomes patterned or regular and has some features of square waves (we shall see an extreme case of this shortly) that was not present in the original analog signal. This is problematic as a square wave introduces harmonic distortion (adding harmonics that were not present before). Furthermore, issues in aliasing may also occur due to the newly added harmonic distortion — as more additional odd harmonics are added to the signal, there is a possibility of the harmonics being greater in frequency than the Nyquist limit (more details on this in Chap. 8).

Suffice it to say for now, a repetitive pattern resembling a square wave emerges adding unwanted harmonics, especially odd harmonics not part of the original signal. An obvious fix for the above problems would of course be increasing the bit resolution which will give us the bit depth to represent the nuances in the low amplitude areas (low QSNR), thereby adding approximately another 6 dB to the dynamic range for every bit added. However, this method is highly uneconomical and does not really address the problem as we cannot know in advance what signal we are going to be dealing with, how loud it will be, and when it will change to a completely different signal. A solution for solving some of these problems is referred to as dithering. In essence, what dithering accomplishes is that it indirectly increases the bit resolution for low amplitude signals especially in the 1 bit range — it actually improves the 1 bit limit to something lower than that.

Dithering is a very simple and clever way to help eliminate some of these artifacts by adding a small error signal to the original analog signal before sampling and quantization. In essence it is making the resulting signal noisier but as the added noise is so small it is for all practical purposes

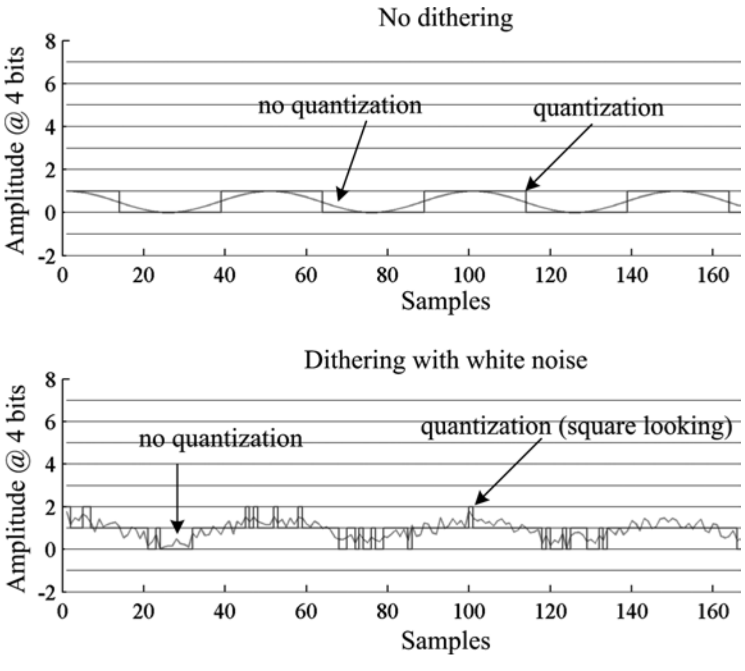


Fig. 7.3. Harmonic distortion fix without dithering (top) dithering (bottom).

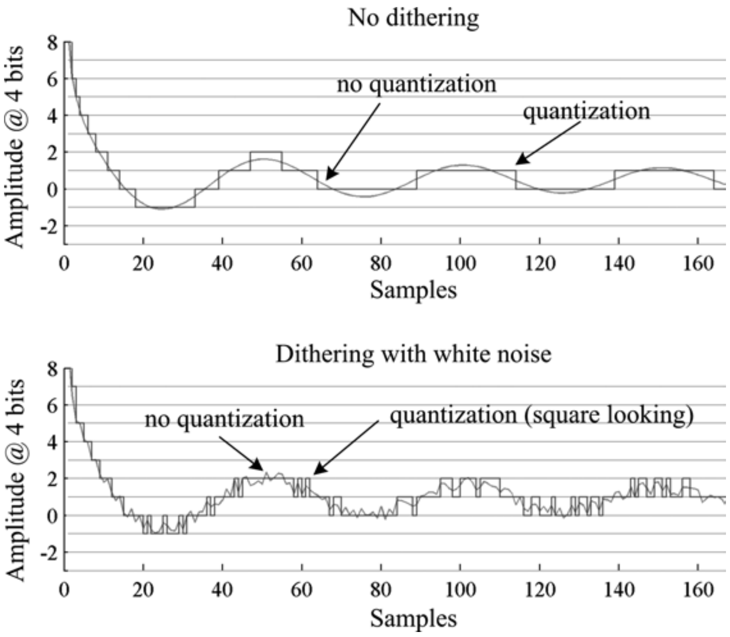


Fig. 7.4. Exponentially decaying cosine without (top) and with dithering (bottom).

negligible and inaudible. When this modified signal is quantized, it will not produce the characteristics of harmonic distortion via a transformation of the patterned quasi-square waves to an un-patterned one. One of the main reason this works for humans is that we tend to mask out noise and perceive it as background (especially when it is low) as our basilar membrane (part of the cochlear in our ear) has an averaging property bringing out repetitive patterns, even if that pattern of interest is engulfed by a relatively high noise floor. Thus, with dithering, we are literally making a decision to swap out harmonic distortion for low amplitude noise to make the artifacts seem less noticeable.

Dithering is illustrated in Figs. 7.3 and 7.4. The top plot of Fig. 7.3 illustrates the extreme case where the analog cosine literally becomes a square wave introducing harmonic distortion to the digitized signal via harmonics that were not present in the original analog signal. The dithered version of the cosine on the other hand at the bottom plot of Fig. 7.3, when quantized, loses this repetitive pattern of the square wave. Finally, Fig. 7.4 shows an exponentially decaying cosine with and without dithering. Clearly the dithered quantized version does a better job eliminating the square wave patterns. The quantization scheme applied in the plots is also referred to as *pulse width modulation* (PWM) which refers to the way the pulse width contracts and expands in accordance with the signal.

8 Musical Examples

There are a number of interesting musical examples that take advantage of some of the concepts we have discussed in this chapter. One such piece is *Kontakte* (1958 $\sim$ 1960) by composer Karlheinz Stockhausen. In this piece, Stockhausen explores many facets of acoustics (among other things) including "contacts" between different sound families, morphing between a select number of sounds, as well as exploring boundaries between pitch, frequency, and roughness by increasing the frequency of a low frequency pulse (a pulse you can count) to the point where the pulse goes through gradual changes of rhythm, roughness/timbre, and pitch. *Kontakte* is also one of the first multi-channel musical works employing a quadraphonic sound projection strategy, where the composer used a loudspeaker mounted on a rotating turntable to record onto four-channel tape via four separate microphones. With this configuration, Stockhausen captured the orbital characteristic of the sound source which could be experienced by the audience in a quadraphonic sound reinforcement setup. With the advent of the tape recorder (the precursor to the digital audio workstation), composer Pierre Schaeffer wrote a number of incredible variations solely via recorded sounds of a creaking door and a sigh called none other than *Variations for a Door and a Sigh* composed in 1963. In this piece, through magnetic tape manipulations such as time stretching/compressing (playing the tape machine slower or faster), extracting and focusing on inherent musicality of the sounds, such as exposing rhythmic, timbral, and melodic features; Schaeffer guides the listener to a sonic experience that is quite unlike any other. This piece is a great example of using *found sounds* in the style of *music concrète*. Some 20 years later, John Oswald coined the term *plunderphonics* and wrote an essay entitled *Plunderphonics, or Audio Piracy as a Compositional Prerogative* (Oswald 1986). He wrote a collection of pieces such as *dab* (materials from Michael Jackson's song *bad*) utilizing existing recordings and thus touching on sensitive issues of copyright and intellectual property, subsequently leading CBS and Michael Jackson filing complaints to the Canadian Recording Industry Association. What's interesting is not just the complex world of copyright (or copyleft!) and the world of litigation, but the fact that the excerpts from the original track are chopped up into very short segments, often no more than one or two seconds or even shorter passages. The composer then takes those short samples and sequences them, builds rhythms, patterns, and does much more, essentially making a “totally” new piece, while at the same time leaving an unmistakable sonic residue of the original track challenging the listener to negotiate between context, content, origin, creativity, and novelty.

Another great example in the area of sampling and computer music can be seen in Jon Appleton’s *Newark Airport Rock* (1969). The piece can be regarded as a sampling expedition by the composer who recorded interviews at the airport asking interviewees to comment on electronic music. In a way, the composer’s job in this sort of project revolves around organizing, juxtaposing, repeating, and perhaps most importantly selecting those sound excerpts that will produce a desired musical composition from the vast number of samples. The piece also resembles a short documentary and clearly is narrative in its musical construct. Computer musicians and computer music researchers often walk the fine line between the world of science and art. This is especially true when testing out newly designed synthesis algorithms or using signal processing concepts in ways they were not “meant” to be used. An example of such a piece is called *The Machine Stops* written by the author in 2000. This piece actually exploits and “misuses” some of the DSP concepts we covered in this chapter - aliasing, sampling, distortion, and the sine wave. In this piece, the artifacts of aliasing are used in a musical context as is distortion, sampling and *audio rate* (above 20 Hz and beyond) *panning* and other processes. The starting point of the composition was just a single wave which really is the focus of the piece — what can we do with one sine wave and some “malicious misuses” of a few basic DSP techniques? That’s the wonderful thing about musical composition — there is really no incorrect way of composing although whether the result is interesting, good, appealing, boring, or intriguing is a whole different matter altogether. From an engineering point of view, however, the results may be regarded as wrong and even devastating, but the composer has the prerogative and extra rope to break rules and standard engineering (and musical!) practices without the need to worry (too much) about the resulting errors, provided it does not hurt anyone!

References and Further Reading

Dodge, C., Jerse, T. A. 1985. Computer Music Synthesis: Synthesis, Composition and Performance, New York, Schirmer Books.

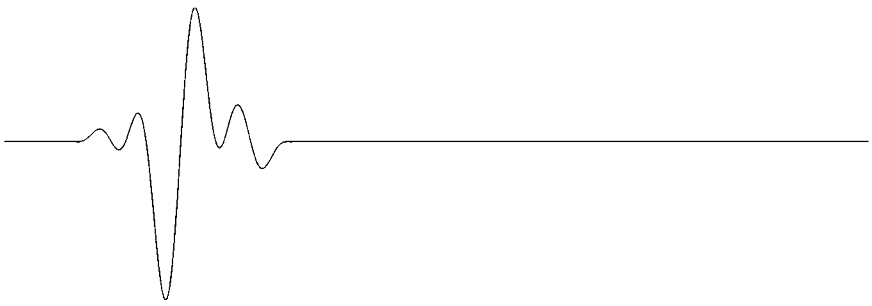
Fletcher H., W. A. Munson 1933. “Loudness, its Definition, Measurement and Calculation”, Journal of Acoustical Society of America, 5, 82–108

Oswald, J. 1986. “Plunderphonics, or Audio Piracy as a Compositional Prerogative”, Musicworks 34

Zwicker E., Fastl H. 1999. Psycho-acoustics, Facts and Models, Springer-Verlag Series in Information Sciences

Chapter 2

TIME-DOMAIN SIGNAL PROCESSING I



1 Introduction

In this chapter and the next chapter, we will introduce a number of important concepts and signal processing techniques that can be found in the time-domain. The term time-domain generally refers to topics (analysis, synthesis, signal modulation, etc.) that have to do with two-dimensional data types, amplitude and time being the two dimensions. The sine wave we have seen in Chap. 1 is a perfect example where the signal is represented in those two dimensions. The counterpart to time-domain is referred to as the frequency-domain which will be formally presented in Chap. 8. Frequency-domain concepts are maybe a bit more difficult to grasp in the beginning for some folks as the concept differs on a fundamental level. Time-domain concepts on the other hand probably will come to us more naturally as we are accustomed to hearing sounds and waveforms in the time-domain. We will thus spend a substantial amount of time in the time-domain and get familiar with various DSP concepts before proceeding to frequency-domain related studies. Topics covered in this chapter will include amplitude envelope computation, pitch detection and autocorrelation, overlap and add concepts, as well as a select number of classic sound synthesis algorithms. As usual, the chapter will conclude with musical examples pertinent to discussed materials.

2 Amplitude Envelope and ADSR

The so-called amplitude envelope is an important yet straightforward concept to grasp as we probably have encountered it on numerous occasions by looking at audio waveforms. The amplitude envelope of a sound object refers to the amplitude contour or the general shape of the signal's amplitude with respect to time. It can be somewhat regarded as a zoomed-out view of a waveform without the subtle details. Amplitude envelopes are especially important sonic features for musical instruments. In particular, it is common to talk about the *ADSR* which stands for A(ttack), D(ecay), S(ustain), and R(elease), dividing the envelope of a musical instrument tone into 4 basic areas as seen in Fig. 2.1.

Figure 2.2 shows a Steinway piano sampled at 44.1 kHz and 16 bits playing a C4 note. The top figure is the waveform of the piano sample, the middle plot the *root mean square* (RMS) amplitude envelope (the topic of next section), and bottom one the RMS with logarithmic amplitude plot. Notice that there is a pretty sharp attack, long sustain, and even longer release in this example. Figure 2.3 shows a snare drum struck with a stick (fortissimo) at the same bit depth and sampling frequency. In the snare drum example, notice how quickly the snare drum sound dies away — at 300 milliseconds the snare drum sound has already reached about −40 dB whereas the piano at 300 milliseconds is still at approximately

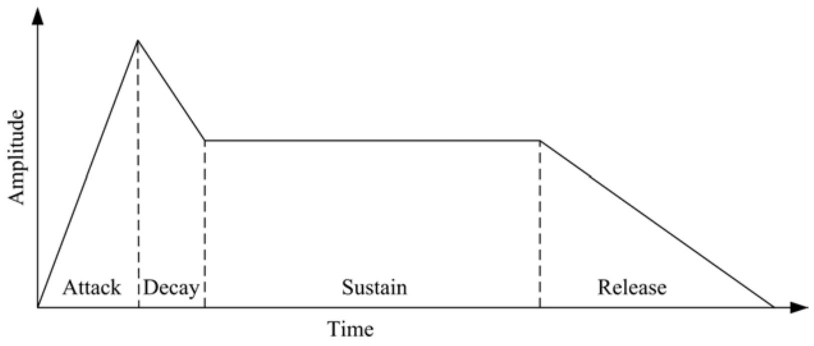


Fig. 2.1. ADSR envelope.

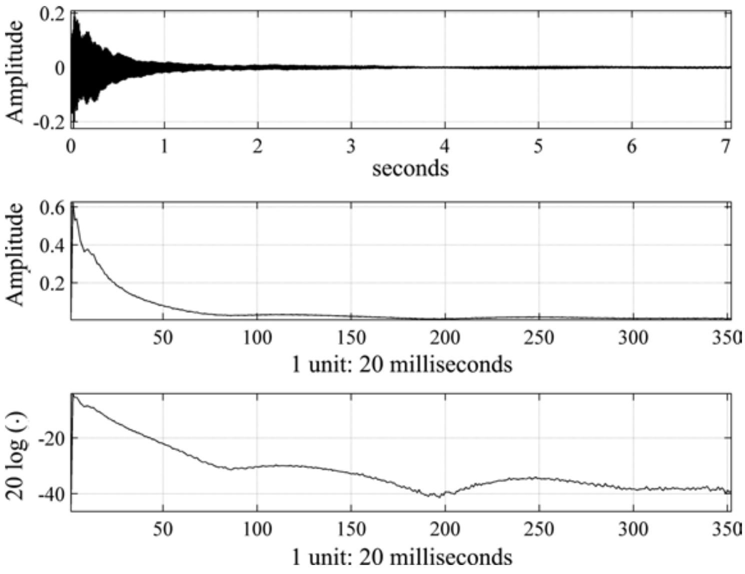


Fig. 2.2. Steinway piano C4, waveform (top), linear envelope (middle), dB envelope (bottom).

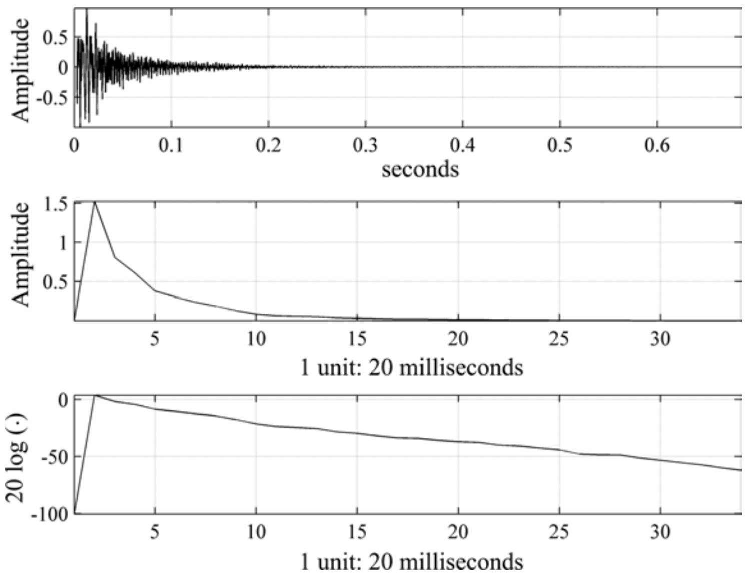


Fig. 2.3. Single snare drum hit, waveform (top), linear envelope (middle), dB envelope (bottom).

-18 dB approaching -40 dB at around 4 seconds. One of the reasons the amplitude envelope and the ADSR structure are important is because by altering the ADSR parameters, the perception of the sound object changes accordingly — the degree of change depends on the type and characteristic of modulation of the envelope. Sometimes even a relatively small change in the amplitude envelope can radically alter the original sonic identity itself, in extreme cases rendering the modulated sound unrecognizable. Try playing a piano chord or single note in MATLAB® after loading it without any alteration and then try playing it in reverse so that it starts playing the last sample first and first sample last (Code Example 2.1). Granted everything has been reversed and not just the envelope, but the perception of the sound object is drastically changed by a very simple process highlighted by the time-reversal of the amplitude envelope.

```
[x, fs] = wavread ('nameOfPianoSample.wav'); % read wave file
sound (x, fs)

disp ('Type any key to continue')
pause

xReverse = flipud (x); %flip the array up -> down
sound (xReverse, fs)
```

Code Example 2.1. Reversing time-domain audio

3 Wavetable Synthesis

Looking at the ADSR structure of a musical note, an interesting observation can be made. Instruments such as the piano, vibraphone, or the electric bass have certain general behavior in the ADSR regions, especially in the attack and steady-state regions. The attack portion of the waveform is generally most complex and chaotic albeit not fully random or "noisy." The attack also generally embodies of a wealth of energy which gradually dissipates by the end of the tone as it loses energy. The steady-state part of a musical instrument sound on the other hand is usually more stable and displays characteristics of quasi-periodicity. Figure 3.1 shows a piano sample at different points in time where we can clearly see the waveform becoming smoother and more patterned towards the end of piano tone.

A type of time-domain synthesis method called *wavetable synthesis* exploits the aforementioned characterstics. This synthesis method stores

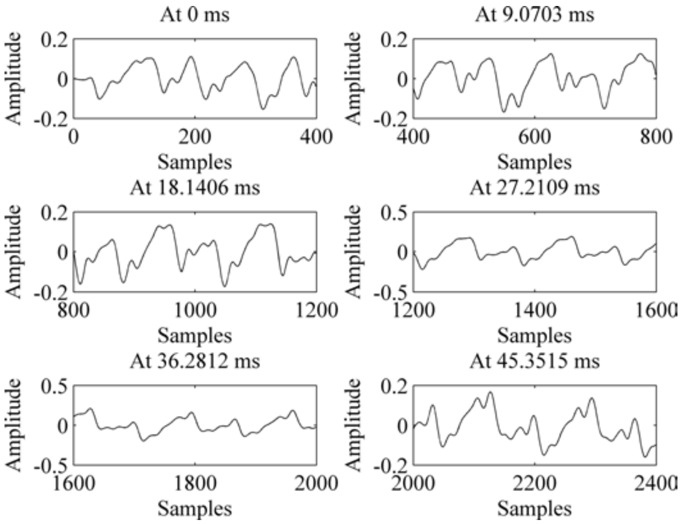


Fig. 3.1. Piano sample at different parts of the overall signal.

only the attack portion of the actual musical instrument waveform and a short segment of the steady-state. Thus, the waveform that is stored in ROM (read only memory) includes only an abbreviated version of the entire signal — the entire attack section and a portion of the steady-state. One of the reasons for doing this is actually very practical — economics. One can save ROM space on your synthesizer allowing for more sounds to be stored and making the synthesizers more powerful and affordable! You will probably think that merely storing the attack and a short segment of the steady-state is too short for practical usage in compositions or performance situations which is indeed true. It is probably fine when playing sharp staccato notes that are shorter or equal to the saved waveform, but if the user wants to sustain it for a longer duration than what is stored in ROM, we will encounter a problem as we do not have enough of the waveform stored for desired playback. The solution to this problem is what wavetable synthesis essentially is.

The steady-state is, for the lack of a better word, steady and (quasi) periodic as seen in Fig. 3.2. This means that it is easy enough to model as it is clearly patterned. What happens in wavetable synthesis is that a portion of the steady-state is selected via *loop start* and *loop end* markers as illustrated in Fig. 3.2.

Once the end of the loop point is reached (after the attack and a small portion of the steady-state), the wavetable pointer will just keep looping

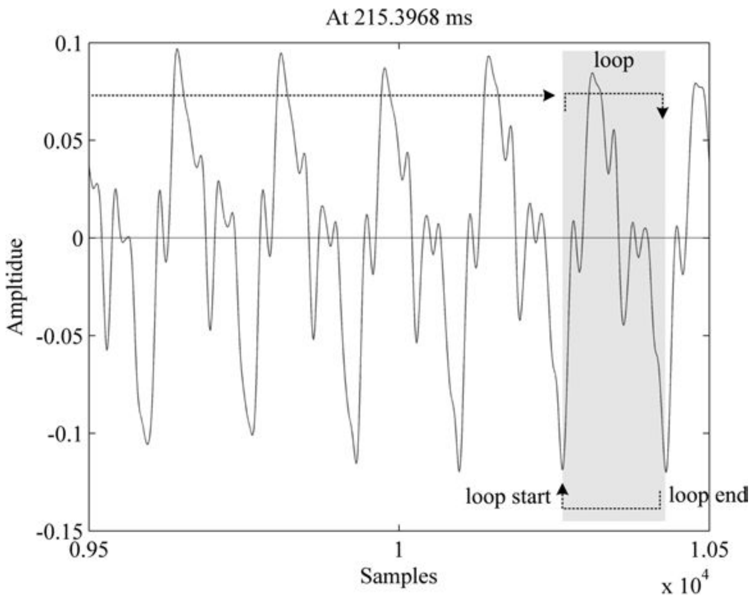


Fig. 3.2. Loop points in wavetable synthesis.

between the loop start and end point markers. Through this looping, the ear is tricked into believing that the waveform has not ended but is actually being sustained. If we furthermore slowly decrease the amplitude while looping (using an envelope), it is also possible to mimic the gradual release characteristic of the original waveform even though we are only using a fraction of the original sound. The total length of this particular piano sound in its entirety is 108, 102 samples long and if we were to use wavetable synthesis to synthesize the piano tone we would only use approximately 3500 samples or 3.2% of the original, thus saving quite a bit of memory! This simple idea of using the attack and a select portion of the steady-state waveforms combined with looping the quasi-periodic part is in a nutshell how wavetable synthesis works. However, putting an idea into practice is often more difficult than meets the eye as there are so many different types of signals with a plethora of waveform shapes. In wavetable synthesis, the trick is thus to find a good set of loop points as close to the beginning part of the steady-state as possible, while at the same time keeping the loop region as short as we can, to save memory. Other tricks to improve the quality of wavetable synthesis include using amplitude envelopes to control the ADSR characteristics as mentioned before as well as opening/closing filters to brighten or dampen part of the synthesized waveform.

4 Windowing, RMS, and Amplitude Envelope

In this section, we will present a method for computing the amplitude envelopes much like the ones shown in Figs. 2.2 and 2.3. We will start with an important concept called *windowing*. When analyzing a signal, one will most likely not analyze it sample-by-sample, especially when it is a large signal, nor would one analyze it as a whole single unit from beginning to end although at times this could indeed be the case. Analyzing a signal on a sample-by-sample basis equates to too much detail and analyzing a signal in its entirety and representing it with one, two, or a couple of numbers is not very informative either. Just think of how we listen to sound — we don't follow each sample (44,100 samples per second for CD), nor do we rarely just take-in and analyze a song from beginning to end in one chunk. We usually analyze a sound by dividing it into smaller sections. Generally, when analyzing signals, a group of samples are "windowed" and then analyzed. For example, let's say we have a signal that is 10,000 samples long and we use a *window size* of 1000 samples. In this scenario, we would window the first 1000 samples, analyze this first group of samples, then move on to the next 1000 samples starting at sample number 1001, analyze this portion of 1000 samples, and continue until we reach the end of the signal. This method of focusing and extracting a range of sequential samples that make up a portion of signal is referred to as *windowing*. Typically a large number of windows result (depending on the window size and signal size of course) and these windows are each individually analyzed thus forming an overall picture of the whole signal.

4.1 *Windowing: More details*

The most common type of window is the *rectangular window* characterized by sharp edges as seen on the left-hand side of Fig. 4.1. The rectangular window has unity gain within the confines of the rectangle itself and is 0 elsewhere according to (4.1), where N is an integer number pertaining to the window length or size and n is the sample index ($n \cdot T$ is the time location in seconds).

$$w[n] = \begin{cases} 1, & 0 \le n \le N-1 \\ 0, & \text{otherwise} \end{cases} \tag{4.1}$$

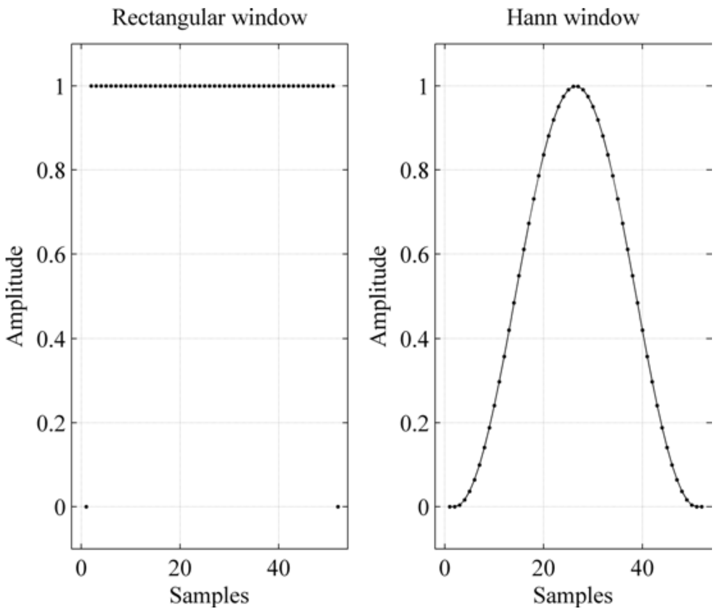


Fig. 4.1. Rectangular and Hann windows 50 samples long.

Previously I mentioned that windowing extracts a portion of a signal, which is true, as that is the result we get. However, when we say that we window a signal, what we actually mean technically is that we multiply the window with a portion of the signal of the same size as follows:

$$x_{\text{win}}[n] = w[n-L] \cdot x[n] \tag{4.2}$$

Here, L is an integer time offset (also known as delay) that indirectly determines the starting sample index of the waveform $x[n]$ — the input samples outside the range of the window are multiplied by 0.

If this is a bit confusing, plugging in some numbers will probably help clear things up a bit. Let's modify Eq. (4.1) a little, add L into the picture, and when we set $N = 50$ and $L = 0$ we get:

$$w[n-L]|_{L=0} = w[n] = \begin{cases} 1, & 0 \le n \le 49 \\ 0, & \text{otherwise} \end{cases} \tag{4.3}$$

Keeping N the same at 50 and setting $L = 1$ we have:

$$w[n-L]|_{L=1} = w[n-1] = \begin{cases} 1, & 0 \le n-1 \le 49 \\ 0, & \text{otherwise} \end{cases} \tag{4.4}$$

Equation (4.4) can be rewritten as (4.5) by adding 1 to the left, right, and middle part of the region of interest ($0 \le n-1 \le 49$) of the window.

$$w[n-1] = \begin{cases} 1, & 1 \le n \le 50 \\ 0, & \text{otherwise} \end{cases} \tag{4.5}$$

The general form of the shifting window thus becomes:

$$w[n-L] = \begin{cases} 1, & L \le n \le N-1+L \\ 0, & \text{otherwise} \end{cases} \tag{4.6}$$

Notice that L (referred to as a delay of L samples) merely shifts the window by L samples on the time axis resulting in windowing of only a select portion (length N) of a signal. For a rectangular window, this simply means that we are extracting, cutting out or slicing out a portion of the waveform without any alteration to the amplitude values within the region of the signal that is being excerpted as depicted in Fig. 4.2.

The following example code shows a simple rectangular window example on a piano signal that is read from the hard-drive.

```
[x, fs] = wavread('nameOfPianoSample.wav'); % read wave file
xWindowed = x(1:500);
sound(xWindowed, fs);
```

Code Example 4.1. Simple rectangular windowing example

You will probably have noticed that a "pop" sound can be heard at the very end and beginning of the rectangular windowed signal. This is not an error per se, although perhaps undesirable. This result is due to the sharp edges of the window which brings us to another type of window — the *Hann* window, sometimes also referred to as the *Hanning* window. As you can see in the middle of Fig. 4.3, the Hann window has smooth edges, peaks at the center of the window, and tapers off at both flanking sides. With this sort of window shape we will not experience the popping characteristics we encountered with the rectangular window. However, this comes at the expense of changing the original waveform's characteristics and shape within the windowed portion. This is shown at the bottom of

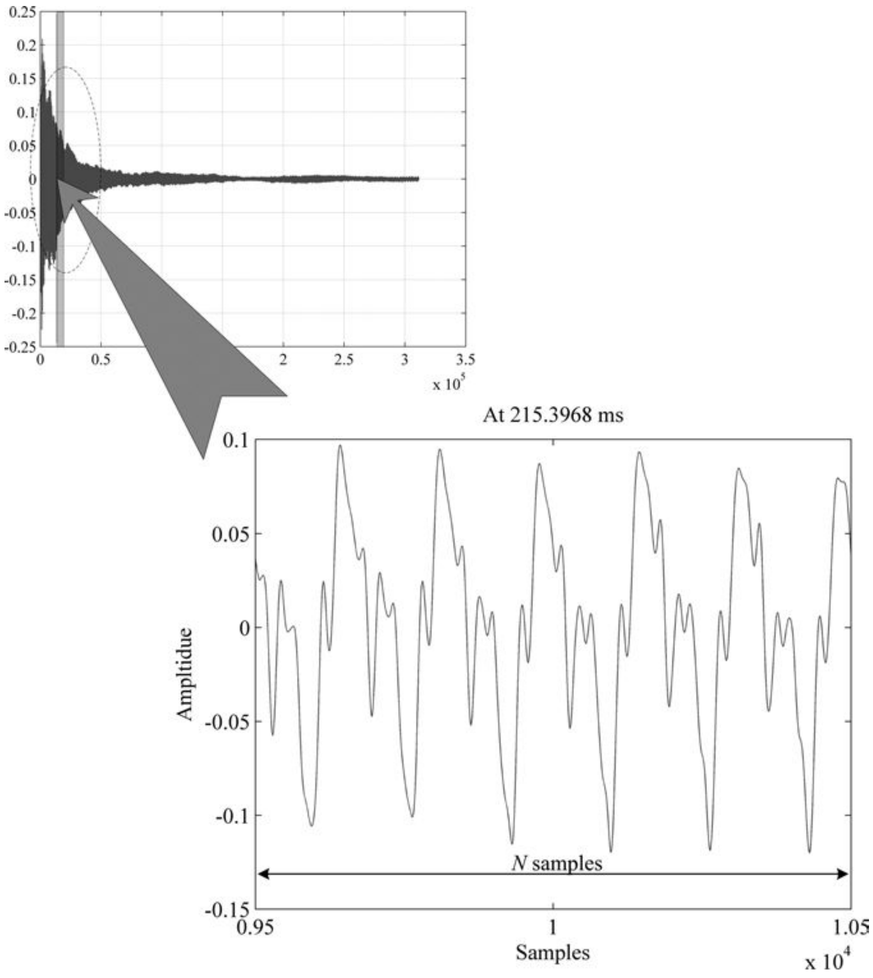


Fig. 4.2. Windowing of piano sample.

Fig. 4.3 depicting the difference between the original signal and the Hann windowed signal.

It is difficult at this stage to describe what a particular window shape does to the waveform it is multiplying, other than addressing the change that occurs on the amplitude level of the waveform — we do not yet have the tools to look at signals in the frequency-domain. However, a general rule of thumb is to remember that any waveform that has sharp edges is more complex than waveforms that have duller or smoother edges. This

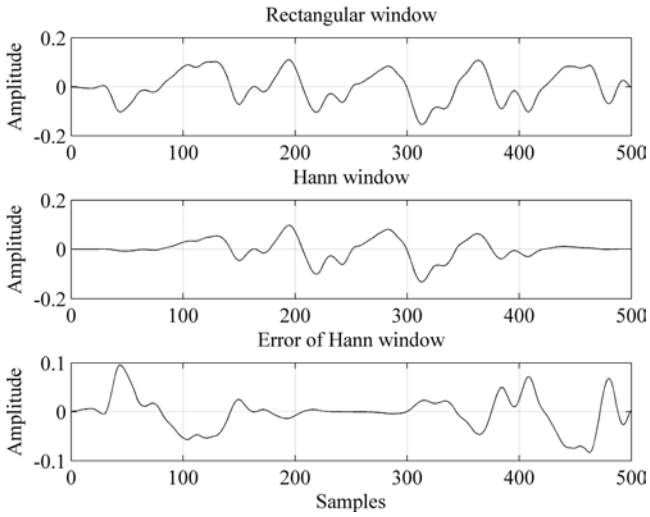


Fig. 4.3. Rectangular (top), Hann windows (middle), and Hann windowing error (bottom).

is intuitive as we know that noisy signals are all over the place, often changing drastically from sample to sample, whereas sine waves which are the opposite, change smoothly from sample to sample and do not have sharp edges. At this point, we could say that the rectangular window will have different types of artifacts compared to the Hann window which are dependent on edge characteristics and shape of the windows. We will revisit the issue of window types in Chap. 8 and discuss some of the important characteristics of various window types and what sort of artifacts we can expect which will help us determine what kind of window to use for what occasion.

4.2 *RMS and amplitude envelope*

From a very simplistic point of view, the amplitude envelope is a zoomed-out examination of the contour of positive amplitude values of a waveform. One popular method for computing the amplitude envelope is the RMS algorithm. RMS stands for *root mean square* and is defined in Eq. (4.7). The name itself pretty much describes the algorithm where L is the window length and n sample number.

$$RMS = \sqrt{\frac{1}{L}\sum_{n=0}^{L-1} x^2[n]} \tag{4.7}$$

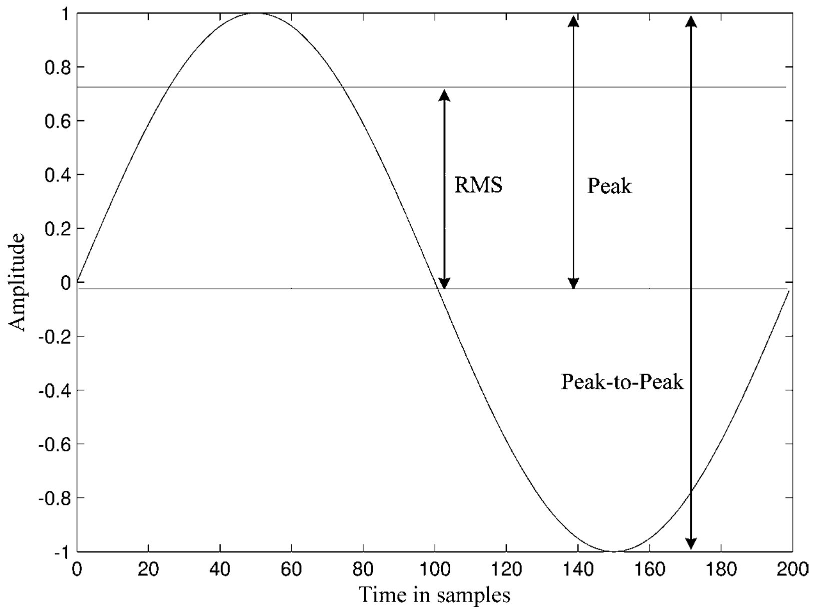


Fig. 4.4. RMS, peak, and peak-to-peak amplitude values.

RMS can be thought of being somewhere in between the arithmetic mean and the peak values of a signal. The mean value will not change much even when there is a high level of activity in a signal which makes the transient response (response pertinent to degree of change) rather poor. On the other extreme, following every sample value will result in an envelope that will be too detailed, with too much transient information which is not the way we hear sounds. The RMS, however, more closely corresponds to our hearing system's sensitivity to the change in intensity of audio signals. Figure 4.4 shows a sine wave with corresponding RMS, peak, and peak-to-peak ranges.

Note that in computing the amplitude envelope using RMS, we will need to window the signal as seen in Fig. 4.5 and Fig. 4.6. What essentially happens through windowing is that the resulting data of the RMS envelope will be shorter in length compared to the original signal as depicted in Fig. 4.6. That should, however, not come as a surprise, as windows act as representative peeks into a portion of a signal reflecting the characteristics of that particular section of time. Smaller window sizes allow for more transient response, whereas larger windows provide a wider view and hence slower transient response. Think of it this way — let's say we have 365 days a year and a paper-based calendar that hangs on the wall. Those types of

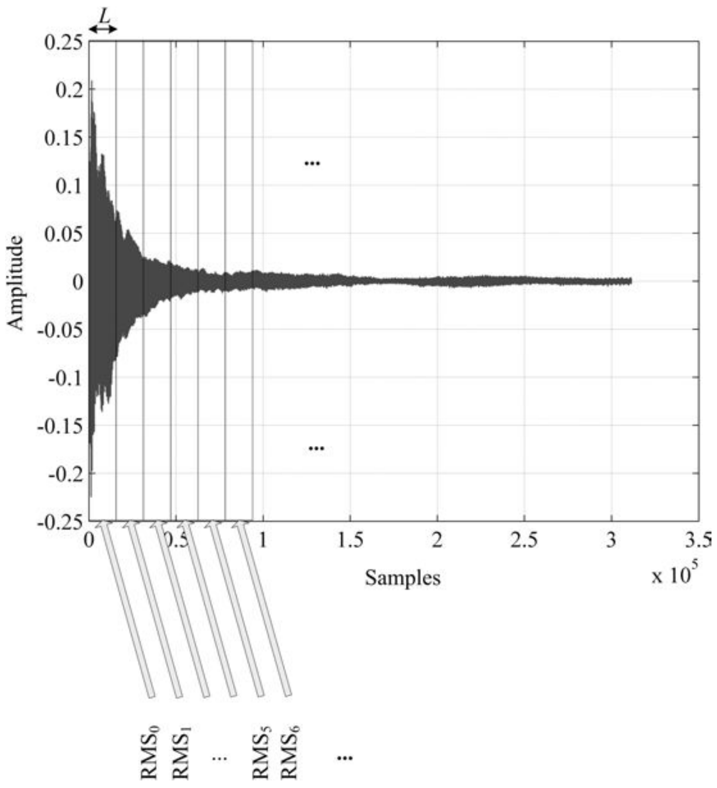


Fig. 4.5. Windowing of waveform and RMS frame computation.

calendars usually have one page per month and approximately 30 days for each page. These 30 days can be regarded as the window size and each month would have a particular characteristic — January is the coldest opposed to August being the hottest (depending where on the globe one lives of course). The point here is that one loses transient response, or the details of the day-to-day changes or week-to-week changes. The transient response thus is a function of the window size. The longer the window, the less transient detail we will get. The shorter the window, the more of the transient particulars we will acquire. The question then is — what is the appropriate size for a window? This is of course a difficult question and there is no simple answer. The answer is dependent on what the analysis setup is and how much detail is required for a certain analysis task. For example, using a window size of one minute for each calendar page would without

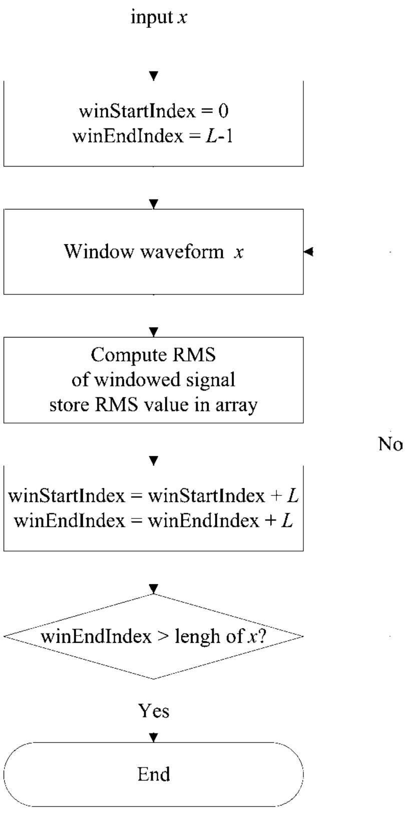


Fig. 4.6. RMS envelope computation.

a doubt be overly excessive; albeit in some situations that may not be the case at all (we would need a huge calendar page though)!

5 Time-Domain Fundamental Frequency Computation

In this section, we will discuss two basic algorithms for computing pitch. The objective in finding the pitch usually refers to finding the fundamental frequency, as pitch is a perceptual aspect of periodic and quasi-periodic sound objects. The two popular time-domain pitch detection algorithms that we will discuss are *zero-crossing-rate* and *autocorrelation.* These two algorithms are relatively simple to implement and quite effective at the same time.

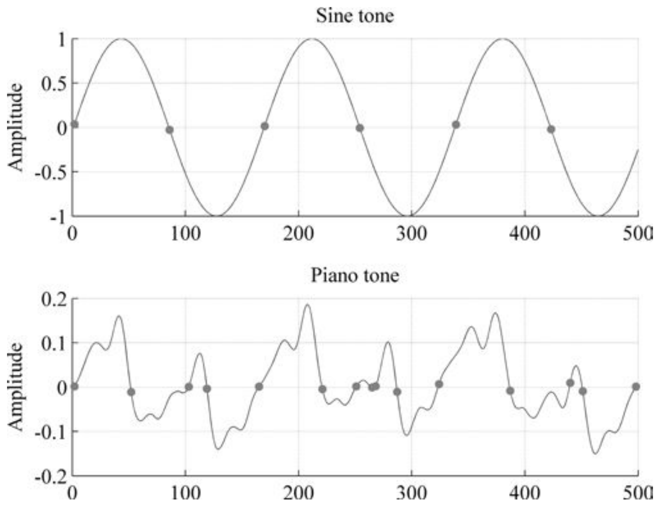


Fig. 5.1. Sine tone (top) and piano tone (bottom) played at C4 (261.1 Hz) f_s = 44.1 kHz.

5.1 *Zero-crossing rate*

Zero-crossing rate is one of the more straightforward methods used for computing the fundamental frequency. As the name suggests, the algorithm works by stepping through the waveform and measuring the time (via samples) between successive zero-crossings ($y = 0$ axis) as seen in Fig. 5.1 (dotted locations just on/below/above amplitude = 0 line).

The location and computation of the zero-crossings for the sine tone at the top of Fig. 5.1 is actually quite simple to estimate as you can imagine. The fundamental frequency is likewise straightforward to compute — the definition of Hertz is cycles per second and we can compute the fundamental frequency of the sine tone as shown in Eq. (5.1). That is, there are a total of 522 zero-crossings per second (44,100 samples) in this example and since we know that we have two zero-crossings per cycle for an ideal sine wave, we can compute the pitch by dividing the number of zero-crossing by two as shown in Eq. (5.1).

$$f = \frac{522\ zcrs/\sec}{2\ zcrs/\text{cycle}} = 261\,\text{Hz} \tag{5.1}$$

Alternatively, the fundamental frequency can also be expressed in terms of samples per cycle which can be determined by the locations of the zero-crossings. One full-resolution occurs at every other zero-crossing and hence the frequency can be computed as seen in (5.2) below.

$$f = \frac{44{,}100 \text{ samples/ sec}}{170 \text{ samples}} = 259.41\,\text{Hz} \tag{5.2}$$

So far so good, but as is the case with most things that seem "too good (simple) to be true," problems often arise and the ZCR method for pitch computation is no exception. The piano waveform at the bottom of Fig. 5.1 is much more complex than the pure sine tone. Due to the complexity of the waveform we get different number of zero-crossings for each cycle which in turn causes the zero-crossing algorithm to become inaccurate. For example, the first full cycle shows 4 crossings, the 2nd cycle 6 crossings, and 3rd cycle 4 crossings. Although the pitch that we hear is the same, the number of zero-crossings for one full cycle of the waveform varies. Generally speaking, the more complex a waveform is, there more problems of such nature will occur — musical signals are usually complex. There are a number of tricks we can use at this point to alleviate the situation, including conditioning the zero-crossings to a limited bandwidth or averaging the zero-crossings over a period of time. To limit the bandwidth, we could restrict the allowable upper and lower fundamental frequencies (Hz):

$$27 \le f \le 4200 \tag{5.3}$$

Equation (5.3) in terms of samples per cycle is shown in (5.4).

$$1633 > \text{samples} > 10 \tag{5.4}$$

The above corresponds to frequencies that are just beyond the highest (C8) and lowest (A0) note on the piano. With this kind of *filter*, very short zero-crossing pairs would be ignored. For example, zero-crossings spanning 6 samples (equivalent to 7350 Hz) would not be counted as zero-crossings.

The zero-crossing points are again shown in Fig. 5.2 for the same sine tone (top) and the piano samples at two different points in time — beginning of the piano tone (middle figure) and during the steady-state/release part (bottom figure) of the piano sample. You will note that the ZCR algorithm works much better in the latter part of the signal — this is not surprising as the piano tone enters the steady-state portion which is quasi-periodic almost resembling a sine wave. At the bottom of Fig. 5.2 you will see that the average ZCR is 84 samples and very close to the sine tone at the same frequency (top figure). It also turns out that the attack region has the most complex structure (farthest from a pure sine tone and closer to a noise signal) and the further away one goes from the attack

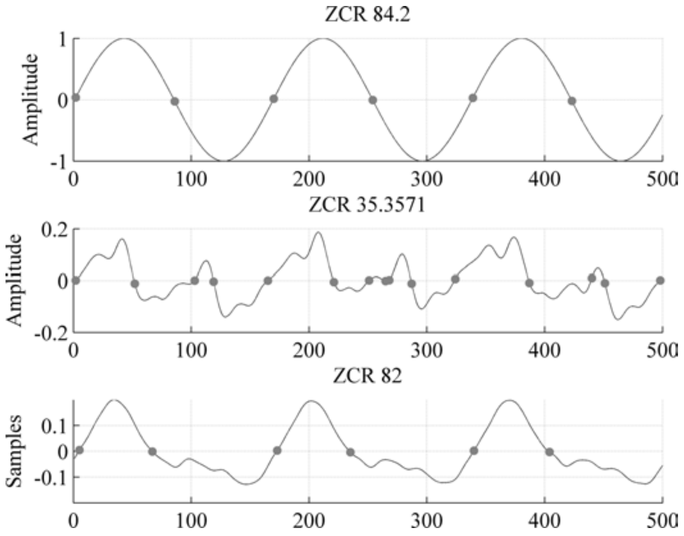


Fig. 5.2. ZCRs and ZC for sine tone (top) and piano (middle and bottom).

towards the steady-state and release parts, the more sine wave-like the waveform generally becomes — the upper harmonics lose energy and only the lower harmonics remain dying out last. We will discuss this further in Chap. 8 after getting acquainted with the frequency-domain. But for now, let's assume that musical instrument sounds that have a sense of pitch are made up of a collection of harmonics (f_k), where each harmonic is ideally in integer relationship to the fundamental frequency f_0 ($f_k = k \cdot f_0$ where k is an integer number).

Another method to help with massaging the signal to make it behave for the purpose of extracting pitch information via the ZCR method is a topic that we already have touched upon — using a similar approach to the RMS method but without the "R" and "S." In other words, instead of using Eq. (4.7) we would simply apply a modified version of the RMS algorithm according to Fig. 5.3 which in its core employs the arithmetic mean.

The flowchart in Fig. 5.3 depicts a simple process: smoothing the signal by filtering out nuances and subtleties via the arithmetic mean, and degrading the time resolution which equates to essentially "time blurring" in order to help the ZCR algorithm work more robustly. Note that this is identical to the RMS algorithm except that now we just compute the arithmetic mean of the windowed portion of a signal (frame) and add an interpolator to the end of the algorithm. The interpolator just serves

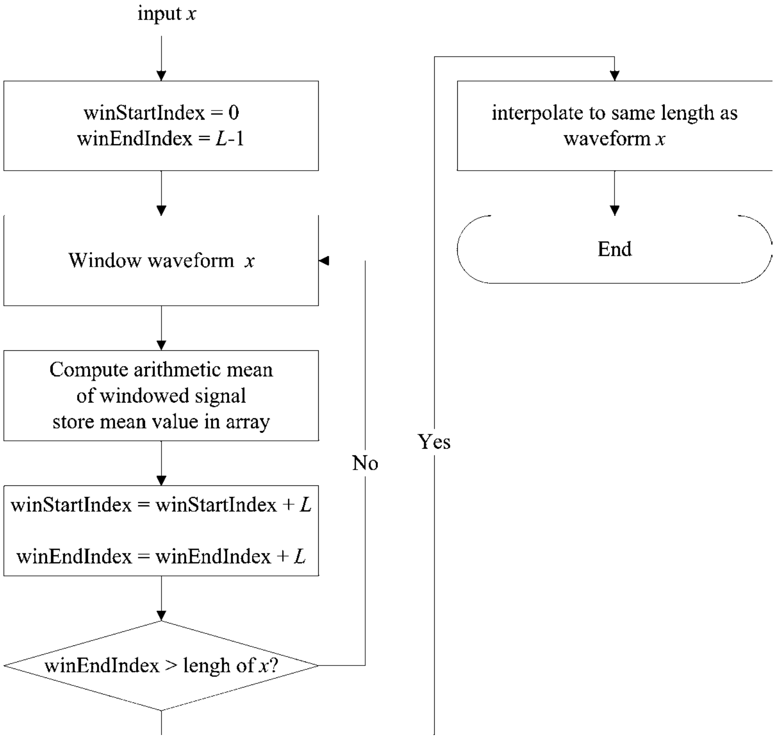

input x
winStartIndex = 0
winEndIndex = L-1
Window waveform x
Compute arithmetic mean of windowed signal store mean value in array
winStartIndex = winStartIndex + L
winEndIndex = winEndIndex + L
winEndIndex > lengh of x?
No
Yes
interpolate to same length as waveform x
End

Fig. 5.3. Frame by frame mean computation and interpolation.

to increase the number of samples to the original length of the signal. Remember that when we window a signal, we will experience loss in the number of samples when compared to the original signal. For example, if we have a signal that is 100 samples long and we use a window size of 10 samples, we will be able to fit in 10 windows resulting in 10 *frames* as they are often referred to. Thus, the number of total samples has decreased 10-fold due to windowing.

Figure 5.4 shows this algorithm in action using a window size of 50 samples (1.13 ms) for computing the arithmetic mean. This type of technique to smooth out a signal is referred to as filtering which we will formally introduce in Chap. 5. This particular type of filter is called the *moving average filter* as it moves through the waveform and computes the average (arithmetic mean). We can see that there is great improvement made if we compare the zero-crossing locations of the top and bottom part of Fig. 5.4, ultimately improving pitch computation using the ZCR method.

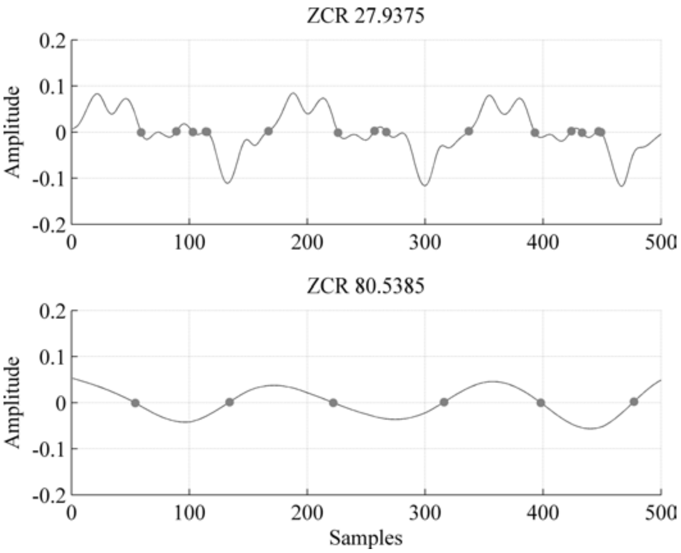


Fig. 5.4. ZCR locations of original (top) and using averaging, and interpolation for smoother waveform (bottom).

5.2 *Autocorrelation*

Autocorrelation is another very popular method for pitch detection that works in the time-domain. The algorithm is based on the notion of *similarity measurement* of itself defined as the sum of products as shown in Eq. (5.5).

$$acf_{xx}[\tau] = x[\tau] * x[-\tau] = \sum_{n=0}^{N-1} x[n] \cdot x[n+\tau] \tag{5.5}$$

In Eq. (5.5), τ is the lag (discrete delay index), $acf_{xx}[\tau]$ is the corresponding autocorrelation value, N is the length of the frame (portion for analysis using a rectangular window), n the sample index, and when $\tau = 0$, $acf_{xx}[\tau]$ becomes the signal's power (squaring and summing the signal's samples within the specified window). Similar to the way RMS is computed, autocorrelation also steps through windowed portions of a signal where each windowed frame's samples are multiplied with each other and then summed according to Eq. (5.5). This is repeated where one frame is kept constant while the other ($x[n+\tau]$) is updated by shifting the input ($x[n]$) via τ.

What does all this mean? We know that ideally, if a signal is periodic, it will repeat itself after one complete cycle. Hence, in terms of similarity, it will be most similar when the signal is compared to a time-shifted version

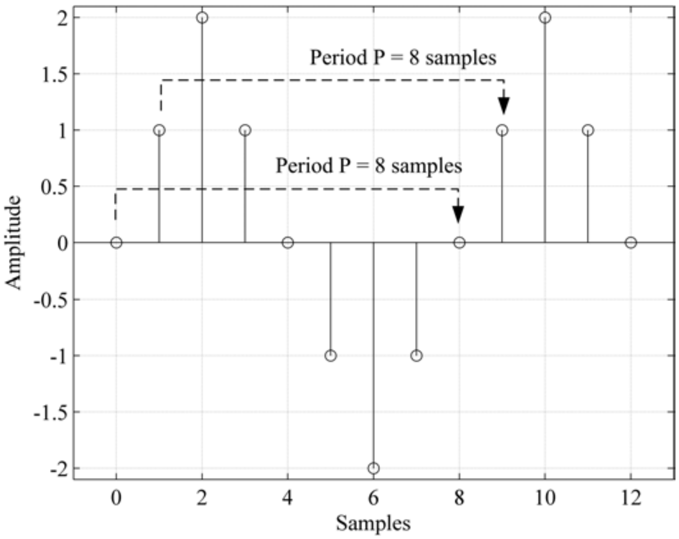


Fig. 5.5. A sine wave with period $P = 8$.

of itself in multiples of its period/fundamental frequency. For example, let's consider the signal in Fig. 5.5 representing a low sample rate sine wave. We know that a perfect sine wave will repeat itself after one cycle, which is every period $P = 8$ (every 8th sample) in the example shown in Fig. 5.5. In the autocorrelation algorithm, the shifting is achieved via τ starting with $\tau = 0$ and stepping through the sine wave by increasing the lag τ. Let's step through a few values of lags τ to get a better grasp of the idea behind the autocorrelation algorithm.

$$ACF[0] = \sum_{n=1}^{2} x[n] \cdot x[n+0] = \sum_{n=1}^{N} x[n]^2 = 1(1) + 2(2) = 5$$

$$ACF[1] = \sum_{n=1}^{2} x[n] \cdot x[n+1] = 1(2) + 2(1) = 4$$

$$ACF[2] = \sum_{n=1}^{2} x[n] \cdot x[n+2] = 1(1) + 2(0) = 1$$

$$ACF[3] = \sum_{n=1}^{2} x[n] \cdot x[n+3] = 1(0) + 2(-1) = -2$$

$$ACF[4] = \sum_{n=1}^{2} x[n] \cdot x[n+4] = 1(-1) + 2(-2) = -5 \tag{5.6}$$

$$ACF[5] = \sum_{n=1}^{2} x[n] \cdot x[n+5] = 1(-2) + 2(-1) = -4$$

$$ACF[6] = \sum_{n=1}^{2} x[n] \cdot x[n+6] = 1(-1) + 2(0) = -1$$

$$ACF[7] = \sum_{n=1}^{2} x[n] \cdot x[n+7] = 1(0) + 2(1) = 2$$

$$ACF[8] = \sum_{n=1}^{2} x[n] \cdot x[n+8] = 1(1) + 2(2) = 5 = ACF[0]$$

Figure 5.6 further shows a graphical version of the computed autocorrelation values from the above example with $N = 2$ and lag (τ) values from 0 to 8. Figure 5.7 shows the results of the autocorrelation calculation. We immediately notice that $ACF[0] = ACF[8]$ and if you further compute the autocorrelation values beyond $\tau = 8$, you will see that $ACF[1] = ACF[9]$ or in general:

$$ACF[\tau] = ACF[\tau + P] \tag{5.7}$$

In other words, whenever there is a shifting of P samples, the autocorrelation result will show a maximum value which gives us vital information regarding a signal's periodicity. The frequency and the pitch value (fundamental frequency) can now be computed using Eq. (5.8) where f_s is the sampling frequency in samples/sec and P the period in samples.

$$f_{Hz} = \frac{f_s}{P} \tag{5.8}$$

Figure 5.8 shows the autocorrelation results for the same piano sound we used with the ZCR method without any modification of the signal (no filtering). In this example, the fundamental frequency ($P = 168$ samples) is quite clearly visible by inspection and can easily be detected with a peak detection algorithm. For example, one could simply design a rough peak detector for this situation by computing all peaks via slope polarity change and filtering out all peaks that are lower than say 40 in their energy.

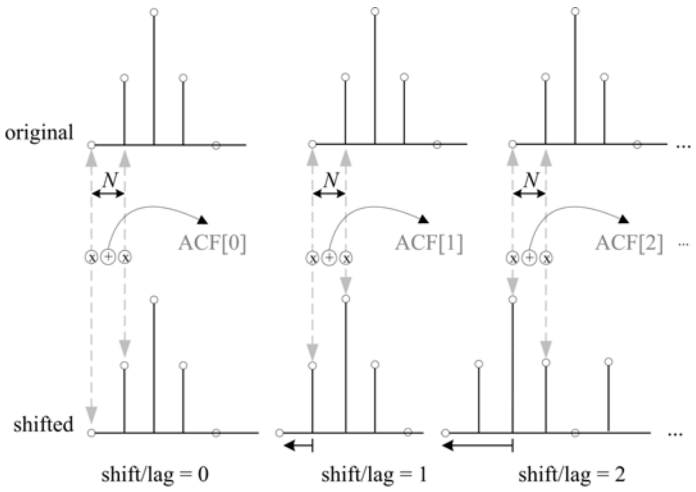


Fig. 5.6. Autocorrelation with $N = 2$, lag $\tau = 0 \ldots 8$.

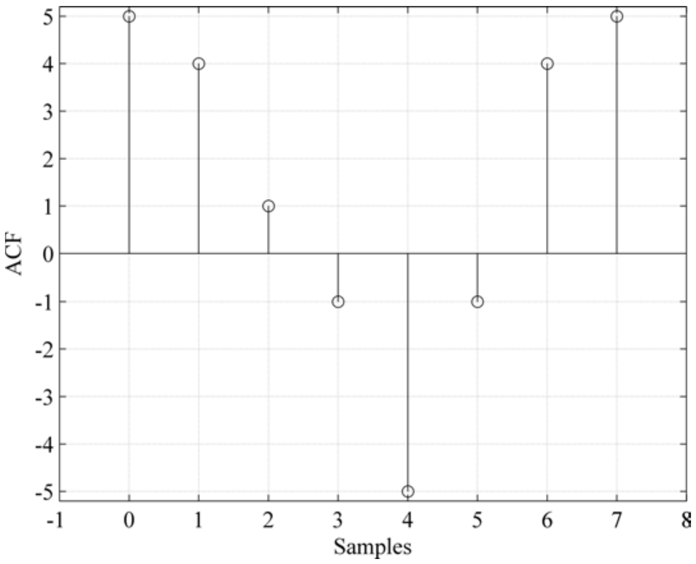


Fig. 5.7. Autocorrelation values for $N = 2$ and lag up to 8.

The autocorrelation-based fundamental frequency computation algorithm is summarized in Fig. 5.9. The slope itself is computed simply via (5.9).

$$\text{slope} = \frac{x[n] - x[n-1]}{2} \tag{5.9}$$

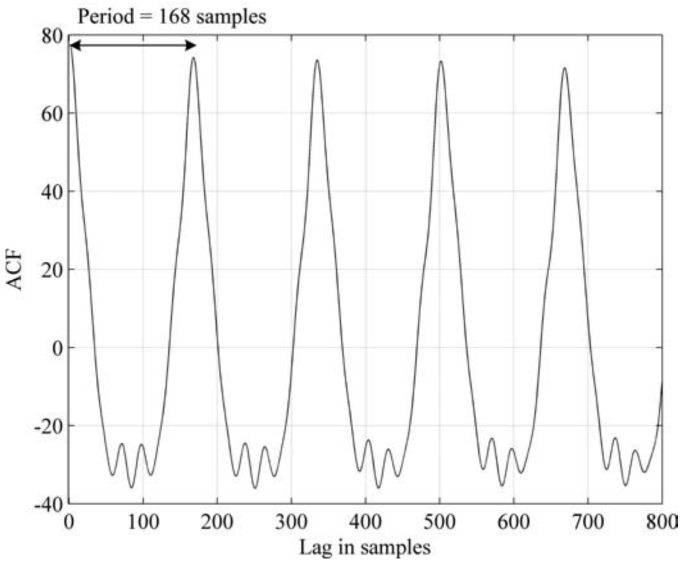


Fig. 5.8. Autocorrelation results for a piano sample.

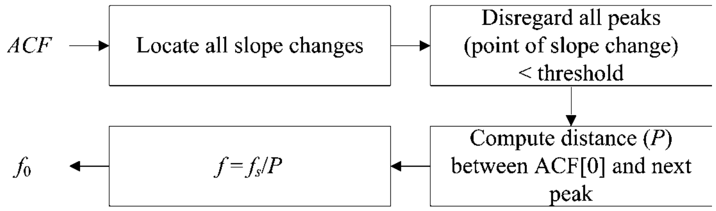

ACF
Locate all slope changes
Disregard all peaks (point of slope change) < threshold
Compute distance (P) between ACF[0] and next peak
f = fs/P
f0

Fig. 5.9. Autocorrelation pitch computation.

In general, a larger frame size (N: frame length) results in clearer autocorrelation peaks as we have more data for analysis. However, the longer the frame length, the less transient response we will obtain, as we compute the ACF over a longer segment of time. If the pitch changes rapidly, using a larger frame size will cause inaccurate computation of the pitch values as we lose transient information. Hence, there has to be some sort of compromise between transient response and the accuracy in computing the fundamental frequency. Also, we can improve the performance of a fundamental frequency detector if we know beforehand what instrument it is that we want to analyze. If for example you were to design a guitar tuner using the autocorrelation method, you would not need to worry about frequencies that belong to the bass guitar only for example and hence can make the maximum frame size considerably shorter (f is inversely proportional to the wavelength λ) improving the overall transient response.

Another method to potentially improve transient response in the autocorrelation algorithm is through a dynamically changing frame size. This method simply uses a "very" long frame size as a guide which is determined by the lowest allowed fundamental frequency. This long frame is used in guiding the search for sharp peaks as longer frame sizes result in clearer and more pronounced maxima. The basic idea is exploiting the fact that if the computed period is short and hence the pitch detected is "high,"

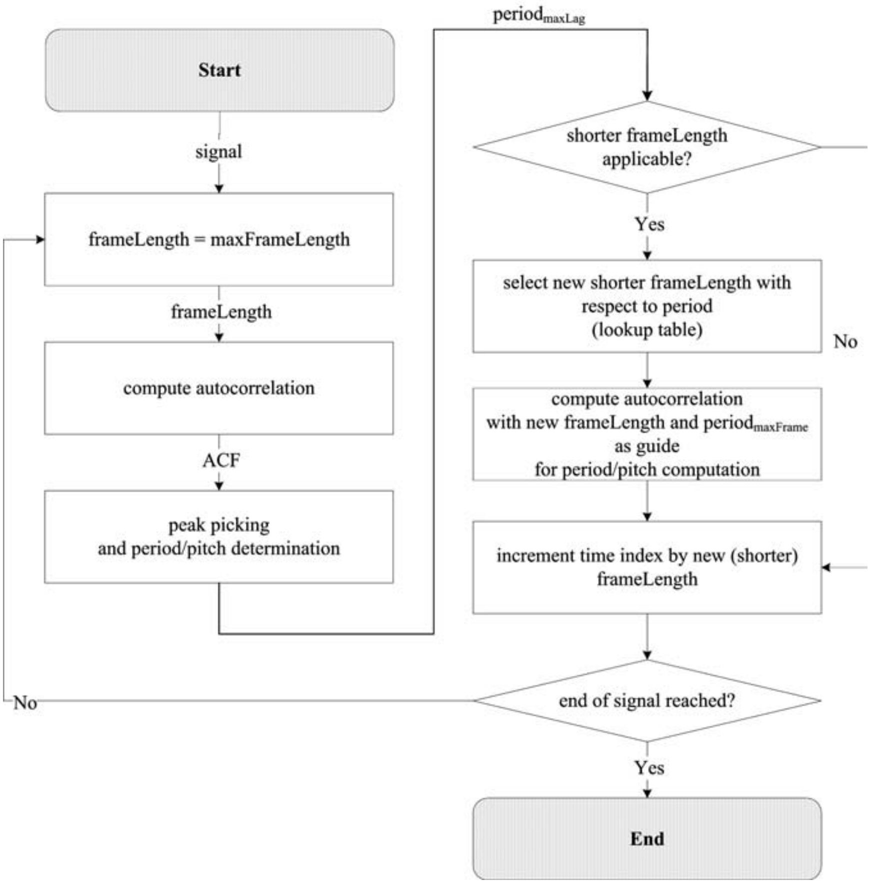

Start
signal
frameLength = maxFrameLength
frameLength
compute autocorrelation
ACF
peak picking and period/pitch determination
period_maxLag
shorter frameLength applicable?
Yes
select new shorter frameLength with respect to period (lookup table)
No
compute autocorrelation with new frameLength and period_maxFrame as guide for period/pitch computation
increment time index by new (shorter) frameLength
end of signal reached?
No
Yes
End

Fig. 5.10. Adaptive lag for improved transient response (Park 2000).

then there would be no need to utilize a long frame to get the job done — a "shorter" frame size will be sufficient. The ensuing benefit is that whenever possible the algorithm selects a shorter frame length thus resulting in improved transient response. This algorithm is outlined in Fig. 5.10.

Although the autocorrelation method works robustly in many cases and is used widely in cell phone and voice coding applications as well as for musical signals, there are probably a lot of instances where it does not do a proper job in computing the fundamental frequency. This often occurs when the local peaks that are not part of the fundamental frequency are not negligible and small enough unlike the local peaks in Fig. 5.8. The problem thus arises as those local peaks become quite large and therefore complicate the peak picking process considerably. We could again apply a moving average filter as we did with ZCR to alleviate this problem, but the bottom line is that there is no perfect, bullet-proof algorithm that will detect all pitches for all instruments with total accuracy. At the end of the day, not only is it difficult to deal with the technical aspects of the algorithms used, but it is also very difficult to compute the pitch itself, as pitch is very much a *perceptual* aspect of sound — psychoacoustic considerations have to be taken into account.

5.3 *Cross-correlation*

Cross-correlation is identical to autocorrelation with the difference being that two different signals are used instead of one to find the correlation values according to Eq. (5.10). Here x_1 and x_2 are two different types of signals, N is the frame length, τ the lag, and n the sample index as usual.

$$r_{x_1 x_2}[\tau] = \sum_{n=0}^{N-1} x_1[n] \cdot x_2[n+\tau] \tag{5.10}$$

For example, instead of using the same waveform as we did in autocorrelation, with cross-correlation we could use a piano waveform and a violin waveform and compute the cross-correlation values between those two signals. Thus, with cross-correlation, we will be able to measure correlation factors between different signal types (for whatever purpose). For example, we could possibly analyze the cross-correlation values between some input waveform with a database of 100 other waveforms stored on our hard-drive to determine similarity relationships to roughly categorize the input sample automatically according to the cross-correlation measure.

6 Sample Rate Conversion

Sample rate conversion refers to the process of changing the sampling frequency of a signal either by adding samples or by removing samples. The process of adding samples is referred to as *up-sampling* and removing samples *down-sampling*. In this section, we will look at how samples are added and removed and will also discuss some of the issues involved again from the time-domain perspective. We will revisit this topic in Chap. 8 and learn more about artifacts and distortion to the resulting signal after introducing the Fourier transform.

6.1 *Up-sampling*

Up-sampling is a process where the number of samples of a signal is increased by adding zero-valued samples as defined in (6.1) where L is the integer up-sampling amount for input signal x with sample index n.

$$y[n] = \begin{cases} x[n/L], & \text{if } n/L \text{ is integer} \\ 0, & \text{if } n/L \text{ is non-integer} \end{cases} \tag{6.1}$$

The above equation may seem a little bit confusing but if we plug in a few numbers it should become clear as to what is going on. For example, if $x[n] = n$, where $n = 0, 1, 2, 3, 4$ and let $L = 2$ we have:

$$\begin{aligned} y[0] &= x[0/2] = x[0] = 0 \\ y[1] &= x[1/2] = 0 \\ y[2] &= x[2/2] = x[1] = 1 \\ y[3] &= x[3/2] = 0 \\ y[4] &= x[4/2] = x[2] = 2 \\ y[5] &= x[5/2] = 0 \\ y[6] &= x[6/2] = x[3] = 3 \\ y[7] &= x[7/2] = 0 \end{aligned} \tag{6.2}$$

From (6.2) we can see that the up-sampled output now is twice ($L = 2$) the size of the original signal x, where zeros are interleaved between the original samples. This is shown in Fig. 6.1.

The leftmost plot shows the original samples and the middle plot the up-sampled version. Note that the up-sampled output has expanded to

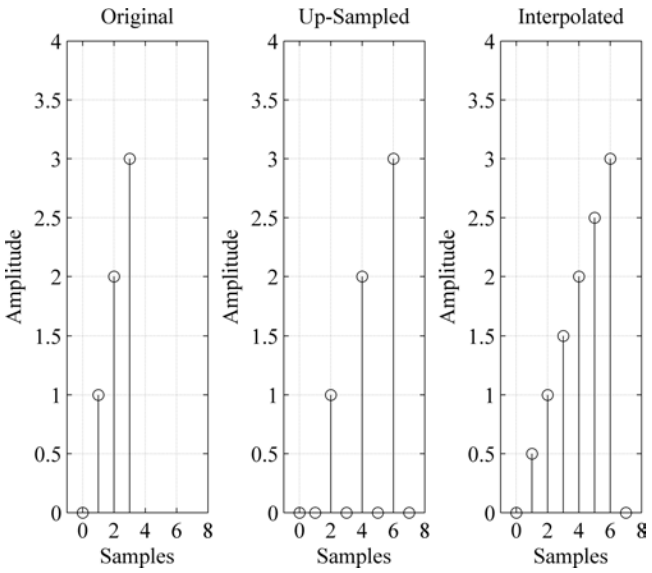


Fig. 6.1. Original signal (left), up-sampled signal (middle), and up-sampling with linear interpolation (right).

twice the number of samples of the original with $L = 2$ and is interleaved with zeroes. Obviously this is not a smooth waveform and when up-sampling signals without smoothing, problems occur. We will get into the details what these issues are in Chap. 8 when we deal with artifacts in the frequency-domain, but for now let's intuitively assert that some sort of smoothing needs to be done in order to make the up-sampled signal look more like the original signal. This smoothing is rendered via interpolation. The rightmost plot in Fig. 6.1 is an up-sampled version of x with linear interpolation (taking the mean of adjacent samples) which makes the resulting up-sampled and interpolated signal smoother completing the overall up-sampling procedure.

6.2 *Down-sampling*

Down-sampling is the opposite of up-sampling and is much like the process of digitizing an analog signal (continuous time/amplitude signal). It can be likened to sampling as the ensuing signal after down-sampling is a limited representation of the original signal. In other words, the number of total samples is reduced when down-sampled similar to digitizing analog signals.

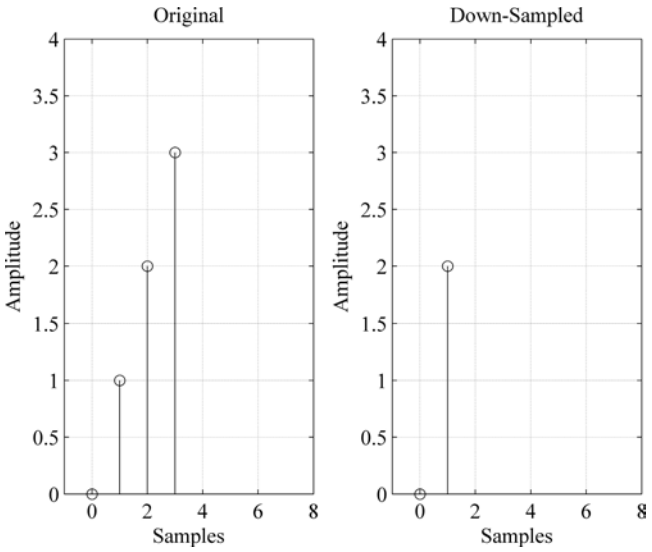


Fig. 6.2. 2-fold down-sampling.

The formal definition of M-fold down-sampling is shown in Eq. (6.3) where M is the integer down-sampling amount for input x and n the sample index.

$$y[n] = x[n \cdot M] \tag{6.3}$$

For example, when $M = 2$ we get the results shown in Fig. 6.2.

As we may have guessed from our introduction of aliasing and sampling in Chap. 1, since down-sampling is essentially a process of limiting a signal (similar to sampling an analog signal), aliasing becomes a problem. If the input x is not band-limited properly before down-sampling, aliasing will occur. We will discuss what sort of *anti-aliasing filters* we will need by observing what happens to a non-band-limited down-sampling process in Chap. 8.

In general, it is also worthwhile to note that up-sampling and down-sampling can be useful in musical applications as modulating the sample rate will lead to interesting musical results. If the sampling rate of the sound output device is kept constant while the sampling rate of the signal is modulated, pitch/frequency characteristics as shown in Table 6.1 result.

Although pitch shifts down or up according to L and M (while f_s is kept the same for the audio output device), due to the increases (L) and decreases (M) of total number of samples, the sound characteristic or its timbre is

Table 6.1. Up-sampling and down-sampling effects on audio signals.

	Total number of samples	Pitch/frequency
Up-sampling	Increases L-fold	Shifts down L-fold
Down-sampling	Decreases M-fold	Shifts up M-fold

altered as well. It is possible to use down-sampling to mimic pitch-shifting pitches upwards, but the so-called "munchkin effect" becomes a hindrance unless of course we want the munchkin effect itself (try the MATLAB® code below on a human voice sample to hear the munchkin effect when using down-sampling). Conversely, up-sampling can also be used to expand signals in time — making a signal twice as long for example. Here also a similar problem arises as the pitch will change according to the addition of samples to the original signal.

```
%Load some audio sample into MATLAB
[x, fs] = wavread('myInstrumentSound.wav');
sound(x, fs)

% try it with up-sampling, for example
sound(upsample (x, 2), fs)

%or down-sampling, for example
sound (downsample(x, 2), fs)
```

Code Example 6.1.Up-sampling4 and down-sampling example

We will learn in Chap. 9 how to change pitch without changing the duration of a signal. In the next section we will show a simple method where we can time-stretch (lengthen) or time-compress (shorten) a signal using the *overlap-and-add* (OLA) algorithm without changing the pitch characteristics.

7 Overlap and Add (OLA)

When trying to make a signal longer or shorter through up-sampling or down-sampling, we observed that change in pitch is an inevitable byproduct which is probably more often than not undesirable. One solution to fixing this problem is achieved by a method called overlap and add (OLA). When using the OLA algorithm, it is possible to time-stretch or time-compress a signal without changing the pitch of the signal — for time-stretching we can make a given signal longer without "additional" signal information. The method is surprisingly simple in its core concept and is outlined in Fig. 7.1 and further described below.

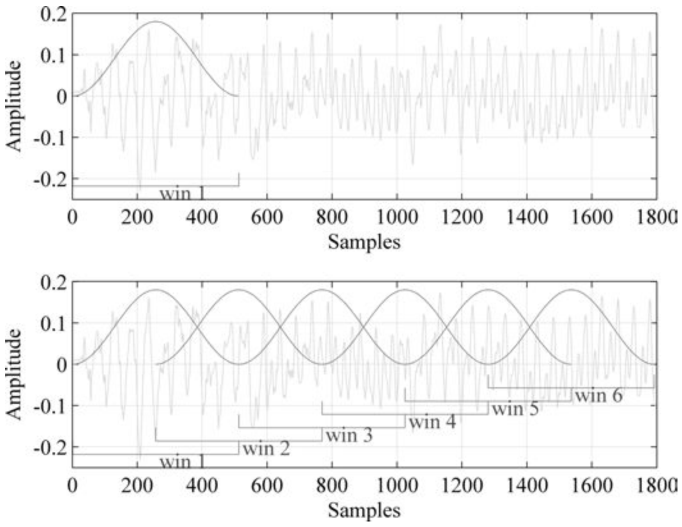


Fig. 7.1. Hann window (top) and OLA (bottom).

To learn the mechanics of this method, let's take the first 1800 samples of the piano sound we used before as an example. The top of Fig. 7.1 shows the Hann window we used to window a portion of the piano signal. In previous windowing examples, we observed that windows were applied in a similar fashion, whereby the next window's start sample index would begin just after the end of the previous window's right edge. In the bottom of Fig. 7.1, we notice that we have 6 windows not in a sequential non-overlapping manner as before, but in an overlapping configuration. To be more precise, in this example we use a 50% window overlap equivalent to 256 samples or 5 ms with a Hann window size of 512 samples equivalent to 110 ms. The start time of each new window is not at window size intervals as we have previously seen, but instead it is equal to the integer multiple of the window size minus the overlap percentage referred to as the *hop size.* The hop size in our example is 256 samples. With 90% overlap we would

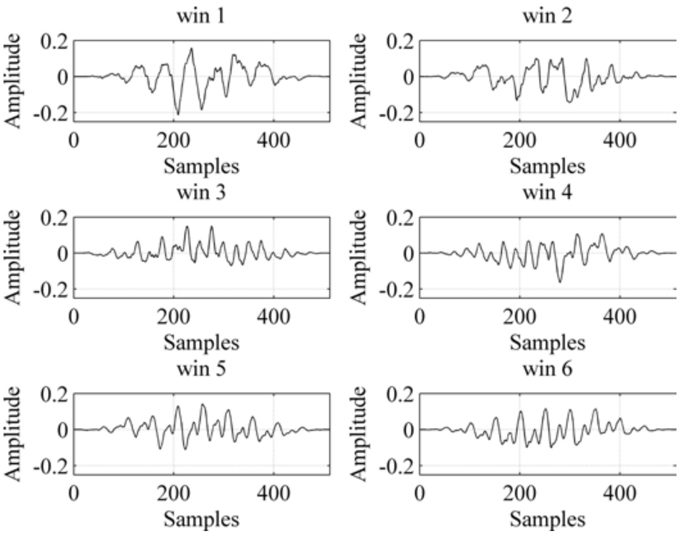


Fig. 7.2. Windowed portion of piano sample.

have a hop size of 180 samples. Figure 7.2 shows the windowed portions of the waveform in Fig. 7.1 which are all 512 samples in duration as specified. Now, if we use the 6 windowed portions of the signal and sequentially put them back together through addition with different overlap percentage, the result will be either a longer signal or a shorter signal depending on how much overlap we choose at the synthesis stage. Note that we could also have used a rectangular window, but opted for the Hann window as it has smooth edges which will help blend (cross-fade) adjacent windowed signals during the overlap and add procedure. The first phase of OLA where we window the signal is referred to as the analysis part and the second stage where we actually overlap-and-add the windowed signals is referred to as the synthesis part. The algorithm encompassing both the analysis and synthesis parts is referred to as overlap-and-add.

If we were to have an analysis overlap of 50% (256 samples) and a synthesis overlap of 75% (384 samples) the resulting signal after OLA will be 25% shorter (total number of samples 1280). If we were to use a synthesis overlap of 25% instead for example, we would have a time-stretched signal that would be 25% longer (total number of samples now 2250). What is interesting is that in either case, we will not experience any pitch alterations as there is no up-sampling or down-sampling involved in the OLA algorithm.

OLA works due to aspects of psychoacoustics and concepts related to gestalt theory that deal with perceptual grouping tendencies. You may remember from Chap. 1 that we perceive time events differently, depending on how fast or slow these events occur. When time events such as pulse trains are set at 1 Hz, we will actually be able to count them and possibly perceive them as rhythm if different accents are used for each pulse. As the frequency is increased we lose the ability to count these pulses and the perception goes from rhythm to sound color or timbre up until around 20 Hz or so. After 20 Hz the pulse train, which really has not changed in structure other than an increase in the number of pulses per second, gradually modulates and is perceived as pitch. As the frequency is further increased towards the upper pitch limits of our hearing system, the perception of pitch now changes to the perception of a *sense* of high frequency. The OLA algorithm thus exploits psychoacoustic characteristics and ambiguity of perception in the 20 Hz to 30 Hz range, equivalent to 30 ms to 50 ms, by applying small window sizes as well as short hop sizes to render the perception of audio signals becoming shorter or longer with little or (ideally) no change in pitch and timbre.

7.1 *OLA: Problems and solutions*

Once again, something as simple as the OLA algorithm rarely works well off-the-shelf and predictably needs further tweaking for the algorithm to work robustly. One of the common problems that accompany the out-of-the box basic OLA arise due to the periodic spacing (constant hop size) of the windows — anything that is periodic and in the perceptual hearing range is detectable by the listener (assuming it is evident enough). Looking back at Fig. 7.1, we note a type of pattern emerging from the combined windows resulting in a hilliness in the envelope. This problem relates to the psychoacoustic issue addressed above — perception of time events as rhythm, roughness, and pitch. In other words, if the *hop size* along with the window size is short enough, the artifacts caused by destructive and constructive interference (see Chap. 4 for details on this type of interference) may either manifest themselves as added rhythmic modulation, roughness, or pitch. A solution to this is applying non-periodic hop-sizes by slightly varying the overlap between windows randomly as we step through the signal. Although this may alleviate some of the rhythmic or pitch-based artifacts, this tweak may introduce noisiness making the resulting output "rougher" sounding.

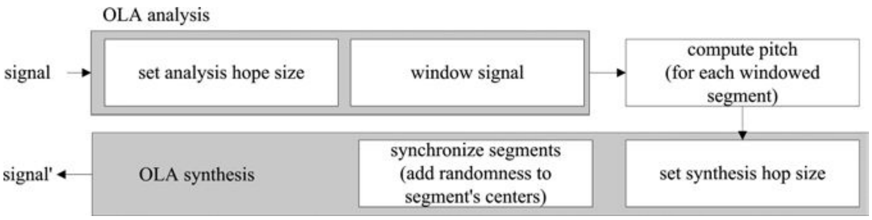

OLA analysis
signal
set analysis hope size
window signal
compute pitch (for each windowed segment)
signal'
OLA synthesis
synchronize segments (add randomness to segment's centers)
set synthesis hop size

Fig. 7.3. PSOLA algorithm.

Another solution in improving the sound quality of the OLA for time expansion and compression for pitch-based sounds in particular, is using the so-called *pitch synchronous overlap-and-add* (PSOLA) approach, also sometimes referred to as the time-domain *harmonic scaling method.* The algorithm itself is actually quite simple whereby the center of each window is aligned and synchronized with the fundamental frequency of the windowed portion of the signal as outlined in Fig. 7.3. The center of each window can be strictly an integer multiple of the fundamental frequency or it can be randomized slightly so that each window's center is not in complete synchrony all the time to dampen effects of overly accentuating aspects of the fundamental frequency in the final output. With PSOLA, because the periodic nature of the windows is masked out by the fundamental frequency, the artifacts caused by OLA are lessened as a consequence. Obviously, in order to get a properly working PSOLA algorithm one must have a properly working pitch tracker, and signals that do not elicit a sense of pitch will not work with the PSOLA algorithm.

8 Musical Examples

Being a bass player I very much appreciate the song *Another One Bites the Dust* by the legendary rock band Queen, released in the early 1980s. The simplicity, yet solid and almost mechanical feel of the rhythmic patterns along with the essential vocal rhythms, without a doubt play a big part of this song's popularity, shooting the mercury thermometer way up the scale. Interestingly, however, are also some of the techniques Queen used in producing this track, which includes reverse playback of piano samples and cymbals sounds, as well as other effects that we take for granted nowadays as they are easily done on an inexpensive desktop computer or laptop. Back in the 1980s it was hardly as easy to do, and what's even more interesting for a rock band is that the song was seemingly (according to some urban legends) built on a "drum and bass loop" — recording a drum pattern and playing that pattern over and over again, commonly done nowadays with sample CDs. Whether this is true or not is beside point, but the song nevertheless brought the machine-like accompaniment structure to the forefront in popular music. Another noticeable part of this song is the alleged *back-masking* and supposedly hidden messages encoded into the waveforms. I will leave it up to you to reverse the whole song and look for any messages there may or may not be. Furthermore, whether you will find something in the waveforms is not that important, but it is nevertheless interesting to see a rock band embrace electronic sound modulation techniques and actively utilize sounds played backwards thus seemingly hinting and encouraging the listener to play the whole song in reverse.

Another pop band that is quite interesting, exploiting DSP techniques in the form of looping, is Radiohead. Albeit looping in this context is not in the microsecond units as is with wavetable synthesis, Radiohead, like many popular bands in the past (*Pink Floyd* and *Money* comes to mind) and especially today with the advent of the computer technology in music, are using sampling and looping techniques to exploit existing segments of previous musical works as a source for musical inspiration. A song called *Idioteque* (2000) is such an example. What makes this work even more intriguing is that Radiohead did not use a portion of a pop song or a "classical" piece but rather opted to sample a short portion of composer Paul Lansky's *mild und leise* which is a piece generated completely via the computer using FM synthesis (see Chap. 4 for details on FM synthesis) and special filters written in Fortran IV by Ken Steiglitz with the harmonic language based on George Perle's 12-tone modal system. *mild und leise* is certainly not your everyday pop music piece and belongs to the genre of art music, or more specifically electro-acoustic and computer music. What surprised me most is how those Radiohead guys found the LP in the first place, and what sort of inspiration from the original piece lead them to use it as the fundamental riff for *Idioteque*. They do eventually create a totally new piece that has very little resemblance to the original character of the piece written 27 years earlier, but the looped sample is unequivocally identifiable. Speaking of electro-acoustic compositions, one intriguing work that utilizes time-domain based DSP processes is *Leviathan* by composer William Schottstaedt. The piece is a time-domain based compositional/signal processing undertaking using 14 sound sources from a ubiquitous "effects LP" including sounds from a train, milk machine, rooster, ship's mast, and water pump. The materials for this *tape piece* as it is referred to, comes from these basic sound sources cut up into 15,000 splices, which are looped and linked together via a custom software program. To add subtle variance to the loops, the composer changed the loop points slightly in the sound objects which resulted in minute differences in the perception of the same sample. It is quite a musical ride in the art of sampling and is a great example how simple sampling techniques can lead to wonderful works of art.

References and Further Reading

Cook R. Perry 2002. Real Sound Synthesis for Interactive Applications. A. K Peters, Ltd.

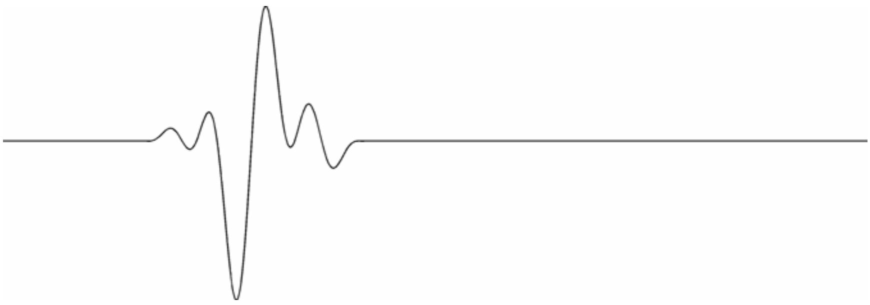
Davis, G. and Jones R. 1989. The Sound Reinforcement Handbook. Milwaukee: Hal Leonard Corporation.

Park, T. H. 2000. "Salient Feature Extraction of Musical Instrument Signals". Master's Thesis, Dartmouth College.

Rabiner, L. R. and Schafer R. W. 1978. Digital Processing of Speech Signals. Prentice-Hall, Englewood Cliffs, NJ.

Chapter 3

TIME-DOMAIN PROCESSES II



1 Introduction

In Chapter 3 we will continue with topics in the time-domain and discuss a number of classic computer music synthesis techniques. Some of the sound synthesis algorithms and techniques discussed in this chapter will include granular synthesis and waveshaping synthesis. We will also introduce a number of popular and widely used DSP effects including the dynamic compressor, dynamic expander, distortion, chorus, and flanging effects commonly encountered in a musician's home, professional studio, and gig bag of a guitar player. Some of the topics covered here will be revisited in later chapters to reinforce our understanding through continued exposure to various concepts and techniques in signal processing.

2 Granular Synthesis

We started off the book introducing the sine wave and presenting audio signals in terms of their wavelike properties — a common way in describing audio signals. However, we are probably also familiar with the particle-like behavior of sound. That is, sound adhering to behavior and theories pertinent to light which also exhibit both particle (photons) and wavelike characteristics. The basis of *granular synthesis* can be traced to the particle theory side of sound. Seemingly, the notion of the *sonic grain* which makes up the particles in granular synthesis, was first introduced in the Dutch scientist Isaac Beeckman's journals which he kept for 30 years until his death in 1637. The sonic grain is later found in the work of Nobel Prize physicist (holography) Dennis Gabor (Gabor 1947), which focuses on the theory of acoustical quanta. The coining of term *grain*, however, is attributed to Iannis Xenakis who has conducted much research in this area and composed musical works based on grains of sounds.

The basic concept of granular synthesis, like many great ideas it seems, is quite simple: use elementary sonic particles (grains) of short duration and juxtapose them horizontally (time) and vertically (amplitude/density) to produce a sound object or *sound cloud*. It is somewhat akin to pixels on the computer monitor where each pixel by itself (comparable to the grain) has little meaning and is not meant to be perceived individually but as a group, where en masse they form a clear and vivid visual image. This idea of an image also exists in sound as a sonic image if you will — if an appropriately small sonic grain and parameters such as grain density and amplitude are carefully defined. One noticeable difference between the pixel model and the grain model is that the grains are often not evenly and sequentially distributed as pixels are, but are rather randomly sprayed onto the sound canvas as further described in the next section. Not surprisingly, however, once again, the main reason granular synthesis works at all is due to our hearing system's limitations and psychoacoustic issues associated with human perception of sounds. Granular synthesis is actually in its core very similar to the OLA algorithm we discussed in Chap. 2, with the major difference lying in the verticality aspect of the windowed signals (grains) in granular synthesis. This verticality represents the density of grains at a given point in time as we shall see below when we introduce details of the granular synthesis algorithm.

2.1 *Basic granular synthesis parameters*

In this section we will introduce the main parameters and concepts utilized in granular synthesis. Table 2.1 shows these basic parameters typically used for granular synthesis.

Table 2.1. Basic granular synthesis parameters.

Parameter	Description
Grain size	Duration of grains typically in milliseconds
Window type	Window shape for excerpting a grain
Grain amplitude	Maximum amplitude of grain
Grain density	Number of grains per second
Grain pitch	Pitch (if present) characteristics

We will first and foremost start by defining the grain which is obviously the most important component in granular synthesis. Grains are defined as small windowed portions of an audio signal typically anywhere between a few milliseconds to 100 milliseconds in duration. The choice for window length is directly related to what sort of effect we want, including the degree of recognition of the original sound source — the window length directly affects the recognition and identity of a single grain upon audition. The shorter the grain size, the more blurring of the identity of the origin of the source when hearing one individual grain. In determining the duration of a grain, frequency content of the original sound object whence the grain comes from also plays a role, perhaps a more subtle one as is evident from Gabor's experiments — Gabor showed that the duration of the grain, so as to be able to recognize the grain's characteristics (such as a grain's pitch information), depended on the frequency characteristics of the grain itself (Gabor 1946). That is, the threshold of discernment of the grain's original make-up is a function of the original signal's frequency composition. The aspect of frequency composition refers to the theory of representing a sound object via a sum of sinusoids at specific amplitude, frequency, and phase values as will be discussed in Chap. 8.

In order to produce a sonic grain we will need to apply a window on a portion of the target signal (windowing is discussed in Chap. 2, Sec. 3). The grain is defined as a windowed portion of input samples $x[n]$ using window $w[n-L]$ where L is the delay (offset or start point where windowing takes place) as shown in (2.1).

$$x_{grain}[n] = w[n-L] \cdot x[n] \tag{2.1}$$

Some common window types for grains include the Hann and triangular windows (see Chap. 8 for details on window types) as shown in Fig. 2.1. Both window types display smooth boundaries tapering off on either side

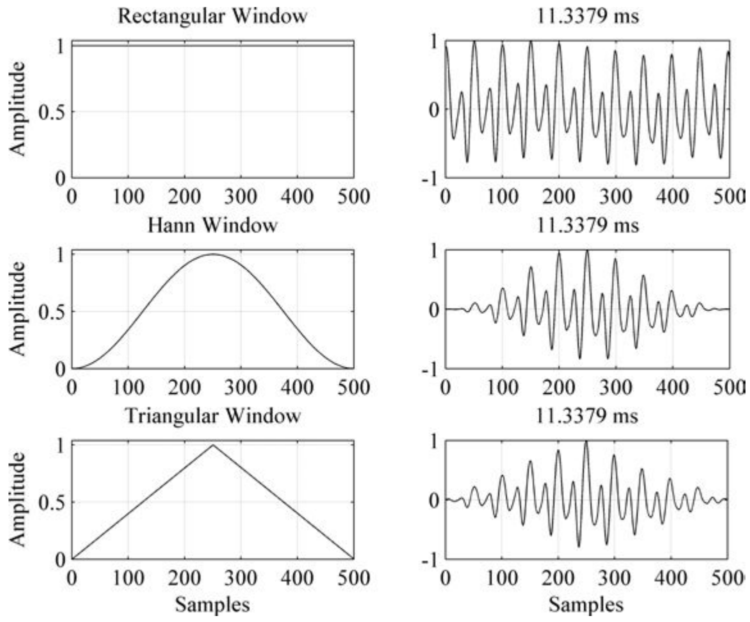


Fig. 2.1. Rectangular, Hann, and triangular windows with respective grains.

which help when mixing and blending the grains during the summing process.

For example, if the input signal $x[n]$ is defined over $n = 0 \ldots 4999$, we could take an excerpted grain from the beginning of the input with $L = 0$ window length $N = 100$ which would be equivalent to windowing the input signal portion $x[0]$, $x[1], \ldots x[98]$, $x[99]$. When $L = 100$ and $N = 100$ we would window the input signal portion $x[100]$, $x[101], \ldots, x[198]$, $x[199]$ with the same window w.

The grain amplitude can be controlled by simply multiplying the grain with a scalar as shown in Eq. (2.2). If we take these individual grains and arrange them appropriately as a group, and when positioned appropriately on the time axis, they will produce the desired amplitude envelope. Thus, controlling the overall amplitude profile can simply be achieved by adjusting the individual amplitude levels of each grain. This can be compared to the pixels on the computer screen, whereby one can brighten a certain area on the monitor by increasing the brightness of a group of pixels in that area.

$$x'_{grain}[n] = \alpha \cdot x_{grain}[n] \tag{2.2}$$

The density of grains, usually defined as grains per second, in general contributes to the richness of the resulting synthesized signal. That is, the greater the density, the richer and louder the sound. As is the case with the ability to control the amplitude of each grain, it is also possible to control the pitch of each grain using up-sampling and down-sampling techniques. This is further discussed in Sec. 2.4.

2.2 *Asynchronous granular synthesis*

One of the most popular types of granular synthesis is asynchronous granular synthesis which we will cover in this book (others include *quasi-synchronous* and *pitch-synchronous granular synthesis*). The main characteristic of the asynchronous version is that when grains are laid out on the time axis, they are spread out randomly as seen in Fig. 2.2. That is, the start time indexes of the grains do not follow a pattern or periodic design, although the total number of grains per second (density) is maintained.

The top part of Fig. 2.3 shows a one second granular synthesis example using a piano sound source for its grains of 500 samples (11.34 milliseconds) in duration, Hann window, grain density 20,000 grains/sec, an exponentially decaying amplitude envelope, and sampling rate of 44.1 kHz. The grains are randomly distributed over a one second period and summed to form the synthesized output. The bottom of Fig. 2.3 shows the first 2000 samples of the synthesized signal displaying traces of the original grain from Fig. 2.1.

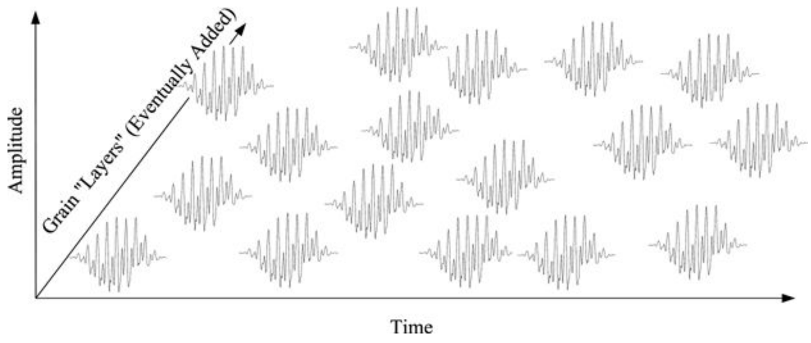


Fig. 2.2. Grains added as layers asynchronously.

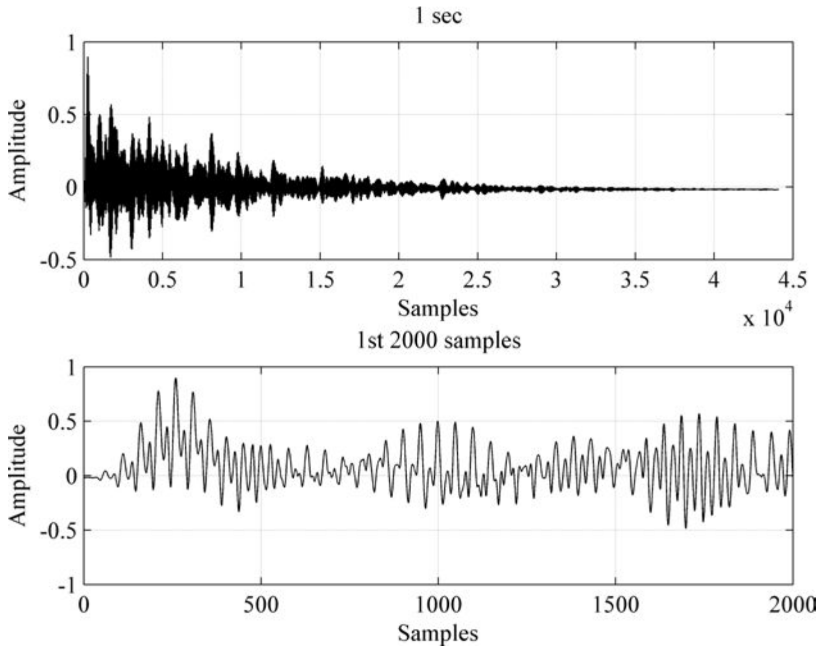


Fig. 2.3. Asynchronous granular synthesis.

2.3 *Pitch shifting and time stretching/compression*

Pitch modification and time-stretching and compression are straightforward in granular synthesis since we are dealing with elemental sonic grains. To change the pitch all we need to do is either down-sample or up-sample to the desired pitch (with appropriate anti-aliasing filtering as discussed in Chap. 7). Altering the duration using granular synthesis is just as uncomplicated — populating a desired duration with sonic grains is all it takes. For example, if the original sound object is only one second long and we want to time stretch it to 3 seconds, all we would need to do is fill 3 seconds with the windowed grains using a suitable grain density value. Of course to make the timbre sound "natural," further tweaking needs to be done, including excerpting different types of grains from the signal's attack, decay, steady-state, and release regions according to your needs — each region of the original sound-byte more often than not exhibit dissimilar timbral qualities.

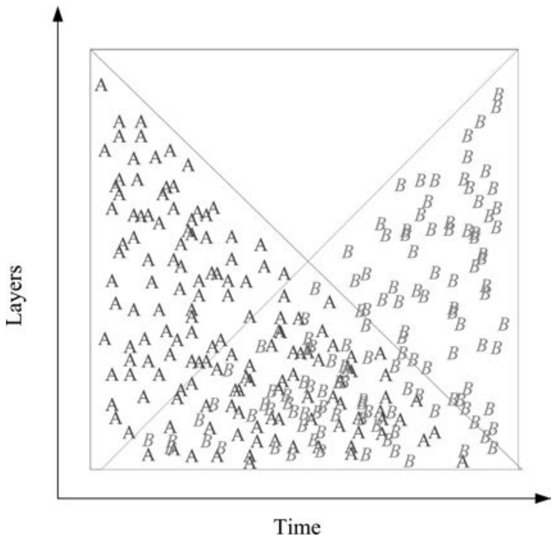


Fig. 2.4. Morphing from timbre A to B.

2.4 ***Sound morphing with granular synthesis***

It is also possible to use granular synthesis to morph between different types of sound objects or to synthesize hybrid sound objects that take grains from two or more different audio signals. For example, if the objective is to smoothly morph from timbre A to timbre B, one could start off with grains from sound source A (100%) only, followed by a transition area which gradually decreases the contribution of grains from A while increasing the ratio of grains from B, and continuing until we are only left with grains from B (100%). This idea is depicted in Fig. 2.4. In this example, the cross-fading between the two types of sound source is linear but it could also just as easily be exponential or some other nonlinear curve.

3 Amplitude Distortion and Waveshaping

In Sec. 3, we will introduce a number of modulation techniques found in the time-domain that deal with manipulating and distorting the amplitude values (dynamics) of a signal. In a way, these types of processes can be thought of shaping the input waveform and therefore fall into the general

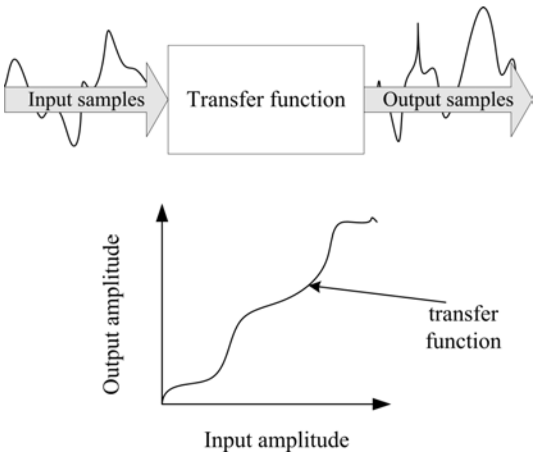


Fig. 3.1. Waveshaping synthesis.

category of *waveshaping*. Waveshaping in general, is a time-domain-based sound modulation method based on *shaping* the waveform according to a *transfer function*. The core concept is shown in Fig. 3.1 — the input signal's amplitude values are shaped or modified by a transfer function which is usually nonlinear (we will discuss linearity in more detail in Chap. 5).

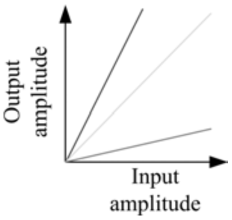


Fig. 3.2. Linear transfer functions.

If the transfer function were linear, it would look like the examples in Fig. 3.2 expressed as $y = a \cdot x$, where a is a scalar representing the slope of the line. Waveshaping-based techniques are also referred to *distortion synthesis* as the input is essentially distorted to produce a desired output much like the dynamic compressor which will be discussed shortly. What makes waveshaping synthesis so interesting is that as the input signal's amplitude value changes as a function of time, so will the output of the *waveshaper*, resulting in a dynamically changing output signal rather than

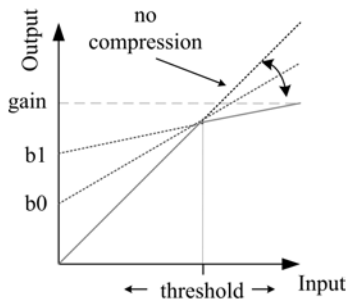


Fig. 3.3. The dynamic compressor.

a static one. Thus, designing and computing the transfer function to get a target sound and timbre is central in waveshaping synthesis.

3.1 *Dynamic compressor*

The *dynamic compressor* is used often in situations for sounds with wide dynamic range such as the electric bass guitar or a drum kit consisting of the kick drum, snare, tom-toms, and so on. What the compressor does is dynamically reduce the amplitude of the input signal and hence effectively reduces the dynamic range of a signal in a nonlinear fashion. Figure 3.3 depicts a very simple compressor showing the basic mechanics behind the concept.

The main effect of the compressor is that it reduces the amplitude of the input signal selectively for a given range — concentrating on the higher amplitude values of the input as can be clearly seen in Fig. 3.3. The result is that the difference between the "high" vs. "low" amplitude values are reduced, decreasing the dynamic range from a broader range to a narrower one. The core idea lies in reducing the overall dynamic range by attenuating the "high" valued amplitude samples while keeping the "low" valued samples untouched. Broad dynamic range refers to the aspect of an instrument being able to produce very soft sounds as well as very loud sounds — the wider the gap between the softest and loudest sound, the wider the dynamic range. The drum kit and the "slap and pop" technique in electric bass playing are great examples of instruments falling into the category of wide dynamic range-based instrument sounds. The amount of compression is mainly determined by two parameters — the threshold and the waveshaper's transfer function. As discussed earlier, the transfer function is simply the shape of the waveshaper that determines the input/output relationship as shown in Fig. 3.3. The amplitude threshold value moves along the input axis (x axis in Fig. 3.3) determining at what input level compression will be applied to. The closer the threshold value is to zero, the more compression there will be. The second parameter in our example is the *affine linear* function $y = mx + b$. As we can see in Fig. 3.3, in the region from 0 to the threshold value, the input (x) is equal to the output (y), whereas beyond the threshold value, the waveshaper takes the form of $y = mx + b$. The linear function becomes linear when $b = 0$ (no compression, $y = x$). The level of compression is controlled by the steepness of the slope (m): the less steep it is, the more compression and vice-versa. Our example is obviously a very rough one and commercial compressors have smooth transition regions (referred as *soft knees*) at the threshold points unlike our basic example shown here. However, the general effects are similar. Below I have listed some MATLAB® code for the main part of the dynamic compressor as discussed above.

```
theSign = sign(x); % retain sign of waveform
x = abs(x); % get absolute value of x
for i=1:length(x)
    if x(i) > threshold
        % compress
        y(i) = (slope*x(i) + intercept)*theSign(i);
    else
        % do not compress
        y(i) = x(i)*theSign(i);
    end
end
```

Code Example. 3.1. Simple dynamic compressor.

In the above code, I first store the sign (polarity) of the input signal and then compute the absolute value of the input which will be used to tell us when we need to compress the input signal. The sign is stored in memory for the synthesis part where the signal $|x|$ is multiplied by its original polarity.

Two more important parameters used to further fine tune and tweak compressors are the compressor *attack time* and *release time*. The attack time and release time simply refer to how quickly (attack) the compressor will start to take effect and how quickly (release) the compressor will stop working — going back to the region below the threshold. Hence, both attack

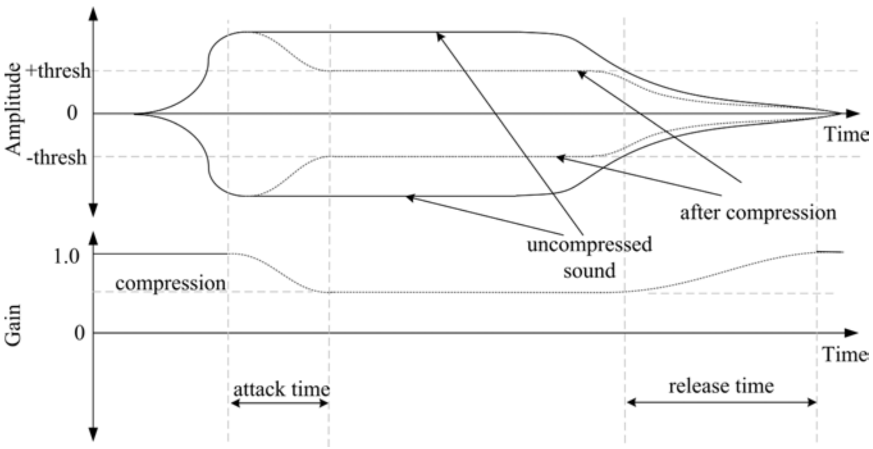


Fig. 3.4. Attack time and release time in compressors.

and release time parameters are essentially controlling the transition time from the uncompressed sound to the compressed sound and vice-versa. These are important issues as without these two parameters, the transition from uncompressed (unprocessed) to compressed output (processed) can be (depending on the input signal type) too sudden, causing subtle, yet often noticeable and undesirable artifacts. Figure 3.4 depicts the idea behind the attack time, release time, and the delay that occurs when the compressor starts to kick in during the attack region and diminish during the release region.

A variation of compression is the *limiter*. The limiter is also a popular signal processing example used in audio applications. By hard-limiting a signal, one can effectively guarantee that the output (through speakers for example) does not exceed a set amplitude level. It is therefore frequently used in live-performances and concert settings whenever microphones are involved where audio feedback can potentially become a major problem.

3.2 *Distortion*

In Chap. 1, we already introduced distortion in the form of clipping rendering in extreme cases a square wave. Since we now understand the workings of the compressor, we can also interpret distortion as a radical dynamic compressor — the $y = mx + b$ becomes a straight line with $m = 0$. The main difference between distortion and compression is mostly in its application. For the distortion effect, the severe clipping characteristic brought about by a high gain input produces additional harmonics, which in turn creates an *intended* richer sound. For the compressor, such clipping characteristics would generally not be attractive as the objective is reducing the dynamic range without (ideally) affecting the general timbre of the output sound. But for distortion, the same artifacts become a desirable musical element for timbral modulation.

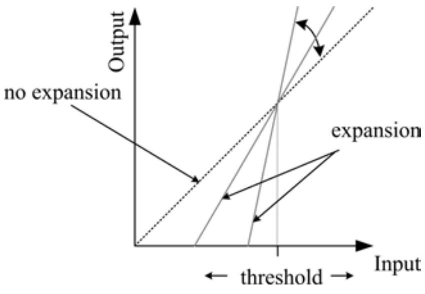


Fig. 3.5. The dynamic expander.

3.3 *Dynamic expander*

The *expander* is the opposite of the compressor. Instead of reducing the gain of the high amplitude level samples, it attenuates the lower ones in effect increasing the dynamic range. As is the case with the compressor, the expander can also be regarded as a special type of a waveshaper. The characteristics of the expander are show in Fig. 3.5.

You will note that similar to the compressor, the slope of $y = mx + b$ and threshold values define the amount of expansion. When $y = x$ there is no expansion due to unity gain. Interestingly, the classic *noise gate* which reduces the noisiness of a signal actually exploits the same concept utilized in the dynamic expander. The difference between the expander and noise gate is that for the noise gate, the input values below the threshold are floored to 0 thus only outputing input values that are above the threshold point. This type of noise gate "reduces" the noise as it will only output a non-zero value when there is a "strong" or "loud" enough signal. Although the effect may be perceived as noise reduction, in actuality there is really nothing exciting happening to the noise itself — when we have a high enough input signal above the noise gate threshold value (typically set to a very low value), the noise will still remain but (hopefully) be masked out by the stronger signal such as the entrance of a snare drum (high SNR ratio). The gate is nevertheless very useful when recording in the studio or in live performance situations where microphones are involved. For example, if the lead vocalist is not singing into the microphone and running around on the stage, there is a danger of amplifying other sounds picked up via the unattended microphone if a gate is not used. However, if we use a gate, the amplifier will only output the microphone signal when in use — when the lead singer is screaming into the microphone.

3.4 *Tape saturation*

In the old days when only magnetic tapes and tape machines were available, recording studios had no choice other than using those analog-based sound recording technologies — technologies which often exhibited nonlinear distortion due to the inherent imperfections of magnetic tape and tape machine technology. One type of distortion that occurs with analog tape machines is referred to as *tape saturation.* Simply put, tape saturation is a smooth and soft compression-type artifact that goes hand-in-hand with analog tape machines which has found much popularity with producers and musicians in the rock/popular music genre. One way of digitally simulating this "error" produced by tape machines is achieved via a smooth and delicate dynamic compression on the audio signal. As we have seen in our previous examples and in Chap. 1, harmonic distortion occurs when signals are clipped. Hence, when compression is applied to any input signal, it will produce additional subtle harmonic artifacts that sometimes are desirable in certain musical situations. Albeit, it is difficult, to say the least, to get rid of these tape machine-based artifacts when using analog tape recording technology, digital systems allows us the flexibility to simulate such irregularities with relative ease and furthermore gives us the option of not using it at all if we do not feel it contributes to the musical result we seek. Analog tape machines in general do not give us this option.

3.5 *Waveshaping synthesis*

The concept of *waveshaping synthesis* is essentially identical to dynamic compression with the main difference lying in the choice/design of the transfer function and the input signal to the waveshaper. The goal in waveshaping synthesis is choosing a transfer function and input signal so that multiple harmonics can be flexibly and accurately controlled and generated. We will discuss how this is accomplished in this section.

3.5.1 *Chebychev polynomials of the 1st kind*

One classic approach that is often encountered in designing the transfer function is the *Chebychev polynomials of the 1st kind* which should not be confused with *Close Encounters of the 3rd Kind.* Polynomials, as we are aware, take the generic form as shown in Eq. (3.1).

$$f(x) = a_N \cdot x^{N-1} + a_{N-1} \cdot x^{N-2} + \cdots + a_1 \cdot x + a_0 \tag{3.1}$$

With the Chebychev polynomials in particular, it is possible to produce specific harmonics if the input to the waveshaper is a cosine waveform. The advantage of using polynomials lies largely in the ability to produce a specific number of harmonics and in the computational efficiency due to the use of a single oscillator. The technique of using Chebychev polynomials to produce specific harmonic structures is also referred to as *spectral matching*, as it has been widely used to match and mimic the harmonic structure of acoustic instruments. The harmonics of any fundamental frequency can be expressed via polynomial expansion (when input x is a cosine wave) shown in Eq. (3.2), where T is the harmonic component and integer k represents the harmonic number, with $k = 1$ denoting the fundamental, $k = 2$ the octave, and T_{k+1} harmonic k+1 and so on.

$$T_{k+1}(x) = 2 \cdot x \cdot T_k(x) - T_{k-1}(x) \tag{3.2}$$

In order for the above relationship to work, the input x has to be limited to a cosine wave. When using the Chebychev polynomial (1st order), it is possible to add harmonics to the base cosine wave (input to the system) with a given fundamental frequency without having additional oscillators, thus making this approach an efficient synthesis method to compose complex sounds via a single sinusoid (making a bunch of sinusoids with one sinusoid). To understand how polynomials can be used to produce specific harmonics, let's start with some basic trigonometric identities and work our way up to the Chebychev polynomials of the 1st kind. We first begin with the following basic trigonometric identities:

$$\cos(a + b) = \cos(a) \cdot \cos(b) - \sin(a) \cdot \sin(b) \tag{3.3}$$

$$\sin(a + b) = \sin(a) \cdot \cos(b) + \cos(a) \cdot \sin(b) \tag{3.4}$$

Using the basic identities from above we can now deduce the following relationships:

$$\begin{aligned}\cos(2\cdot\theta) &= \cos(\theta+\theta)\\ &= \cos(\theta)\cdot\cos(\theta)-\sin(\theta)\cdot\sin(\theta)\\ &= \cos^2(\theta)-\sin^2(\theta)\\ &= \cos^2(\theta)-\left\{1-\cos^2(\theta)\right\}\\ &= 2\cdot\cos^2(\theta)-1\end{aligned} \tag{3.5}$$

$$\begin{aligned}\cos(3\cdot\theta) &= \cos(\theta)\cdot\cos(2\cdot\theta)-\sin(\theta)\cdot\sin(2\cdot\theta)\\ &= \cos(\theta)\cdot\left\{2\cdot\cos^2(\theta)-1\right\}-\sin(\theta)\cdot\{2\cdot\sin(\theta)\cdot\cos(\theta)\}\\ &= 2\cdot\cos^3(\theta)-\cos(\theta)-2\cdot\sin^2(\theta)\cdot\cos(\theta)\\ &= 2\cdot\cos^3(\theta)-\cos(\theta)-2\cdot\left\{1-\cos^2(\theta)\right\}\cdot\cos(\theta)\\ &= 2\cdot\cos^3(\theta)-\cos(\theta)-2\cdot\cos(\theta)+2\cdot\cos^3(\theta)\\ &= 4\cdot\cos^3(\theta)-3\cdot\cos(\theta)\end{aligned} \tag{3.6}$$

Following the method we used to compute $\cos(2\cdot\theta)$ and $\cos(3\cdot\theta)$ along with the trigonometric identities, we can also compute subsequent cosines of the form $\cos(m\cdot\theta)$ where m is an integer. The first five are listed below:

$$\cos(0\cdot\theta) = 1 \tag{3.7}$$

$$\cos(1\cdot\theta) = \cos(\theta) \tag{3.8}$$

$$\cos(2\cdot\theta) = 2\cdot\cos^2(\theta)-1 \tag{3.9}$$

$$\cos(3\cdot\theta) = 4\cdot\cos^3(\theta)-3\cdot\cos(\theta) \tag{3.10}$$

$$\cos(4\cdot\theta) = 8\cdot\cos^4(\theta)-8\cdot\cos^2(\theta)+1 \tag{3.11}$$

Let's stop here and make a variable assignment by letting $x = \cos(\theta)$ and define $T_k(x)$ such that we get Eq. (3.12) where k is an integer number.

$$T_k(x) = \cos(k\cdot\theta) \tag{3.12}$$

Rewriting Eq. (3.7) $\sim$ (3.11) using the newly defined $T_k(x)$s we have:

$$T_0(x) = \cos(0\cdot\theta) = 1 \tag{3.13}$$

$$T_1(x) = \cos(1\cdot\theta) = \cos(\theta) = x \tag{3.14}$$

$$T_2(x) = \cos(2\cdot\theta) = 2\cdot x^2-1 \tag{3.15}$$

$$T_3(x) = 4\cdot x^3-3\cdot x \tag{3.16}$$

$$T_4(x) = 8 \cdot x^4 - 8 \cdot x^2 + 1 \tag{3.17}$$

$$T_5(x) = 16 \cdot x^5 - 20 \cdot x^3 + 5 \cdot x \tag{3.18}$$

$$\text{etc.}$$

At this point we will observe a pattern emerging, enabling us to put the above polynomial expansion into a generic form as shown in (3.19). This is what we started out as in Eq. (3.2):

$$T_{k+1}(x) = 2 \cdot x \cdot T_k(x) - T_{k-1}(x) \tag{3.19}$$

The above is known as the Chebychev polynomial of the 1st kind. Why is this useful again? The result is very useful because when setting the input $x = \cos(\theta)$ as we have done above (with cosine amplitude set to 1), each Chebychev polynomial, which forms a particular waveshaper, will output only one specific harmonic. That is, when setting $x = \cos(\theta)$ the Chebychev polynomial $T_k(x) = \cos(k \cdot \theta)$ will produce a cosine signal at harmonic k without the need for additional oscillators which can otherwise be a burden on the computational load of a processor. Thus, if we were to combine a number of different $T_k(x)$ components, which could represent the harmonic structure of a trumpet for example, we will be able (via waveshaping synthesis) to produce a synthetic trumpet sound with numerous harmonics without using an equal number of oscillators. For example, the waveshaper could look something like Eq. (3.20) resulting in an approximation of a square wave sound.

$$F(x) = T_1(x) + \frac{1}{3}T_3(x) + \frac{1}{5}T_5(x) + \frac{1}{7}T_7(x) + \frac{1}{9}T_9(x) + \frac{1}{11}T_{11}(x) \tag{3.20}$$

As mentioned before, this particular method of determining what waveshaper to use is called spectral matching — matching the harmonic structure of existing acoustic instruments or any other sound for that matter. One of the main reasons this was popular back in the day is not only due to the computationally expensive oscillators, but also because it rendered interesting timbral results when *not* using cosine waves as input, thus making it a very powerful sound synthesis tool.

Since we are dealing with digitally sampled signals, note that we also have to be careful about the upper Chebychev polynomial signals which need frequency restrictions as we do not have infinite bandwidth. In other words, we have to make sure that the resulting harmonics (dictated by k), which are dependent on the fundamental frequency f_0 of the cosine wave, are below the Nyquist limit:

$$k \cdot f_0 < \frac{fs}{2} \tag{3.21}$$

For complex signals that are made up of many more cosine components of various frequencies (which we do not know beforehand), we should band-limit the input signal before subjecting it to a particular Chebychev polynomial $T_k(x)$. This band-limiting technique is referred to low-pass filtering covered in Chap. 7. The good news is that when making music there is no wrong way of doing things as long as it positively serves the musical composition in the end. If the musical result is satisfactory to the composer or sound designer, then it really does not matter if we have artifacts like aliasing.

4 Some Familiar Time-Domain DSP Effects

In this section, we will introduce a couple of DSP processes that we are probably familiar with or at least have heard of at some point in time.

4.1 *Equal power panning*

The term *panning* is ubiquitously used in audio but actually comes to us from the film-making folks referring to the rotation of the camera to get a panoramic view of a scene. In audio, panning originates from the word *pan pot* which in turn is a short-hand for *panoramic potentiometer*, the potentiometer being a variable resistor like the volume dial on your stereo or your light dimmer in the living room. Panning in 2-channel audio applications distributes an audio signal to two output channels and two speakers. Imagine turning a dial counterclockwise and clockwise to make a sound come out of the left and right speakers in their extreme positions and having the sound be generated equally through both speakers at the 12 o'clock position.

This concept can be implemented on a digital system. Let's say the maximum output of a signal is a floating point number at 1.0 and we wish to pan it to two channels via two speakers. A simple solution for panning the input $x[n]$ to the left and right output channels $y_{left}[n]$ and $y_{right}[n]$ is:

$$y[n]_{left} = x[n] \tag{4.1}$$

$$y[n]_{right} = 1.0 - x[n] \tag{4.2}$$

Although this will work quite nicely, there is one small issue that we have overlooked: the aspect of our hearing system and how we perceive the dynamic level of audio signals. You will recall from Chap. 1 that we hear sound in the form of power, which means that it is the square of the input amplitude. In other words, the above panning algorithm would actually not quite do the trick in equally and smoothly panning the signal from left to right (if we turn the knob at a constant angular velocity). In order to make the panning effect be *perceived* linearly we would need to use the inverse of the square operator which is the square root. Obviously the square root is not a linear function but the important point is that we *perceive* the operation imposed on the signal to be linear, as the square is neutralized by the square root rendering a linearly changing pan when turning the pan pot linearly. The modified perceptually linear panning equations will then become:

$$y[n]_{left} = \sqrt{x[n]} \tag{4.3}$$

and

$$y[n]_{right} = \sqrt{1.0 - x[n]} \tag{4.4}$$

You may also have heard about the $-3\,\text{dB}4$ number in panning. The $-3\,\text{dB}$ refers to the dB level at center position of the pan pot which is equal to $20 \cdot \log_{10}(0.707)$ or the 0.707 amplitude value obtained by the root operator above (the root of 0.5 is 0.707, where 0.5 is halfway between 0.0 and 1.0 denoting half of the amplitude in each channel). Figure 4.1 shows equal power panning in amplitude and dB units.

4.2 *Delays*

The last three digital effects that we will discuss in this chapter are all based on delaying an input signal by various degrees. By changing the amount of delay and mixing (adding) the delayed signal with the original signal, we can get interesting musical effects commonly used in everyday audio applications. We will formally introduce difference equations, delays, and fractional delays in Chap. 5 and 7.

4.2.1 *Echo, chorus, and flanging*

The first of the delay-based effects we will deal with is the so-called *echo* (there was actually a band called *Echo and the Bunnymen* back in

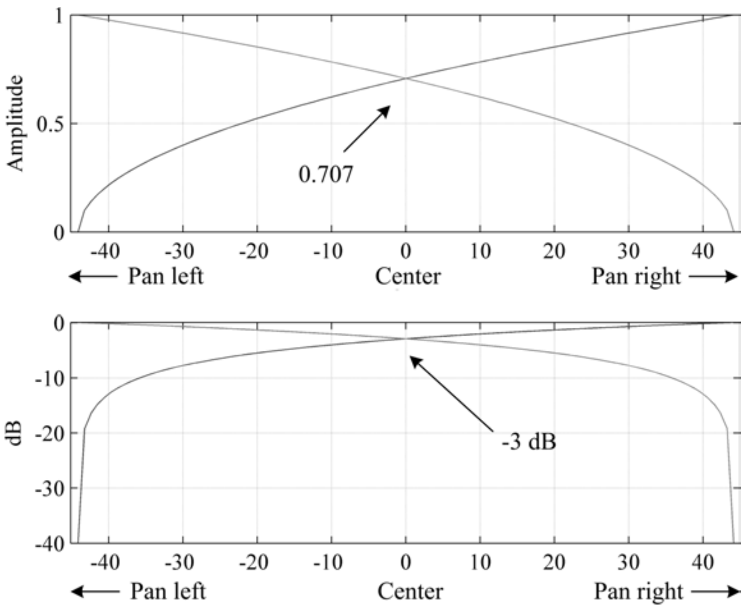


Fig. 4.1. Equal power panning with root of the amplitude (top) and dB of the root amplitude (bottom).

1980s which did not have much to do with the effect, however). We have undoubtedly experienced the echo effect in a large hall, cathedral or perhaps the Grand Canyon or Seol Ak-San in Korea, where a loud yahoo is bounced off the canyon walls at the speed of sound and comes back to us thus causing an echo effect. The echo can simply be represented as shown in Eq. (4.5).

$$y[n] = x[n] + b_N \cdot x[n - N] \tag{4.5}$$

N is the delay of the bounced signal coming back to us and is typically in the range of 10 to 50 milliseconds, although there is no hard limit as longer delays simply mean longer echoes. b_N is the coefficient or the scalar multiplier which attenuates the bouncing signal as they are softer due to loss of energy from all the bouncing around.

Delays that are typically shorter than that of the echo are commonly referred to as chorus delay. However, there is a difference between an echo and chorus and that is the dynamically changing delay nature of the chorus effect. The delay time commonly smoothly varies from around 10 to 25 milliseconds achieving a *doubling effect*. This doubling effect sounds as if two people (or audio signals) are singing together. With more delay lines added to the system, we would get the effect of having a large number of people singing at the same time (a chorus) — hence the name chorus effect. It so happens that when we have more than one person singing or playing a musical instrument with the same pitch, there will always be some amount of detuning or slight variation in the fundamental frequency. This is due to the imperfections in human performance, the musical instrument itself (violins, voices, French horns, etc.), and the distance between the instruments/sound sources (as well as where the listener is with respect to each sound source). Even if the instruments are "perfectly" tuned, there will always be a certain amount of detuning especially as the number of instruments grows. The digital chorus algorithm hence mimics this imperfection (which makes a sound thicker and more dense) by using a variable delay line:

$$y[n] = x[n] + x[n - g[n]] \tag{4.6}$$

$g[n]$ outputs an integer delay that varies slowly with time bounded by upper and lower delay time limits. For example, $g[n]$ could be a low frequency digital sine oscillator also referred to as a *low frequency oscillator* (LFO).

Back in the good ole days of analog tape recorders, engineers would sometimes have two audio tracks on two separate tape machines with the exact same recording or program as they are called in the recording world (analog tape recorders are still used in some recording studios although their usage has gone down drastically due to digital technology). What people would do is slightly change the speed of the reel-to-reels by applying hand pressure against the flanges (rim) and cause slight deviations in playback speed — the ensuing effect is referred to as *flanging.* Not surprisingly, the flanger is actually identical to Eq. (4.6) and hence is essentially the same as the chorus algorithm. The issue of human perception of sound makes the flanger different from the chorus and not its core algorithm, with the most significant difference being the amount of delay time. For the flanger, the delay time is typically below 10 milliseconds opposed to 10 to 25 milliseconds for chorus. With all the delay effects we have mentioned here, constructive and deconstructive interference (see Chap. 4, Sec. 4 for more details) obviously plays an important role and an interesting concept known as *comb-filtering* results which will be introduced in Chap. 7 when we talk about filters and reverberation.

5 Musical Examples

One of the first composers to use granular synthesis in musical composition is Iannis Xenakis. Xenakis used the concept of granular synthesis in making clouds of sound from crackling amber and burning coal recorded with a microphone in 1958. The piece called *Concrète PH* was part of architect Le Corbusier's multi-media extravaganza *Poème Electronique*, showcasing the latest research from electric/electronic company Philips at the 1958 World Fair in Brussels. The piece was, however, only used as an interlude between sessions (approximately three and a half minutes long) when spectators would enter and leave the Philips Pavilion where the multi-media show (featuring the main piece *Poème Electronique* by Edgard Varese) was held. It is quite an extraordinary piece in part due to the sheer technical difficulty in doing something like this in the late 1950s by using short samples as grains and working with tape splices in putting together hundreds and hundreds (probably many more) grains of sound. But what's more amazing is that Xenakis was thinking about the idea of the particle property of sound, concentrating on timbre, density, sound clouds, and sound itself, whereby incorporating those ideas in creating a new musical work. Twenty eight years later composer Barry Truax who is also well know for his compositions using granular synthesis, composed a piece called *Riverrun* (1986) — the first word in James Joyce's *Finnigan's Wake.* The metaphor of the piece is the river, which is characterized by stasis and flux, calmness and chaos, and everything in between. Technically speaking, Truax used grain densities typically between 100 and 2000 grains/second. What makes this piece such a great example of granular synthesis in action is the idea behind the basic water droplet taking the role of a grain forming the smallest rivulets, streams, and a massive river of immense energy guiding the listener on a wild ride from the perspective of a water droplet.

Other effects we have briefly discussed in this chapter including distortion and compression can be found in most popular music recordings. In fact, compression is an important part in the whole modern recording culture and also critical in the mastering process — massaging a recording before it hits the market. Distortion and compression is also used in a very artistic and creative way to bring about a certain musical character. One such artist who is no stranger in using these effects in order to bring about a specific sound color is Trent Reznor. Albums such as *With Teeth* is such an example, where harshness, distortion, and sometimes low-fi sound is exploited and used not only on instruments but also the voice.

References and Further Reading

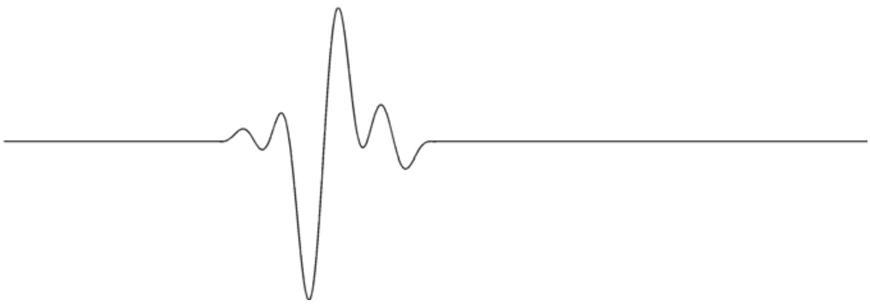
Cohen, H. F. 1984. Quantifying Music. The Science of Music at the First Stage of the Scientific Revolution, 1580–1650. Springer.

Gabor, D. 1946. “Theory of Communication”, The Journal of the Institution of Electrical Engineers, 93(3), 429–457.

Gabor, D. 1947. “Acoustical quanta and the theory of hearing”, Nature, 159(4044), 591–594.

Chapter 4

SINE WAVES



1 Introduction

We began the book with a brief introduction of the sine wave as a way to start getting our hands dirty and begin with something that a lot of us (hopefully) are probably familiar with in one way or another. In this chapter, we will revisit some of the concepts related to the sine wave along with the complex number system, which will lead us to the powerful *Euler formula*. We will later see the importance of this formula as a tool for dealing with trigonometric functions and harmonic oscillation, especially in Chap. 8 when we get to know Mr. Fourier. Towards the end of the chapter, we will learn one of the most simple, yet effective and economically successful sound synthesis algorithms, frequency modulation synthesis. We again conclude Chap. 4 with the introduction of some musical examples and composers who have used some of the concepts we cover here.

2 Sinusoids Revisited

At the beginning of Chap. 1, we started with the sinusoid and briefly discussed the three components that characterize it — amplitude, frequency, and initial phase as shown in Eq. (2.1).

$$y(t) = A \cdot \sin(2 \cdot \pi \cdot f \cdot t + \phi) \tag{2.1}$$

It is customary to denote the time varying part of Eq. (2.1) as ω — the radian frequency (rad/sec) where f is the frequency in Hertz.

$$\omega = 2 \cdot \pi \cdot f \tag{2.2}$$

The phase ϕ in the Eq. (2.1) describes the phase offset with respect to a reference point, and is more precisely referred to as the *initial phase* at reference point $t = 0$. In the above equation ϕ is a constant. To distinguish the initial phase from a time-variant version of the phase, the term *instantaneous phase* is used and can be expressed as shown in Eq. (2.3) [note that ϕ itself can be a function of time: $\phi(t)$].

$$\Theta(t) = 2 \cdot \pi \cdot f \cdot t + \phi = \omega \cdot t + \phi \tag{2.3}$$

This may look a little confusing as we have lumped the frequency component and initial phase component and called it instantaneous phase, but if you regard it in such as a way that frequency f in combination with ϕ merely determines a particular radian value (the instantaneous phase) at each time instant t, the above equation and expression should make sense [the output of $\Theta(t)$ is obviously in radians as the units of f and t cancel each other out]. That is, the measurement of following the total "radian distance" travelled by a sinusoid over a period starting at $t = 0$ to $t = M \cdot T = M \cdot 1/f_s$ is equivalent to the instantaneous phase. Viewing the instantaneous phase from a different perspective, via a sinusoid with frequency f which stays constant, the derivative of $\Theta(t)$ changes according to Eq. (2.4).

$$\frac{d\Theta(t)}{dt} = 2 \cdot \pi \cdot f \tag{2.4}$$

The instantaneous phase can be also be viewed as an integration of the *instantaneous frequency* $f(\tau)$ from $\tau = 0$ to t, as integration and differentiation have reciprocal relationships.

$$\Theta(t) = \int_0^t 2 \cdot \pi \cdot f(\tau) d\tau \tag{2.5}$$

Like instantaneous phase, instantaneous frequency $f(\tau)$ is a time-variant version of the frequency component, distinguishing it from the constant and unchanging frequency f. The instantaneous frequency is represented according to Eq. (2.6).

$$f(t) = \frac{1}{2 \cdot \pi} \frac{d\Theta(t)}{dt} \tag{2.6}$$

In a digital system the instantaneous frequency is commonly computed via the change in phase over a given period of time (samples). We will see how these important definitions become helpful in understanding frequency modulation synthesis later in this chapter as well as the phase vocoder in Chap. 9.

3 Imaginary, Complex Numbers, and Euler's Formula

Imaginary numbers are as the term suggests, numbers that are not real — numbers that cannot be represented by real numbers, those numbers mapped on a one-dimensional system. I say one-dimensional, as real numbers can only increase or decrease in value on a linear slider-like scale. For example, if we consider 0 to be the center of the real number system, we would be able to go right and find larger real valued numbers, or move to the left, towards the negative side and equivalently find negatively decreasing real numbers. This sort of system is one-dimensional as it only requires the x-axis so-to-speak. Although complex numbers may seem counterproductive and may even be regarded initially as having been created only to make our lives more miserable, imaginary numbers are, however, very powerful and useful. As a matter of fact, the complex number system is essential in enabling us to tackle certain problems in often remarkably simple ways which would otherwise be extremely difficult.

Consider the general 2nd order polynomial shown in Eq. (3.1) and its two possible roots seen in Eq. (3.2):

$$a \cdot x^2 + b \cdot x + c = 0 \tag{3.1}$$

$$x = \frac{-b \pm \sqrt{b^2 - 4 \cdot a \cdot c}}{2 \cdot a} \tag{3.2}$$

Now, all this is great until we get a solution for the polynomial where the square root part $b^2 - 4 \cdot a \cdot c < 0$. For example, if we set $a = b = c = 1$ we would get (3.3) which is pretty funky.

$$x = \frac{-1 \pm \sqrt{-3}}{2} \tag{3.3}$$

We basically hit a dead end as we cannot define negative square root terms using a real number system — this is where the imaginary number system comes to our rescue. If we define an imaginary number to be expressed as in Eq. (3.4), we can deal with anomalies such as negative square roots that would otherwise not be possible to address with the real number system alone: we denote the square root of negative 1 as "i."

$$i = \sqrt{-1} \tag{3.4}$$

Hence, with the aid of the imaginary number system, the solution to the above polynomial will result as follows:

$$x = \frac{-1 \pm \sqrt{1-4}}{2} = -\frac{1}{2} \pm i\frac{\sqrt{-3}}{2} = -0.5 \pm i\frac{3}{2} - 0.5 \pm i1.5 \tag{3.5}$$

The general form $a + ib$ is referred to as a complex number system where a is the real part and b the imaginary part. Imaginary numbers are also often denoted with the letter "j" instead of the "i" in the form of $a + jb$, especially in engineering books to avoid potential confusion: letter "i" in electrical engineering is usually a parameter reserved for electrical current. We will use j for the imaginary numbers henceforth in this book.

If we view real numbers to be governed by a one-dimensional system as shown in Fig. 3.1 (for simplicity integers are used here) we can view complex numbers as a two-dimensional system as seen in Fig. 3.2. As complex numbers can be represented in a two-dimensional plane we can use trigonometric representations to jump from rectangular to polar coordinate systems as depicted in Fig. 3.3.

The rectangular form is the notation we used above in the form of $c = a + jb$. The polar form can be expressed using angles and Euclidian

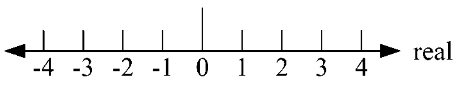


Fig. 3.1. A one-dimensional system: real numbers.

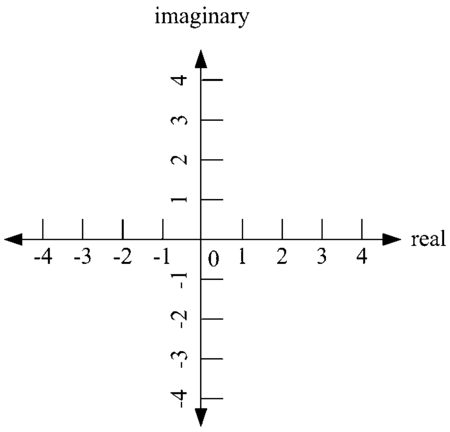


Fig. 3.2. A two-dimensional system: imaginary numbers.

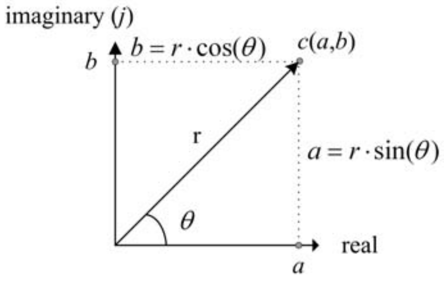


Fig. 3.3. Rectangular and polar forms.

distances to represent the same point c.

$$b = r \cdot \cos(\theta) \tag{3.6}$$

$$a = r \cdot \sin(\theta) \tag{3.7}$$

$$r = \sqrt{a^2 + b^2} \tag{3.8}$$

$$\theta = \arctan\left(\frac{b}{a}\right) \tag{3.9}$$

3.1 *Euler's formula*

Euler's formula named after Swiss mathematician Leonhard Euler, is by many mathematicians (and non-mathematicians for that matter!) regarded as one of the most elegant and beautiful mathematical theorems. Some of the reasons for this consensus can be attributed to the fact that it can be used to describe simple harmonic motion, integer numbers, complex numbers, real numbers, etc. For us, it will be a very powerful tool that will help us greatly in dealing with difficult problems that use aspects of harmonic motion as we shall see later in this chapter and other chapters. The Euler's formula is defined as follows:

$$e^{j\theta} = \cos\theta + j\sin\theta \tag{3.10}$$

We will immediately note that using the Euler formula, we can now put $c = a + jb$ into a more compact form as shown in Eq. (3.11) with amplitude $r = 1$.

$$c = r \cdot e^{j\theta} \tag{3.11}$$

Two very useful identities for sines and cosines in terms of the above exponential expressions are as shown in Eqs. (3.12) and (3.13).

$$\cos\theta = \frac{e^{j\theta} + e^{-j\theta}}{2} \tag{3.12}$$

$$\sin\theta = \frac{e^{j\theta} - e^{-j\theta}}{2j} \tag{3.13}$$

The above can be verified by plugging in Euler's formula for each exponential component $e^{+j\theta}$ and $e^{-j\theta}$. It may, however, still be unclear why Euler's formula is so important in signal processing, but if we read on, we shall soon see in the next section how practical and helpful it actually can be.

4 Sinusoidal Modulation Techniques I: Amplitude

In Secs. 4 and 5 we will discuss a number of classic sinusoidal modulation techniques. We will see how simple alteration of amplitude and frequency components of sinusoids can result in very musically rich material. We start by focusing on the amplitude component of sinusoids.

4.1 *Beating*

In Chap. 1 we briefly introduced the notion of constructive and deconstructive interference where two or more waveforms such as cosine waves reinforce, weaken, and in extreme cases totally cancel each other out. In this section, we will delve a little deeper into this subject which will segue to the concept of *beating*. When two sinusoids with two different frequencies are summed the following relation can be observed:

$$A \cdot \cos f_1 + A \cdot \cos f_2 = 2 \cdot A \cos\left(\frac{f_1 - f_2}{2}\right) \cdot \cos\left(\frac{f_1 + f_2}{2}\right) \tag{4.1}$$

From Eq. (4.1) we can see that summing two cosines is equivalent to multiplying two cosines with the sum and difference of the frequencies. As a matter of fact, Eq. (4.1) is the reciprocal of *ring modulation* discussed shortly in Sec. 4.2. If we let the difference between the two frequencies $f_1 - f_2$ be small, say between 1 to 10 Hz or so, a perceptual phenomenon referred to as *beating* is perceived. Figure 4.1 shows two cosine waves at 400 Hz and 402 Hz with a sampling rate of 8 kHz.

In this example, we will hear the average frequency 401 Hz $(f_1 + f_2)/2$ being modulated with a low frequency of 2 Hz $(|f_1 - f_2|)$. Interestingly enough, although (4.1) shows that we should actually hear a beating frequency of $(f_1 - f_2)/2$, which is equivalent to 1 Hz, this is not what

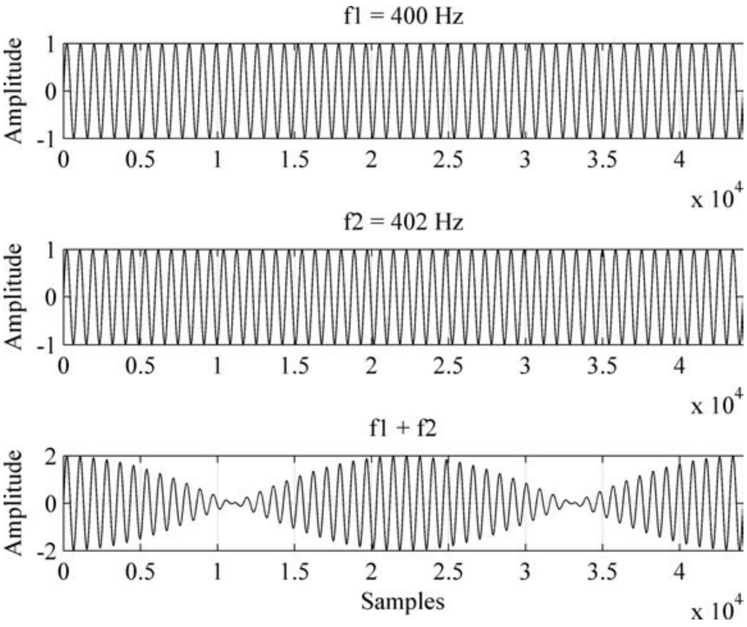


Fig. 4.1. Beating: cosine wave at $f_1 = 400$ Hz and $f_2 = 402$ Hz.

we really hear. The reason for this again has to do with perception and our hearing system. At low frequencies such as 1 Hz, we will in reality hear a beating frequency twice the modulation frequency, that is:

$$f_{beating} = 2 \cdot \frac{f_1 - f_2}{2} = f_1 - f_2 \tag{4.2}$$

The reason we perceive it as twice the modulation frequency is because the low frequency cosine modulator ($f_1 - f_2$ component) zero-crosses twice during one full cycle as we can clearly see in Fig. 4.2. With every zero-crossing, the amplitude of the resulting signal of Eq. (4.1) will be 0 — this occurs twice during one period and creates a sense of beating.

We could at this point forget about proving Eq. (4.1), as using trigonometric identities and such might seem a bit tedious. However, if we use Euler's formula we will see how straightforwardly we can prove that the right-hand side is equal to the left-hand size in (4.1). We'll start by rewriting the right-side of Eq. (4.1) in terms of $e^{j\theta}$ by defining θ_1 and θ_2 as

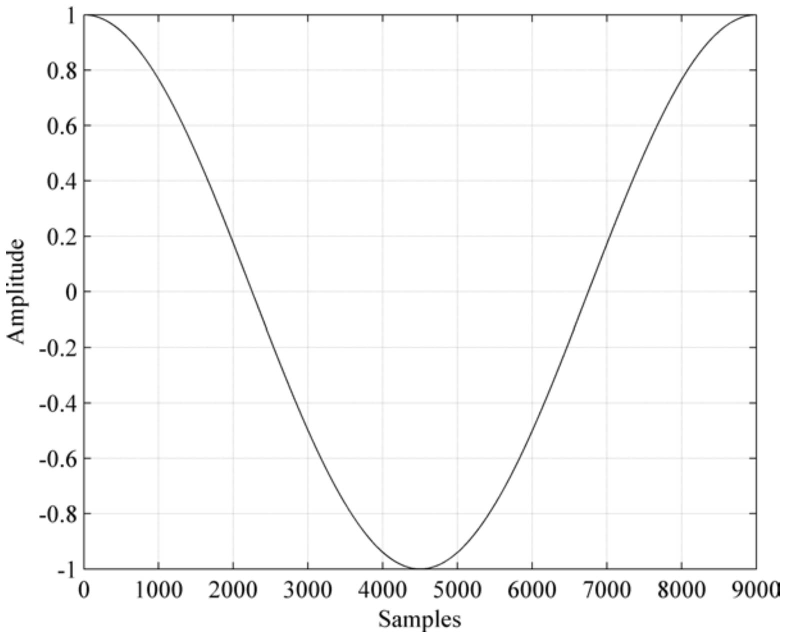


Fig. 4.2. One complete cosine resolution.

follows:

$$\theta_1 = \frac{f_1 - f_2}{2} \tag{4.3}$$

$$\theta_2 = \frac{f_1 + f_2}{2} \tag{4.4}$$

We then plug in θ_1 and θ_2 into (4.1) and use the Euler identify for the cosine and get:

$$\begin{aligned}
2 \cdot A \cdot \cos\left(\frac{f_1 - f_2}{2}\right) \cdot \cos\left(\frac{f_1 + f_2}{2}\right) &= 2 \cdot A \cdot \cos\theta_1 \cdot \cos\theta_2 \\
&= 2 \cdot A \cdot \left\{\left(\frac{e^{j\theta_1} + e^{-j\theta_1}}{2}\right) \cdot \left(\frac{e^{j\theta_2} + e^{-j\theta_2}}{2}\right)\right\} \\
&= \frac{A}{2} \cdot \left\{\left(e^{j\theta_1} + e^{-j\theta_1}\right) \cdot \left(e^{j\theta_2} + e^{-j\theta_2}\right)\right\} \\
&= \frac{A}{2} \cdot \left\{e^{j(\theta_1+\theta_2)} + e^{j(\theta_1-\theta_2)} + e^{j(\theta_2-\theta_1)} + e^{-j(\theta_1+\theta_2)}\right\} \qquad (4.5)
\end{aligned}$$

Now rewriting θ_1 and θ_2 in terms of f_1 and f_2 in (4.5) we get:

$$\begin{aligned}
&\frac{A}{2} \cdot \left\{e^{j(\theta_1+\theta_2)} + e^{j(\theta_1-\theta_2)} + e^{j(\theta_2-\theta_1)} + e^{-j(\theta_1+\theta_2)}\right\} \\
&\quad = \frac{A}{2} \cdot \left\{e^{jf_1} + e^{-jf_2} + e^{jf_2} + e^{-jf_1}\right\} \\
&\quad = \frac{A}{2} \cdot \left\{\left(e^{jf_1} + e^{-jf_1}\right) + \left(e^{jf_2} + e^{-jf_2}\right)\right\} \\
&\quad = A \cdot \left\{\frac{\left(e^{jf_1} + e^{-jf_1}\right)}{2} + \frac{\left(e^{jf_2} + e^{-jf_2}\right)}{2}\right\} \\
&\quad = A \cdot \cos f_1 + A \cdot \cos f_2 \qquad (4.6)
\end{aligned}$$

Note in the penultimate step we used the cosine identity from Eq. (3.12) and the result is the expected the sum of the two cosines from Eq. (4.1).

4.2 *Amplitude modulation and ring modulation*

Amplitude modulation (AM) and *ring modulation* synthesis techniques for audio are two classic sound synthesis examples. As we shall see, AM and ring modulation synthesis are only slightly different from each other.

Furthermore, they are also related to beating and are remarkably simple algorithms that have the potential of producing interesting timbral results for musical, sound design, and sound effects applications.

4.3 *Amplitude modulation (AM)*

The term modulation often refers to the altering of the frequency, phase, or amplitude component of an oscillator. In the case of amplitude modulation, the focus is on altering the amplitude components of oscillators involved — the simplest configuration entails two oscillators, namely the *carrier* and *modulator*. The carrier refers to the oscillator that is being altered and the modulator refers to the oscillator that changes the carrier. The classic definition of AM synthesis as it is used in computer music (i.e. in the audible frequency range), is defined as shown in Eq. (4.7) where f_c and A_c are the carrier frequency and amplitude components, and f_m and A_m the modulator frequency and amplitude components respectively.

$$y = \cos(f_c) \cdot \{A_m \cdot \cos(f_m) + A_c\} \tag{4.7}$$

$$\begin{aligned} y &= \cos(f_c) \cdot \{A_m \cos(f_m) + A_c\} \\ &= A_m \cdot \cos(f_c) \cdot \cos(f_m) + A_c \cdot \cos(f_c) \end{aligned} \tag{4.8}$$

Note that Eq. (4.7) can be expanded to Eq. (4.8) which represents the amplitude modulated signal in two parts – the modulated part and the original part. In other words, the original carrier oscillator is mixed to the resulting modulated part via parameter A_c. If we look at Eq. (4.8) more closely, we will note that it can be rewritten and expanded to (4.9) due to the sum and difference relationship found when multiplying sinusoids as we have already seen in beating and proven in (4.6).

$$\begin{aligned} \cos(f_c) \cdot \{A_m \cos(f_m) + A_c\} &= A_m \cdot \cos(f_c) \cdot \cos(f_m) + A_c \cdot \cos(f_c) \\ &= \frac{A_m}{2}\cos(f_c + f_m) + \frac{A_m}{2}\cos(f_c - f_m) + A_c \cdot \cos(f_c) \end{aligned} \tag{4.9}$$

The result is that the right-hand side includes the original carrier component and two new components with altered frequency and amplitude characteristics. This is the classic AM synthesis algorithm which has been used in a variety of audio applications including making “robotic voices” in science fiction movies early on in the motion picture industry. The discrete version of Eq. (4.7) is shown in Eq. (4.10) where n is the sample time index and f_s the sampling frequency.

$$y[n] = \cos\left(2 \cdot \pi \cdot f_c \cdot \frac{n}{f_s}\right) \cdot \left\{A_m \cos\left(2 \cdot \pi \cdot f_m \cdot \frac{n}{f_s}\right) + A_c\right\} \qquad (4.10)$$

Figure 4.3 shows the sum and difference phenomenon in amplitude modulation and Fig. 4.4 depicts the carrier oscillator in the top plot,

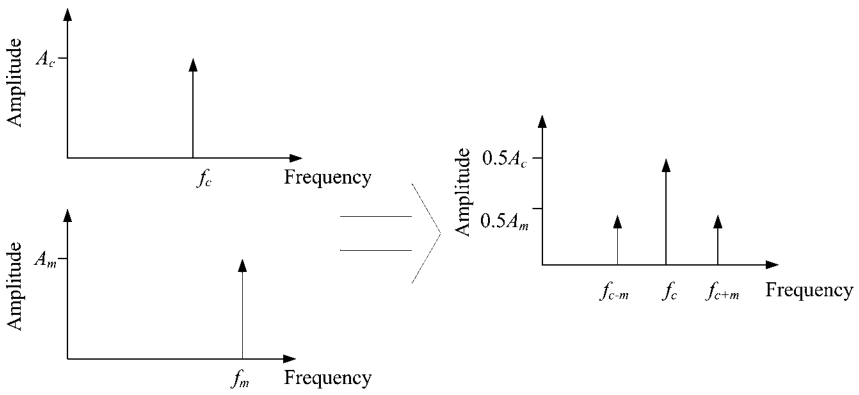


Fig. 4.3. Sum and difference and carrier component in output of AM synthesis.

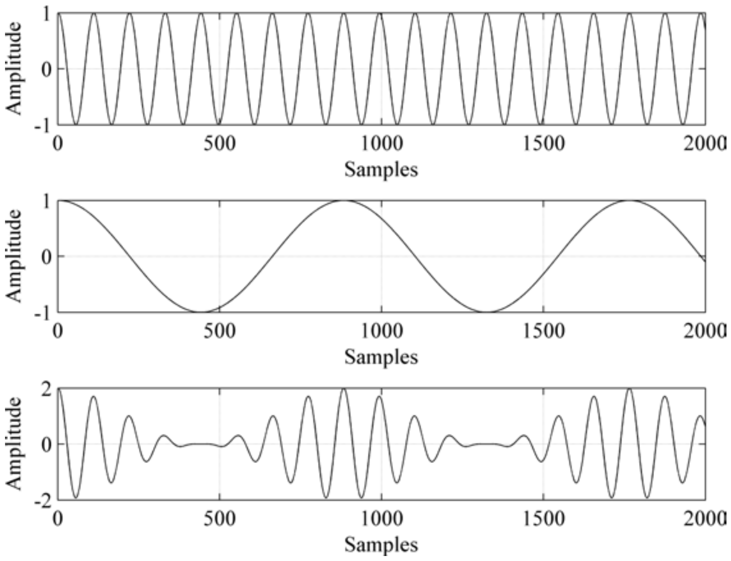


Fig. 4.4. AM synthesis $f_c = 400$ Hz, $f_m = 50$ Hz, $A_c = A_m = 1$, $f_s = 44.1$ kHz with carrier (top), modulator (middle), and synthesized output (bottom).

modulator in the middle plot, and resulting modulated signal in the bottom plot. When AM modulation frequency is in a low range below 10 Hz or so, it is usually referred to as *tremolo* which is a musical term. In this range, as it is below the limit of pitch perception, it is actually more perceived as change in amplitude as we have seen in beating, rather than change in timbre.

4.4 *Ring modulation*

Now that we know how AM works, it becomes a cakewalk in defining ring modulation:

$$y = A_m \cdot \cos(f_m) \cdot \cos(f_c) \tag{4.11}$$

The difference between AM synthesis and ring modulation is that in the latter we exclude the carrier component when multiplying the two oscillators. Hence, the output consists of only the sum and difference parts as we can see in Eq. (4.12).

$$y = A_m \cdot \cos(f_c) \cdot \cos(f_m) = A_m \cdot \frac{\cos(f_c + f_m) + \cos(f_c - f_m)}{2} \tag{4.12}$$

The discrete version is now simply as shown in Eq. (4.13) with n and f_s being the sample index and sampling frequency respectively as usual.

$$y[n] = A_m \cos\left(2 \cdot \pi \cdot f_m \cdot \frac{n}{f_s}\right) \cdot \cos\left(2 \cdot \pi \cdot f_c \cdot \frac{n}{f_s}\right) \tag{4.13}$$

Figure 4.5 shows how the carrier component does not appear in the output signal when two oscillators are modulated and (as expected) only includes the sum and difference components.

4.4.1 *Ring modulation with complex signals*

What becomes interesting in ring modulation as well as amplitude modulation synthesis is that we can also use non-single oscillator-based modulators and/or carriers, thus making the output much more complex and potentially more interesting. For example, the synthesis output may be the result of signals such as your favorite song modulated with an orchestral sample. Audio signals such as these are complex in nature and can be represented by a large number of finely tuned oscillators. We will see in Chap. 8 why this is the case, when we will become familiar with the Fourier transform. For the time being, let's loosely assume that the fundamental

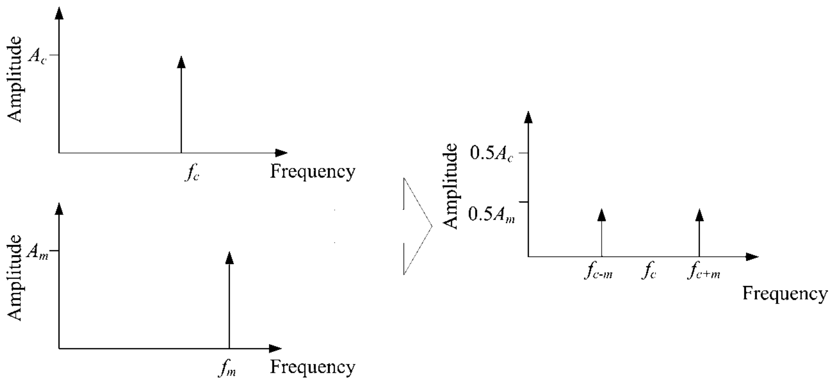


Fig. 4.5. Ring modulation: absence of carrier in output.

building blocks of sound are sinusoids and that complex sounds (especially musical sounds that are periodic) can be represented by a very large number of specific sinusoidal components. That is, if we had a large collection of sinusoids, say 2,048, with specific amplitude, frequency, and phase values for each, we would be able to represent a complex audio waveform. The large collection of cosine waves when summed over a given period of time will result in constructive and deconstructive interference, which will in turn shape the ensuing output waveform. This output will very closely resemble the original complex waveform. Obviously the trick is obtaining and computing the specific amplitudes, frequencies, and phases for each sinusoid and this will be discussed in Chap. 8.

For now, assuming we believe complex signals such as music can be decomposed into sinusoidal components, we can further assume that when multiplying two complex waves, the resulting effect will be equivalent to multiplying a large number of cosine waves (carriers and the modulators), where each multiplication of each cosine/sine wave (each having their own amplitude, frequency, and phase characteristics) will produce a sum and difference computation block according to Eq. (4.12). Thus, ring modulation for generic audio signals can represented via Eq. (4.14).

$$y[n] = a_m \cdot x_m[n] \cdot a_c \cdot x_c[n] \tag{4.14}$$

In this case, we have two generic audio signals, one being the modulator and the other the carrier with respective gain factors. Due to the commutative law of multiplication, the order of the multiplication actually becomes irrelevant and the notion of the carrier vs. the modulator on a mathematical level becomes insignificant as well.

5 Sinusoidal Modulation Techniques II: Frequency

In amplitude modulation synthesis, we multiplied two cosines resulting in the sum and difference of the carrier and modulation frequency components. Unlike AM synthesis, however, in *frequency modulation synthesis* or FM synthesis for short (Chowning 1973), we modulate the frequency component itself instead of the amplitude components. In a nutshell, FM synthesis can be elegantly represented according to Eq. (5.1).

$$y(t) = A_{carrier} \cdot \sin\{2 \cdot \pi \cdot f_{carrier} \cdot t + A_{mod} \cdot \sin(2 \cdot \pi \cdot f_{mod} \cdot t)\} \tag{5.1}$$

As in AM synthesis, FM synthesis includes the carrier's amplitude and frequency components ($A_{carrier}$ and $f_{carrier}$) as well as the modulator's amplitude and frequency components (A_{mod} and f_{mod}). A_{mod} is commonly referred to as the *modulation index*, determining the amount of modulation with $A_{mod} = 0$ resulting in no modulation, as can be verified in Eq. (5.1) — the ensuing output would simply be the carrier signal. Notice that Eq. (5.1) is actually not as complex as it looks, as we can break it up into two sinusoidal components as shown below where $g(t)$ is the modulating sinusoid.

$$y(t) = A_{carrier} \cdot \sin\{2 \cdot \pi \cdot f_{carrier} \cdot t + g(t)\} \tag{5.2}$$

$$g(t) = A_{mod} \cdot \sin(2 \cdot \pi \cdot f_{mod} \cdot t) \tag{5.3}$$

You may have noticed that the above equation actually does not directly modulate the carrier frequency, but rather contributes to the instantaneous phase, and in fact resembles *phase modulation*. We will, however, see below why Eq. (5.1) is actually referred to as FM synthesis shortly.

From a musician's point of view, frequency modulation is commonly observable in music performance situations, such as during voice performances. We all have experienced vocal performers employ vibrato when they sing, especially when holding a high pitched note at the pinnacle of a melody, often embellished with low frequency oscillation of the held pitch. This essentially is frequency modulation. Frequency modulation synthesis, in the traditional sense uses a modulation frequency that is in the pitch range. This condition does not produce the perception of low frequency modulation in the form vibrato, but rather a change in timbre.

This is similar to amplitude modulation — when the modulation frequency of AM is below the pitch range, the output becomes a time event and we perceive it as beating or tremolo. On the other hand, when the modulation frequency is in the pitch range, the result is perceived as change in timbre.

5.1 *FM: Sidebands and the Bessel function*

What makes FM synthesis unique is above all, the resulting timbre. It has seen a great deal of commercial success especially in the 1980s with the legendary digital synthesizer *Yamaha DX 7* (still used today). The timbre in FM synthesis is manipulated via the so-called *sidebands*. Recall in amplitude modulation synthesis that we were able to describe the resulting synthesized signal via the sum and difference frequency components. These two components are referred to as sidebands — the carrier at the center, flanked by frequency components $f_1 + f_2$ and $f_1 - f_2$ corresponding to the sidebands. In FM synthesis, there are many more of these sidebands making the synthesized timbre much more complex and interesting. It also turns out that the sidebands adhere to the following relationship where k is an integer:

$$f_{sidebands} = f_{carrier} \pm f_{mod} \cdot k \tag{5.4}$$

This is depicted in Fig. 5.1 where we observe the flanking modulation frequency components (sidebands) around the carrier frequency $f_{carrier}$. Putting FM synthesis in the more traditional form with modulation index β we have:

$$y(t) = A_{carrier} \cdot \cos(\omega_{carrier} t + \beta \cdot \sin(\omega_{mod} t)) \tag{5.5}$$

Equation (5.5) can further be expressed in terms of its sideband frequency amplitude components as shown in Eq. (5.6).

$$y(t) = \sum_{p=-\infty}^{p=+\infty} J_p(\beta) \cdot \cos\{(\omega_c + p \cdot \omega_m)\} t \tag{5.6}$$

$J_p(\beta)$ is referred to as the *Bessel function* and serves as the weights (amplitude values) for the carrier and sideband components $[\cos(\omega_c + p \cdot \omega_m)]$. The sideband $\omega_m = 2\pi f_m$ (radians) is dictated by frequency f_m (Hertz) and integer p controlling the sideband number — the sidebands spread out from the carrier frequency f_c. The Bessel function is shown in

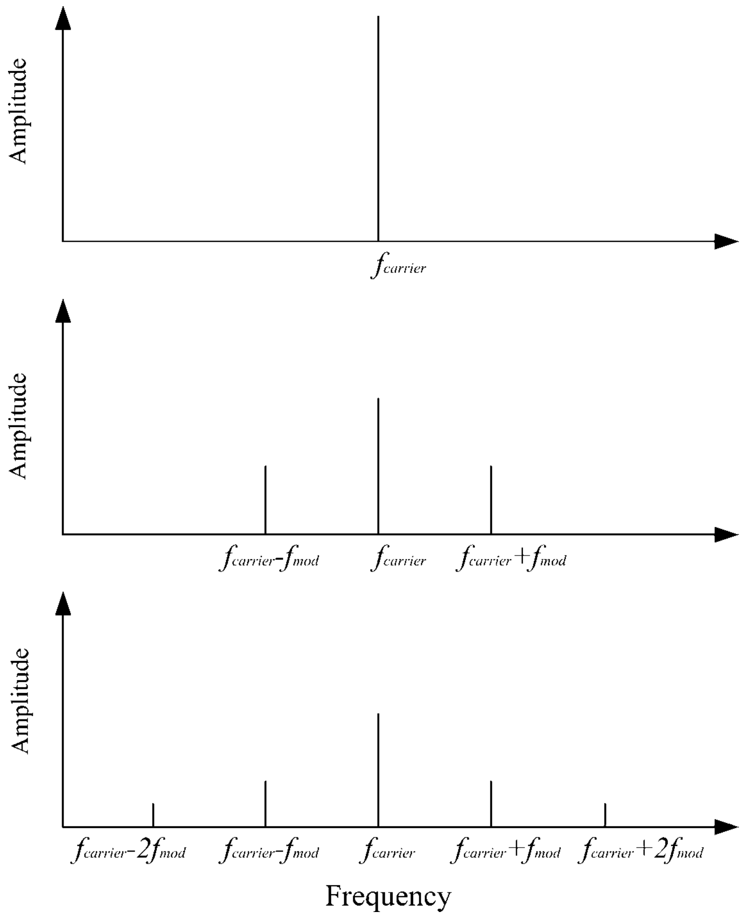


Fig. 5.1. Carrier (top), carrier and two side bands (middle), and carrier with multiple sidebands (bottom).

Eq. (5.7). Table 5.1 summarizes the parameters used in the Bessel function.

$$J_p(\beta) = \sum_{k=0}^{\infty} \frac{(-1)^k \cdot \left(\frac{\beta}{2}\right)^{p+2k}}{k!(p+k)!} \tag{5.7}$$

Equation (5.7) is also sometimes expressed as:

$$J_p(\beta) = \sum_{k=0}^{\infty} \frac{(-1)^k \cdot \left(\frac{\beta}{2}\right)^{p+2k}}{k!\Gamma(p+k+1)} \tag{5.8}$$

Table 5.1. FM synthesis parameters.

Parameter	Description
p	Order of sidebands (0 is carrier, 1 is first sideband, etc...)
k	Integer from $0 \sim K$
$J_p(\beta)$	Bessel function amplitude of sidebands and carrier
β	Modulation index

Notes:
$0^0 = 1$
$0^q = 0$, $q > 0$ and $q \neq 0$
$0! = 1$

where

$$\Gamma(p) = (p-1)! \tag{5.9}$$

To better understand what is happening and how the Bessel function works, consider an example where $p = 0$ in Eq. (5.6). The $p = 0$ case refers to a synthesized output signal that includes the carrier only as we have no sidebands at all and get:

$$\begin{aligned} y(t)\big|_{p=0} &= J_p(\beta)\cdot\cos[(\omega_c + p\cdot\omega_m)]t \\ &= \left\{\sum_{k=0}^{\infty}\frac{(-1)^k\cdot\left(\frac{\beta}{2}\right)^{0+2k}}{k!(0+k)!}\right\}\cdot\cos[(\omega_c + 0\cdot\omega_m)]t \\ &= \left\{\sum_{k=0}^{\infty}\frac{(-1)^k\cdot\left(\frac{\beta}{2}\right)^{2k}}{k!k!}\right\}\cdot\cos(\omega_c)t \end{aligned} \tag{5.10}$$

That is, we only have the carrier $\cos(\omega_c t)$ component as there are no sidebands at $p = 0$. When $p = 1$, we have one sideband added at both sides of f_c: $\omega_c \pm \omega_m$. Thus, the subsequent sideband's amplitude values are also computed using the same approach as above by increasing the sideband order p.

$$\begin{aligned} y(t)\big|_{p=1} &= \sum_{p=-1}^{p=1} J_p(\beta)\cos[(\omega_c + p\cdot\omega_m)]t \\ &= J_{-1}(\beta)\cdot\cos[(\omega_c - 1\cdot\omega_m)]t \\ &\quad + J_0(\beta)\cdot\cos[(\omega_c + 0\cdot\omega_m)]t + J_1(\beta)\cos[(\omega_c + 1\cdot\omega_m)]t \end{aligned} \tag{5.11}$$

If we expand sideband at $p = +1$ in (5.11) we have the following:

$$\begin{aligned} &J_1(\beta) \cdot \cos[(\omega_c + 1 \cdot \omega_m)]t \\ &= \left\{ \sum_{k=0}^{\infty} \frac{(-1)^k \cdot \left(\frac{\beta}{2}\right)^{1+2k}}{k!(1+k)!} \right\} \cdot \cos[(\omega_c + 1 \cdot \omega_m)]t \end{aligned} \tag{5.12}$$

Observe that the numerator part of $J_p(\beta)$ will always be 0 for $p > 0$ and $\beta = 0$, due to the fact that $(\beta/2)^{p+2k} = 0$; it will be a non-zero value only for $k = 0$ ($0^0 = 1$). Note also that the total energy is all encompassed in the carrier component when there is no modulation ($\beta = 0$ and $p = 0$) which translates to no sidebands. However, as the modulation index is increased the number of sidebands, and as consequence, the sound's complexity changes accordingly as shown in Figs. 5.1 and 5.2. The result is that the total energy from the carrier components gets distributed to the sidebands. For example, plugging in $\beta = 2$ at $p = 0$ into Eq. (5.7) yields the following results:

$$\begin{aligned} &\sum_{k=0}^{\infty} \frac{(-1)^k \cdot \left(\frac{2}{2}\right)^{2k}}{k!k!} = \sum_{k=0}^{\infty} \frac{(-1)^k \cdot (1)^{2k}}{k!k!} \\ &= \sum_{k=0}^{\infty} \frac{(-1)^k \cdot}{(k!)^2} = \left(\frac{1}{1}\right) - \left(\frac{1}{1}\right) + \left(\frac{1}{4}\right) - \left(\frac{1}{36}\right) + \left(\frac{1}{157}\right) - \cdots \end{aligned} \tag{5.13}$$

To summarize, when $\beta = 0$ (no modulation) only the carrier exists, when β increases in value, the bandwidth increases whereby the energy which was initially fully contained in the carrier at $\beta = 0$, is spread and distributed to the sidebands as determined by the Bessel function. Also, the larger the value of p (sideband order) the smaller the Bessel function becomes. Hence, the characteristics of a FM synthesized sound are a function of the modulation index β and the modulation frequency.

The output characteristics as a function of β and p work well in the context of musical sounds, especially in the area of musical instrument sounds — acoustic instruments generally display decaying harmonic amplitude characteristics. Generally, when observing the harmonic structure of an instrument sound, higher harmonics (those with higher frequency) tend to have less energy whereas lower frequency harmonics more energy. As we can see in Fig. 5.2, the FM synthesis algorithm mimics this behavior via the Bessel function.

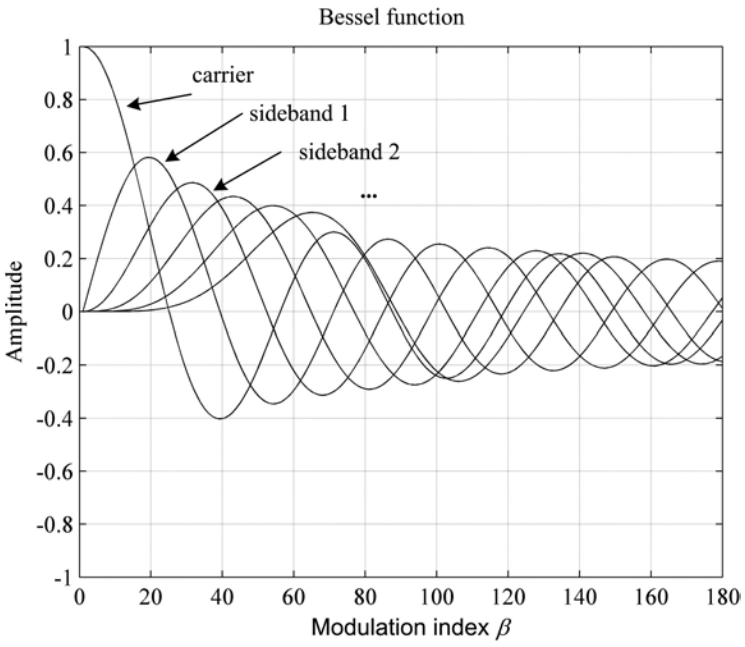


Fig. 5.2. Bessel function for carrier and first 5 sidebands.

5.2 *Modulation index*

As discussed in the previous sections, the modulation index in FM synthesis corresponding to the scalar multiplier of the modulator amplitude, determines the sideband make-up, allowing for control over the timbre of the synthesized sound. The modulation index β is, however, often expressed as the ratio between the modulation amplitude to the modulation frequency according to Eq. (5.14).

$$\beta = \frac{A_{mod}}{f_{mod}} \tag{5.14}$$

This ratio between the modulator amplitude and frequency can be derived solving the integral of the instantaneous phase. Let's start by first considering the cosine oscillator shown in Eq. (5.15).

$$y(t) = A_c \cdot \cos(2 \cdot \pi \cdot f \cdot t) \tag{5.15}$$

We know that in the definition of FM synthesis, frequency is modulated. That is, f becomes a function of time $f(t)$ — the instantaneous frequency as discussed in the beginning of Sec. 5. $f(t)$ in turn is defined by the carrier frequency f_c and modulator component $A_m x(t)$ as shown in Eq. (5.16).

$$f(t) = f_c + A_m \cdot x(t) \tag{5.16}$$

When the modulator $x(t)$ is itself also a sinusoid, the modulated instantaneous frequency $f(t)$ becomes:

$$f(t) = f_c + A_m \cdot \cos(2 \cdot \pi \cdot f_m \cdot t) \tag{5.17}$$

The instantaneous phase of the signal $y(t)$ can be represented by substituting scalar f in Eq. (5.15) with the instantaneous frequency $f(t)$, and taking the integral of the instantaneous frequency $f(t)$ as shown below.

$$\begin{aligned} y(t) &= \cos\left(2 \cdot \pi \cdot f(t) \cdot t\right) \\ &= \cos\left(2 \cdot \pi \cdot \int_0^t \{f_c + A_m \cdot \cos(2 \cdot \pi \cdot f_m \cdot \tau)\}\, d\tau\right) \\ &= \cos\left(2 \cdot \pi \cdot \int_0^t f_c d\tau + 2 \cdot \pi \cdot A_m \cdot \int_0^t \{\cos(2 \cdot \pi \cdot f_m \cdot \tau)\}\, d\tau\right) \\ &= \cos\left(2 \cdot \pi \cdot f_c \cdot t + \frac{A_m}{f_m} \cdot \sin(2 \cdot \pi \cdot f_m \cdot t)\right) \end{aligned} \tag{5.18}$$

We can see that when the smoke finally clears, we get the same results as Eq. (5.1) — this is what we started off this section with. However, as seen at the end of (5.18) above, we now have the modulation index defined as the ratio between the modulation amplitude and modulation frequency as shown in (5.19).

$$y(t) = A_c \cdot \cos\left(2 \cdot \pi \cdot f_c \cdot t + \frac{A_m}{f_m} \cdot \sin(2 \cdot \pi \cdot f_m \cdot t)\right) \tag{5.19}$$

The discrete version can now be easily obtained from (5.19) becoming Eq. (5.20) where n is the sample number as usual. This equation can now be used to implement FM synthesis in software.

$$y[n] = A_c \cdot \cos\left(2 \cdot \pi \cdot f_c \cdot \frac{n}{f_s} + \frac{A_m}{f_m} \cdot \sin\left(2 \cdot \pi \cdot f_m \cdot \frac{n}{f_s}\right)\right) \tag{5.20}$$

5.3 *General topics in FM control parameters*

It goes without saying that the modulation index and modulation frequency are the most important aspects in FM synthesis. In general, the greater the modulation index the greater the spread of the sidebands. There are some basic FM synthesis attributes that we can talk about without going into too much detail — how to make the FM synthesis algorithm sound like a trumpet or bell for example.

Looking back at Eq. (5.16) we note that A_m controls the amount of frequency deviation from center frequency f_c with the resulting peak frequency deviation determined by A_m.

$$\Delta f = A_m \tag{5.21}$$

Interestingly enough, this means that the frequency swing from f_c is not a function of the modulation frequency f_m but rather modulation amplitude A_m. Equation (5.19) is thus often seen in the format shown below for analog and digital versions of FM synthesis.

$$y(t) = A_c \cdot \cos\left(2 \cdot \pi \cdot f_c \cdot t + \frac{\Delta f}{f_m} \cdot \sin(2 \cdot \pi \cdot f_m \cdot t)\right) \tag{5.22}$$

$$y[n] = A_c \cdot \cos\left(2 \cdot \pi \cdot f_c \cdot \frac{n}{f_s} + \frac{\Delta f}{f_m} \cdot \sin\left(2 \cdot \pi \cdot f_m \cdot \frac{n}{f_s}\right)\right) \tag{5.23}$$

Here, we have simply replaced A_m by the peak frequency deviation Δf in Hertz, making the interpretation and usage of the FM algorithm more intuitive.

To make interesting sounds or model existing acoustic instruments (Schottstaedt 1985), the simple FM algorithm has to be tweaked in many ways. One important aspect is the carrier frequency to modulation frequency ratio commonly referred to as the *c*:*m* ratio, where *c* and *m* are carrier/modulation integer numbers (Truax 1985). The ratio of *c*:*m* is of course important as they will affect the sideband characteristics. I will, however, not spend too much time on this here, as a lot has already been written about this subject. The topic on creating specific timbre is also widely available on the Internet for those who would like to find out more. Before leaving our discussion in FM synthesis, there is, however, an important issue in controlling the timbral quality of FM that I want to touch upon — dynamically changing FM synthesis control parameters. That is, when modeling brass sounds for example, it is important to have the control parameters change over time, expressed in the form of control envelopes. The notion of dynamic control change is critical, as the quality of sound changes during the lifetime of an acoustic instrument. Generally speaking, the beginning part of a sound (attack) consists of a rich spectrum making it bright (higher modulation index), which then transitions into a more steady-state region with higher partials and harmonics gradually dying off (becomes duller and requires fewer FM sidebands) towards the end of the waveform. Thus, using an envelope for the FM algorithm parameters is essential to render close approximations to acoustic instrumental sounds as well as timbres that sound "alive" and "natural."

6 Musical Examples

There are countless examples of sine waves and their applications in music, including when they are used with the concept of beating and tuning. For those of you who are not guitar players, you may have heard and seen guitarist tune their strings while playing harmonics on two strings. The reason for that is actually quite simple, and it focuses on the concept of beating. If we play the same notes on two different strings we will hear beating if they are not exactly in tune. If they are in tune, beating will not be audible. The reason harmonics are used instead of notes that are held down on a fret is simply because it is difficult to hold down notes and simultaneously turn the tuning pegs (we only have two hands). Compositional examples of beating can also be seen in Alvin Lucier's album called *Still and Moving Lines of Silence in Families of Hyperbolas* from the early 1970s. In the twelve pieces in this album, Lucier has musicians sound sixteen long tones separated by silences which are played against one or two fixed oscillator tones. For each note played, the performed tones coming from the musicians are slightly raised or lowered producing audible beats of various speeds. Thus, the larger the frequency differences between the synthetic tones to the performed tones, the faster the beating — when in total unison, no beating will be perceived. This example of using interesting concepts borrowed from the scientific community for musical endeavors is just one of many compositional strategies that can be found in Alvin Lucier's works.

The concept of sum and difference frequencies has also been exploited in musical applications. One notable person who used this concept is Leon Theremin, the inventor of a unique monophonic instrument called the *theremin*. It has found usage by rock bands such as *Led Zeppelin* and many others, although the most virtuoso performer probably was Clara Rockmore. The theremin is an instrument like no other and is controlled by simply waving your hands in close proximity to two antennae without ever making any physical contact with the machine. The basic theory in changing the audible monophonic pitch relies on two essential components — the sum and difference concept and capacitance. The theremin has two antennae, one vertical for pitch control and one horizontal for amplitude control. Two oscillators are used for the pitch antenna tuned to frequencies beyond the audible range, with one fixed while the other changeable. It so turns out that when the hand gets close to the antenna, a slight change in capacitance occurs, this in turns faintly alters the oscillator's resonant frequency. When the hand is far away enough from the antenna, there will be no change in the frequency of the dynamic oscillator thus the sum and difference of the frequency will be 0 Hz plus some very high frequency that is outside our hearing range. However, as the hand approaches the antenna causing a slight change in the dynamic oscillator's resonant frequency, the resulting sum and difference frequency will also be affected. The sum of the frequencies of the two oscillators will always be beyond our upper hearing limits as before and thus inaudible, the difference tone, however, will be detectable as it lies within our hearing range and pitch range.

FM synthesis is largely regarded as one of the most successful synthesis algorithms for timbre manipulation and has especially seen wide usage and acceptance not just by specialists in academia and labs, but also by the general public. The success of the algorithm can be attributed not only to its possibilities of mimicking existing instrument sounds or synthesizing totally "new" sounds, but also to the fact that it is an algorithm that can produce a very rich and complex sound without having to use a multitude of expensive oscillators. If we took a time-machine back to the 1980s, we may remember that CPUs were very slow and expensive (and people wore some serious baggy pants with *Thompson Twins* leading the way). In other words, it was very difficult for the personal computer user and the general public to have access to digital synthesizers — until the advent of FM synthesis. With FM synthesis, the whole climate changed as one could make incredibly rich sounds with just two oscillators and a couple of envelopes. As a consequence, a large portion of the PC soundcards incorporated a FM synthesis-based sound engine until it more or less got replaced by PCM-driven soundcards. The mastermind behind FM synthesis is John Chowning who also composed some memorable pieces including *Phoné* (1980–1981), *Turenas* (1972), and *Stria* (1977), commissioned by IRCAM (Institut de Recherche et Coordination Acoustique/Musique) for one of the institute's major concert series entitled *Perspectives of the* 20th *Century*. *Stria* is a classic example showcasing the vast timbral capabilities of FM synthesis and also offers a demonstration of utilizing a sound synthesis algorithm to compose a musical artwork. *Stria* is rigorously composed (rather than improvised) and is entirely put together using FM synthesis algorithms. It also exploits various aspects of the Golden mean ratio (1.608) on a structural level as well as pitch level. In particular, Chowning used the Golden mean for the carrier frequency to modulation frequency ratio c:m. In using the Golden mean with a c:m ratio that is 1 to some power of the Golden mean, the resulting spectral components in the low-order range would also adhere to powers of the Golden mean. The sonic results are quite remarkable not only keeping the piece consistent on an architectural design level, but also presenting a unified structure of spectral information via the ensuing music.

You probably have heard ring modulation or amplitude modulation in musical compositions and other audio contexts even if you were not aware of it — a good example being sound effects used in science fiction movies. In particular, they have found much popularity by the sound effects guys who have applied the synthesis method (especially before digital audio technology became accessible) to make "robotic" voice sounds. In musical examples, ring modulation can be heard in the classic work of Stockhausen's *Mantra* released in 1970. In this piece, the composer uses two ring modulated pianos, electronics, shortwave radio, and other sound sources. The two pianos corresponding to the carriers are modulated in real-time via sine wave modulator signals. The modulating sine waves change according to the score, creating a distinct and "new" piano sound similar in concept to John Cage's *prepared piano*. The difference between Cage's and Stockhausen's usage of altering the timbre of the piano is that there are physical elements inserted into the guts of the piano in Cage's piece, whereas in Stockhausent's *Mantra*, the pianos are modulated electrically without any physical contact. The ring modulation effect in *Mantra* is often quite subtle which actually makes the piece that much more interesting, as the subtlety between electronics and non-electronically manipulated sounds induces and encourages an intensive listening environment. More common examples of the usage of amplitude modulation are with electric guitars — applying sine waves for the modulator typically in a low frequency range around 1 to 7 Hz or so. The intro soundtrack to the TV series from 1990 *Twin Peaks* by David Lynch is a great example of amplitude modulation (especially the instrumental version). Here the composer Angelo Badalamenti utilizes a "clean" electric guitar sound (seemingly performed by either guitarist Eddie Dixon or Vinnie Bell), modulated by a low frequency sine wave setting up an eerie mood perfect for the atmosphere of the show.

References and Further Reading

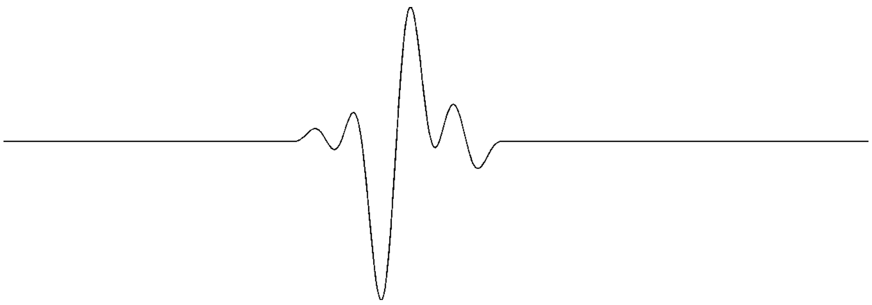
Chowning, J. M. 1973. "The Synthesis of Complex Audio Spectra by Means of Frequency Modulation", Journal of the Audio Engineering Society, 21(7), 526–534.

Schottstaedt, B. 1977. "The Simulation of Natural Instrument Tones Using Frequency Modulation with a Complex Modulating Wave", Computer Music Journal 1(4), 1977; reprinted in Foundations of Computer Music, C. Roads and J. Strawn (eds.). MIT Press, 1985.

Truax, B. 1978. "Organizational Techniques for C:M Ratios in Frequency Modulation", Computer Music Journal, 1(4), 39–45; reprinted in Foundations of Computer Music, C. Roads and J. Strawn (eds.). MIT Press, 1985.

Chapter 5

LINEAR TIME-INVARIANT SYSTEMS



1 Introduction

In this chapter, we will concentrate on LTI (linear time-invariant) systems, which are very important and have specific properties critical in many digital signal processing scenarios. It is probably not an exaggeration to say that much of the DSP techniques that we deal with are based on the approach of decomposing a complex problem into a number of simpler sub-problems, where each simpler sub-component is processed individually, and where the sub-components as a group yield the desired final result. Systems that adhere to LTI conditions allow us to readily employ this type of problem solving strategy for complex problems. The most common approach in describing LTI systems is through *difference equations.* We actually have already seen difference equations in action in Chap. 2 in the form of the moving average filter, as well as in Chap. 3, where we introduced the basic chorus effect. Before we jump into the details and define what LTI systems are, we will start this chapter by first revisiting our moving average algorithm introduced in Chap. 2.

2 Difference Equations: Starting with the Moving Average Algorithm

The moving average algorithm we used in Chap. 2, Sec. 5.1 is again shown in Eq. (2.1). You will remember that it was applied in smoothing out a signal to help locate zero-crossing points in fundamental frequency computation.

$$y[n] = \frac{x[n] + x[n-1] + \cdots + x[n-L-1]}{L} \tag{2.1}$$

$$y[n] = \frac{1}{L}\sum_{k=0}^{L-1} x[n-k] \tag{2.2}$$

n is the sample number and L an integer constant determining how many past input samples (pertinent to the window size) are to be added together and scaled. The equation above actually may be perceived to be more complicated than it actually is (a lot of math equations tend to do that for some reason or another). Upon closer scrutiny, however, we will realize that it is indeed something we are very familiar with — it is an equation that computes the arithmetic mean, presented in the form of a the so-called difference equation. The difference equation thus simply describes some sort of input to output relationship. Let's plug in a few numbers to see how this actually works by letting $L = 2$ and using an input sequence x as follows:

$$x[n] = \begin{cases} n, & 0 \le n \le 9 \\ 0, & \text{elsewhere} \end{cases} \tag{2.3}$$

The first few output values $y[n]$ are as shown below (we assume that $x[\text{negative } n] = 0$):

$$\begin{gathered} \ldots \\ y[-1] = \frac{x[-1] + x[-2]}{2} = \frac{0+0}{2} = 0 \\ y[0] = \frac{x[0] + x[-1]}{2} = \frac{0+0}{2} = 0 \\ y[1] = \frac{x[1] + x[0]}{2} = \frac{1+0}{2} = 0.5 \\ \ldots \end{gathered} \tag{2.4}$$

$$y[9] = \frac{x[9] + x[8]}{2} = \frac{9+8}{2} = 9.5$$
$$y[10] = \frac{x[10] + x[9]}{2} = \frac{0+9}{2} = 4.5$$
$$y[11] = \frac{x[11] + x[10]}{2} = \frac{0+0}{2} = 0$$
$$\cdots$$

As we can see, the arithmetic mean $y[n]$ is computed for each sample n by moving along the sequence of input samples. We could have also used a longer window size L, which would have yielded in a greater span of samples for the mean computation. The hop size, as introduced in Chap. 2, is also often set to integer L and not 1 as we have done in the above example. That is, if the hop size is equal to the window size, the window is shifted according to the length of the window size rather than a single sample. A word of caution is called for at this point, however, as a longer L yields less transient response — less transient response in a sense that more of the timing details get lost as the window size increases. Transient response as mentioned in Chap. 2, refers to the time-response or how fast a system reacts to the input. Below we can see the moving average algorithm with a controllable hop size M.

$$y[n] = \frac{1}{L}\sum_{k=0}^{L-1} x[M \cdot n - k] \tag{2.5}$$

To put this into perspective, let's say we use the above parameters of $L = 2$ to compute the moving average for the input samples with a hop size $M = 2$ samples rather than 1. This will give us the following moving average values (we again assume that $x[\text{negative } n] = 0$ as we did before):

$$\begin{aligned}
&\cdots\\
y[0] &= \frac{x[0\cdot 2] + x[0\cdot 2 - 1]}{2} = \frac{x[0] + x[-1]}{2} = \frac{0+0}{2} = 0\\
y[1] &= \frac{x[1\cdot 2] + x[1\cdot 2 - 1]}{2} = \frac{x[2] + x[1]}{2} = \frac{2+1}{2} = 1.5\\
y[2] &= \frac{x[2\cdot 2] + x[2\cdot 2 - 1]}{2} = \frac{x[4] + x[3]}{2} = \frac{4+3}{2} = 3.5\\
y[3] &= \frac{x[3\cdot 2] + x[3\cdot 2 - 1]}{2} = \frac{x[5] + x[4]}{2} = \frac{5+4}{2} = 4.5
\end{aligned} \tag{2.6}$$

$$y[4] = \frac{x[4 \cdot 2] + x[4 \cdot 2 - 1]}{2} = \frac{x[8] + x[7]}{2} = \frac{8+7}{2} = 7.5$$
$$y[5] = \frac{x[5 \cdot 2] + x[5 \cdot 2 - 1]}{2} = \frac{x[10] + x[9]}{2} = \frac{0+9}{2} = 4.5$$
$$y[6] = \frac{x[6 \cdot 2] + x[6 \cdot 2 - 1]}{2} = \frac{x[12] + x[11]}{2} = \frac{0+0}{2} = 0$$
$$\ldots$$

We can clearly note that for a hop size $M = 2$, we will have fewer number of non-zero output values $y[\cdot]$: 10 moving average values for $M = 1$ and 5 for $M = 2$. This is expected, as our update rate for the output is higher and computed at every other sample for $M = 1$ and slower when computed at every 2 samples for $M = 2$. For such small number of samples and small window size L, this may not be such a big factor, but when we are dealing with larger number of samples and higher sampling rates, it becomes an important issue. Therefore, we have to be mindful in choosing the right window size L and hop size M.

2.1 *Causality*

In the above moving average example, we may have noticed that for $n < 0$ nothing happens to the output. That is, the output is equal to 0. Such systems are referred to as *causal systems* — only when the "power switch" is turned on at $n = 0$, does the system output a value. Stated in another way, a system is causal if the output depends solely on present and past input values. Hence, *non-causal systems* can be regarded as systems that have some sort of initial state that is non-zero and affects the output before an input value is given to the system. Figure 2.1 depicts this concept where the top figure represents the input sequence, middle figure a causal system that only outputs when there is an input at sample number $n = K$, and a non-causal system at the bottom which outputs something even before it receives any input data.

2.2 *Difference equations: General form*

The above example of the moving average is called a non-recursive difference equation as there is no feedback component — it does not have a "y" component in the form of $y[n - m]$ where integer $m > 1$. The general form of the difference equation is shown in Eq. (2.7) which includes both

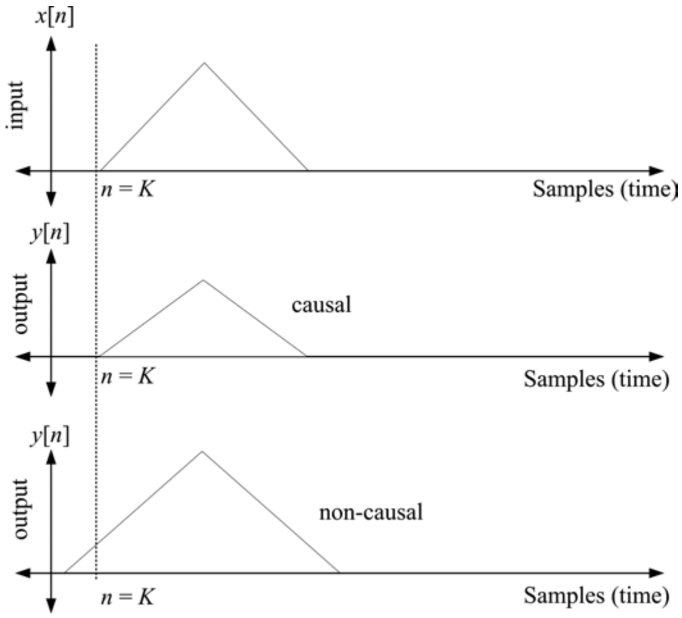


Fig. 2.1. Causal and non-causal example.

the non-recursive and recursive parts.

$$y[n] = a_1 y[n-1] + a_2 y[n-2] + \cdots + a_N y[n-N] + \cdots$$
$$b_0 x[n] + b_1 x[n-1] + b_2 x[n-2] + \cdots + b_L x[n-L] \quad (2.7)$$

Equation (2.7) can also be expressed in a more compact form using summation as shown in Eq. (2.8).

$$y[n] = \sum_{k=1}^{K} a_k y[n-k] + \sum_{k=0}^{L} b_k x[n-k] \quad (2.8)$$

The first part of the right-hand side ($y[\cdot]$) determines the recursive characteristic and the second part of the right-hand side ($x[\cdot]$) the non-recursive characteristic. The general difference equation thus represents the input/output relationship which can describe complex systems with simple addition and multiplication based on present and past input samples at time n.

Equations (2.7) and (2.8) are the general forms of the difference equation, and as we can now see, our moving average algorithm is just a special version of the general difference equation. The moving average algorithm happens to have no feedback components and the scalar b_k's are all equal to $1/L$. We can thus understand the moving average type difference equation as simply the averaging of a couple of numbers resulting in smoothing of a signal as observed in Chap. 2, Sec. 5.1. In essence, by setting the scalars (also referred to as coefficients or weights) and the length (also referred to the order or size) of the difference equation, we can design specifically behaving systems known as *filters*. This is what the moving average algorithm in effect represents and is hence called the moving average filter. We will discuss filters in more detail in Chap. 7.

Note that the moving average difference equation is predicated by the assumption that multiplying the input with coefficients a_k, and delaying them [k in Eqs. (2.7) and (2.8)] will not cause any "problems." Difference equations which satisfy LTI (linear-time invariant) properties explained in the next section make a system like the moving average system work without "problems" — without having to worry about issues that may arise due to operations such as delay and scalar multiplication of each of the input (and output) components.

3 Linear-Time Invariant (LTI) Systems

Imagine we are in a home recording studio where we are recoding a 3-piece band consisting of a bass guitar, electric guitar, and drum kit. We are limited to using one microphone for the bass and guitar each, and 2 microphones for the drum kit (we don't have any more microphones due to our low budget). Surprisingly, the out-of-commission wooden sauna room which we used as a practice space is very "flat" sounding and each performer takes turns in playing and recording each part onto a digital audio workstation (DAW). It so happened that the bass player could not make it to the session at the last minute due to some errands, and recorded her track the following day, whereas everyone else was able to put down their parts. After spending 15 hours in the studio, with way too much coffee coupled with only a few hours of sleep, the three musicians are now ready to mix-down (sum) the signals (drums, bass, and guitar) by amplifying and attenuating (multiply) each track to a final format consisting of two audio channels (stereo left and right). They are thus, for the first time, able to monitor the sound of their band without having to play at the same time and the drummer says — "we're better than I thought" ... The resulting perception of the mix, although the musicians played their instruments separately and at different times and not together as a band, sounds as if they are all performing concurrently at the same time and same space. On top of that, during the mix-down session, a lot of equalization and effects were used rendering a pretty impressive demo tape. The reason this scenario is possible — recording each instrument individually at different times and mixing them without artifacts, is due to linearity and time invariance properties — the topic of this section.

3.1 *Linearity property: Scalability and superposition*

Linearity as the term suggests, comes from the word line and basically represents any function which behaves in such a way — the input and output creates a line as seen in Fig. 3.1.

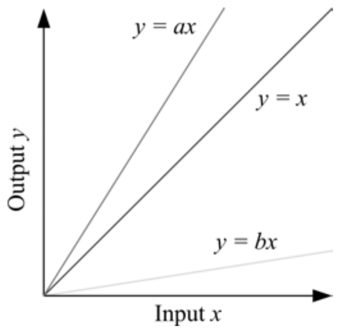


Fig. 3.1. Linearity examples.

If we look at linearity more closely, a linear function is a function that satisfies two conditions — *scalability* (homogeneity property) and *superposition* (additive property). The scalability property for linearity is probably the easier concept of the two to grasp and is met when the following condition is met as shown in Eq. (3.1).

$$T(\alpha \cdot x[n]) = \alpha \cdot T(x[n]) = \alpha \cdot y[n] \tag{3.1}$$

$T(x[n])$ is the output for input $x[n]$ and α is a scalar multiplier. The superposition property on the other hand utilizes the additive property as shown in Eq. (3.2) and (3.3). If the scalability and superposition conditions are met for a particular system, we say that it is linear.

$$x_3[n] = x_1[n] + x_2[n] \tag{3.2}$$

$$\begin{aligned} T(x_3[n]) &= T(x_1[n] + x_2[n]) = T(x_1[n]) + T(x_2[n]) \\ &= y_1[n] + y_2[n] = y_3[n] \end{aligned} \tag{3.3}$$

In a nutshell, the above states that the output of the system will not change whether one subjects a system with each input individually/separately or if all inputs are combined (summed) beforehand and inputted to the system together. This can be likened to the guitar and drums playing together onto one audio track vs. recording them separately onto individual tracks and summing them afterwards — the results will be the same. Actually, it is common practice to record the snare, high-hat, toms, and cymbals all on separate tracks in order to have flexible control during the mix-down and production phase of the studio recording process.

3.2 *Time-invariance property: Time-shift invariance*

A time-invariant system is characterized by the dependence of the output to a time-shifted input. In such a system, a time-shift in the input component will have a corresponding time-shift in the output. The time invariance property is met if the following condition is upheld.

$$y_1[n] = x_1[n] = x[n-L] = y[n-L] \tag{3.4}$$

Thus, a system is said to be time-invariant, if the shifted input $x_1[n](= x[n-L])$ produces an output $y_1[n]$ which is equally delayed by L samples and characterized $y[n-L]$. In other words, a delay imposed on the input should simply correspond to the same delay in the output. This can be likened to the bass player from the above example where she comes to lay down the bass tracks a day later unaffecting the final mix (as if they had played together at the same time) — a time-invariant system does not get affected by *when* the signal in question is presented to the system. Systems that meet both linearity and time-invariance properties are referred to as LTI systems.

Let's look at some examples to get a better feel for determining if a system is LTI with the following classic example: $y[n] = x^2[n]$. We will start testing $y[n]$'s linearity characteristics by inspecting its superposition and scalability property. For scalability, we find that it is not scalable — we multiply the input by α before subjecting it to the system and multiply the system by α itself and compare the results.

$$T(\alpha \cdot x[n]) = \alpha^2 \cdot x^2[n] = \alpha^2 \cdot y[n] \tag{3.5}$$

$$\alpha \cdot T(x[n]) = \alpha \cdot x^2[n] = \alpha \cdot y[n] \tag{3.6}$$

Clearly (3.5) and (3.6) are not the same and as this example has already failed one of the linearity properties, it is nonlinear. We can verify this just by plugging in some numbers as well. For example let's set $\alpha = 2$ and $x[n] = 6$. This means $2 \cdot 6 = 12$ and the final output is $12^2 = 144$ (scaling the input and subjecting it to system). Scaling the system by α yields $2 \cdot (6^2) = 72$. Let us now check the superposition condition as well for fun (although we already have established that it is not linear). For superposition, we use $x_3[n] = x_1[n] + x_2[n]$ to find that additivity is not met as $T(x_1[n] + x_2[n])$ is not equal to $y_1[n] + y_2[n]$ [see (3.3)]. That is, (3.7) is not equal to (3.8) — there is an extra term: $2 \cdot x_1[n] \cdot x_2[n]$. Since superposition is not met it cannot be linear.

$$\begin{aligned} T(x_3[n]) &= T(x_1[n] + x_2[n]) = (x_1[n] + x_2[n])^2 \\ &= x_1^2[n] + 2 \cdot x_1[n] \cdot x_2[n] + x_2^2[n] \end{aligned} \tag{3.7}$$

$$y_3[n] = y_1[n] + y_2[n] = x_1^2[n] + x_2^2[n] \tag{3.8}$$

For time-invariance verification, we use (3.4) and compare the delayed input and delayed output. Clearly (3.9) and (3.10) are identical and hence the system is time-invariant although nonlinear. It is therefore a non-LTI system.

$$y_1[n] = x_1[n] = x^2[n - L] \tag{3.9}$$

$$y[n - L] = x^2[n - L] \tag{3.10}$$

Another example is shown below in (3.11). We can probably tell from inspection that it is linear and will skip testing linearity.

$$y[n] = x[n/M] \tag{3.11}$$

To test for time-variance, however, we use the same approach as in our previous example and get the results shown in (3.12) and (3.13). As we can see, the two equations are not the same and hence time-variant. Thus, both of our examples are actually non-LTI systems.

$$y_1[n] = x_1[n] = x[n/M - L] \tag{3.12}$$

$$y[n - L] = x[(n - L)/M] \tag{3.13}$$

Using the same method for testing LTI outlined in our two examples, we will find that the moving average filter we were introduced to in the beginning of this chapter, is in fact an LTI system — all it actually embodies is a collection of delayed input samples and input coefficients equal to the scalar $1/L$.

3.3 *Importance of LTI systems in DSP*

When a system is linear, it will adhere to properties of superposition and scalability. This is very important in DSP as a considerable bulk of topics and problems therein are based on the notion of *divide and conquer*. For example, a complex system can, thanks to linearity and time-invariance, be divided into smaller sub-systems that are less complex. This is very powerful, as complex problems can often be decomposed and rearranged in some preferred order and into simpler, smaller, and more manageable problems and ultimately be recombined to render the final result as depicted in Fig. 3.2.

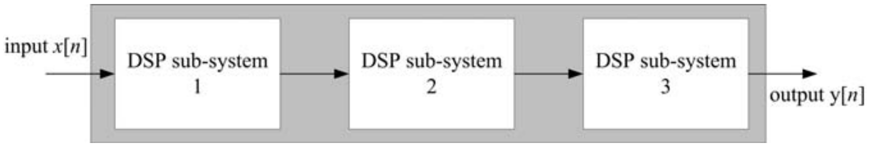


Fig. 3.2. Divide and conquer and LTI.

This idea can also be extended to multi-input systems if each of the sub-systems are LTI as well. Such an example is shown in Fig. 3.3, which is the recording scenario of the 3-instrument band we began Sec. 3. Thus, any system with more than one input and output will be LTI if each sub-system is LTI as well.

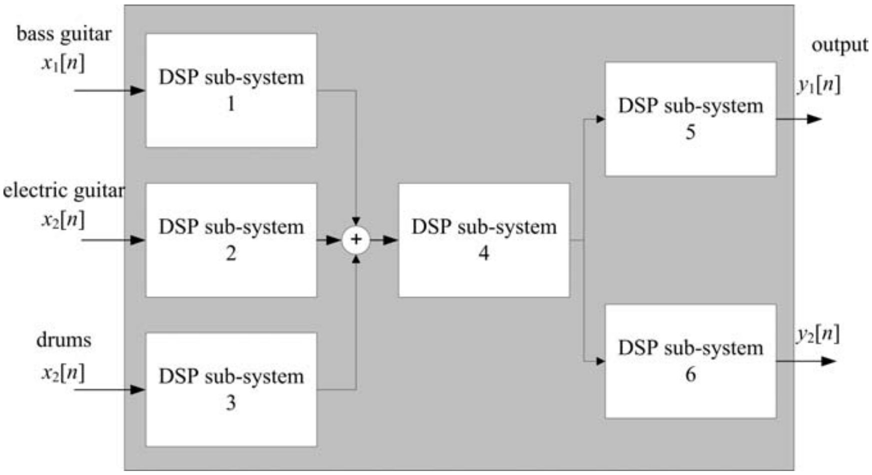


Fig. 3.3. Multiple inputs and outputs.

4 Impulse Response

Some of us may remember when we were little kids finding something strange in the garden that looked like a lizard or something scary and were unsure of its identity and wondering if it was even something "alive." At least I remember that whenever I found something unusual, but could not identify it, and was not too scared, I would poke at it with a stick to see if it was something alive or just part of a dead tree branch that just happened to have a peculiar shape. This is kind of what the impulse response is. That is, poking a system or that thing to see what kind of characteristics it has by observing the system after the poke — in the case of the lizard-looking thingy, the poke, would probably equate to the lizard rapidly running away or in an extreme case biting you!

This poking and observing the output in signal processing is done via the so-called *impulse*, *unit impulse*, or *delta function* which all refer to the same thing. Theoretically, the impulse is infinitely short (time-wise) and infinitely high (amplitude-wise) whose integral (area) is equal to 1. This is obviously more of a conceptual idea and does not exist in the physical world. The impulse is defined as shown in Eq. (4.1) and is symbolized by the Greek letter delta (δ). The observation of the impulse response is achieved by poking the system with the input set to the unit impulse.

$$x[n] = \delta[n] \tag{4.1}$$

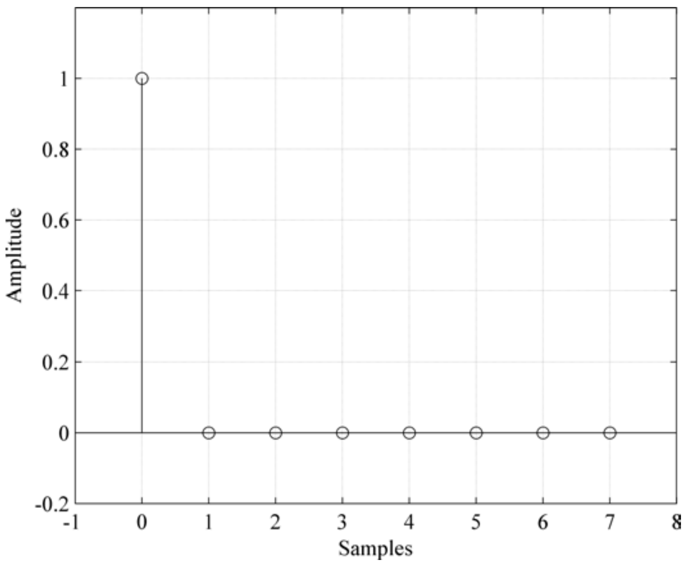


Fig. 4.1. Impulse signal.

The delta function is only defined at one sample point where it is normalized to 1 and is 0 elsewhere as shown below in Eq. (4.2) and Fig. 4.1.

$$\delta[n] = \begin{cases} 1, & n = 0 \\ 0, & \text{elsewhere} \end{cases} \tag{4.2}$$

Let's say we have a digital system that we know nothing about except that it is a non-recursive system. That is, it does not have any delayed $y[\cdot]$ components which in turn means that the difference equation will take the form of Eq. (4.3).

$$y[n] = \sum_{l=0}^{L} b_k \cdot x[n-l] \tag{4.3}$$

If we use an impulse as the input and observe the output or its impulse response, (4.3) will become as shown in Eq. (4.4) when we set $x[n] = \delta[n]$.

$$y[n] = \sum_{l=0}^{L} b_k \cdot \delta[n-l] \tag{4.4}$$

By using the definition of the delta function, we further observe that the output will result in the following manner.

$$\begin{aligned} y[0] &= \sum_{l=0}^{L} b_l \cdot \delta[0-l] = b_0 \cdot \delta[0] + \cdots + b_L \cdot \delta[0-L] \\ &= 1 \cdot b_0 + 0 + \cdots + 0 = b_0 \qquad (4.5) \\ y[1] &= \sum_{l=0}^{L} b_l \cdot \delta[1-l] = b_0 \cdot \delta[-1] + b_1 \cdot \delta[0] + \cdots + b_L \cdot \delta[1-L] \\ &= 0 + 1 \cdot b_1 + \cdots + 0 = b_1 \qquad (4.6) \\ &\quad \cdots \\ &\quad \cdots \\ y[L] &= \sum_{l=0}^{L} b_l \delta[1-L] = b_0 \cdot \delta[-L] + \cdots + b_{L-1} \cdot \delta[1] + b_L \cdot \delta[0] \\ &= 0 + \cdots + 0 + 1 \cdot b_L = b_L \qquad (4.7) \end{aligned}$$

In signal processing, the impulse response as computed above is formally denoted as $y[n] = h[n]$, when the input is an impulse sample as shown below.

$$x[n] = \delta[n] \qquad (4.8)$$

$$y[n] = h[n] \qquad (4.9)$$

From (4.6) and (4.7) we can say that $h[n]$ follows (4.10).

$$h[n] = \begin{cases} b_n, & n = 0 \ldots L \\ 0, & \text{elsewhere} \end{cases} \qquad (4.10)$$

Thus, the input and output relationship becomes:

$$x[n] = \delta[n] \rightarrow y[n] = h[n] \qquad (4.11)$$

and Eq. (4.3) becomes:

$$h[n] = \sum_{l=0}^{L} b_k \cdot \delta[n-l] = \sum_{l=0}^{L} h_k \cdot \delta[n-l] \qquad (4.12)$$

The importance of obtaining the impulse response as shown above may not be that apparent now. However, when we become familiar with convolution in Sec. 5 for example, we will see that if we know the impulse response

($h[n]$) of a system, we can determine the system's output for any given input sequence. This is, however, not the only important aspect of the impulse response. It is also used to gain insight into a system's characteristics including stability information as we shall see shortly and in more detail in Chap. 6, when we study the frequency response.

We can also experience impulse responses in the physical world including in your immediate surrounding space. Try clapping loudly (once only) in whatever room you are currently in and listen to the characteristics of that room (hopefully you are not in the library or in a crowded public bathroom). What you have done in essence is excited the system (the room) with an impulse (impulse = clap) and what you are hearing is the impulse response. You will note that as you clap in different types of spaces (or different parts of the same room for that matter), like your bathroom with tiles, a heavily carpeted room or in your car, the impulse response will yield very different types of responses. Although, the clap is by no means an ideal unit impulse as defined above, the experiment gives you a good sense as to why the unit impulse and impulse response are useful and important in DSP.

4.1 *Finite impulse response* (*FIR*) *and infinite impulse response* (*IIR*)

Finite and *infinite impulse response* systems, more commonly referred to as FIR and IIR systems, basically refer to systems that have finite output sequences and an infinite output sequences respectively. When an impulse excites a system and the output reaches a state where it only produces zeros, it is referred to as an FIR system. From the general difference equation we know:

$$y[n] = \sum_{k=1}^{N} a_k y[n-k] + \sum_{k=0}^{L} b_l x[n-k] \tag{4.13}$$

and if we take the $y[n-k]$ components out of the above equation we get:

$$y[n] = \sum_{k=0}^{L} b_k x[n-k] \tag{4.14}$$

Furthermore, using the definition of the impulse response we get the impulse response of the general FIR system according to Eq. (4.15).

$$h[n] = \sum_{k=0}^{L} b_k \delta[n-k] \tag{4.15}$$

On the other hand, if a system outputs non-zero values and never reaches a state where it outputs 0s, then that system is referred to as an IIR system. From the general difference equation we can directly get the impulse response of the IIR system as shown in (4.16) (replacing the $y[\cdot]$s with $h[\cdot]$s and $x[\cdot]$s to $\delta[\cdot]$s).

$$h[n] = \sum_{k=1}^{N} a_k h[n-k] + \sum_{k=0}^{L} b_k \delta[n-k] \tag{4.16}$$

For all practical purposes, however, stable IIR systems will eventually reach a "zero state" where very small values are outputted that are negligible in digital systems. IIR systems, however, can also become unstable and "blow up" — speakers' cones will end up rupturing due to too much feedback for example. In terms of recursive and non-recursive system characteristics or feedback and feed-forward characteristics as it is commonly referred to, IIR systems have a feedback part whereas FIR systems do not.

4.2 *Stability and IIR systems*

There are additional important issues concerning FIR and IIR systems, namely issues in stability. If a system is stable, it will generally at some point in time output zeros only (when the "power switch" is turned off), if it is not stable, its output may increase in amplitude and eventually break or "blow up." A great example of such an unstable system as mentioned in the previous subsection is the feedback one sometimes hears at concerts involving loudspeakers and microphones — the input signal to the microphone becomes the output of the loudspeakers, which in turn becomes the input to the microphone causing a vicious cycle of feedback amplification. This results in that infamous painful "howling" sound, which can in extreme cases damage the speakers and mixer and perhaps even worse, bring damage to one's ears.

Consider the following difference equation of an IIR system:

$$y[n] = x[n] + a \cdot y[n-1] \tag{4.17}$$

We compute the impulse response as usual by replacing the output and input with $h[n]$ and $\delta[n]$. That is,

$$y[n] = h[n] \tag{4.18}$$

$$x[n] = \delta[n] \tag{4.19}$$

and the difference equation (4.17) becomes the impulse response shown below:

$$h[n] = \delta[n] + a \cdot h[n-1] \tag{4.20}$$

For a causal system (nothing happens until you turn on the power switch), we have

$$h[n] = 0, \quad n < 0 \tag{4.21}$$

and the impulse response sequence results in:

$$\begin{aligned} h[0] &= \delta[0] + a \cdot h[-1] = 1 + 0 = 1 \\ h[1] &= \delta[1] + a \cdot h[0] = 0 + a \cdot 1 = a \\ h[2] &= \delta[2] + a \cdot h[1] = 0 + a \cdot a = a^2 \\ &\vdots \qquad\qquad\quad \vdots \\ h[k] &= \delta[k] + a \cdot h[k-1] = a^k \end{aligned} \tag{4.22}$$

Because it is an IIR system, we need to check for its stability (feedback may get out of control) and note that this system is unstable when $|a| > 1$. That is, if the absolute value of a is greater than 1, the output will eventually start increasing in amplitude without limitation. If $|a| < 1$, the system is stable and is said to be bounded. The reason the IIR system is unstable when $|a| > 1$ is due to the a^k in (4.22). That is, as k increases $|a^k|$ ($|a| > 1$) will grow without bound.

Notice that IIR systems have the potential of becoming unstable due to feedback components as we have seen above if we are not careful. FIR systems, however, are always stable as it has no opportunity to cause feedback — when we turn off the power switch, an FIR system will eventually come to rest at some point in time. This is summarized in Table 4.1.

Table 4.1. FIR and IIR comparison.

	Feedback component?	Necessarily reaches zero?	Inherently stable?
FIR	No	Yes	Yes
IIR	Yes	No	No

5 Convolution

In this section, we will examine convolution. Convolution is yet another very important concept in DSP and useful process for musical applications. As we mentioned at the beginning of Sec. 4, if we know the impulse response of a system without knowing its difference equation, we can determine the output of that system by using the theorem of convolution.

Let's first define convolution and then try to explain what its implications are and why it often comes up in DSP, especially when audio is involved. Since we now know how LTI systems work, we can say that if a system is LTI, an input unit impulse will yield an impulse response as shown below.

$$\delta[n] \xrightarrow{\text{yields}} h[n] \tag{5.1}$$

We can also state that a shifted delta function will result in a shifted impulse response as shown in (5.2) (for an LTI system).

$$\delta[n-m] \xrightarrow{\text{yields}} h[n-m] \tag{5.2}$$

Following the same line of thought, we can also say that any input sequence can be described as the sum of weighted and shifted samples, that is:

$$x[n] = \sum_{m=-\infty}^{m=+\infty} x[m] \cdot \delta[n-m] \tag{5.3}$$

You will note that $x[n]$ will only exist when $n = m$ in Eq. (5.3), meaning that the following will also stand true if it is an LTI system.

$$x[m] \cdot \delta[n-m] \xrightarrow{\text{yields}} x[m] \cdot h[n-m] \tag{5.4}$$

Finally, we can also add a summation to both sides of (5.4) due to the superposition property of linearity and arrive at Eq. (5.5).

$$\sum_{m=-\infty}^{m=+\infty} x[m] \cdot \delta[n-m] \xrightarrow{\text{yields}} \sum_{m=-\infty}^{m=+\infty} x[m] \cdot h[n-m] \tag{5.5}$$

The above relationship is called *convolution*. If we set the right-hand side of (5.5) to $y[n]$ we arrive at the formal definition of convolution as shown in (5.6), where the asterisk denotes the convolution operator. We say input x is convolved with h where h is the impulse response. Thus, as we asserted before, if you know the impulse response of a system without knowing its difference equation, you will be able to compute the output with just the impulse response and any input sequence.

$$y[n] = x[n] * h[n] = \sum_{m=-\infty}^{m=+\infty} x[m] \cdot h[n-m] \tag{5.6}$$

One important application of convolution is what is commonly referred to as convolution-based reverb. If you did the little experiment of clapping in your room, you may have realized that the impulse response of a room can be recorded and stored as a digital sequence ($h[n]$). Now, if you go to a big cathedral and clap you will hear long decaying sound characteristics which are due to the reflective surfaces and enormity of the space. This sound can obviously be recorded using a portable recorder as well. Upon transferring this impulse response (the sound from the point of the clap, until it reaches 0 amplitude) into your computer, you can next close-mike your voice (holding your microphone very close to your mouth) and capture a very "dry" voice recording (little room or space influence). Finally, using Eq. (5.6) you can convolve the two signals (voice: $x[\cdot]$, impulse response $h[\cdot]$ the cathedral) and the sonic result will be as if you were to hear your voice inside the cathedral without actually being in the cathedral. As a matter of fact, one could use any type of sound for the input — convolve your guitar solo excerpt with the cathedral's impulse response and get instant solo-guitar in the cathedral effect you always wanted to have.

Let's work on a numerical convolution example using the following input $x[n]$ and impulse response $h[n]$.

$$x[n] = \{1, 2, 3\} \tag{5.7}$$

$$h[n] = \{4, 5\} \tag{5.8}$$

We can set $m = 0 \ldots 2$ in Eq. (5.6) and express $y[n]$ as shown in (5.9), since there are no values before $x[0]$ and after $x[2]$, covering both x and h components (x has the greater number of samples of the two).

$$y[n] = x[n] * h[n] = \sum_{m=0}^{m=2} x[m] \cdot h[n-m] \tag{5.9}$$

Now, if we compute each output sample we get:

$$y[0] = x[0] \cdot h[0-0[+ x[1] \cdot h[0-1] + x[2] \cdot h[0-2] = 1(4) + 0 + 0 = 4 \tag{5.10}$$

and the subsequent output values simply become:

$$\begin{aligned} y[1] &= 13 \\ y[2] &= 22 \\ y[3] &= 15 \\ y[4] &= 0 \end{aligned} \tag{5.11}$$

This type of tedious number crunching work lends itself well to implementation as a computer program, be it C/C++, Java or MATLAB®. Although initially, programming such algorithms may seem a bit complex in the beginning, the work becomes easier and easier with practice. The reward of coding an algorithm such as convolution into a computer language is of course having an automatic convolution calculator!

5.1 *Convolution "Need to Knows"*

There are several principal properties that are important when it comes to convolution. We will briefly discuss a few in this section. Convolution is commutative meaning that it does not mathematically matter which part is designated as the impulse response and which part the input as the output will yield the same results.

$$y[n] = x[n] * h[n] = h[n] * x[n] \tag{5.12}$$

Thus, convolving x with h opposed h with x does not affect the result of the output. Linearity is by definition commutative and for convolution to work, the system in question needs to be an LTI system. We will see in Chap. 8 when we revisit convolution in the frequency-domain, how convolving pairs can be interchanged — convolution in the time-domain translates to multiplication in the frequency-domain. Speaking of the commutative property of convolution, we cannot neglect to mention its associative property as shown in Eq. (5.13).

$$y[n] = g[n] * (x[n] * h[n]) = (g[n] * x[n]) * h[n] \tag{5.13}$$

The above equation also shows that we can convolve more than two signals by using the results in the parenthesis as an intermediate step and convolving this part with the signal outside the parentheses. The distributive property for convolution is also met as shown in Eq. (5.14), thanks to linearity.

$$y[n] = g[n] * (x[n] + h[n]) = g[n] * x[n] + g[n] * h[n] \tag{5.14}$$

Another important issue when using convolution is the length of the resulting output signal and limiting the summation of Eq. (5.6) — when convolving two signals, with length M and N, the resulting length of the convolved signal will be as shown in (5.15).

$$convolution_{length} = M + N - 1 \tag{5.15}$$

Although Eq. (5.6) shows a summation from negative to positive infinity for m, this is unnecessary in practical applications as the multiplication part would yield 0s for regions where the input and impulse response are undefined (equal to 0). For the above numerical convolution example, the convolved signal length is $3 + 2 - 1 = 4$ as expected. Hence, we only need to compute the first 4 samples of $y[n]$ $(n = 0, 1, 2, 3)$.

6 System Diagrams and Digital Building Blocks

We have been introduced to difference equations which consist of addition, multiplication, and delay operators for input and output components. We have also taken the liberty to informally use visual ways of showing signal flow as seen in Figs. 3.2 and 3.3. Although difference equations in the traditional "written-out" format are especially useful and compact for representing "long" difference equation types, an alternate graphical method of presenting the same difference equation is sometimes desirable. This is somewhat analogous to command-line-based software systems opposed to GUI-centric software systems, such as the computer music languages Supercollider vs. Max/MSP for example. The command-line version is often more compact but not necessarily informative (at least upon first inspection) opposed to the GUI version which often gives a better initial overall idea of the system being scrutinized.

As expected, multiplication, addition, and delays can be expressed via block diagrams and are often helpful in visualizing and understanding an LTI system. Each block has an input (or inputs) and output node with its respective unique functionality as shown in Figs. 6.1, 6.2, and 6.3.

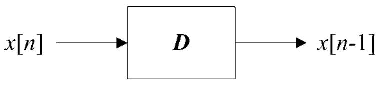


Fig. 6.1. Single delay block.

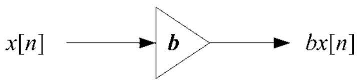


Fig. 6.2. Multiplier (gain) block.

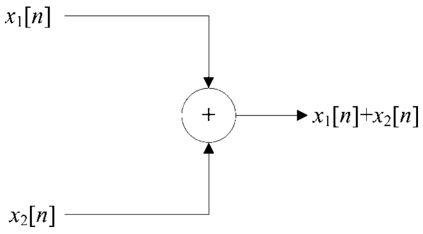


Fig. 6.3. Adder block.

To see how this works in a practical situation, let's take a three-point moving average difference equation and use the system diagram to represent it.

$$y[n] = \frac{x[n] + x[n-1] + x[n-2]}{3} \tag{6.1}$$

Looking at Eq. (6.1), we note that it has two adders, two unit delays, and one multiplier. Thus, the system diagram will look something like Fig. 6.4 where $b_0 = b_1 = b_2 = 1/3$. The system diagram for the general

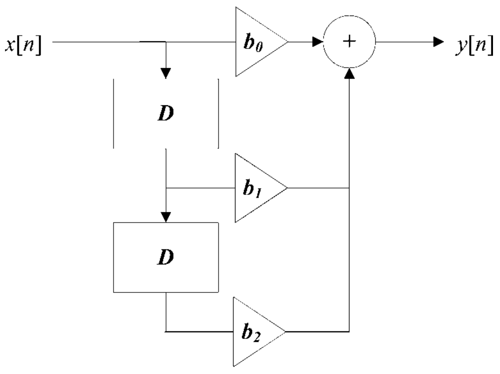


Fig. 6.4. 3-point moving average system diagram.

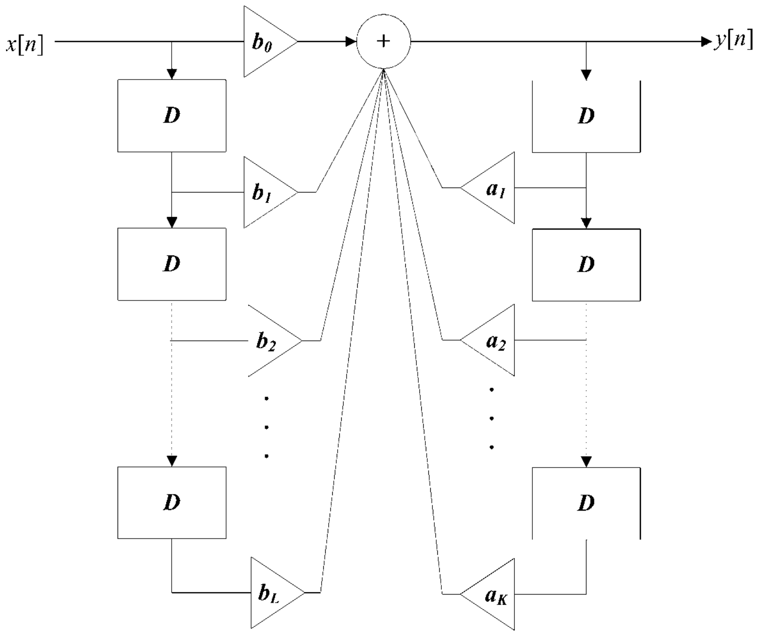


Fig. 6.5. General difference equation system diagram.

difference equation (2.8), which includes the non-recursive and recursive parts, is shown in Fig. 6.5. The left-hand side represents the feedforward components and the right-hand side the feedback components with their respective delays and coefficients.

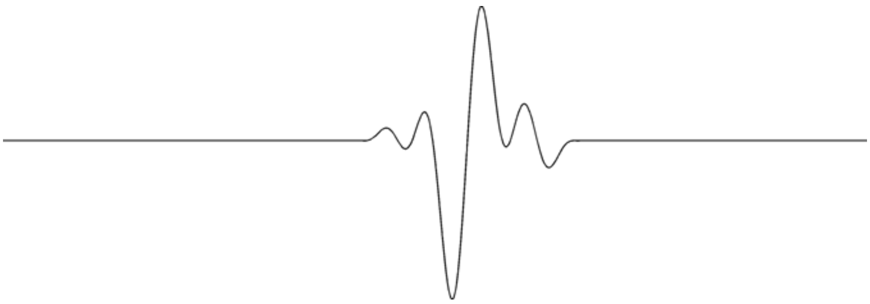
7 Musical Examples

We briefly mentioned the application of convolution in music where it is often used to add reverb to a dry sound. There are numerous commercial software applications and plug-ins which use this method to produce artificial reverberation, where the quality of the reverb is a function of how well the impulse responses were recorded. For example, some software have a single impulse response for the entire concert hall, while others have different impulses taken from various locations within the space, thus more realistically reflecting the reverberation characteristics when changing the reverberation parameters with respect to the stage. However, one can also readily find recordings of impulse responses on the Internet taken from large rooms, jazz dubs, recording studios, buildings, and so on. Convolution is easy enough to implement in software and once you have interesting and clean impulse responses, making your audio clips sound as if they are heard inside a particular space where the impulse response was taken from should be a breeze. Convolution, however, can also be used in other interesting ways, and that is not in the traditional sense of adding reverb, but utilizing non-impulse response-like signals and convolving them together. This can at times result in interesting and sometimes surprising timbres. One such composition is by composer Travis Scharr who recorded a bunch of impulse responses at the Superdome in New Orleans after Hurricane Katrina hit the city. The piece is called *Of Horror and Hope* (2007) and loosely depicts the horrific conditions inside the Superdome offered to some of those who chose to stay in the city during the period of bombardment by the hurricane. This old American city survived a devastating flood with the majority of the city under water topped by indescribable destruction in 2005. In this piece, Mr. Scharr uses convolution to its fullest extent by mainly convolving the piano with storm sounds, the sonic roar of cheering crowds, and Superdome impulse responses. This compositional strategy renders an eerie feeling of a sonic deluge which later in the piece makes way to a *soundscape* depicting the city's rebirth.

There is also an interesting band called the *Convolution Brothers*, which consists of three performers — Miller Puckette, Cort Lippe, and Zack Settel who have a common connection of working at IRCAM (Institut de Recherche et Coordination Acoustique/Musique) in Paris. The Convolution Brothers, as far as I know, do not dwell on convolution-based processes alone but incorporate alternate controllers, a gamut of computer music programming systems, and anything and everything else for that matter. This includes materials that are produced inside and outside the confines of particular software and hardware, and embracing uninhibitive performance practices. Oh yes, humor is a big part of their music! A little trivia before concluding this chapter: Miller Puckette is of course the creator of Max/MSP and Pd (pure data) which are two of the most popular and successful computer music systems that exploit the system diagram approach in controlling and producing/processing/manipulating digitally generated sounds and MIDI.

Chapter 6

FREQUENCY RESPONSE



1 Introduction

In Chap. 6, we will concentrate on the topic surrounding *frequency response.* The frequency response is defined as the characteristics of a system's output in terms of its magnitude and phase response when excited by an input signal. This input signal subjected to the system will have constant amplitude and phase but varying frequency. You may have heard folks talk (and maybe even passionately argue) about loudspeakers and microphones having some sort of frequency response that is this and that — this is exactly what they are referring to: the response of a system. In our example, the loudspeaker is the system and the input some sort of test signal with fixed amplitude and phase but changing frequency. In particular, audiophiles often talk about how flat such responses are. The flatness refers to a loudspeaker's or microphone's performance — how accurately and consistently (add little coloring and distortion) it reproduces/records sounds for a wide frequency range, ultimately determining how truthfully a device reflects the original input source. Unfortunately a perfect system does not exist, and some sort of distortion occurs at some frequencies, thus

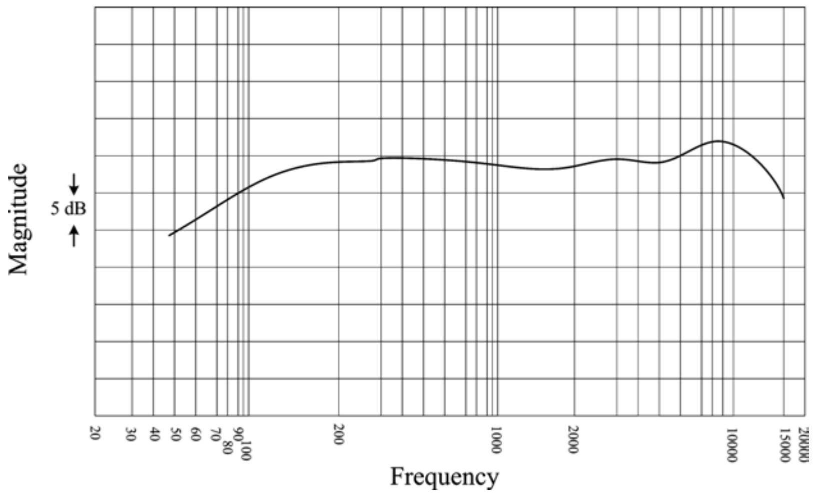


Fig. 1.1. Frequency response plot.

the need for frequency response plots or *Bode* diagrams (log frequency scale opposed linear frequency scale) to assess a particular system's performance.

Fig. 1.1 shows a typical frequency response (magnitude) plot, with the x-axis showing the log-frequency scale and magnitude in dB on the y-axis. Let's read on to find out more about these interesting plots and topics concerning the frequency response.

2 The Frequency Response

To formally define the frequency response, we set the input signal as shown in Eq. (2.1) with unity gain (magnitude of Euler identity is 1), 0 initial phase, and the *digital frequency* θ as shown in Eq. (2.2).

$$x[n] = e^{j\theta n} \tag{2.1}$$

$$\theta = \omega \cdot T = 2 \cdot \pi \cdot f \cdot T \tag{2.2}$$

Equation (2.2) and the digital frequency concept can be better understood if you take the following scenario into consideration. Let's say we are inputting an analog sinusoid $x(t)$ into an ideal sampler as shown below in (2.3).

$$x(t) = A \cdot \cos(\omega t + \phi) \tag{2.3}$$

When sampled, the output of the sampler will be in the form of $x[n \cdot T]$ (or shorthand $x[n]$ as we have learned in Chap. 1) corresponding to the discrete-time version of the analog input $x(t)$ where n is the sample index, T the sampling period in seconds/sample or seconds (as samples are unitless), ω the continuous-time angular frequency in radians/second, and ϕ the initial phase in radians.

$$x[nT] = A \cdot \cos(\omega \cdot nT + \phi) \tag{2.4}$$

Equation (2.4) can be put into a more compact form as show in (2.5), where θ is the digital frequency of the discrete-time cosine and ω the continuous-time angular frequency defined as $2\pi f$. The digital frequency θ has the units radians/sample as $\theta = \omega T = 2\pi \cdot f \cdot T$. That is, $2\pi \cdot f \cdot T$ has the units radians/sample because the units of $2\pi \cdot f \cdot T$ is (radians)(Hz)(second/sample) = (radians)(1/second)(second/sample) = radians/sample.

$$x[n] = A \cdot \cos(\theta \cdot n + \phi) \tag{2.5}$$

Next we convolve the input $x[n]$ with an impulse response $h[m]$ as introduced in the previous chapter and get:

$$\begin{aligned} y[n] &= \sum_{m=-\infty}^{m=\infty} h[m] \cdot x[n-m] = \sum_{m=-\infty}^{m=\infty} h[m] \cdot e^{j\theta(n-m)} \\ &= e^{j\theta n} \cdot \sum_{m=-\infty}^{m=\infty} h[m] \cdot e^{-j\theta m} = x[n] \sum_{m=-\infty}^{m=\infty} h[m] \cdot e^{-j\theta m} \\ &= x[n] \cdot H(e^{j\theta}) \end{aligned} \tag{2.6}$$

The summation part in step 4 of (2.6) is denoted as shown at the end of (2.6) and is defined as the *frequency response*. It also goes by different names, including the *eigenvalue* and *characteristic value*. The frequency response is thus formally defined according to Eq. (2.7).

$$H(e^{j\theta}) = \sum_{m=-\infty}^{m=\infty} h[m] \cdot e^{-j\theta m} \tag{2.7}$$

$$H(e^{j\theta}) = \frac{y[n]}{x[n]} \tag{2.8}$$

We can thus compute a system's frequency response, if we know the system's impulse response as we can see from the relationship offered in

(2.6) and (2.7). Furthermore, observe that $H(e^{j\theta})$ is the output vs. input ratio (2.8). The general difference equation is shown in (2.9) and when the input is set to $x[n] = e^{j\theta n}$ and output $y[n]$ set to Eq. (2.10) [see Eq. (2.8)], the difference equation can be rewritten as shown in (2.11).

$$y[n] = \sum_{k=1}^{k=N} a_k y[n-k] + \sum_{k=0}^{k=L} b_k x[n-k] \tag{2.9}$$

$$y[n] = x[n] \cdot H(e^{j\theta}) = e^{j\theta n} \cdot H(e^{j\theta}) \tag{2.10}$$

$$e^{j\theta n} \cdot H(e^{j\theta}) = \sum_{k=1}^{k=N} \left[a_k e^{j\theta(n-k)} \cdot H(e^{j\theta}) \right] + \sum_{k=0}^{k=L} \left[b_k e^{j\theta(n-k)} \right] \tag{2.11}$$

Rearranging (2.11) we arrive at Eq. (2.12).

$$H(e^{j\theta}) = \frac{\sum_{k=0}^{k=L} b_k e^{-j\theta k}}{1 - \sum_{k=1}^{k=N} a_k e^{-j\theta k}} \tag{2.12}$$

What Eq. (2.12) states is that we can now directly compute the frequency response of a system if we merely know its difference equation, vary the input frequency with unity gain, and set the initial phase to 0. The configuration of Eq. (2.12) basically shows a ratio of the sum of input coefficients divided by the sum of output coefficients. As mentioned at the beginning of this chapter, the frequency response is the observation of the magnitude and phase output when the input frequency is varied while keeping everything else constant. The frequency response's magnitude and phase characteristics can be now computed via Eq. (2.12). When plugging in particular difference equation coefficients along with their corresponding delay characteristics, we will be able to represent the *magnitude response* of the frequency response (2.12) as shown in Eq. (2.13).

$$H(e^{j\theta}) = M \cdot e^{j\Theta(\theta)} \tag{2.13}$$

The magnitude $M(\theta)$, or simply M is:

$$M = \left| H(e^{j\theta}) \right| \tag{2.14}$$

The *phase response* $\Theta(\theta)$ is:

$$\Theta(\theta) = \angle H(e^{j\theta}) = \tan^{-1}\left(\frac{\mathrm{imag}(H(e^{j\theta}))}{\mathrm{real}(H(e^{j\theta}))} \right) \tag{2.15}$$

Figure 1.1 basically shows Eq. (2.14) with the input frequency swept from 20 to 20 kHz.

2.1 *Characteristics and properties of $H(e^{j\theta})$*

There are some interesting and important properties when dealing with the frequency response as summarized in Table 2.1, including properties of periodicity, frequency range of interest, and symmetry. Let us first start by looking at the frequency range of interest.

Table 2.1. $H(e^{j\theta})$ characteristics.

Frequency range of interest	$0 \le \theta < \pi$
Periodicity	2π
Symmetry	Even magnitude Odd phase

2.1.1 *Frequency range and Nyquist revisited*

In Chap. 1, we were first introduced to the Nyquist theorem, sampling frequency, and the notion of aliasing. We stated that aliasing occurred when the condition $f_{max} < f_s/2$ was not met leading to artifacts and distortion in the form of shifting of frequency components above this limit to lower frequency values. In terms of radian frequency, $f_s/2$ is equivalent to π as derived below (2.19), where the digital frequency θ has the usual units of radians/sample, ω radians/second, T seconds/sample, and f_s samples/second.

$$\theta = \omega \cdot T = 2 \cdot \pi \cdot f \cdot T = 2 \cdot \pi \cdot \frac{f}{f_s} \tag{2.16}$$

or

$$\theta = 2 \cdot \pi \cdot \frac{f}{f_s} \rightarrow f = \theta \cdot \frac{f_s}{2 \cdot \pi} \tag{2.17}$$

From the Nyquist limit we know that the frequencies in question need to be bounded by (2.18)

$$0 \le f < \frac{f_s}{2} \tag{2.18}$$

and when substituting f in Eq. (2.18) with Eq. (2.17) we get the Nyquist limit in radians/samples (digital frequency) as shown in (2.19) and finally (2.20) after simplification.

$$0 \leq \theta \cdot \frac{f_s}{2 \cdot \pi} < \frac{f_s}{2} \tag{2.19}$$

$$0 \leq \theta < \pi \tag{2.20}$$

Having the sampling limit in radians will help us greatly in dealing with the frequency response. It will also help show how aliasing occurs via the elegant Euler formula without the need to refer to specific frequency values in Hertz as π now refers to $f_s/2$, whatever f_s may be for a given system. To see how aliasing occurs, consider Fig. 2.1 where we have the mirror symmetry characteristic around the DC (0 frequency). Also, notice that at $\pm\pi$ from DC, there seem to be more symmetrical characteristics observable as well. We know that π refers to the Nyquist limit which means that (at least according to the plot) $|H|$ at radian frequencies $\pi \pm \phi$ should be identical. In other words, any frequency that goes beyond π (such as $\pi + \phi$) should alias back to a lower frequency ($\pi - \phi$). This is exactly what we observed in Chap. 1 Sec. 4.2.1 during aliasing.

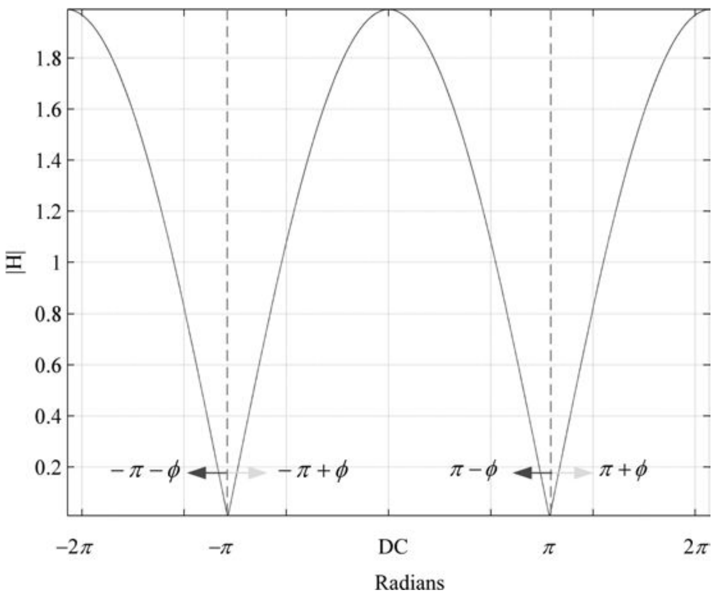


Fig. 2.1. Frequency response from -2π to $+2\pi$.

Of course taking my word for this seemingly plausible phenomenon is not enough, but if we can prove that this phenomenon is actually the case, then we have revealed through frequency response analysis, why and how aliasing occurs. In order to accomplish this proof we need to show (2.21) holds true.

$$\left|H(e^{j(\pi+\phi)})\right| = \left|H(e^{j(\pi-\phi)})\right| \tag{2.21}$$

We start with the definition of frequency response (2.7) by letting $\theta = \pi - \phi$.

$$\begin{aligned} H(e^{j\theta})\big|_{\theta=\pi-\phi} &= \sum_{m=-\infty}^{m=\infty} \big|_{\theta=\pi-\phi} h[m] \cdot e^{-j\theta m} \\ &= \sum_{m=-\infty}^{m=\infty} h[m] \cdot e^{-j(\pi-\phi)\cdot m} \\ &= \sum_{m=-\infty}^{m=\infty} h[m] \cdot e^{-j\pi\cdot m} \cdot e^{j\phi\cdot m} \end{aligned} \tag{2.22}$$

We know from the Euler identity that $e^{j\theta} = \cos\theta + j\sin\theta$ which we will use to expand the exponential component with the π in (2.22). Upon expansion, we note that the sine component disappears as it is 0 for all integer m (at any integer multiples of π the sine function is 0). The cosine component on the other hand, turns into a "polarity function" where it is positive for all even m and negative for all odd m as shown below.

$$e^{-j\pi\cdot m} = \cos(\pi\cdot m) - j\sin(\pi\cdot m) = \cos(\pi\cdot m) - 0 = (-1)^m \tag{2.23}$$

Hence, Eq. (2.22) can be expanded to:

$$\begin{aligned} H(e^{j\theta})\big|_{\theta=\pi-\phi} &= \sum_{m=-\infty}^{m=\infty} h[m] \cdot (-1)^m \cdot e^{j\phi\cdot m} \\ &= \sum_{m=-\infty}^{m=\infty} h[m] \cdot (-1)^m \cdot \{\cos(\phi\cdot m) + j\sin(\phi\cdot m)\} \end{aligned} \tag{2.24}$$

Using the same method for $\theta = \pi + \phi$ we have:

$$\begin{aligned} H(e^{j(\pi+\phi)})\Big|_{\theta=\pi+\phi} &= \sum_{m=-\infty}^{m=\infty} h[m] \cdot e^{-j(\pi+\phi)\cdot m} \\ &= \sum_{m=-\infty}^{m=\infty} h[m] \cdot e^{-j\pi\cdot m} e^{-j\phi\cdot m} \\ &= \sum_{m=-\infty}^{m=\infty} h[m] \cdot (-1)^m \cdot e^{-j\phi\cdot m} \\ &= \sum_{m=-\infty}^{m=\infty} h[m] \cdot (-1)^m \cdot \{\cos(\phi\cdot m) - j\sin(\phi\cdot m)\} \end{aligned} \tag{2.25}$$

Now, apart from the negative sign between the cosine and sine, Eqs. (2.24) and (2.25) are identical. If we let the real components and imaginary components of both equations take the form of $a+jb$, we get a relationships for $H(e^{j(\pi-\phi)})$ and $H(e^{j(\pi+\phi)})$ as shown below:

$$\begin{aligned} H(e^{j(\pi-\phi)}) &= \sum_{m=-\infty}^{m=\infty} h[m] \cdot (-1)^m \cdot \{\cos(\phi\cdot m) + j\sin(\phi\cdot m)\} \\ &= \sum_{m=-\infty}^{m=\infty} h[m] \cdot (-1)^m \cdot \cos(\phi\cdot m) \\ &\quad + j\sum_{m=-\infty}^{m=\infty} h[m] \cdot (-1)^m \cdot \sin(\phi\cdot m) = a + jb \end{aligned} \tag{2.26}$$

$$\begin{aligned} a &= \sum_{m=-\infty}^{m=\infty} h[m] \cdot (-1)^m \cdot \cos(\phi\cdot m) \\ b &= \sum_{m=-\infty}^{m=\infty} h[m] \cdot (-1)^m \cdot \sin(\phi\cdot m) \end{aligned} \tag{2.27}$$

Using the same approach for $H(e^{j(\pi+\phi)})$ we get:

$$H(e^{j(\pi+\phi)}) = \sum_{m=-\infty}^{m=\infty} h[m] \cdot (-1)^m \cdot \{\cos(\phi\cdot m) + j\sin(\phi\cdot m)\}$$

$$= \sum_{m=-\infty}^{m=\infty} h[m] \cdot (-1)^m \cdot \cos(\phi \cdot m)$$

$$- j \sum_{m=-\infty}^{m=\infty} h[m] \cdot (-1)^m \cdot \sin(\phi \cdot m) = a - jb \qquad (2.28)$$

Finally, since we know that the magnitude of a complex vector is computed as the square root of sum of the squares as shown in Eq. (2.29)

$$|H| = \sqrt{real^2 + imaginary^2} \qquad (2.29)$$

we find that the magnitude response of frequency response for $\pi - \phi$ and $\pi + \phi$ is:

$$\left|H(e^{j(\pi-\phi)})\right| = \left|H(e^{j(\pi+\phi)})\right| = \sqrt{a^2 + b^2} \qquad (2.30)$$

Hence, as far as the magnitude $|H(e^{j\theta})|$ is concerned, $\theta = \pi - \phi$ and $\theta = \pi + \phi$ are exactly the same; and when taking the polarity of the real and imaginary components into account (rectangular coordinate system), a reflection about the real axis occurs as shown Fig. 2.2. Looking at it from the frequency point of view, what our proof says is that anything that

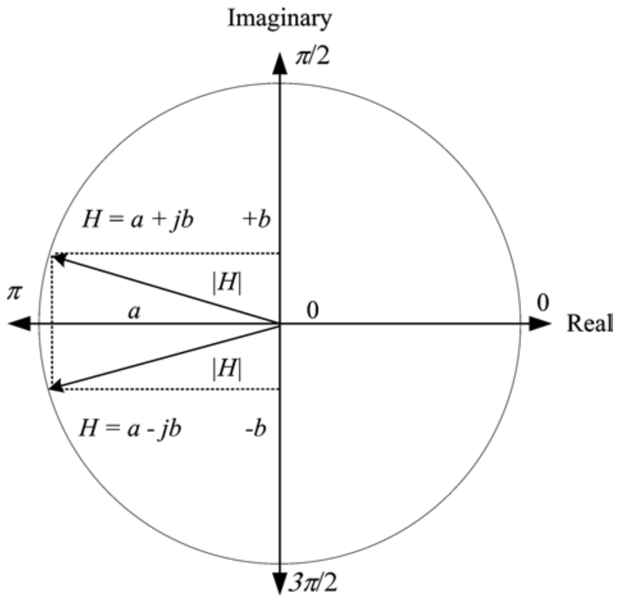


Fig. 2.2. Magnitude in rectangular coordinate system (note polarity).

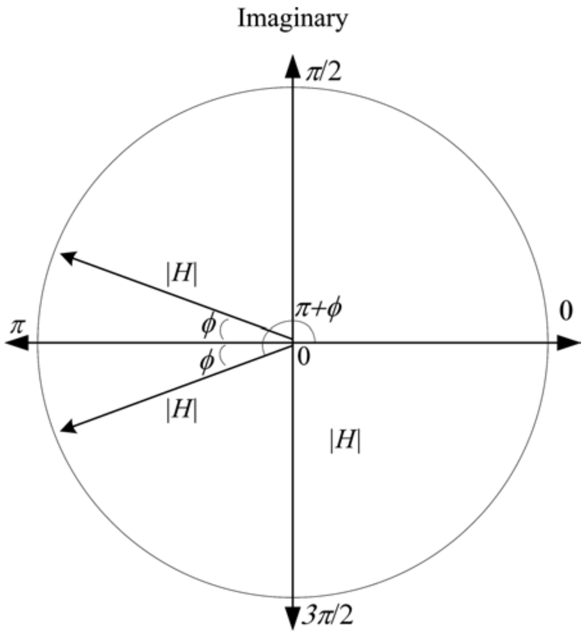


Fig. 2.3. Frequency (in radians) for $\theta = \pi + \phi$ and $\theta = \pi - \phi$.

goes beyond π (negative imaginary side of y-axis) will be reflected back to positive imaginary side as shown in Fig. 2.3. This is aliasing.

We will "re-revisit" aliasing and the Nyquist limit in Chap. 8 after introduction of the Fourier transform where will see how it can be viewed from yet another perspective!

2.1.2 *$H(e^{j\theta})$ and periodicity property*

We have already seen some of the periodic properties for $H(e^{j\theta})$ in Figs. 2.2 and 2.3 and we can probably guess the source of this repetitive characteristic — due to the cosine and sine properties of the Euler formula which are periodic for all integer values k corresponding to $2 \cdot \pi \cdot k$. Thus, if we can show that (2.31) holds true, then we have proven periodicity for the frequency response.

$$H(e^{j\theta}) = H(e^{j(\theta + k\cdot\pi)}) \tag{2.31}$$

We start with the right-hand side of Eq. (2.31).

$$\begin{aligned} H(e^{j(\theta+k\cdot\pi)}) &= \sum_{m=-\infty}^{m=+\infty} h[m]\cdot e^{-j(\theta+k\cdot 2\cdot\pi)} \\ &= \sum_{m=-\infty}^{m=+\infty} h[m]\cdot e^{-j\theta}e^{-jk\cdot 2\cdot\pi} \end{aligned} \tag{2.32}$$

Again, with Euler coming to the rescue we see that for all integer values k, the rightmost exponential part of Eq. (2.32) becomes unity as shown in Eq. (2.33).

$$e^{-jk\cdot 2\cdot\cdot\pi} = \cos(k\cdot 2\cdot\pi) - j\sin(k\cdot 2\cdot\pi) = 1 - 0 = 1 \tag{2.33}$$

Thus, Eq. (2.32) simply is reduced to Eq. (2.34) and observe that the attribute of periodicity described in Eq. (2.31) indeed holds true.

$$\begin{aligned} H(e^{j(\theta+k\cdot\pi)}) &= \sum_{m=-\infty}^{m=+\infty} h[m]\cdot e^{-j\theta\cdot m}e^{-jk\cdot 2\cdot\cdot\pi\cdot m} = \sum_{m=-\infty}^{m=+\infty} h[m]\cdot e^{-j\theta\cdot m}\cdot 1 \\ &= \sum_{m=-\infty}^{m=+\infty} h[m]\cdot e^{-j\theta\cdot m} \end{aligned} \tag{2.34}$$

2.1.3 *Symmetry*

Referring back to Fig. 2.1 (also look at Fig. 2.2 and 2.3), we see that the magnitude is symmetrical about the real axis resembling a cosine function — if we look at a cosine function as a time sequence (with zero initial phase) and increase or decrease the frequency by plus or minus ϕ at $x = 0$ (time axis), the magnitude will be the same (even symmetry) about the $x = 0$ axis, while the sine will exhibit odd symmetry. In order to show that this exact mirror image actually takes place all we have to do is show that:

$$\left|H(e^{j\theta})\right| = \left|H(e^{-j\theta})\right| \tag{2.35}$$

By now, we have hopefully become comfortable with the Euler formula and can quickly prove Eq. (2.35).

$$\begin{aligned} H(e^{j\theta}) &= \sum_{m=-\infty}^{m=+\infty} h[m] \cdot e^{-j\theta \cdot m} \\ &= \sum_{m=-\infty}^{m=+\infty} h[m] \cdot \{\cos(\theta \cdot m) - j\sin(\theta \cdot m)\} = c - jd \end{aligned} \tag{2.36}$$

$$\begin{aligned} H(e^{-j\theta}) &= \sum_{m=-\infty}^{m=+\infty} h[m] \cdot e^{j\theta \cdot m} \\ &= \sum_{m=-\infty}^{m=+\infty} h[m] \cdot \{\cos(\theta \cdot m) + j\sin(\theta \cdot m)\} = c + jd \end{aligned} \tag{2.37}$$

Once again we see that the magnitudes of both frequency responses are the same where c and d are real number constants for a given θ as shown below in (2.38).

$$\left|H(e^{j\theta})\right| = \left|H(e^{-j\theta})\right| = \sqrt{c^2 + d^2} \tag{2.38}$$

On the topic of symmetry in phase response, we can clearly see by looking at Fig. 2.2 that it exhibits odd symmetry. That is, $\tan^{-1}(H) = -\tan^{-1}(H)$.

$$\angle H(e^{j\theta}) = \tan^{-1}\left(\frac{+b}{a}\right) \tag{2.39}$$

$$\angle H(e^{-j\theta}) = \tan^{-1}\left(\frac{-b}{a}\right) \tag{2.40}$$

$$\angle H(e^{j\theta}) = -\angle H(e^{-j\theta}) \tag{2.41}$$

2.2 *More stuff on the frequency response*

There are two common ways to view and plot the frequency response of a system — using linear magnitudes and log magnitudes, log magnitude being one of the more standard ones. One of the reasons that the log dB version is the more standard way of representing frequency response plots

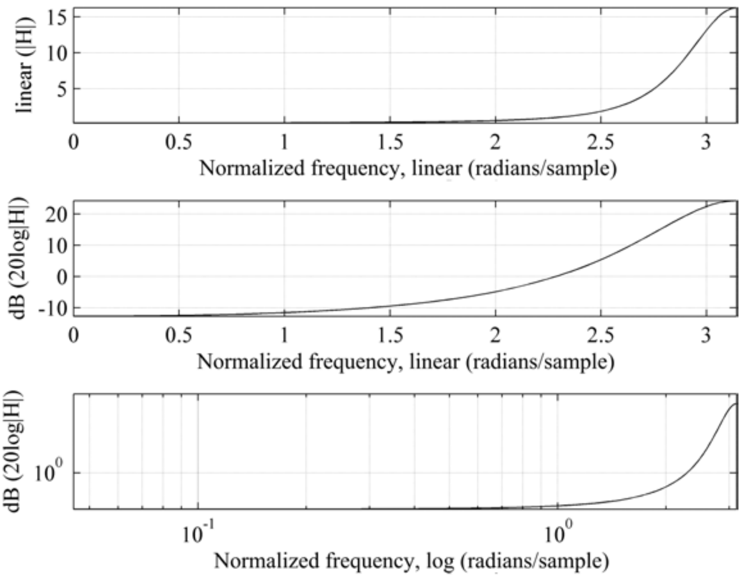


Fig. 2.4. Frequency response plot linear and logarithmic Bode plots.

is a practical one.

$$\left|H(e^{j\theta})\right|_{\mathrm{dB}} = 20 \cdot \log_{10}\left(\left|H(e^{j\theta})\right|\right) \tag{2.42}$$

As seen in Fig. 2.4, the top plot is a linear plot of the magnitude response and the bottom two plots dB magnitude plots. It is clear that the linear plot does not show the details pertinent to the behavior of the lower magnitude values (notice that the linear version's y-axis range is from around 0 to 15 whereas the dB plot spans from -10 to $+20$). The dB version does a better job in presenting the subtleties and intricacies found in the less peaky and lower magnitude areas which are also important. Another reason for the choice of dB is of course due to the way we hear the dynamics of sound which is logarithmic rather than linear. Furthermore, as mentioned at the beginning of Sec. 1, Bode diagrams are also often used when presenting frequency responses by making the frequency axis a logarithmic scale as well. Once more, the reason for that is practical as well, as we hear frequencies logarithmically — Bode diagrams better represent how humans perceive frequencies. Also, very important is to note that the x-axis is presented in the normalized frequency (π x radians/sample) format corresponding to the digital frequency $\theta = \omega T$. It is referred to the normalized frequency as a 0 is equivalent to 0 radians/sample (DC) and π radians/sample to the Nyquist frequency ($f_s/2$).

Let's finish off this section with an example and put our newly learned concepts into practice by computing the frequency response for a simple feedback IIR system as shown in Eq. (2.43).

$$y[n] = x[n] + a \cdot y[n-1] \tag{2.43}$$

Using the difference equation method for computing the frequency response Eq. (2.12) and noting that there is one feedback component with weight a we get:

$$H(e^{j\theta}) = \frac{1}{1 - a \cdot e^{-j\theta}} \tag{2.44}$$

Next, since all the action is in the denominator part we assign it dummy variables to make things seem less cluttered. Thus, for a given θ, we get the relationship shown in Eqs. (2.45) and (2.46) and compute $|H(e^{j\theta})|$.

$$1 - a \cdot e^{j\theta} = 1 - a \cdot (\cos\theta - j\sin\theta) = A + jB \tag{2.45}$$

where

$$A = 1 - a \cdot \cos\theta, \quad B = a \cdot \sin\theta \tag{2.46}$$

$$\begin{aligned} |H(e^{j\theta})| &= \left|\frac{1}{A+jB}\right| = \left|\frac{1}{(A+jB)} \cdot \frac{(A-jB)}{(A-jB)}\right| \\ &= \left|\frac{A-jB}{A^2-B^2}\right| = \left|(C-jD)\right| = \sqrt{C^2+D^2} \end{aligned} \tag{2.47}$$

$$C = \frac{A}{A^2-B^2} \tag{2.48}$$

$$D = \frac{B}{A^2-B^2} \tag{2.49}$$

Note that in Eq. (2.47) we employ the $(A-jB)/(A-jB)$ trick to get rid of the awkward imaginary component in the denominator part. When we put the frequency response in its alternate polar form, we arrive at the result shown below:

$$H(e^{j\theta}) = Me^{j\Theta(\theta)} \tag{2.47}$$

where

$$\sqrt{C^2 + D^2} = M \tag{2.48}$$

$$\Theta(\theta) = \tan^{-1}\left(\frac{D}{C}\right) \tag{2.49}$$

All of the above could be easily coded into a programming language such as MATLAB® to make our lives easier once we have figured out the formula as we did above. Figure 2.5 shows the frequency response plot of the above difference equation with $a = 0.5$, linear and dB magnitudes, and frequency in radians/sample.

This type of filter is referred to as a low-pass filter as it passes low frequency regions and rejects high frequency regions. Filters are on the menu in our next chapter.

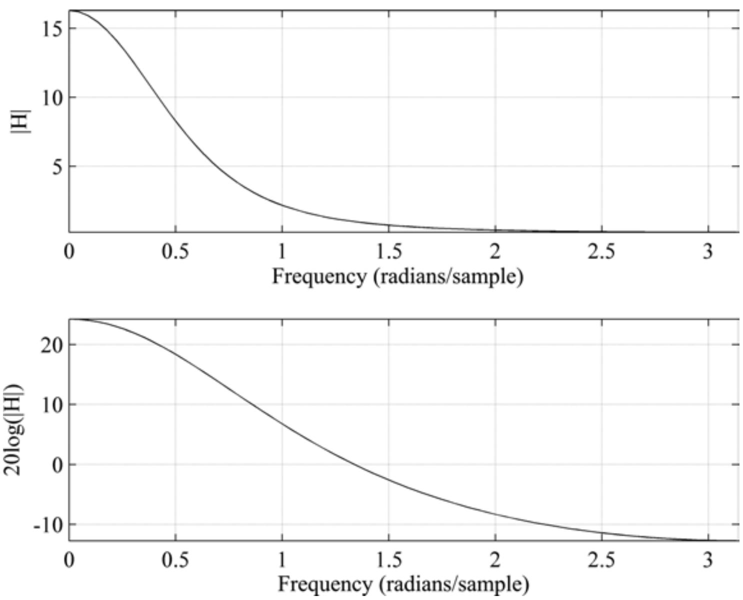


Fig. 2.5. Frequency response of $y[n] = x[n] + 0.5 \cdot y[n-1]$. Linear magnitude (top), dB magnitude (bottom).

3 Phase Response and Phase Distortion

In Sec. 2, we defined the phase response $\Theta(\theta)$ of a system as the arctangent of the imaginary part divided by the real part of the transfer function $H(e^{j\theta})$. As shown in Eq. (3.1), the phase response corresponds to the phase characteristics of a system when it is subjected to an input with unity gain, 0 initial phase, and changing frequency.

$$\Theta(\theta) = \angle H(e^{j\theta}) = \tan^{-1}\left(\frac{imag(H(e^{j\theta}))}{real(H(e^{j\theta}))}\right) \tag{3.1}$$

We also introduced the concept of initial and instantaneous phase in Chap. 1, Sec. 2 and Chap. 4, Sec. 2 and characterized the initial phase as the phase *offset* observed at $t = 0$ and the instantaneous phase the time-variant version.

The phase response as seen in (3.1) basically denotes this concept of offset of a signal that changes with the input frequency. That is, if we inputted a cosine signal into a system $H(\cdot)$, we would get an output that will not only be altered in its magnitude response but also its phase response — phase will behave differently depending on the frequency of the cosine signal. Another perhaps more intuitive way of looking at phase and the phase response, is regarding phase from a time delay perspective that occurs when an input signal passes through a system $H(\cdot)$. An example of such a system could be your guitar (input signal) plugged into your computer via an audio interface (let's pretend it has 0 latency and does nothing to the signal) and a software application (the system H) adding a distortion effect. Ideally, you would want the output to only have the distortion effect à la Jimmy Hendrix, without any delay. That is, if the delay is large enough due to whatever reason in the software, say, around one second or so, it would nearly be impossible to play the guitar in real-time with your band, as cool as it might look like having a computer on stage. However, if the delay is short enough you will indeed be able to synchronize easily with the rest of the musicians and yourself included of course. A more interesting scenario is when the phase is different depending on what note one plays — more delay for higher notes and less delay for lower pitched notes for example. This "propagation time" in the form of delay through the software is very closely linked to the phase response which we will elaborate on in this section.

Figure 3.1, shows the phase response as well as the magnitude response for the simple difference equation shown in (3.2) with one delay component

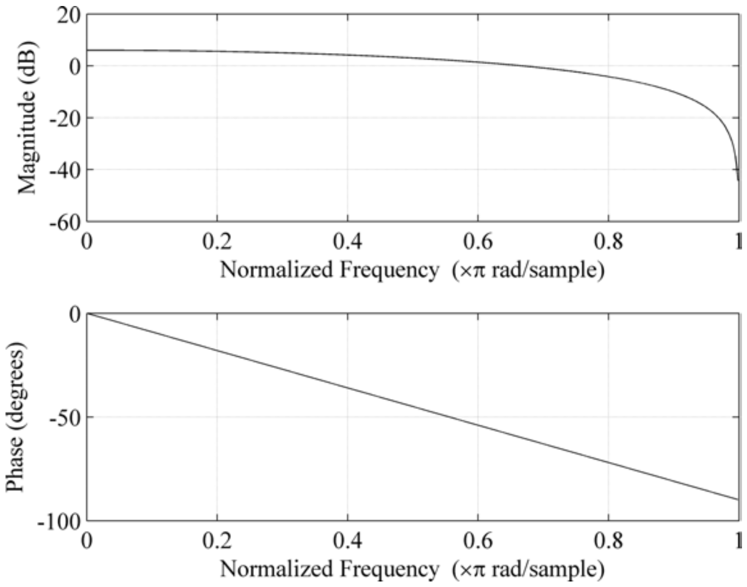


Fig. 3.1. Magnitude and phase response of $y[n] = x[n] + x[n-1]$.

and no feedback components — the simplest FIR low-pass filter.

$$y[n] = x[n] + x[n-1] \tag{3.2}$$

Much like the moving average filter, if we divided the sum of input components by 2 ($x[n]/2 + x[n-1]/2$) or multiplied the sum of $x[n]$ and $x[n-1]$ by 0.5, we would have a simple two-point arithmetic mean difference equation. As mentioned above, this filter, like the IIR difference equation of our previous example, is a low-pass filter — all averaging filters are low-pass filters actually. We will delve into filters in the next chapter where we introduce the most common filter types and some interesting applications especially found in music.

To analyze this system, let's first put the difference equation into the frequency response format using Eq. (2.12). This results in Eq. (3.3) below.

$$H(e^{j\theta}) = 1 + e^{-j\theta} = 1 + \cos\theta - j\sin\theta \tag{3.3}$$

We will recognize in Eq. (3.3) that it has real and imaginary parts as depicted in Fig. 3.2 allowing us to use Eq. (3.1) to compute the phase

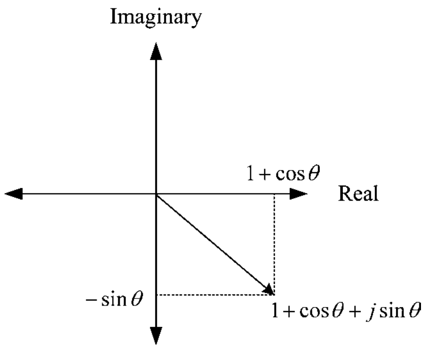


Fig. 3.2. Complex plot of $H(e^{j\theta})$.

response.

$$\Theta(\theta) = \tan^{-1}\left(\frac{imag(H(e^{j\theta}))}{real(H(e^{j\theta}))}\right)$$
$$= \tan^{-1}\left(\frac{-\sin\theta}{(1+\cos\theta)}\right) \tag{3.4}$$

Exploiting two basic trigonometric identities shown in (3.5) and (3.6) we can simplify the phase response to (3.7).

$$\cos(2\cdot\theta) = 2\cdot\cos^2\theta - 1 \longleftrightarrow \cos(\theta) = 2\cdot\cos^2\left(\frac{\theta}{2}\right) - 1 \tag{3.5}$$

$$\sin(2\cdot\theta) = 2\cdot\sin\theta\cdot\cos\theta \longleftrightarrow \sin(\theta) = 2\cdot\sin\left(\frac{\theta}{2}\right)\cdot\cos\left(\frac{\theta}{2}\right) \tag{3.6}$$

$$\Theta(\theta) = \tan^{-1}\left(\frac{-\sin\theta}{1+\cos\theta}\right) = \tan^{-1}\left(\frac{-\sin(\theta/2)\cdot(2\cdot\cos(\theta/2))}{2\cdot\cos^2(\theta/2)}\right)$$
$$= \tan^{-1}\left(\frac{-\sin(\theta/2)}{\cos(\theta/2)}\right) = \tan^{-1}\left(\tan(-\theta/2)\right) = -\theta/2 \tag{3.7}$$

From Eq. (2.2), we know that $\theta = \omega T$ and hence (3.7) can be rewritten in terms of the continuous-time angular frequency ω in radians which in turn is defined as $2\pi f$:

$$\Theta(\theta) = -\frac{\theta}{2} = -\frac{\omega\cdot T}{2} \tag{3.8}$$

Looking again at Fig. 3.1, we see that the phase response is linear, meaning that the phase delay (in samples or seconds) is constant (phase delay is the subject matter in the next section). In other words, the output will simply be a non-time-distorted waveform version of itself on the output side, where the output is a delayed version of the input with high frequency components attenuated. This linear phase characteristic may not be as obvious as it could/should be if we simply look at the bottom of Fig. 3.1. However, if you view the x axis unit as radians/sample, you will note that it actually represents how many radians a particular frequency will have "traveled" in terms of "radian distance" over one sample period. For example, a 1 Hz signal would need a lot of samples (exactly f_s number of samples) to make one complete cycle or stated inversely, a 1 Hz signal will not travel much in one sample time. The inverse is true for high frequencies — higher frequencies will make more resolutions per second and hence move along the phase axis further than a lower frequency component. This explains the larger phase degrees values for the higher frequencies and lower phase degree values for the lower frequencies at the bottom of Fig. 3.1. The y-axis can also be represented in radians rather than degrees. The simple formula to jump back and forth from degrees to radians is:

$$\text{degrees} = \text{the\_radian} \cdot \frac{180}{\pi} \tag{3.9}$$

The important point and characteristic in this example is that this particular system exhibits a linear phase response. We will come back to the linearity aspect of phase response shortly but let's first take a look at the phase delay in the next section which can make viewing phase a little bit more intuitive.

3.1 *Phase delay*

It is sometimes more useful and intuitive to view the phase response in terms of sample delay or time delay, rather than viewing it as in Fig. 3.1 via phase degrees or radians. Much like one can regard a time-offsetted cosine wave by $\pi/2$ as a sine wave (for a 1 Hz sine and cosine wave the $\pi/2$ radians would be equivalent to quarter of a second), the phase response may also be presented in terms of samples/time rather than phase angles or radians — this is known as the *phase delay.* Thus, for phase delay, the goal is to make the resulting phase output in units of samples (or time if multiplied by $T = 1/f_s$) and that is exactly what Eq. (3.10) accomplishes.

$$\tau_P(\theta) = -\frac{\Theta(\theta)}{\theta} \tag{3.10}$$

Hence, for the simple FIR low-pass difference equation (3.2), the phase delay becomes a half sample time delay as shown in Eq. (3.11) and Fig. 3.3.

$$\tau_P(\theta) = -\frac{\Theta(\theta)}{\theta} = -\frac{-\frac{\theta}{2}}{\theta} = \frac{1}{2} \tag{3.11}$$

We can say that for the FIR system in our example, whenever an input signal is subjected to this particular transfer function $H(\cdot)$ or difference equation, one must expect a constant 1/2 sample delay, regardless of the input frequency. If we want to represent the phase delay in terms of time instead of samples, we simply multiply it by the sampling interval $T(= 1/f_s)$

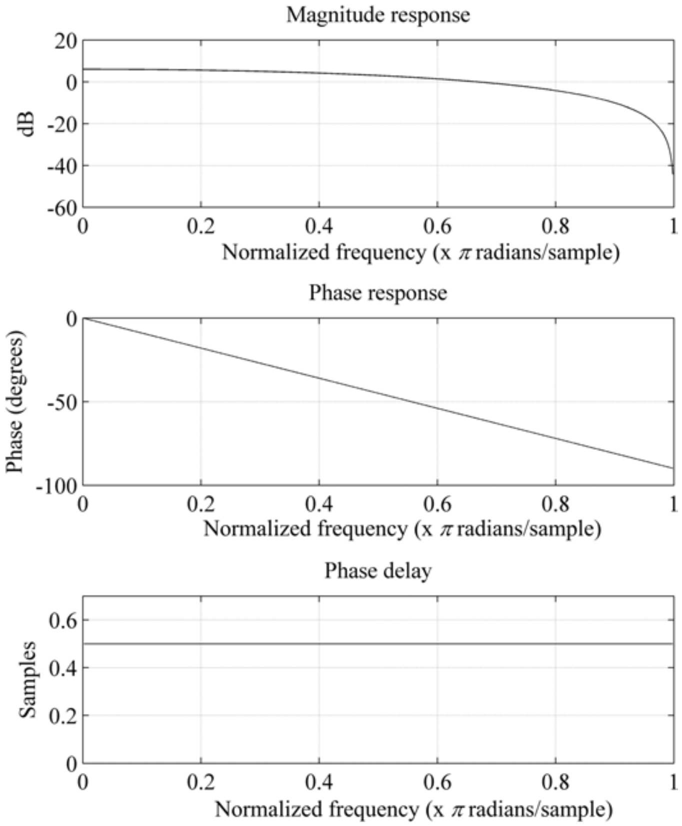


Fig. 3.3. Plots of magnitude/phase response and phase delay for $y[n] = x[n] + x[n-1]$. Top plot shows the magnitude response, the middle plot the phase response, and the bottom the phase delay in samples.

as shown in Eq. (3.12). For a system with $f_s = 8\,\text{kHz}$ or $T = 1/8000 = 0.125\,\text{ms}$, the phase delay amount would be a constant $\tau_p = 0.000125/2 = 0.0625\,\text{ms}$ across the board which is not too bad!

$$t_p(\theta) = \tau_P(\theta) \cdot T = \tau_P(\theta) \cdot \frac{1}{f_s} \tag{3.12}$$

One way to analyze how we arrived at the definition of phase delay is by equating a sine function with initial phase offset ϕ (in radians) and a sine function with phase delay t_p (in seconds) as in Eq. (3.13) (adapted from Claerbout 1998).

$$\sin(\omega t - \phi) = \sin \omega(t - t_p) \tag{3.13}$$

The inside part of the right-hand side parentheses of Eq. (3.13) can be further simplified to Eq. (3.14) and solving for t_p we get Eq. (3.15).

$$\omega t - \phi = \omega(t - t_p) = \omega t - \omega t_p \tag{3.14}$$

$$t_P = \frac{\phi}{\omega} \tag{3.15}$$

Comparing Eq. (3.10) to Eq. (3.15) we can clearly see that one is the continuous-time version (seconds and continuous angular frequency) and the other the discrete-time (samples and digital angular frequency) version with an additional negative polarity for the discrete-time phase delay.

$$t_P(\omega) = +\frac{\phi}{\omega} \longleftrightarrow \tau_P(\theta) = -\frac{\Theta(\theta)}{\theta} \tag{3.16}$$

3.2 *Linearity and phase*

Systems that have linear phase responses output phase delay characteristics that are constant over the whole frequency range. The simple FIR example from above is such a filter, with a constant phase response of half a sample. We also figured out what the phase delay is and noticed that it helps us observe the phase characteristics in perhaps a more intuitive way via sample/time delay, opposed to degrees or radians.

However, maybe unsurprisingly, some transfer functions do not exhibit linear phase responses. That is, different amount of phase influence is exerted on the input signal, depending on the frequency of the input. Nonlinear phase is not a highly desirable feature, especially when the phase characteristics vary substantially with frequency. One way to better understand why linearity in phase is important is to consider the following extreme example. Let's say we have an audio CD that has a music track consisting of electric bass, piccolo flute, and a 64-note electric piano where the electric bass plays notes below 100 Hz, piano between 100 and 600 Hz, and the piccolo flute above 600 Hz only. When we play this track on a CD player we would expect the three instruments to play in time as it was performed and recorded at the studio (assuming that these guys played it "perfectly" both in terms of timing and pitches — there is actually a fusion jazz band named *Casiopea* which has an album called *Perfect Live* from the 1980s, and these guys really do sound like MIDI sequenced machines, although they play all their instruments "live" while dancing in their 80s Ace Frehley-like costumes: check out *Street Performer*). However, our homebuilt CD player happens to be a bit funky and has a very interesting phase response — frequencies below 100 Hz exhibit a phase delay of 2 seconds, 100 to 600 Hz no delay, and 600 Hz and above a delay of 1 second. The sound of the trio ensemble that we would hear via this particular CD player would be quite bizarre (albeit maybe interesting) — the three instruments would appear not to be playing together at all, but rather out of synchrony as shown in Fig. 3.4.

This is what happens with nonlinear phase response-based systems resulting in often undesirable time-domain signal distortion. In other words, if we subject an input such as the above to a nonlinear phase system, the output signal's shape will be altered as each input frequency component is subject to different delay amounts. Fig. 3.5 shows the nonlinear phase

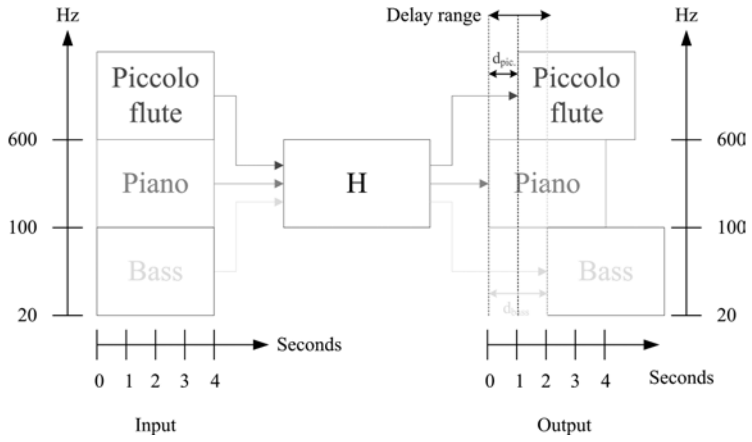


Fig. 3.4. Trio band of bass, piano, and flute with CD player (H) having different (nonlinear) phase delay.

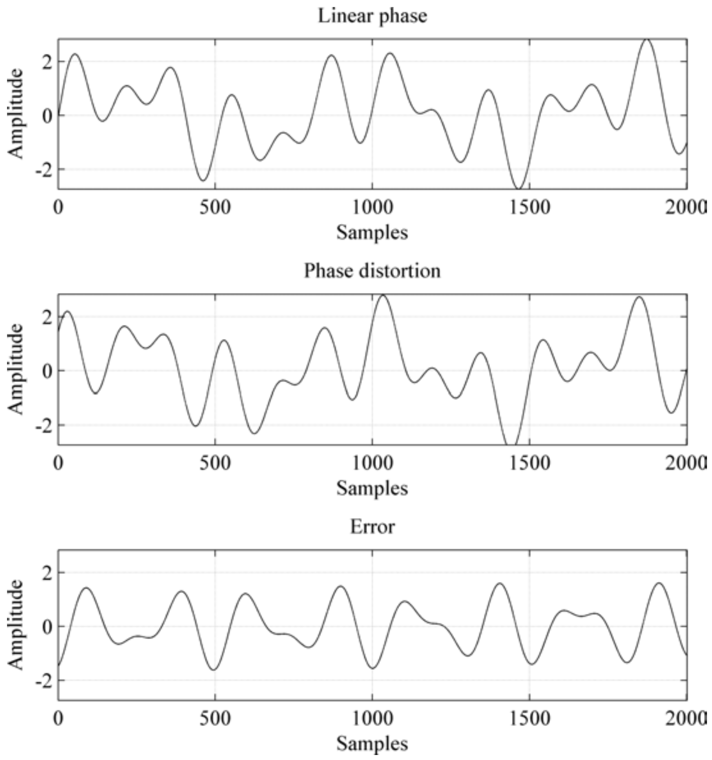


Fig. 3.5. Linear phase (top), nonlinear phase (middle), error (bottom).

characteristics of a signal where the top plot has linear phase, middle plot nonlinear phase, and the bottom plot the error between the top and middle plots corresponding to the amount of temporal smearing. As nonlinear phase distortion alters the signal in a subtle way, nonlinear phase equalizers are usually used in the mastering process to add as little modification to the resulting signal as possible. One of the methods for analyzing nonlinear phase is through the so-called *group delay*, the topic in Sec. 3.3 and 3.4.

3.3 *Phase response and continuous phase*

When computing the phase response, we will sometimes come across plots that seem to abruptly change along the frequency axis, jumping to a distant phase value as seen in Fig. 3.6. If we look carefully at the phase response, we will note that if we offset the latter part (at around 0.45π radians) of the phase response by $-\pi$ radians, we will be able to make the phase response

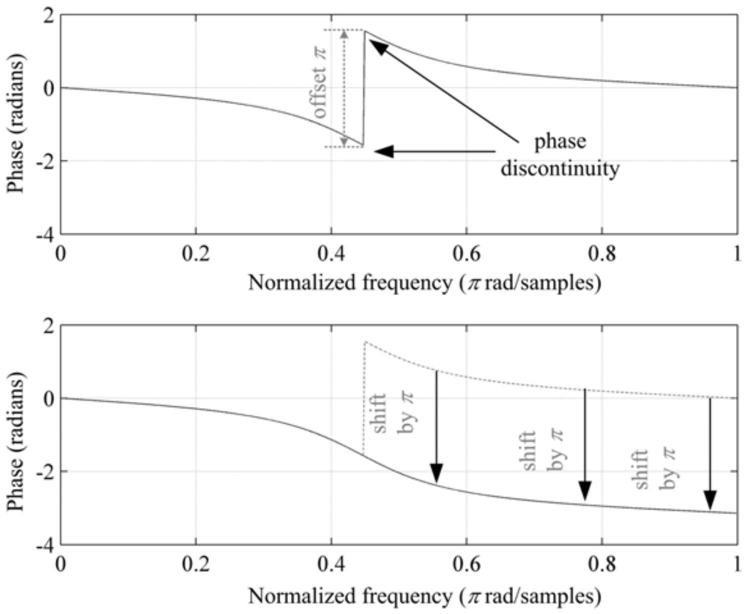


Fig. 3.6. Top is discontinuous phase, bottom shows phase shift by π in our example.

become a smooth function of frequency. This type of offset addition is sometimes necessary for obtaining the phase response. This special process for smoothing out phases that "overshoot" the 2π margin to make the response smooth is referred to as *phase unwrapping.*

This type of jump is due to the periodicity and modulo characteristics of sines and cosines. Whenever the phase surpasses the 2π radian point a resetting of the phase to 0 occurs — modulo of 2π (360 degree). This is much like the modulo 12 system found in watches, in the twelve-tone equal-temperament tuning system, and in the months per year.

3.4 *Group delay*

Group delay (also referred to as differential time delay corresponding to group velocity in wave-propagation theory) is a concept useful for analyzing phase responses that are nonlinear. That is, useful for phase responses that are dependent on the input frequency as discussed in the previous section leading to temporal smearing and phase distortion. For example, group delay is an issue in the area of loudspeaker design as it is inevitably added to the signal being reproduced. On top of the actual measurable phase delays for different frequencies, as we do not hear all frequencies equally well, different thresholds for group delay exists as well. Hence, a high quality sound reproduction system team will design and tweak its group delay characteristics (among many other things) by considering the threshold of perceptibility of group delays. An interesting aspect of group delay distortion and its perceptibility is that not only does it depend on threshold values that are a function of frequency (group-delay curve), but also the type of signal and state of training (subject's familiarity with certain kinds of sounds) of the listeners (Blauert and Laws 1978).

Let's now define group delay formally and see how it is represented mathematically. The group delay is defined in Eq. (3.17) as the derivative of phase response with respect to the digital frequency with units in samples.

$$\tau_g(\theta) = -\frac{d\Theta(\theta)}{d\theta} \tag{3.17}$$

If the phase response output is linear or in the form of Eq. (3.18) (as in our FIR $y[n] = x[n] + x[n-1]$ example), the group delay and phase delay become the same.

$$\Theta(\theta) = \alpha \cdot \theta \tag{3.18}$$

The term group delay can be a little bit misleading as it is not the average delay of a system, but rather, provides insights about the nonlinearity characteristics in terms of the slope values computed from the phase response. The basic group delay concept can be somewhat better understood by considering a pair of sinusoids as the input to an unknown system $H(\cdot)$ as shown in Eq. (3.19) and Fig. 3.7 (adapted from Claerbout 1998).

$$x(t) = \cos(\omega_1 t) + \cos(\omega_2 t) \tag{3.19}$$

If the output of the system $y(t)$ results in no change with respect to the frequency or amplitude components of the two input sinusoids, but adds different amount of phase delay (as observed at the output), the output

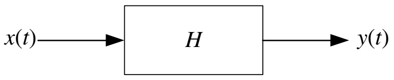


Fig. 3.7. Input and output of system H.

$y(t)$ will resemble something like Eq. (3.20).

$$y(t) = \cos(\omega_1 t - \phi_1) + \cos(\omega_2 t - \phi_2) \tag{3.20}$$

We know from Chap. 4, Sec. 4 that the trigonometric identify for Eq. (3.19) results in Eq. (3.21) (sum and difference) where the difference frequency generally refers to the beating we hear, which in turn corresponds to the amplitude envelope of a signal. Now, if we use the same trigonometric identity to get Eq. (3.20) into the "envelope multiplying the signal" format, Eq. (3.22) will yield.

$$x(t) = 2 \cdot \cos\left(\frac{\omega_1 - \omega_2}{2} t\right) \cdot \cos\left(\frac{\omega_2 + \omega_1}{2} t\right) \tag{3.21}$$

$$y(t) = 2 \cdot \cos\left(\frac{\omega_1 - \omega_2}{2} t - \frac{\phi_1 - \phi_2}{2}\right) \cdot \cos\left(\frac{\omega_2 + \omega_1}{2} t - \frac{\phi_2 + \phi_2}{2}\right) \tag{3.22}$$

Let's look at the amplitude envelope part of Eq. (3.22) (beating component: difference frequency) and rewrite it in terms of time delay as well as phase offset as shown below. We do this by following the same method we used in arriving at the phase delay by starting with Eq. (3.23). The left-hand side of Eq. (3.23) shows the envelope part from Eq. (3.22) in terms of phase offset in radians and the right-hand side shows the equivalent time offset t_g.

$$\cos\left(\frac{\omega_1 - \omega_2}{2} t - \frac{\phi_1 - \phi_2}{2}\right) = \cos\left(\frac{\omega_1 - \omega_2}{2}(t - t_g)\right) \tag{3.23}$$

Comparing the variable parts within the parantheses only we have the following relationships:

$$\left(\frac{\omega_1 - \omega_2}{2} t - \frac{\phi_1 - \phi_2}{2}\right) = \left(\frac{\omega_1 - \omega_2}{2} t - \frac{\omega_1 - \omega_2}{2} t_g\right) \tag{3.24}$$

$$-\frac{\phi_1 - \phi_2}{2} = -\frac{\omega_1 - \omega_2}{2} t_g \tag{3.25}$$

$$\phi_1 - \phi_2 = (\omega_1 - \omega_2) t_g \tag{3.26}$$

$$\frac{\phi_1 - \phi_2}{\omega_1 - \omega_2} = t_g \tag{3.27}$$

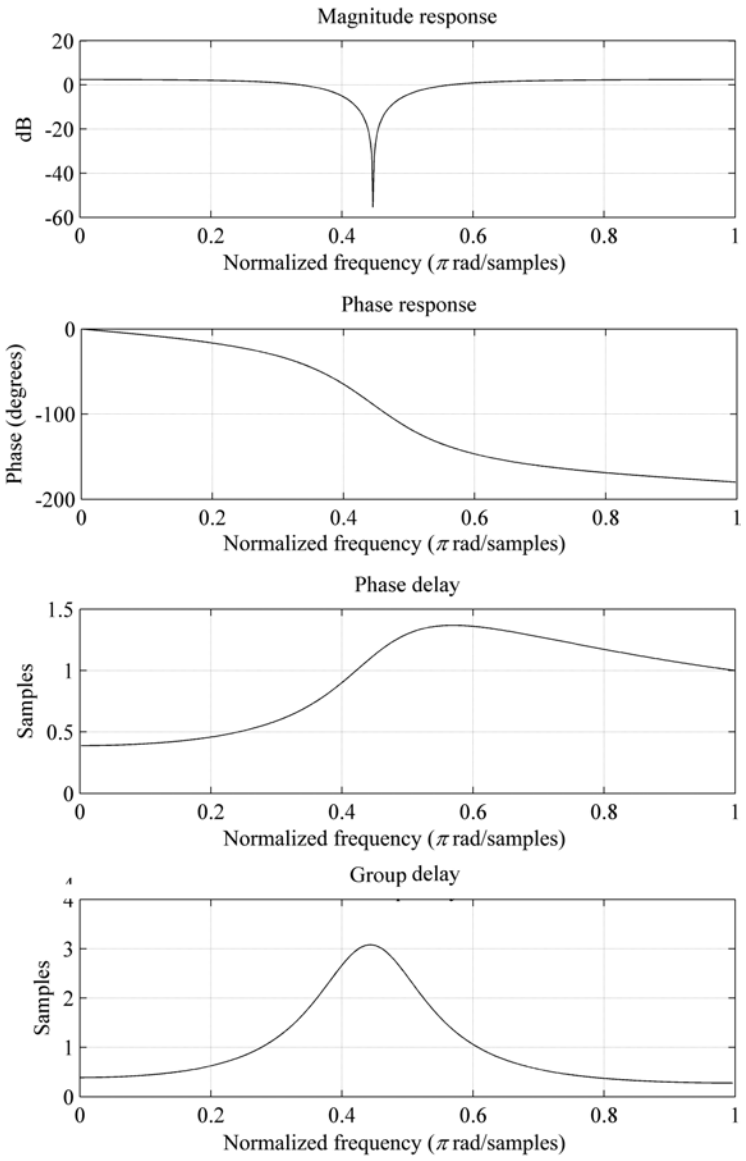


Fig. 3.8. Plot of magnitude/phase response and phase/group delay of band-stop filter. Magnitude response (top), phase response (2nd), phase delay (3rd), group delay (bottom).

Thus, in Eq. (3.27) we see that the denominator and numerator correspond to the change in frequency and phase respectively and hence Eq. (3.27) can be written as Eq. (3.28).

$$t_g = \frac{\phi_1 - \phi_2}{\omega_1 - \omega_2} = \frac{\Delta\phi}{\Delta\omega} \tag{3.28}$$

Equation (3.28) in the continuous form can be written as:

$$t_g = \frac{d\phi}{d\omega} \tag{3.29}$$

The group delay in terms of digital frequency is then as shown in Eq. (3.30) which is what we defined it as when starting this section — see Eq. (3.17).

$$\tau_g = -\frac{d\Theta}{d\theta} \tag{3.30}$$

Figure 3.8 shows the magnitude/phase responses, phase delay, and group delay plots for a band-stop filter — a filter that rejects only a select frequency range (again, we'll learn about filters formally in the next chapter). This is the same difference equation used in Fig. 3.5 where the phase jumps have been unwrapped to make it continuous. We can clearly see that the phase response is nonlinear as it is not a straight line (second plot from top) but also because the phase delay and group delay are not the same. What we can gather from the various plots is that the delay of the system varies from about 0.5 to 3.5 samples with the largest amount of change occurring around 0.45 radians as exemplified at the bottom plot of Fig. 3.8 (group delay plot).

4 The (Almost) Magical Z-Transform

Up until now we have learned ways to compute the frequency response of LTI systems and although they are perhaps not the most complicated procedures, they do, however, seem not to be the easiest to compute either. In this section, we will introduce yet another way of computing the frequency response referred to as the *z-tansform*. Why another method? One of the reasons is that it is very helpful as it is an "easy" procedure for quickly and painlessly (well, almost) computing the frequency response. We start with the definition of the z-transform as shown in Eqs. (4.1) and (4.2) which basically state that when we perform the z-transform on input $x[n]$ we get $X(z)$. Now, that's pretty simple indeed, and in order to go "ahhh, that's what this means," let's introduce a few more important concepts in order to reach that state of "ahhh-ness."

$$x[n] \xrightarrow[z-trans.]{} X(z) \tag{4.1}$$

$$Z(x[n]) = X(z) = \sum_{n=-\infty}^{+\infty} x[n]z^{-n} \tag{4.2}$$

A very important property of the z-transform is shown in Eq. (4.3) known as the *shifting property.* The shifting property refers to a delay in the input sample $x[\cdot]$, in our case a delay of m samples, and its influence on the z-transform.

$$Z(x[n-m]) = \sum_{n=-\infty}^{+\infty} x[n-m]z^{-n} \tag{4.3}$$

By letting $k = n - m \rightarrow n = k + m$ and plugging it into Eq. (4.3) we get:

$$\begin{aligned} \left. \sum_{n=-\infty}^{+\infty} x[n-m]z^{-n} \right|_{n=k+m} &= \sum_{k=-\infty}^{+\infty} x[k]z^{-(k+m)} \\ &= \sum_{k=-\infty}^{+\infty} x[k]z^{-k}z^{-m} \\ &= z^{-m} \sum_{k=-\infty}^{+\infty} x[k]z^{-k} \end{aligned} \tag{4.4}$$

We note that the summation part at the end of (4.4) is identical to the form of $Z(x[n])$ where the difference is only in the variable name: $n = k$. Hence, by changing the variable name from k to n, we can rewrite Eq. (4.4) as shown below.

$$x[n-m] \xrightarrow[z-trans.]{} z^{-m}X(z) \tag{4.5}$$

$$Z(x[n-m]) = z^{-m} \sum_{n=-\infty}^{+\infty} x[n]z^{-n} = z^{-m}X(z) \tag{4.6}$$

This is a very interesting result as it says that whenever there is shifting (delay) in the z-transform, the shift can be brought to the outside (multiplication) of the z-transform summation in the form of "z to the power of the delay" resulting in Eqs. (4.5) and (4.6). So far so good ... But what do these things have to do with the frequency response? Read on to find out more ...

4.1 *What does all this mean? Part I*

In this section, we will see how to directly perform the z-transform on any difference equation and quickly describe the system's frequency response via the so-called *poles* and *zeros* on the *z-plane*. Let's perform the z-transform on the general difference equation (4.7). The mechanics of the z-transform just entails transforming (mostly capitalizing and changing variables names!) the input, output, and delay components as shown in Eqs. (4.8) and (4.9).

$$y[n] = \sum_{k=1}^{k=N} a_k y[n-k] + \sum_{k=0}^{k=L} b_k x[n-k] \tag{4.7}$$

$$x[n-m] \rightarrow z^{-m} \cdot X(z) \tag{4.8}$$

$$y[n-m] \rightarrow z^{-m} \cdot Y(z) \tag{4.9}$$

The general difference equation when transformed yields Eqs. (4.10) and (4.11).

$$Y(z) = Y(z) \sum_{k=1}^{k=N} a_k z^{-k} + X(z) \sum_{k=0}^{k=L} b_k z^{-k} \tag{4.10}$$

$$Y(z)\left(1 - \sum_{k=1}^{k=N} a_k z^{-k}\right) = X(z) \sum_{k=0}^{k=L} b_k z^{-k} \tag{4.11}$$

When rearranging Eq. (4.11) so that we have it in the form $Y(z)/X(z)$ (transfer function) Eq. (4.12) results.

$$\frac{Y(z)}{X(z)} = \frac{\sum_{k=0}^{k=L} b_k z^{-k}}{1 - \sum_{k=1}^{k=N} a_k z^{-k}} \tag{4.12}$$

This looks very familiar and upon comparing it with the definition of the frequency response of Eq. (2.12), we can see that by setting $z = e^{j\theta}$ Eqs. (4.12) and (2.12) become identical:

$$H(e^{j\theta}) = H(z)|_{z=e^{j\theta}} \tag{4.13}$$

Although the transformation from the difference equation to $H(z)$ (and hence the frequency response) is quite straightforward itself, I don't think we are quite yet ready to buy that computing the frequency response itself has been facilitated in any way. This facilitation is, however, achieved in the next section, when we utilize poles and zeros.

4.2 *What does all this mean? Part II: poles and zeros*

We will see in this section how poles and zeros of the z-transform are obtained and how they can be used to get the frequency response of a system. The poles and zeros are attained by solving for the roots of the denominator and numerator parts of $H(z)$ — setting the numerator and denominator equal to zero (providing they have delay components) and finding the solutions for the numerator and denominator with respect to z. Via the solution of the numerator and denominator, which correspond to the location of the poles and zeros, we can quickly determine the frequency response characteristics of a system as they directly reflect the resonant and dampening characteristics of a system as seen in Fig. 4.1.

$$H(z) = \frac{\text{numerator}(z)}{\text{denominator}(z)} \tag{4.14}$$

$$\text{numerator}(z) = 0 \tag{4.15}$$

$$\text{denominator}(z) = 0 \tag{4.16}$$

Resonant locations refer to specific frequencies locations defined by the poles reinforcing and amplifying the magnitude values at specific frequency locations. Dampening characteristics refer to frequency locations generally defined by zeros which dampen or reduce the magnitude values of a frequency response. The resonating and dampening characteristics can be intuitively

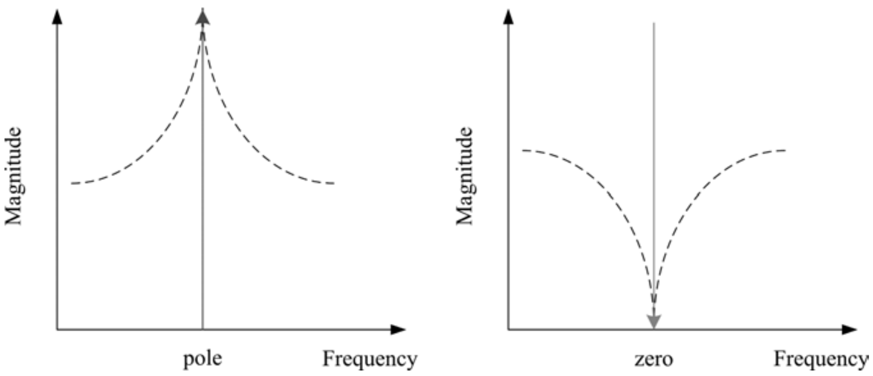


Fig. 4.1. Pole and zero pulling and pushing characteristics.

understood by using poles and zeros — poles correspond to the roots of the denominator meaning that the transfer function $H(z)$ will take the form $Y(z)/0$. In other words, the divide by zero will render the $H(z)$ to go to infinity. For the magnitude response plot, this would induce the magnitude response envelope's shape to resemble a tent propped up at the pole location so-to-speak as seen in the left side of Fig. 4.1. The zeros on the other hand take the form $0/X(z)$ which will cause $H(z) = 0$, thus pulling the tent towards 0 or the "ground" via the tent pegs. Let's read on to find out more about this pulling and propping characteristic of magnitude response $|H|$.

4.3 *What does all this mean? Part III: the unit circle and the z-plane*

To get a better grasp how the poles and zeros relate to the formation of the magnitude response, let's use the IIR difference equation $y[n] = x[n] + 0.5 \cdot y[n-1]$ as an example to compute its frequency response via the z-transform method. We start by first computing $H(z)$.

$$H(z) = \frac{1}{1 - 0.5 \cdot z^{-1}} = \frac{z}{z} \cdot \frac{1}{(1 - 0.5 \cdot z^{-1})} = \frac{z}{z - 0.5} \tag{4.17}$$

For the denominator and numerator roots, we equate both parts to zero yielding the following two equations for the pole and zero respectively.

$$z - 0.5 = 0 \tag{4.18}$$

$$z = 0 \tag{4.19}$$

Solving for the roots of the poles and zeros we have 0.5 for the pole and 0 for the zero. Next, we will introduce the concept of the unit circle and the accompanying z-plane to make sense of how poles and zeros influence and shape the magnitude response's magnitude envelope. The unit circle as the name implies, refers to a circle with radius 1 centered at the origin and is plotted on a 2-dimensional z-plane with imaginary and real axes as shown in Fig. 4.2. Traditionally, a pole is represented with the symbol "x" and a zero with an "o" on this z-plane.

Imagine that the unit circle is drawn by sweeping the digital frequency θ in a counterclockwise direction from 0 to π (DC to Nyquist) with unity gain as seen in Fig. 4.3 via $1 \cdot e^{j\theta}$. We will remember that the definition of the frequency response is the observation of a system's magnitude and phase response when the system is excited by an input signal with constant amplitude and phase while sweeping its input frequency. The $e^{j\theta}$ in essence

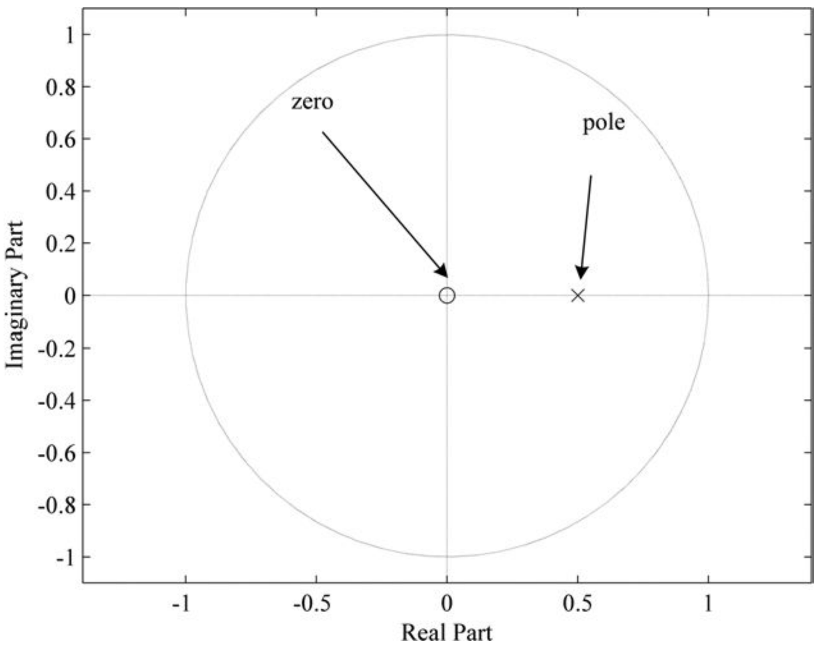


Fig. 4.2. Unit circle with pole and zero at 0.5 and 0.

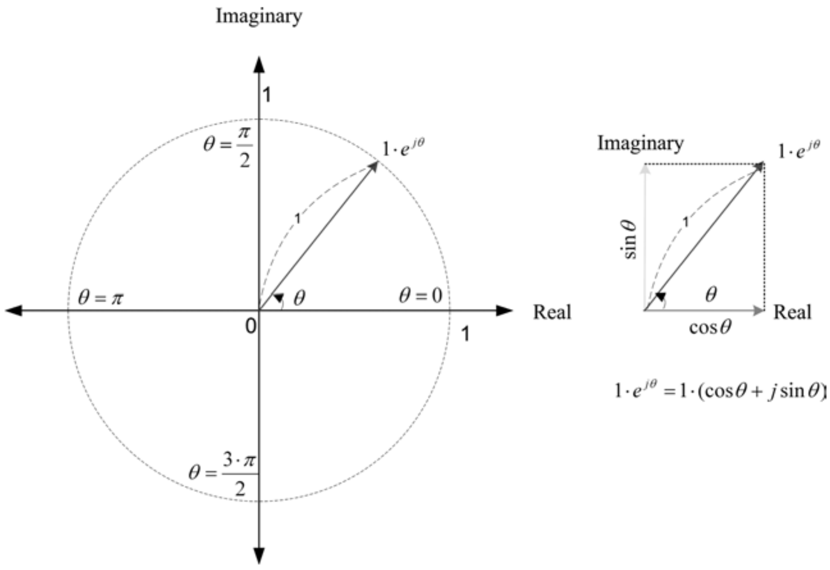


Fig. 4.3. Unit circle and Euler identity.

is used in such a capacity — frequency sweep a system (transfer function) while maintaining unity gain and 0 initial phase, with the goal of observing the magnitude and phase response vs. frequency. For the above difference equation example, the pole and zero are plotted in Fig. 4.2. If the roots of the poles and zeros had been complex in the form of $a + jb$, the pole and/or zero locations would have been above or below the $b = 0$ line.

Our last step is finally figuring out how to plot and sketch the magnitude response from the information provided by the poles and zeros on the z-plane. We know that for the frequency response, our range of interest for frequency is from 0 to $f_s/2$ or 0 to π in radians. This corresponds to the first two quadrants of the unit circle. It was mentioned before that generally, poles have a resonating quality (amplifying) and zeros have a dampening (attenuating) quality and this is what is shown in Fig. 4.4 with respect to the magnitude response of the system. We see in Fig. 4.4 that the pole has its strongest influence (resonance) on the magnitude $|H(z)|$ at 0 radians whereas the zero strongest influence (dampening) at π — the pole is closest to DC and zero closest to the Nyquist in this example. Just before $\pi/2$ radians we can see that there is a cancellation of the pole and zero — at this frequency, the zero and pole have equal "force" but with reverse polarity so-to-speak (one pulling and the other pushing) thus cancelling each other

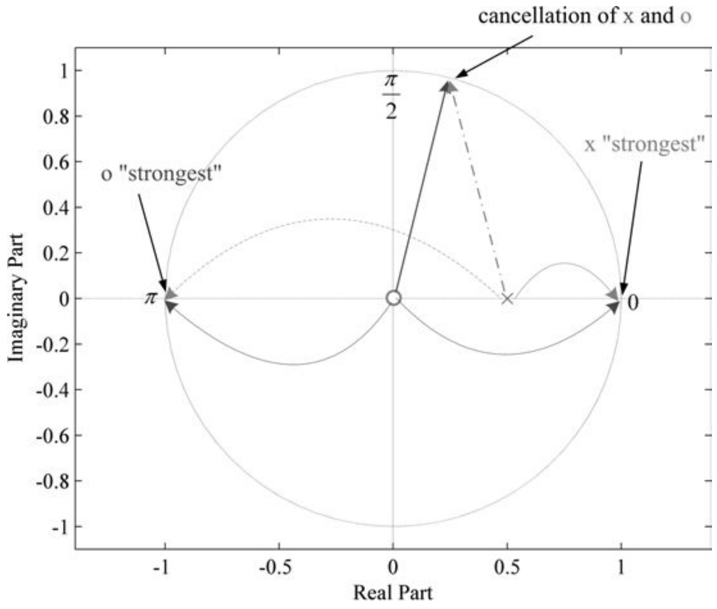


Fig. 4.4. Important characteristics of pole and zero.

out. The characteristics of $|H(z)|$ and its relationship to poles and zeros can also be explained when we bring back the relationship between $Y(z)$ and $X(z)$ from Sec. 4.1.

$$|H(z)| = \frac{|Y(z)|}{|X(z)|} \tag{4.20}$$

We have from our difference equation example the following transfer function relationship.

$$|H(z)| = \left|\frac{z}{z - 0.5}\right| = \frac{|z|}{|z - 0.5|} \tag{4.21}$$

This means that the denominator and numerator represent a Euclidian distance measure and $H(z)$ a ratio of such distances determined by $Y(z)$ and $X(z)$. This is shown in Fig. 4.5 where we can see that the magnitude of $|H(z)| = |Y(\mathrm{z})|/|X(z)|$ is once again strongest at 0 radians, weakest at π, and equal to 1 when $|Y(\mathrm{z})| = |X(z)|$ just before $\pi/2$ which is congruous to the results we obtained in Fig. 4.4. Note that in this example, the $|Y(z)|$ component stays constant as it is located at $z = 0$ (origin of the unit circle), whereas the $|X(z)|$ component increases in magnitude causing a decrease of $|H(z)|(= |Y(\mathrm{z})|/|X(z)|)$ as we approach π. Therefore, knowing

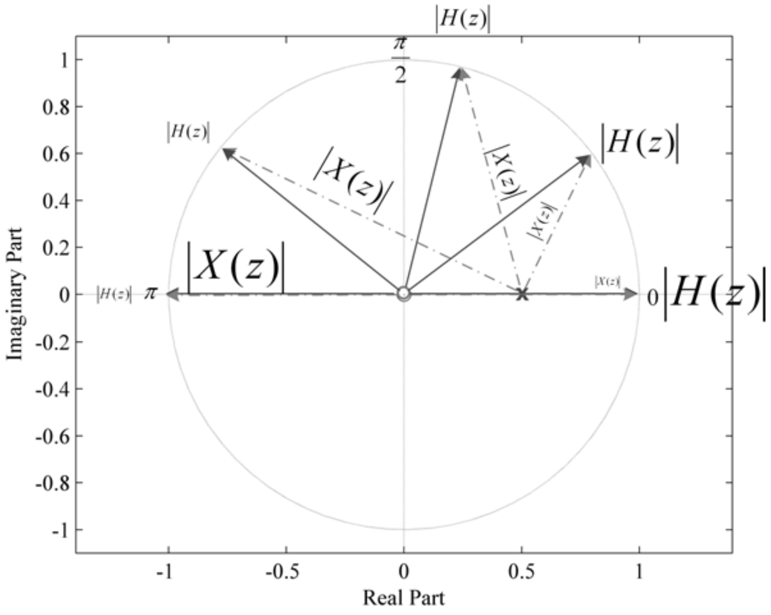


Fig. 4.5. Overall magnitude plot via frequency sweep from 0 to π.

the locations of dips, cancellations, and peaks with respect to the poles and zeroes on the z-plane, it is possible to quickly sketch the frequency response on paper as shown in Fig. 4.6. Figure 4.7 shows a detailed computer plot of the same difference equation's frequency response.

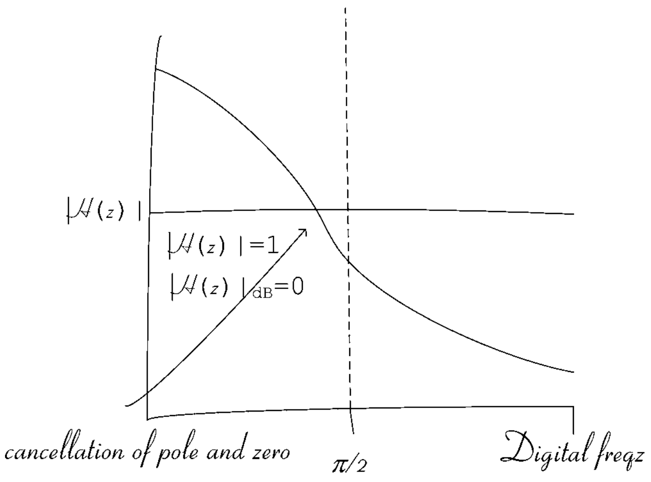


Fig. 4.6. Frequency response sketch via zero/pole for $y[n] = x[n] + 0.5 \cdot y[n-1]$.

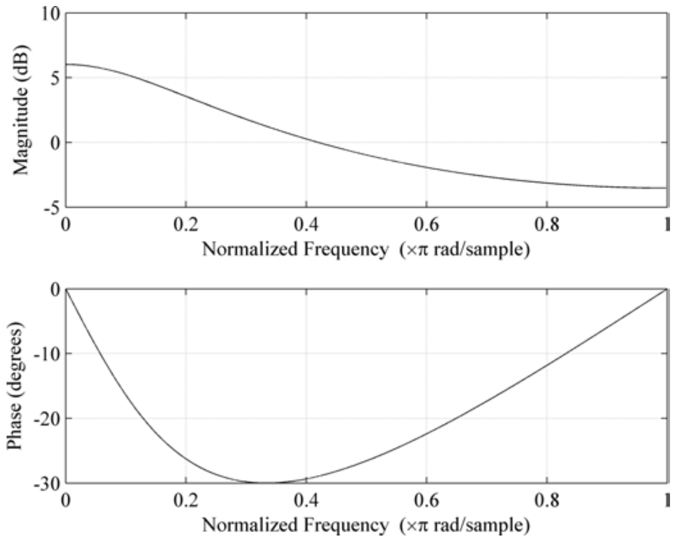


Fig. 4.7. Detailed computer-based frequency/phase response plots of $y[n] = x[n] + 0.5 \cdot y[n-1]$. Top shows the magnitude response and bottom the phase response [in MATLAB®: *FREQZ*(1, [1 -0.5])].

4.4 *More "complex" systems*

Oftentimes, when one solves for roots of the z-transform, complex solutions are obtained. That is, we get zeros/poles spread throughout the z-plane in the form of $a + jb$ unlike our previous simpler example. For instance, if an all zero (FIR) system takes on the form

$$H(z) = a \cdot z^2 + b \cdot z + c \tag{4.22}$$

the general solution for this second order polynomial is as usual:

$$z = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{4.23}$$

But when the root part is less than zero as shown in Eq. (4.24) the system will result in a complex solution which in turn will spread the zeros on the real and imaginary axes — place zeros and poles on a plane.

$$b^2 - 4ac < 0 \tag{4.24}$$

On top of that, when the roots are complex, the solution will always result in conjugate pairs. That is, each root will exist as shown in Eq. (4.25).

$$z = a \pm jb \tag{4.25}$$

Thus, a reflection occurs on the $b = 0$ imaginary axis in the z-plane. Roots obviously may exist for the denominator (pole) as well as the numerator (zero) which is the way difference equations and filters are generally structured.

5 Region of Convergence (ROC)

An important concept to consider with z-transforms is the so-called *region of convergence* (ROC). The ROC is important as it defines where the z-transform can exist. The formal definition of the ROC is represented by the convergence of the summation of the z-transform and is met when (5.1) is satisfied for an impulse sequence $h[n]$.

$$\sum_{n=-\infty}^{n=+\infty} \left| h[n] \cdot z^{-n} \right| < \infty \tag{5.1}$$

To show what conditions satisfy the convergence of Eq. (5.1) (boundedness), we note that the above equation is in the form of the geometric series. We can thus break up the summation into negative and positive sample index parts according to Eq. (5.2).

$$\sum_{n=-\infty}^{n=+\infty} \left|h[n] \cdot z^{-n}\right| = \underbrace{\sum_{n=-\infty}^{n=-1} \left|h[n] \cdot z^{-n}\right|}_{negative} + \underbrace{\sum_{n=0}^{n=+\infty} \left|h[n] \cdot z^{-n}\right|}_{positive} \tag{5.2}$$

What we can observe from the above is that the system can be divided into a non-causal and causal part, where the negative summation segment corresponds to the non-causal part and the positive summation causal part. Let's simplify things a bit before coming to a general approach in determining the ROC by looking at the geometric series. The general form of the geometric series is as follows:

$$S_N = \sum_{n=0}^{n=N} z^n = 1 + z^1 + \cdots + z^{N-1} + z^N \tag{5.3}$$

By multiplying the sum S_N with z, we get the following relationship:

$$z \cdot S_N = z + z^2 + \cdots + z^{N-1} + z^N + z^{N+1} \tag{5.4}$$

If we subtract the z-multiplied sum $z \cdot S_N$ (5.4) from S_N (5.3) to get rid of all the common components between $z \cdot S_N$ and S_N, we arrive at Eqs. (5.5) and (5.6).

$$S_N - z \cdot S_N = 1 - z^{N+1} \tag{5.5}$$

$$S_N(1 - z^{-1}) = 1 - z^{-N-1} \tag{5.6}$$

With further simplification we can express the sum S_N of the geometric series in a very compact format as shown in Eq. (5.7).

$$S_N = \frac{1 - z^{N+1}}{1 - z}, \quad \text{for } z \neq 1 \tag{5.7}$$

When $z = 1$ in Eq. (5.7), a "divide by zero" issue arises causing S_N to go to infinity. When N is set to ∞, S_N can be further simplified as shown in (5.8), provided that $0 < |z| < 1$ (a fractional number). If $|z|$ is greater than 1, the z^∞ component will reach infinity causing S_∞ to reach infinity as well. However, if $|z|$ is smaller 1, it will approach 0 and make the result stable (bounded).

$$S_\infty = \frac{1 - z^{\infty+1}}{1-z} = \frac{1 - z^{\infty}}{1-z} = \frac{1}{1-z}, \quad \text{for } |z| < 1 \tag{5.8}$$

Let's go back to the definition of the ROC in Eq. (5.2) and look at the negative summation part and rewrite it as shown below. This is achieved by changing the polarity of the power of z^{-n} to z^{+n} which in turn also changes the characteristics of the index n for the summation.

$$\sum_{n=-\infty}^{n=-1} \left| h[n] \cdot z^{-n} \right| = \sum_{n=1}^{n=\infty} \left| h[-n] \cdot z^{n} \right| \tag{5.9}$$

We note that this series is looks very similar to the one we derived above in Eq. (5.7) with the exception of the additional $h[-n]$ in (5.9) and 0^{th} power component in Eq. (5.3). For simplicity, let's set $h[-n] = 1$. Now, we can easily put (5.9) into the (5.4) structure by subtracting the 0^{th} z-component from the summation of Eq. (5.9) and designating the summation range from $n = 0$ to ∞ as shown below ($z^0 = 1$).

$$\sum_{n=-\infty}^{n=1} |h[n] \cdot z^{-n}| = \sum_{n=1}^{n=\infty} |1 \cdot z^{n}| = \sum_{n=0}^{n=\infty} |z^n - 1| \tag{5.10}$$

As before, we can also derive the geometric sum from 0 to N only, as shown below in Eq. (5.11).

$$\sum_{n=0}^{n=N} |z^n - 1| = \left| \frac{1 - z^{N+1}}{1-z} - 1 \right| \tag{5.11}$$

The summation will converge if $|z| < 1$ as shown below.

$$\left| \frac{1 - z^{N+1}}{1-z} - 1 \right|_{N=\infty} = \left| \frac{1-0}{1-z} - 1 \right| = \left| \frac{1-(1-z)}{1-z} \right|$$
$$= \frac{z}{1-z} = \frac{z}{1-z} \cdot \frac{z^{-1}}{z^{-1}} = -\frac{1}{1-z^{-1}}, \quad \text{for } |z| < 1 \tag{5.12}$$

This is where the z-transform is defined. In other words, for this particular system where $h[-n] = 1$, the ROC is within the unit circle as shown in Fig. 5.1 (not including $|z| = 1$).

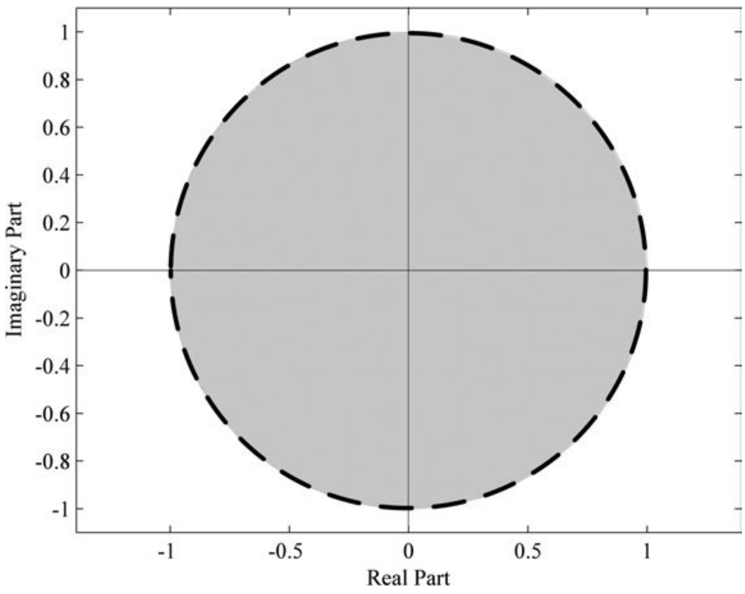


Fig. 5.1. ROC for $h[-n] = 1$, $|z| < 1$.

Let's now consider an example of a more interesting $h[n]$ for a non-causal system as shown below and determine its ROC.

$$h[n] = \begin{cases} a^n, & \text{for} \quad n < 0 \\ 0, & \text{otherwise} \end{cases} \tag{5.13}$$

Computing the z-transform for the summation region from $-N$ to -1 (since this is a non-causal system) we arrive at (5.14). When $N = \infty$, we get (5.15), provided $|a^{-1}z| < 1$ as shown in Fig. 5.2 where the inequality can be rewritten as shown in (5.16).

$$\begin{aligned} \sum_{n=-N}^{n=-1} h[n] \cdot z^{-n} \bigg|_{h[n]=a^n} &= \sum_{n=-N}^{n=-1} a^n \cdot z^{-n} = \sum_{n=0}^{n=N} a^{-n} \cdot z^n - 1 \\ &= \sum_{n=0}^{n=N} \left(a^{-1} \cdot z\right)^n - 1 = \frac{1 - (a^{-1}z)^{N+1}}{1 - a^{-1}z} - 1 \end{aligned} \tag{5.14}$$

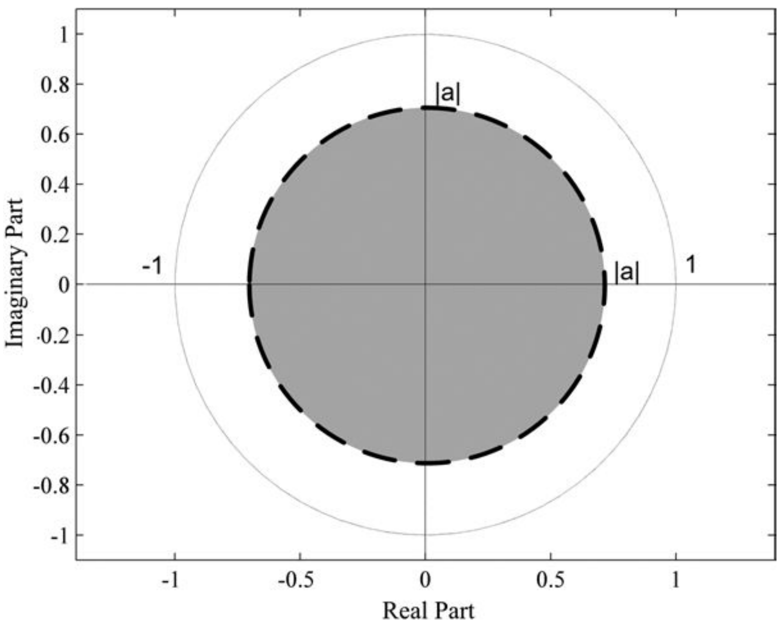


Fig. 5.2. ROC for non-causal $h[n] = a^n$, $|z| < |a|$.

$$\left.\frac{1-(a^{-1}z)^{N+1}}{1-a^{-1}z} - 1\right|_{N=\infty} = -\frac{1-0}{1-a^{-1}z} - 1$$

$$= \frac{a^{-1}z}{1-a^{-1}z} = -\frac{1}{1-az^{-1}}, \quad \text{for } |a^{-1}z| < 1 \tag{5.15}$$

$$|a^{-1} \cdot z| < 1 \rightarrow |z| < |a| \tag{5.16}$$

This means that the z-transform for $h[n] = a^n$ will converge to (5.15) which has a zero at $z = 0$ and pole at $z = a$ as $1/(1 - a^{-1}z) = z/(z - a^{-1})$ in (5.15). It so turns out that the poles of a z-transform cannot occur within the region of convergence as the z-transform does not converge at a pole, the ROC itself, however, is bounded by poles instead. In Fig. 5.2, the zero will be located at the origin and the pole at $z = a$ as previously mentioned.

Note also that $h[n]$ can also be expressed in terms of the unit-step function, that is

$$h[n] = a^n \cdot u[-n-1] \tag{5.17}$$

and observe that the exponential term only exists for negative power n terms. In other words, since $u[-n-1]$ only exists for negative n integer values (which is the definition of a non-causal system), a^n can be rewritten as a^{-n} (or $1/a^n$) for $n \geq 1$.

5.1 *Causal system ROC*

For a causal system, a system that is essentially the positive summation component of Eq. (5.2) we can derive the ROC by using the same approach we applied for the non-causal system. Let's analyze the ROC for $h[n]$ as seen below.

$$h[n] = \begin{cases} a^n, & \text{for} \quad n \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{5.18}$$

$$\begin{aligned} \sum_{n=-\infty}^{n=+\infty} \left| h[n] \cdot z^{-n} \right| &= \sum_{n=0}^{n=\infty} \left| h[n] \cdot z^{-n} \right| \\ &= \sum_{n=0}^{n=\infty} \left| a^n \cdot z^{-n} \right| = \sum_{n=0}^{n=\infty} \left| (a \cdot z^{-1})^n \right| \end{aligned} \tag{5.19}$$

In (5.19) we can tell from inspection that the summation up to N becomes:

$$\sum_{n=0}^{n=N} (a \cdot z^{-1})^n = \frac{1-(a \cdot z^{-1})^{N+1}}{1-a \cdot z^{-1}} \tag{5.20}$$

When N is set to infinity, we get the following result:

$$\sum_{n=0}^{n=\infty} (a \cdot z^{-1})^n = \frac{1-(a \cdot z^{-1})^{\infty+1}}{1-a \cdot z^{-1}} = \frac{1}{1-a \cdot z^{-1}} \quad \text{for } \left| a \cdot z^{-1} \right| < 1 \tag{5.21}$$

The inequality in (5.21) can be rewritten as (5.22) and is shown in Fig. 5.3.

$$\left| a \cdot z^{-1} \right| = \left| \frac{a}{z} \right| \quad \text{or } |a| < |z| \tag{5.22}$$

Thus, for causal systems, the ROC lies outside of the unit circle.

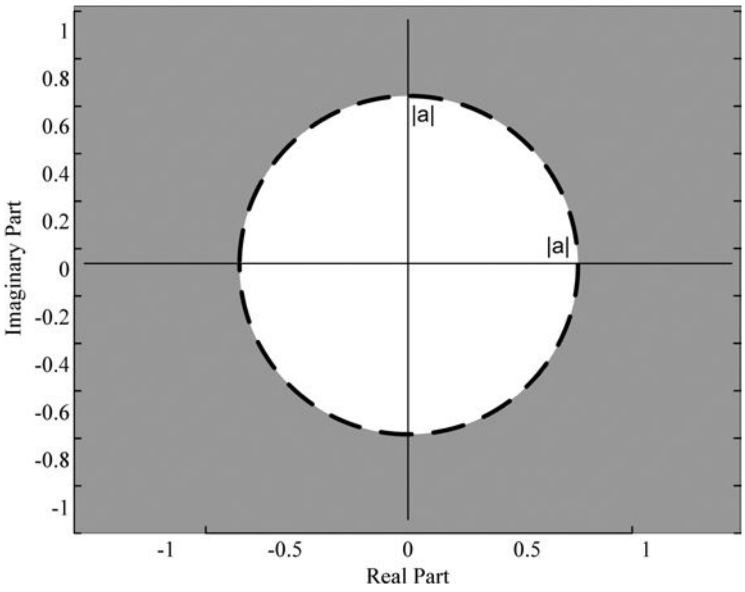


Fig. 5.3. ROC for causal $h[n] = a^n$, $|a| < |z|$.

5.2 *Mixed-causality systems*

For systems that include both causal and non-causal parts such as Eq. (5.23), a circular band resembling a two-dimensional doughnut is formed as depicted in Fig. 5.4. This reflects the ROC's inequalities as shown in Eqs. (5.24) and (5.25).

$$h[n] = a^n \cdot u[n] + b^n \cdot u[-n-1] \tag{5.23}$$

We know that the causal component results in Eq. (5.24) and the non-causal component in Eq. (5.25) as we have seen in our previous analysis.

$$|z| > |a| \tag{5.24}$$

$$|z| < |b| \tag{5.25}$$

Due to linearity (since we are dealing with LTI systems) we can combine the two results into one inequality Eq. (5.26) which is what we see in Fig. 5.4 and Eq. (5.26).

$$|a| < |z| < |b| \tag{5.26}$$

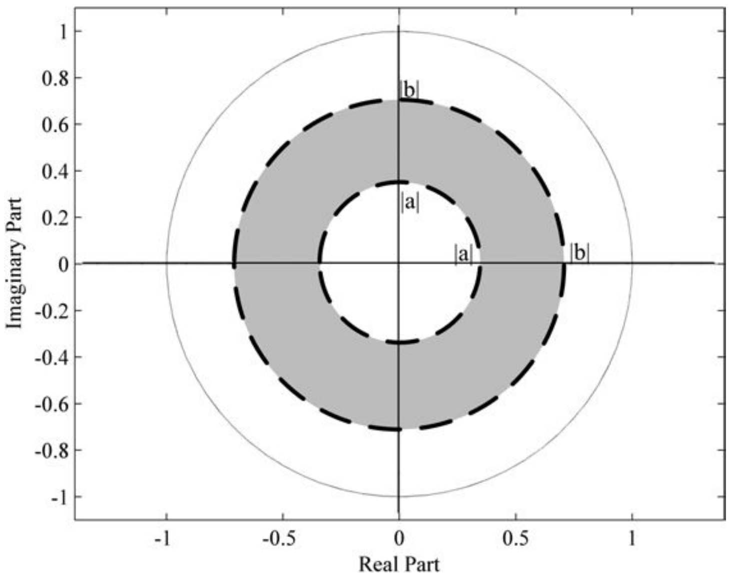


Fig. 5.4. ROC for a mixed causality system, $|a| < |z| < |b|$.

Table 5.1. ROC regions.

System Type	ROC and System Characteristics
Causal	$\|z\| > a$ (Fig. 5.3)
	All poles are inside of unit circle (for stability, see next section)
Non-causal	$\|z\| < a$ (Fig. 5.2)
Mixed-causality	$\|a\| < \|z\| < \|b\|$ (Fig. 5.4)

5.3 *ROC summary*

From our discussion of the ROC in the previous sections we have learned that poles cannot lie inside the ROC but rather define where the ROC boundaries occur. Causal, non-causal, and mixed causality systems as discussed throughout Sec. 5 directly determine the ROC regions as summarized in Table 5.1 where a is the radius of the pole.

6 Stability and the Unit Circle

Stability in a nutshell is determined by the behavior of pole locations in the z-plane as was introduced in Chap. 5 Sec. 4.2. FIR systems or systems that do not have poles are by definition stable as they do not include feedback components — there is no fear of the system blowing up, unless of course the input blows up ... Stability is formally defined by Eq. (6.1) which essentially articulates that if the unit-sample response eventually decays to 0 at some point in time (system is bounded), it is considered stable.

$$\sum_{n=-\infty}^{n=+\infty} |h[n]| < \infty \tag{6.1}$$

Now, from the z-plane and pole perspective, this can be readily explained as systems with poles that include feedback components. As mentioned in Chap. 5, FIR systems necessarily have a finite impulse response and eventually come to rest when the input stops. IIR systems on the other hand, do not technically come to rest even when the input stops outputting samples. Due to computational precision issues of digital systems, however, if an IIR system is stable, they too will output 0 after a certain time (we only have so many bits to represent minute numbers such as 10^{-53} for example, which for all practical purposes equals 0). Since FIR systems are inherently stable, we do not need to worry about their instability. For IIRs, however, their stability (a function of the characteristics of its poles) has to be examined.

A sufficient condition for stability is met if the poles of a system are inside the unit circle. This characteristic can be seen for the unit-sample response of the difference equation used in Chap. 5, Sec. 4.2 as shown below.

$$y[n] = x[n] + a \cdot y[n-1] \tag{6.2}$$

$$h[k] = \delta[k] + a \cdot h[k-1] = a^k \tag{6.3}$$

$$H[z] = \frac{Y[z]}{X[z]} = \frac{1}{1 - az^{-1}} \tag{6.4}$$

In order for this system to be stable, it must satisfy the infinite summation condition of Eq. (6.5) as shown below. This basically means that a stable system's impulse response (like a quick on-off signal) decays to 0 at some point in time as the sum of the impulse response will necessarily be a finite value.

$$\sum_{n=0}^{n=+\infty} |h[n]| < \infty \tag{6.5}$$

Again, using the geometric series, we know that the summation will result in (6.6) for our IIR example.

$$\sum_{n=0}^{n=+\infty} |h[n]| = \sum_{n=0}^{n=N} |a^n| \Bigg|_{N=\infty} = \frac{1-|a|^{N+1}}{1-|a|}\Bigg|_{N=\infty} \tag{6.6}$$

$$|a| < 1 \tag{6.7}$$

For (6.6) to converge, $|a|$ must be smaller than 1 due to the $|a|^{N+1}$ component: if $|a| > 1$, (6.6) would grow without bound. Viewed from the z-plane, the pole needs to be within the unit circle and the ROC is obtained via Eq. (6.8) as derived in Sec. 5.

$$\sum_{n=-\infty}^{n=+\infty} \left| h[n] \cdot z^{-n} \right| < \infty \tag{6.8}$$

Thus, for a causal system, stability is ensured if the poles are inside the unit circle and the ROC includes the unit circle. For a non-causal system the opposite is true for the poles — all poles must be outside the unit circle with the ROC again including the unit circle as before. The reason that the poles are outside the unit circle in the non-causal case is because the power of the $h[n]$ are raised to the negative power, thus inverting the condition of the pole characteristics as discussed in Sec. 5.

7 The Inverse Z-Transform

To state it quite simply, the inverse z-transform takes us from the "z-domain" back to the time-domain. That is, given (7.1), the inverse z-transform of $H(z)$ will yield $h[n]$ as shown in (7.2).

$$H(z) = Y(z)/X(z) \tag{7.1}$$

$$Z^{-1}\{H(z)\} = h[n] \tag{7.2}$$

For example, if we used the same IIR difference equation $y[n] = x[n] + y[n-1]$ from one of our previous examples, the z-transform will yield (7.3).

$$H(z) = \frac{Y(z)}{X(z)} = \frac{1}{1-az^{-1}} \tag{7.3}$$

If we multiplied $H(z)$ by 2 for some reason and wanted to go back to the time-domain difference equation we would get

$$\frac{Y(z)}{X(z)} = \frac{1 \cdot 2}{1 - az^{-1}} \tag{7.4}$$

$$Y(z)(1 - az^{-1}) = 2 \cdot X(z) \tag{7.5}$$

$$Y(z) = 2 \cdot X(z) + az^{-1} \cdot Y(z) \tag{7.6}$$

which will finally yield our time-domain difference equation (7.7) remembering that $x[n] \rightarrow X(z)$, $y[n] \rightarrow Y(z)$, a delay is $g[n-L] \rightarrow z^{-L} \cdot G(z)$ from Sec. 4.

$$y[n] = 2 \cdot x[n] + a \cdot y[n-1] \tag{7.7}$$

The inverse z-transform becomes powerful not just because it can be used to obtain the difference equation, but also due to the fact that it can be utilized to compute the output $y[n]$ via $Y(z)$ or the impulse sequence $h[n]$ from $H(z)$. There are a couple of common ways to achieve this feat, including the *long division method*, *Taylor series expansion method*, and the *residue theorem method*. These methods will be briefly touched upon in the next subsection.

7.1 *Long division method*

Let's go back to the definition of the z-transform (4.2) for signal $x[n]$ as shown below. The sequence $x[n]$ of course can be anything, including the impulse sequence $h[n]$. If we consider $X(z)$ as expressed in some polynomial format as shown in Eq. (7.9), we observe a very convenient property that links $X(z)$ and $x[n]$ — the input $x[n]$ components are the coefficients (multiplier/weights) of the z^{-n} in $X(z)$.

$$\begin{aligned} X(z) &= \sum_{n=-\infty}^{+\infty} x[n]z^{-n} \\ &= \cdots + x[-1] \cdot z^{1} + x[0] \cdot z^{0} + x[1] \cdot z^{-1} + \cdots \end{aligned} \tag{7.8}$$

$$X(z) = \cdots + a \cdot z + b + c \cdot z^{-1} + d \cdot z^{-2} + e \cdot z^{-3} + \cdots \tag{7.9}$$

By inspection we can clearly see the following relationship by comparing Eqs. (7.8) and (7.9):

$$\begin{gathered} \ldots \\ x[-1] = a \\ x[0] = b \\ x[1] = c \\ x[2] = d \\ x[3] = e \\ \ldots \end{gathered}$$

Thus, knowing the polynomial representation of $X(z)$ or any z-transform for that matter, will give us $x[n]$, or in the case of $Y(z)$ and $H(z)$, $y[n]$ and $h[n]$ respectively. The above $X(z)$ example is a pretty simple one, a difference equation that has no feedback components — no denominator components. It is, however, also common to encounter IIR difference equations in signal processing making it a bit more complicated to find the sequence of polynomials with powers of z^{-n}. This is where the long-division method comes into play. As the name suggests, the long division method is congruous to the way learned to divide numbers back in primary school.

For example, let's consider a transfer function $H(z)$ given as Eq. (7.10) consisting of two poles and two zeros.

$$H(z) = \frac{z^2 + z}{(z + 0.5)(z - 0.3)}, \quad |z| > 0.5 \tag{7.10}$$

We start the long division process by first expanding $H(z)$ yielding Eq. (7.11).

$$H(z) = \frac{z^2 + z}{(z + 0.5)(z - 0.3)} = \frac{z^2 + z}{z^2 - 0.3z + 0.5z - 1.5} = \frac{z^2 + z}{z^2 + 0.2z - 1.5} \tag{7.11}$$

Then, performing long-division will produce the results shown in Eq. (7.12).

$$\begin{array}{r l}
 & \underline{1 - 0.8z^{-1} + 0.8z^{-2} \ldots} \\
z^2 + z - 1.5 \big) & z^2 + 0.2z - 1.5 \\
 & \underline{z^2 + z + 1.5} \\
 & 0 - 0.8z + 0 \\
 & \quad \underline{-0.8z - 0.8 + 1.5z^{-1}} \\
 & \qquad 0 + 0.8 - 1.5z^{-1} \\
 & \qquad 0.8 + 0.8z^{-1} - 1.2z^{-2} \\
 & \qquad \ldots
\end{array} \tag{7.12}$$

Since the coefficients of z^{-n} correspond to the $h[n]$ components from the definition of the z-transform $H(z)$, as before, we can directly obtain the values $h[n]$ from inspection for this causal system.

$$\begin{aligned}
h[0] &= 1 \\
h[1] &= -0.8 \\
h[2] &= 0.8 \\
&\ldots
\end{aligned}$$

7.2 *Taylor series expansion method*

In Sec. 5, we have already used the Taylor series expansion to derive the ROCs and noted that the infinite summation in the form of Eq. (7.12) converges when $|z| < 1$.

$$S_N = \sum_{n=0}^{n=N} z^n = 1 + z^1 + \cdots + z^{N-1} + z^N \tag{7.13}$$

$$\sum_{n=0}^{\infty} z^n = \frac{1}{1-z} \tag{7.14}$$

We also earlier derived the following more general relationship of the summation and the Taylor series expansion.

$$\sum_{n=0}^{n=\infty} (a \cdot z^{-1})^n = \frac{1 - (a \cdot z^{-1})^{\infty+1}}{1 - a \cdot z^{-1}} = \frac{1}{1 - a \cdot z^{-1}}, \quad \text{for } \left|a \cdot z^{-1}\right| < 1 \tag{7.15}$$

From the inverse z-transform point of view, Eq. (7.15) will yield a time sequence of the form $h[n] = a^n$ (see Sec. 5.1). Thus, if we can put the z-transform into a structure resembling Eq. (7.15), we can compute the time sequence which is essentially what this particular method is all about. For example, given $H(z)$ as shown in Eq. (7.16), the objective is to express $H(z)$ in the appropriate Taylor series expansion form.

$$H(z) = \frac{1 - a + z - bz^{-2}}{1 - bz^{-2} - az^{-1} + abz^{-3}}, \quad |z| > a, \ a > b \tag{7.16}$$

Let's try to break it up and do some factorization which leads us to (7.17). This is another instance where linearity is convenient as it allows breaking up a seemingly complex problem into more manageable smaller parts — we recognize that the resulting two terms both have the Taylor series structure.

$$\begin{aligned} H(z) &= \frac{1 + z^{-1} - (a+b)z^{-2}}{1 - bz^{-2} - az^{-1} + abz^{-3}} = \frac{1 + z^{-1} - bz^{-2} - az^{-2}}{(1 - az^{-1})(1 - bz^{-2})} \\ &= \frac{1}{1 - az^{-1}} + z^{-1} \cdot \frac{1}{1 - bz^{-2}} \end{aligned} \tag{7.17}$$

We therefore rewrite Eq. (7.17) as Eq. (7.18) and expand it to Eq. (7.19):

$$H(z) = \sum_{n=0}^{n=\infty} (a \cdot z^{-1})^n + z^{-1} \cdot \sum_{n=0}^{n=\infty} (b \cdot z^{-2})^n \tag{7.18}$$

$$\begin{aligned} H(z) &= (a \cdot z^{-1})^0 + (a \cdot z^{-1})^1 + (a \cdot z^{-1})^2 + \cdots + z^{-1} \\ &\quad \times \left\{(b \cdot z^{-2})^0 + (b \cdot z^{-2})^1 + (b \cdot z^{-2})^2 + \cdots\right\} \\ &= 1 + az^{-1} + a^2z^{-2} + a^3z^{-3} + \cdots + z^{-1} + bz^{-3} + b^2z^{-5} \\ &\quad + b^3z^{-7} + \cdots \\ &= 1 + (a+1)z^{-1} + (a^2)z^{-2} + (a^3 + b^1)z^{-3} \\ &\quad + (a^4)z^{-4} + (a^5 + b^2)z^{-5} + \cdots \end{aligned} \tag{7.19}$$

Thus, when $H(z)$ can be simply expressed in the form of Eq. (7.15) the resulting $h[n]$ via inverse z-transform can be obtained as shown in Eq. (7.20) after grouping all the delay components (z^{-k}) together to acquire the delay coefficients $h[n]$. That is, the last line in Eq. (7.19) is organized in terms of the delay z^{-k} and their respective coefficients/weights.

$$h[n] = \begin{cases} 1, & n = 0 \\ a+1 & n = 1 \\ a^2, & n = 2 \\ a^n + b^{(2n+1)-2}, & n = \text{odd and } n > 2 \\ a^n, & n = \text{even and } n > 2 \\ 0, & \text{elsewhere} \end{cases} \tag{7.20}$$

7.3 *Contour integration/residue theorem method*

The last method for computing the inverse z-transform that we will discuss here uses concepts from *complex variable theory.* Using complex variable theory, a relationship between $h[n]$ and $H(z)$ can be obtained as shown in Eq. (7.21). The integration is over a closed contour encircling the origin $(z = 0)$ counterclockwise inside the ROC.

$$x[n] = \frac{1}{2\pi} \oint X(z) \cdot z^{n-1} dz \tag{7.21}$$

Computing the time-sequence directly from Eq. (7.21) is usually not that straightforward but since most z-transforms in DSP tend to be rational functions (ratio of two polynomials $Y(z)/X(z)$ for example), an easier method called the Cauchy's reside theorem method can be used to arrive at $x[n]$. The right-hand side of Eq. (7.21) can be expressed using Cauchy's residue theorem as shown below, where p_k refers to the K number of poles $p_1, p_2, \ldots, p_k, \ldots, p_K$ and "Res" the residue of pole p_k.

$$\frac{1}{2\pi_j} \oint H(z) \cdot z^{n-1} dz = \sum_{k=1}^{K} \text{Res}_{z \to p_k} [H(z) \cdot z^{n-1}] \tag{7.22}$$

Calculation of the residue is accomplished via Eq. (7.23) where p_k is the m^{th} order pole and (7.24) is used for the 1st order pole $(m = 1)$ case [Eq. (7.23) reduces to Eq. (7.24) for $m = 1$].

$$\text{Res}_{\mathrm{z=p_k}}[H(z) \cdot z^{n-1}] = \frac{1}{(m-1)!} \cdot \lim_{z \to p_k} \frac{d^{m-1}}{dz^{m-1}} [(z - p_k) \cdot H(z) \cdot z^{n-1}] \tag{7.23}$$

$$\text{Res}_{z=p_k}[H(z) \cdot z^{n-1}] = \lim_{z \to p_k} (z - p_k) \cdot H(z) \cdot z^{n-1} \tag{7.24}$$

Equation (7.22) actually looks more complex than meets the eye as we will see in our example below where we will compute the $h[n]$ for (7.25).

$$H(z) = \frac{z(z+1)}{(z-0.5)(z+0.3)}, \quad |z| > 0.5 \tag{7.25}$$

From (7.25) we find 2 poles (roots of denominators) for this system.

$$z - 0.5 = 0 \rightarrow z = 0.5 \tag{7.26}$$

$$z + 0.3 = 0 \rightarrow z = -0.3 \tag{7.27}$$

We can also see that the order of the poles are one ($m = 1$). Thus, $h[n]$ can be computed using Eq. (7.24) as follows:

$$\begin{aligned}
h[n] &= \sum_{k=1}^{2} \mathrm{Res}_{z \to p_k}[H(z) \cdot z^{n-1}] \\
&= \mathrm{Res}_{z=0.5}[H(z) \cdot z^{n-1}] + \mathrm{Res}_{z=-0.3}[H(z) \cdot z^{n-1}] \\
&= \left. (z-0.5) \cdot \frac{z(z+1)}{(z-0.5)(z+0.3)} z^{n-1} \right|_{z=0.5} \\
&\quad + \left. (z+0.3) \cdot \frac{z(z+1)}{(z-0.5)(z+0.3)} z^{n-1} \right|_{z=-0.3} \\
&= \left. \frac{(z+1)}{(z-0.3)} \cdot z^n \right|_{z=0.5} + \left. \frac{(z+1)}{(z-0.5)} \cdot z^n \right|_{z=-0.3} \\
&= \frac{1.5}{0.2} 0.5^n - \frac{0.7}{0.8}(-0.3)^n \\
&= 7.5 \cdot (0.5)^n - 0.875 \cdot (-0.3)^{\mathrm{n}}
\end{aligned} \tag{7.28}$$

For more complex systems and higher order poles, the solution is more tedious to get, but the actual mechanics in computing the results are similar.

To end this section, let's prove Eq. (7.22) to make us feel better. We start by first plugging in Eq. (4.2) for $X(z)$ into Eq. (7.22) which actually gets us pretty far.

$$\begin{aligned}
x[n] &= \frac{1}{2\pi j} \oint X(z) \cdot z^{n-1} dz \\
&= \frac{1}{2\pi j} \oint \left\{ \sum_{m=-\infty}^{+\infty} x[m] z^{-m} \right\} \cdot z^{n-1} dz
\end{aligned}$$

$$= \sum_{m=-\infty}^{+\infty} \left(x[m] \cdot \frac{1}{2\pi j} \oint \left\{ z^{-m} \cdot z^{n-1} dz \right\} \right)$$

$$= \sum_{m=-\infty}^{+\infty} \left(x[m] \cdot \frac{1}{2\pi j} \oint z^{n-m-1} dz \right) \tag{7.29}$$

To get us to through the final lap, we borrow the following relationship from complex function theory helping us get rid of the integration in Eq. (7.28) (Porat 1997).

$$\frac{1}{2\pi j} \oint z^k dz = \delta[k+1] = \begin{cases} 1, & k = -1 \\ 0, & k \neq -1, k \in \text{integer} \end{cases} \tag{7.30}$$

In our problem, we are working with discrete time indexes and have $k = n - m - 1$ [see exponential term at end of Eq. (7.29)] thus when we plug in $n - m - 1$ for k in Eq. (7.30) we have:

$$\frac{1}{2\pi j} \oint z^k dz = \delta[k+1] = \frac{1}{2\pi j} \oint z^{n-m-1} dz = \delta[n-m-1+1] = \delta[n-m] \tag{7.31}$$

Plugging this back into (7.29) conveniently erases the integration part, leaving us with $x[n]$ as the result as expected. Remember that the delta function only exists when $n - m = 0$ and this occurs when $m = n$ where the delta function itself becomes 1.

$$\sum_{m=-\infty}^{+\infty} \left(x[m] \cdot \frac{1}{2\pi j} \oint z^{n-m-1} dz \right) = \sum_{m=-\infty}^{+\infty} \Big(x[m] \cdot \delta[n-m] \Big) = x[n] \tag{7.32}$$

As we have seen in this section, there are numerous ways of computing the inverse z-transform where each method has certain advantages as well as disadvantages. In the next section, we conclude the technical part of the chapter with a number of MATLAB® functions that implement some of the algorithms we covered here.

8 Useful Tools in MATLAB®

There are a few useful functions in MATLAB® which facilitate frequency response computation including *FREQZ* (*B*, *A*, *N*) and *INVFREQZ* (*H*, *W*, *NB*, *NA*). *FREQZ* (*B*, *A*, *N*) computes the magnitude and phase response for a given difference equation of order N and `INVFREQZ (H, W, NB, NA)` calculates the difference equation coefficients when fed with the frequency response `H`, frequency points W, and numerator and denominator coefficient order `NB` and `NA`. Other useful functions include the `ZPLANE (Z, P)` function which displays the z-plane and maps out the zeros and poles defined by `Z` and `P`, conveniently computed via function `ROOTS (C)`. There are also functions such as `GRPDELAY(B, A, N)` and `IMPZ (B, A, N)` used to compute the group delay and impulse response of a system respectively.

9 Musical Examples

Examples of the z-transform in music are not that common per se — although the next chapter's topic which will focus on filtering is related to it. One interesting topic that we have come across in this chapter is the issue of stability. As we have learned, in DSP, it is important to have a stable system. It is likewise an important issue and powerful source for ideas and creativity for musical composition and performance on various levels. In the previous chapter, we were briefly introduced to audio feedback, also known as the *Larsen effect*, often occuring when the microphone and speaker configuration in sound reinforcement is not properly set up. Audio feedback is in many cases undesirable, but that has not kept artists from exploiting it in a creative manner, adding it to their vast arsenal of tools for manipulation of musical material. Popular musicians, such as electric guitar gurus including Jimi Hendrix and Pete Townsend of *The Who* have used this unstable aspect of amplified sound in their works such as *Machine Gun* where Hendrix displays an array of electric guitar techniques including feedback. *The Who* like many rock bands have also utilized feedback musically in many of their compositions, one the notable albums being *Tommy*, their first so-called rock opera released in 1969, followed by *Quadrophenia* four years later in 1973. The way guitar feedback works is quite interesting as there is no microphone in the audio loop to speak of, but rather, only an electric guitar, its pick-ups, and the guitar amplifier. One usually sees the guitar player crouching in front of the amp as if offering the electric guitar to some unknown deity in a seemingly bizarre sacrificial manner. The reason for this rather interesting posture is practical — the electric guitar and amp form an input-output closed loop. With high enough gain on the amp side, this closed loop sets up a feedback configuration — the guitar strings that vibrate and get amplified by the amplifier/speakers, subsequently, due to the high gain on the amp side and the proximity of the guitar to the amp, induces the strings on the guitar to physically vibrate in a feedback fashion. It is as if the acoustics of the amplified sound from the speaker/amp actually pluck the guitar and not the guitarist.

Speaking of opera, another artist who has used feedback in his compositions is Robert Ashely. Composer/performer Robert Ashley is perhaps best known for his work in the area of opera-for-television. In a piece called *Wolfman* written in the early 1960s for live voice, tape, and vocal feedback, the microphone gain is set extremely high, allowing for the most subtle and "private" sounds to come through while at the same time setting up a very unstable sound reinforcement environment very favorable for audio feedback. This piece is quite aggressive in its timbre and is supposedly meant to be performed by a "sinister lounge singer." It is not surprising that some folks call audio feedback howling which maybe is how the title of Ashely's piece Wolfman came into existence. According to the program notes (Alga Marghen 2002), the piece came about because a singer-friend of Ashley's had hoped for Morton Feldman to write a piece which did not come to fruition. Ashley seemingly composed Wolfman "knowing" that the vocalist would not perform it which indeed turned out to be the case. This resulted in Ashley performing it himself at Charlotte Moorman's festival in 1964 in New York. Interestingly enough, Wolfman's perceptual results are in the dimensions of very high decibels which are very different from Feldman's compositional style, but at the same time, Ashley brings about those sounds by extremely subtle and softly spoken vocal sounds albeit directed into a high gain microphone ...

References and Further Reading

Alga, M. 2002. Wolfman — Robert Ashley. Alga Marghen label, Italy.

Blauert, J., Laws, P. 1978. "Group Delay Distortions in Electroacoustical Systems", Journal of the Acoustical Society of America, 63(5), 1478–1483.

Claerbout, J. 1998. http://sepwww.stanford.edu/sep/prof/toc_html/.

Greenberg, M. D. 1988. Advanced Engineering Mathematics, Prentice Hall International Editions, ISBN 0-13-011552-5.

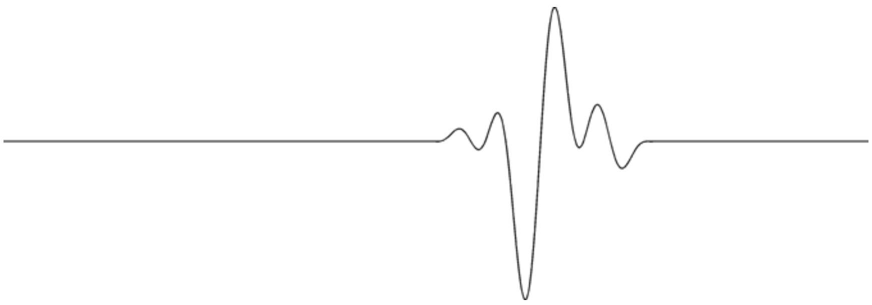
Porat, B. 1997. A Course in Digital Signal Processing. Wiley and Sons.

Smith, J. O. III 2007. Introduction to Digital Filters with Audio Applications, W3K Publishing, http://books.w3k.org/, 2007, ISBN 978-0-9745607-1-7.

Strum, R. D. and D. E. Kirk 1988. First Principles of Discrete Systems and Digital Signal Processing, Reading MA: Addison-Wesley.

Chapter 7

FILTERS



1 Introduction

Filters are very commonly found in everyday life and include examples such as water filters for water purification, mosquito nets that filter out bugs, bouncers at bars filtering the incoming guests according to age (and other criteria), and air filters found in air conditioners that we are sometimes a bit too lazy to change/clean periodically. As you may have guessed, in DSP, digital filters are ubiquitous as well and are used in a plethora of engineering, musical, and audio applications which is our focus in this chapter. Formally, in signal processing at least, a filter only modifies the amplitude and phase components of a signal. For example, if we have a sinusoid with amplitude A, frequency f, and initial phase ϕ, in filtering, only A and ϕ are modified. In the following sections, we will introduce a number of important filter types as well as interesting examples found in the area of audio and musical signal processing.

2 Low/High/Band-Pass and Band-Stop Filters

One of the most widely used and important filters is the *low-pass* filter. This was briefly mentioned in Chap. 1, when we introduced sampling and aliasing. It was also touched upon a little bit in Chap. 2, where we discussed down-sampling and up-sampling. You will remember that during sampling and analog to digital signal conversion, aliasing may occur depending on whether the highest frequency component of the analog input signal meets the Nyquist limit according to Eq. (2.1).

$$f_{\max} < \frac{f_s}{2} \tag{2.1}$$

In a situation where the above inequality is not met, aliasing occurs. But since we do not know what the input to the sampler will be and in order to guarantee that aliasing does not occur, we need to filter the analog signal so that the there are no frequency components above $f_s/2$ during the digitization process. Stated in another way, to avoid aliasing, we must pass only those frequencies that are below $f_s/2$. The low-pass filter essentially accomplishes this task — passing low frequency components while stopping higher frequency components as illustrated in Fig. 2.1. The frequency region that is passed and (ideally) unaffected by the filter is referred to the *passband* frequency area whereas the frequency range that is filtered or stopped, the *stopband*. Figure 2.1 also shows the *cutoff frequency* (f_c)

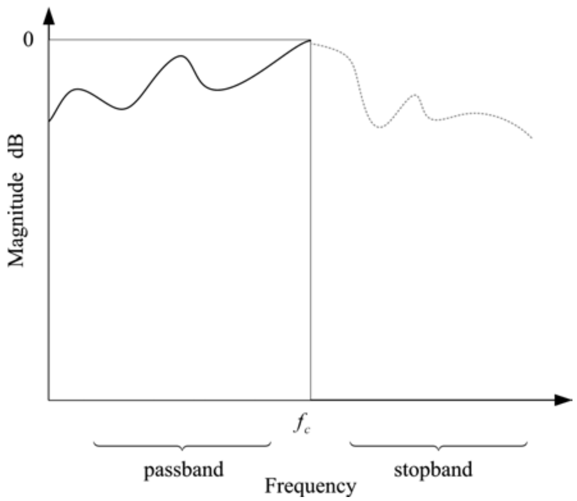


Fig. 2.1. Ideal low-pass filter.

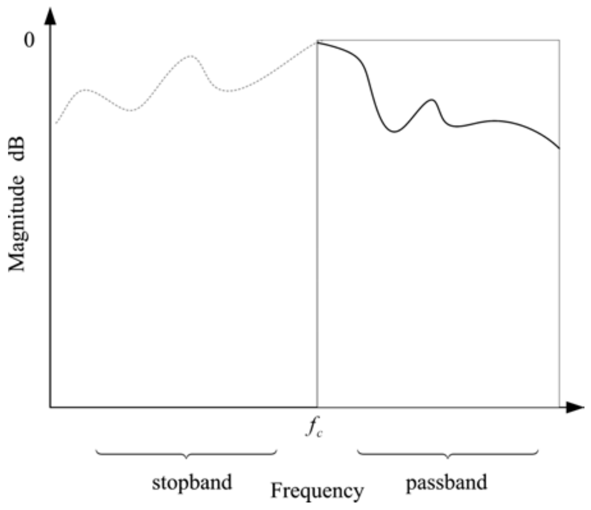


Fig. 2.2. Ideal high-pass filter.

which essentially determines the passband and stopband characteristics — the ideal low-pass filter cuts off frequency components above f_c. For an anti-aliasing filter, f_c would be equivalent $f_c = f_s/2$.

Figure 2.2 shows the ideal *high-pass filter* which stops frequency components below f_c. In Fig. 2.3, we see an ideal *band-pass filter* and the corresponding *center frequency* (f_0) as well as the upper/lower frequency bounds defining the passband region. We will further elaborate on this topic in the next section with the introduction of the Q-factor (this factor characterizes the sharpness of the band-pass region of the band-pass filter). Figure 2.4 shows the *band-stop* filter (also referred to as *band-reject filter* or *notch filter*) which is the opposite of the band-pass filter as it stops only a certain region of frequencies and passes everything else around it. Notice that the filters we have depicted in Figs. 2.1 to 2.4 are being referred to as *ideal filters*, also sometimes referred to as *brick-wall* filters. When implementing actual filters, however, brick-wall filters cannot be realistically attained and various parameters need to be tweaked to render a desired result. The cutoff frequency f_c denotes the frequency location corresponding to 0.707 (−3 dB) of the magnitude of the passband region. The number 0.707 comes from folks in electrical engineering where it is defined as half the power of the passband region. As power is proportional to the voltage squared, half the power will result in $1/\sqrt{2}$ of the voltage which is equal to 0.707.

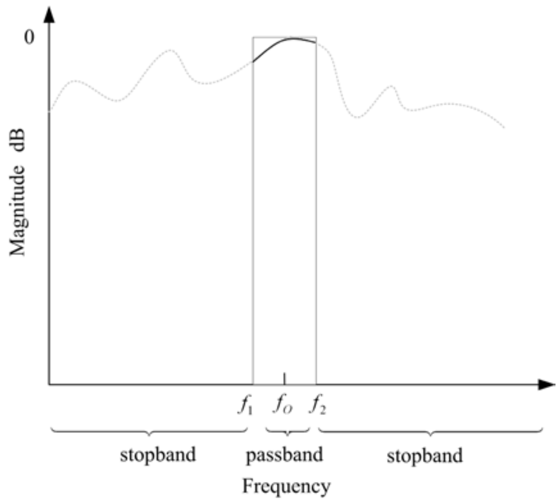


Fig. 2.3. Ideal band-pass filter.

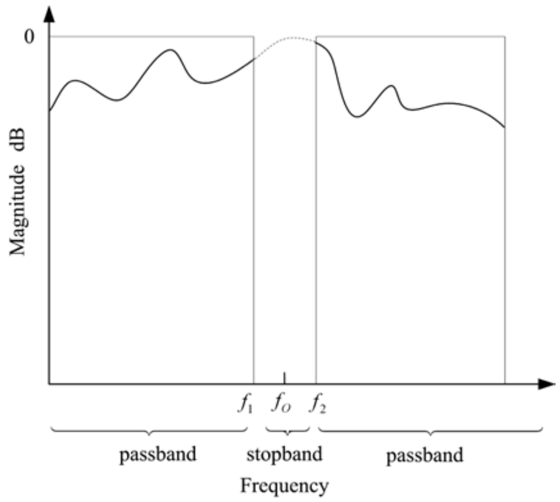


Fig. 2.4. Ideal band-stop filter.

2.1 *Filter design specifications*

In this book, we will only moderately dive into of the details of various algorithms for computing filter coefficients à la pencil-and-paper and will rather focus more on fundamental filter concepts and filter specifications

Table 2.1. Main filter characteristics.

Characteristic	Description
Passband	Unity gain region (ideally): 0 dB or 1 non-dB magnitude
Stopband	Attenuation region
Transition Band	Region between passband and stopband
Ripple	Amount of ripple tolerance allowed for filter
Filter Order/Length/Size/Tap	Number of delay components

which determine the behavior of filters. Programs such as MATLAB® can be used to design filters that meet desired specifications and will be discussed here as well. The main characteristics that determine filters are summarized in Table 2.1 for the low-pass, high-pass, band-pass, and band-stop filter. You will note that unlike ideal filters, real filters do not behave like brick walls as mentioned above, but rather are smoother around the passband/stopband edges and also inevitably introduce artifacts to the filtered signal. Therefore, filter parameters must be carefully chosen to meet a desired output as elaborated on in more detail below.

2.2 *Passband, stopband, and transition band*

The *passband* and *stopband* refer to the frequency regions that are to be (ideally) passed unchanged and frequency regions that are to be (ideally) stopped respectively. An ideal filter has unity gain and no distortion in the passband and infinite attenuation in the stopband. This however, not the case for real-world filters and some amount of distortion occurs in the passband as well as the stopband, allowing small amounts of the stopband region to trickle through. As we can see in Fig. 2.5 and subsequent figures in Sec. 2.4, unlike the ideal filter, a *transition band* exists between the stopband and passband — there is a smooth transition from the passband to the stopband and the other way around. Although the transition band is not limited to any specific values, it is usually required that the magnitude in the transition band decreases monotonically (always keeps decreasing for passband to stopband regions, always keeps increasing for stopband to passband transition bands). One of the reasons why a sampling rate of 44.1 kHz is used in audio instead of 40 kHz is due to this transition band — a 4.1 kHz headroom is left at the upper end of the spectrum to partly address the non-brick-like behavior of low-pass filters. In an ideal world, this extra buffer would not be necessary as our hearing limitation is around 20 kHz, suggesting a 40 kHz sampling rate which should be sufficiently high to fully capture audio signals digitally. But then again, we sometimes have folks who can hear like bats . . . Hence, another reason for the need of some extra legroom and a slightly higher sampling rate than theoretically necessary.

2.3 *Cutoff frequency*

The cutoff frequency as mentioned in the beginning of section, refers to the frequency location that determines where the passband and stopband will begin and end. It is defined as the frequency location corresponding to a magnitude of 0.707 of the passband region (passband = 1.0). In practical filter design situations, the cutoff frequency is further divided into the passband and stopband frequency components due to the transition band. These frequency boundaries are also referred to as *edge frequencies* characterizing the transition band.

2.4 *Filter order, filter sharpness, and ripple*

When implementing digital filters, the issue of a filter's sharpness (closeness to a brick-wall filter) and the amount of *ripple* in the passband is an important one. The shorter and smaller the transition band, the closer it is to a brick-wall filter. Also, the less amount of ripple (further explained in the next paragraph) a filter has, the closer the passband is to a straight line (closer to an ideal filter), passing the signal's desired frequency components with minimum distortion. The general rule of thumb is that the sharper a filter, the more delay components are needed (higher order filter). For an FIR filter, this equates to a larger number of zeros in the z-plane resulting in greater delay. Using an IIR filter, one can apply fewer number of delays (smaller order filter) due to the feedback and resonant pole components (introduced in Chap. 6) for similar filter characteristics to the FIRs. Thus, when designing filters, one of the important factors to take into account is the compromise between filter order and filter steepness/accuracy. For example, if one designs a filter for monitoring earthquakes that automatically triggers an alarm, it would be advisable to use an IIR filter rather than a FIR filter. IIR filters inherently have a smaller filter length and hence a shorter delay as a consequence. In other words, IIR filters have better transient response and react more quickly to a given input signal such as seismic energy in our example — this equates to an earlier reaction of the alarm system to the earthquake, allowing more time for the community to evacuate (however small the time advantage may be). So, why do we not always use IIR filters? The answer is primarily stability. Stability needs to be carefully addressed with all types of IIR filters — unlike FIR filters, feedback filters are not intrinsically stable as discussed in Chap. 6. IIR filters do not automatically come with this feature. Another important trade-off that plagues the world of filter design is between ripple (further discussed below) and the length of the transition band — decreasing one will only serve to increase the other, kind of like yin and yang.

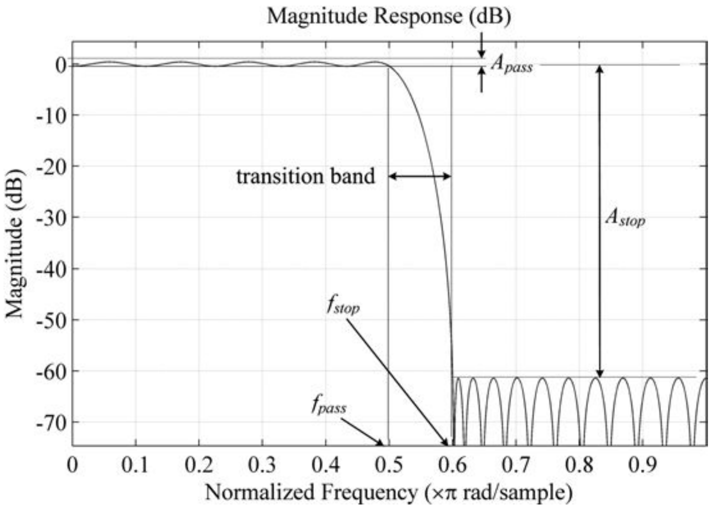


Fig. 2.5. LPF FIR frequency response.

As we can see in Fig. 2.5 and Fig. 2.6 (FIR low-pass filter), there are a number of additional filter specifications that we did not see in the brick-wall filter. These include the amount of wiggle known as *ripple* in the passband region, defined in terms of the peak-to-peak magnitude A_{pass} as shown in Fig. 2.5. The ripple specification is also sometimes represented as tolerance amount, referring to the amount of tolerance from unity gain (0 dB) in the passband as shown in Eq. (2.2) where δ^+ and δ^- are the positive and negative tolerance factors of the magnitude response in the passband respectively.

$$1 - \delta^- \leq \left|H(e^{j\theta})\right| \leq 1 + \delta^+ \tag{2.2}$$

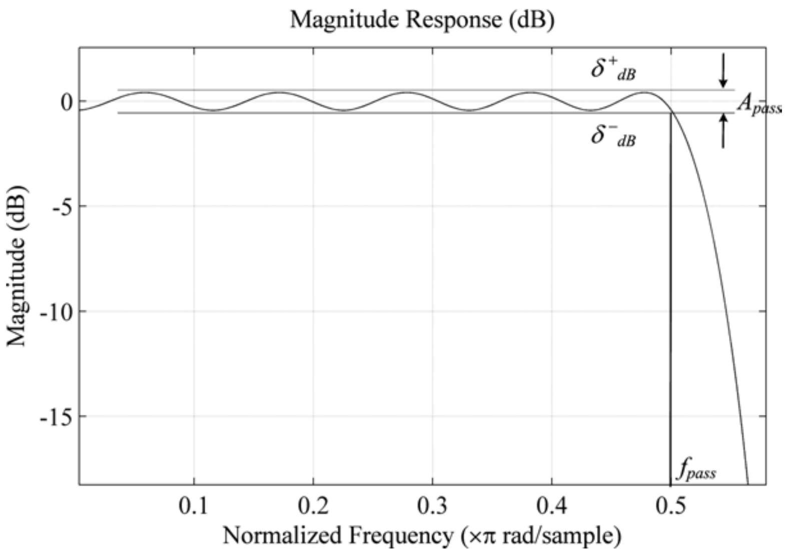


Fig. 2.6. LPF FIR passband ripple zoom-in.

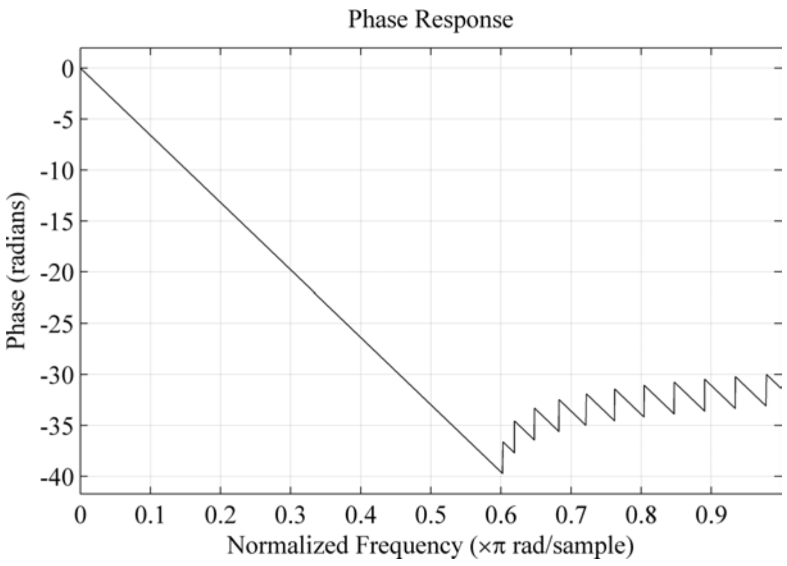


Fig. 2.7. LPF FIR phase response.

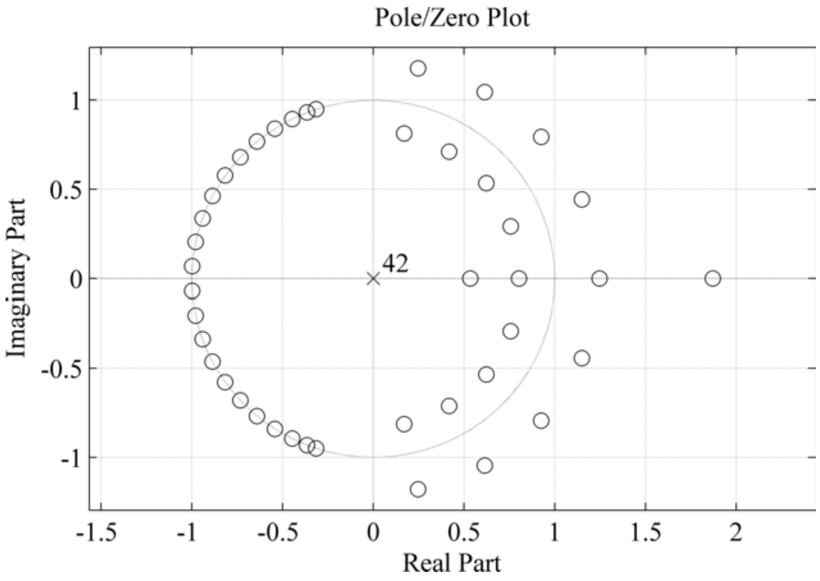


Fig. 2.8. LPF FIR zero plot of 42 zeros.

We may recall that zeros generally tend to pull the magnitude response envelope of a system towards zeros while poles tend to do the opposite — push it away from the pole location on the z-plane. This pushing and pulling effect influences the ripple characteristics of a filter.

$$\text{attenuation} = 20 \cdot \log_{10}\left(\frac{A_{\text{out}}}{A_{\text{in}}}\right) \tag{2.3}$$

Other filter specifications include the attenuation amount (usually in dB) of the stopband A_{stop} with respect to the passband as defined in Eq. (2.3) with input and output amplitude A_{in} and A_{out} and stopband/passband edge frequency f_{stop} and f_{pass}.

Figures 2.7 and 2.8 show the FIR low-pass filter's phase response and the z-plane plot with 42 zeros. The next three figures, Figs. 2.9, 2.10, and 2.11 show a FIR high-pass filter with the same number of zeros concentrated in the lower frequency regions which pull down the magnitude response for the lower frequencies.

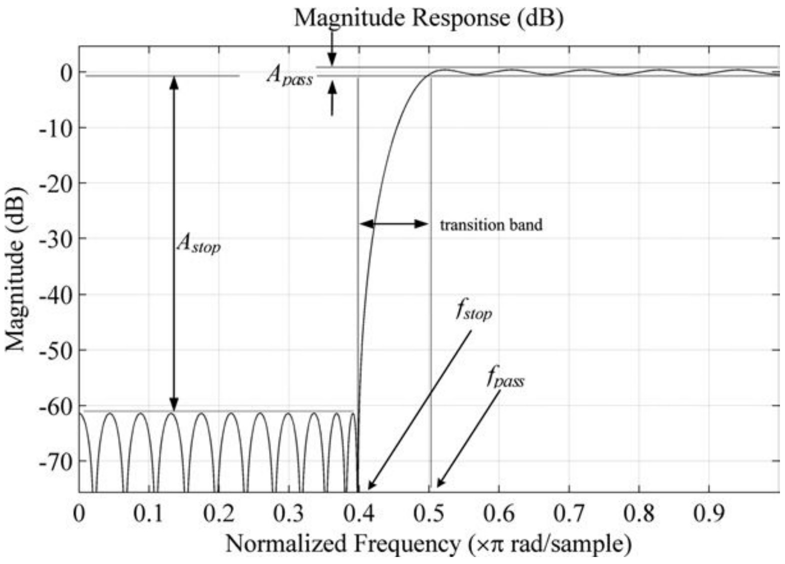


Fig. 2.9. HPF FIR magnitude response.

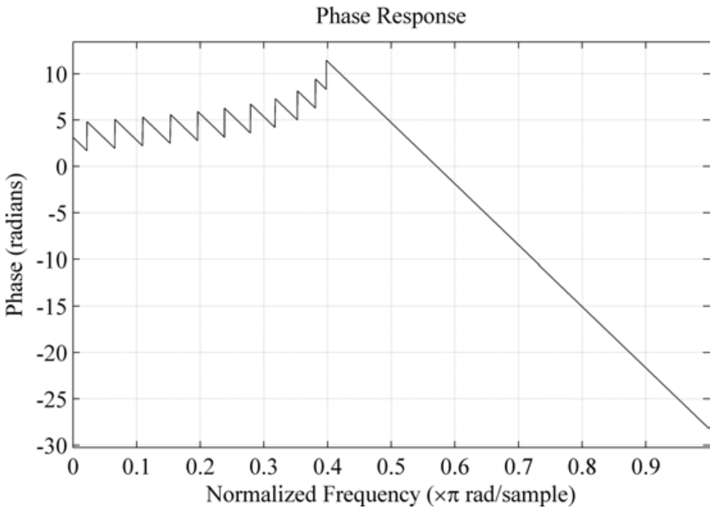


Fig. 2.10. HPF FIR phase response.

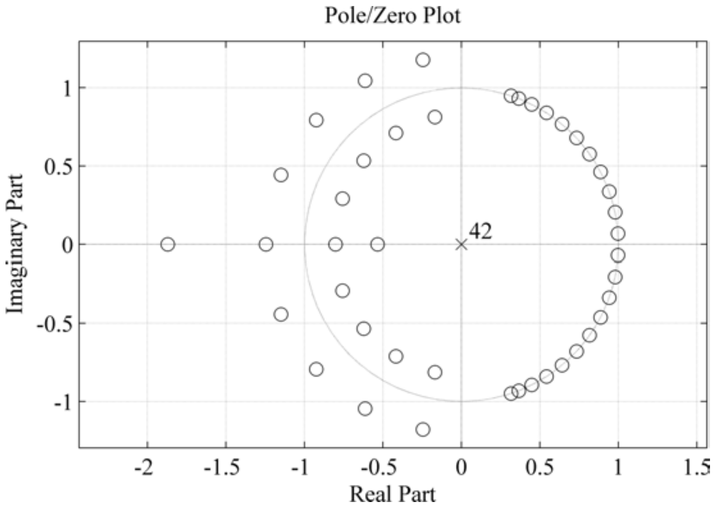


Fig. 2.11. HPF zero plot of 42 zeros.

For the band-pass filter case, we have a pair of bandpass regions, passband frequency edges, stopband frequency edges, and a single passband region defined by A_{pass}. In essence, the sharpness of the bandpass region is defined by the passband and stopband frequency edges and passband tolerance parameters. The magnitude response, phase response, and the z-plane with 44 zeros are shown in Figs. 2.12, 2.13, and 2.14.

Another way to quantify the passband sharpness in a bandpass filter is via the so-called *quality factor*, *Q-factor* or simply Q. The Q-factor is

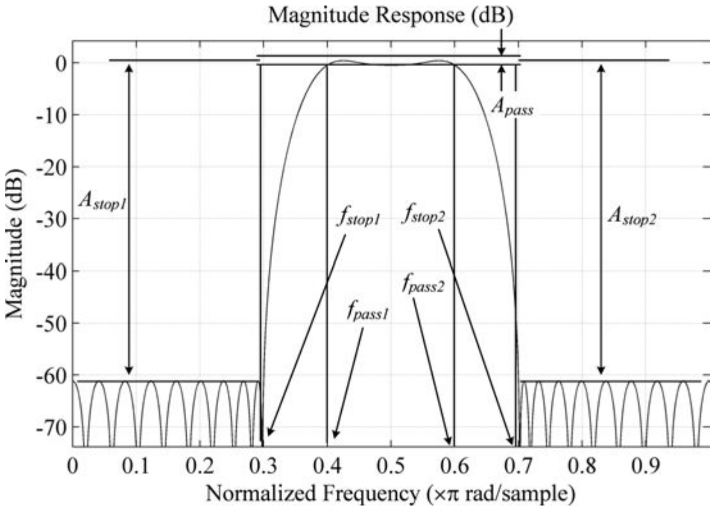


Fig. 2.12. BPF FIR magnitude response.

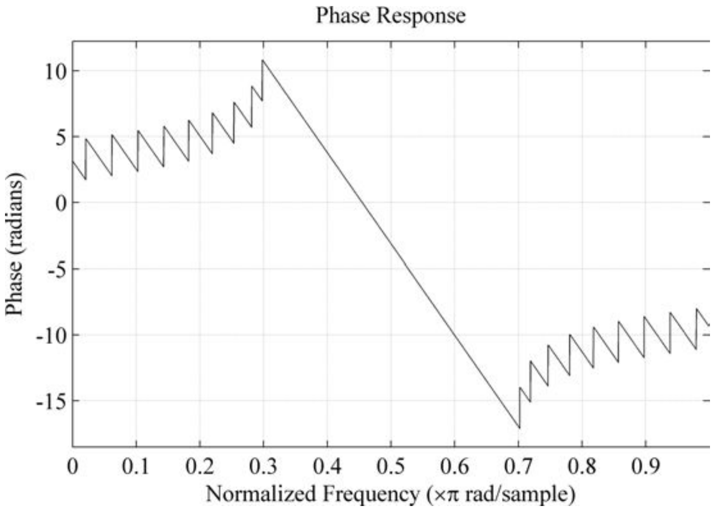


Fig. 2.13. BPF FIR phase response.

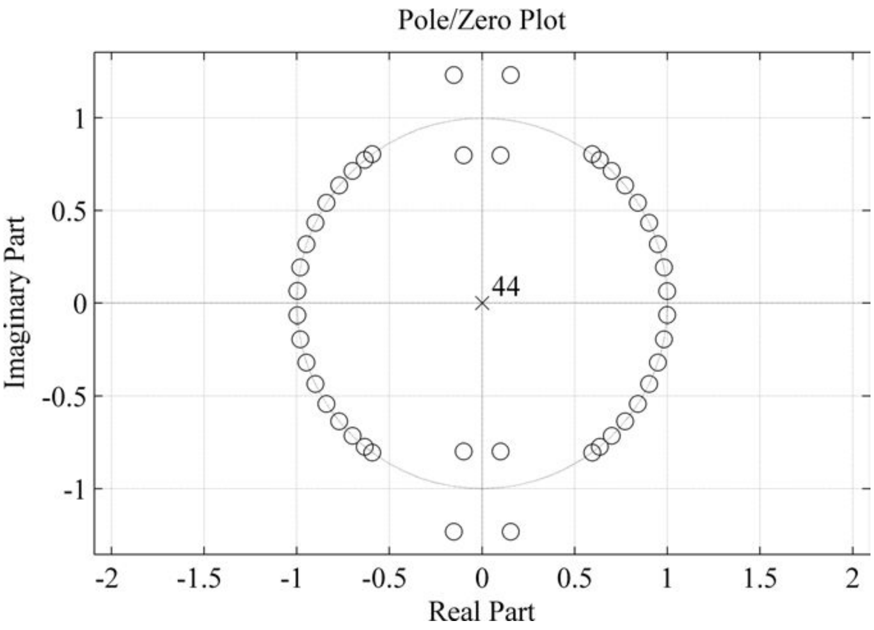


Fig. 2.14. BPF FIR zero plot with 44 zeros.

defined as follows:

$$Q = \frac{f_0}{\Delta f} \tag{2.4}$$

$$\Delta f = f_2 - f_1 \tag{2.5}$$

Equation (2.4) basically says that a high Q filter produces a narrow bandwidth or conversely a narrow bandwidth Δf produces a high Q. The narrowing/broadening is centered on the center frequency f_0 with Δf determining the bandwidth. Δf is defined as the $-3\,\text{dB}$ frequency boundary of the passband region as depicted in Fig. 2.15.

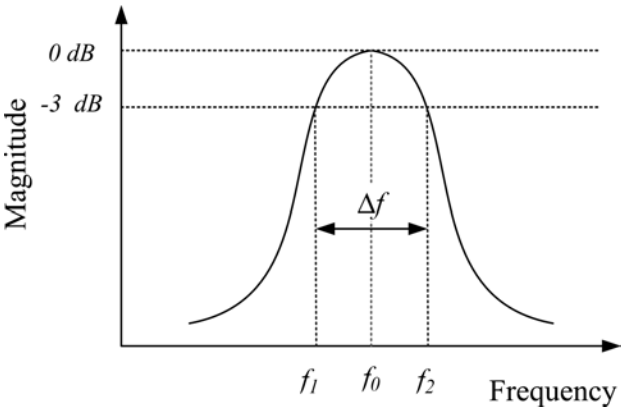


Fig. 2.15. Q-factor and bandwidth.

The parameters for the band-stop filter are shown in Fig. 2.16. It depicts a FIR-based band-pass filter with 40 zeros. The phase response and the z-plane show the layout of the 40 zeros as seen in Figs. 2.17 and 2.18 respectively.

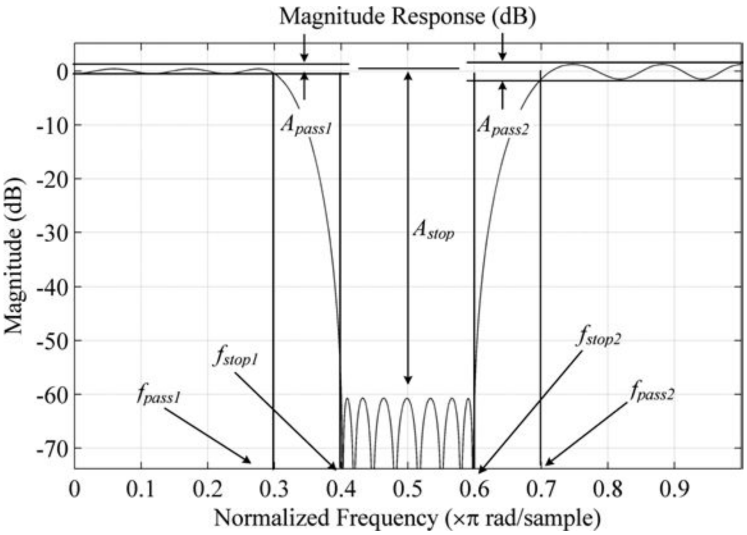


Fig. 2.16. BSF FIR magnitude response.

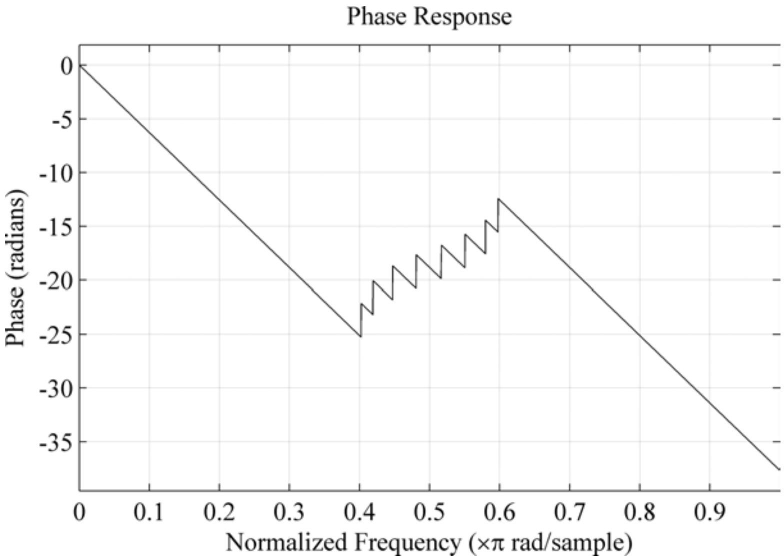


Fig. 2.17. BSF FIR phase response.

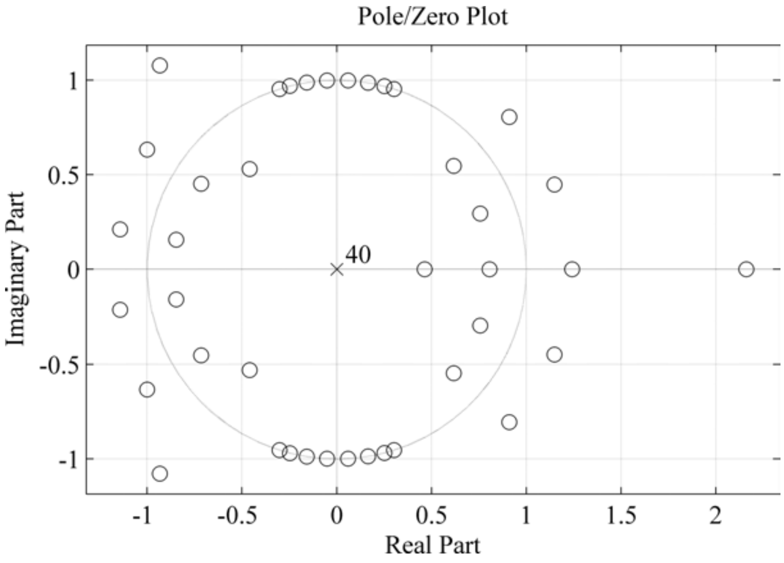


Fig. 2.18. BSF FIR phase response with 40 zeros.

There are a number of standard methods in designing filters — computing parameters in accordance to the desired filter specification which include among others, Butterworth, Chebychev, and elliptic/Cauer filter design algorithms (these will be briefly discussed in the next section). Butterworth filters are often popular in music editing type situations as they are very flat in the passband and stopband regions. The Chebychev I filter design method on the other hand can give the user flexible control over the ripple behavior in the passband and stopband regions. This is shown in Figs. 2.19 and 2.20 where an IIR low-pass filter is plotted. Figure 2.20 shows the zoomed-in passband portion of Fig. 2.19 and the corresponding z-plane with 10 zeros and poles in Fig. 2.21. An example using the elliptical filter design method is also shown in Fig. 2.22. The elliptical filter design method allows for *equiripple* (same ripple characteristics) in both passband and stopband regions and generally meets filter requirements with the lowest filter order.

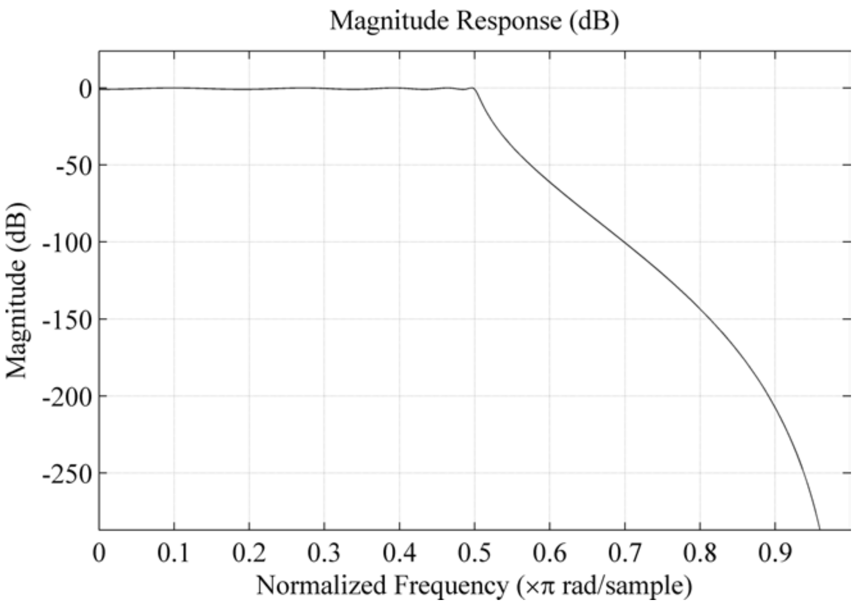


Fig. 2.19. Chebychev type I LPF (IIR).

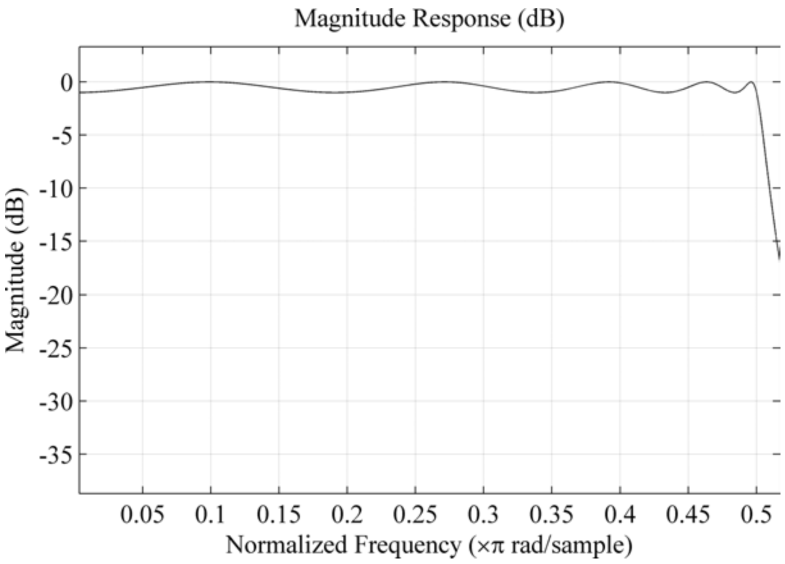


Fig. 2.20. Chebychev type I LPF (IIR) zoomed-in view of ripples.

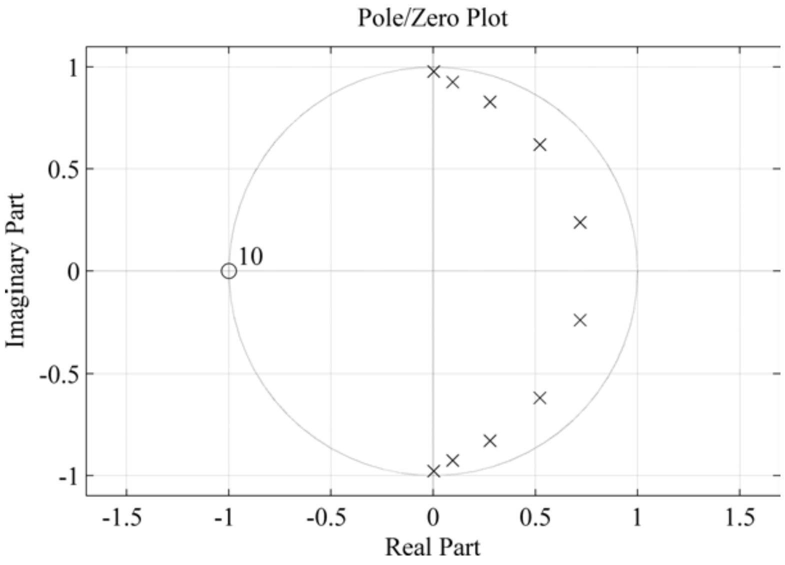


Fig. 2.21. Chebychev type I LPF (IIR) zero and pole plot.

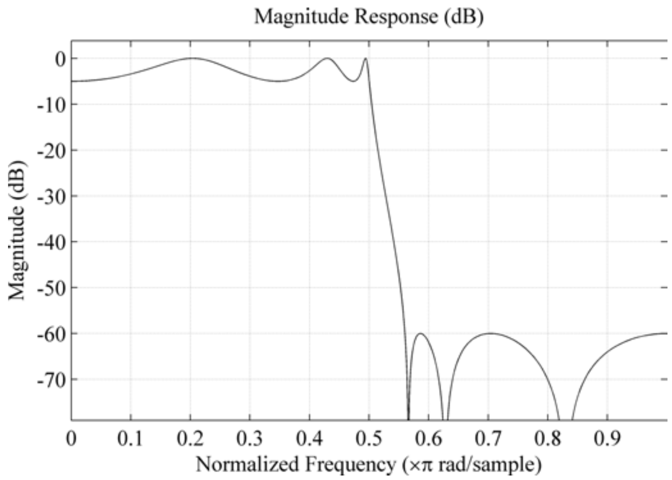


Fig. 2.22. Elliptical filter LPF, equiripple.

Figures 2.23 and 2.24 show the implementation of a filter via FIR and IIR filter design techniques using the equiripple method with the same specifications where $f_{pass} = 0.5 (= f_s \cdot 0.5)$, $f_{stop} = 0.51$, $A_{pass} = 1\,\text{dB}$, and $A_{stop} = 60\,\text{dB}$. Figures 2.25 and 2.26 show the respective zero and pole structure for both filters. We can clearly see that the IIR filter has smaller numbers of zeros and poles with similar stopband attenuation to the FIR filter.

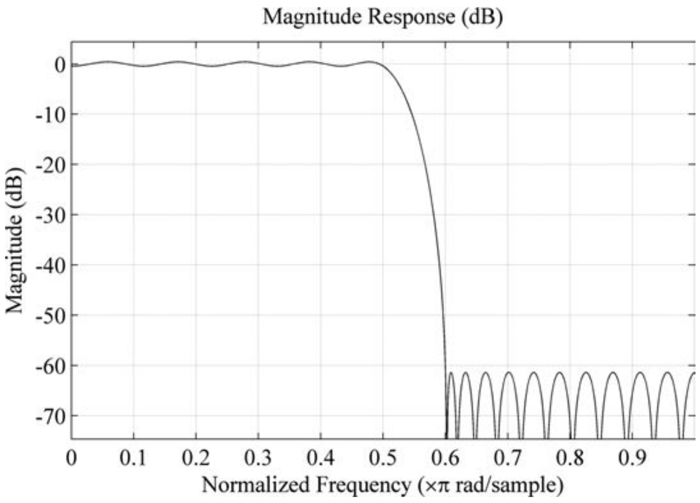


Fig. 2.23. FIR LPF filter.

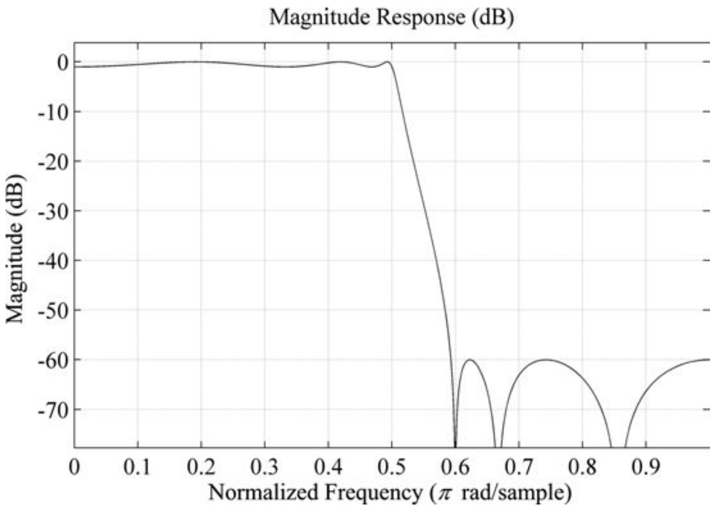


Fig. 2.24. IIR LPF filter.

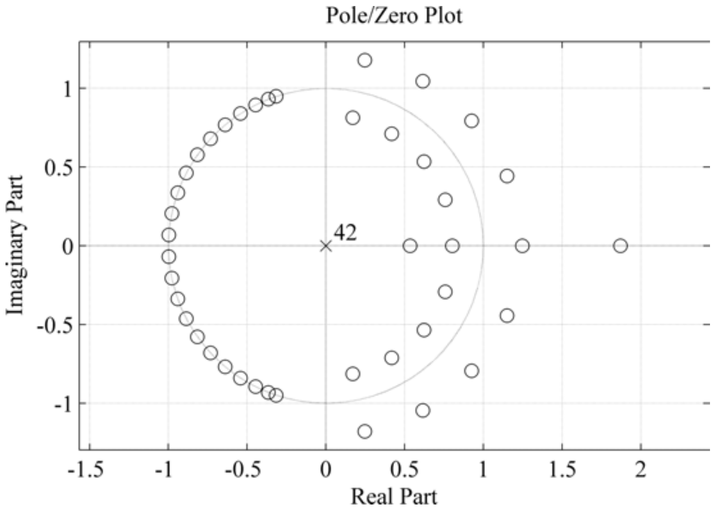


Fig. 2.25. FIR LPF zeros and poles.

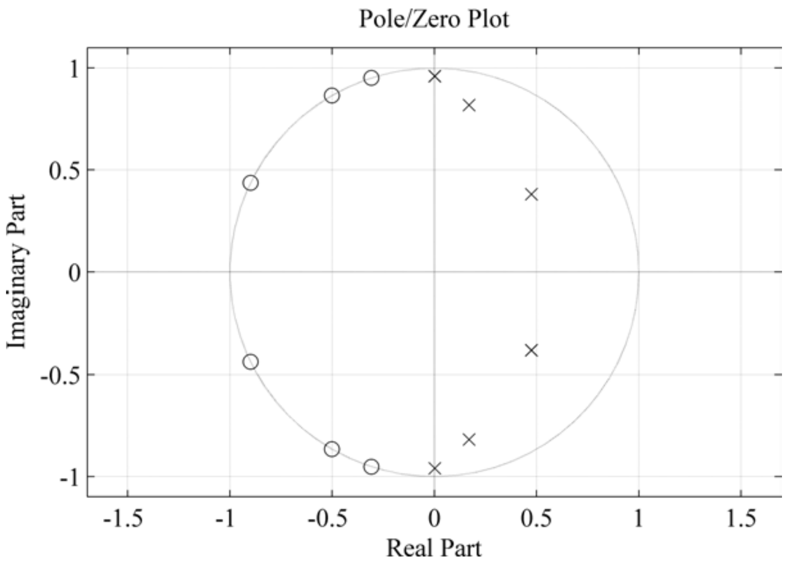


Fig. 2.26. IIR LPF zeros and poles.

We vividly recognize that although both filters adhere to the same specifications, the IIR filter is much shorter in length at 6 poles and 6 zeros whereas the FIR filter requires 42 zeros.

2.5 *MATLAB® filter design tools*

As mentioned at the onset of this chapter, we will not discuss the ins and outs of designing FIR/IIR filters *per se*, as there are well-developed and sophisticated computer programs and many standard DSP books readily available, detailing the various algorithms and procedures. We will, however, introduce a number of MATLAB® functions that can be used for this purpose, after briefly discussing some of the common approaches for filter design.

For designing FIR filters, that is, filters without feedback components, the two main methods are the *impulse response truncation* (IRT) *method* and *windowing method.* The IRT method (Porat 1997) is one of the simplest ones and is based on truncating an infinite impulse response (ideal filters have infinite impulse response characteristics). Although it is simple in structure, it has some adverse effects and undesirable frequency-domain characteristics, rendering it somewhat inadequate when used by itself. On the other hand, the windowing method is more practical and can be used to alleviate some of the artifacts of the IRT method and (as the name implies) is based on choosing an appropriate window type for a desired filter design (we will discuss windows in more detail in Chap. 8 and introduce their various characteristics viewed in the frequency-domain). The general rule of thumb with filters is that the higher the order, the steeper the filter — but with a higher order filter, the longer the delay. However, even when using the windowing method for FIR filter design, the filter order that results may not necessarily be the minimum possible and hence not guaranteed to be optimal. Thus, additional design methods such as the *least-squares design* and *equiripple design* methods are commonly used to arrive at optimal filter orders. Equiripple design can be used to control the maximum error for the frequency bands of interest while constraining the filter's order. Interestingly enough, for IIR filter design, the most common design approach is actually designing an analog IIR filter first and then transforming it to an equivalent digital filter. Some of the most popular classes of filters in the analog world include Butterworth, Chebychev, and elliptic filters.

Table 2.2 summarizes some of the functions available in the MATLAB® *Signal Processing Toobox.* For each of the functions, if it is an IIR filter design function, it will return the filter coefficients corresponding to the numerator and denominator of $H(\cdot)$, whereas if it is a FIR type, only the numerator coefficients will be returned as shown below and introduced in Chap. 6.

$$H(e^{j\theta}) = \frac{\sum_{k=0}^{k=L} b_k e^{-j\theta k}}{1 - \sum_{k=1}^{k=N} a_k e^{-j\theta k}} \tag{2.6}$$

One thing to watch out for in MATLAB® is that the coefficients a_k are represented as positive weights. That is, Eq. (2.6) actually looks like (2.7) in MATLAB® and hence caution is advised in using the right polarity for the denominator coefficients. Also, note that a_0 will always be equal to 1 — compare (2.7) and (2.6).

$$H(e^{j\theta}) = \frac{b_0 + b_1 e^{-j\theta} + \cdots + b_L e^{-j\theta L}}{a_0 + a_1 e^{-j\theta} + \cdots + a_N e^{-j\theta N}} \tag{2.7}$$

Looking at the coefficients from a difference equation point of view, be aware that in MATLAB®, the difference equation will by default assume that the feedback coefficients are negative rather than positive as shown in (2.8).

$$\begin{aligned} a_0 \cdot y[n] &= b_0 \cdot x[n] + b_1 \cdot x[n-1] + \cdots + b_L \cdot x[n-L] \\ &\quad - a_1 \cdot y[n-1] - \cdots - aN \cdot y[n-N] \end{aligned} \tag{2.8}$$

You will also note that in Table 2.2 there are parameters/arguments listed for each function. In general, for FIR filters, only the feed-forward coefficients exist, whereas for the IIR feedback filters, feed-forward and feedback coefficients need to be defined. Each function will return two vectors B and A corresponding to the numerator and denominator coefficients for (2.7) respectively. *Wn* defines the cutoff frequency and is represented in normalized frequency (π x radians/sample) — the digital frequency. Thus, a *Wn* at 0.5 is equivalent to $\pi/2$ which in turn represents 2 kHz for a f_s = 8 kHz system. You will also have observed that there are functions that output the optimal order for a given IIR filter — *BUTTORD*(·), *CHEBYORD1*(·), *CHEBYORD2*(·). These will return the optimal order number N when the user specifies the passband and stopband frequencies (*Wp* and *Ws*) and the corresponding passband and stopband ripple specifications (*Rp* and *Rs*). Finally, in *FIR2* (·) we see some interesting parameters that are not seen anywhere else, namely F and A. These two vectors directly determine the filter shape where *F* represents the frequency points and *A* the corresponding magnitude response. Thus, the MATLAB® command *PLOT* (*F*, *A*) will literally plot the shape of the filter itself. As usual, the frequency is represented as the digital frequency normalized between 0.0 and 1.0 where 1.0 corresponds to half the sampling rate.

Another very useful MATLAB® function is the *FILTER*(*B,A,X*) function. The *FILTER*(*B,A,X*) function basically renders the output by filtering the input *X* with filter coefficients *B* and *A*. An example is shown in the next section using some of the functions presented here.

Table 2.2. MATLAB® functions for filter design.

Function name	Type	Description
FIR1 (*N, Wn*)	FIR	Designs *N*th order filter. You can also use a 3rd parameter (*'high', 'stop', 'bandpass'*) to designate the filter type.
FIR12 (*N, F, A*)	FIR	Designs an arbitrarily shaped filter using windowing.
BUTTER (*N, Wn*)	IIR	Designs *N*th order Butterworth filter. You can also use a 3rd parameter (*'high', 'low', 'stop'*) to designate the filter type. If *Wn* is two-element vector *Wn = [W1 W2]* it can be also used as a BPF.
BUTTORD (*Wp, Ws, Rp, Rs*)	IIR	Returns the lowest Butterworth filter order *N* that loses no more than *Rp* dB in the passband and at least *Rs* dB attenuation in the stopband.
CHEBY1 (*N,R,Wn*)	IIR	Designs *N*th order Chebychev Type I filter with *R* dB peak-to-peak ripple where you can also use a 4th parameter (*'high', 'stop', 'stop'*) to designate the filter type. Two-element vector for *Wn* can also be used.
CHEB1ORD (*Wp, Ws, Rp, Rs*)	IIR	Returns the lowest Chebychev Type I filter order *N* that loses no more than *Rp* dB in the passband and at least *Rs* dB attenuation in the stopband.
CHEBY2 (*N, R, Wn*)	IIR	Designs *N*th order Chebychev Type II filter with *R* dB peak-to-peak ripple where you can also use a 4th parameter (*'high', 'stop', 'stop'*) to designate the filter type. Two-element vector for *Wn* can also be used.
CHEB2ORD (*Wp, Ws, Rp, Rs*)	IIR	Returns the lowest Chebychev Type II filter order *N* that loses no more than *Rp* dB in the passband and at least *Rs* dB attenuation in the stopband.
ELLIP (*N, Rp, Rs, Wn*)	IIR	Designs *N*th order Elliptical filter with Rp dB peak-to-peak passband ripple and a minimum of *Rs* dB stop attenuation where you can also use a 4th parameter (*'high', 'stop', 'stop'*) to designate the filter type. Two-element vector for *Wn* can also be used.
ELLIPORD (*Wp, Ws, Rp, Rs*)	IIR	Returns the lowest Elliptical filter order *N* that loses no more than *Rp* dB in the passband and at least *Rs* dB attenuation in the stopband.

3 Filter Examples

In this section, we will present a number of interesting filter topics including subtractive synthesis, fractional delays, comb-filters, physical modeling and the plucked string, and all-pass filters. These topics are indeed very intriguing and form the backbone of many applications in computer music of the past and present.

3.1 *Subtractive synthesis and filters*

Subtractive synthesis debuted in the 1960s in the shape of the musical synthesizer keyboard with the widely popular Moog synthesizer. Simply put, it is based on filtering a *rich* sound source whereby making the resulting filtered sound less rich. This may seem a bit strange as we are striving to make a sound less rich, but it so happens that sounds considered generally to be musical have a tendency to be structured and hence less rich and generally less complex than typical "noisy" signals. On a scale of richness, we can loosely consider white noise the most complex and rich, while a perfect oscillator least rich. In subtractive synthesis, source signals often in the form of noise are commonly used as starting points for non-pitched timbres such as percussive sounds like drums, whereas square, sawtooth, and triangular waves are more commonly used to carve out pitched timbres. In a way, subtractive synthesis could be likened the process of sculpturing — beginning a project with a block of shapeless granite, chipping and carving away on the amorphous object until a more defined characteristic object emerges. A typical subtractive synthesis setup is depicted in Fig. 3.1.

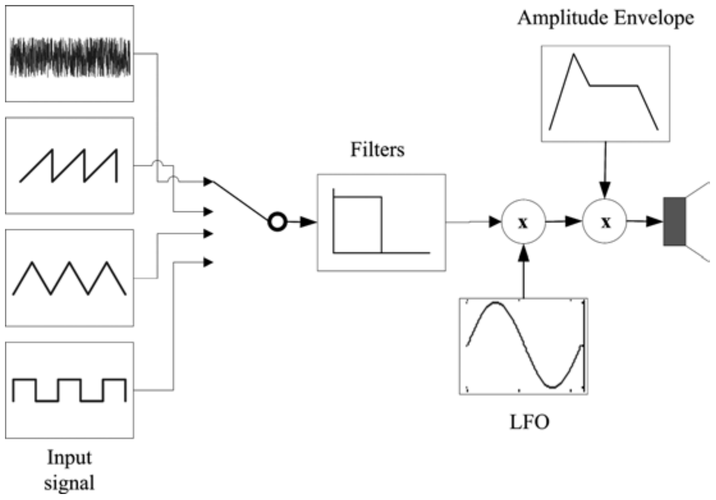


Fig. 3.1. Typical subtractive synthesis system.

It shows an amplitude envelope generator which we discussed in Chap. 2, and low frequency oscillator (LFO) which further imposes low frequency modulations such as amplitude modulation and pitch modulation to the resulting signal.

Below is some MATLAB® code implementing a very basic subtractive synthesis algorithm using the *RAND*(·) function to generate the source signal (noise) and using Butterworth filter design tools to implement a band-pass filter. I have also included an exponentially decaying amplitude envelope to make the filtered noise behave even more like a musical sound. In this example, I opted to use a low-pass and high-pass filter combination to implement a band-pass filter.

```
fs = 44100;
x = rand (fs,1);                      % make random noise
x = x - mean(x);                      % take out DC

% make band-pass filter using LPF and BPF
[b1, a1] = butter (10, 0.5);
[b2, a2] = butter (10, 0.4, 'high');

% filter using above coefficients
y1 = filter (b1,a1,x);
y2 = filter (b2,a2,y1);

% envelope the filtered signal
env = 1./exp ((0:(length(y2)-1))/fs).^10;

% play the signal
sound (env'.* y2, fs)
```

Code Example 3.1. Basic subtractive synthesis example using filters.

In the above example, the *RAND*(·) function generates 44,100 samples or a 1 second signal ($f_s = 44.1\,\text{kHz}$). On the 3rd line, the DC part of the signal is taken off so that *x* oscillates at the "$y = 0$" axis. The *BUTTER*(·) filter design function produces a 10th order (10 delay components) low-pass and high-pass filter with cutoff frequencies at 0.5π and 0.4π respectively. It thus implements a serial band-pass filter when combined. The coefficients are then used to filter the signal by first low-passing it followed by high-passing the result of the 1st filter. Finally, the filtered signal is enveloped to give it a more characteristic amplitude envelope consequently rendering a more realistic musical quality.

3.2 *Bi-quadratic filter* (*a.k.a. Bi-quad filter*) *and the Wah-Wah filter*

A simple but great filter example in musical application is the wah-wah filter. This filter is commonly used by electric guitarists and sometimes by electric bassists as well as by keyboard players. You may have heard this in guitar licks by Jimmy Hendrix or a plethora of funky rhythm guitar grooves. One of the most popular guitar pedals (little metallic boxes you step on to control effects for your instrument) is the *Jim Dunlop Cry Baby* and as you may have guessed, it is a wah-wah pedal designed for the guitar player. The reason it is called the wah-wah pedal (or stomp box in general, as musicians refer to it) is that when an audio signal such as the electric guitar is passed through it, the filter makes it go "wah — wah — wah ..." (wah-wah ... cry baby ...). Usually the "wah-wah-ness" or "wwwaaahhh-wwwaaaahhh-ness" is controlled manually with a potentiometer (also known as the foot-pedal) but auto-wah-wah pedals also exist which automatically renders the wah-wah effect without having to manually produce it with your foot. The concept behind this popular effect is actually remarkably straightforward, although the timbral and musical results very interesting and perhaps even complex sounding. The theory behind the wah-wah is implemented by merely varying the center frequency of a band-pass filter — and that's it. Pretty simple huh? One way to implement a dynamically changing band-pass filter (changing the center frequency) is using the popular *bi-quad* filter. The bi-quadratic filter is basically a two-pole and two-zero IIR filter that can be flexibly used to implement high-pass, low-pass, band-pass, and band-stop filters. In this section, we will see how it can be used to implement a band-pass filter to construct the wah-wah effect.

The origin of the name itself — bi-quadratic or its shorthand equivalent bi-quad filter, can be traced to the transfer function. That is, the typical difference equation for a bi-quad filter resembles Eq. (3.1) and its corresponding transfer function (3.2).

$$y[n] = x[n] + a_1 \cdot x[n-1] + a_2 \cdot x[n-2] - b_1 \cdot y[n-1] - b_2 \cdot y[n-2] \tag{3.1}$$

$$H(z) = \frac{1 + a_1 \cdot z^{-1} + a_2 \cdot z^{-2}}{1 + b_1 \cdot z^{-1} + b_2 \cdot z^{-2}} \tag{3.2}$$

Equation (3.2) can of course be rewritten in the more common form by multiplying z^2 to the numerator and denominator resulting in Eq. (3.3).

$$H(z) = \frac{z^2 + a_1 \cdot z + a_2}{z^2 + b_1 \cdot z^1 + b_2} \tag{3.3}$$

We can see in Eq. (3.3) that both the numerator and denominator are in the form of quadratic polynomials (quadratic equations refer to polynomial equations of the second degree) — two quadratic polynomials or bi-quads and hence its name. Now, let's try to put the above transfer function into a more compact form by finding the roots of the numerator and denominator. We know that the roots of the numerator and denominator take on the usual form shown below in (3.4) and (3.5) respectively.

$$z = \frac{-a_1 \pm \sqrt{a_1^2 - 4 \cdot a_2}}{2} \tag{3.4}$$

$$\begin{aligned} z &= \frac{-b_1 \pm \sqrt{b_1^2 - 4 \cdot 1 \cdot b_2}}{2 \cdot 1} = \frac{-b_1 \pm \sqrt{b_1^2 - 4 \cdot b_2}}{2} \\ &= \frac{-b_1}{2} \pm \sqrt{\left(\frac{b_1}{2}\right)^2 - b_2} \end{aligned} \tag{3.5}$$

Let us concentrate on the 2nd order polynomial roots of the denominator, namely the poles which determine the resonant characteristics of the bi-quad filter. The general solution of the above equations can be represented in terms of real plus imaginary components as seen in Eq. (3.6). For real valued roots, there would obviously be no imaginary components.

$$z = A \pm jB \tag{3.6}$$

If the roots require an imaginary axis (inside of the square root is negative), they will be represented as conjugate pairs as discussed in Chap. 6 and seen in Sec. 2.4 earlier in this chapter. Equation (3.6) can also be represented in polar form using the Euler identity as follows:

$$z = R \cdot e^{\pm j\theta} \tag{3.7}$$

R is the Euclidian distance defined by the real and imaginary components A and B as seen below with the phase shown in Eq. (3.9).

$$R = \sqrt{A^2 + B^2} \tag{3.8}$$

$$\theta = \tan^{-1} \frac{B}{A} \tag{3.9}$$

Remembering from Chap. 6 that the poles need to stay within the unit circle for stability, we make sure of the following:

$$R < 1 \tag{3.10}$$

The same approach can be taken for the numerator corresponding to the zeros on the z-plane. Thus, we can now represent the transfer function in Eq. (3.3) using the Euler identifies as shown in (3.11) where the subscript zr refers to the zeros and p to poles. We also find that the imaginary components conveniently cancel each other out in penultimate step of Eq. (3.11).

$$\begin{aligned} H(z) &= \frac{(z - R_{zr} \cdot e^{j\theta_{zr}}) \cdot (z - R_{zr} \cdot e^{-j\theta_{zr}})}{(z - R_p \cdot e^{j\theta_p}) \cdot (z - R_p \cdot e^{-j\theta_p})} \\ &= \frac{z^2 - R_{zr} \cdot e^{-j\theta_{zr}} \cdot z - R_{zr} \cdot e^{j\theta_{zr}} \cdot z + R_{zr}^2 \cdot e^{j\theta_{zr}} \cdot e^{-j\theta_{zr}}}{z^2 - R_p \cdot e^{-j\theta_p} \cdot z - R_p \cdot e^{j\theta_p} \cdot z + R_p^2 \cdot e^{j\theta_p} \cdot e^{-j\theta_p}} \\ &= \frac{z^2 - R_{zr} \cdot e^{-j\theta_{zr}} \cdot z - R_{zr} \cdot e^{j\theta_{zr}} \cdot z + R_{zr}^2 \cdot 1}{z^2 - R_p \cdot e^{-j\theta_p} \cdot z - R_p \cdot e^{j\theta_p} \cdot z + R_p^2 \cdot 1} \\ &= \frac{z^2 - R_{zr} \cdot (\cos\theta_{zr} - j\sin\theta_{zr}) \cdot z - R_{zr} \cdot (\cos\theta_{zr} + j\sin\theta_{zr}) \cdot z + R_{zr}^2}{z^2 - R_p \cdot (\cos\theta_p - j\sin\theta_p) \cdot z - R_p \cdot (\cos\theta_p + j\sin\theta_p) \cdot z + R_p^2} \\ &= \frac{z^2 - 2 \cdot R_{zr} \cdot \cos\theta_{zr} \cdot z + R_{zr}^2}{z^2 - 2 \cdot R_p \cdot \cos\theta_p \cdot z + R_p^2} \end{aligned} \tag{3.11}$$

The results of (3.11) can then be rewritten into the form shown in (3.12) by dividing the numerator and denominator by z^2 or multiplying by z^{-2}.

$$\begin{aligned} H(z) &= \left(\frac{z^2 - 2 \cdot R_{zr} \cdot \cos\theta_{zr} \cdot z + R_{zr}^2}{z^2 - 2 \cdot R_p \cdot \cos\theta_p \cdot z + R_p^2} \right) \cdot \left(\frac{z^{-2}}{z^{-2}} \right) \\ &= \frac{1 - 2 \cdot R_{zr} \cdot \cos\theta_{zr} \cdot z^{-1} + R_{zr}^2 \cdot z^{-2}}{1 - 2 \cdot R_p \cdot \cos\theta_p \cdot z^{-1} + R_p^2 \cdot z^{-2}} \end{aligned} \tag{3.12}$$

We can now eyeball the difference equation from the above transfer function where the feed-forward components (numerator) and the feedback components (denominator) result in the following two-tap IIR system:

$$\begin{aligned} y[n] = x[n] &- 2 \cdot R_{zr} \cdot \cos\theta_{zr} \cdot x[n-1] + R_{zr}^2 \cdot x[n-2] \\ &+ 2 \cdot R_p \cdot \cos\theta_p \cdot y[n-1] - R_p^2 \cdot y[n-2] \end{aligned} \tag{3.13}$$

We note that this bi-quad difference equation is actually a function of both the radii R_{zr}/R_p and frequency θ_{zr}/θ_p. Also, upon closer scrutiny, we observe that the second tap's coefficients $(n-2)$ are independent of θ for both the feedback and feed-forward components. This has some important implications as we can tweak the frequency response and the system's resonant structure by changing R_p while keeping θ_p unaltered. The digital frequencies θ_p and θ_{zr} are the pole and zero frequencies respectively and from Chap. 6, we remember that they have the relationship shown in (3.14) for sampling period T $(1/f_s)$. Thus, we can define the frequency f as the center frequency (resonant frequency) commonly denoted as f_o as shown in Eq. (3.15) (we used θ_p instead of θ_o so that there is no confusion between zeros and poles).

$$\theta = \omega \cdot T = 2 \cdot \pi \cdot f \cdot T \tag{3.14}$$

$$f_o = \frac{\theta_p}{2 \cdot \pi \cdot T} \tag{3.15}$$

In general, moving R_p towards 1 increases the gain at a particular resonant frequency f_o (the absolute value of $|R_p|$ has to be smaller than 1.0 in order for this filter to be stable). We also note that R_p simultaneously acts as a dampening coefficient, and when $R_p = 0$, the bi-quad filter becomes an FIR filter due to the absence of feedback components.

Since zeros can be regarded to generally have anti-resonance characteristics (when inside the unit circle), having a pulling effect of the frequency response contour (see Chap. 6 for details) at the zero locations, one common method of making the bi-quad a band-pass filter is by placing the two zeros at DC ($z = 0$) and Nyquist ($z = -1$) on the z-plane (real axis). This is shown in Fig. 3.2.

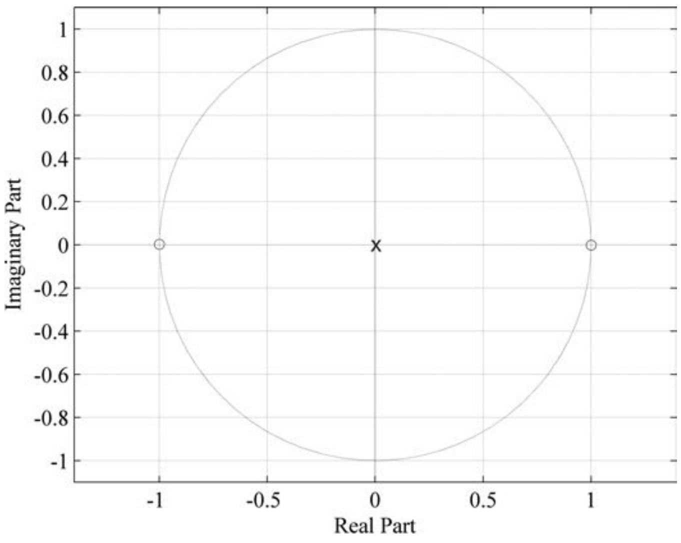


Fig. 3.2. Bi-quad's zero locations acting as a band-pass filter.

Thus, in implementing the band-pass filter via the strategic placing of the zeros, all we need to do is solve the 2nd order polynomial of the numerator of the transfer function by setting $z = 0$ and $z = -1$ [zero locations in Eq. (3.3)]. The solution for the numerator part of the transfer function is given by

$$G(z) = z^2 + a_1 \cdot z + a_2 \tag{3.16}$$

and the roots for the zeros are solved via Eq. (3.4) and setting $z = 1$ and $z = -1$ as shown again below for convenience.

$$z = \frac{-a_1 \pm \sqrt{a_1 - 4 \cdot a_2}}{2} \tag{3.17}$$

Consequently, we get two equations which is sufficient for solving for two unknowns a_1 and a_2.

$$z\big|_{z=1} = \frac{-a_1 + \sqrt{a_1 - 4 \cdot a_2}}{2} = 1 \tag{3.18}$$

$$z\big|_{z=-1} = \frac{-a_1 - \sqrt{a_1 - 4 \cdot a_2}}{2} = -1 \tag{3.19}$$

Rewriting Eqs. (3.18) and (3.19) and solving for a_1 and a_2 yields the following feed-forward coefficients.

$$-a_1 + \sqrt{a_1 - 4 \cdot a_2} = 2 \tag{3.20}$$

$$-a_1 - \sqrt{a_1 - 4 \cdot a_2} = -2 \tag{3.21}$$

Adding (3.20) and (3.21) yields (3.22) thus giving us a_1 as shown below.

$$-2 \cdot a_1 = 0 \rightarrow a_1 = 0 \tag{3.22}$$

Solving for a_2, by rearranging either Eq. (3.20) or (3.21) and plugging in a_1 $(= 0)$ we get $a_2 = -1$.

$$-a_1 + \sqrt{a_1 - 4 \cdot a_2} = 2$$

$$\sqrt{a_1 - 4 \cdot a_2} = 2 + a_1$$

$$a_1 - 4 \cdot a_2 = (2 + a_1)^2$$

$$a_2 = \frac{a_1 - (2 + a_1)^2}{4} \tag{3.23}$$

$$a_2\big|_{a_1=0} = \frac{0 - (2 + 0)^2}{4} = -1 \tag{3.24}$$

In terms of the transfer function, the feed-forward component is as shown in Eq. (3.25) and the resulting complete transfer function as shown in Eq. (3.26) when the numerator and denominator are divided by z^2 as before. The difference equation from inspection can be quickly determined from (3.26) and is revealed in Eq. (3.27).

$$G(z) = (z-1)(z+1) \tag{3.25}$$

$$H(z) = \frac{1 - z^{-2}}{1 - 2 \cdot R_p \cdot \cos\theta_o \cdot z^{-1} + R_p^2 \cdot z^{-2}} \tag{3.26}$$

$$y[n] = x[n] - x[n-2] + 2 \cdot R_p \cdot \cos\theta_o \cdot y[n-1] + R_p^2 \cdot y[n-2] \tag{3.27}$$

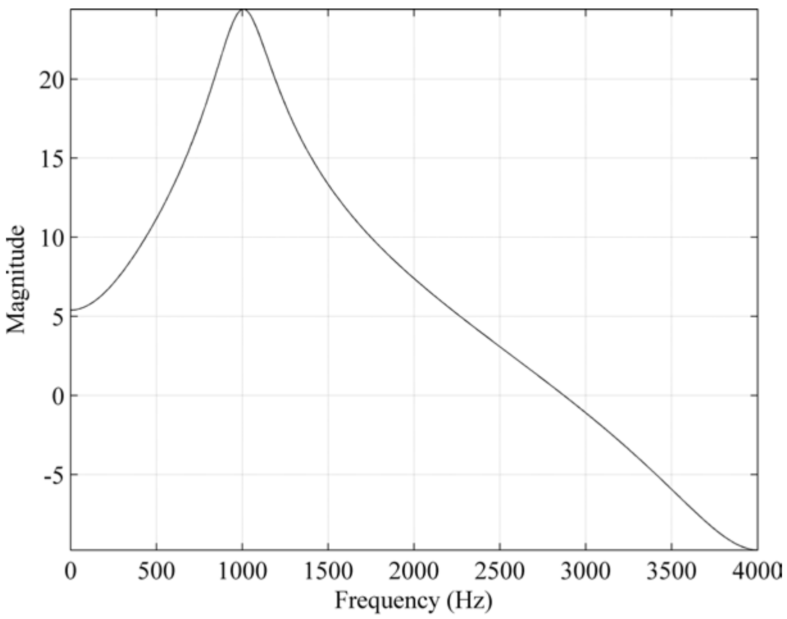
By placing the zeros at the 0 and $f_s/2$ frequency locations (DC and Nyquist), we can now use it as a dynamic band-pass filter as it allows us to vary the resonant frequency f_o in Eqs. (3.14) and (3.15) — and voilà we have our wah-wah bi-quad filter ready for some action. However, before we go off to code the bi-quad filter into our favorite computer language, there is one more interesting issue that needs to be addressed, namely the *Q-factor* issue. We will not go into a tangent and delve on the details of the Q-factor and its relationship to bandwidth which is best explained through Laplace transform analysis; but suffice it to say for now, R_p is used to control the sharpness of the BPF. That is, higher R_p values result in higher pole gain and narrower bandwidth (sharper tent) and vice-versa. The next four plots (Figs. 3.3 and 3.4) show a number of different f_o and R_p values with zeros at the DC and Nyquist locations obtained via the MATLAB® code shown below.

```
Rp = 0.98;            %  coeff. for individual
                      %  control over Q
f0 = 3000;            %  center frequency in Hertz
fs = 8000;            %  sampling frequency in Hertz
wT = 2*pi*f0*1/fs;    %  analog frequency in radians

% The difference equation coefficients
b = [1 0 -2];
a = [1 -2*Rp*cos(wT) Rp^2];

% frequency and phase response
freqz (b, a)
```

Code Example 3.2. Bi-quad band-pass filter.



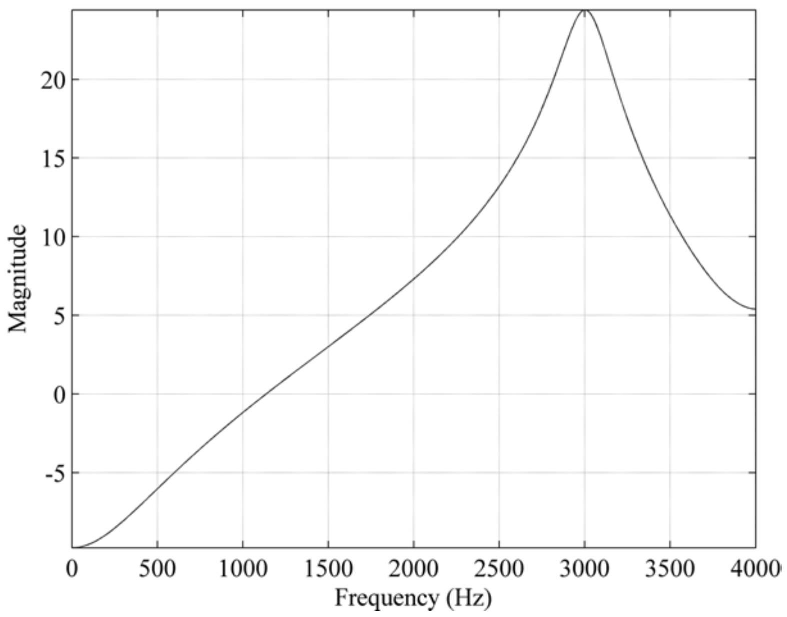
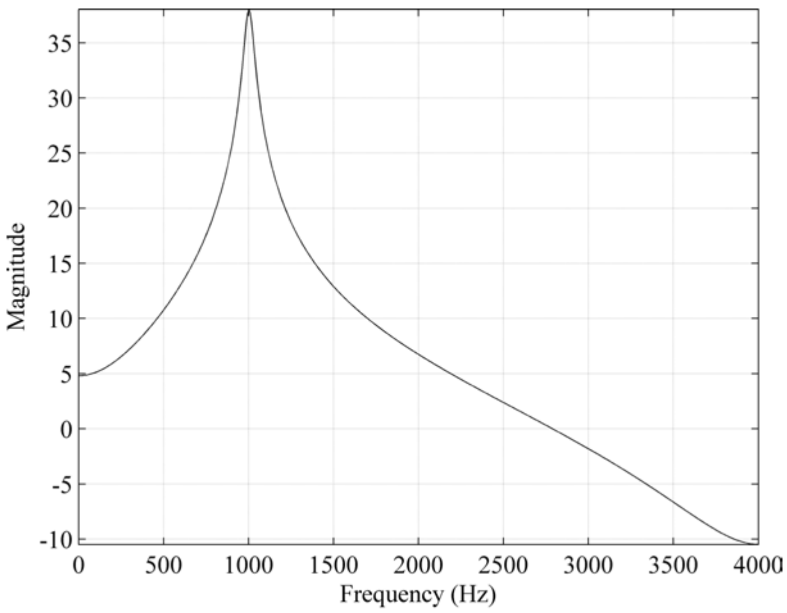


Fig. 3.3. Bi-quad band-pass filter with $R_p = 0.90$, $f_o = 1$ kHz (top) and 3 kHz (bottom).



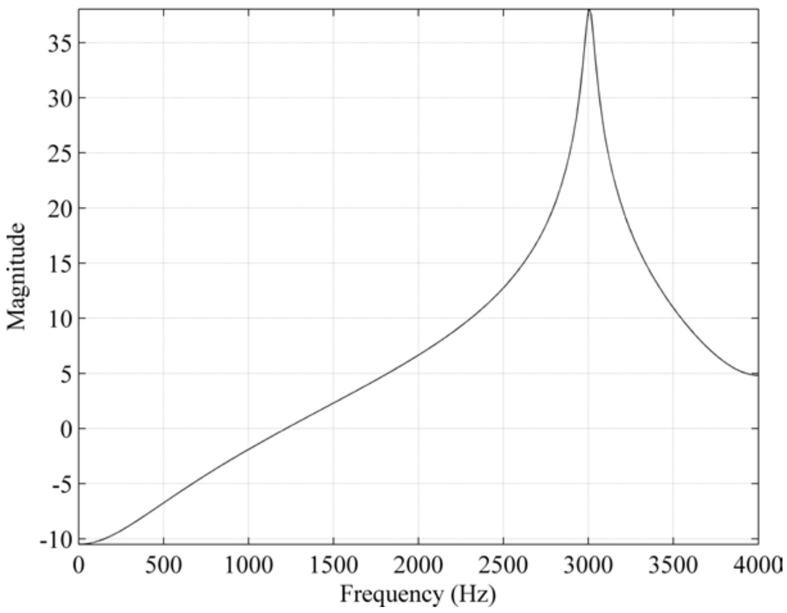


Fig. 3.4. Bi-quad band-pass filter with $R_p = 0.98$, $f_o = 1\,\text{kHz}$ (top) and $3\,\text{kHz}$ (bottom).

Another interesting effect somewhat similar to the wah-wah filter is the so-called phaser effect which is implemented not via a variable band-pass filter as we have seen with the wah-wah example, but via a dynamically changing band-stop (notch) filter. Notch filters actually have characteristics of strong phase shifts around the notch frequency areas and when the unprocessed signal is combined with the processed signal, various degrees of phase cancellations result. We could design a dynamic notch filter by placing zeros and poles so as to render a band-stop filter configuration as we did for the wah-wah effect and the band-pass filter. We will not present it here, as we have shown you the methods and tools to accomplish this task — sounds like a good project for a bank holiday.

3.3 *The Comb-filter*

A very interesting and musically effective filter is the so-called *comb-filter*. It is named the comb-filter due to shape of the magnitude response as seen in Fig. 3.5 — it resembles a hair comb. There are two ways of implementing the comb-filter — via FIR and IIR designs as is usually the case for any filter. The comb-filter's feed-forward difference equation and its frequency

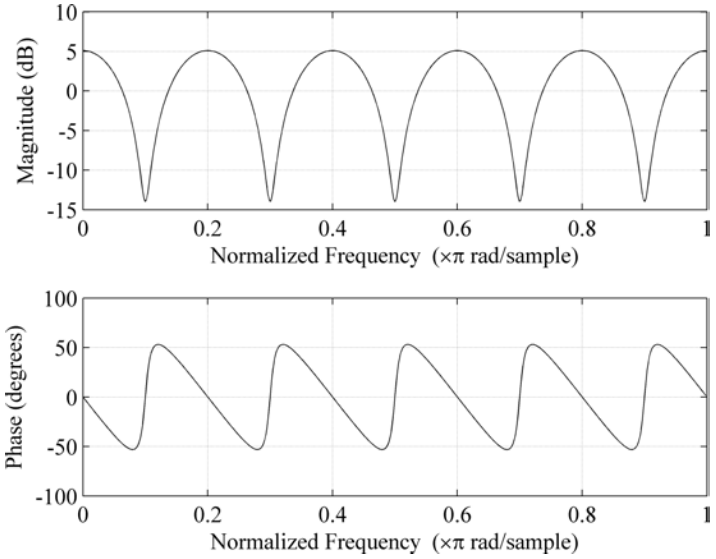


Fig. 3.5. FIR comb-filter with $L = 10$ and $b_1 = 0.8$ with magnitude response (top) and phase response (bottom) (MATLAB® code *FREQZ*([*1 ZEROS*(*1, 8*) *0.8*]).).

response are shown in Eq. (3.28) and Fig. 3.5 where L is the amount of delay.

$$y[n] = x[n] + b_1 \cdot x[n - L] \tag{3.28}$$

The difference equation for the IIR version of the comb-filter is shown in Eq. (3.29) and Fig. 3.6. As expected, the sharpness of the peaks are more pronounced due to the resonant characteristics of the poles in the IIR version.

$$y[n] = x[n] + a_1 \cdot y[n - L] \tag{3.29}$$

In Fig. 3.7, we can see a range of different coefficient values for the FIR and IIR comb-filters with the delay length L set to 11 samples. We can clearly observe that the sharpness of the valleys (zeros) and peaks (poles) are a direct function of the filter coefficients — the closer the coefficients are to the unit circle, the more pronounced the peaks become and vice-versa for both FIR and IIR versions of the comb-filter. Let's now take a closer look at the comb-filter and try to interpret it with the tools we have learned so far.

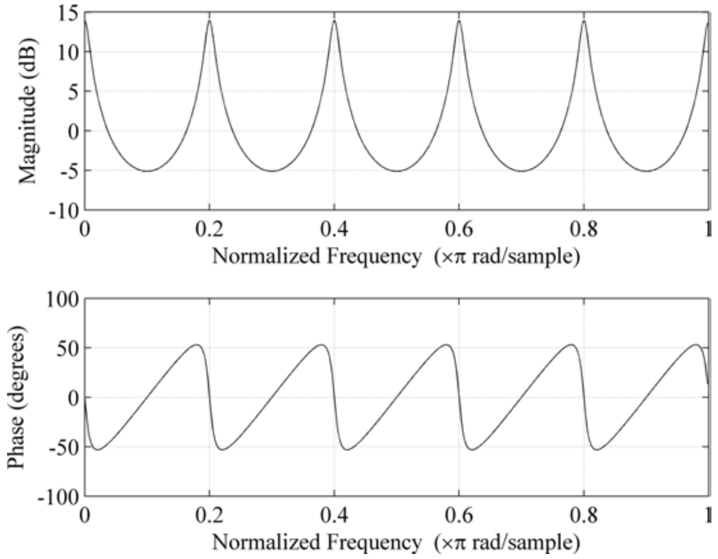


Fig. 3.6. IIR comb-filter with $L = 10$ and $b_1 = 0.8$ with magnitude response (top) and phase response (bottom).

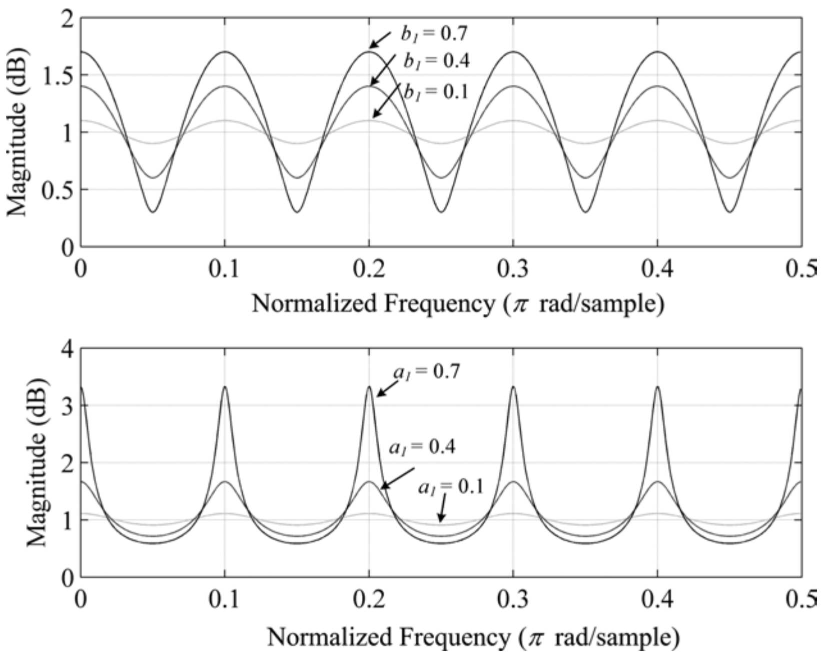


Fig. 3.7. Frequency response of comb-filter with coefficients from 0.1 to 0.9 in 0.3 increments. Top is FIR and bottom is IIR version.

3.3.1 *Comb-filter interpretation*

Just by looking at the FIR and IIR difference equations, we can somewhat see how we get a frequency response as seen in Figs. 3.6 and 3.7. That is, in Eqs. (3.28) and (3.29), every Lth delayed sample is accentuated by adding its "Lth presence" to the output $y[n]$. Unlike the FIR version, however, the IIR version keeps an attenuated echo of the input samples in its feedback buffer delay line which gets accentuated every Lth sample as the output again becomes the input. Hence, in the IIR version of the comb-filter, every Lth sample is emphasized, reemphasized, re-reemphasized, re-re-reemphasized, … etc. due to the feedback loop. This results in a perceivable pitched output signal if the output is in the pitch range — the delay is short/long enough so that we can hear a pitch.

To get a more thorough understanding of the comb-filter and its tendency to accentuate certain samples dictated by the delay L, let's conduct an impulse response analysis. By passing the filter with a unit impulse, we can see a pattern emerging — a pulse at regular intervals characterized by integer multiples of the delay length L as shown in (3.30).

$$\begin{aligned}
&h[0] = \delta[0] + 0 = 1\\
&h[1] = 0 + 0 = 0\\
&\vdots \qquad\qquad \vdots\\
&h[L] = 0 + a_1 \cdot 1 = a_1\\
&h[L+1] = 0 + 0 = 0\\
&\vdots \qquad\qquad \vdots\\
&h[2 \cdot L] = 0 + a_1 \cdot h[2 \cdot L] = a_1^2\\
&\vdots \qquad\qquad \vdots\\
&h[k \cdot L] = 0 + a_1 \cdot h[k \cdot L] = a_1^k
\end{aligned} \tag{3.30}$$

For the filter to be stable, the IIR coefficient has to adhere to the following condition:

$$|a_1| < 1 \tag{3.31}$$

This condition for the IIR coefficient should be evident from the results in (3.30) as the coefficient's power increases at integer multiples of L. In other words, if $|a_1|$ is not a number that is smaller than 1, the comb-filter will eventually (and quickly) blow up — reach a state where the output will increase without bound. On the other hand, if $|a_1|$ is some number smaller than 1, the filter output will at some point in time reach a very small number. This small number, after a while will approach 0, thus making the filter stable (see Chap. 6 for further details on stability issues with IIR systems). Hence, via the impulse response interpretation, we see how every Lth sample gets some serious attention in the case of the IIR comb-filter.

Let's next analyze the comb-filter via the frequency response and z-transform approach and interpret the filter in terms of its zeros and poles. First, since it is an IIR filter with no feed-forward component, we should expect an all-pole z-plane as shown in (3.32) with $|a_1| < 1$ for stability.

$$H(z) = \frac{1}{1 - a_1 \cdot z^{-L}} \tag{3.32}$$

We can solve the denominator for its roots and obtain the pole locations as follows

$$\begin{gathered}
1 - a_1 \cdot z^{-L} = 0\\
z^L - a_1 \cdot z^L \cdot z^{-L} = 0
\end{gathered} \tag{3.33}$$

arriving at (3.34),

$$z^L = a_1. \tag{3.34}$$

For simplicity, let's set $a_1 = 1$ (this should, however, NOT be the case for this IIR filter for stability's sake, but it'll make the math look easier!), in which case Eq. (3.34) becomes even simpler as shown in Eq. (3.35).

$$z^L = 1 \tag{3.35}$$

We know from algebra that Eq. (3.35) will have L number of roots (anything to the Nth power will have N roots). As usual, to compute the frequency response, we need to set $z = e^{j\theta}$ in Eq. (3.35) resulting in (3.36).

$$z^L\Big|_{z=e^{j\theta}} = (e^{j\theta})^L = e^{j\theta \cdot L} = 1 \tag{3.36}$$

Now, when we apply the Euler identity to Eq. (3.36), we recognize that the imaginary component is 0 at all times — there is no imaginary component on the right hand side of Eq. (3.37). Stated in another way, the result must be a real number and must be equal to 1. Hence, we need to find the specific θ values that will satisfy Eq. (3.37) and recognize that the solution is met when θ adheres to Eq. (3.38).

$$\cos(\theta \cdot L) + j\sin(\theta \cdot L) = 1 \tag{3.37}$$

$$\theta = k \cdot 2\pi/L, \quad \text{where } k = 1, 2, \ldots L \tag{3.38}$$

In other words, whenever $\theta = k \cdot 2\pi/L$, the imaginary component (sine part) disappears, leaving only the real part — the real component (cosine part) equals 1 as shown below.

$$\begin{aligned} e^{j\theta \cdot L}\Big|_{\theta=k\cdot 2\cdot\pi} &= \cos\left\{k\left[\left(\frac{2\pi}{L}\right)\cdot L\right]\right\} + j\sin\left\{k\left[\left(\frac{2\pi}{L}\right)\cdot L\right]\right\} \\ &= \cos(2\cdot\pi\cdot k) = 1 \end{aligned} \tag{3.39}$$

What Eq. (3.38) tells us when viewed from the z-plane is that θ represents L number of points on the unit circle (when $a_1 = 1$), equally subdivided into $2\pi/L$ angular increments as shown in Fig. 3.8.

Going back to Eq. (3.34) we can see that for $|a_1| < 1$, the poles in the z-plane will be inside the unit circle which guarantees stability for the

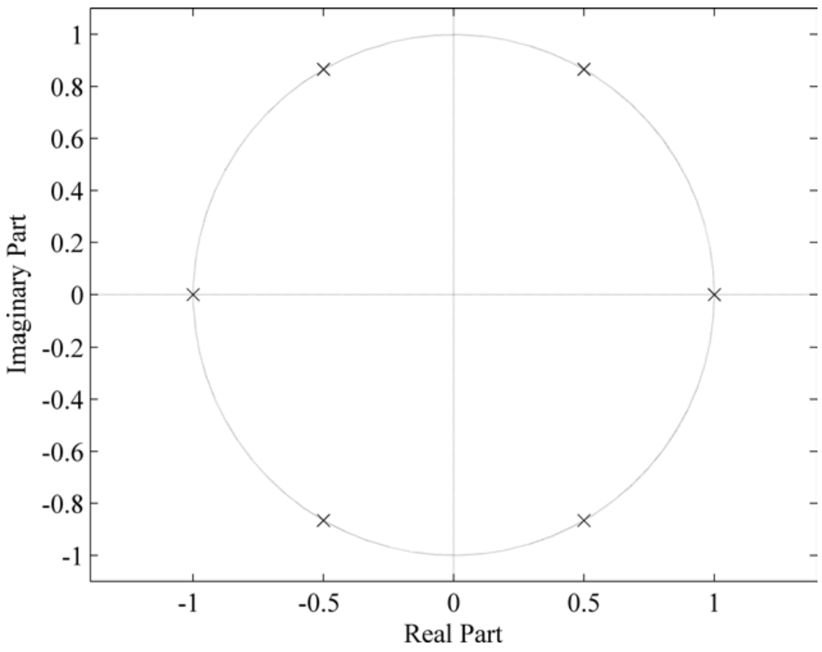


Fig. 3.8. The z-plane showing pole configuration for IIR comb-filter $L = 6$ and $a_1 = 1$.

IIR comb-filter as depicted in Fig. 3.9. A similar analysis can also be done for the FIR version of the comb-filter which I'll let you set aside for a nice Sunday afternoon's DSP session. However, note that the zero locations for the FIR are *between* the pole locations of the IIR comb-filter, although equidistantly spaced as before as shown in Fig. 3.10 (see also Fig. 3.7). This should make sense as the peaks of the FIR filter must occur at the same frequency locations as the IIR version which means that the zeros of the FIR filter (which have pulling tendencies unlike the poles which have pushing characteristics) must be placed between each peak location (poles are placed *at* each peak location and zeros between peaks).

3.3.2 *Comb-filter examples*

Comb-filters are very musically effective and have been used widely by the musical community as well as music technology researchers, especially as signals with no pitch characteristics can be filtered into pitched musical materials. A simple example of filtering a signal using the IIR comb-filter is shown in the short MATLAB® Code Example 3.3.

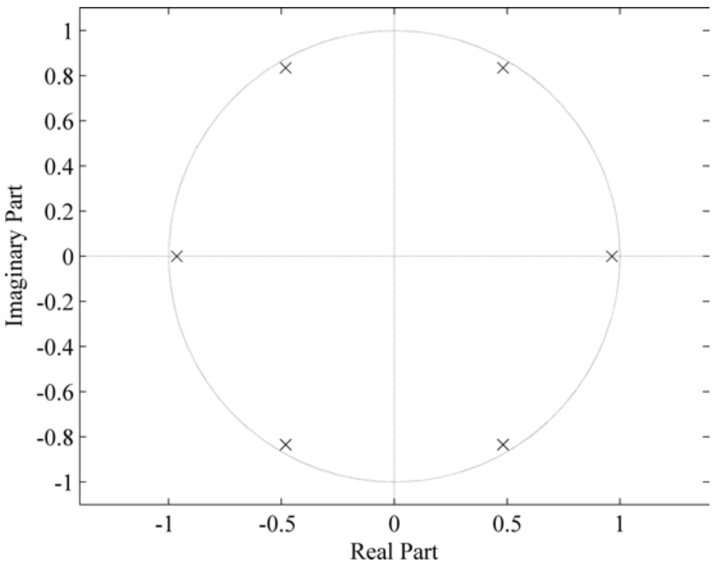


Fig. 3.9. Stable IIR comb-filter with $a_1 = 0.8$ and $L = 6$.

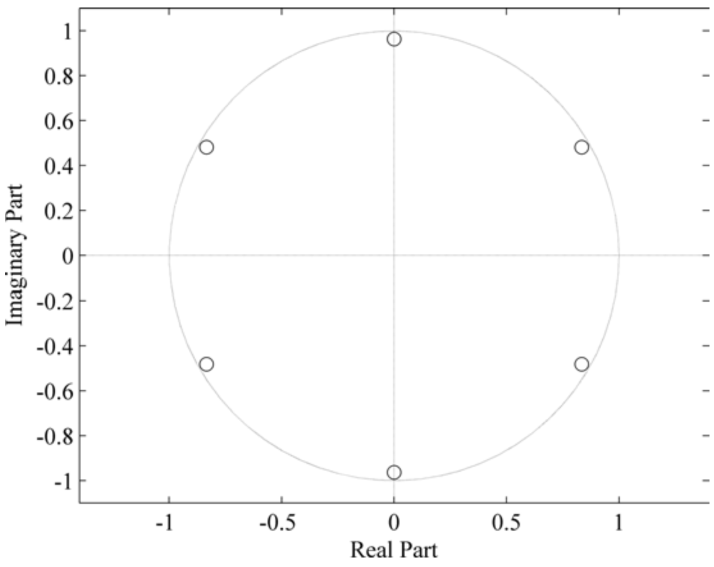


Fig. 3.10. Zero locations for FIR comb-filter with $b_1 = 0.8$ and $L = 6$.

```
% read the audio file
[x, fs] = auread ('loveme22.au');

% set parameters
f    = 440;          % set frequency for comb-filter
L    = round(fs/f);  % set delay length
coef = 0.99;         % set dampening coefficient

% construct the IIR filter coefficients into an array
% and filter
b = [1 zeros (1,L-1) coef];
y = filter (1, b, x);

% play the resulting sound
sound (y, fs);
```

Code Example 3.3. Comb-filtering of audio signal with fundamental frequency 440 Hz.

In the above code, the MATLAB® function *AUREAD*$(\cdot)$ (or *WAVREAD*) simply reads a soundfile and stores it in the input variable *x*. This is followed by initializing a few of the comb-filter parameters and setting frequency *f* to tune the comb-filter and determine the corresponding integer delay line *L*. We use the *ROUND*$(\cdot)$ function as *L* needs to be an integer. In this example, we use an IIR coefficient of 0.99 making the filter quite sharp and set the overall IIR coefficients *b* accordingly. Note that we pad the IIR weights with zeroes using the MATLAB® function *ZEROS*$(\cdot)$ to build the appropriate delay line array. Finally, we filter the signal and play the output using the *SOUND*$(\cdot)$ function. Depending on what sort of sound file you used for the input, you will hear different reinforcement characteristics of pitchiness subjected onto the input signal. Play around with the frequency *f*, comb-filter coefficient *coef*, and other sound files to get a feel for the comb-filter.

Let's use a specific kind of input signal for the above comb-filter — namely a white noise signal. Even though white noise does not have any pitch characteristics, passing it through the comb-filter will produce a pitched sound. The MATLAB® code with white noise as input is shown in Code Example 3.4. Note that this code is essentially the same as example Code Example 3.3, the difference being the input signal *x* which is now a white noise signal. Here, we used the *RAND* $(\cdot)$ MATLAB® function to generate a 1 second noise burst. The line *x* = *x* − *mean*(*x*), as usual, simply makes the noise signal have a 0 mean value so that the input *x* vibrates on the $y = 0$ axis (DC offset removal). We further add an amplitude envelope to the output signal to give it more of a musical character as shown in Code Example 3.5.

```
% set fs and generate white noise signal of length 1 sec
fs = 22050;
x  = rand(fs,1);
x  = x − mean(x);  % get rid of DC offeset

% set comb-filter coefficients
f = 220;          % fundamental
L = round(fs/f); % delay length
coef = 0.99;      % IIR coefficient

% build delay vector and filter
b = [1 zeros(1,L−1) coef];
y = filter(1, b, x);

% play sound
sound(y, fs);
```

Code Example 3.4. Comb-filtering of noise.

```
% set fs rate and generate white noise of length 1 sec
fs = 22050;
x = rand(fs,1);
x = x − mean(x); % get rid of DC offeset

% set comb-filter coefficients
f = 220;          % fundamental
L = round(fs/f); % delay length
coef = 0.99;      % IIR coefficient

% build delay vector and filter
b = [1 zeros(1,L-1) coef];
y = filter(1, b, x);

% create amplitude envelope for output
decay = 8;
expEnv = exp ((0:(length(y)−1))/length(y));
expEnv = (1./expEnv).^ decay;

% envelope output signal
y = y .* expEnv';

sound(y, fs); % play sound
```

Code Example 3.5. Comb-filtering of white noise with amplitude envelope.

The resulting sound is quite remarkable — not only does the comb-filter make a noisy signal into a pitched one, the mere addition of a simple amplitude envelope makes it even musically interesting and maybe, dare I say, naturally sounding. Play around with the amplitude envelope's decay characteristics (or try a different envelope shape altogether) as well as the fundamental frequency parameter — what musical instrument does it kind of sound like? I think if we had to make a guess, we could say that it sounds like a really bad and bright metal-stringed guitar — is this a coincidence? Read on to find out in the next section.

3.4 *String vibration and standing waves*

We may remember from our physics class at some point in our lives that one memorable lab session where we fixed a rope to a door-knob, held the other end of the rope, stretched the string to make it tight, and gave it a healthy jolt, causing a pulse to propagate along the rope. If we really poke around hard in our long term memory section of the brain, we may also recollect seeing a wave traveling towards the door knob, bouncing back once reaching it and returning back to your hand but in an inverted form as seen in Fig. 3.11.

If the string was tapped really hard, the back and forth propagation would go on maybe a tad bit longer, with the upside down wave bouncing at the hand and making its way back towards the door-knob but with a less pronounced hump and finally dying out completely after two or three trips back and forth. This is what happens to a stringed instrument when plucked as shown in Fig. 3.12.

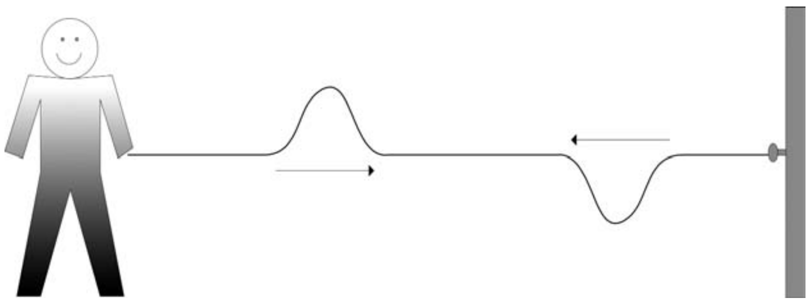


Fig. 3.11. Our favorite physics experiment at Ms. Newtown's class.

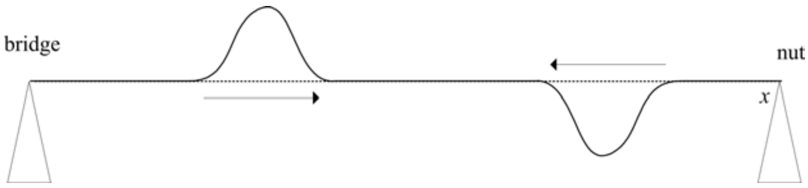


Fig. 3.12. Vibrating string on a guitar with bridge and nut.

An important mathematical formula for a vibrating string can be expressed as a combination (superposition principle) of two traveling waves in opposite directions according to Eq. (3.40).

$$y(x,t) = y_l(t + x/c) + y_r(t - x/c) \tag{3.40}$$

$$c = \sqrt{\frac{\mathrm{K}}{\rho}} \tag{3.41}$$

In Eq. (3.40), x is the displacement of the string and c the speed of the traveling wave. The traveling wave's speed c can be further characterized by the string's tension and linear mass density ρ as shown in Eq. (3.41).

If we go back to the physics experiment and excite the rope at a regular interval, the scenario would result in forward and backwards propagating waves, traveling at the same time and causing interference. The ensuing results produce what are referred to as *standing waves* — when we have two traveling waves propagating in opposite directions on the same string, the traveling waves do not actually "travel" anymore but stand still vibrating vertically instead.

It so turns out that in a real world situation, a string tied at both ends of length $L/2$ causes natural resonances to occur at $k \cdot 2\pi/L$ radians/sec where k is an integer number. This can be viewed as standing waves in a vibrating string situation, causing multiple vibrational modes to occur as well as corresponding nodes as shown in Fig. 3.13. The first mode is equivalent to the fundamental frequency, the second mode, with a center node that vibrates twice as fast, equivalent to the octave and so on. An interesting trait about nodes is that if one were to precisely touch a node of any mode during its vibration, the touching of the node would not affect the vibration of that particular mode in any way whatsoever (of course this is physically impossible to achieve without loss of energy due to friction of some amount between the finger and string). On the other hand, if one would touch the anti-node (the maximum of each mode) we would dampen

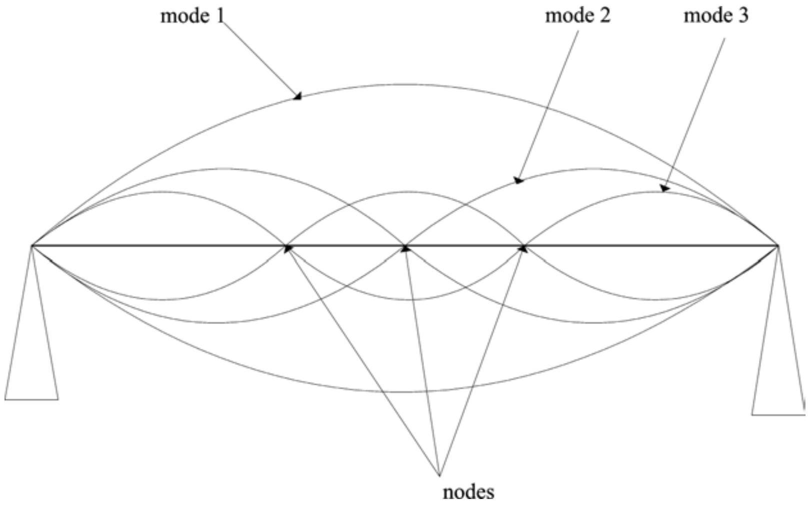


Fig. 3.13. Modes of vibration and corresponding nodes.

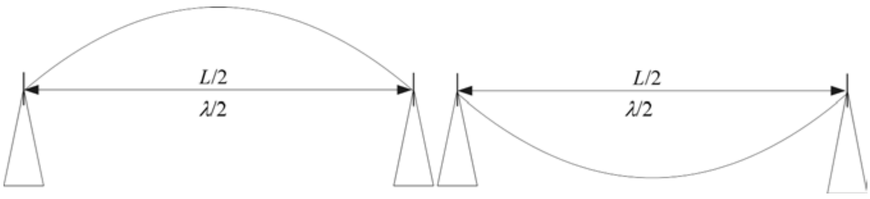


Fig. 3.14. First and second half of a round trip.

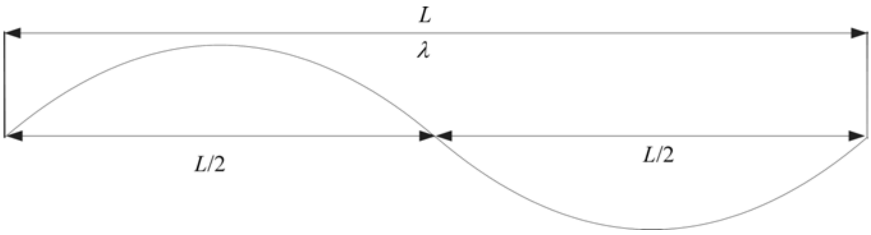


Fig. 3.15. Full cycle after first mode makes a full round trip.

that particular mode and hence corresponding harmonics that have maxima at the same anti-node location, causing those harmonic to cease to vibrate.

Note also that the length of the string is directly related to the wavelength λ as shown in Fig. 3.14. That is, the lowest frequency (first mode) which corresponds to the fundamental frequency needs to travel back and forth once and make a round trip in order to complete a full cycle (Fig. 3.15). The modes and nodes are very important in stringed instruments and can be thought of harmonics which in general heavily influence the sound characteristics or timbre of musical instruments and audio signals.

In the above physics experiment, there was a considerable amount of friction between the rope and fingers/hands causing the vibration of the string to decay rather quickly. However, a firm base for the string at either end (called the nut and bridge for stringed instruments) will cause the decay and dampening of the left and right traveling waves to be prolonged, thus sustaining the vibration of the string. For an ideal string vibration system where there is no energy lost at all, the left and right traveling waves would reflect off the boundaries in perpetuity. Obviously this is not the case in the real world as energy dissipates mainly in the form of acoustic energy as well as thermal and friction-based energy lost at the bridge/nut part of an instrument. All of this stuff that we are talking about here can be approximated and represented with digital difference equations and digital filters — you may have observed that the resonant characteristics of strings are similar to the behavior of comb-filters ($k \cdot 2\pi/L$). This brings us to the next topic where we move from the real world of the physical to the virtual world of modeling known as physical modeling.

3.5 *Physical modeling synthesis and the plucked string model*

In the previous section, we talked about the fundamental behavior of strings and the superposition principle of left and right traveling waves that express standing waves in stringed instruments. In this section, we will show how we can model those physical real-world characteristics using modeling techniques known as *physical modeling.* Physical modeling synthesis refers to a research field where the main aim is to model acoustic instruments through mathematical approximations and algorithms. Physical modeling, however, is not limited to modeling of acoustic instruments and many examples including mimicking analog synthesizers and circuitry such as compressors, amplifiers, and general vacuum-tube-based technologies are also being implemented via digital means. Physical modeling synthesis models are also often referred to as *waveguide* models and have had much success in mimicking acoustic instruments due to the numerous contributions by researchers especially from Stanford University. Waveguide models are typically constructed via digital filters and delay lines making it ideal for implementation using digital signal processing techniques.

To come up with a difference equation and filter for the plucked string model, we will need to try to go from the real-world string to the digital-world string model. We will start by digitally representing the left and right traveling waves of Fig. 3.12 and Eq. (3.40) as shown in Fig. 3.16.

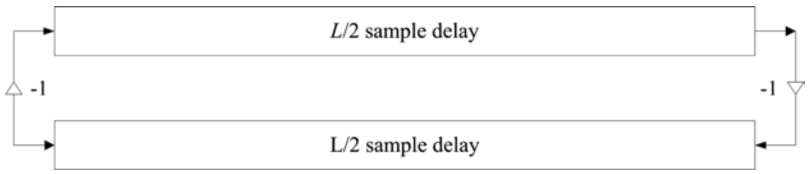


Fig. 3.16. Simplified digital waveguide of the left and right traveling wave.

What we have done here is replace the left and right traveling wave paths with two digital delay lines of length $L/2$ and substituted termination points with inverting multipliers at the bridge and nut part of the plucked string system. More specifically, since the delay lines are digital, the system will actually look more like Fig. 3.17. We have also added the loss of energy factor for each half of the string itself (delay line $L/2$) which is characterized by the variable g and furthermore have included the loss of energy at the bridge and nut denoted by coefficients $-a$ and $-b$. The energy loss g can be regarded as internal friction within the string and drag with the surrounding air during the production of sound (Smith 1992).

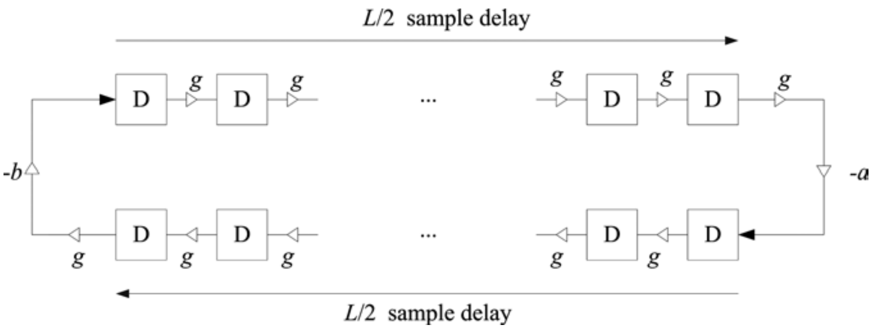


Fig. 3.17. Digital waveguide with loss at each delay unit.

As we can see from Fig. 3.17, when the propagating "digital wave" reaches the end of each digital delay line, its sign gets inverted and dampened by the negative multiplier $-a$ and $-b$. If a, b, and g were equal to 1, there would be a full reflection at the termination points and no loss of energy within the string itself, causing the traveling wave to loop forever. If $|a| < 1$ or $|b| < 1$ or $|g| < 1$, the result will be a gradual loss of energy causing the wave propagation to stop after a number of runs. As we know from experience, a guitar string never sustains vibration "forever" and decays after a short period. The dampening factors a, b, and g reflects this attribute in the string model.

Now, let's further simplify the above configuration as depicted in Fig. 3.18 by exploiting the LTI's commutative property — we combine the overall internal energy loss factor of the string and lump it into a single variable g^L:

$$g^{L/2} \cdot g^{L/2} = g^L \tag{3.42}$$

The termination dampening factors can likewise be grouped into a single dampening variable as shown below.

$$(-b) \cdot (-a) = a \cdot b = c \tag{3.43}$$

Since we're at it, we may as well consolidate losses g and c into a single loss component q as shown in Eq. (3.44)

$$q = g^L \cdot c \tag{3.44}$$

and finally combine the two delay lines of length $L/2$ into a single long delay line L. The beauty of LTI systems comes into full force here ...

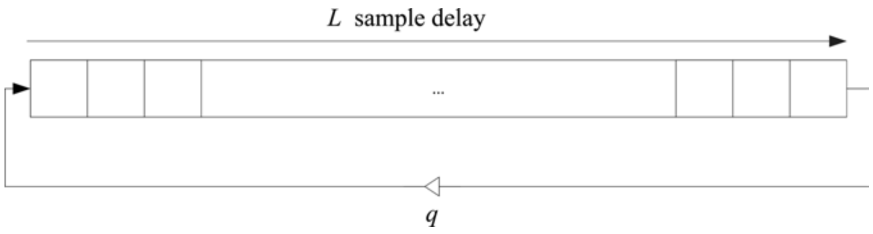


Fig. 3.18. Consolidated dampened plucked string.

Next, if we rearrange and modify the above and add an input x, output y, and a feedback delay component, we arrive at the configuration shown

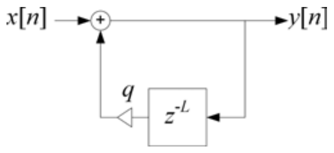


Fig. 3.19. Ideal string feedback filter.

in Fig. 3.19. This filter will be familiar to you as we have studied this in Sec. 3.3 — the feedback comb-filter with $|q| < 1$ for stability!

In Sec. 3.3.2, we already listened to the feedback comb-filter and it did not really sound like a realistic plucked string sound. One of the main reasons this is the case is that for the above ideal string comb-filter model, all modes of vibrations are dampened equally. That is, all of the harmonics die away at the same rate with q determining their fate of decay. This, however, is not how it works for real strings. As a matter of fact, for real strings, the higher modes generally die away faster than the lower ones, with the fundamental mode usually dying away last. One very clever trick to reflect this in the plucked string model is shown in the *Karplus-Strong* plucked string model (Karplus and Strong 1983). Kevin Karplus and Alex Strong came up with this beautifully simple, yet very effective idea of basically inserting a feedback low-pass filter to Fig. 3.19. Because the low-pass filter is part of the feedback loop, it repeatedly low-passes the output signal, thus dampening the high frequency components more and more as time passes. This small tweak effectively models the different decay rates of the modes of a vibrating string — higher frequencies as a result decay faster and lower ones are less affected. The result of this wonderfully simple filter is shown in Fig. 3.20.

The low-pass filter used in the Karplus-Strong model is the simplest two-point average filter given by the difference Eq. (3.45). This difference equation simply adds the current sample and the last sample and divides

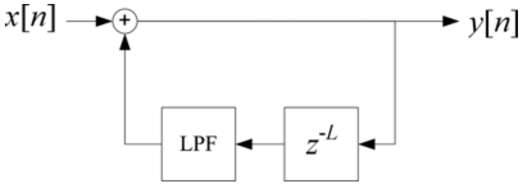


Fig. 3.20. Basic Karplus-Strong plucked string algorithm.

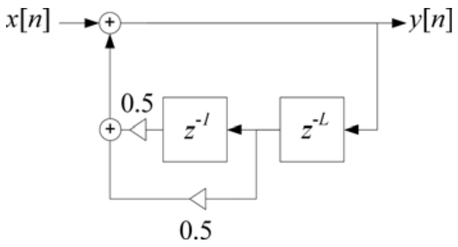


Fig. 3.21. System diagram of Karplus-Strong model with low-pass filter.

the sum by two — computing the average.

$$y[n] = \frac{x[n] + x[n-1]}{2} \tag{3.45}$$

Finally, when we write out the difference equation of Fig. 3.20 using the two-point averaging filter, we arrive at the result shown in Eq. (3.46) and the basic Karplus-Strong plucked string model system diagram (Fig. 3.21).

$$y[n] = x[n] + \frac{y[n-L] + y[n-L-1]}{2} \tag{3.46}$$

To take the plucked string model for a spin, we fill and initialize the delay line with white noise simulating a pluck (this is sort of like having thousands of picks and simultaneously picking the string at all points at the same time) as shown in Code Example 3.6 (compare it to the previous straightforward feedback comb-filter and notice the difference in timbre).

The magnitude and phase responses of the plucked string model are shown in Fig. 3.22. We can see (and hear) that with the introduction of the low-pass filter, each of the harmonics now have different dampening characteristics. We can further observe how the poles gravitate away from the unit circle and towards the origin as we approach the Nyquist limit as seen on the z-plane in Fig. 3.23. This is as expected and desirable as it translates into having more dampening for the higher frequency components and less for the lower ones. There are of courses still many problems and tweaks that we have not elaborated on here and will leave it for future study for the curious ones as the algorithms are well documented in numerous papers and articles (refer to end of this chapter for leads).

```
fs = 22050;
len = 2;          % 2 seconds

f = 220;          % fundamental frequency
L = round(fs/f); % equivalent delay length

x = rand(1, L);  % make noise: L number of samples

x = [x zeros(1,(len*fs-length(x)))]; % pad 0 to make 2 seconds

a = [1 zeros(1,L-1) -0.5 -0.5];      % feedback coefficients
y = filter(1, a, x);                 % filter the pluck

sound(y, fs);
```

Code Example 3.6. Basic Karplus-Strong plucked string.

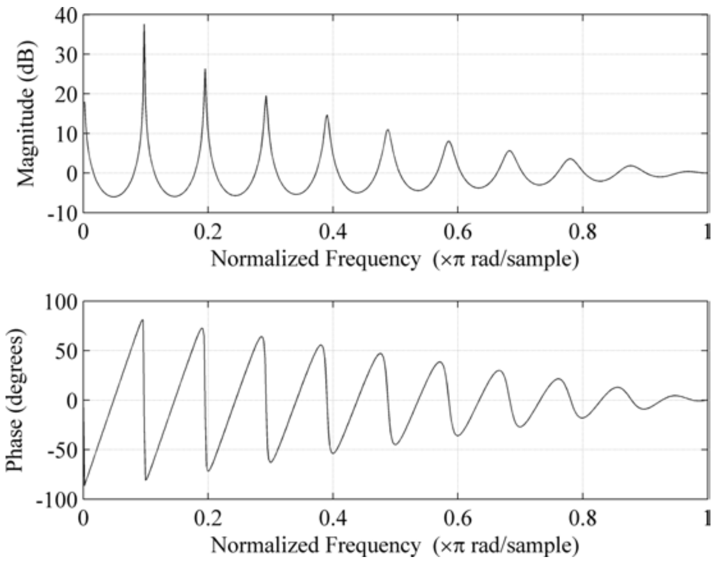


Fig. 3.22. Karplus-Strong filter frequency response for $L = 20$.

Before we move on to other topics pertinent to filters, let's briefly analyze the frequency response by performing the z-transform on the difference equation (3.46). This will result in (3.47).

$$H(z) = \frac{1}{1 - 0.5z^{-L} - 0.5z^{-(L+1)}} \tag{3.47}$$

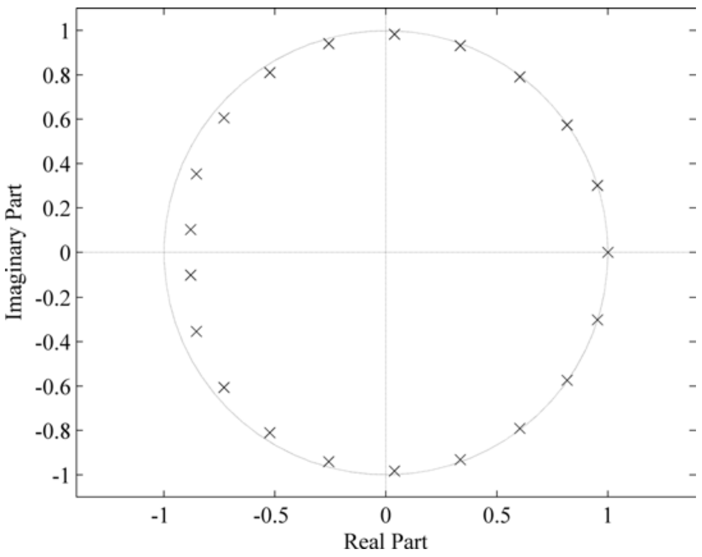


Fig. 3.23. Karplus-Strong z-plane for $L = 20$.

$H(z)$ can be rewritten as seen in Eq. (3.48) by bringing out the common z^{-L} delay component.

$$H(z) = \frac{1}{1 - 0.5z^{-L} - 0.5z^{-(L+1)}} = \frac{1}{1 - z^{-L} \cdot \left(\frac{1+z^{-1}}{2}\right)} \tag{3.48}$$

We recognize that this reflects how we actually started out analyzing the plucked string model in Fig. 3.20 — a delay line followed by a low-pass filter architecture. That is, the z^{-L} corresponding to a lossless delay line (frequency response 1) and $(1 + z^{-1})/2$ corresponding to an averaging filter representing energy dissipation as shown in Eq. (3.49) and Fig. 3.24.

$$H(z) = \frac{1}{1 - H_{delay}(z) \cdot H_{LPF}(z)} \tag{3.49}$$

This is one of the great conveniences that LTI systems afford us in signal processing. It allows us to add additional LTI DSP blocks and modules to an existing LTI system to modify, tweak, remove, and alter a system's behavior. It is thus not a stretch of the imagination to see how we got the frequency response shown in Fig. 3.22 as it is a feedback comb-filter colored by a two-point low-pass filter in the feedback loop. As a matter of fact, we can modify and extend the original Karplus-Strong algorithm by inserting

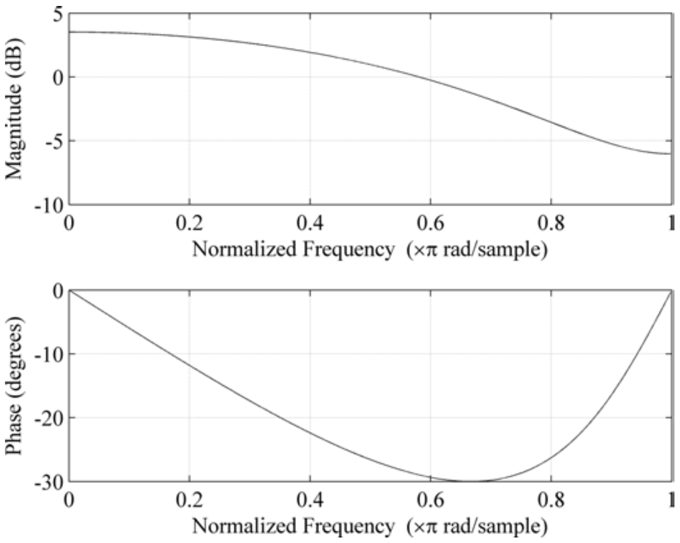


Fig. 3.24. Averaging filter frequency response.

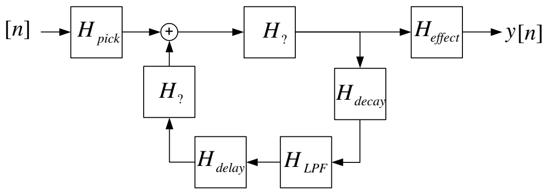


Fig. 3.25. Additional filters for a growing guitar model.

additional specialized DSP blocks to simulate the decay-time alteration, moving pick, more accurate tuning (the basic model we talked about here uses non-fractional delay, see Sec. 3.6.3 for details on fractional delays), and pluck position as shown in Fig. 3.25.

It is also possible to model different types of plucked string instrument such as acoustic guitars, electric guitars, basses, as well as the Indian sitar. The sitar is quite an interesting instrument that has a uniquely shaped bridge causing friction to occur between the string and bridge itself. This is usually undesirable for regular guitars as the result manifests in the form of a buzzing sound, much like buzzing that occurs when strings hit the frets. For the sitar, however, the buzzing is a critical element providing the instrument its character. One simple approach in mimicking the sitar's buzzing timbre is utilizing a waveshaping-type approach (Park, Li 2008) to model the bridge. This helps in distorting the normal mode of string vibration as discussed earlier, resulting in a dynamically changing buzzing sound. Improving and tweaking an instrument model obviously does not have to stop here, as we can also model the sitar body, add a sympathetic string vibration block, and maybe even include digital effects such as distortion for the electric sitar, and go on to do some amplifier modeling . . .

3.5.1 *Direct implementation of difference equations*

Up to this point, we used the *FILTER* MATLAB® function to implement the filters and difference equations. However, these will obviously only work in the MATLAB® environment with the appropriate toolbox — if you want to port it to a more generic language such as C/C++ you will need to directly code the difference equations. For example, if we were to program the plucked string model directly without any proprietary functions/methods Code Example 3.6 will look like Code Example 3.7. As you will note, the only real difference between Code Examples 3.6 and 3.7 is the manual implementation of the multiplication/addition and the buffer updates which is hidden from the user in the *FILTER* function. The results for both code examples will, as expected, be identical.

```
...

yBuffer = [zeros (1, N+1)];

...

for i = 1: iterations
     % filter signal
       y(i) = x(i) + 0.5*(yBuffer (N) + yBuffer (N+1));

     % update delay line
     % start from back and go towards beginning
     for j = (length (yBuffer)-1):-1:1
        yBuffer (j+1) = yBuffer (j);
     end

     yBuffer(1) = y(i); % store current sample to delay
     buffer
end

...
```

Code Example 3.7. Basic Karplus-Strong plucked string direct filter implementation.

Looking closer at Code Example 3.7, you will see that we initialized the delay line (`yBuffer`) of length $N + 1$ to zero before filtering. That is, making the system a causal system where nothing happens until the "power switch" gets turned on. We can easily set our initial conditions in our standard implementation of the plucked string code as needed. An important thing to remember about buffer updating is that we need to always shift the oldest buffer element first (delete from the buffer) and then replace it with the next oldest one. The most current sample becomes the last addition to the buffer after all the shifts have been completed as shown in Code Example 3.7 [`yBuffer(1) = y(i)`]. Finally, it is also possible to set/retrieve the initial/final conditions of a filter using the MATLAB® *FILTER* function by using a fourth input argument (`Zf`) and an additional output (`Zi`) argument: `[Y, Zf] = FILTER(B,A,X,Zi)`. With these optional parameters, the initial buffer settings/values (`Zi`) and the final buffer settings/values (`Zf`) can be set and retrieved for the feedforward and feedback components respectively.

3.6 *Phase as a filtering application*

There are countless audio effects that are used at the recording stage, music production stage, sound design, and compositional stages in musical creation as well as in live performance situations. One of the most popular and widely used effect is the chorus effect. It can be quite easily implemented using DSP filters and has found much popularity in a vast number of musical situations to make a sound more "full" and "lush." It is also at times used for voice doubling — vocalists who do not have a full and thick voice, lack "presence," or have shaky pitch often borrow technology to double the recorded voice either synthetically via DSP techniques, or manually by singing the same vocal line twice on top each other (while persevering the original recorded vocal line). The latter obviously takes much more time as singing the same vocal line in exactitude is not an easy task to say the least (it is in reality impossible and actually undesirable as we will soon find out). We will begin the topic of phase exploitation in musical applications starting with the chorus effect which we briefly introduced in Chap. 3, Sec. 4.2.1.

3.6.1 *The chorus effect*

The origin of the name chorus effect most likely comes from the sonic result that is produced by a group of singers as opposed to a solo singer. The number of voices directly brings about the quality of richness and lushness via the ensemble of voices — the chorus. This is also evident in the traditional Western symphony orchestra culture where an army of approximately 100 instrumentalists produce a formidable auditory experience, an experience a single instrument and even a smaller group of instruments cannot achieve. The string section in an orchestra is such an example, typically consisting of about 20 violins, 10 violas, 10 cellos, and 6 double basses. It is difficult to describe the perceptual magnificence of the cumulative sound when the strings all play together at the same time, even if just playing the same note in unison. The artificial modeling of the chorus effect is actually quite straightforward in concept and takes into consideration errors that occur when instrumentalists (including the voice) play (or sing) together. Let's say we have two violinists who are reading from the same score and are playing the exact same notes in terms of pitches and note durations. No matter how hard they try to play together perfectly, there will always be some small amount of error and total synchrony is for all practical purposes unattainable. As a matter of fact, the aspect of synchrony becomes worse with the addition of more and more violins. These errors are essentially small delays between the instruments (timing) and a small amount of detuning (frequency) due to the microscopic tuning differences between instruments. Again, no matter how hard one tries to exactly tune an acoustic instrument, when two or more acoustic instruments play together, there will be a very small amount of detuning. The amount of detuning will depend on the instrument type, note played, dynamics, duration of a note, duration of the piece being played, as well as distance between the sound sources and the listener. These errors, however, are not necessarily negative features for musical purposes as it results in a rich and sometimes even "warm" timbre if the artifacts are not extreme. Of course with electronic instruments, especially digital instruments the tuning aspect is less of a problem and at times electronic instruments are detuned purposely to give it a more natural feel (add error) — the chorus effect feel if you will. I remember doing this with the Yamaha V50, an FM-based synthesizer back in the early 1990s — programming two of the same electric piano timbres onto two MIDI channels and detuning each slightly by a couple of cents which resulted in a pretty smooth and warm electric piano sound. The digital version of the synthetic chorus effect and its basic architecture is shown in Fig. 3.26 and discussed in detail in the next section.

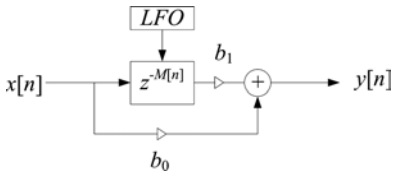


Fig. 3.26. Basic chorus filter architecture.

3.6.2 *Multi-tap filters and filter banks*

Now, how do we implement a digital chorus filter? As described above, the two main features are the delay amount and detune amount. The delay lines used in digital chorus are actually of a time-varying type, implemented as a bank of multi-tap delay configuration as seen in Figs. 3.26, 3.27, and 3.28. The input voice or signal is routed into a number of channels and delayed, reflecting the number of voices where each channel's delay amount changes dynamically with time. This variable delay produces the detuning characteristic of chorus, mimicking the delay between real instruments and the ensuing detuning effect. Notice that in essence, Fig. 3.26 is a feed-forward comb-filter discussed in Sec. 3.3 with a difference equation and

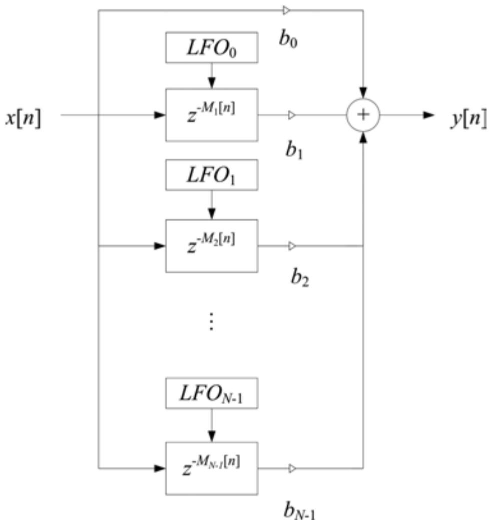


Fig. 3.27. Multi-bank filter configuration.

delay line Mas shown below.

$$y[n] = x[n] + b_0 \cdot x[n - M] \tag{3.50}$$

For the chorus filter, M is a function of time, typically controlled via an LFO (low frequency oscillator). The LFO gently sweeps between its minimum and maximum amplitude limits, controlling the delay amount. The LFO sweeper would typically look like (3.51) with maximum amplitude A_{LFO}, sample index n, LFO frequency f_{LFO}, and sampling frequency f_s.

$$M[n] = A_{LFO} \cdot \sin(2 \cdot \pi \cdot f_{LFO} \cdot \frac{n}{f_s}) \tag{3.51}$$

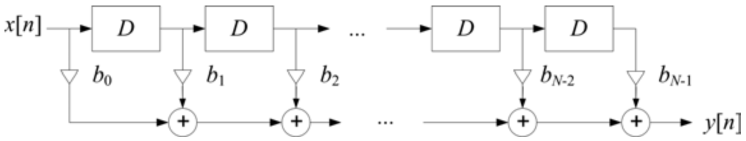


Fig. 3.28. Multi-tap delay line.

The general multi-tap delay architecture is shown in Fig. 3.28 and Eq. (3.52) which can be viewed as a linear combination of weighted input and weighted delayed ($D = 1/f_s$) input samples (FIR filter) with a total maximum delay of $(N - 1)/f_s$.

$$\begin{aligned} y[n] = b_0 \cdot x[n] + b_1 \cdot x[n-1] + \cdots + b_{N-2} \cdot x[n - N - 2] \\ + b_{N-1} \cdot x[n - N - 1] \end{aligned} \tag{3.52}$$

It would be rare to have a configuration like Fig. 3.28, where each delay component is added to the output. It is more common to see delay lines lumped together prior adding the contribution of the lumped delayed results to the output. In any case, a dynamically changing discrete delay line can be implemented by tapping into the various delay locations in Fig. 3.28. For example, if we want a delay increase from $K - 100$ to K samples, all we would need to do is tap into the $K - 100$th sample, then tap into $K - 99, \ldots$ and finally K. The accessing of this delay line, also commonly referred to as a buffer, can be accomplished via a LFO-based indexing scheme as used in the chorus effect.

For example, if the LFO in the chorus effect is a sine oscillator, we could set the frequency to a low value such as 0.5 Hz and set the amplitude component of the sine wave to smoothly vary between the start tap location and end tap location — in our example that would be equivalent to $K-100$ to K. Below is the code for a simple single-bank chorus effect processor.

```
[x, fs] = auread ('loveme22.au');  % read audio file

buffer = zeros(1,1000); % allocate and init. buffer (delay line)
y = zeros(1,length(x)); % allocate and init. output

f = 0.1;          % LFO frequency
startIdx = 500;   % start delay in samples

sineTable = abs (sin(2*pi*f*[0:length(x)-1]/fs)); % make LFO

% tap index from startIdx to end of buffer
bufferIndex = round(sineTable*(length(buffer)-
startIdx))+startIdx;

% run chorus filter
for i = 1:length(x)
   y(i) = x(i) + buffer (bufferIndex(i));

   % update the buffer: shift right (last is oldest)
   buffer(2:end) = buffer (1:end-1); % shift
   buffer(1) = x(i);                 % update
end

% play the sound
sound(y, fs)
```

Code Example 3.8. Simple, single-bank chorus effect.

Code Example 3.8 could easily be modified so that we have a multi-bank chorus configuration: add more buffers with their respective LFO and delay characteristics; insert the buffers into the main loop so that the program would look something like in Code Example 3.9. In both examples, we used some of the special short-cuts offered in MATLAB® to update the buffers by exploiting the *:* and *end* operators without explicitly using nested *for* loops as we did in Code Example 3.7. In the end, it does not matter what method one uses, as different systems and computer-based languages have their own quirks, idiosyncrasies, and special functions/operators to deal with specific problems as well as enabling efficient computation of particular algorithms. The goal is ultimately getting the correct answer with the desired efficiency.

```
...
for i=1:length(x)
   y(i) = x(i) + buffer1(bufferIndex1(i)) +···
              buffer2(bufferIndex2(i)) +···
              buffer3(bufferIndex3(i));

   % update the buffer: shift right (last is oldest)
   buffer1(2:end) = buffer1(1:end-1);
   buffer2(2:end) = buffer2(1:end-1);
   buffer3(2:end) = buffer3(1:end-1);

   % update
   buffer1(1) = x(i);
   buffer2(1) = x(i);
   buffer3(1) = x(i);
end
...
```

Code Example 3.9. Simple, single-bank chorus effect.

3.6.3 *Fractional delay*

Due to the dynamically changing nature of the chorus delays, typically smoothly changing from 10 ms to about 50 ms, we need to implement fractional delay lines. If we were to use non-fractional delays, we would only be able to implement step-wise delays dictated by the sampling rate f_s and integer K as seen in Eq. (3.53). This means that each delay increment or decrement would be confined to integer units of $T = 1/f_s$ seconds. Two common methods for achieving fractional delays are linear interpolation and all-pass filtering, the latter discussed in Sec. 3.6.5.

$$\text{delay}_{non\text{-}fractional} = K \cdot \frac{1}{f_s} \tag{3.53}$$

Linear interpolation of delay is quite straightforward to implement and is shown in Eq. (3.54) where τ is the fractional delay amount, $y_{FD}[n-\tau]$ is the delayed output sample value, *frac*(τ) is the fractional part (mantissa), and *int*(τ) is the integer part of the fractional delay τ.

$$y_{FD}[n-\tau] = a_{\text{int}(\tau)} \cdot y[n - \text{int}(\tau)] + a_{\text{int}(\tau)-1} \cdot y[n - \text{int}(\tau) - 1] \tag{3.54}$$

$$a_{\text{int}(\tau)} = 1 - \text{frac}(\tau) \tag{3.55}$$

$$a_{\text{int}(\tau)-1} = \text{frac}(\tau) \tag{3.56}$$

$$a_{\text{int}(\tau)} + a_{\text{int}(\tau)-1} = 1 \tag{3.57}$$

Note that the resulting fractional delay in essence is just a linear weighting of two adjacent delayed samples. For example, if the fractional delay we want is 6.5, the weights $a_{\text{int}(\tau)}$ and $a_{\text{int}(\tau)-1}$ will be both equal to 0.5, while the integer delays equal to 6 and 7 respectively:

$$y_{FD}[n-\tau]|_{\tau=6.5} = 0.5 \cdot y[n-6] + 0.5 \cdot y[n-7]$$

However, if the delay is 6.8 instead of 6.5 samples, the fractional delay will be:

$$y_{FD}[n-\tau]|_{\tau=6.8} = 0.2 \cdot y[n-6] + 0.8 \cdot y[n-7]$$

Again, the beauty about LTI systems is that we can just add the above fractional delay difference equation block to the integer multi-tap delay block and render a dynamically adjustable fractional delay filter version as shown in Fig. 3.29. With this configuration, the multi-tap block takes care of the dynamically changing integer delays and the fractional delay block, the fractional delay part. Eq. (3.54) now simply becomes the difference equation (3.58), provided that the integer part of the delay is taken care of in the multi-tap filter. Thus, the combination of integer delay in series with

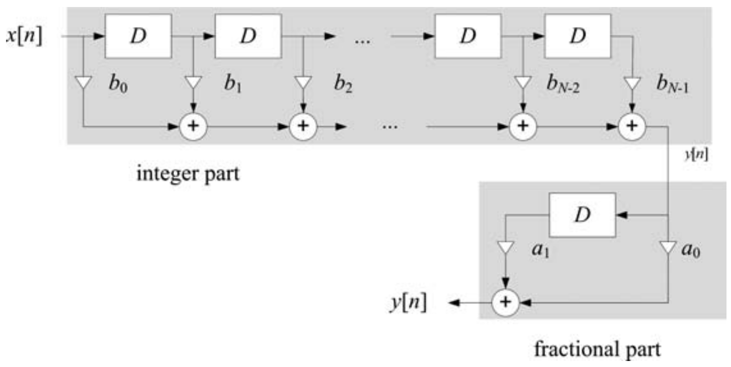


Fig. 3.29. Multi-tap filter with fractional delay.

fractional delay will give us a dynamic fractional delay line.

$$y_{FD}[n-\tau] = a_0 \cdot y[n] + a_1 \cdot y[n-1] \tag{3.58}$$

$$a_0 = 1 - \text{frac}(\tau) \tag{3.59}$$

$$a_1 = \text{frac}(\tau) \tag{3.60}$$

3.6.4 *The flanger effect*

As introduced in Chap. 3, Sec. 4.2.1, the flanging effect is very similar to the chorus effect as it also uses a variable delay line. The main difference is that the flanger only uses one additional voice and the delay range is at a lower $1 \sim 10$ ms range, opposed to $20 \sim 30$ ms which is generally used for the chorus effect. Like the chorus effect (but even more so here for the flanger), we do not hear the time delay in terms of an echo as the delay is too short; rather we hear it is as an effect, thereby changing the resulting timbre. The word flanging originates from the analog reel-to-reel days where two tape recorders having the same program (recorded signal) are played back simultaneously in synchrony. In this scenario, each tape machine would forcefully and alternately be subtly slowed down via the application of hand pressure against the flanges (rim of the tape reels with the hands thus causing a small amount of asynchrony followed by re-synchronization upon release of the rim). The system diagram is shown in Fig. 3.30. As we can see, it is indeed identical to the chorus effect in its basic architecture.

As mentioned above, the flanger, however, has characteristics that are different from the chorus effect mainly due to the range of delay amount and the number of voices used. In particular, the comb-filtering effect is a prominent characteristic of the flanger effect that sticks out, accomplished by the feed-forward comb-filter illustrated in Eq. (3.61) — an input signal $x[n]$ (tape machine 1) in combination with a delayed version $x[n-M]$

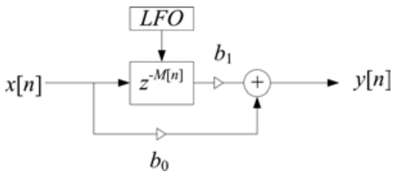


Fig. 3.30. Flanger filter architecture.

(tape machine 2).

$$y[n] = b_0 \cdot x[n] + b_1 \cdot x[n - M] \tag{3.61}$$

As we have seen in Sec. 3.3, feed-forward comb-filters use zeros to shape the frequency response resembling a comb. This is exactly what happens when the original signal and its slightly delayed image are summed (along with the dynamically changing delay line). The summing of the two input signals and the resulting output can be regarded as a consequence of constructive and deconstructive interference as the delay of the image is varied with respect to the original input signal, causing the notches (where the zeros are located) to appear in the frequency response. Similar to the implementation of the chorus system, we can likewise implement fractional delay to introduce smooth sweeping of the delay as discussed in Sec. 3.6.3.

3.6.5 *The all-pass filter*

The all-pass filter as the name implies, passes all frequencies without alteration to their magnitude values. That is, it has a flat frequency response as shown in Fig. 3.31.

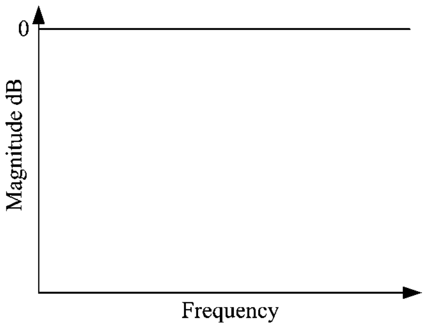


Fig. 3.31. All-pass filter frequency response.

But such a filter does not seem to make much sense, nor does it seem very exciting at all as there is no change in the magnitude response and seemingly no change to the signal itself. This is somewhat true but not entirely accurate — there is modification to the signal albeit not to its magnitude response. The reader may have noticed that there was not that much talk about the phase aspect when dealing with filters in this chapter. One of the reasons is that phase is not the most noticeable and evident

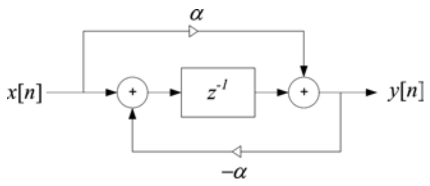


Fig. 3.32. 1st order all pass filter system diagram.

feature in filtering and signal processing — perception of phase change is often quite subtle especially when compared to magnitude change. However, I hope that with the introduction of all-pass filter in this section, we will get a better feel for this thing called phase and further appreciate its significance and importance in signal processing.

The difference equation for a 1st order all-pass filter is shown in (3.62), the z-transform in (3.63), and system diagram in Fig. 3.32.

$$y[n] = \alpha \cdot x[n] + x[n-1] - \alpha \cdot y[n-1] \tag{3.62}$$

$$H(z) = \frac{\alpha + z^{-1}}{1 + \alpha \cdot z^{-1}} \tag{3.63}$$

Figure 3.33 further depicts the zero and pole configuration which clearly shows the pole and the zero cancelling each other out (pulling and pushing at the same frequency points) and hence producing a flat frequency response.

To verify that the numerator and denominator of the transfer function are indeed the same when computing the magnitude response $|H(e^{j\theta})|$ (equates to unity gain response), we multiply the numerator and denominator of Eq. (3.63) by $e^{j\theta/2}$ — multiplying the numerator and denominator with $e^{j\theta/2}$ does not change anything as it is analogous to multiplying it by one. However, by doing so, the results will reveal what we are looking for.

$$\begin{aligned}
H(e^{j\theta}) &= \frac{\alpha + e^{-j\theta}}{1 + \alpha \cdot e^{-j\theta}} = \frac{e^{j\theta/2}}{e^{j\theta/2}} \cdot \frac{\alpha + e^{-j\theta}}{1 + \alpha \cdot e^{-j\theta}} \\
&= \frac{\alpha \cdot e^{j\theta/2} + e^{-j\theta/2}}{e^{j\theta/2} + \alpha \cdot e^{-j\theta/2}} \\
&= \frac{\alpha \cdot \{\cos(\theta/2) + j\sin(\theta/2)\} + \{\cos(\theta/2) - j\sin(\theta/2)\}}{\{\cos(\theta/2) + j\sin(\theta/2)\} + \alpha \cdot \{\cos(\theta/2) - j\sin(\theta/2)\}} \\
&= \frac{(1+\alpha) \cdot \cos(\theta/2) + j(\alpha - 1) \cdot \sin(\theta/2)}{(1+\alpha) \cdot \cos(\theta/2) - j(\alpha - 1) \cdot \sin(\theta/2)}
\end{aligned}$$

$$= \frac{r \cdot e^{+j\theta/2}}{r \cdot e^{-j\theta/2}}$$
$$= \frac{e^{+j\theta/2}}{e^{-j\theta/2}} = 1 \cdot e^{+j\theta} \tag{3.64}$$

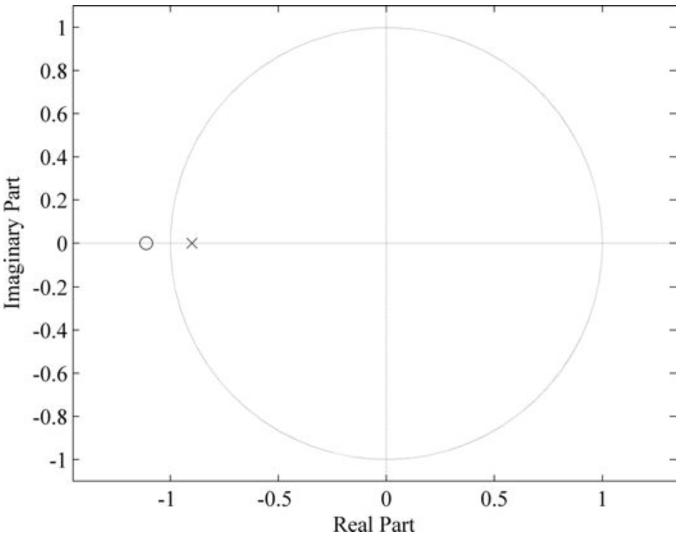


Fig. 3.33. Zero and pole configuration for 1st order all pass filter with $\alpha = 0.9$.

This trick causes the numerator and denominator to become conjugate pairs thus making our life easier in computing the frequency response $|H(e^{j\theta})|$. To simply things a bit, midway into (3.64), we let r equal the Euclidian distance, defined by the real and imaginary components as shown in Eq. (3.65) (as usual, the Euclidian distance is merely the root of the squared sum of the real and imaginary parts).

$$r = \sqrt{\{(1+\alpha)\cdot\cos(\theta/2)\}^2 + \{(\alpha-1)\cdot\sin(\theta/2)\}^2} \tag{3.65}$$

We can see that when the numerator and denominator are conjugate pairs, the magnitude response will result in unity gain yielding $|H(e^{j\theta})| = 1$. Now, jumping into the issue of phase which is the true topic of the all-pass filter, we compute the phase in terms of α in Eq. (3.66) by taking the arctangent of the imaginary and real components. Figures 3.34 and 3.35 show the phase response plots for increasing α values at 0.1 incremental steps.

$$\begin{aligned}\phi(\theta) &= \tan^{-1}\left(\frac{(\alpha-1)\cdot\sin(\theta/2)}{(\alpha+1)\cdot\cos(\theta/2)}\right) - \tan^{-1}\left(-\frac{(\alpha-1)\cdot\sin(\theta/2)}{(\alpha+1)\cdot\cos(\theta/2)}\right)\\ &= \tan^{-1}\left(\frac{(\alpha-1)}{(\alpha+1)}\cdot\tan(\theta/2)\right) + \tan^{-1}\left(\frac{(\alpha-1)}{(\alpha+1)}\cdot\tan(\theta/2)\right)\\ &= 2\cdot\tan^{-1}\left(\frac{(\alpha-1)}{(\alpha+1)}\cdot\tan(\theta/2)\right) \qquad (3.66)\end{aligned}$$

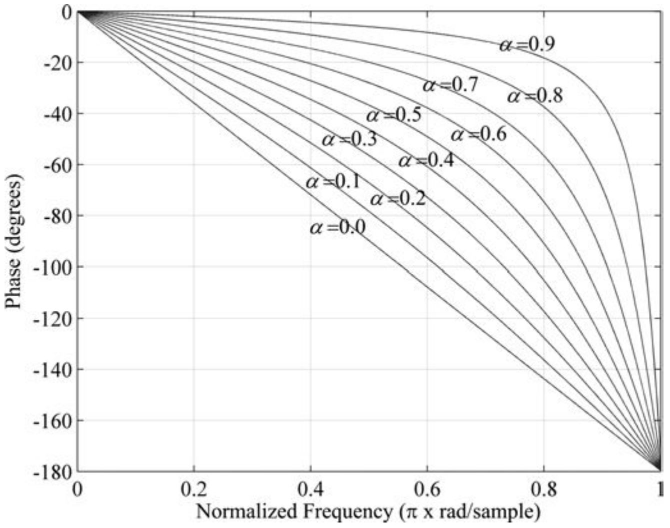


Fig. 3.34. All-pass filter phase response in degrees.

We can clearly see what effect the filter coefficient α has on the phase response in Figs. 3.34 and 3.35 — larger α values decrease the fractional delay of the resulting signal. We also note that the phase behaves almost linearly in the low to mid frequency ranges (DC to around $\pi/2$), becoming more and more nonlinear as we float towards the Nyquist frequency. Thus, an all-pass filter is often used to implement fractional delays by selecting an appropriate filter coefficient α value which is especially useful for signals that do not contain high frequency content.

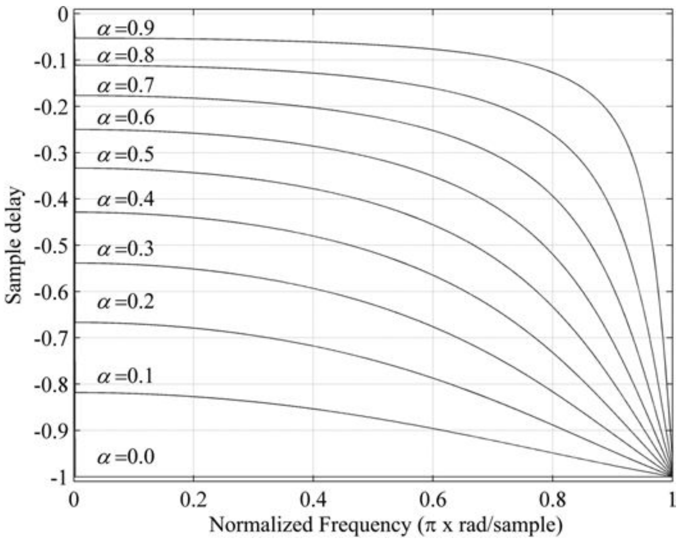


Fig. 3.35. All-pass filter phase response in fractional sample delay.

3.6.6 *Very basic all-pass filter reverb*

Reverb is a very interesting and very important part of sound propagation and sound perception. Whenever or wherever we hear sound there is a certain amount of reflection that occurs when sound waves bounce from various types of surfaces and objects. You probably have experienced sound in very reverberant spaces such as cathedrals, subway stations, banks, and your bathroom where the walls are constructed via ceramic tiles that reflect sound very well (I actually spent about 1 week installing marble tiles in our bathroom which was quite an experience — not something that I would do again — but boy, the acoustics sure changed). As seen in Fig. 3.36, reverb is perceived by the listener via the cumulative effect of the sound source's reflection from walls, ceilings, floors, and any other surfaces and obstacles in the sound wave's path. Some reflections take longer to reach the listener as they are reflected at a distant wall or are reflected numerous times before actually reaching the ear, while others reach the listener quickly. Not surprisingly, however, the sound waves that propagate longer distances or get reflected off many surfaces, generally lose more energy when reaching the listener compared to those that reach the ear more swiftly. This is an important part of artificial reverberation.

Sounds that reach the ears early are referred to as *early reflections* and sources that reach the listener directly as the *direct sound.* Early reflections

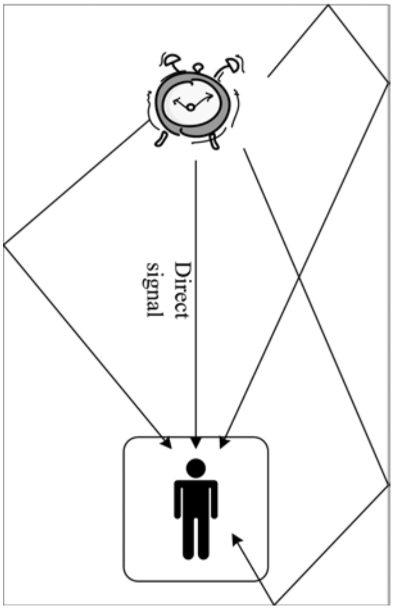


Fig. 3.36. Early reflections.

are approximately below 100 ms in duration, reaching the ear after the first few reflections and are especially helpful in providing the listener auditory cues regarding the size of a space and room. Early reflections are also commonly divided into 1st order and 2nd order reflections. These correspond to the sound wave's number of bounces from surfaces before reaching the listener. Reflections that are reflected several times fuse together and create what is commonly referred to as *reverberation.* Figure 3.37 depicts a typical plot of reverberation characteristics of an impulse signal where the x-axis shows the delay amount and the y-axis the amplitude of the reflected impulses. Figure 3.37 basically tells us that reflected images of a sound source are generally lower in amplitude, displaying a decaying amplitude envelope with increase in time delay. This makes good sense as the impulse signal would need to travel a longer distance if a wall were to be further away or if it would bounce around numerous times before reaching the ears, thus losing energy during its journey to the listener. We also note that as the delay increases (usually around 100 ms or so), a myriad of closely spaced reflected images of the impulse fuse together into a lump which in essence contributes to the formation of reverberation effect.

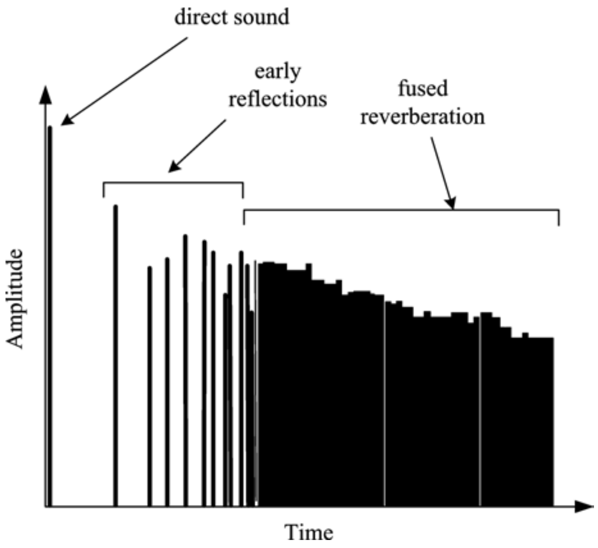


Fig. 3.37. Reverberation characteristics of an impulse signal.

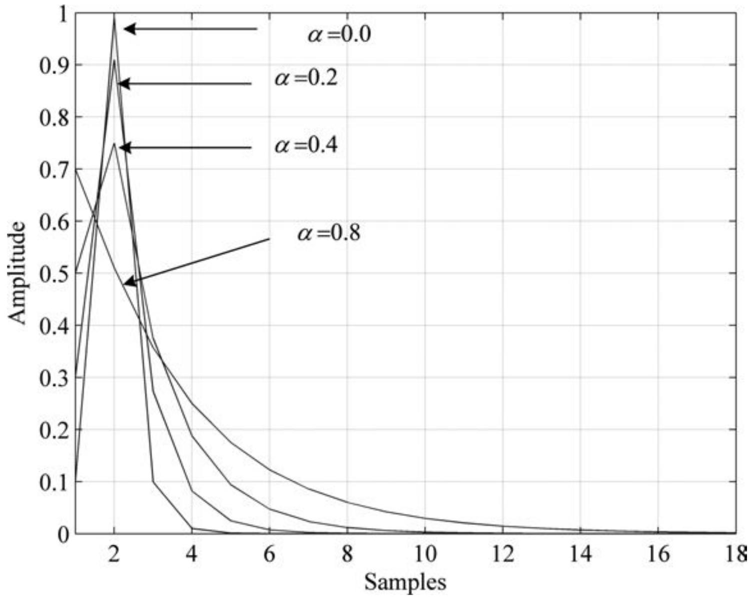


Fig. 3.38. Impulse response of all-pass filter with varying α.

Let's go back to the all-pass filter (3.62) and observe its impulse response shown in Fig. 3.38. We note two main things: the impulse response output decays smoothly and the decay characteristics are controlled via the filter coefficient α (the absolute value of impulse response is plotted to see the decaying characteristics more clearly).

Now, instead of using the original all-pass filter of Eq. (3.62), let's increase the delay length so that the feedback and feed-forward components have a larger delay N value as shown in Eq. (3.67).

$$y[n] = \alpha \cdot x[n] + x[n-N] - \alpha \cdot y[n-N] \tag{3.67}$$

When we plot the impulse response of Eq. (3.67), something interesting happens: the impulse gets delayed by N samples while retaining the decaying quality of the original all-pass filter as shown in Figs. 3.39, 3.40, and 3.41.

Thus, the all-pass filter of a higher order than 1 can be used to somewhat mimic the reflection characteristics of a sound bouncing off surfaces and is exploited as a popular approach in devising artificial reverberation-based algorithms that are not convolution-based designs. Notice how nicely the impulse decays as the time index increases as if it were reflecting off distant/close walls.

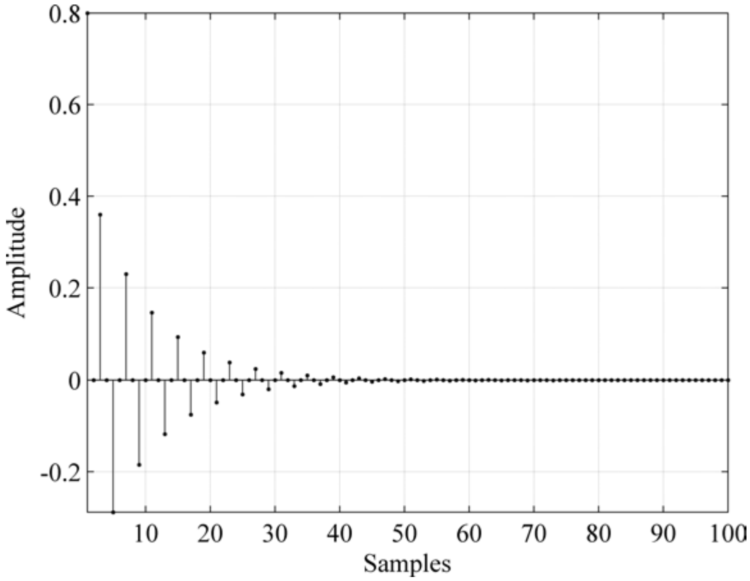


Fig. 3.39. All-pass filter $N = 1$ (original all-pass filter), $\alpha = 0.8$.

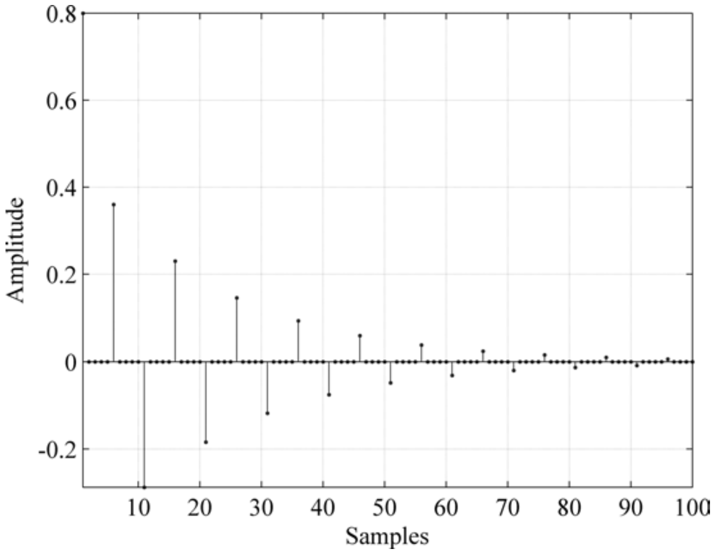


Fig. 3.40. All-pass filter $N = 4$, $\alpha = 0.8$.

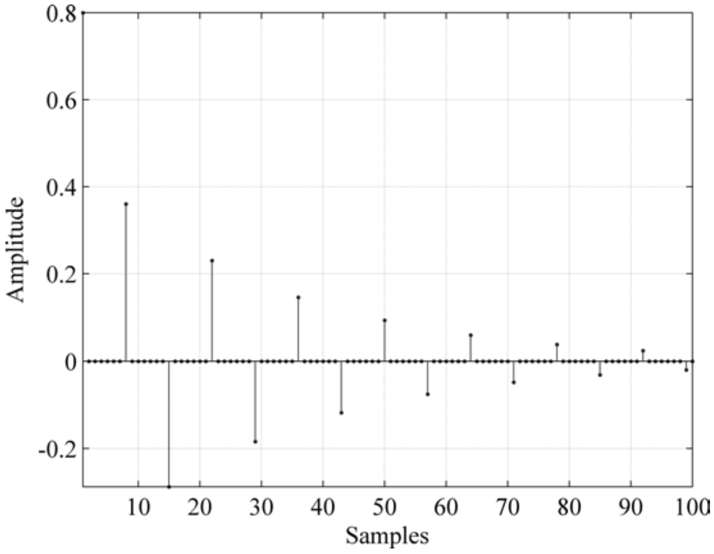


Fig. 3.41. All-pass filter $N = 6$, $\alpha = 0.8$.

Below is a quick MATLAB® implementation of a very basic and simple all-pass filter reverb algorithm.

```
a    = 0.9;    % the all pass coefficient
fs   = 20000;  % sampling frequency
ratio = 1/25;  % 1/25th of a second for reflection of sound

% reflection length in samples: filter order
reflectionLength = round(fs*ratio);

% impulse response
x = rand(1,fs/20);        % make short impulse signal 1/20th sec
x = x-mean (x);           % get rid of DC
x = [x zeros(1,fs*2)];    % pad with zeros to make actual impulse

% filter signal via all-pass filter to make reverb
y = filter ([a zeros (1, reflectionLength) 1], ...
        [1 zeros (1, reflectionLength) a], x);

soundsc (y, fs)
```

Code Example 3.10. Basic all-pass filter reverb.

In this MATLAB® example, we have used 1/25th of a second for the reflection parameter in the implementation of the artificial reverb using the all-pass filter. The script should be straightforward to understand as we already have acquainted ourselves with all of the functions and methodologies used in Code Example 3.10 in previous code examples. This basic implementation of all-pass reverb may not sound as realistic as one would like it to — this is the case with most, if not all rudimentary DSP algorithms — they require further tweaking.

Manfred Schröder (Schroeder and Logan 1961, Schroeder 1962) is largely credited for the research in artificial reverb and started experimenting with the all-pass filters and concatenating multiple all-pass filters structures in series to get better results (his name should actually be used with the umlaut ö, but you will find more hits on the Internet with "oe"). As shown in Fig. 3.42, he also used a combination of parallel comb-filters followed by all-pass filters in series which does a much better job than the basic one we presented here with just a single all-pass filter.

The topic of artificial reverb has developed remarkably since Schröder's early research (it has come a long way since playing back recorded sound into an echo chamber and recording the reverb imposed on the original dry sound through a microphone), including using a low-pass filter in the

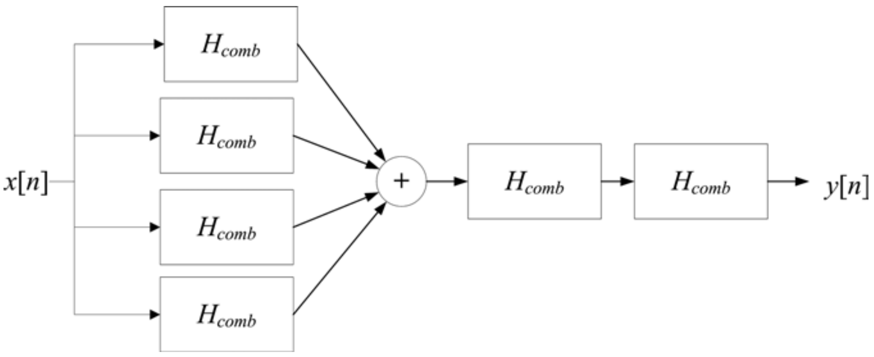


Fig. 3.42. Schröder model for digital reverberation.

feedback loop, much like the plucked string algorithm, to bias the decay of higher frequency components. Other innovations, tweaks, and additions of advanced algorithms include the modeling of specific room sizes, room geometry, sound source, and listener location.

4 Musical Examples

There are many interesting musical examples using filters and it is no exaggeration that all of the music recorded and produced today goes through some sort of filter. It may be a song by Madonna such as *Music* (2000) where the vocal part is swept with a band-pass filter or Jimmy Hendrix's guitar modulated through the *JH-1 Dunlop Cry Baby* wah-wah pedal. However, one very interesting piece that uses the comb-filter extensively is Paul Lansky's *Night Traffic* (1990). In this piece, the core sounds originate from Route 1 in the Princeton, New Jersey area where the composer recorded sounds generated from automobiles that sped on a two-lane per direction highway. Paul Lansky uses the comb-filter as a means to and end — creating a musical work rather than a demonstration of the possibilities of comb-filters *per se*. It is evident that the composer plays with the notion of identity and origin of the sound sources (and the blurring thereof) but goes beyond the exploration of inherent noisiness of the whizzing cars and trucks or the mere sequencing, organizing, and overlaying of those found sounds. Lansky intricately drives the comb-filters via the quickly-becoming-boring traffic noise to infuse musicality and drama into the sound sources, thereby opening up a new auditory world and presenting a theatrical and musical experience of the otherwise abstract nature of the vehicles passing by. In a sense, the traffic patterns and motifs are used as gestures that are ambiguous enough not to give away its absolute identity while rendering musical strokes that seemingly access certain parts of one's long/short term memory banks, though, perhaps in an unfamiliar fashion. A life lesson to be learned from the process of making this piece is that for field recording outings, it is of utmost importance to have your headphones on your head the correct way. It seems that when Paul Lansky first started recording the cars speeding by on Route 1, he was wearing the headphones backwards — left ear being enclosed by the right headphone and right ear by the left headphone. As the composer was using a stereo microphone he almost walked into the wrong direction and towards the path of the speeding vehicles as he was distracted by the sounds that engulfed his headphones...

Another exciting composer relevant to our topic in this chapter is Alvin Lucier. The composer is maybe best-known for his regular utilization of somewhat atypical approaches to musical composition, where at times the process itself almost resembles a physics experiment more than a compositional exercise. One of his landmark pieces is called *I am Sitting in a Room* (1969). The piece is self-explanatory in the truest sense of the word as the composer narrates the process of the work in the recording which in itself comprises part of the composition. The text is as follows:

"I am sitting in a room different from the one you are in now. I am recording the sound of my speaking voice and I am going to play it back into the room again and again until the resonant frequencies of the room reinforce themselves so that any semblance of my speech with perhaps the exception of rhythm, is destroyed. What you will hear, then, are the natural resonant frequencies of the room articulated by speech. I regard this activity not so much as a demonstration of a physical fact, but more as a way to smooth out any irregularities my speech might have."

What is fascinating is that with the above mentioned iterative process, the recorded voice, reflecting the resonant structure of the room, becomes unintelligible towards the end, only retaining the basic gesture of the original speech pattern and becoming more and more harmonic — in a sense more and more musical. Thus, although the piece begins with the familiar voice, by the closing stages of the piece, the voice modulates to something completely different — the resonant structure of the room repeatedly filters the voice into submission. This technique of exploiting the resonant structure for musical purposes is somewhat similar to the talk box. In the case of the talk box, the source signal (often an electric guitar) is fed to a resonant box, the box being the vocal cavity that modulates the signal according to the resonant structure of one's vocal cavity. Talk boxes can sometimes be seen at concerts where the source signal such as an electrical guitar is literally piped through a small hose directed towards the mouth which in turn is attached close to a vocal microphone. With this configuration, the source signal (guitar) is projected into the vocalist's mouth, thus modulating the source signal by altering the vocal cavity's shape. The resulting modulated sound is picked up by the microphone and amplified through the loudspeakers, making the guitar seemingly talk. Numerous musicians have used this effect including Pink Floyd (*Pigs* 1977), Bon Jovi (*Livin' on a Prayer* 1986), and Nick Didkovsky (*Tube Mouth Bow String* 2007) who used four *Banshee Talkboxes* to compose a piece for string quartet. Speaking of banshee, a monstrous piece that has little to do with DSP is Henry Cowell's *The Banshee* (1925) for solo piano — in this work, the modulation of the traditional piano sound does not happen electronically but literally via hands-on techniques with the performer modulating the familiar piano timbre via sweeping, scraping, strumming, and plucking the strings directly. As they say, "there's more than one way to skin a cat."

References and Further Reading

Frerking Marvin, E. 1994. Digital Signal Processing in Communication Systems. Vab Nostrand Reinhold. New York, NY, USA.

Karplus, K. and Strong A. 1983. "Digital Synthesis of Plucked String and Drum Timbres", Computer Music Journal, 7(2), 43–55.

Jaffe, D. and Smith J. O. 1983. "Extensions of the Karplus-Strong Plucked String Algorithm", Computer Music Journal, 7(2), 56–69.

Schroeder, M. R. and Logan B. F. 1961. "Colorless Artificial Reverberation", Journal of the Audio Engineering Society, 9(3), 192.

Schroeder, M. R. 1962. "Natural-sounding artificial reverberation", Journal of the Audio Engineering Society, 10(3), 219–233.

Moore, F. R. 1990. Elements of Computer Music, Prentice Hall: Englewood Cliffs, NJ, USA.

Moorer, J. 1979. "About this Reverberation Business", Computer Music Journal, 3(2), 13–28.

Morse, P. M. 1936. Vibration and Sound. Published by the American Institute of Physics for the Acoustical Society of America: Woodbury, New York USA, 1976 (1st ed., 1936; 2nd ed., 1948).

Park, T. H. and Li Z. 2008. "All this Buzz about the Sitar: Physical Modelingly Speaking", Proceedings of the 14th International Congress on Sound and Vibration, Daejon, South Korea.

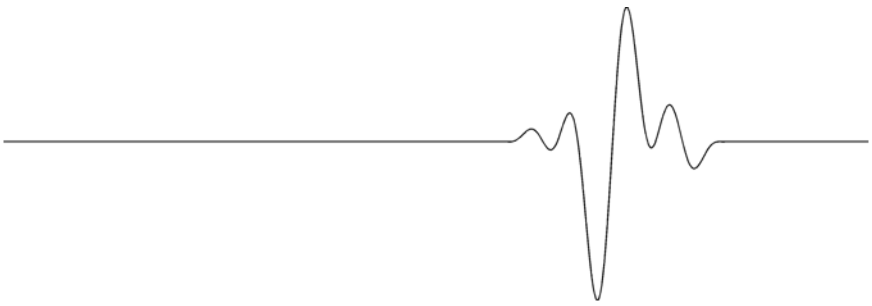
Porat, Bocz 1997. A Course in Digital Signal Processing. Wiley and Sons.

Smith, J. O. 1992. "Physical Modeling Using Digital Waveguides", Computer Music Journal, 16(4), 74–91.

Sullivan, Chcrlie R. 1990. "Extending the Karplus-Strong Algorithm to Synthesize Electric Guitar Timbres with Distortion and Feedback", Computer Music Journal 14(3), 26–37.

Chapter 8

FREQUENCY-DOMAIN AND THE FOURIER TRANSFORM



1 Introduction

Up to this point we have mainly concentrated on topics based in the time-domain while at the same time having gotten glimpses of frequency-domain related concepts in bits and pieces in previous chapters. In this chapter, we will fully engage into theories and issues concerning the frequency-domain. The backbone behind concepts of the frequency-domain revolves around a very important topic in signal processing called the Fourier transform. It is an understatement to say that it is one of the most powerful concepts in DSP (as well as other fields of study such as mathematics, physics, electrical engineering, computer science, . . .) and important for various signal processing applications including sound synthesis, sound re-synthesis, sound manipulation, filtering, and spectral analysis. Although we have talked about numerous issues concerning frequency components comprising a sound object and how sound (or any signal for that matter) can be described via specific sinusoidal components, we have had no tools to determine the specificity of those components. However, with the Fourier transform, we will finally be able to describe and compute each of those sinusoids' magnitude, frequency, and phase values.

2 Additive Synthesis

Before we get into the crux of the Fourier transform let us first introduce *additive synthesis*. This initial introduction will help us better understand the underlying concept behind the Fourier transform itself. Additive synthesis can be regarded as the opposite of subtractive synthesis introduced in Chap. 7, Sec. 2.1. In subtractive synthesis, we started out with a rich and complex signal such as a square wave or white noise signal and sculpted away specific frequency regions, basically making the resultant signal less rich. In additive synthesis however, the fundamental approach is not carving out frequency regions, but rather engaging in a task that is quite the opposite — constructing a signal by summing specific sinusoids. That is, rendering a sound object via simple sinusoids (which we could loosely refer to as building blocks of sound) accomplished through the adding of particular sinusoids with specific frequency, amplitude, and phase values.

The classic example for demonstrating additive synthesis is the synthesis of the square wave, as approximated according to Eq. (2.1) where f is frequency in Hz, k an integer number determining each sine wave's frequency component, and n the sample index ($k = 0$ refers to the fundamental frequency).

$$y[n] = \sum_{k=0}^{K} \frac{\sin\{2\pi \cdot f \cdot (2k+1) \cdot n\}}{2k+1} \tag{2.1}$$

Upon closer inspection, you will recognize that the frequencies used in (2.1) are odd frequencies only. Thus, the resulting output $y[n]$ becomes a complex signal limited to a collection of odd harmonics and its fundamental frequency. In addition, you may have also noted that the amplitude values decrease with the increasing harmonic index k, characterized by the same $2 \cdot k + 1$ divisor that we used to define the structure for the harmonics.

Figures 2.1–2.3 show plots of the square wave additive synthesis algorithm in action using 1, 3, 10, 50, 100, and 200 harmonics (K) plus a fundamental frequency at $f_0 = 1$ Hz. The upper plots of each figure (two sub plots each) display the individual sine waves with increasing harmonic frequencies (fundamental and subsequent harmonics) that make up the complex signal. The lower plots of each figure show those summed

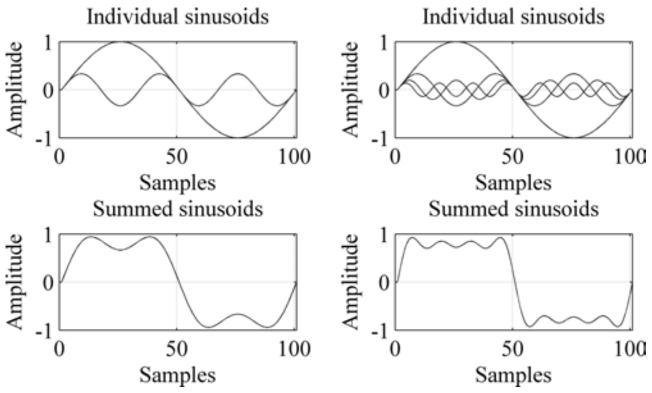


Fig. 2.1. Additive synthesis of square wave with 1 (left) and 3 (right) harmonics.

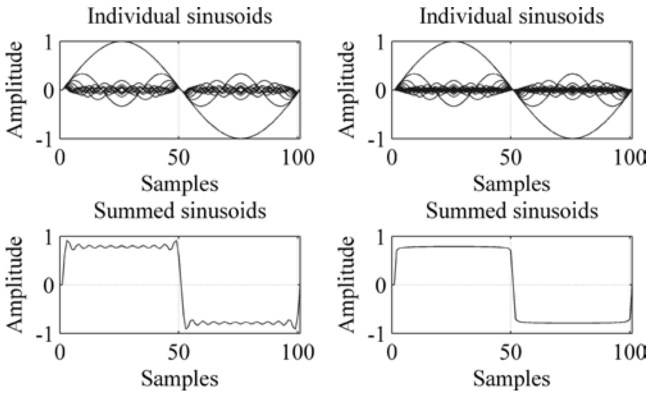


Fig. 2.2. Additive synthesis of square wave with 10 (left) and 50 (right) harmonics.

sinusoids rendering as a single waveform (x-axis is as usual samples and y-axis amplitude).

We can clearly see that the more sinusoids (K becoming larger) we employ, the closer the resulting waveform will look and sound like a square wave. This intuitively makes sense as the long waveforms, corresponding to lower frequency-based sinusoids, only contribute to the general shape of the final signal — they oscillate slowly compared to higher frequency sinusoids. That is, higher frequency sinusoids oscillate faster (with smaller amplitude values in our example) within a given timeframe and take the role of fine tuning the synthesized output by making sharper edges which characterize square waves.

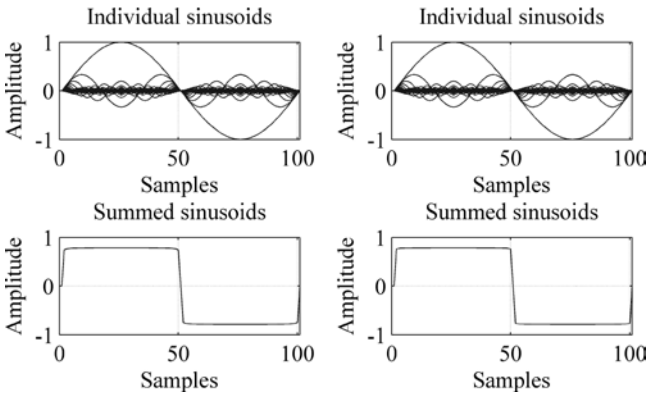


Fig. 2.3. Additive synthesis of square wave with 100 (left) and 200 (right) harmonics.

The triangular wave is very much like the square wave in its sinusoidal make-up and also consists of odd harmonics. However, the harmonics roll off exponentially as shown in Eq. (2.2).

$$y[n] = \frac{8}{\pi^2}\sum_{k=0}^{K}\sin\left\{\frac{\pi\cdot(2k+1)}{2}\right\}\cdot\left\{\frac{\sin\{2\pi\cdot f\cdot(2k+1)\cdot n\}}{(2k+1)^2}\right\} \tag{2.2}$$

Yet another complex wave called the sawtooth wave can be approxi- mated with Eq. (2.3) where in this case, the full harmonic series is included (odd and even). Like in our square wave example, in general, more sine waves contribute to a better approximation of the sawtooth wave.

$$y[n] = \frac{2}{\pi}\cdot\sum_{k=1}^{K}\sin\left\{\frac{\pi\cdot k}{2}\right\}\cdot\left\{\frac{\sin\{2\pi\cdot f\cdot k\cdot n\}}{k}\right\} \tag{2.3}$$

Interestingly enough, additive synthesis did not (and does not) always exist in the form of electronic synthesizers that we are accustomed to, whether outfitted as huge machines with 10,000 cables hanging about or more slick and compact digital additive synthesizers with a few elegantly looking buttons and a data-wheel. The popular Hammond organ is such an additive synthesizer which utilized "tonewheels" (91 discs) that spun to mechanically produce specific "sine waves" controlled via nine drawbars, each in turn controlling nine levels of amplitude values. So what does all of this have to do with the Fourier transform? Read on to find out more ...

3 The Fourier Transform

The *Fourier transform* is named in honor of Jean Baptiste Joseph Fourier who is largely credited for the development of its theory. In a nutshell, the Fourier transform decomposes a signal into an infinite number of weighted sinusoids with specific frequency and phase values and was developed while trying to solve the mathematical solution to heat transfer in the early 1800. In the case of additive synthesis and approximation of the square wave, these weights and frequencies correspond to the decreasing amplitude values and odd harmonics.

The *continuous-time* and *continuous-frequency Fourier transform* is shown in (3.1). $X^F(\omega)$ is the resulting Fourier transform, ω angular frequency (radians per second), and $x(t)$ the continuous-time input signal.

$$X^F(\omega) = \int_{-\infty}^{+\infty} x(t) \cdot e^{-j\omega t} dt, \quad \omega \in \Re \tag{3.1}$$

The above definition of the Fourier transform will exist, provided that the integral is bounded. Thus, a sufficient condition for the existence of the Fourier transform is as follows:

$$\int_{-\infty}^{+\infty} |x(t)| dt < \infty \tag{3.2}$$

The condition in (3.2) basically limits the input signal so that the integral of the input doesn't blow up much like an unstable IIR filter. In order to recover the time-domain signal and return from the frequency-domain back to the time-domain, we have the inverse Fourier transform as shown in (3.3).

$$x(t) = \frac{1}{2\pi} \int_{-\infty}^{+\infty} X^F(\omega) \cdot e^{+j\omega t} d\omega, \quad t \in \Re \tag{3.3}$$

The above is essentially the continuous form of the digital square wave additive synthesizer we encountered in Sec. 2 — adding the sinusoids with specific frequency and phase values to produce a desired time-domain signal (the square wave). If we did the opposite, that is, perform a forward Fourier transform on a square wave signal, we would obtain the fundamental frequency and its harmonic structure in terms of its weights, frequency, and phase values — the characteristic square wave architecture with odd harmonics and idiosyncratic decaying amplitude values described in Eq. (2.1).

4 The Discrete-Time Fourier Transform (DTFT) and the Discrete Fourier Transform (DFT)

The continuous-time and continuous-frequency Fourier transform in (3.1) uses integration instead of summation which is of course impractical for computer based applications. Hence, we need to develop a discrete-time and discrete-frequency version for it. When we sample the input signal and convert the integral to a summation (4.1) and ω to $\theta \cdot n$ in (4.2), we have what is called the *discrete-time Fourier transform* (DTFT). The DTFT is thus computed according to (4.3). The final step towards a fully discrete version of the Fourier transform is sampling the frequency axis which will result in the *discrete Fourier transform* (DFT). We will discuss how this final step is taken shortly. For now, let's concentrate on the DTFT. The DTFT exists provided that the summation is bounded and is sufficiently met by condition (4.4) similar to the continuous-time Fourier transform.

$$\int_{-\infty}^{+\infty} \to \sum_{n=-\infty}^{n=+\infty} \tag{4.1}$$

$$e^{j\omega} \to e^{j\theta n} \tag{4.2}$$

$$X^f(\theta) = \sum_{n=-\infty}^{n=+\infty} x[n] \cdot e^{-j\theta n} \tag{4.3}$$

$$\sum_{-\infty}^{+\infty} |x[n]| < \infty \tag{4.4}$$

The inverse Fourier transform of the DTFT is given by Eq. (4.5).

$$x[n] = \frac{1}{2\pi} \int_{-\pi}^{+\pi} X^f(\theta) \cdot e^{+j\theta n} d\theta, \quad n \in Z \tag{4.5}$$

Note that the angular frequency in the DTFT is not yet discrete, meaning that it can take any real value (the discrete frequency is relevant to the DFT discussed shortly). Thus, in the DTFT, we use the digital frequency θ in radians/samples as we have done in previous chapters. The term digital frequency may be a bit misleading at first ($\theta = \omega \cdot T$) as when we think digital, we have the tendency to automatically think discrete. However, what the digital frequency actually refers to is a frequency value at a specific discrete-time $n \cdot T$ which is used to differentiate it from the continuous-time frequency and the discrete frequency as we will see when we introduce the DFT.

The DTFT operates on sampled and windowed input signals of window size N. We know what happens when sampling the time axis for a continuous signal such as $x(t)$ and should be comfortable with that concept, but sampling the frequency axis, however, may be a little more confusing. In order to get a better grasp of how the frequency axis is sampled, it is important to grapple with the idea of the Fourier transform itself and how it maps a windowed time-domain signal to a frequency-domain spectrum. So, before embarking on describing the details and the math behind the DFT and frequency axis sampling, let's first get acquainted with the Fourier transform's workings, what it represents, and observe what happens when a windowed time-domain frame gets converted to a frequency-domain frame. Figure 4.1 shows the procedure of the discrete Fourier transform with $x[n]$ as the input signal and $X^d[k]$ the resulting discrete Fourier transformed spectrum with discrete frequency index k.

Let's say our sampled time-domain signal $x[n]$ looks something like Fig. 4.2, consisting of 1,536 samples and sampled at 44.1 kHz representing

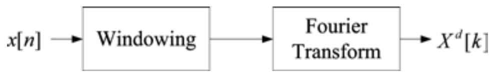


Fig. 4.1. Fourier transform procedure.

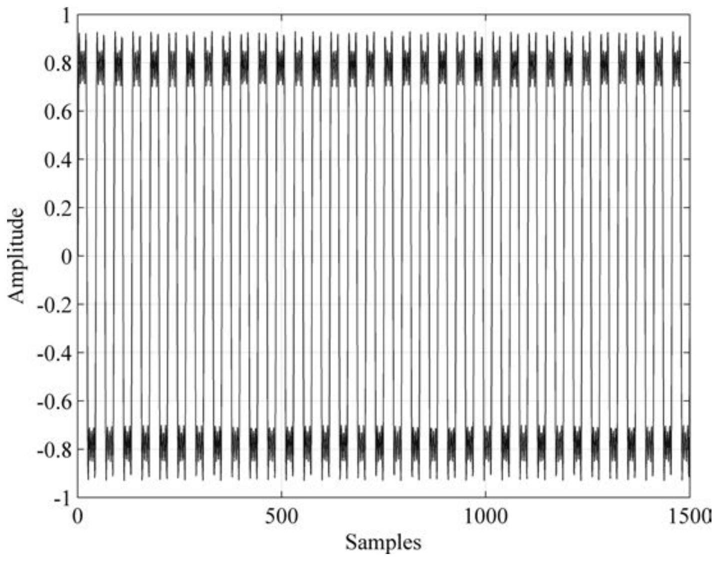


Fig. 4.2. Square waveform using 3 harmonics and 1,536 samples in duration.

a 1 kHz square wave synthesized via additive synthesis with 3 odd harmonics. If we follow the Fourier transform procedure for the DFT, we will first need to window the time-domain signal as shown in Fig. 4.3. In this example, we use a rectangular window of length $N = 512$ samples and divide the input signal $x[\cdot]$ into three equal frames of 512 samples. This is depicted in the middle of Fig. 4.3 where the first windowed signal is shown.

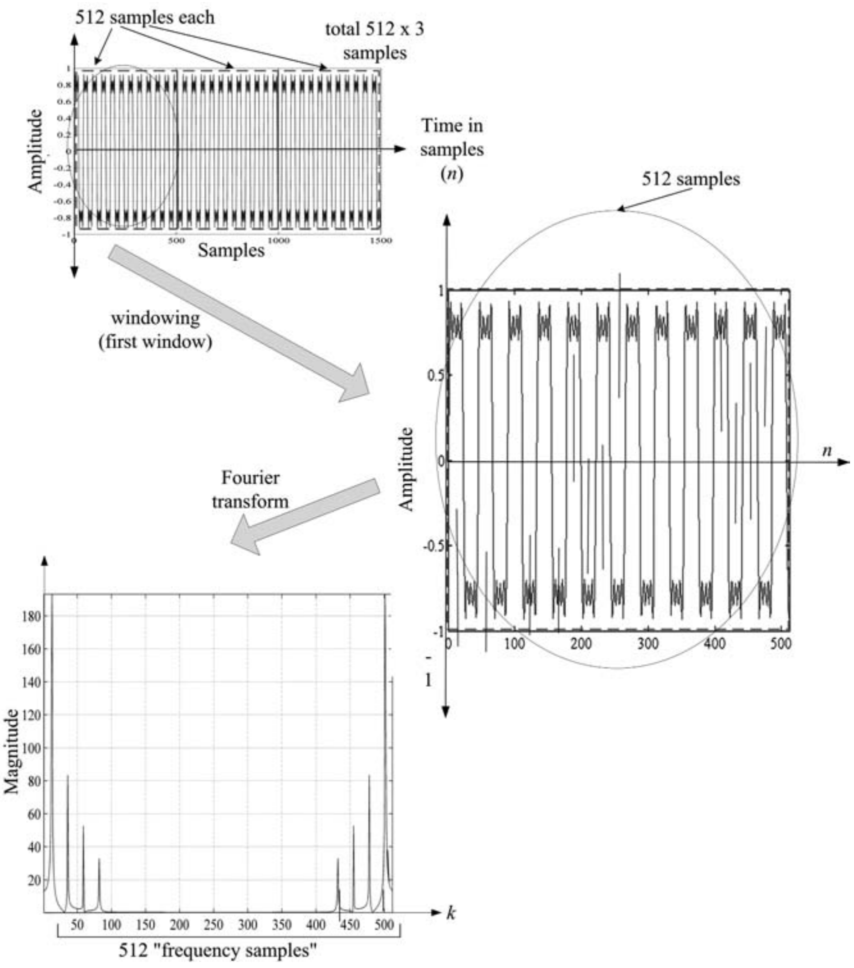


Fig. 4.3. Time-domain to frequency-domain transformation.

At the bottom part of Fig. 4.3, the Fourier transform takes place and the time-domain frame of 512 samples gets transformed to a frequency-domain representation. Notice one very important aspect of this transformation is a mapping of 512 time-domain samples to 512 frequency-domain samples. In other words, the x-axis which used to be time (sample index n) now gets converted to frequency with frequency index k — the x-axis transforms from a time-domain axis to a frequency-domain axis. These windowed time sequences of 512 samples are each represented by 512 frequency samples making up a *frame* after the Fourier transform. Each frame which consists of 512 frequency samples is referred to as a spectrum. The discrete points on the x-axis (frequency axis) are commonly referred to as frequency bins or just bins.

If it is still a little unclear what the Fourier transform actually represents, for the sake of argument, let's consider the RGB color model which uses a combination of various degrees of red, green, and blue to produce a wide range of colors. On the computer, each RGB color typically is represented by an integer number ranging from 0 to 255 (8 bits). Thus, when these so-called primary colors are combined and weighted appropriately, a plethora of additional shades of colors can be produced. Now, let's consider a motion-picture film that uses the 24 picture-frames per second standard rendering the illusion of continuous motion when projected on the screen, much like a sequence of samples (44,1000 samples/sec for example) give us the illusion of continuous sound. Going back to the RGB idea, let us pretend we want to analyze the color map of a portion of a film — 6 frames at a time. That is, apply an analysis window size of 6 picture frames equivalent to 1/4 of a second (if we use the 24 frames/sec standard) and have the result be represented in terms of the integer levels of red, green, and blue for those 6 frames as a whole. In this analysis setting, we would get an idea of the RGB content for every quarter of a second for the whole film. In essence, we go from a time-domain system of moving images to a "primary color-domain" system, with the color-domain system representing only the RBG levels for each group of frames being analyzed (we could of course have used more frames or less frames for the analysis part). Thus, the "x axis" can be viewed to change from a time-domain moving picture representation to a color-domain RGB representation scheme as shown in Fig. 4.4.

The Fourier transform can loosely be compared to the above example where instead of using six picture frames, we used 512 time-samples for each frame and in place of the 3 primary colors with various weights in the RGB case, we used 512 frequency components with their respective magnitude and phase values. Each of these frequency components represent sinusoids with unique frequency values and their respective magnitude values characterizing the time-domain windowed signal of 512 samples. Thus, the 512 sample windowed signal becomes a frequency-domain representation of 512 sinusoids with their corresponding 512 frequency, magnitude, and phase values. In summary, these 512 frequency-domain sinusoids provide an overview of the 512 time-domain samples in terms of a collection (sum) of sinusoids.

Now, returning to the 3 odd harmonics-based square wave additive synthesis example and its Fourier transform shown in Fig. 4.5, we actually see 8 sinusoids (strong peaks) rather than 4 — twice as many as a matter of fact. This seems a bit strange as we would expect one fundamental

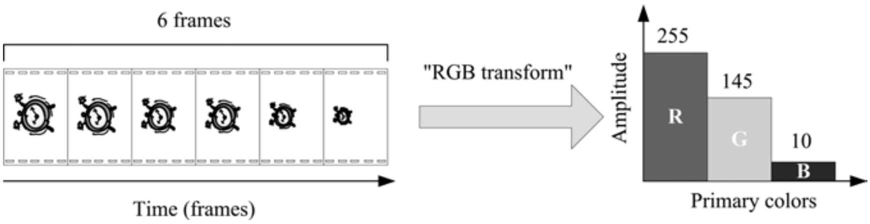


Fig. 4.4. RGB levels of the first six frames of a color cartoon.

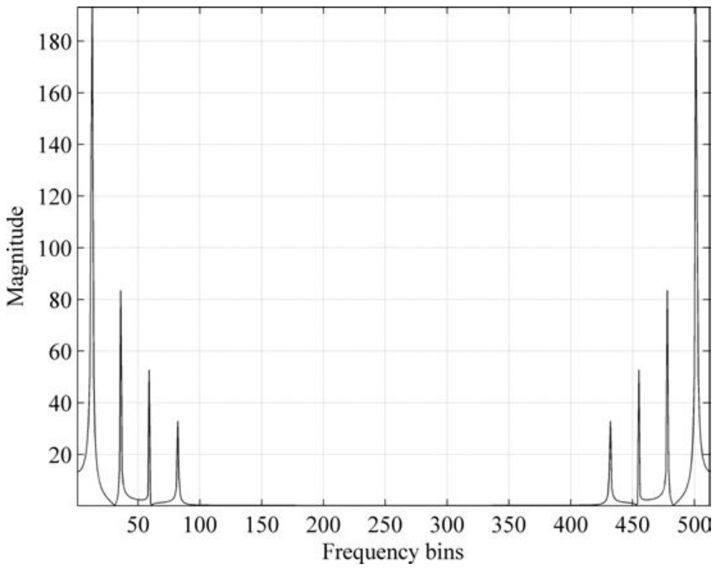


Fig. 4.5. Fourier transform of square wave.

frequency component and three odd harmonics. Upon closer scrutiny, we will also notice that the first 4 peaks of the spectrum seem to resemble the latter 4 peaks in a mirror reflection-like manner. We will see why this happens shortly, but it should not be too much of a leap to imagine that the Euler identity (made up of sines and cosines) has something to do with this (similar to when we did our frequency response analysis). If we accept that there is some sort of repetition in the spectrum, it should be evident that only half of the full-range of 512 bins is of interest to us. Thus, as expected, if we look at the first 256 frequency slots, we can clearly see the fundamental and the first three odd harmonics with decreasing magnitude values in Fig. 4.6.

Let's now look at the mathematical details behind the DFT by starting with the discrete-time Fourier transform (DTFT) with discrete time index n and angular digital frequency θ in radians per sample.

$$X^f(\theta) = \sum_{n=-\infty}^{n=+\infty} x[n] \cdot e^{-j\theta n} \tag{4.6}$$

where

$$e^{-j\theta n} = \cos(\theta \cdot n) - j\sin(\theta \cdot n) \tag{4.7}$$

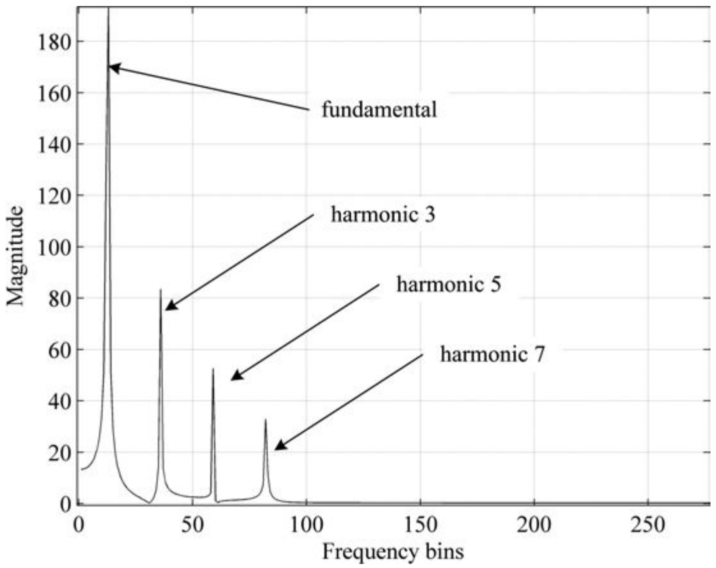


Fig. 4.6. Frequency bins of interest displaying fundamental and 3 harmonics.

We now *sample* the frequency axis (as did for the time axis) which extends from 0 to 2π equivalent 0 to f_s Hz by dividing this range into N equally spaced frequency slots according to Eq. (4.8).

$$\text{unit frequency} = \frac{2\pi}{N} \tag{4.8}$$

Note that at this point, just like in time-domain sampling, we are performing sampling albeit in the frequency-domain. That is, in time-domain sampling we had T (in seconds) representing the sampling interval between samples and here we have $2\pi/N$ representing the frequency sampling interval (units in frequency). Thus, the sampled digital frequency θ can be represented as $\theta[k]$, with integer k determining the discrete frequency index (the frequency index is also often referred to as frequency bin or just bin). This is shown below in Eq. (4.9) for a DFT window length of N (in our previous example $N = 512$).

$$\theta[k] = \frac{2\pi k}{N}, \quad 0 \le k \le N-1 \tag{4.9}$$

Replacing θ in Eq. (4.6) with (4.9), we finally arrive at the DFT algorithm shown in Eq. (4.10) where $X^d[k]$ is the DFT for discrete frequency bin/index k, input sample $x[n]$ with discrete time index n, and window length in time samples N.

$$X^d[k] = \sum_{n=0}^{n=N-1} x[n] \cdot e^{-j\frac{2\pi k n}{N}}, \quad 0 \le k \le N-1 \tag{4.10}$$

Also, notice that when we plug in the range for k into Eq. (4.9), $\theta[k]$ has the range shown in (4.11).

$$0 \le \theta[k] \le \frac{2\pi k(N-1)}{N} \tag{4.11}$$

This means that the range of the sampled frequency $\theta[k]$ is from 0 up to 2π minus one frequency sampling unit of $2\pi/N$. We do not include the Nth frequency component itself as that would be equal to the zero frequency component (DC) which is already in the spectrum. Observe also that the number of time samples of the windowed input signal always equals the total number of bins (frequency samples).

The inverse DFT is given in Eq. (4.12) and it is common to use the notation W_N when dealing with the DFT or the IDFT for convenience as shown below:

$$x[n] = \frac{1}{N}\sum_{k=0}^{N-1} X^d[k] \cdot e^{+j\frac{2\pi kn}{N}} \tag{4.12}$$

$$W_N = e^{j2\pi/N} \tag{4.13}$$

Using this shorthand, the DFT can also be written as (4.14).

$$X^d[k] = \sum_{n=0}^{n=N-1} x[n] \cdot e^{-j\frac{2\pi kn}{N}} = \sum_{n=0}^{n=N-1} x[n] \cdot W_N^{-kn} \tag{4.14}$$

4.1 *Magnitude, Phase, and Other Basic Properties of the DFT*

As you may have already deduced by looking at the DFT, the output $X^d[k]$ will be a complex number due to the Euler identify — the DFT plots shown here are the magnitude plots. Since the DFT output is a complex number-based vector following the general form of Eq. (4.15), the magnitude which is commonly used to analyze the DFT, is computed according to (4.16) for bin k, and the phase as the arctangent as shown in (4.17).

$$X^d[k] = a_k + jb_k \tag{4.15}$$

$$\left|X^d[k]\right| = \sqrt{a_k^2 + b_k^2} \tag{4.16}$$

$$\angle X^d[k] = \arctan\frac{b_k}{a_k} \tag{4.17}$$

Figure 4.7 shows the first few harmonics of the additive synthesis square wave with a DFT size of 32 samples. As mentioned in the previous section, due to the (almost) mirror image characteristics of the DFT at $N/2$, our area of interest for each frame needs only be from $k = 0$ to $k = 15$ in this example — from DC (bin 0) to Nyquist (bin $N/2 - 1$). After the 16th bin ($k = 15$) we can almost observe an exact mirror image, the inexactitude being the absence of the DC component at the far right end of the frequency axis as seen in Fig. 4.7.

Figure 4.7 also shows three different units of the frequency axis all referring to the same thing further summarized in Table 4.1. For a $f_s = 44.1$ kHz system, bin 15, in terms of angular frequency $\theta[15]$ and Hertz will be 2.94 radians and 20,671.875 Hz respectively. One "click" further to the right will result in π radians or 22,050 Hz.

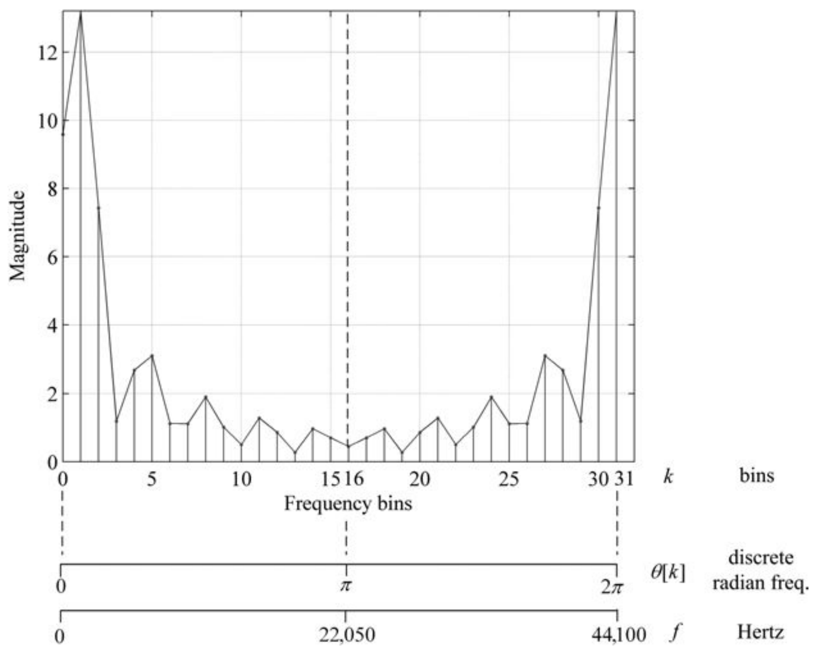


Fig. 4.7. Symmetry characteristics of DFT for $N = 32$ and $f_s = 44.1$ kHz.

Table 4.1. Frequency axis units.

Units	Description
Frequency bin k	$k = 0 \ldots N-1$ where N is an integer number DFT length
Sampled angular frequency $\theta[k]$	$\theta[k] = 2\pi k/N$
Frequency in Hertz	$f_{Hz} = f_s \cdot (k/N)$ where f_s is the sampling frequency

4.2 *Time Resolution vs. Frequency Resolution in DFTs*

In the previous section's Fig. 4.7, we mentioned that we used the first few harmonics and purposely did not give the exact number of harmonics that were used for synthesis. Upon first glance, we could maybe say that we see four harmonics and a fundamental frequency component, as we can with relative certainty make out 5 peaks. However, this is not correct. If we applied a smaller frequency sampling interval, that is, a larger DFT window

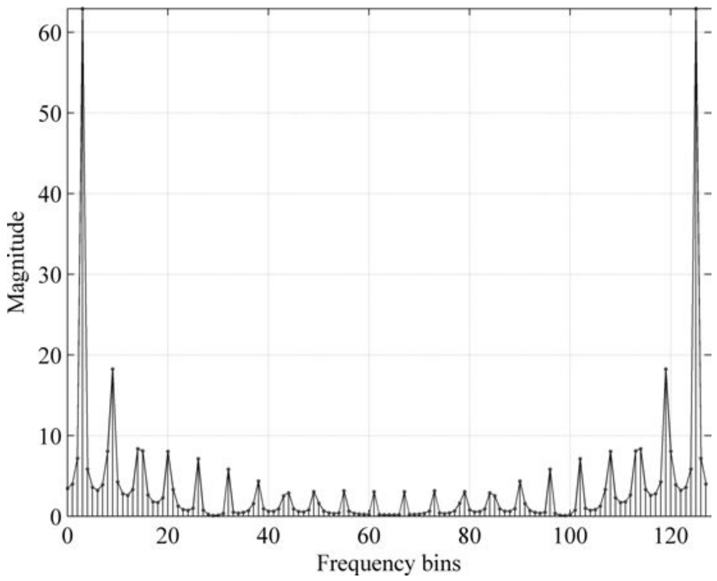


Fig. 4.8. Square wave where 10 harmonics are clearly visible with DFT size 128.

size of say 128 samples, we arrive at a very different spectral frame for the same signal as seen in Fig. 4.8.

By increasing the DFT size we have as a consequence increased the frequency resolution, or stated inversely, decreased the frequency axis sampling interval as defined in Eq. (4.9). Jumping from $N = 32$ to $N = 128$ results in dividing the frequency axis into equally spaced 128 bins rather than just 32 ($44100/128 = 344.53$ Hz opposed to $44100/32 = 1378.125$ Hz). We could actually go even higher to say 1,024 samples for example and obtain a frequency axis sampling interval of $44100/1024 = 43.07$ Hz. Then why, you may ask, do we not always use a "large(er)" window size? If we carefully observe what is happening during the process of changing the window size, we will recognize that as we increase the length of the window (increase the frequency resolution) we require more time-domain samples. That is, if we choose a window size of 1,024 samples we do get a narrow 43.07 Hz frequency resolution in the frequency-domain. However, as a consequence of using 1,024 samples (equivalent to approximately $1024/44100 = 23.21$ ms of samples) we are requiring a larger time excerpt than if we had used 32 time samples ($32/44100 = 0.73$ ms). In other words, from the time-domain perspective, we would only get one spectral frame for

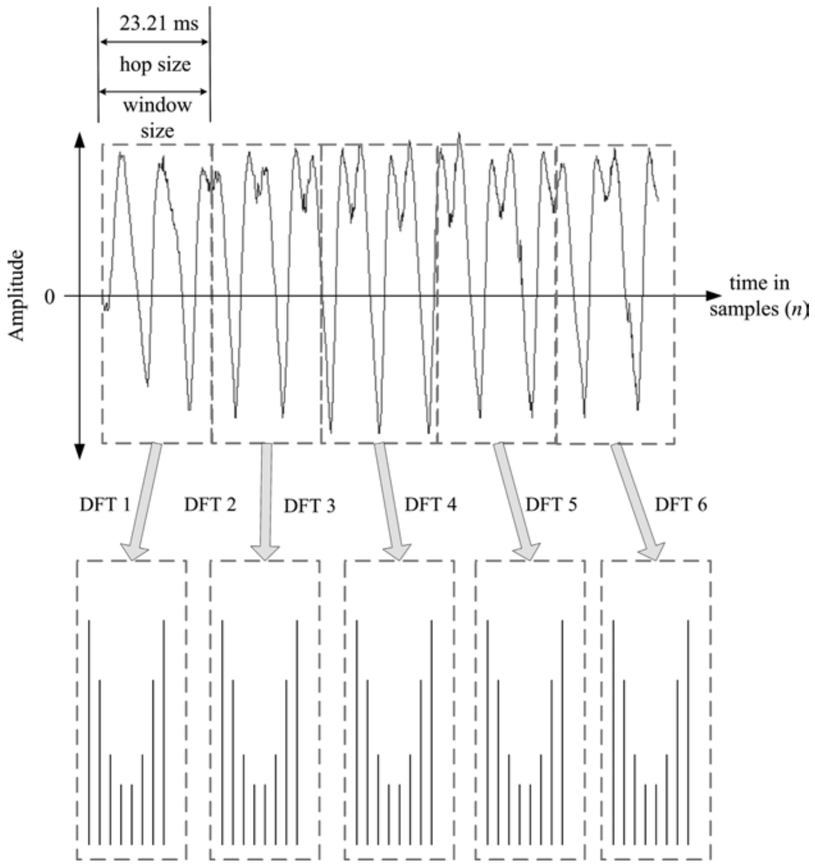


Fig. 4.9. Time and frequency domain problems with the DFT.

every 23.21 ms for the 1,024 window size, thus reducing the time resolution albeit increasing frequency resolution as seen in Fig. 4.9.

This is in fact an artifact of the DFT — increasing the frequency resolution necessarily decreases the time resolution and vice-versa. Hence, a compromise always exists between frequency resolution vs. time resolution characterized by an inverse proportionality relationship between the two and care has to be taken when deciding on the choice of the DFT size. So, what is a good DFT size? The answer is a bit difficult to arrive at without more background information as it depends on many factors such as what we want to get out of the DFT, what sort of signal we are analyzing, how much detail we need, and so forth. In short, there is no simple universal answer. However, there is an algorithm that can help us somewhat alleviate this issue regarding frequency vs. time resolution. This is the topic in the next section.

5 Short-Time Fourier Transform (STFT)

The inherent problems with frequency and time resolution in the DFT are sometimes problematic in practical applications where both high time and frequency resolution characteristics are not just desirable but necessary. So, how do we increase frequency resolution as well as time resolution at the same? One way to help in obtaining a good approximation of fine time resolution as well as frequency resolution is via the *short-time Fourier transform.* The algorithm is remarkably simple yet very effective and works mechanically similar to the overlap-and-add method discussed in Chap. 2, Sec. 7. When computing successive DFT frames for an input signal, the window increments adhere to the size of the window itself as discussed in the previous section. That is, the *hop size* for each window is equal to the window size and there is no overlap between any windows. This does not have to be case. Using the OLA method where the windows are overlapped, we utilize a hop size that is smaller than the window size, which in turn renders a greater number of frames for a given timeframe when compared to a non-overlapping window setting. In other words, we have more spectral frames for a given period of time increasing time resolution without affecting frequency resolution as the window size stays the same. The more overlap there is, the greater number of DFT frames we will obtain — improve time resolution without degradation of frequency resolution. The time resolution in seconds for a given hop size is computed via Eq. (5.1) where *hop* is the hop size in samples and f_s the sampling frequency.

$$\text{time resolution} = \frac{hop}{f_s} \tag{5.1}$$

In practice, the hop size is often defined as the percentage of overlap between windows. For example, a 50% overlap for a DFT size of 128 would result in a hop size of 64 samples. This idea of overlapping and windowing using the STFT is depicted in Fig. 5.1 with an overlap of 40% equivalent to 12.73 milliseconds. We can also further increase the overlap percentage, thus decreasing the hop size, which will as a consequence increase the number of computed spectral DFT frames for the same portion of a signal being analyzed. The interesting result the STFT method provides is that we are

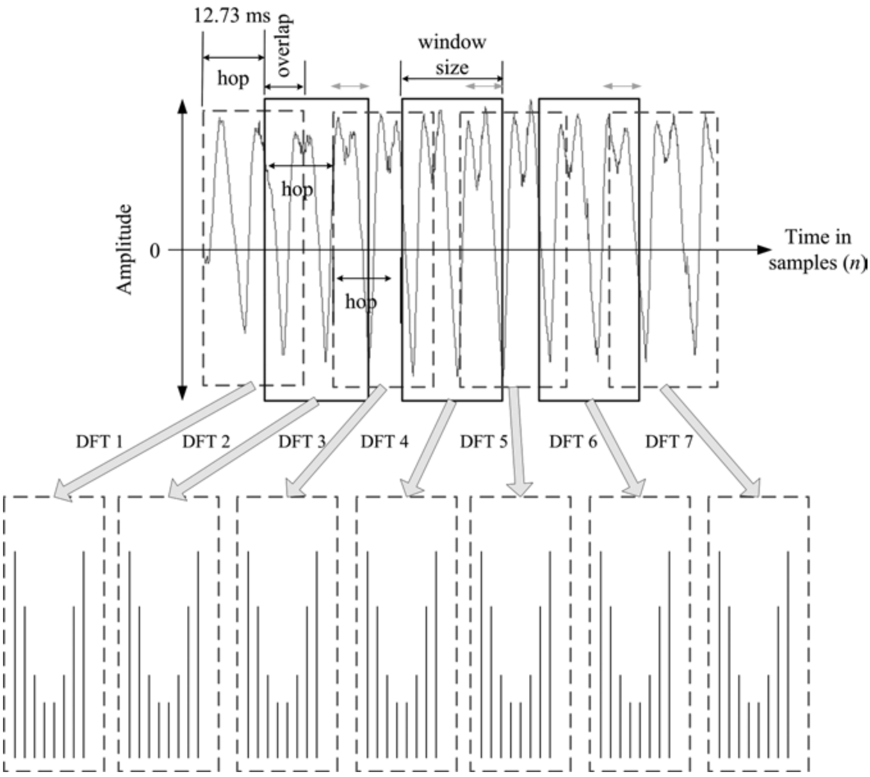


Fig. 5.1. STFT algorithm using overlap and hop size smaller than the window size.

able to acquire the best of both worlds — good time resolution and good frequency resolution. The hop size determines the time resolution while the window size determines the frequency resolution. Thus, when using the STFT for analysis, both time and frequency resolution can be substantially improved compared to using the non-overlapping windowing method alone. It is then no surprise that for most spectral analysis situations, STFT is used instead of non-overlapping DFTs.

6 Zero Padding

When using the DFT with a particular window size, it is possible to fit a smaller time-domain portion of a signal into a window that is greater in size. For example, let's say we have a time-domain signal of 54 samples but want to use a DFT window size of 64 samples giving us a frequency

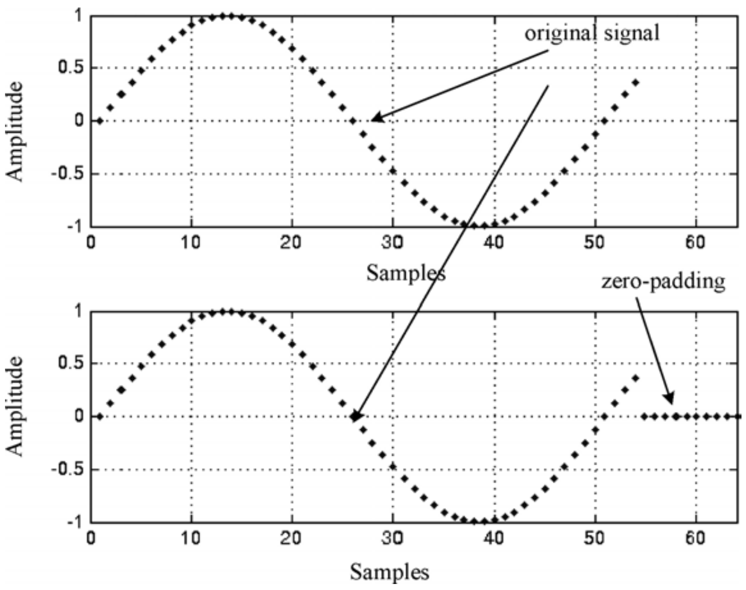


Fig. 6.1. Zero-padding in time-domain signal. Non-zero padded (top) and zero-padded (bottom).

resolution of 689.06 Hz (44100/64). But as we can see, we do not have enough samples to use a window size of 64 samples. What do we do? We use a process called *zero-padding*. What zero-padding does is simply fill the remaining holes with zero-valued samples at the tail of the sequence as shown in Fig. 6.1. In this example, we zero-pad the 54 sample signal with 10 zeroes.

The zero-padding method allows us to use higher DFT sizes offering higher frequency resolution ... kind of. The resulting spectrum turns out to be an interpolated version of the original spectral frame $X^d[\cdot]$. This does not mean that we are getting "more" information per se, as we are merely padding the signal with zeros, although in effect, we have a higher frequency resolution-based spectrum of 64 divisions rather than 54 divisions of the frequency axis. Zero-padding is very useful when using the FFT (fast Fourier transform) as we shall see in Sec. 10 (we will also see why we have been using window sizes of the form 2^N).

The top of Fig. 6.2 shows the DFT of the sine wave from Fig. 6.1 with DFT window size and signal size of 54 samples and the bottom figure shows the zero-padded version using a DFT size of 64 samples with the addition of 5 frequency samples to the tail of the spectrum (note that we are only

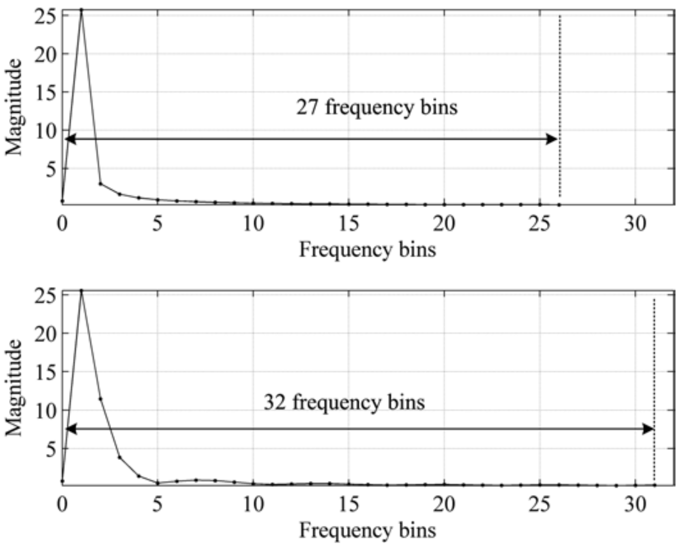


Fig. 6.2. Zero-padding effect in frequency-domain spectrum. Original X^d (top) and zero-padded X^d (bottom).

showing half of the spectrum's magnitude values). We can clearly see that the spectra look more or less the same for the exception of the zero-padded version having more frequency bins — a stretched and interpolated version of the original 54 bin spectrum. Now, you may think if zero-padding in the time-domain provides interpolation in the frequency-domain then zero-padding in the frequency-domain should result in interpolation in the time-domain. As it turns out this is indeed the case. We won't delve deeper into this topic here, but suffice it say that undesirable artifacts often show up in the interpolated time sequence after the inverse DFT and is therefore not commonly used as an interpolator.

7 Aliasing Revisited

In Chap. 1, Sec. 4.2 we introduced the concept of aliasing and showed how we could "sort of" interpret this concept in the time-domain — explain what was happening and compute the aliased sinusoids' frequencies. We also were able to see why aliasing occurred through frequency response analysis in Chap. 6, Sec. 2.1.1 and verified the mirror imaging-like characteristics of the magnitude at the Nyquist limit. Now we also have Jean Baptiste on our side to help us shed even more light on this important concept. In this section, we will tackle the concept of aliasing viewed from the frequency-domain's perspective.

In our examples up to now, we pretty much confined our attention and interest to the frequency region 0 to π which is equivalent to 0 to $f_s/2$. What happens when we go beyond π, $2\pi, 3\pi, \ldots$? This should not be too difficult to answer — as mentioned before, one of the core characteristics of the Fourier transforms is the Euler identify. That is, the Fourier transform (*FT*) is periodic with 2π so that (7.1) holds true where m is any integer and ω_x the radian frequency.

$$FT(\omega_x) = FT(\omega_x + 2\pi \cdot m) \tag{7.1}$$

Thus, if we kept on increasing or decreasing the frequency and inspected the spectrum, we would obtain something like Fig. 7.1.

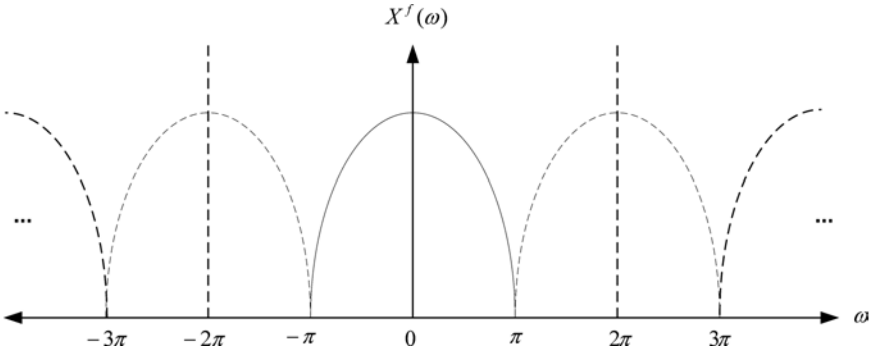


Fig. 7.1. Periodicity and mirror images in spectrum.

We recognize from Fig. 7.1 that the "original" spectrum is found between $-\pi$ to $+\pi$ (or 0 to 2π) and within this region, we are only really interested in half of its content as the other half is a mirror-like image of itself. Furthermore, if we consider the original spectrum to be from $-\pi$ to $+\pi$, we get an infinite number of replicas of the original, extending to either direction of the frequency axis.

Remember in Chap. 1, Sec. 4.2 when we swept a sine wave and witnessed frequencies that went beyond the Nyquist limit reflected back to a lower frequency value? This can now be verified using the Fourier transform as well. Let's apply the same parameters we used in Chap. 1, Sec. 4.2, namely $f_s = 100$ Hz, and frequencies increasing from 1, 5, 20, 50, 95, and 105 Hz. The Fourier transforms for each is shown in Figs. 7.2–7.4 with a window size of 128 samples.

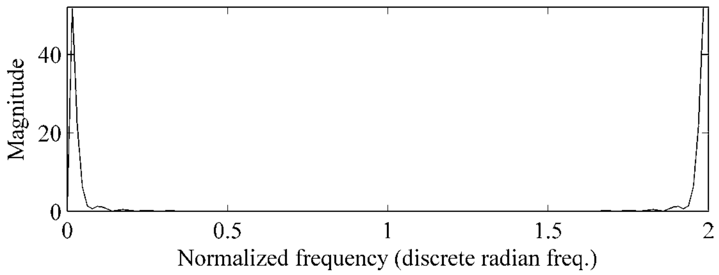


Fig. 7.2. Sine wave at 1 (top) and 5 Hz (bottom).

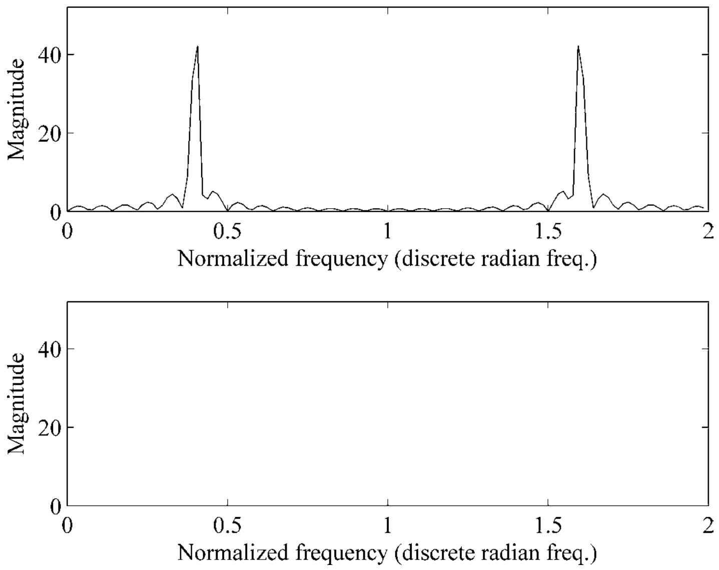


Fig. 7.3. Sine wave at 20 (top) and 50 Hz (bottom).

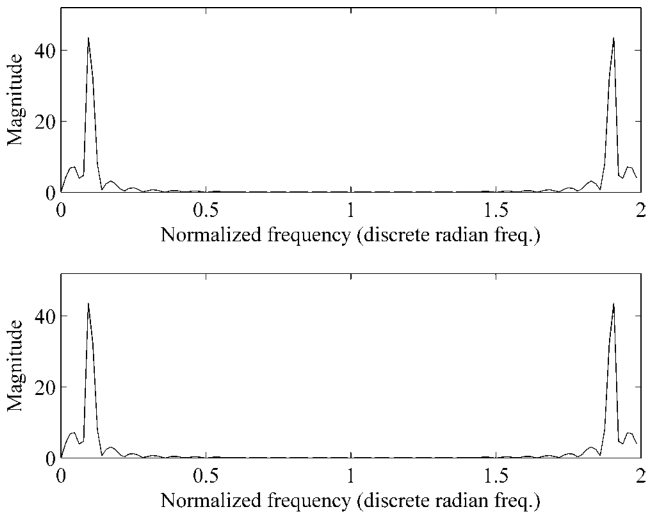


Fig. 7.4. Sine wave at 95 Hz (top) and 105 Hz (bottom).

As we expected, we can unequivocally witness this "bouncing-off" of the mirror phenomenon at the Nyquist limit as we did in Chap. 1. We again confirm that the 5 Hz, 95 Hz, and 105 Hz sine waves are indistinguishable from one another as we observed previously and that the 50 Hz sinusoid is equivalent to the 0 Hz signal. In these figures we are obviously dealing with the magnitude values and hence there is no polarity change in the amplitude values as encountered in Chap. 1. However, like before the mirror's boundary is at π ($f_s/2$) and the image is found in the region between π and 2π or $f_s/2$ and f_s. If we kept on increasing/decreasing the frequency, this yo-yo like behavior will go on in perpetuity as shown in Fig. 7.1.

8 Another Look: Down-Sampling and Up-Sampling Revisited

In this section, we will revisit what happens during down-sampling and up-sampling in the frequency-domain and get a better handle on why we need low-pass filters to fix artifacts which would otherwise cause problems.

8.1 *Down-Sampling*

As introduced in Chap. 2, Sec. 6.2, down-sampling simply refers to reducing the number of samples per second in the time-domain. To refresh our

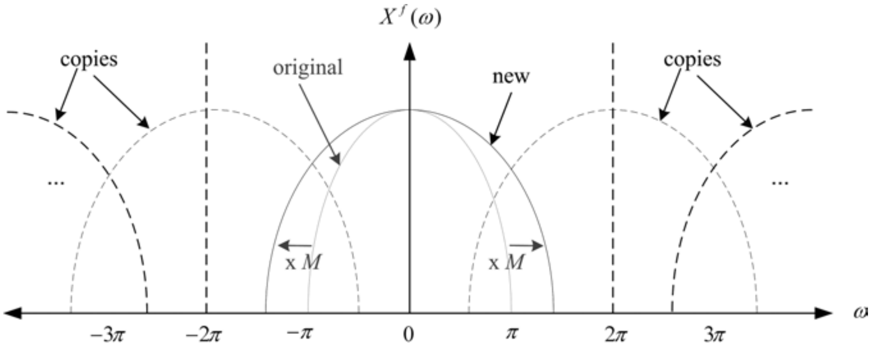


Fig. 8.1. Aliasing caused by down-sampling.

memory — if we have a sampling rate of 8 kHz for signal $x[n]$ of 2 seconds in length which is equivalent to 16,000 samples and we reduce the total number to 8,000 samples by plucking every other sample, we would have essentially down-sampled the signal by a factor of two.

Let's consider a signal's spectrum, such as the one from the previous section (Fig. 7.1) and down-sample it by M and see what happens (M is an integer number). In Fig. 8.1 we observe that down-sampling causes the original spectrum to broaden in width on the frequency axis by a factor of M. As a matter of fact, it exceeds the Nyquist limit at $\pm\pi$ and causes the images of the original spectrum to spill over and fold over to our area of interest which is from 0 to π (this actually happens for all the other images as we can clearly see in Fig. 8.1). Thus, the frequency components which were supposed to go beyond the Nyquist limit (π or $f_s/2$) are now aliased back to a lower frequency value! For a complex signal with a few hundred or thousands of sinusoidal components such as the one in our example, down-sampling without proper low-pass filtering can cause undesirable distortion. An intuitive way to understand why we get a broadening of the spectrum is to simply consider the cycle of a sine wave. That is, for lower frequencies the value for cycles per second is small — a 1 Hz sine wave will have one cycle per second and for a sampling rate of 8 kHz that would entail 8,000 samples describing one cycle. For a 2 Hz signal, equivalent to two cycles per second, this will result in 4,000 samples per cycle. In essence, down-sampling reduces the number of samples per cycle and as a consequence increases the frequency/pitch. Thus, due to this reasoning, we automatically get an increase in frequency during down-sampling. This is expected as we observed and asserted in Chap. 2 — decrease in the number of samples increases the pitch (for a pitched signal). If a sine wave is down-sampled by two the result would be an increase in frequency by an octave.

The aliased versions of the original can be expressed as follows (Porat 1997):

$$X^f{}_{(\downarrow M)}(\theta) = \frac{1}{M} \sum_{m=0}^{M-1} X^f \left(\frac{\theta - 2\pi \cdot m}{M} \right) \tag{8.1}$$

where M is the down-sampling factor, m an integer referring to the copies of the original, and θ the radian angular frequency per sample. Equation (8.1) may seem somewhat complicated at first but if one looks at it for a minute or so, the right hand side may be less convoluted than meets the eye — the right-hand side merely adds via super-positioning, M-shifted and frequency scaled (divisor M) versions of the original Fourier transform $X^f(\theta)$. This reflects the resulting net down-sampled aliased spectrum caused by the copies of the original spectrum (between $-\pi$ to $+\pi$): spectral overspill into neighboring frequency regions marked by integer multiples of the Nyquist limit. The overall $1/M$ scalar outside the summation is there to conserve the overall energy as we would expect the total energy to remain the same. The net aliased spectrum of Fig. 8.1 would look something like Fig. 8.2 for $M = 2$. Table 8.1 shows where the original spectrum's magnitude values at frequency points $\pm\pi$ are scaled to during down-sampling for $m = 0$ to 1.

In order to avoid aliasing during down-sampling we need to low-pass filter the signal *before* down-sampling or digitization. If we do it after the sampling process, we will already have inadvertently caused aliasing to occur (if sinusoidal components higher than the Nyquist frequency exist). Since down-sampling by a factor of M increases the bandwidth by M,

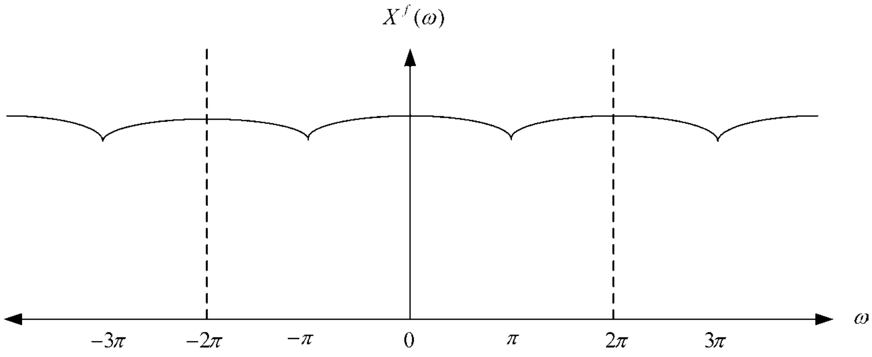


Fig. 8.2. Resulting net aliased spectrum for $M = 2$.

Table 8.1. Down-sampled locations of original $\theta = \pm\pi$.

	Frequency (θ)	Fourier transform $X^f(\theta)$	Magnitude
Original	$\pm\pi$	$X^f(\theta)$	0
$m = 0$	$\pm 2\pi$	$X^f\left(\frac{\theta}{2}\right)$	0
$m = 1$	$4\pi, \mathbf{0}$	$X^f\left(\frac{\theta-2\pi}{2}\right)$	0

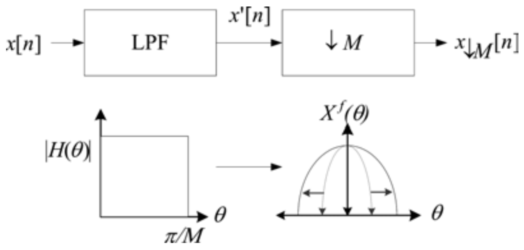


Fig. 8.3. LPF followed by down-sampling.

we would need to use a low-pass filter at π/M or lower. The reason we would consider using a lower cut-off frequency at all is due to the fact that real filters cannot be brick-wall filters as discussed in Chap. 7. Because each filter has some sort of transition band, we will need to take into account a small amount of safety when designing any filter. To avoid aliasing, we would hence need to follow the procedure in Fig. 8.3. This is sometimes referred to as decimation opposed to down-sampling, the latter referring to deleting samples without low-pass filtering beforehand whereas decimation referring to low-pass filtering followed by down-sampling.

In a way, the reason we get aliasing when down-sampling a signal should intuitively make sense if regarded from a digitization of analog signals point of view. Think of the process of sampling an analog, continuous-time signal. We know that in such a situation, aliasing will occur when a frequency component higher than the Nyquist exists in the analogue signal. In other words, in the continuous-time sampling case, what we are essentially performing is "down-sampling" an analog signal — the analog signal has infinite time/frequency-resolution whereas the digital version a finite one (larger to smaller bandwidth). Thus, down-sampling after sampling is just another form of re-sampling an already sampled signal (discrete-time signal). If during down-sampling the highest frequency of the signal is smaller than π/M we would not need to do any low-pass filtering and as there would be no aliasing concerns.

8.2 *Up-Sampling*

When we up-sample a signal, we increase the number of net samples and add (interleave) zeros to the original signal. For example, if we up-sample by a factor of two, we add a zero at every other sample. During up-sampling (or *expansion* as it is sometimes called when viewed from the time-domain), the spectrum changes from our original spectrum in Fig. 7.1 to the new spectrum shown in Fig. 8.4.

As we might have already expected, during up-sampling, the spectrum shrinks to a smaller bandwidth opposed to a broadening one as was the case with down-sampling. The up-sampling process also produces unwanted replicas of the shrunken spectrum as observed in Fig. 8.4. As with decimation, this undesired artifact caused by the addition of zeros can be eliminated using a low-pass filter *after* up-sampling with cutoff frequency π/L. Thus, for the above example, we follow the procedure depicted in Fig. 8.5.

It should not be that much of a mystery to understand how the low-pass filter eliminates the problem caused by inserting zeros into a

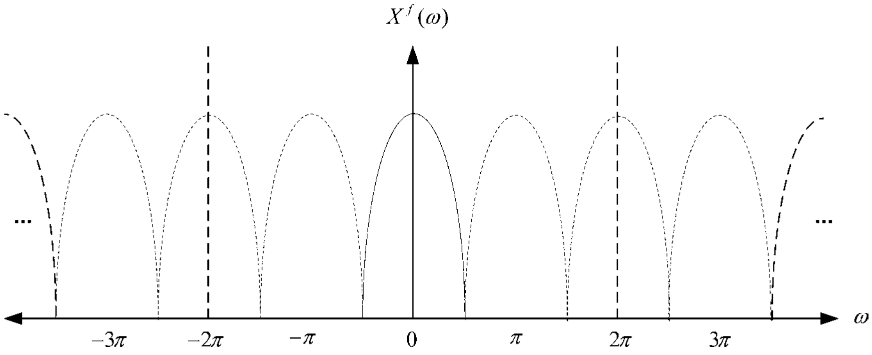


Fig. 8.4. Up-sampling by factor $L = 2$.

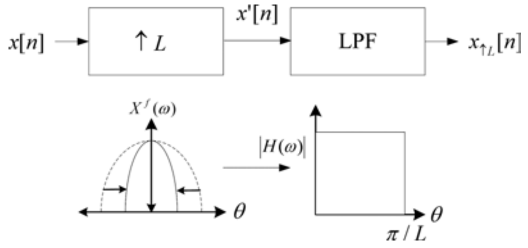


Fig. 8.5. Up-sampling and low-pass filtering at π/L.

signal — low-pass filters make signals smoother. Thus, a zero-interleaved signal, resembling a "potholed" sequence of numbers, will be smoothed out in the time-domain which has the effect of eliminating the replicas in the frequency-domain. Figure 8.6 shows the up-sampling process where the top plot depicts a sine wave sampled at f_s = 40 Hz with frequency = 1 Hz,

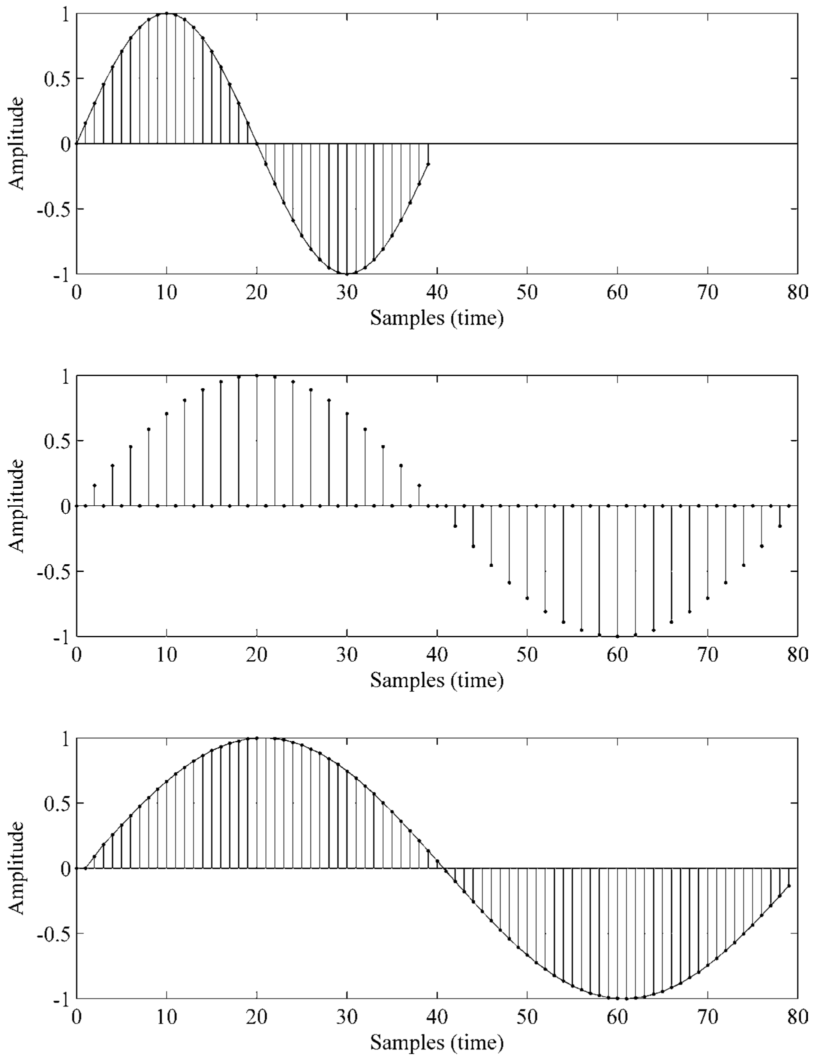


Fig. 8.6. Original signal with $f_s = 40$ Hz, $f = 1$ Hz sine wave (top), zero interleaved and up-sampled at $L = 2$ (middle), low-pass filtering of up-sampled signal (bottom).

the middle plot the up-sampled version with 0 interleaving, $L = 2$, and the bottom plot the low-pass filtered result completing the up-sampling process. In this example, we used a 2nd order Butterworth low-pass filter. A particular quality of up-sampling that we can vividly hear in audio signals is the decrease in bandwidth as we can predict from the Fourier transform results — the spectrum contracts towards the DC making the signal sound duller and less bright due to the reduction of high frequency components.

9 Windowing Revisited: A View from the Frequency-Domain Side

When excerpting a portion of a signal for analysis, synthesis, or for whatever purpose, we accomplish this excerpting through windowing. We have already used windows in RMS computation, the OLA algorithm, Fourier transform, STFT, and many other situations and are undoubtedly comfortable with the concept. As we have seen in Chap. 2, Sec. 4, different window types have different time-domain characteristics verifiable just by plotting the windows in the time-domain. As we may remember, the rectangular window is the "simplest" one having the sharpest edges, whereas the Hann window for example, has smooth flanking boundaries helping with fading-in and fading-out the excerpted signal in the time-domain.

The shape of a particular window has specific effects in the frequency-domain and each type of window causes accompanying characteristic ripples. Thus, the objective in choosing the right window is closely related to minimizing those ripples and other window parameters as they add distortion to the DFT outputs — in order to use DFTs we need to use a window of some sort. More specifically, the main parameters that determine the behavior of windows are the *main lobes*, *side lobes*, and *roll-off* characteristics. To help us better understand these three important parameters pertinent to a window's choice, let's start with the 32-sample rectangular window.

The three typical characteristics that are the result of choice of a particular window type are seen for the rectangular window in Fig. 9.1. The main lobe basically determines how much resolution is given to a signal in terms of separating closely distanced peaks (peaks close in frequency such as two sinusoidal components). Hence, windows with main-lobes of small widths contribute to better frequency resolution and help

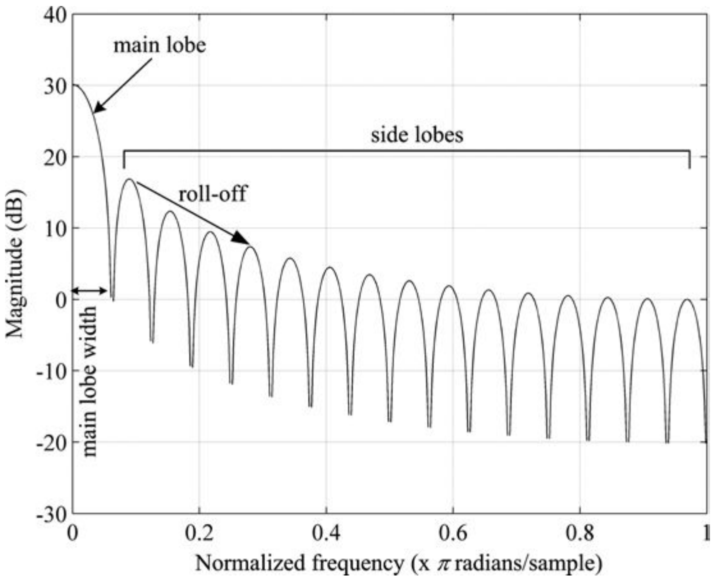


Fig. 9.1. Rectangular window in frequency-domain.

in differentiating and segregating frequency components that are close to each other in frequency. All other lobes are referred to as side lobes. Side lobe characteristics are also a function of the window type, coloring and distorting the input signal, mainly via the strength of the side lobe magnitude values. The side lobes contribute to a certain amount of energy leakage into adjacent bins with respect to the main lobe. This may cause less salient peaks in adjacent areas to be buried, if side lobes are too high in energy. The roll-off presents the decreasing magnitude characteristics of the side lobes. Thus, it is usually desirable to have a narrow main lobe and low magnitude side lobe configuration for any given window.

To get a better idea what all this means, let's use two sine tones with amplitude of 1.0 and a rectangular window of 32 samples and zero-pad it to 1,024 samples. As shown in Fig. 9.2, the window will be multiplied with each sinusoid and will pass only those samples that fall in the area bounded by sample number 1 and 32. We have used a smaller "actual" window size (where amplitude is unity) to help visualize the effect of the main lobes and side lobe characteristics on the two sinusoids.

The next two figures (9.3, 9.4) show the two rectangular windowed sine waves at 500 Hz and 1 kHz. We can clearly locate and identify the main lobes and side lobes in each plot — the distinctive wide main lobe stands

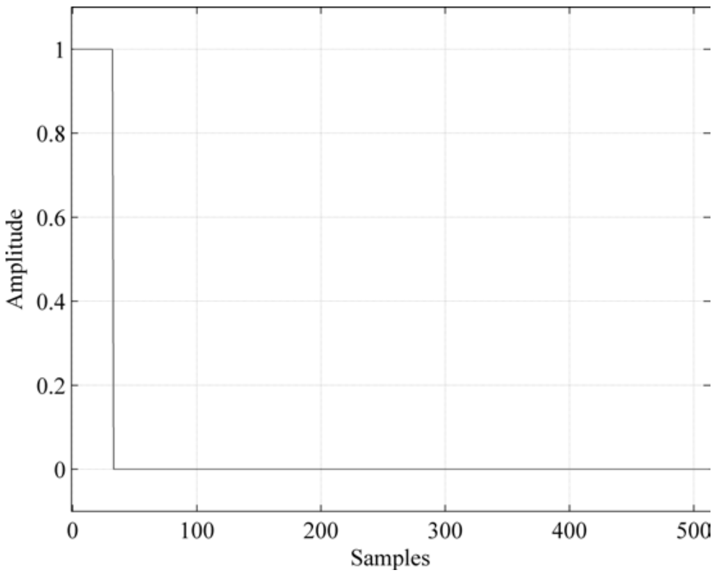


Fig. 9.2. Rectangular window size 32 samples zero padded to 1,024 samples.

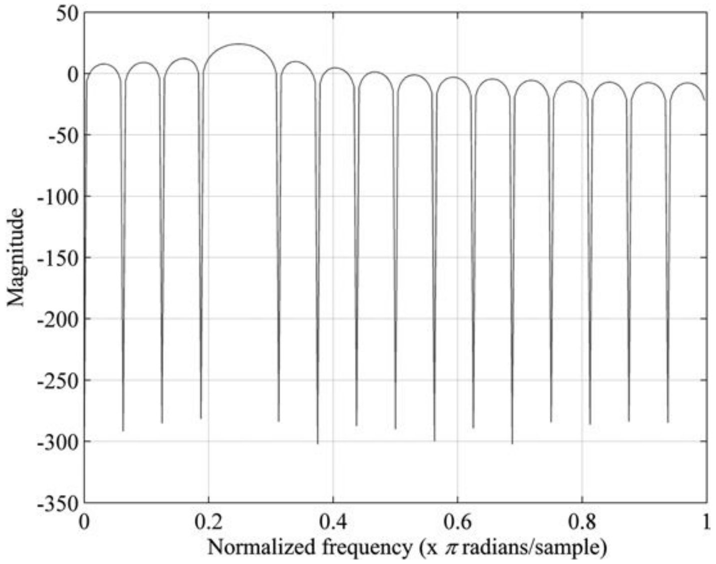


Fig. 9.3. DFT of sine tone at 500 Hz using rectangular window.

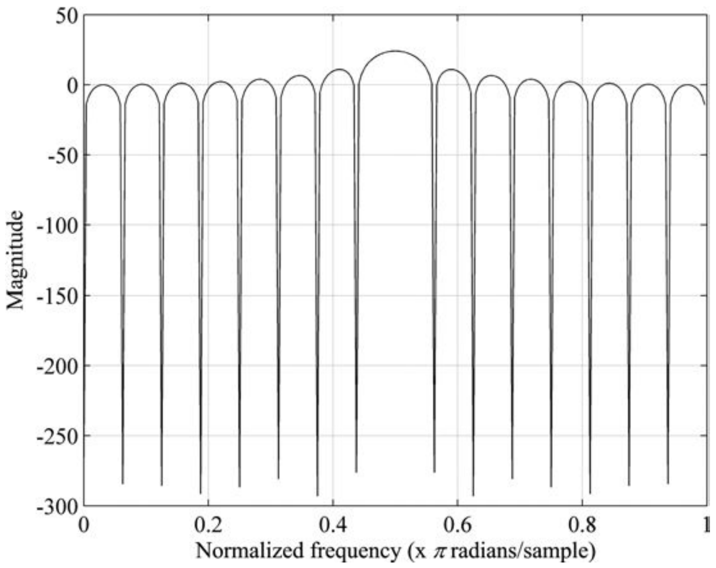


Fig. 9.4. DFT of sine tone at 1,000 Hz using rectangular window.

out and wraps around the peak representing the sine tone's frequency. The side lobes taper off on the flanking side of the peak in both figures as expected. Now, what happens when we have the two sine tones at 500 Hz and 1 kHz mixed together? The result of the complex tone is shown in Fig. 9.5. Although we can differentiate the two peaks corresponding to the 500 and 1 kHz sine tones, we also see that additional distortion takes place due to the net cumulative leakage in the side lobes contributed by the two windowed sine waves.

Now, let's use a smaller amplitude value for the 500 Hz sine wave and see what happens. In Fig. 9.6, we employed only 10% of the original amplitude value of the 500 Hz tone and left the 1 kHz tone unchanged. As we can see, the DFT plot becomes more ambiguous than before and it actually becomes more difficult to see what is happening — the peak of the 500 Hz sine wave is not as peaky anymore and almost disappears into jungle of side lobes. Musical signals and audio signals in general are far more complex than what we are seeing here and have many more salient peaks and harmonics that constitute a complex audio signal. This makes the identification of peaks that much more difficult in part due to the distortion effects of windowing. Thus, in general as stated above, it is highly desirable to keep the side lobe magnitudes as low as possible, not only to alleviate

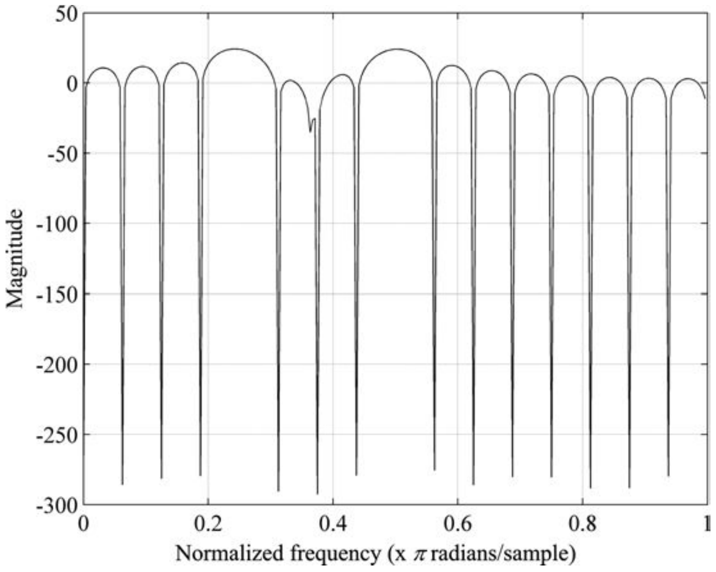


Fig. 9.5. DFT of the two 500 Hz and 1 kHz sine tones combined.

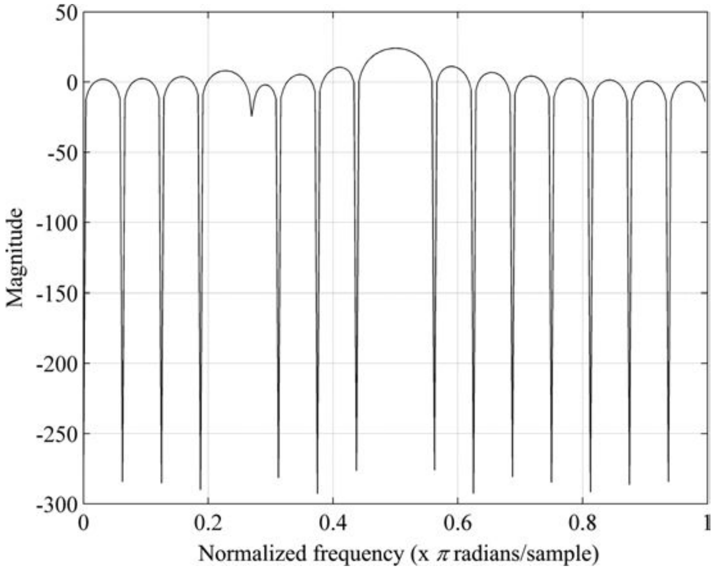


Fig. 9.6. 500 Hz tone at 10% of original amplitude, 1 kHz sine tone unchanged.

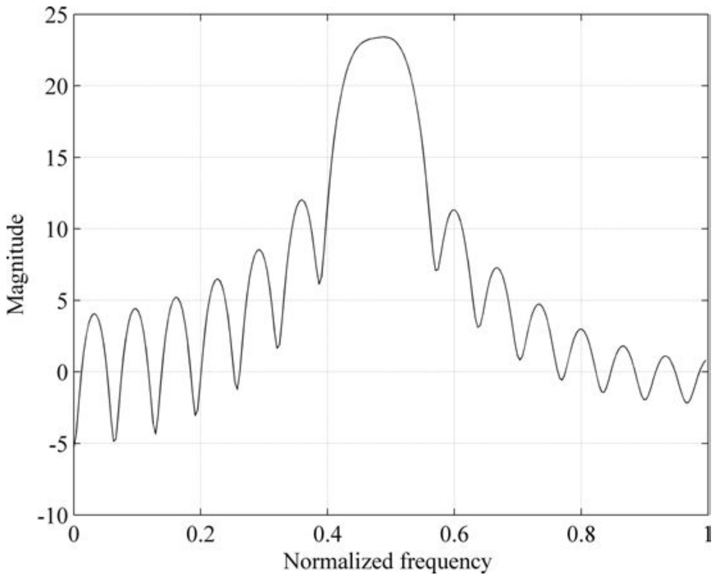


Fig. 9.7. 500 Hz moved up to 920 Hz and 1 kHz tone unchanged.

energy leakage and spectral smearing, but also to make the main lobe itself more pronounced as a consequence.

As a final example to help illustrate the importance of the main lobe and side lobes, let us move the two sine tones closer together in frequency by shifting the 500 Hz signal to 920 Hz while keeping the amplitude levels the same at 1.0 as before. As we can observe in Fig. 9.7, although we would expect to see two peaks corresponding to each sinusoid, we only see one big and bulky one — sort of looks like the mountain from *Encounters of the Third Kind*. In any case, the 920 Hz component (and the 1 kHz component) seems to have been lost in the process in part due to the main lobe characteristics. The important point here is that we have to be careful or at least know what the main lobe and side lobe characteristics will be when looking at the spectrum and analyze the data accordingly. This will allow us to make informative assessments and guesses of what we are analyzing.

A simple approach for increased frequency resolution is of course to use a larger window size (DFT size) as discussed in Sec. 4. This, however, does not necessarily solve the side lobe problem for all cases. Hence, depending on the signal you wish to analyze or manipulate, a wide array of windows exist which have distinct characteristics and perhaps just the right features for your application. Some representative window types and their main characteristics are discussed in the following sub-sections.

9.1 *Rectangular Window*

Rectangular windows are the most common types of windows where the window itself is a collection of ones only — unity gain applied to the entire window as seen in Fig. 9.8. The main lobe width is $4\pi/N$ (N is the number of samples), first side-lobe attenuation is -13.3 dB, and has a roll-off of 20 dB/decade characteristic. There is also a substantial amount of leakage factor which contributes to frequency smearing due to the rather high dB side-lobe levels. At the same time, it is relatively well suited for transient-based signals — signals that change quickly.

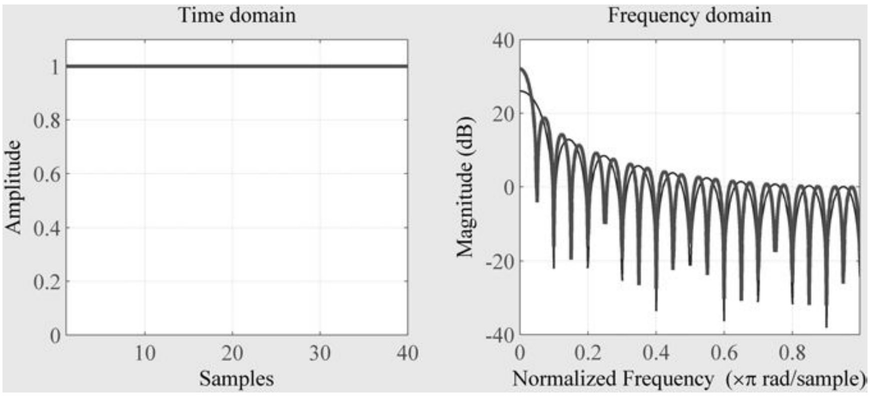


Fig. 9.8. Rectangular window.

9.2 *Hann Window*

The Hann window also sometimes referred to as the Hanning window, achieves a side-lobe reduction by superposition. It turns out that the implementation of the window is achieved via three *Dirichlet kernels* which are shifted and added together resulting in partial cancellation of the side-lobes (see Sec. 9.6 on definition of the Dirichlet kernel). The resulting characteristics of the Hann window which is sometimes called the cosine window, has a first side-lobe level at -32 dB, main lobe width of $8\pi/N$, and a 60 dB/decade roll-off rate. We can also observe a considerably sharp roll-off rate and a wider main-lobe width with better frequency leaking factor than the rectangular window due to the high roll-off.

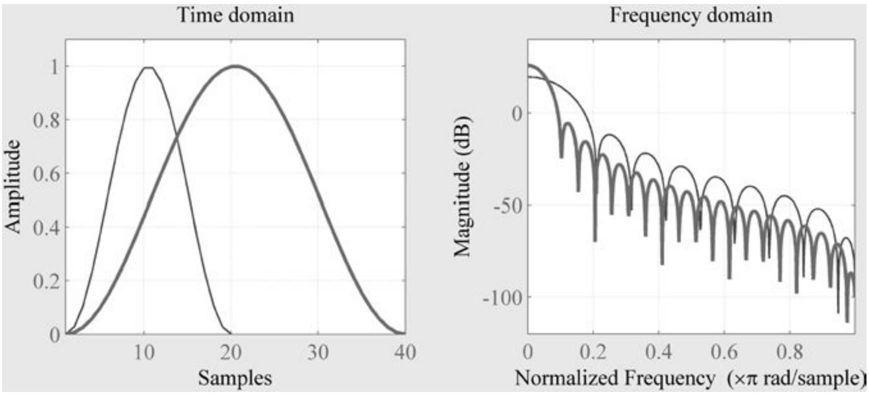


Fig. 9.9. Hann window.

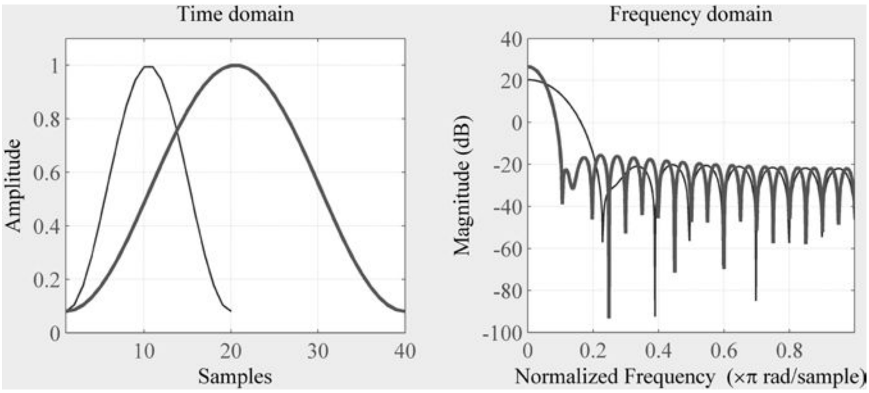


Fig. 9.10. Hamming window.

9.3 *Hamming Window*

The Hamming window is similar to the Hann window with some modifications in weighting the Dirichlet kernels. The time and frequency-domain plots of the windows are shown in Fig. 9.10. The main-lobe is $8\pi/N$ with -43 dB side-lobes and roll-off of 20 dB/decade. One significant feature of the Hamming window is the non-zero values at both the head and tail of the window and therefore is also sometimes referred to as the half-raised cosine window. Due to the sharper edges, it somewhat behaves more like a rectangular window than does the Hann window. It therefore has a narrower main lobe and slightly better transient features. Another interesting observation about the Hamming window is that the first side-lobe is actually smaller than the 2nd side-lobe. Hamming windows are widely used in audio-based spectral analysis and synthesis applications.

9.4 *Blackman Window*

The Blackmann window has a -58 dB side-lobe, a slightly large main-lobe width of $12\pi/N$, and roll-off of a steep 60 dB/decade rate. N is again the window length in terms of samples as shown in Fig. 9.11. Although the main-lobe is on the wider side of the main-lobe comparison chart, contrasted to other windows, the side lobes fall on the lower end of the window spectrum.

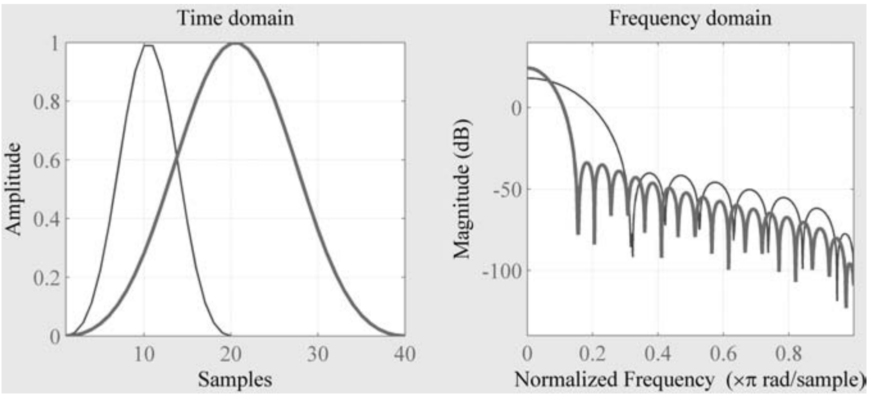


Fig. 9.11. Blackman window.

9.5 *Chebychev and Kaiser Windows*

Some windows such as the Chebychev and Kaiser windows have more direct parametric control over the shape of the windows. For example, a side-lobe attenuation parameter R is used in the Chebychev window controlling the side-lobe attenuation with respect to the main-lobe magnitude as shown in Fig. 9.12 and 9.13 for different window lengths.

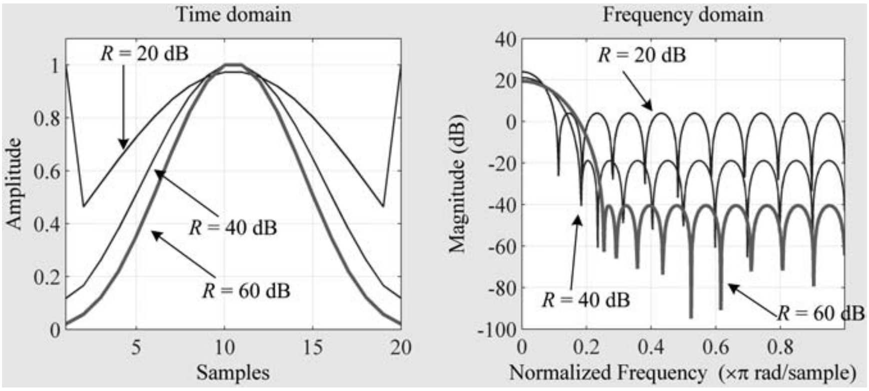


Fig. 9.12. Chebychev window, $N = 20$.

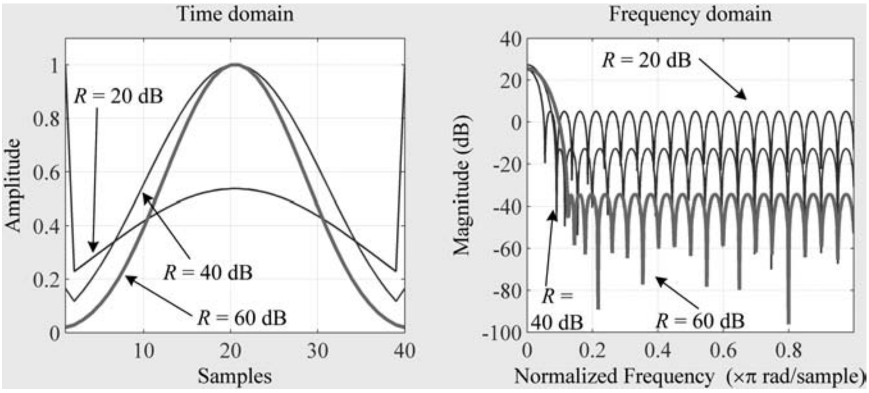


Fig. 9.13. Chebychev window, $N = 40$.

For the Kaiser window, a coefficient commonly denoted as β determines the side-lobe attenuation characteristics as seen in Fig. 9.14 and 9.15 for window lengths $N = 20$ and $N = 40$.

As we have seen in this section, various window types exhibit specific characteristics, thus making the choice of the window an important one — there is a suitable window type for a given application and signal type.

9.6 *Not Just More Windowing Stuff*

Hopefully by this time, we have a pretty good grasp on the topic of windowing and its various characteristics, especially concerning issues

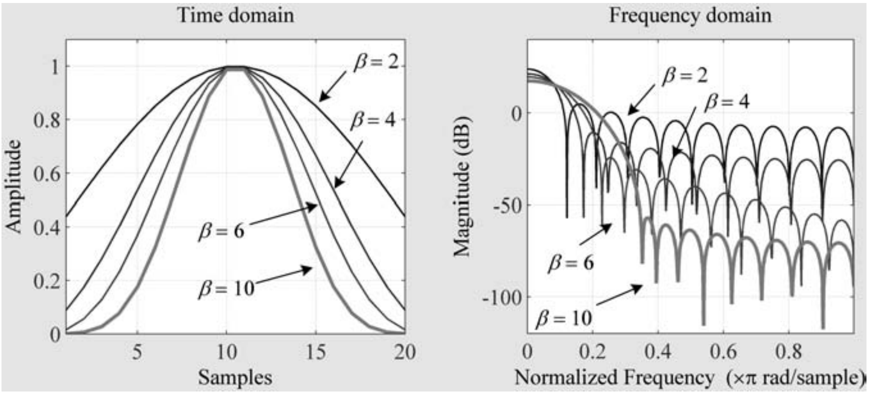


Fig. 9.14. Kaiser window $N = 20$.

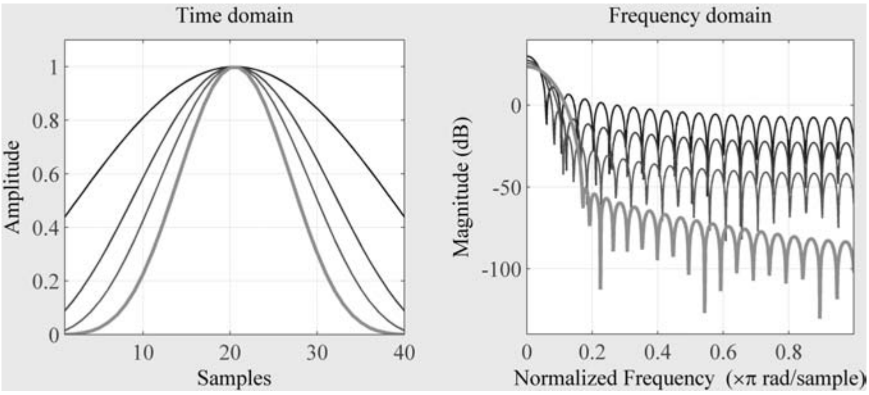


Fig. 9.15. Kaiser window $N = 40$.

pertinent to various types of distortions we should expect in the spectrum when using a particular window. Before we leave this topic, let us take a more detailed and final look at the rectangular window and its relationship to the *sinc* function which gives a window its character in the frequency-domain. Consider the rectangular window $w[n]$ as follows:

$$w[n] = \begin{cases} 1, & 0 \leq n \leq N-1 \\ 0, & \text{otherwise} \end{cases} \tag{9.1}$$

The discrete time Fourier transform (DTFT) for $x[n]$ will thus be as seen in Eq. (9.2) where θ is the digital angular frequency in radians per sample as usual.

$$X^f(\theta) = \sum_{n=0}^{N-1} w[n] \cdot x[n] \cdot e^{-j\theta n} \tag{9.2}$$

Let's for now assume that $x[n] = 1$ for all values of n and since the window is unity between 0 and $N-1$, we can rewrite (9.2) and simplify it using the geometric series:

$$\begin{aligned} X^f(\theta) &= \sum_{n=0}^{N-1} 1 \cdot 1 \cdot e^{-j\theta n} \\ &= \sum_{n=0}^{N-1} (e^{-j\theta})^n \\ &= \frac{1 - e^{-j\theta N}}{1 - e^{-j\theta}} \end{aligned} \tag{9.3}$$

Now, if reconfigure the above result so that we can express the numerator and denominator in terms of sines we have:

$$\begin{aligned} X^f(\theta) &= \frac{1 - e^{-j\theta N}}{1 - e^{-j\theta}} \\ &= \frac{\left(e^{j\theta N/2} - e^{-j\theta N/2}\right) \cdot e^{-j\theta N/2}}{\left(e^{j\theta/2} - e^{-j\theta/2}\right) \cdot e^{-j\theta/2}} \\ &= \frac{\left(e^{j\theta N/2} - e^{-j\theta N/2}\right)}{\left(e^{j\theta/2} - e^{-j\theta/2}\right)} \cdot e^{-j\theta(N-1)/2} \\ &= \frac{\left(\frac{e^{j\theta N/2} - e^{-j\theta N/2}}{2j}\right)}{\left(\frac{e^{j\theta/2} - e^{-j\theta/2}}{2j}\right)} \cdot e^{-j\theta(N-1)/2} \\ &= \frac{\sin(\theta \cdot N/2)}{\sin(\theta/2)} \cdot e^{-j\theta(N-1)/2} \end{aligned} \tag{9.4}$$

The function

$$D(\theta, N) = \frac{\sin(\theta \cdot N/2)}{\sin(\theta/2)} \tag{9.5}$$

is referred to as the *Dirichlet* kernel. The magnitude plot is show in Fig. 9.16 for window size $N = 20$ and Fig. 9.17 where $N = 40$ for $\theta = 0$ to 2π.

As our range of frequency interest is from 0 to π, we are not concerned with data that goes beyond π or $f_s/2$. However, for now we note that (9.5)

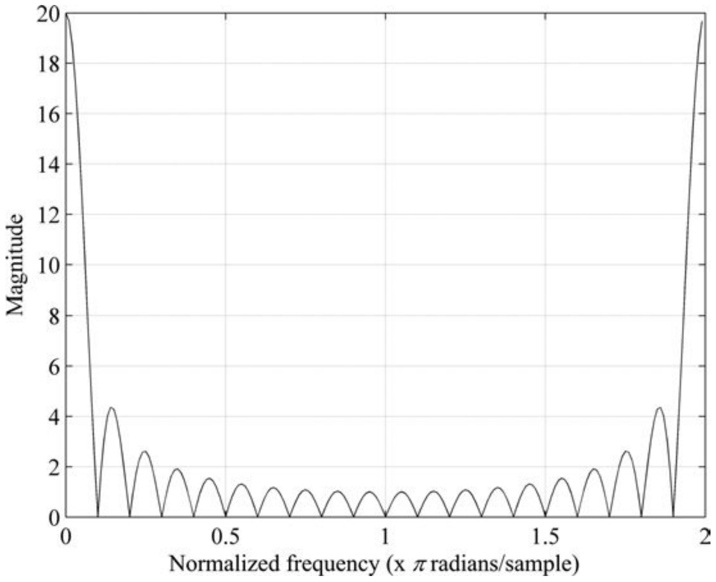


Fig. 9.16. Dirichlet kernel for $N = 20$ from 0 to 2π.

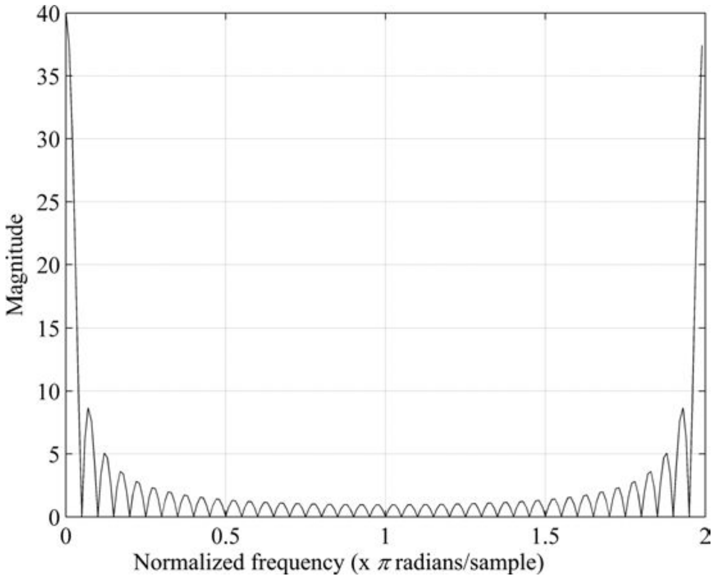


Fig. 9.17. Dirichlet kernel for $N = 40$ from 0 to 2π.

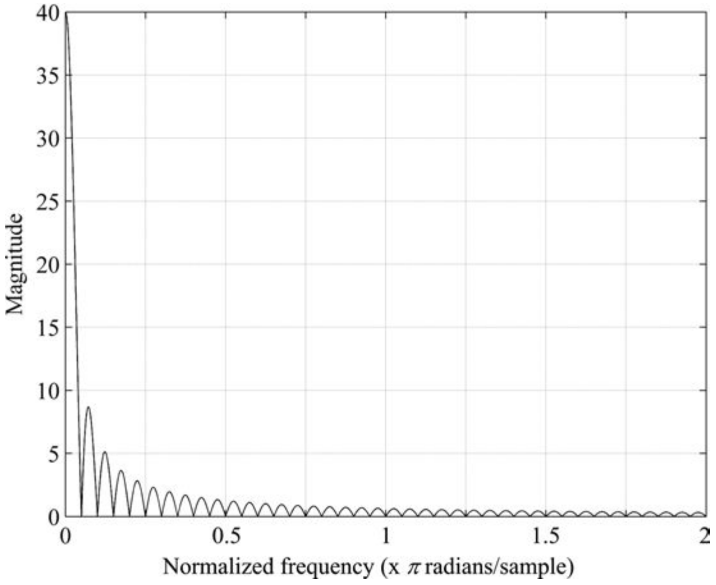


Fig. 9.18. Sinc(x) function.

looks very similar to a *sinc* function or more specifically, like a *normalized sinc function* defined as Eq. (9.6) and shown in Fig. 9.18.

$$\operatorname{sinc}(x) = \frac{\sin(\pi \cdot x)}{\pi \cdot x} \tag{9.6}$$

We can see in Fig. 9.17 that the sinc(x) function closely resembles the Dirichlet kernel up until the center of the frequency axis. However, upon closer inspection we realize that the difference between the two increases as we move towards π. Furthermore, after π, the Dirchlet seems to form a mirror image of itself whereas the sinc(x) function decays further as shown in Fig. 9.19.

One very important characteristic regarding the Dirichlet kernel pertinent to the topic of DSP is the location of the zeros-crossings in the frequency-domain. That is, the first zero occurs in the frequency axis pertinent to a single full cycle of a sinusoid within the rectangular window in the time-domain. The longer the window is, the lower (frequency bin) the first zero will turn out to be. In our above example, we have a 40 sample rectangular window. For $N = 40$, we would get the first zero at the frequency location corresponding to a sinusoid that makes one complete cycle for that window size. Thus, the larger the window size N, the lower the frequency of the first zero will become (remember that time

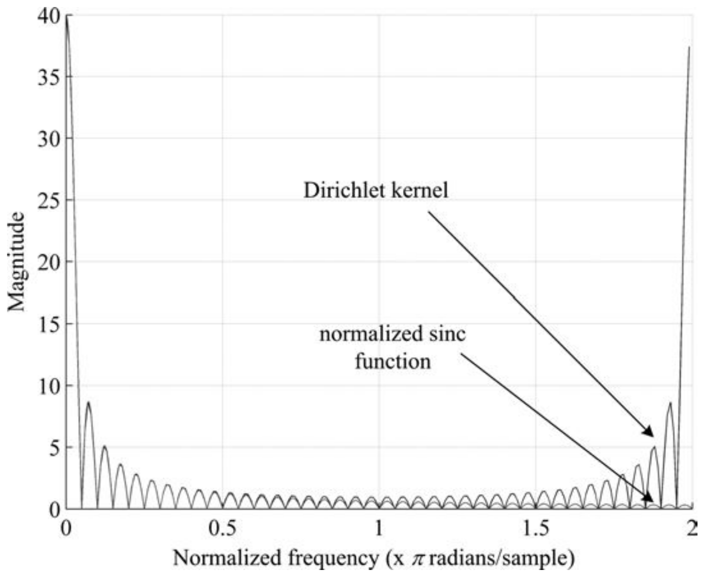


Fig. 9.19. Comparison between normalized sinc function and Dirichlet kernel.

and frequency have reciprocal relationship $f = 1/t$). This explains why we get a narrower main lobe when using a larger window size. The subsequent zero locations occur at harmonics of the first zero: 2nd zero occurring at the frequency pertinent to two full cycles in the rectangular window, 3rd zero occurring at the frequency equivalent to 3 full cycles within the rectangular window . . . etc.

We also saw at the start of this section that the main-lobes form around a sinusoid's frequency and side-lobes flank each sinusoid. This sort of behavior can now be explained by considering the following example where the input is a complex sinusoid $x[n]$ as defined below.

$$x[n] = A \cdot e^{j\omega_x n} \tag{9.7}$$

If we compute the DTFT $X^f(\theta)$ with window $w[n]$ (defined over $n = 0 \ldots N-1$) we get the following:

$$\begin{aligned} X^f(\theta) &= \sum_{n=0}^{N-1} w[n] \cdot x[n] \cdot e^{-j\theta n} \\ &= \sum_{n=0}^{N-1} w[n] \cdot A \cdot e^{j\theta_x n} \cdot e^{-j\theta n} \end{aligned}$$

$$= A \cdot \sum_{n=0}^{N-1} w[n] \cdot e^{-j(\theta - \theta_x)\cdot n}$$

$$= A \cdot X_w^f(\theta - \theta_x) \tag{9.8}$$

The result in (9.8) is quite interesting as it says that the complex sinusoid $x[n]$ windowed by $w[n]$ results in a shifted version of the window's DTFT (X_w^f) by the sinusoid's frequency θ_x. Now, this explains why the main lobe is centered at a sinusoid's frequency when windowed as it gets shifted by θ_x, the input frequency as we have seen at the beginning of this section.

10 The Fast Fourier Transform (FFT)

Musicians nowadays use the Fourier transform in customized algorithms to alter, modify, analyze, and synthesize sound in real-time during performance settings or in the composition studio/lab/home. Needless to say, the ubiquity of the DFT has been made possible by the increasing speed of processors (and the fall in prices of personal computers) which has seen remarkable progress to say the least. Back in the day, a room full of hardware could only do a fraction of what your home computer can do today and with even more power with multi-core processors, it is difficult to imagine what we will have in 10–20 years from now. In any case, an intriguing reason that makes it possible for us to use the DFT in a digital system with little to no wait-time is due to a breakthrough algorithm by Cooley and Tukey in 1965 (Cooley and Tukey 1965). This invention is called the *fast Fourier transform* commonly referred to as the FFT. I will not spend many cycles on this topic as it is well documented on the Internet and other introductory DSP books and will only discuss the basic ideas and concepts behind the algorithm. In a nutshell, what Cooley and Tukey discovered was that when the DFT of length N is a non-prime number (composite), a divide-and-conquer method would allow for faster computation of the DFT. The overall DFT operation could essentially be decomposed into a number of shorter sub-DFTs where the collection of shorter DFTs together would require less computation, compared to directly using Eq. (4.10). What's more is that when the DFT size is constrained to the power of two, the order of operations would drastically be reduced from N^2 to $N \cdot \log_2 N$ computations, the former being the computation needed for directly computing the DFT. This method of breaking the Fourier transform into two smaller sub-transforms of $N/2$ at each step is known as *radix-2* (also potentially a cool band name) and is probably one of the most widely used methods for the implementation of the FFT. You will remember in previous DFT examples, we used window sizes of power of two and this is the reason why ....

In computing the FFT, a lot of clever rearranging takes place so as to reuse computed results of one sub-transform and apply the same results to another sub-section — sort of like recycling. One interesting feature is the so-called *butterfly* configuration — this is not a special Tae Kwon Do technique but actually an algorithm. The reason for this name should be evident when looking at Fig. 10.1 which shows a 3-stage, 8-point FFT computation diagram with a bunch of butterfly-like looking configurations at each stage. We also notice a lot of recycling in various sections of the diagram where nothing is wasted in the process ... "tight" as a good friend of mine would say.

In Fig. 10.1, we used the W_N notation which makes things a lot neater. You will remember from (4.13) that W_N is the exponential term in the

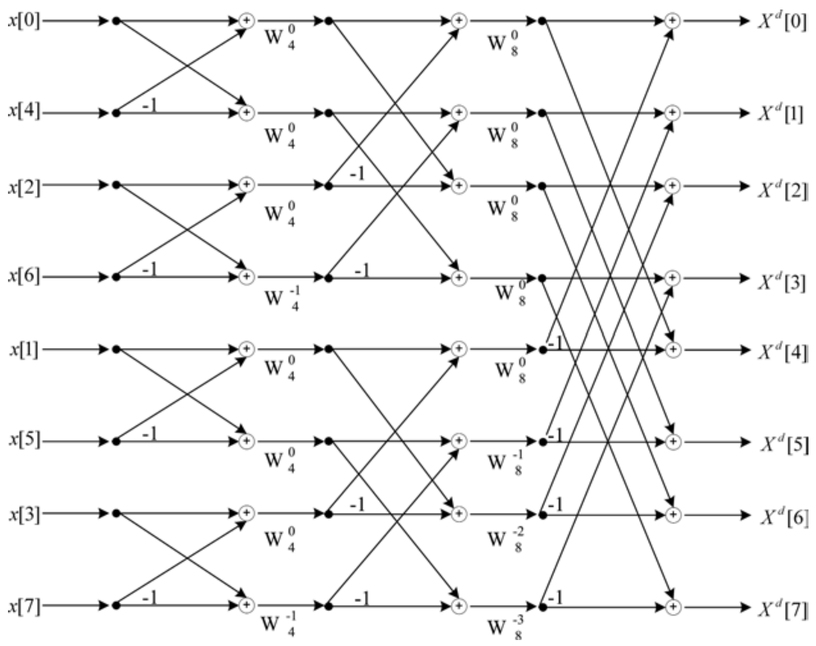


Fig. 10.1. A 3-stage, 8-point DFT computation and the butterfly.

DFT summation as shown below.

$$W_N = e^{j2\pi/N} \tag{10.1}$$

This exponential term will adhere to the same rules of multiplication and raising it to the power of M will result in (10.2) as expected.

$$W_N^M = (e^{j2\pi/N})^M = e^{j2\pi M/N} \tag{10.2}$$

11 Convolution (also) Revisited

In Chap. 5, Sec. 5, we introduced convolution and its usefulness is audio applications as defined below where $x[n]$ is the input signal and $h[n-m]$ the shifted impulse response.

$$y[n] = x[n] * h[n] = \sum_{m=-\infty}^{m=+\infty} x[m] \cdot h[n-m] \tag{11.1}$$

There are some interesting characteristics regarding convolution including its commutative property as the input can be regarded as the impulse and vice-versa yielding the same results. However, another very interesting characteristic is that convolution in the time-domain is equivalent to multiplication in the frequency-domain. This is shown in Eq. (11.2). The commutative property for convolution, if (11.2) holds true, can be easily verified by the right hand-side of (11.2) as multiplication itself is of course commutative.

$$y[n] = x[n] * h[n] \leftrightarrow DFT\{y[n]\} = DFT\{x[n]\} \cdot DFT\{h[n]\} \tag{11.2}$$

$$y[n] = IDFT\{DFT_x \cdot DFT_h\} \tag{11.3}$$

Hence, the convolved output $y[n]$ can be computed via performing the DFT for each individual signal $x[n]$ and $h[n]$, followed by multiplication of the spectra, and finally computing the inverse Fourier transform of the product. One of the main reasons convolution is more convenient to implement via the DFT (actually FFT) on the computer is none other than the advantage gained in processing speed for the two FFTs and one IFFT vs. Eq. (11.1). Generally speaking, it is more efficient using the FFT-based convolution than using the time-domain convolution version. The proof of (11.2) is not that hard to produce (and will set this one also aside for a lazy Sunday afternoon at the local coffee shop) but if we think about it for a little bit, Eq. (11.2) intuitively makes good sense. Convolving two signals such as the impulse response of a hall and the dry recording of a voice will result in the voice being heard "within" the hall or the hall "within" the voice. Looking at this phenomenon from the frequency-domain, convolution will result in each signal's spectrum getting "colored" by the other signal's spectrum via multiplication — multiplication in a Boolean algebraic sense is AND, accentuating commonalities whereas addition is OR which is all encompassing.

11.1 *Circular Convolution and Time-Aliasing*

The convolution method we have discussed above is actually called *circular convolution* and comes with artifacts. The artifact of circular convolution is that the resulting signal after convolution is *time-aliased.* To try to understand what time-aliasing is and why this happens, consider the following example. Let's say we have two signals, the input $x[n]$ having a length of $M = 512$ samples and the impulse $h[n]$ at $L = 500$ samples. If we use the time-domain convolution algorithm to compute $y[n]$, we know that we will get $512+500-1$ samples (refer back to Chap. 5, Sec. 5 if this seems bizarre). That is, $M + L - 1$ is the resulting convolved signal length that will be produced. Now, if we use the frequency-domain version to compute the DFTs, we would probably by default use the largest window size of the two and zero-pad the shorter signal with zeros. In other words, use a DFT size of $M = 512$ (we would in practice, however, use the FFT algorithm but the results will be the same) and zero-pad the impulse signal's tail with 12 zeros making both signals equal in length. After this, we would follow it up with multiplication in the frequency-domain and finally proceed to do the IDFT of the of spectral multiplication of $X^d[\cdot]$ and $H^d[\cdot]$. So far so good . . . or is there a problem? There is indeed a problem. You will have realized that after IDFT, we only have 512 time samples and not 1,011 $(M+L-1)$ that is required. This is referred to as time-aliasing.

Figure 11.1 shows an example of a correct version of DFT-based convolution for input of length $M = 146$ samples (top of Fig. 11.1), impulse response of length $L = 73$ samples (middle figure), and the convolved output at the bottom of Fig. 11.1. As expected, when the two signals are convolved, the output yields a sequence of $146 + 73 - 1 = 218$ samples (bottom figure). If we were to use a DFT size of 146 samples for both the input and impulse, perform the DFT for each, multiply the resulting spectra, and then take the IDFT of the product, we would only get 146 time samples, although

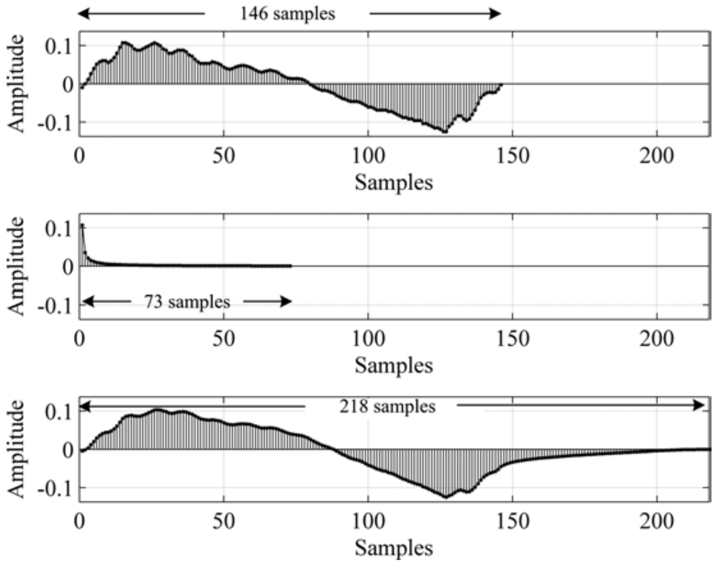


Fig. 11.1. Input signal (top), impulse (middle), and convolution output (bottom).

the correct size should be $M + L - 1 = 218$. In this case, time-aliasing occurs as the DFT size is too small and the 72 samples $(218 - 146 = 72)$ corresponding to the tail bit of the correctly convolved signal will spill over to the beginning of the signal. This is depicted in Fig. 11.2, which shows three sub-plots of the same convolved signal at the top, middle, and bottom plots. We know that the time-aliased version has only a resulting signal length of 146 samples whereas the correctly convolved signal 218 samples. The spill over occurs as the 72 samples at the tail end will add to the beginning of the convolved signal thus distorting the output. The net time-aliased convolved signal is shown in Fig. 11.3.

Time-aliasing can be easily fixed by properly zero-padding each signal to the correct DFT length before multiplication: zero-pad to $M + L - 1$. When using the FFT, we would obviously want a window size of power of two for efficiency. So, when choosing the size of the FFT, we simply select the next closest power-of-two window size, pad it with zeros before multiplication, and delete the excess samples in the time-domain after IFFT to meet the $M + L - 1$ requirement as summarized in Fig. 11.4.

The inverse of convolution is referred to as *deconvolution*. Stated in another way, let's say you know the impulse response and the output of a system and want to know what the input was. Is there a way to get this

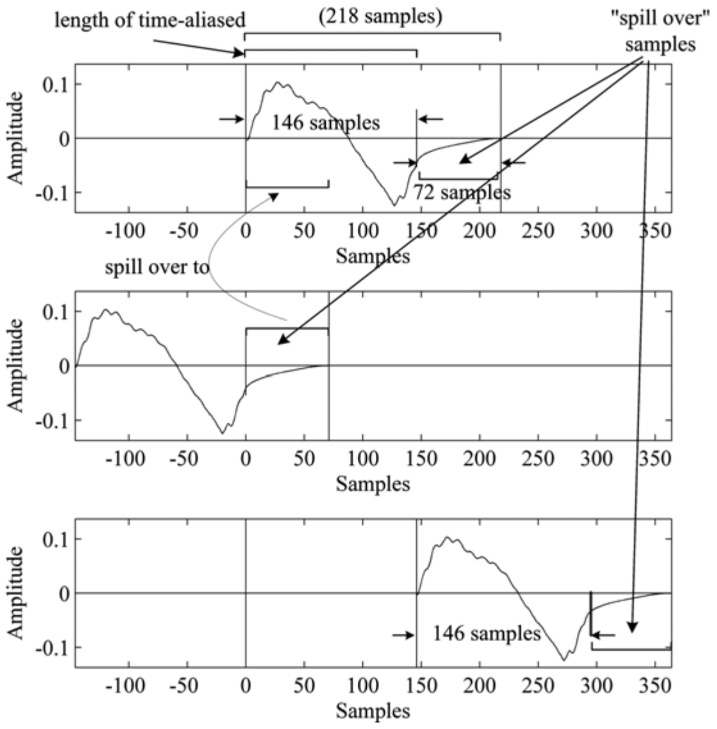


Fig. 11.2. Time aliasing due to circular convolution.

input? The answer is yes — through deconvolution. Deconvolution is far too complex to deal with in the time-domain but fortunately in the frequency-domain, it becomes merely a matter of division of the DFT frames in question. That is, as convolution is a multiplication process of the spectral frames of the input signal and impulse response in the frequency-domain (11.4), any one of the three components (output, input, impulse response) can be computed if any of the other two are known.

$$DFT_y = DFT_x \cdot DFT_h \tag{11.4}$$

For example, if we know the impulse response and the output of a system, we can quickly deconvolve the two by dividing the DFT of $h[\cdot]$ and $y[\cdot]$ and compute $x[\cdot]$ according to (11.5).

$$DFT_x = \frac{DFT_h}{DFT_y} \tag{11.5}$$

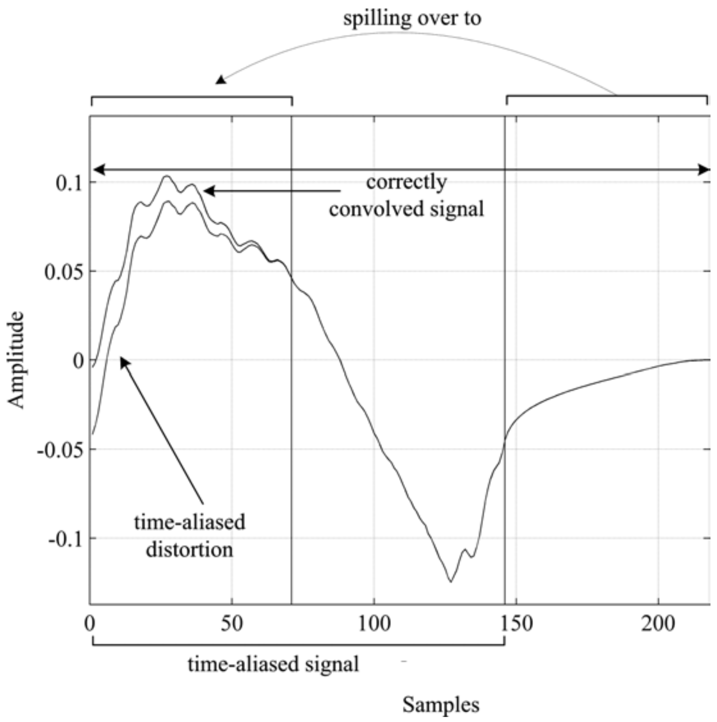


Fig. 11.3. Net result of time aliasing in output due to circular convolution.

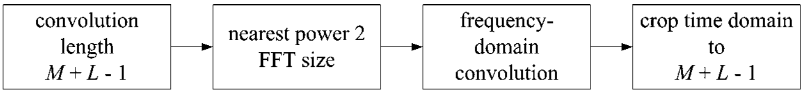

convolution length $M + L - 1$
nearest power 2 FFT size
frequency-domain convolution
crop time domain to $M + L - 1$

Fig. 11.4. Frequency-domain convolution procedure using FFT.

12 One More Look at Dithering

Remember a long time ago in Chap. 1, Sec. 7.1 when we talked about dithering and how quantization and rounding can lead to harmonic distortion? This can now be verified using the Fourier transform and knowing that a square wave can be described in terms of odd harmonics as discussed in additive synthesis in the beginning of this chapter. Let's take another look at Fig. 12.1 from Chap. 1.

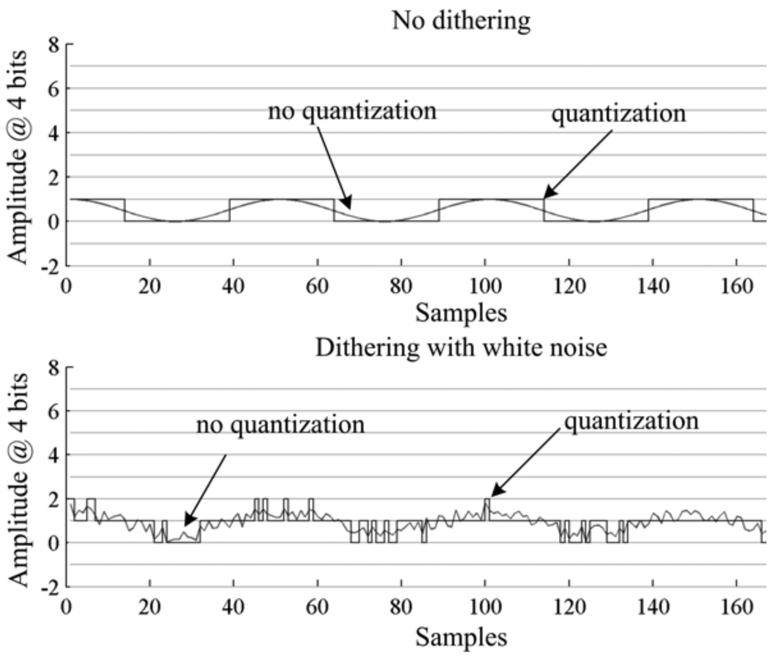


Fig. 12.1. No dithering (top), dithering with white noise (bottom).

The top of Fig. 12.1 shows what happens during quantization and sampling for a sine wave at 4 bits and the bottom plot the white noise-based dithering result before and after quantization. Quantized signals generally look square and rigid while analog signals smooth. Figure 12.2 shows the Fourier transform of the above two signals and as we can see, it shows as expected, harmonic distortion in the upper plot — harmonics that were not there prior quantization and rounding are manifested in the digitized output. The lower plot depicting the dithered version shows only one sinusoidal component (only one strong peak) at the appropriate frequency without any additional harmonics at the expense of low energy noise padding the floor. The noise floor is minimal and for most practical purposes cannot be perceived as they have too little energy, especially when the SNR is at a healthy level resulting in the "signal" *masking* out the background noise. This sort of phenomenon is also quite intuitive — just by looking at the plots we will probably immediately notice the peaks more than the grassy-looking noise floor. The strong peaks seem to overshadow the "background" so-to-speak.

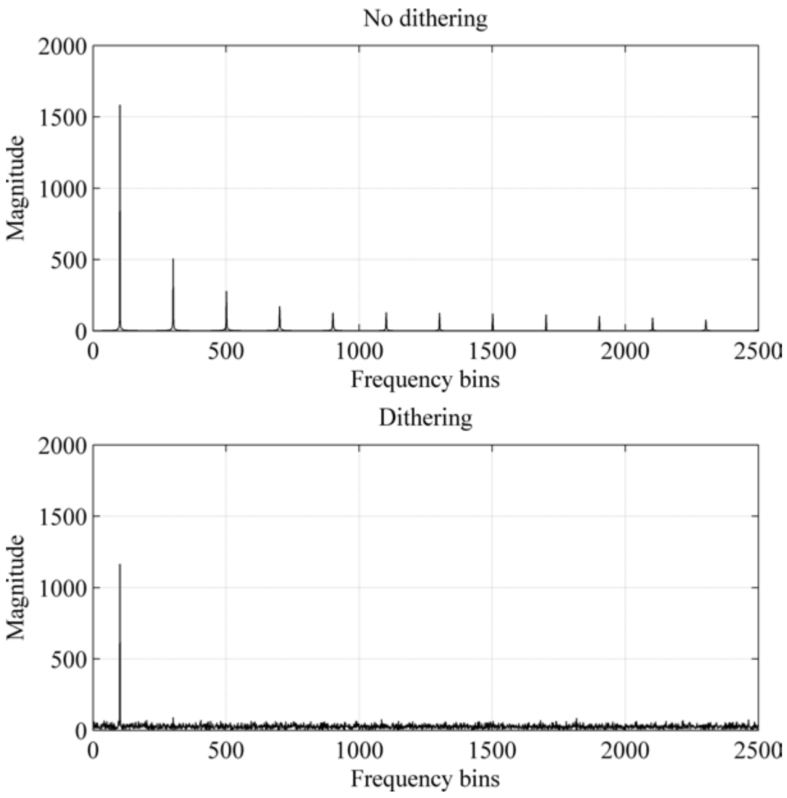


Fig. 12.2. Fourier transform of non-dithered and dithered sine wave.

13 Spectrogram

Although it is quite interesting to look at a single DFT frame and analyze a signal's frequency structure, not that much information can be gained and extracted, especially if it is a real signal — like every-day audio signals used in music and compositional situations. Those signals change with time — static signals would probably make a musical piece quite uninteresting unless of course it is about the minute subtleties or a prolonging drone. But even in such a piece that has seemingly no change, there would still be some amount of variance although maybe not perceived as foreground material. We have seen in Sec. 5 that the short-time Fourier transform can be helpful in this regard — allow analysis of transient-based sounds while keeping the frequency resolution robust for each spectral frame. The spectrogram in

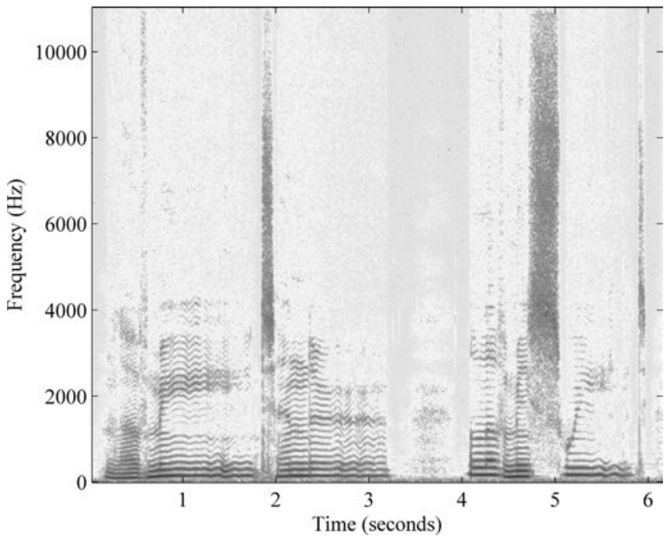


Fig. 13.1. Spectrogram plot.

essence is a multi-vector-based STFT mapped onto a 3-dimensional plot of time vs. frequency with the 3rd dimension corresponding to magnitude usually color coded (or mapped onto a grey scale system). It is common practice to have the x axis refer to time and the y axis frequency as shown in Fig. 13.1.

When first looking at a spectrogram, it may be difficult to immediately see what is happening (nevertheless they almost always look cool!). One very helpful part of visually analyzing signals via the spectrogram (and hence Fourier transform) is that whatever is horizontal (periodic or quasi periodic) in the time-domain will become vertical in the frequency-domain. If the periodicity is sustained (goes on for a while) we will generally see a horizontal line forming in the spectrogram. If it is not periodic (horizontal) in the time-domain, it will show up as a flat spectrum in the DFT frame and a block of noise in the spectrogram — i.e. pattern-less and vertical. Thus, in our spectrogram plot, armed with these very basic attributes regarding signals, we can observe that at just before 2 seconds, between 3 and 4 seconds, at around 5 seconds, and by the end of the spectrogram, non-patterned (non-periodic) attributes or noise-like sounds are present. While in the other parts, pitched signals with harmonics are observable with occasional low frequency modulation as well as amplitude modulation.

The signal in this example is in fact a male voice singing "love me tender, love me sweet." In general, the noisy areas pertain to the attack parts of the signal, where around the 2 second mark the "t" in "tender" is sung and in the 5 second area the "s" in "sweet" is vocalized. We will discuss vocoders in the next chapter and also present the LPC filter which uses the idiosyncrasies of consonants and vowels to codify and extract the so-called LPC coefficients.

14 Fourier Transform Properties and Summary

In this section, we have included some of the main properties of the Fourier transform as shown in Table 14.1. Some of them may seem more obvious than others but they should all, nevertheless, be helpful when dealing with the Fourier transform. Table 14.2 gives an overview of the various types of Fourier transforms.

Table 14.1. Summary of important Fourier transform properties.

Property	Summary	
Linearity	$g[n] = ax \cdot [n] + b \cdot y[n] \leftrightarrow Z(\theta) = a \cdot X^f(\theta) + b \cdot Y^f(\theta)$ where a, b are constants	(14.1)
Periodicity	$X^f(\theta) = X^f(\theta + 2 \cdot \pi \cdot k)$ where θ is real, k is an integer	(14.2)
Time shift	$y[n] = x[n-m] \leftrightarrow Y^f(\theta) = e^{-j\theta m} X^f(\theta)$ where m is an integer	(14.3)
Frequency shift	$y[n] = e^{-j\kappa n} \cdot x[n] \leftrightarrow Y^f(\theta) = X^f(\theta - \kappa)$ where κ is real	(14.4)
Multiplication	$y[n] = x[n] \cdot h[n] \leftrightarrow Y^f(\theta) = X^f(\theta) * H^f(\theta)$	(14.5)
	$y[n] = x[n] * h[n] \leftrightarrow Y^f(\theta) = X^f(\theta) \cdot H^f(\theta)$	(14.6)
Symmetry	$X^f(-\theta) = \bar{X}^f(\theta)$	(14.7)
	$\mathrm{Re}\{X^f(-\theta)\} = \mathrm{Re}\{X^f(\theta)\}$	(14.8)
	$\mathrm{Im}\{X^f(-\theta)\} = -\mathrm{Im}\{X^f(\theta)\}$	(14.9)
	$\left\lvert X^f(-\theta)\right\rvert = \left\lvert X^f(\theta)\right\rvert$	(14.10)
	$\angle X^f(-\theta) = \angle - X^f(\theta)$	(14.11)

Table 14.2. Summary of different Fourier transforms.

Time / Frequency	Continuous	Discrete
Continuous	Continuous-Time Fourier Transform $$X^F(\omega) = \int_{-\infty}^{+\infty} x(t)e^{-j\omega t}dt$$ $$x(t) = \frac{1}{2\pi}\int_{-\infty}^{+\infty} X^F(\omega)e^{j\omega t}d\omega$$	Discrete-Time Fourier Transform $$X^f(\theta) = \sum_{n=-\infty}^{n=+\infty} x[n]\cdot e^{-j\theta n}$$ $$x[n] = \frac{1}{2\pi}\int_{-\pi}^{+\pi} X^f(\theta)\cdot e^{j\theta n}d\theta$$
Discrete	Continuous-Time Fourier Series $$x(t) = x(t+T)$$ $$X^S[k] = \frac{1}{T}\int_0^T x(t)\cdot e^{-j\frac{2\pi kt}{T}}dt$$ $$x(t) = \sum_{k=-\infty}^{k=+\infty} X^S[k]e^{j\frac{2\pi kt}{T}}$$	Discrete Fourier Transform $$X^d[k] = \sum_{n=0}^{n=N-1} x[n]\cdot e^{-j\frac{2\pi kn}{N}},$$ $$0 \le k \le N-1$$ $$x[n] = \frac{1}{N}\sum_{k=0}^{N-1} X^d[k]\cdot e^{j\frac{2\pi kn}{N}}$$

```
f = 4000;
fs = 22050;
fftLength = 1024;                 % window length

x     = sin(2*pi*f*[0:1/fs:1]); % make the sine wave
ft    = fft(x, fftLength);      % do FFT, use rect. window
ftMag = abs(ft);                % compute magnitude

% plot the results both in linear and dB magnitudes
subplot(2, 1, 1), plot(ftMag)
title('Linear Magnitude')
ylabel('magnitude'), xlabel('bins')

subplot(2, 1, 2), plot(20*log10(ftMag))
title('dB Magnitude')
ylabel('dB'), xlabel('bins')
```

Code Example 14.1. FFT.

15 MATLAB® and Fourier Transform

As you may have already anticipated, MATLAB® (when used with the *Digital Signal Processing Toolbox*) includes FFT and spectrogram (STFT) functions as well as a number of very useful frequency-domain tools. The functions available in MATLAB® are quite straightforward to use — at least in a sense that the STFT and FFT are readily available and straightforward to use. An example of using the FFT is shown in Code Example 14.1 where the resulting plots are the three subplots of the FFT of a sine wave at 4 kHz — linear magnitude, log magnitude, and dB magnitude.

16 Musical Examples

It is no exaggeration to say that the Fourier transforms is probably used everywhere in the musical world today, especially in the area of plug-ins for manipulation and modulation of sounds in digital audio workstations. Most composers who work with computers have at some point used, or are actively using (whether aware of it or not) Fourier transform-based tools to compose, edit, mix, master, and manipulate music. Obviously, there is nothing special regarding its usage; on the contrary it is probably so common and ubiquitous that no-one really talks about it per se. There are, however, a number of interesting compositions that have borrowed ideas pertinent to the Fourier transform during a time when composers did not have computers automatically churn out the specific spectral structure of sound objects. In the 1950s, a wealth of research was being conducted by composers especially in Germany and one such composer was Karlheinz Stockhausen who was very much concerned (in the early years) with *Elektronische Musik* opposed to *musique concrète*. Elektronische Musik referred to production of sounds via oscillators whereas music concrète techniques involved compositional ideas centered on sound production via *found sound objects* — sounds recorded with a microphone and a tape recorder. In his piece called *Klangstudie I* (sound studies) composed in 1952, Stockhausen applied additive synthesis methods whereby layering the output of oscillators in a very tedious and laborious fashion — playing two oscillators at a time and recording them onto a tape machine, then playing another pair of tones using the same oscillators while playing back the pre-recorded sinusoids via the same tape machine, and finally recording the resulting summed sinusoids onto a second tape recorder.

These studies seemingly were based on physicist Herman von Helmoltz's studies in timbre. Helmholtz used the so-called *Helmholtz resonators* to literally perform manual Fourier analysis using highly tuned flasks. These Helmholtz resonators had two openings, one to let the sound in from the outside world, and another one that would be held against the ear for monitoring. Each flask would be tuned to a specific frequency and only produce a tone if that particular frequency component existed in the input signal! A more recent composer named Paul Koonce has also used the Fourier transform widely in a number of his musical works. One such piece is *Hot House* (1996). In this piece, the composer deals heavily with the issue of *timbral enharmonic change.* That is, addressing issues concerning timbral commonalities and changing (or morphing) from one sound object to another by using common timbral features and threading them in a continuous sonic projection. For example, in one part of the piece, the foreground curtain of sound is initially that of car horns which slowly change shape into a chugging train. The train in turn seemingly comes to a stop and is perceived as if punctuating the ending of a musical section as it reaches a train station while expelling its pent-up steam. The steam briefly affords a foggy atmosphere where the commotion of people chattering and shouting accentuates the instability of the sonic environment and all of a sudden one is transported to a completely different sound space where the connecting dots from one space to another are the hissing steam of the train and the sound of a tea kettle in someone's home ... and the journey continues ...

References and Further Reading

Porat, B. 1997. *A Course in Digital Signal Processing.* Wiley and Sons.

Cooley, J. and J. Tukey 1965. "An Algorithm for the Machine Calculation of Complex Fourier Series", *Mathematics of Computation*, 297–301.

Chapter 9

SPECTRAL ANALYSIS, VOCODERS, AND OTHER GOODIES

1 Introduction

In our final chapter of the book, we will introduce some interesting approaches and methods in spectral analysis and synthesis for audio signals as well as musical applications. We will also discuss an intriguing topic related to the voice, namely topics concerning *vocoders*. We will start off the chapter with a number of spectral analysis techniques including peak detection, fundamental frequency detection, and then move on to wrap our heads around vocoders that have found much popularity in the area of music composition, production, and speech synthesis research alike.

1.1 *Musical signals and important nomenclatures*

With almost certainty, independent of our cultural heritage, we have probably been exposed to traditional musical sounds, sounds that we may not consider to be musical, and sounds that lie in between musical and non-musical. It is difficult or perhaps even impossible to categorize sounds into two camps — musical sounds vs. non-musical sounds. The difficulty probably lies heavily on the issue of context which often seems to play a critical role in the perception of sound and music. For some folks, noisy signals may not be considered musical at all (such an airplane propeller spinning or an army of frogs in a large pond late at night) while others may regard cacophony as totally musical and utilize noisy materials for a composition. Some great musical examples include *Back in the USSR* by the Beatles and *Money* by Pink Floyd. In both pieces, sounds that are normally not considered "musical" are used very effectively to augment the experience of the two songs. In a way, it is probably fair to say that the spectrum of sound is really infinite, which may be loosely regarded as spanning from a single pure sine tone, to complex tones, to white noise. However, the characteristics of a small section of this infinite spectrum of sonic amalgam, closely adhere to particular patterns and behavior and are likely (at least at current time) the most commonly accepted types of musical sounds. These sounds which we have seen in abundance even in this book are often pitched sounds — sounds that are quasi-harmonic eliciting a sense of pitch.

Sound objects that are pitch-based have a special trait — spectral peaks in the frequency-domain that follow a certain pattern which go by many names including peaks, harmonics, partials, and overtones. Although these terms are similar in many ways, they do refer to very specific characteristics and thus, at times, can result in their misuse. The term peak is probably the most general expression used in describing spectral saliencies referring to any projecting point characterized by positive slope followed by a negative slope in the spectrum. Peaks also generally stand out from the background noise which is probably a collection of small peaks and part of the noise floor itself, in which case, they should perhaps not be referred to as peaks (although technically correct). The location of peaks on the frequency axis also does not necessarily have to abide to any sort of logical pattern. Those types of peaks that do not adhere to frequency-based patterns are also referred to as partials — strong and salient frequency components that constitute a complex tone which necessarily need not be in some sort of integer relationship to each other. Peaks and partials that are characterized in integer (quasi or otherwise) relationship to one another in frequency are referred to as harmonics. This pattern usually constitutes an integer relationship to the fundamental frequency — the frequency that we would normally perceive as the pitch as shown in Eq. (1.1) for integer vales n.

$$f_n = n \cdot f_0 \tag{1.1}$$

The first harmonic is usually referred to as the fundamental and is often given a special case notation as f_0, the 2nd harmonic ($n = 2$) as the first occurring octave, the 3rd harmonic 3 times the fundamental frequency and so on. The term overtone is often used in musical contexts and refers to the harmonics that lie above the fundamental frequency. This naming scheme may be a potential source for confusion as the 1st overtone corresponds to the 2nd harmonic, and the 2nd overtone to the 3rd harmonic. The degree of confusion may even be potentially worsened when dealing with complex tones that only have odd harmonics (or even harmonics) in which case the 2nd harmonic which will be the 3rd harmonic, will be equivalent to the 1st overtone of the harmonic series whereas the 5th harmonic the 2nd overtone. So, a word of caution is advised in making sure the right nomenclature is used in the right situation.

A pitch-based sound object's characteristic timbre or sound color, as some like to refer to it, is profoundly influenced by the harmonic structure — location of the harmonics and the magnitude of the harmonics. This ranges from the typical clarinet sound exemplified by strong presence of odd harmonics, to stringed instruments demonstrating subtle harmonic expansion (tendency of the location of harmonics gradually increasing with respect to the ideal locations when moving towards the Nyquist) caused by the stiffness of strings. In most cases, real harmonics found in acoustic instruments and other musical situations rarely line up perfectly according to Eq. (1.1) but are found near those ideal harmonic locations. This error between the ideal harmonic location and actual harmonic location is perhaps one of the most interesting aspects of a sound's timbre, making it seem "alive" and "natural." I will leave the topic of timbre at this point as it requires a few chapters in itself, if not a whole book, to delve deeper into it, and will get back on track by presenting some basic spectral analysis techniques in the next section.

2 Spectral Analysis

Spectral analysis refers to the scrutiny and examination of a signal in the frequency-domain most commonly via the Fourier transform. When analysis is conducted on the computer, it is usually accomplished via the DFT and the STFT as discussed in our penultimate chapter. To state it bluntly, when performing spectral analysis, we are striving to find patterns, any peculiarities, and salient features that will give us clues as to what idiosyncrasies (if any) exist. In this section, we will cover a few of the topics and approaches surrounding spectral analysis.

2.1 *Long-term average spectrum (LTAS)*

The long-term average spectrum shown in Eq. (2.1) is a method for viewing the average spectra of a sound object (or any signal). The underlying algorithm takes advantage of the fact that the background noise is averaged towards the floor while reoccurring peaks are accentuated and made clearer. This result is due to the averaging nature of the algorithm which has an effect of canceling out background noise and accentuating peaks if they occur in some sort of repeating pattern, especially patterns that do not deviate much from the average.

$$LTAS = 1/J \sum_{m=0}^{M-1} X_m^d \tag{2.1}$$

M is the number of STFT frames, m the frame index, X_m^d the mth spectral frame containing magnitude and phase components. However, when using LTAS (due to averaging which is basically a low-pass filter), some amount of the transient information is lost. So, depending on the application and what your needs are, LTAS may often be helpful especially as a pre-analysis process to try to get an overview before zooming-in to conduct a more detailed analysis. Alternatively, for some applications, an elaborately thorough scrutiny may not be required, in which case the LTAS may be a perfect solution. Figure 2.1 shows an electric bass guitar signal at $f_s = 44.1\,\text{kHz}$ and Fig. 2.2 shows a plot of the DFT of the middle part of the electric bass and the corresponding LTAS with window size 1,024 samples and overlap of 40%. The top plot in Fig. 2.2 shows the 139th frame of the electric bass and the bottom plot the LTAS of the signal itself. In this case, we can get a better picture of the overall spectral structure reflecting the harmonics and fundamental frequency of this sound example.

2.2 *Log vs. linear*

When analyzing the spectrum of a windowed signal, there are usually a few ways to view and analyze the results. One of the options that we have is log base 10 vs. linear magnitude representation. The main difference between

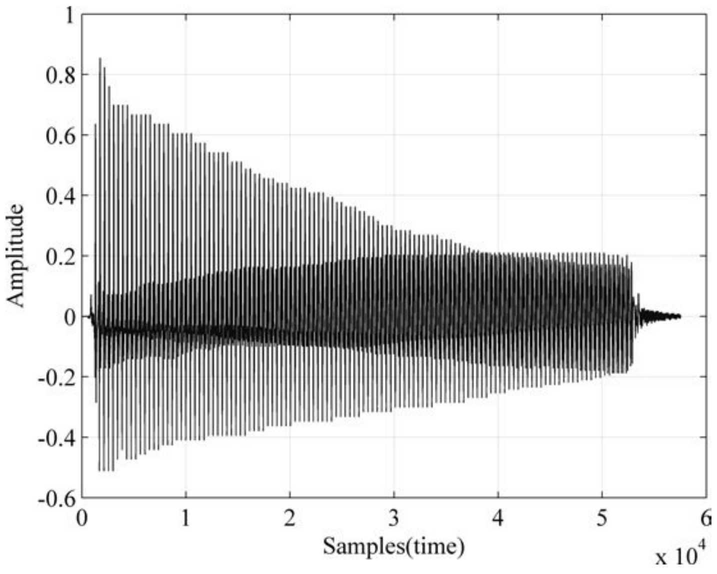


Fig. 2.1. Plot of electric bass waveform.

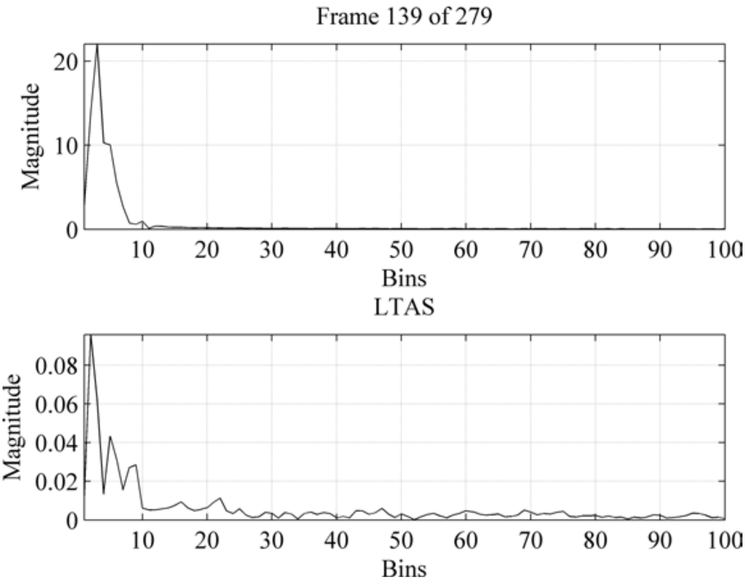


Fig. 2.2. Single DFT (top) and LTAS (bottom) plots of electric bass.

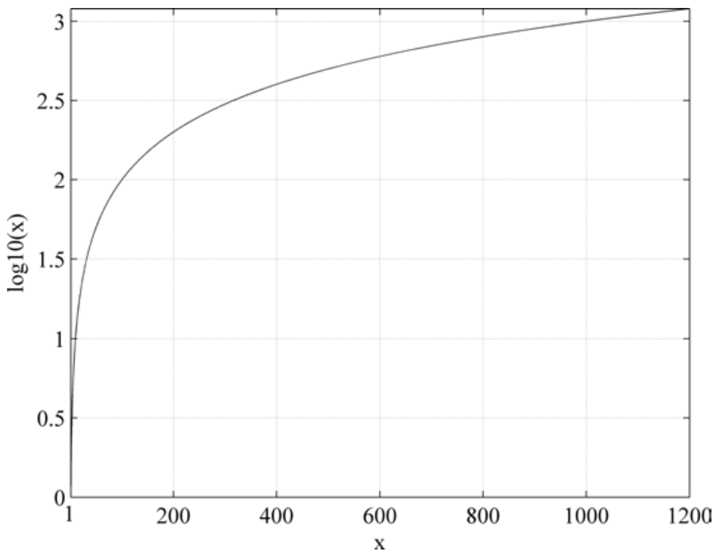


Fig. 2.3. Log function.

log and linear representations for displaying a spectrum's magnitude is that the log version generally brings out more of the details and the linear version less details as discussed in Chap. 6. It may be a bit misleading to say that we get more details in the log magnitude plot as there is really no additional information gained. However, due to the characteristics of the log function as seen in Fig. 2.3, which generally makes larger values smaller and smaller values larger, subtleties that would normally not be noticeable may be brought out. Figures 2.4 and 2.5 show plots of the linear and log magnitude plots of the same electric bass guitar from our previous example.

The linear magnitude plot, however, is not useless either and as a matter of fact it is far from it and is as important as the log version. It is especially helpful in showing the saliency of strong peaks and resonant structures as we can vividly see in both Figs. 2.4 and 2.5. The choice between log and linear should be a function of what we want look for. No one version is better than the other, they are just different.

The log function is also often applied to the x axis frequency. As perception of frequency is more logarithmic than linear, the log-frequency representation may at times be additionally relevant to the way humans perceive sound when dealing with pitched audio signals for example.

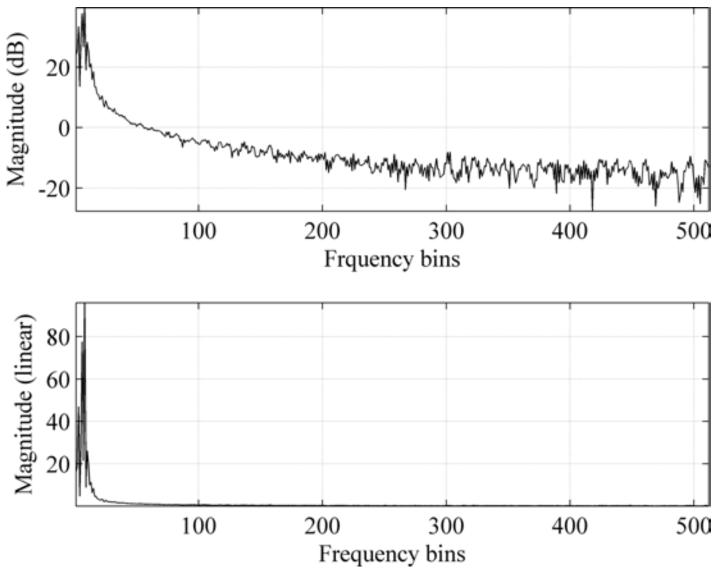


Fig. 2.4. Log and linear spectral magnitude (full spectrum).

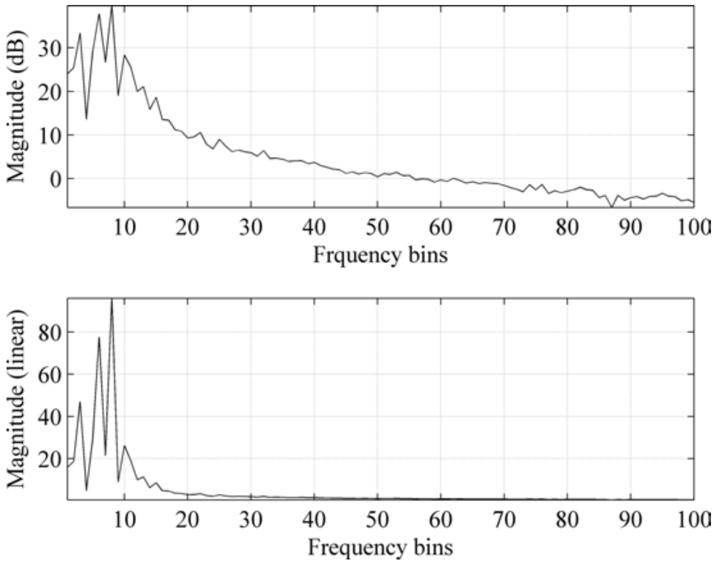


Fig. 2.5. Log and linear spectral magnitude (bins 1 to 100).

One simple way to understand the nonlinearity of pitch is to consider the octave. The octave is defined as twice the fundamental frequency (or more generally twice the frequency of the reference frequency). For example, if we have a fundamental frequency $f_0 = 100\,\text{Hz}$, the 1st octave would be at 200 Hz. The next octave would be at 400 Hz ($100 \cdot 4$ or $200 \cdot 2$ the octave of the 2nd harmonic) and the octave following the 400 Hz would be 800 Hz and so on. The increase in octave is generally perceived by our hearing system as being just that — an increase of twice the frequency. However, the increase in the actual amount of frequency does not seem linear at all. For the above example, let's limit to the allowable frequency resolution to integer values. The first "frequency distance" between f_0 and f_2 is 100 Hz, for f_2 and f_4 it would be 200 Hz, f_4 and f_8 400 Hz, and so on. Although we perceive the frequency change or the distance to be pretty much the same for each octave shift, the actual frequency distance expands as we go up the frequency scale. With the log function this expansion is made more linear in *perception* although not mathematically speaking. If this still sounds a bit funky, think of the piano keyboard. Keys on the piano all have the same physical spacing — all white keys are of a certain width and all black keys are of another width. However, although the frequency distance increases with every increase in octave, the physical keyboard spacing stays the same — linear. We as performers also feel that this layout "feels" right, meaning that the piano keyboard does not feel awkwardly scaled. It would indeed be interesting to have the piano key spacing change as is the case for stringed instruments — the higher you go up the frequency, the shorter the fret distance becomes. This is of course due to the increase in frequency — as one moves up the frequency scale on the guitar neck it expands in frequency resulting in compression of the wavelength and thus shortening of the distance between frets (wavelength and frequency are inversely proportional).

2.3 *Spectral peaks, valleys, and spectral envelope*

Looking at the spectrum in Fig. 2.6, we note some interesting features. Two notable ones are the spectral peaks and valleys. Spectral peaks are especially important as they give information regarding the salient resonant structure of a sound object. Valleys can be loosely viewed as dampening or anti-resonant locations as well as zeros from a filter nomenclature perspective — just imagine having a pole at each of the peaks and zeros in between peaks.

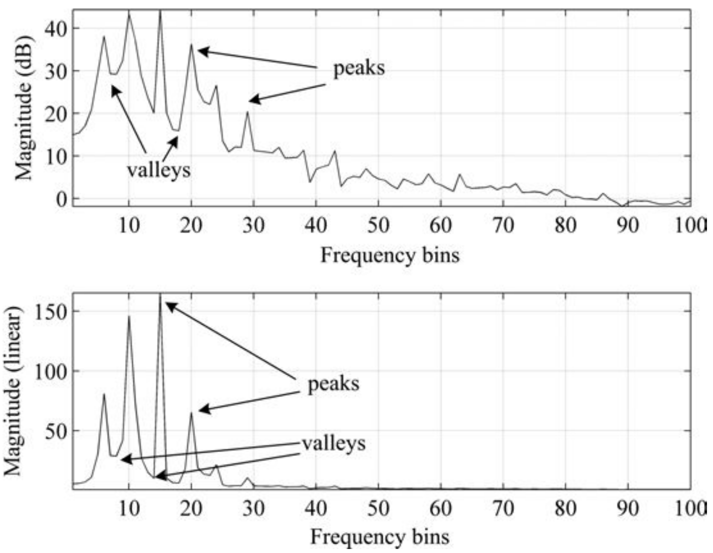


Fig. 2.6. Spectral peaks and valleys.

The most pronounced peaks and valleys of a spectrum contribute to the global quality of a sound object. However, there is also a lot of subtlety in the spectrum which are not unimportant but at the same time perhaps comparatively not as significant as prominent peaks and valleys. In a way, when we look at a spectrum, what we see at first glance is the outline/contour and shape of the spectrum defined by its magnitude values. This is referred to as the *spectral envelope*. It probably seems quite straightforward for us (humans) to look at such a spectrum and analyze the saliency of peaks (and valleys), but to do it automatically on the computer can be a more difficult task. This seems to be the case for many problems — what humans do with relative ease is more difficult to do with computers whereas what's simple to accomplish on the computer (such as summing the first 2 million odd integer numbers starting at 0, squaring it, and dividing the results by 5.100276) is hard for a person to do.

The definition of a peak as mentioned in the beginning of this chapter is actually quite straightforward and is described as any point that involves a particular change in slope — positive to negative. Valleys are the opposite, with the slope changing from negative to positive. This sort of analysis is, however, sometimes not entirely informative as the resulting data set is very large as shown in Fig. 2.7.

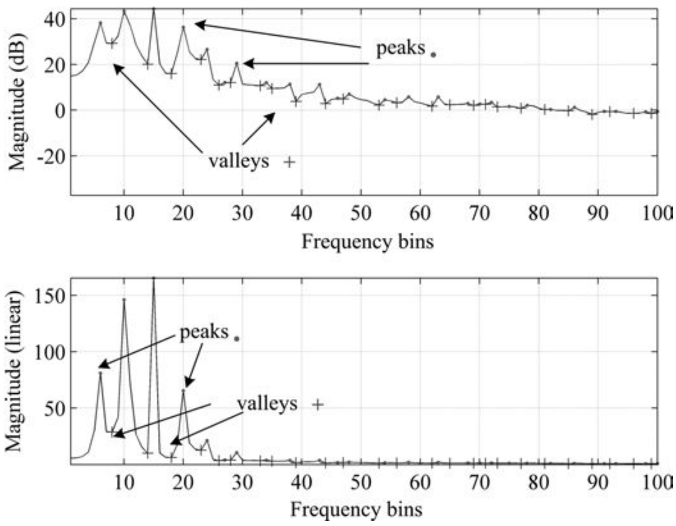


Fig. 2.7. All peaks and valleys of a spectrum.

In Fig. 2.7, we merely computed the change in polarity by walking along the spectrum and defining peaks as positive to negatively changing locations and valleys as negative to positively changing points. Alternatively, we could have analyzed for valleys using a peak analyzer by inverting the spectrum and looking for positive to negatively changing slope points. Various approaches in obtaining more useful data in the spectrum exist, including setting a threshold to filter out, say, peaks that are below 50% of the maximum peak; setting a threshold so that the difference between a peak and the immediate adjacent valley be greater than a certain static value or adaptive value; or simply picking the first 10 largest peaks and 10 smallest valleys as the most representative ones. Obviously all of these approaches come with their own problems and there is indeed no perfect algorithm due to the vast array of signals out there.

There are many ways of computing the spectral envelope including low-pass filtering the spectral frame or using the RMS windowing method we exploited for time-domain envelope computation. The RMS spectral envelope is depicted in Fig. 2.8. Note that when using the RMS envelope approach, down-sampling takes place due to the use of windows — the number of data points representing the spectrum will be decimated. Thus, some sort of interpolation often needs to follow the RMS algorithm to equalize the envelope size with respect to the spectrum's original size.

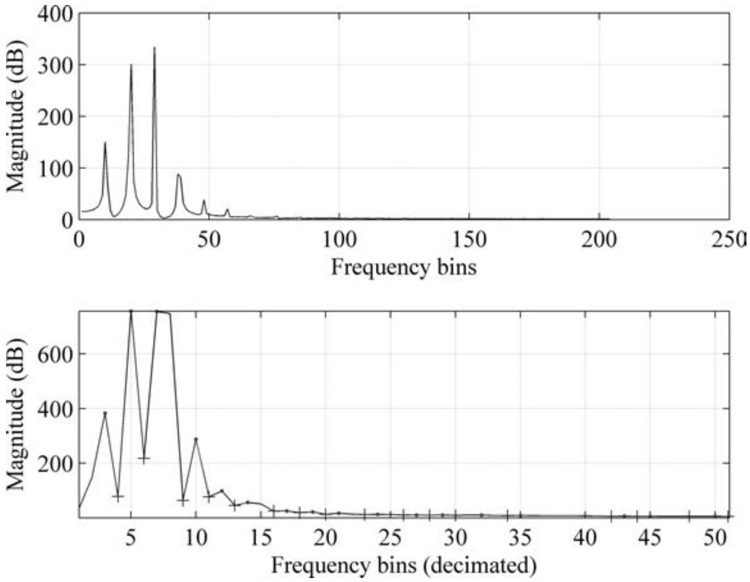


Fig. 2.8. Peaks of RMS spectral envelope.

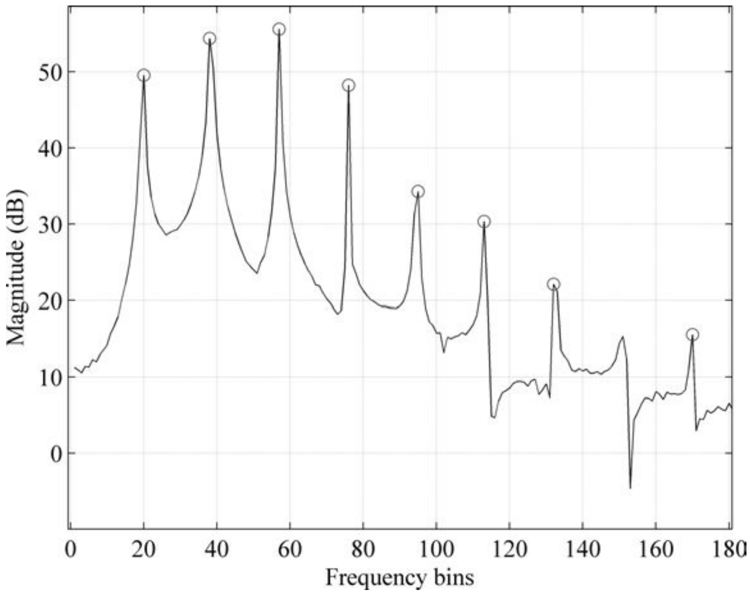


Fig. 2.9. Direct salient peak extraction from full spectrum.

Other ways of extracting salient peaks is to try to filter out passing peaks — peaks that are not part of the ensemble of the salient peaks. This, however, is more difficult as the issue of ascertaining which are "passing" peaks and which are not is easier said than done. Figure 2.9 shows the result of such an algorithm. It does a pretty decent job at picking salient peaks but is by no means perfect — whatever perfect may refer to. More information and documentation regarding the algorithm can be found on the Internet, downloadable at http://music.princeton.edu/~park.

2.4 *Extraction of fundamental frequency and harmonics*

Sound objects that elicit a sense of pitch usually have a spectral structure that includes the fundamental frequency and its various harmonics with the fundamental frequency being the pitch that we normally perceive. Figure 2.10 shows such an example with a fundamental frequency of 100 Hz.

It is, however, not always the case that the fundamental is the strongest peak and at times it does not even exist (this is referred to as the *missing fundamental* phenomenon). In the case of the missing fundamental, we will still perceive the fundamental frequency as the pitch although it is not physically present in the spectrum. Furthermore, musical pitch of an audio signal is a perceptual feature of sound whereas frequency is an absolute term, which can be measured with an oscilloscope for example. Musical sounds in the real world are quite complex consisting of many harmonics and it should then perhaps not be that much of a surprise that with the presence of a healthy number of harmonics, the sensation of pitch is improved (Truax 1978). There is still much active discussion regarding exactly how we perceive pitch and although it is quite an interesting and very important topic, we will redirect our attention to exploring DSP algorithms for automatic fundamental frequency estimation.

There are many ways of computing the fundamental frequency and there is no one ultimate method to estimate it as an algorithm that does a great job on one type of signal can perform poorly for other types of

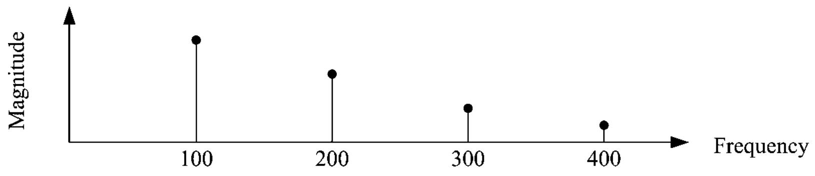


Fig. 2.10. Harmonic structure of a pitched spectrum.

signals. We have already seen several time-domain-based pitch estimation algorithms in Chap. 2, Sec. 5. Some additional algorithms for estimating the fundamental frequency include *cepstral analysis*, *harmonic product spectrum*, and the *inverse comb-filter* method. The last one technically does not belong to the frequency-domain algorithm category but nevertheless exploits the frequency response of the inverse comb-filter to estimate the fundamental frequency. We will briefly introduce these three algorithms starting with the inverse comb-filter method.

2.4.1 *Inverse comb-filtering*

Inverse comb-filtering as mentioned above is actually not a frequency-domain process per se but its application in fundamental frequency estimation can be understood conveniently in the frequency-domain. The filter's FIR difference equation is given below.

$$y[n] = x[n] - b_1 \cdot x[n-N] \tag{2.2}$$

We may remember from Chap. 7, Sec. 3.3 that the FIR forward comb-filter is given by Eq. (2.3).

$$y[n] = x[n] + b_1 \cdot x[n-L] \tag{2.3}$$

Both filters are very similar in shape as we can see in Figs. 2.11 and 2.12 but their behavior have important differences. The forward comb-filter has notches at around 0.1, 0.3, ... as seen in Fig. 2.11. The inverse comb-filter, however, has notches around 0.2, 0.4, ... — between the notches of the forward comb-filter. The way inverse comb-filters are used for fundamental frequency analysis is by capitalizing on the fact that pitched signals have a tendency of demonstrating some sort of harmonic structure. That is, when viewed via the Fourier transform, the power spectrum will show strong harmonics along with the fundamental frequency. Thus, if we employ an inverse-comb-filter and tune it so that the notches align with the harmonics and the filtered signal, after inverse-comb-filtering will lose energy as the harmonics are attenuated. The more aligned the notches are to the harmonic locations, the lower the energy will be for the inverse comb-filtered signal.

Figure 2.13 shows an example of this technique where we utilized an IIR inverse comb-filter according to Eq. (2.4) for a flute sound played at C#4.

$$y[n] = x[n] - a_1 \cdot y[n-N] \tag{2.4}$$

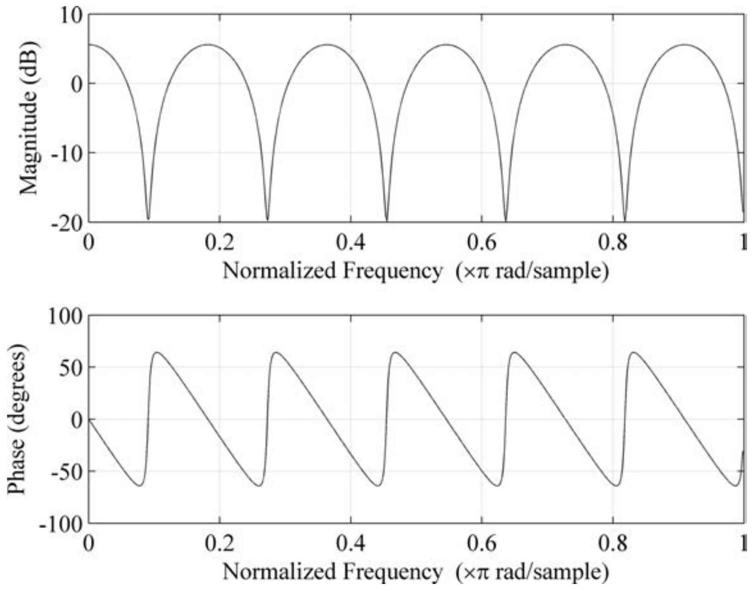


Fig. 2.11. FIR comb-filter.

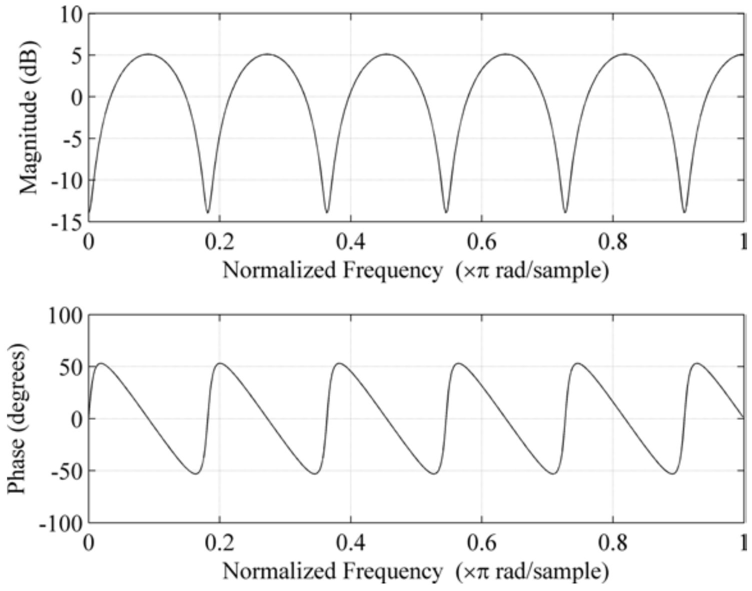


Fig. 2.12. FIR inverse comb-filter.

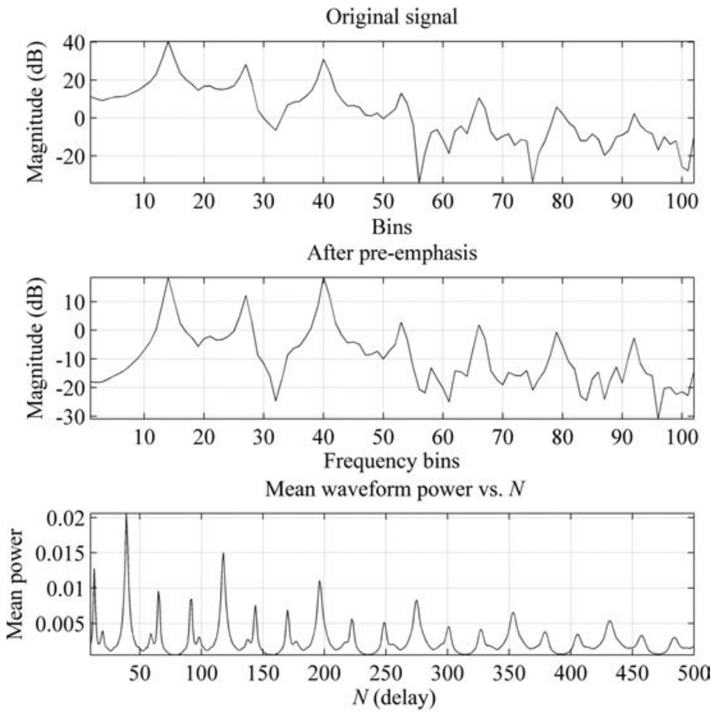


Fig. 2.13. Inverse IIR comb-filtered pitch detection with pre-emphasis filter. Original power spectrum (top), spectrum after pre-emphasis (middle), mean signal energy vs. filter delay N (bottom).

Before sweeping the inverse comb-filter to find the delay N that minimizes the output energy of the filtered signal, the input signal was subjected to a *pre-emphasis* filter. A pre-emphasis filter is simply a very basic high-pass filter which can sometimes be helpful in flattening out the spectrum of an audio signal. Musical instrument signals usually have stronger harmonics in the lower frequency areas and a decaying magnitude contour with the increase in harmonic number much like Fig. 2.10. Thus, with the pre-emphasis filter shown in Eq. (2.5), the flute signal is somewhat flattened out, generally helping the analysis when applying the inverse comb-filter as it notches each harmonic more uniformly.

$$y[n] = x[n] - b_1 \cdot x[n-1] \tag{2.5}$$

You will also remember that we already covered the sister of this simple high-pass filter, namely the 1st order FIR low-pass filter in Chap. 6, Sec. 3. Thus, the 1st order FIR high-pass filter is actually equivalent to the lowest order FIR inverse-comb-filter. The FIR low-pass filter has a similar relationship to the forward comb-filter of Eq. (2.3). Using this methodology for pitch estimation in our example resulted in a fundamental frequency of 551.25 Hz for $f_s = 44.1\,\text{kHz}$ ($N = 80$ samples, $f_0 = f_s/N = 551.25\,\text{Hz}$). The correct frequency should be around 554.365 Hz, assuming that the flute has been tuned and played correctly.

Note also that at every multiples of 80 samples or so there is a pretty large dip that occurs in the mean power plot (bottom of Fig. 2.13). This is expected as the flute is (quasi) harmonic and the filtering of the signal with the inverse comb-filter will result in dips at multiples of the fundamental frequency. Furthermore, starting at around the 40 sample point, we observe large peaks at multiples of around 80 samples for the mean power vs. delay plot as well. This is again what we should expect as the inverse comb-filter not only attenuates harmonics but also amplifies harmonics when the notch frequencies are between harmonics. Hence, if we had used the forward comb-filter instead, we could have achieved the same results by searching for the mean power of the filtered signal that rendered the maximum mean energy opposed to the minimum mean energy as depicted in Fig. 2.14.

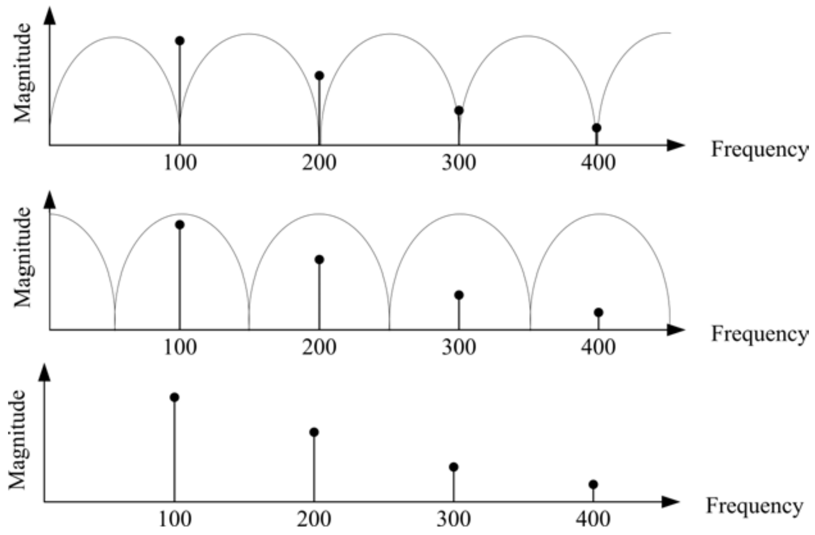


Fig. 2.14. Inverse comb-filter notching out the harmonics (top), inverse-comb filter amplifying harmonics (middle), original harmonics (bottom).

2.4.2 *Cepstrum analysis*

The *cepstrum* method for pitch analysis is also based on the DFT or rather the inverse-DFT of the log of the DFT to be more accurate. The name itself is a wordplay on spectrum coined by Bogert, Healy, and Tukey by reversing the first 4 letters yielding cepstrum. A bunch of other familiar terms were used by Bogert such as *liftering*, *rahmonics*, and *quefrency*. Bogert explains that in general "... we find ourselves operating on the frequency side in ways customary on the time side and vice versa" (Bogert *et al.* 1964) and hence the shuffling of letters. Cepstrum was initially applied for the detection of echoes and has found wide usage in acoustics, speech, and other areas in signal processing. The algorithm itself is quite straightforward and is shown in Eq. (2.6).

$$\hat{s}[n] = DFT^{-1}\{\log DFT(x[n])\} \tag{2.6}$$

$\hat{s}[n]$ is the real cepstrum component of the inverse DFT having the units of quefrency — corresponding to the unit of time, which can then be analyzed for peaks rendering the fundamental frequency.

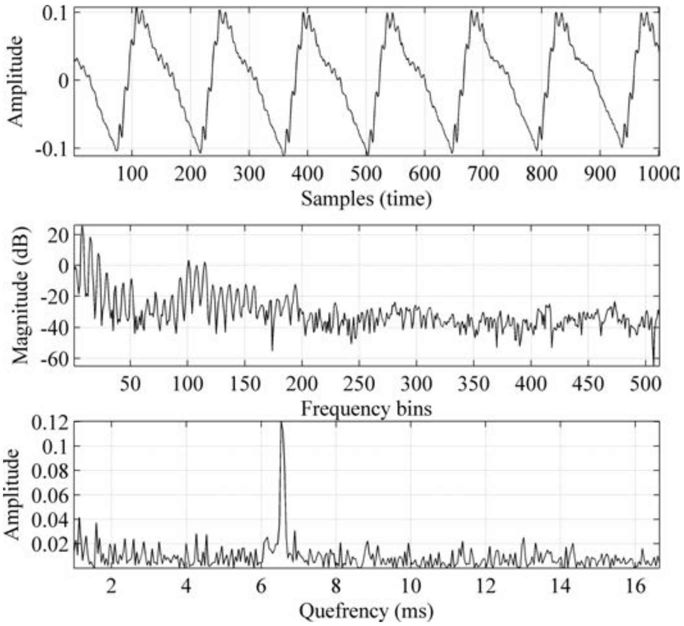


Fig. 2.15. Singing voice. Time sequence (top), power spectrum (middle), cepstrum (bottom).

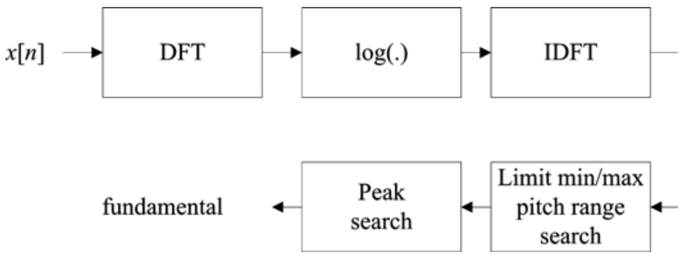


Fig. 2.16. Fundamental frequency computation flowchart via the cepstrum.

An example is shown in Fig. 2.15 where we have used a short excerpt of a singing voice. The top figure shows the time sequence, middle figure the power spectrum in dB, and the bottom figure the cepstrum. The fundamental frequency is commonly computed by finding the maximum peak for a given region in the cepstrum — that is, limiting the search to a minimum and maximum frequency range for pitch. For example, for the singing voice's cepstrum, the minimum frequency was set to 60 Hz and maximum frequency to 5 kHz resulting in a fundamental frequency of 179.27 Hz. The overall process is shown in Fig. 2.16.

2.4.3 *Harmonic product spectrum*

The harmonic product spectrum method (Schroeder 1968) is also a Fourier transform-based pitch detection algorithm. This algorithm also takes advantage of the tendency of pitched musical signals to demonstrate strong harmonic structures. The algorithm works by down-sampling spectra and multiplying the resulting spectra according to Fig. 2.17.

The down-sampling process in the frequency-domain essentially pulls the harmonics towards the DC essentially lining up the fundamental and the subsequent harmonics (if present) — down-sampling by 2 lines up harmonic 2 (octave) with the fundamental, down-sampling by 3 lines up harmonic 3 with the fundamental, ... etc. When the spectra are multiplied together, a strong peak will develop at the fundamental frequency location. In an ideal system, where, say, all the harmonics are at ideal harmonic locations (integer multiples of the fundamental) with an amplitude of 1 and all other areas 0, the situation would be like performing a Boolean AND on the spectrum, resulting in one single peak at the fundamental frequency location only (all other areas will be equal to zero) — the fundamental frequency is the only point where it is not at least once multiplied by zero as shown in Fig. 2.18. In a real-world spectrum example, the harmonics

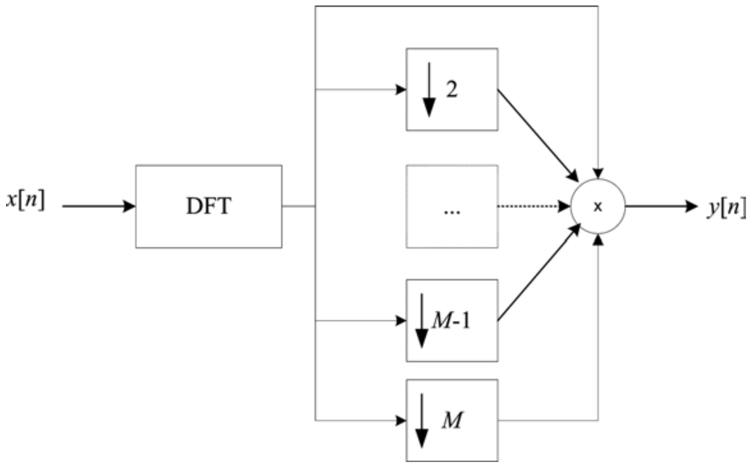


Fig. 2.17. Harmonic product spectrum algorithm and down-sampling.

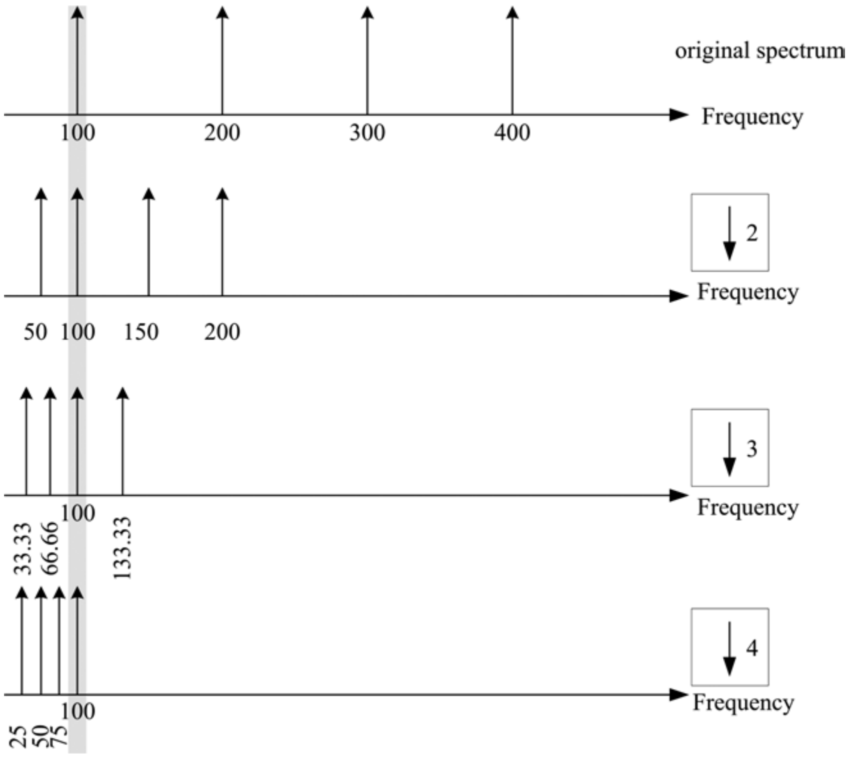


Fig. 2.18. Effect of down-sampling of spectrum.

and fundamental do not have the same magnitude values. Furthermore, and more importantly, areas where there are no harmonics are not equal to zero. However, the general concept still applies to musical signals as the harmonic locations have quasi-integer ratios and the non-harmonic regions are often significantly smaller than that of harmonics. Thus, in general, finding the largest peak reflecting the product of the shifted spectra would mean finding the fundamental frequency.

3 Vocoders (Voice Coders)

Speech and the voice have always been and will probably always be very important for musical composition, performance, and music research. Thus, it is maybe not such an astonishing fact that research in electronically generating speech and the singing voice has received much attention. The *vocoder*, which stands for voice coder-decoder or voice codec (coder-decoder), as it is used in the speech engineering community, was brought to the limelight by Homer Dudley in the late 1920s when he worked at Bells Labs in New Jersey, USA (Dudley 1929). A vocoder ordinarily consists of a voice encoder part and a voice decoder part. The encoder part analyzes the speech signal and the decoder utilizes the analyzed information to re-synthesize speech. The research surrounding the voice began as a way to synthesize voice for communication purposes, with the earliest demonstrations seen at the 1939 World's Fair in New York and San Francisco through a system called the *voder* (voice operation demonstrator, not to be confused with *Vader*, *Darth*). The voder was operated via a keyboard interface exciting a bunch of band-pass filters via noise and pulse-like signals to render an artificial human voice sound (Flangan 1972). These band-pass filters essentially mimicked the so-called *formant* structures of the voice — resonant frequency structures largely reflecting the behavior of the vocal tract and the timbral signature of individual voices. In this section, we will introduce a number of important vocoders and examine the basic mechanics behind each of the algorithms.

3.1 *Channel-vocoder*

The *channel vocoder* gets its name from the group of band-pass filters it uses in implementing a speech analyzer/synthesizer — the input signal, such as the voice is divided into N frequency channels via N band-pass

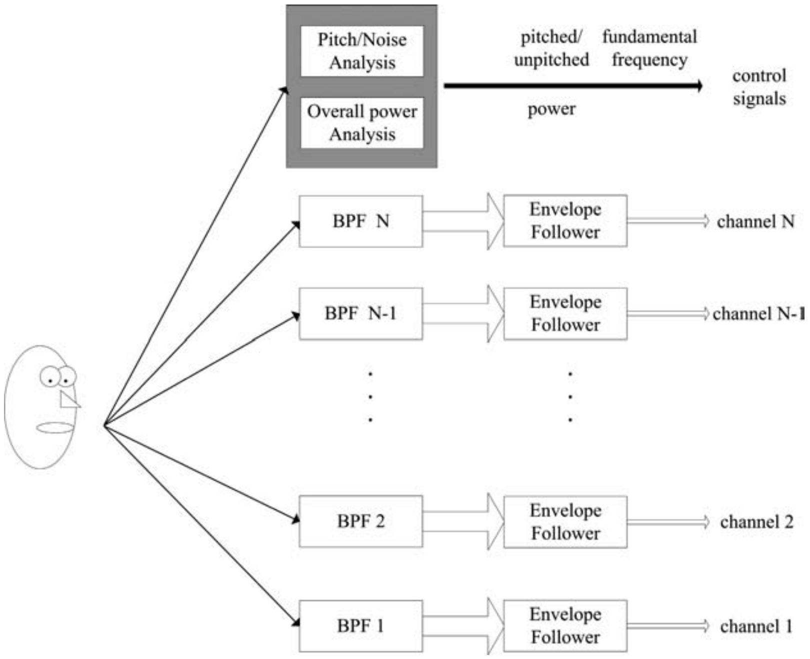


Fig. 3.1. Channel vocoder: encoder section.

filters as seen in Fig. 3.1 much like an equalizer in our home and car stereo system. The idea behind the exploitation of the band-pass filters is that at the encoder part, the voice is divided into N frequency channels or bands which are in turn individually transmitted to the decoder and synthesized by summing the N band-passed channels from the speech signal. The individual channels by themselves do not amount to much and are unintelligible (depending on the number of bands). However, when the output signals of each of those channels are combined, they form a clear speech signal akin to the original. As mentioned in the introduction of this section, the concept was developed by Homer Dudely (Dudley 1929) who initially used 10 banks of band-pass filters (250 Hz to 3 kHz) to build his channel vocoders. The channel vocoder was also used during World War II by Roosevelt and Churchill (Doyle 2000) in planning the *D-Day* invasion. They utilized the system as a way to communicate secretively by applying concepts borrowed from cryptography. This 007-like speech system's name, based on the channel vocoder was *SIGSALY* and the prototype was coined *green hornet* due to the buzzing sound the system made when synthesizing speech (you will see why the early vocoder used to make lots of buzzing sounds below).

3.1.1 *Filter banks, envelope followers, and the encoder*

Returning to our encoder part of the channel vocoder (Fig. 3.1), we can see N banks of filters each accompanied by envelope followers. After a speech signal is divided into N bands, each band is subjected to an amplitude envelope follower implemented via full-wave rectification and low-pass filtering. Full-wave rectification refers to making negative values positive, which can be digitally implemented by just taking the absolute value of each input sample. Equation (3.1) shows an example of a one-pole low-pass filter.

$$y[n] = (1 - a) \cdot x[n] + a \cdot y[n - 1] \tag{3.1}$$

As can be observed in the difference equation, the coefficient a $(0.0 < a < 1.0)$ acts as a weight distributor, where more weight given to the input component decreases the emphasis given to the feedback component and vice-versa.

Figure 3.2 shows the frequency response of the envelope follower of Eq. (3.1) for various coefficients values a. The envelope follower is sometimes

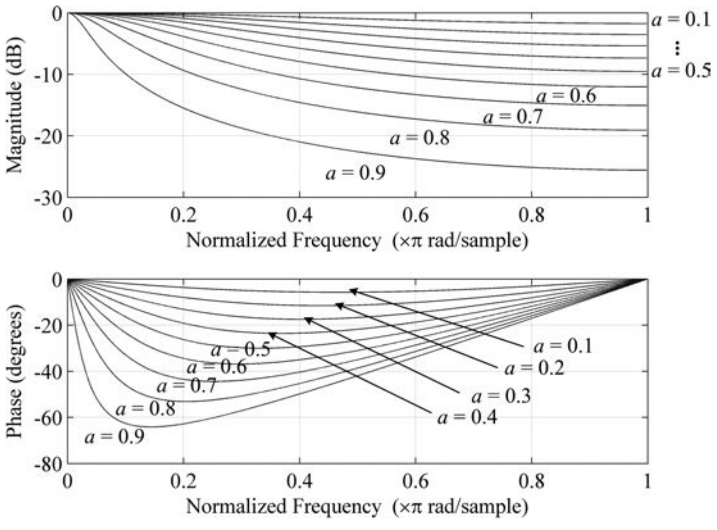


Fig. 3.2. Envelope follower and low-pass filter frequency response for $a = 0.1$ to 0.9.

enhanced by using different coefficient values for rising and falling parts of a signal. That is, for rising portions of a signal, we want the envelope to respond quickly with less low-pass filtering. For falling parts, we want to produce a softer landing, making the filter output decay slower than the attack portion with more low-pass filtering.

Thus, for each channel, we now have the energy levels as a function of time represented by the envelope follower — this is sent to the receiver. The overall energy level of the signal can be computed via RMS as we did when computing the amplitude envelope in Chap. 2, Sec. 3.2.

$$RMS = \sqrt{\frac{1}{L}\sum_{n=0}^{L-1} x^2[n]} \tag{3.2}$$

As usual, when computing the RMS, we will need to window a portion of a signal [L in Eq. (3.2)] and set an appropriate hop size much like the STFT (Chap. 8, Sec. 5) and the RMS amplitude envelope (Chap. 2, Sec. 4). Figure 3.3 shows the frequency response of 24 filter banks computed using the MATLAB® butterworth-design band-pass filters, where the

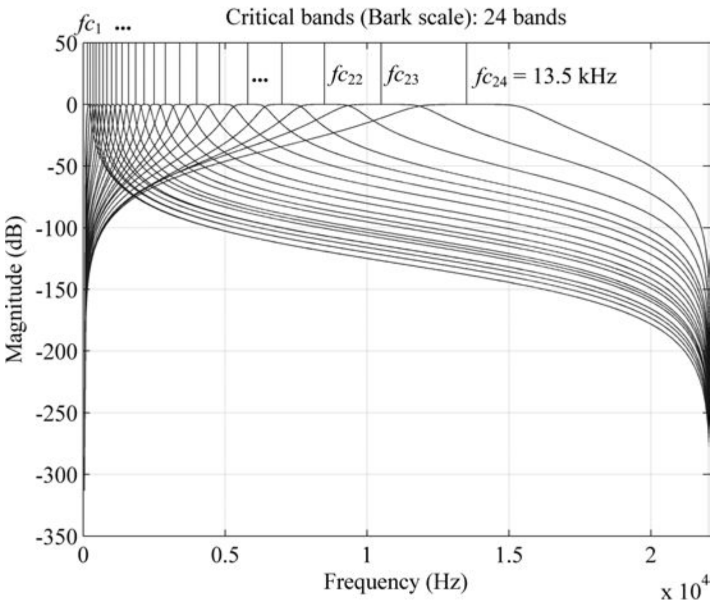


Fig. 3.3. Bark scale filter banks of 24 bands.

center frequencies of the bands are in accordance with perceptual frequency divisions of the *Bark* scale (see appendix for more details). We could just as well have also used linear division for the filter banks. However, as our hearing system is nonlinear (both in frequency and amplitude), nonlinear scales such as the Bark scale are more appropriate when band-pass filter banks are used in audio applications (Zwicker and Hastl 1999). The locations of the center frequencies adhere to what are called *critical bands*, where the Bark scale is one of such scales conforming to perceptual tendencies of frequency perceived by humans. Critical bands basically denote the bandwidth (via lower and upper frequency bounds) for a given center frequency that activates the same area on the basilar membrane (sheet of fibers in the inner ear inside the cochlea) in response to an incoming sine wave at that center frequency. For example, when a sine wave is generated at 45 Hz, according to the Bark scale, the perception of other sine tones between 49 Hz and 51 Hz will not be distinguishable from the 45 Hz signal given that intensity levels are the same. This bandwidth changes with the center frequency and widens as we go up the frequency axis.

In Fig. 3.4 we can see the envelope follower in action. The top plot shows the input voice's waveform and the subsequent plots the envelope follower output for a number of critical bands from a total of 24 bands. Looking at the plots we spot that each filtered band has different amounts of energy at different points in time. Only when all bands are recombined in the decoder part do we get the fully de-fragmented version of the synthesized voice signal.

3.1.2 *Voiced and unvoiced analysis and the decoder*

Before we proceed to the decoder part of the channel vocoder, we note that we still have not figured out what the pitched/noise analysis part does in the encoder. When we speak and talk to one another, it may perhaps be surprising to find out that the produced speech is a mixture of pitched and un-pitched signals. Consonants or *unvoiced* speech segments resemble noise-like signals, whereas vowels in speech show strong periodic behavior and hence appear as pitched or *voiced.* Try to sing in any pitch range that you are comfortable at using the letter "s" or "f." Now try it with the letter "a" or "e." You will have discovered that with the first two letters, it is literally impossible to hold a pitch, whereas for the latter two, singing a pitched "a" or "e" is definitely doable. This may be quite remarkable

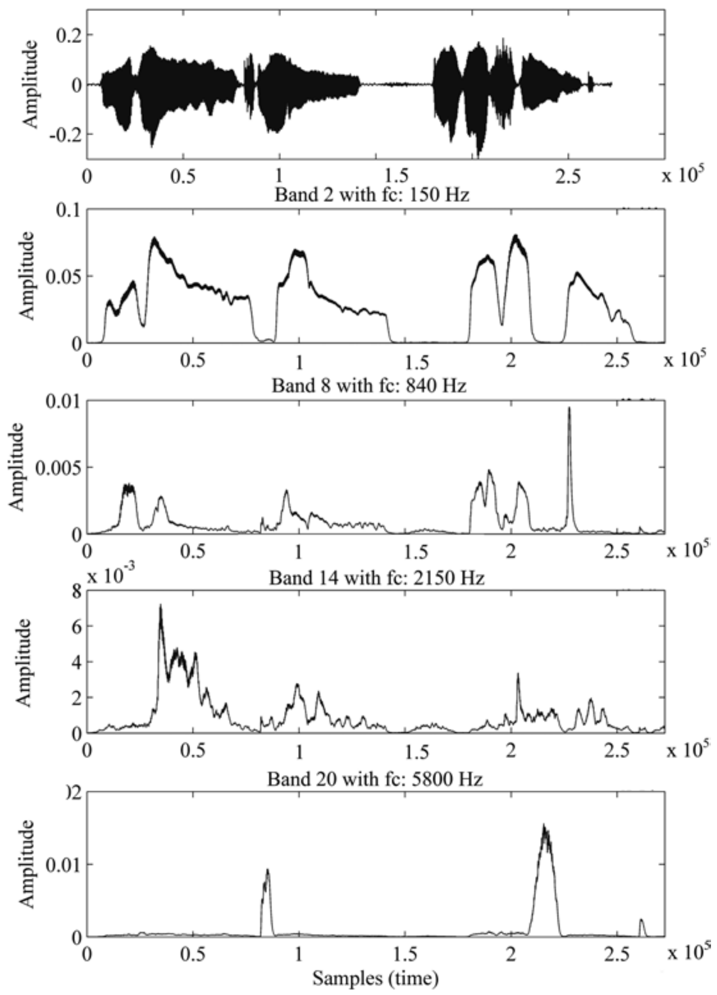


Fig. 3.4. Envelope follower for 4 bands. Full waveform (top), $f_c = 150\,\text{Hz}$ (2nd), $f_c = 840\,\text{Hz}$ (3rd), $f_c = 2.15\,\text{kHz}$ (3rd), $f_c = 5.8\,\text{kHz}\,\text{Hz}$ (bottom).

for those of us who were not aware of this interesting phenomenon as this idiosyncratic feature of the voice basically makes speech a potentially very musical signal by default.

Reeling back to the channel vocoder, we now know what the pitch/noise analysis block in the encoder relates to. So far so good, but why do we need to know whether a speech signal is voiced or unvoiced? The reason for this is because the channel vocoder adheres to a particular synthesis model called the *source-filter* model. The source in our case is the voiced/unvoiced signal and the band-pass filters correspond to the filter part of the source-filter model architecture. Thus, in such a system, the source excites the filter banks at the decoder side (and sometimes even in the encoder part — this is called analysis-by-synthesis) during the speech synthesis stage. Depending on the source type (voiced or unvoiced), a switch will determine whether to generate a pulse train or noise signal. In a way, the decoder has the easy end of the problem and exploits this source-filter paradigm of the human speech system — it uses the voiced/unvoiced flag along with the amplitude envelopes to drive each filter with fundamental frequency information received from the transmitter/encoder. When the analyzed signal is unvoiced, noise is inputted into each filter bank and when it is voiced, a pulse train is generated for the band-pass filters. The output of each filter is then summed rendering our synthetic speech signal. This idea is summarized in Fig. 3.5.

Also, note that in both in Figs. 3.1 and 3.5 there is no actual audio signal transmitted to the decoder. The signals sent to the decoder only consist of the envelopes characterizing the energy level in each band, *control signals* that tell the decoder whether it is voiced/unvoiced, and the overall energy for a given portion of a window. The reason why the

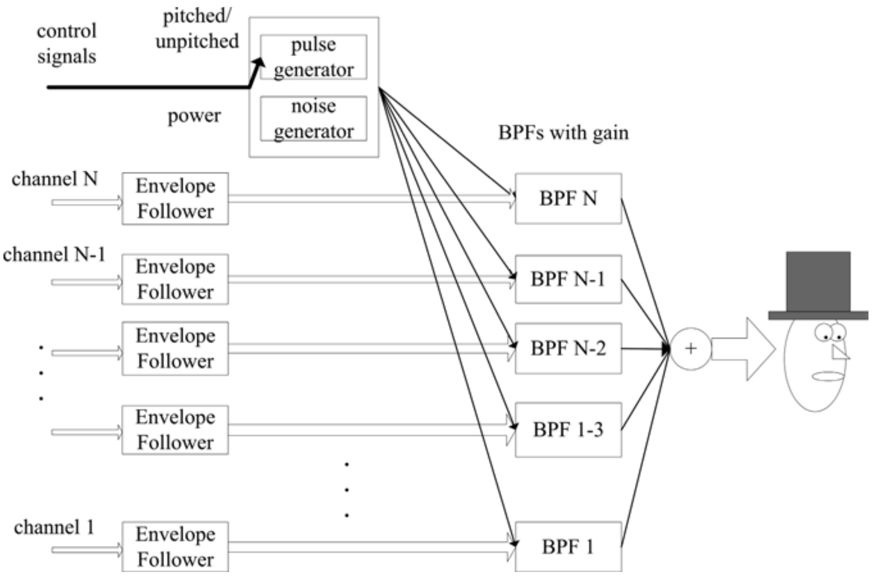


Fig. 3.5. Channel vocoder: decoder side.

vocoder goes through so much effort on the encoder side is to transmit a bandwidth limited representation of the original signal for synthesis. The goal is to reduce the amount of information that needs to be transmitted. Vocoders are used primarily in voice communication scenarios and cell-phone technology areas relying heavily on techniques to compress the voice data and only send data that is absolutely necessary to minimize the data amount being sent.

3.1.3 *Voiced and unvoiced decision-making*

Stepping back for a moment, we realize that we have not yet discussed how to actually analyze a signal for its voiced and unvoiced quality. We will therefore present some of the common voiced/unvoiced analysis algorithms in this section starting with the simplest one — zero-crossing.

3.1.3.1 Zero-crossing analysis

Our acquaintance with zero-crossing analysis takes us back to the early days of Chap. 2, Sec. 5.1, where we used it for fundamental frequency detection. The same algorithm can also be used to determine whether a signal is voiced or unvoiced. The concept of zero-crossing itself is clear-cut as we have learned in Chap. 2 and when combined with our knowledge of audio signals, the zero-crossing algorithm also becomes very useful in meeting our needs in this particular problem. As we know, noisy signals (unvoiced) are all over the place, especially when viewed as a waveform — there is ideally no regularity or patterns to discern. Periodic signals (voiced), on the other hand, are characterized by repetitive patterns and hence exhibit cyclic tendencies as seen in Fig. 3.6. Consequently, the zero-crossing rate for unvoiced signals tends to be "pretty high" whereas for voiced signals "pretty low." Now, saying something is low and high really is like saying something is warm and cold, which does not mean much unless we label it with some hard numbers. But even when we do have hard numbers blinking on our screen, it is impossible to determine with absolute certainty if a signal is voiced or unvoiced. However, there tends to be a range of zero-crossings that suggest unvoiced vs. voiced characteristics as shown in Fig. 3.7.

In Fig. 3.7, a window size of 10 ms was used in plotting a histogram of the average zero-crossing rate (ZCR) for both voiced and unvoiced speech signals (Rabiner and Schafer 1978). Although we can vividly make out some amount of overlap, the Gaussian average of the zero-crossing rate for unvoiced signals is around 49 per 10 ms interval vs. 14 for voiced signals

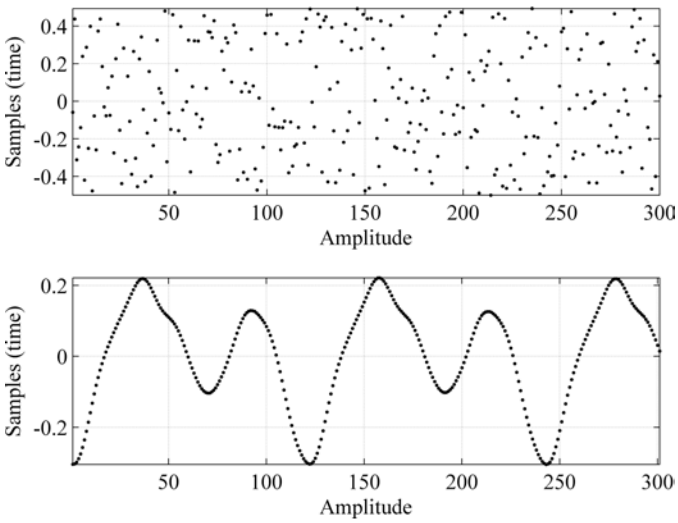


Fig. 3.6. Noisy signal (top) vs. periodic signal (bottom).

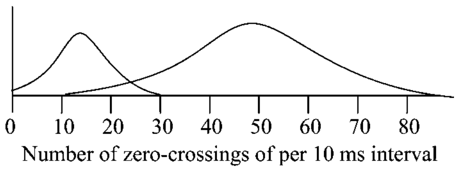


Fig. 3.7. Distribution of ZCR for voiced vs. unvoiced signals (after Rabiner and Schafer 1978).

for the same 10 ms interval. Figure 3.8 shows the waveform and the zero-crossings using a window of 10 ms singing "Love me tender, love me sweet." In this example, we can discern without much difficulty where the noisy parts are: in the "t" part of "tender" (at around 40,000 samples), at the short pause (approx. 72,000 samples), at the "s" of "sweet" (approx. 104,000 samples), and at the end of the signal.

Figure 3.9 shows the actual number of zero-crossings throughout the duration of the waveform. Figure 3.10 shows part of the waveform from around 40,000 samples where the "t" in "tender" is sung. We can vividly discern that the two regions show the stereotypical unvoiced vs. voiced waveform characteristics — the first part is noise-like displaying no particular pattern and the second part, where there are low zero-crossings, is patterned and quasi-periodic.

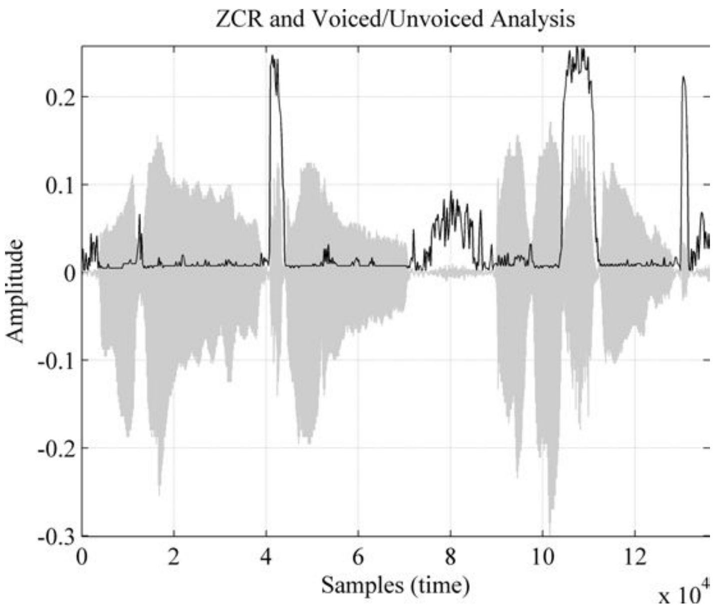


Fig. 3.8. Waveform of singing voice imposed on contour of zero-crossings.

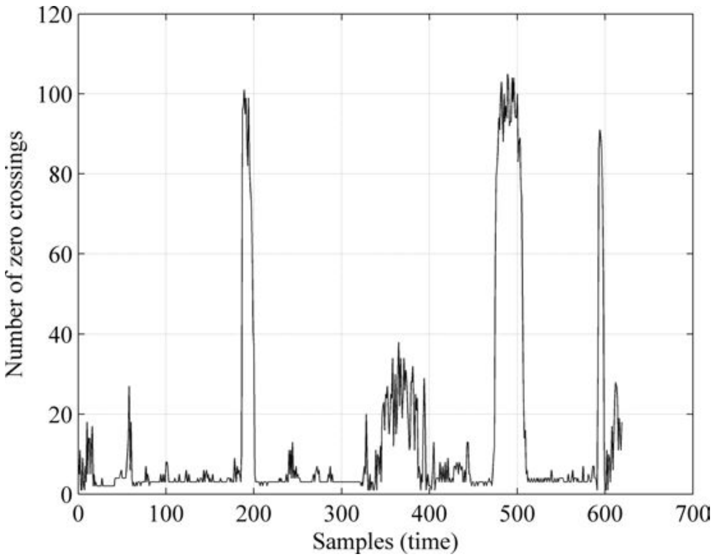


Fig. 3.9. Plot of actual zero-crossings of waveform using 10 ms windows.

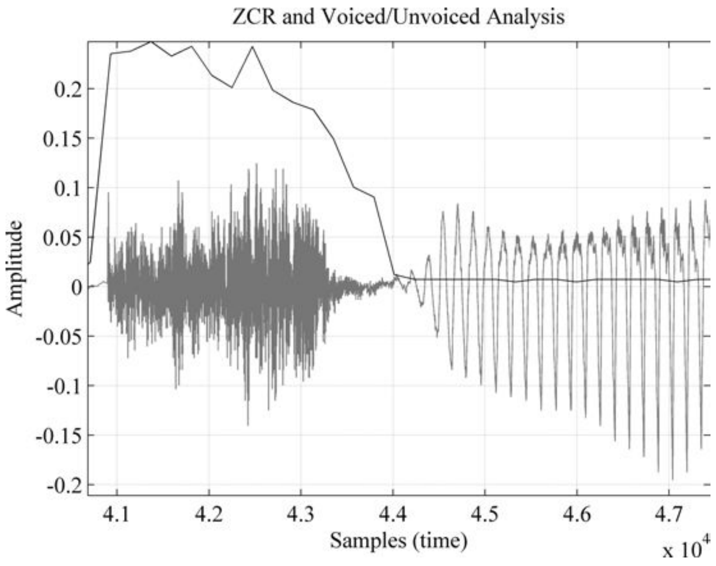


Fig. 3.10. Zoom-in of waveform: "t-en(der)"

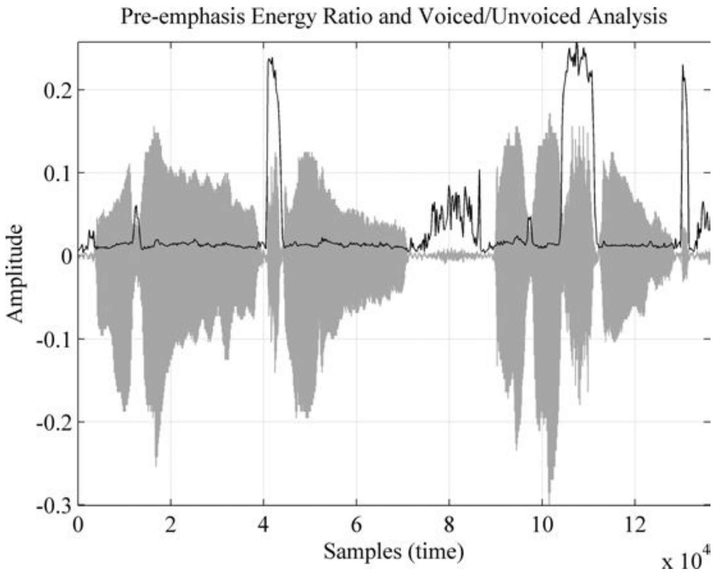


Fig. 3.11. Analysis using the pre-emphasis energy ratio method.

3.1.3.2 Pre-emphasized energy ratio

In the *pre-emphasized energy ratio* (Knodoz 2004) method for voiced and unvoiced speech analysis, the focus is on the variance of the difference of the amplitude values between adjacent samples. The variance of the difference has a tendency to be lower for voiced portions compared to unvoiced regions of a signal — this is in line with Fig. 3.6, which shows the gradual and smooth change between samples for voiced signals and the random quality typical of unvoiced signals. The algorithm is shown in Eq. (3.3). The reason it is called the pre-emphasized energy ration is due to the pre-emphasis high-pass filter in the numerator part [see Eq. (2.5)].

$$\mathrm{Pr} = \frac{\sum_{n=1}^{n=N} |x[n] - x[n-1]|}{\sum_{n=1}^{n=N} |x[n]|} \tag{3.3}$$

Figure 3.11 shows the algorithm's performance using the same singing voice sample and window length of 10 ms.

3.1.3.3 Low-band to full-band energy ratio

In speech, voiced signals tend to have energy concentrated in the lower frequency ranges, whereas unvoiced signals throughout the frequency range. This is not the case just for speech signals but also for musical signals, especially acoustic instrumental sounds as we have already seen throughout this chapter and previous chapters — the magnitude of harmonics tend to decay with the increase in frequency towards the Nyquist limit. The *low-band to full-band energy ratio* algorithm computes the ratio of energy by setting the numerator to the low-pass filtered signal and the denominator to the original unaltered signal according to Eq. (3.4). When the input signal x is voiced, x and x_{LPF} will be more or less the same, meaning that the ratio in Eq. (3.4) will be close to 1. This is due to the fact that low-pass filtering will do very little to the numerator part of Eq. (3.4) since it does not contain much high frequency energy. However, if it is unvoiced, the input x will exhibit a flatter spectrum (noise is ideally flat) and thus x_{LPF}will yield a lower energy level. In summary, LF will render a low value for unvoiced regions of a speech signal and high value for voiced signals.

$$LF = \frac{\sum_{n=1}^{n=N} x_{LPF}^2[n]}{\sum_{n=1}^{n=N} x^2[n]} \tag{3.4}$$

Figure 3.12 shows the performance of the low-band to full-band energy ratio algorithm again using the same voice signal from our previous examples

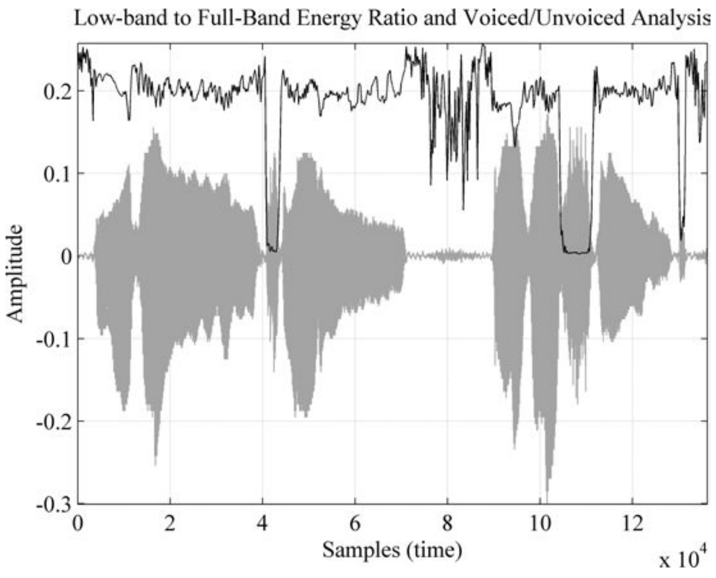


Fig. 3.12. Analysis using the low-band full-band energy ratio method.

with a window size of 10 ms as before. Note that in this case, the unvoiced parts have low energy and the voiced parts high energy.

3.1.3.4 Spectral flatness measure

The *spectral flatness measure* (SFM) can also be used to determine voiced and unvoiced parts of a signal and is computed via the ratio of the geometric mean (Gm) vs. the arithmetic mean (Am) of the DFTs of a signal as shown in Eq. (3.5). Signals that are sinusoidal result in lower measurements of SFM (approaching 0) whereas signals exhibiting more noise-like characteristics (adhere to flatter and de-correlated spectra) cause the SFM to approach 1.

$$SFM_{dB} = \frac{Gm}{Am} = \frac{\left(\prod_{k=0}^{k=N-1} X^d(k)\right)^{\frac{1}{N}}}{\frac{1}{N}\sum_{k=0}^{N-1} X^d(k)} \tag{3.5}$$

$$\prod_{k=0}^{k=N-1} X^d(k) = X^d(0) \cdot X^d(1) \cdot \ldots \cdot X^d(N-2) \cdot X^d(N-1) \tag{3.6}$$

Note that in the geometric mean we use multiplication and $1/N$th power opposed to summation and division used in the arithmetic mean. Due to the multiplication process in the geometric mean, one should watch out for any zeros in the data set as any zero, no matter how large the other numbers are, will cause the product to produce 0 and hence result in a zero geometric mean. Also, note that the arithmetic mean is always greater or equal to the geometric mean due to Eq. (3.7), where the a_N coefficients are real positive numbers. For unvoiced signals, the geometric mean and arithmetic mean will yield similar results causing SFM to approach 1, whereas for voiced signals the geometric mean will become smaller resulting in reduced SFM values.

$$\frac{(a_1 + a_2 + \cdots + a_N - 1 + a_N)}{N} \geq (a_1 \cdot a_2 \cdot \ldots \cdot a_{N-1} \cdot a_N)^{1/N} \tag{3.7}$$

When using logarithmic identities (multiplication is equivalent to summation and power is equivalent to multiplication in the log world), the *Gm* can be expressed as Eq. (3.8). Equation (3.8) is thus sometimes referred to as the log-average as that is what is being computed before the exponential operator brings it to the original scale.

$$\left[\prod_{k=0}^{k=N-1} X^d(k)\right]^{1/N} = \exp\left(\frac{1}{N}\sum_{k=0}^{k=N-1} \ln(X^d[k]\right) \tag{3.8}$$

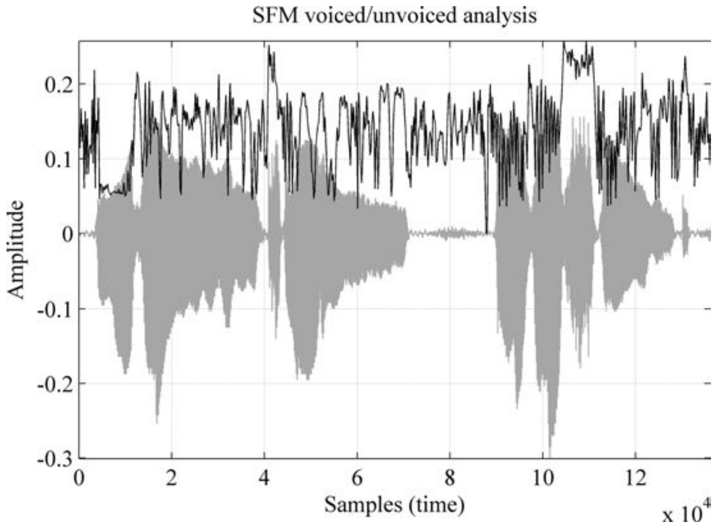


Fig. 3.13. SFM analysis for voicing determination using window size of 10 ms.

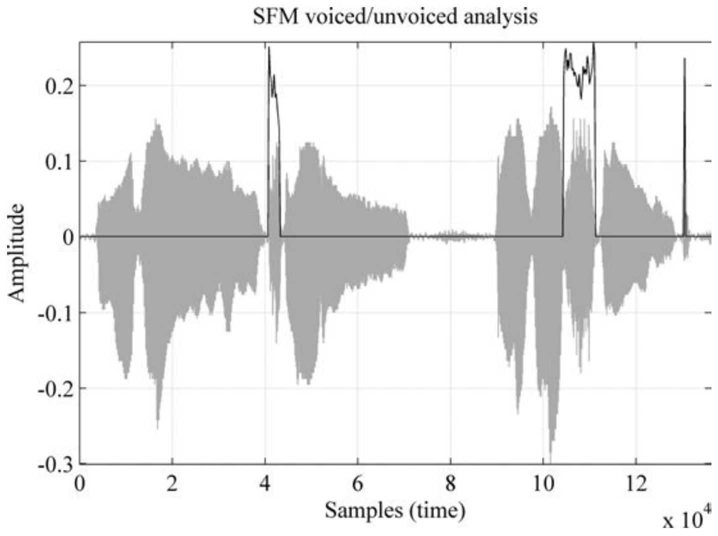


Fig. 3.14. SFM analysis for voicing determination using window size of 40 ms.

Figure 3.13 shows the results of the SFM algorithm using a 10 ms second window as before without overlapping STFT frames. We can see that the SFM seems to a bit more subtle, offering details with a shorter window size, which in turn translates to loss of frequency resolution. However, when we increase the window size (increase frequency resolution) while keeping the hop size unchanged, the SFM plot becomes more vivid as shown in Fig. 3.14. The parameters used in Fig. 3.14 include a window size of 11.61 ms (512 samples) and STFT with 10 ms overlap (220 samples) at $f_s = 44.1\,\text{kHz}$.

3.2 *Linear predictive coding (LPC)*

The next type of vocoder we will present in this chapter is the *linear predictive coding* (LPC) vocoder pioneered by Bishnu S. Atal at Bells Labs while working with Max Mathews (inventor of the first computer music language) in the mid/late 1960s. Early on, LPC was called *adaptive predictive coding* but soon linear predictive coding took over which has stuck ever since. Like the channel vocoder, LPC originates from the speech research literature and has found much popularity in the computer music community especially by composers Charles Dodge and Paul Lansky who have extensively applied it in their early computer music works.

The theory itself can simply be thought as a signal analysis/synthesis algorithm predicting the current sample $x[n]$, via P number of past samples when appropriately weighted in a FIR filter configuration according to Eq. (3.9) — a linear combination of past input samples.

$$\begin{aligned}\hat{x}[n] &= a_1 \cdot x[n-1] + a_2 \cdot x[n-2] + \cdots + a_p \cdot x[n-P] \\ &= \sum_{k=1}^{k=P} a_k \cdot x[n-k] \end{aligned} \tag{3.9}$$

$\hat{x}[n]$ denotes the estimated sample, $x[n-k]$ the delayed past samples, and a_k the weights of past samples. The objective is to try to find the P number of coefficients a_k, which will make the predicted current sample $\hat{x}[n]$ and the actual sample $x[n]$ as close as possible. Thus, in trying to minimize the error $e[n]$ between $\hat{x}[n]$ and $x[n]$, the optimum weights are computed which are referred to as the LPC coefficients as shown below.

$$e[n] = x[n] - \hat{x}[n] = x[n] - \sum_{k=1}^{k=p} a_k \cdot x[n-k] \tag{3.10}$$

This somewhat makes intuitively sense as it is in a way learning from the past (history) to predict the future so-to-speak (although we do seem to have knack for not learning from past mistakes!). The LPC method is also sometimes referred to as the *short-term prediction filter* as it is used to predict short-term trends ($20-30$ ms) and is inadequate in predicting long-term trends (known as *long-term prediction* such as pitch).

As we can garner from Eq. (3.9), the filter itself is FIR-based and the z-transform of the error $e[n]$ results (commonly known as the *residual* signal) in Eqs. (3.11) and (3.12) with the transfer function denoted as $A(z)$.

$$\begin{aligned} E(z) &= X(z) - \hat{X}(z) \\ &= X(z) - X(z) \cdot \sum_{k=1}^{k=p} a_k \cdot z^{-k} \\ &= X(z) \cdot \left(1 - \sum_{k=1}^{k=P} a_k \cdot z^{-k}\right) \end{aligned} \tag{3.11}$$

$$A(z) = 1 - \sum_{k=1}^{k=P} \alpha_k \cdot z^{-k} \tag{3.12}$$

We can thus represent the input part $X(z)$ as a function of the transfer function $A(z)$ and $E(z)$ as in Eq. (3.13).

$$X(z) = \frac{E(z)}{A(z)} \tag{3.13}$$

The LPC analysis part is most powerful when coupled with the LPC synthesis counterpart whereby we can think of this configuration as an analysis-by-synthesis structure. We know that the analysis part is an all-zero filter since it is an FIR filter, however, if we take the FIR coefficients and put it on its head, the FIR filter becomes an IIR filter — an all-pole filter as shown in Eq. (3.14).

$$\frac{1}{A(z)} = \frac{1}{1 - \sum_{k=1}^{k=P} \alpha_k \cdot z^{-k}} \tag{3.14}$$

This all-pole filter is essentially a resonant filter describing the vocal tract. So what (3.13) actually says is that if we know the error $E(z)$ and the filter $A(z)$ we can compute $X(z)$ and hence reproduce the input signal $x[n]$ by subjecting the error signal to the transfer function $1/A(z)$. Stated differently, when filter $1/A(z)$ is excited by the residual signal $e[n]$ (constituting a source-filter model as in the channel vocoder) we get $x[n]$ back.

The vocal tract itself can be compared to a conglomerate of small sections of tubes contributing to the way our voice sounds and the LPC model is very effective in describing these tubes and the resulting *formant* structure of human speech sounds. Formants are resonant frequency locations typically ranging from 3 to 5 for human voice sounds which are distinct in location along the frequency spectrum (sort of like poles in a tent giving the tent its particular shape — the tent's formant make-up). Due to the tube-model paradigm inherent in LPC systems, some musical instruments exhibit stronger formant characteristics than others. For example, woodwind instruments such as the oboe and clarinet generally show one or two pronounced formant regions as they are more or less generally shaped like tubes.

The source-filter model of the LPC system is shown in Figs. 3.15 and 3.16. The decoder is configured in terms of the excitation signal generator (voice/unvoiced) plus the filter model and the encoder takes on the role for analyzing and transmitting all the relevant information to the decoder for synthesis. As we can see, the global structure is not that different from the channel vocoder. The excitation signal provided to the

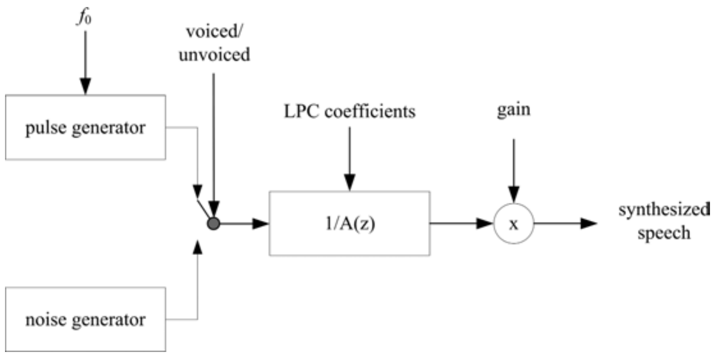


Fig. 3.15. Synthesis part of LPC.

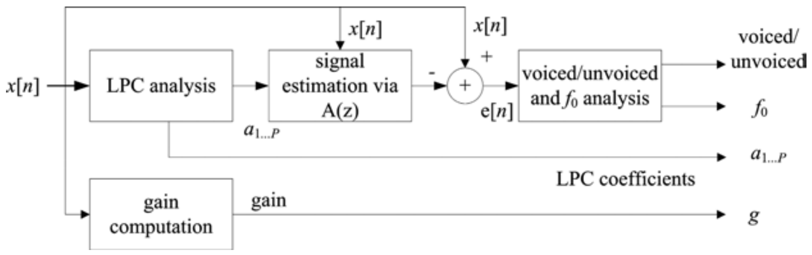


Fig. 3.16. Analysis part of LPC system.

all-pole filter is either a pulse generator or noise generator for voiced and unvoiced signals respectively. As mentioned in the previous section, the reason for this toggle between pulse and noise is due to the fact that when it is voiced (vowel sounds for example), pitch is present, and when unvoiced (consonants for example), the excitation (error $e[n]$) basically becomes a flat and random noise signal.

Thus, if the analysis part can determine the fundamental frequency information by analyzing the residual signal $e[n]$ using autorcorrelation for example, we can excite the all-pole filter in the synthesis part with a pulse train at the computed fundamental frequency. If it is not pitched, we use a noise signal to drive the filter. The voiced/unvoiced decision is made by employing algorithms discussed in Sec. 3.1.3. The gain g is computed using methods such as RMS energy for a windowed portion of the signal being analyzed. The gain is selected so that the output (synthesized) and input (original) energy is the same for a given period of time. This pulse/noise and filter coefficient setup is used for low bit rate voice codes (coder-decoder) often found in cell phone systems such as GSM. Although the overall structure of vocoders is not that different, one of the most intriguing distinctions is that in the LPC encoder, the input signal itself is synthesized and used in the encoding process. Thus, the encoder part actually does both analysis and synthesis and is hence referred to as an *AbS* (analysis-by-synthesis) system. Figure 3.16 is, however, not entirely accurate for common LPC vocoders, as in real-life applications, only part of the residual signal is sent over to the decoder for re-synthesis. Vocoders that do not send the residual signal over the network at all and hence save a considerable chunk of bandwidth are referred to as *codebook vocoders* — *codebook excited linear predication* (CELP) being one example. Because sending the residual signal (even if it is only a portion of the original) is expensive from a data compression point of view, CELP codecs analyze the residual signal on the encoder side and send an index to a *codebook*, rather than the actual residual signal. The codebook is locally available to both the encoder and decoder and contains a bunch of residual signals which are used to excite the all-pole filter during the re-synthesis stage at the decoder upon receipt of the best fitting residue index. Thus, by sending an index number opposed to the actual residual signal pays high dividends in data compression. Another block we have omitted in Fig. 3.16 is the pre-emphasis filter before analysis in the encoder part and the *de-emphasis* filter after re-synthesis in the decoder part. We already encountered the pre-emphasis filter in Sec. 2.4.1 introduced as the simplest FIR high-pass filter (2.5). The reason it is used before LPC analysis lies in the fact that the computation of LPC coefficients is more robust when the spectrum under consideration is more evenly balanced on the frequency axis (speech and musical signals tend to have concentration of energy in the lower frequency regions). The pre-emphasis filter helps in distributing the energy evenly across the spectrum through high-pass filtering. Thus, to bring the signal back to its originally "biased" spectral tilt, a de-emphasis filter is applied just before the synthesized signal is sent out to the world. The de-emphasis filter is the simplest low-pass filter as shown in Eq. (3.15).

$$y[n] = x[n] - b_1 \cdot x[n-N] \tag{3.15}$$

Figure 3.17 shows the LPC analysis and synthesis results for an electric bass guitar signal. We can see that the estimated signal and the original signal are almost identical both in the time-domain and frequency-domain. This is further confirmed by the residual (error) signal shown at the top of Fig. 3.17. At the bottom of Fig. 3.17, the all-pole filter's frequency response

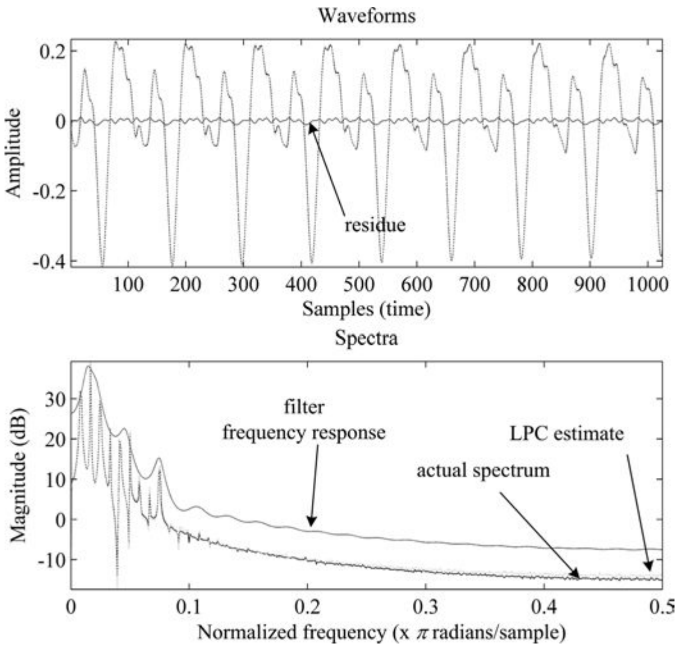


Fig. 3.17. Example of LPC in action for electric bass guitar signal.

is plotted along with the original signal's DFT and the estimated signal's DFT which are again so close that they are nearly indistinguishable in this example. Now, if we were to use the LPC as a vocoder in terms of the encoder/decoder configuration, voiced/unvoiced block, and fundamental frequency detection to determine the excitation signal for the all-pole filter in the decoder, the musical prospects are indeed very exciting. We are well armed at this point to tackle such a mini-project and will leave it for another session on a Sunday evening at the lab (a.k.a. known as the bedroom with a laptop and headphones so that we do not disturb the unsuspecting person next door with weird sounds we produce).

3.3 *LPC coefficient computation*

We will not go too deeply into the topic of LPC coefficient computation algorithms as they can be found readily in speech-based signal processing texts and cookbooks in numeric techniques. However, we will briefly touch upon the two most popular methods for computing the LPC coefficients $[a_1 \cdots a_P$ in Eq. (3.5)] — the *autocorrelation* and *covariance* methods.

The autocorrelation method gets its name as the algorithm for computing the LPC coefficients is based on none other than the autocorrelation vector of a speech frame. The autocorrelation vector is passed through the *Levinson-Durbin* algorithm which computes the LPC coefficients by recursively and efficiently computing inverse matrices of the autocorrelation vector. The algorithm requires windows that are tapered at the boundaries — having flanking ends starting and ending at 0 such as the hamming window. Hence, some information is lost during this process and frequency-domain distortion takes place but the algorithm has nevertheless found much success in obtaining the LPC coefficients and is particularly desirable as it guarantees filter stability. Remember that the synthesis filter is an all-pole filter and in order for it be stable the roots or the poles need to be within the unit circle — such a filter is referred to as a *minimum-phase* filter and has all its roots within the unit circle. The algorithm is summarized below in Eqs. (3.16) and (3.17) where $r[\cdot]$ is the autocorrelation vector and a_k the LPC coefficients.

$$r[\tau] = \sum_{n=0}^{n=N-1} x[n] \cdot x[n+\tau], \quad \tau \geq 0 \tag{3.16}$$

$$\sum_{k=1}^{k=P} a_k \cdot r[|k-m|] = -r[m], \quad m = 1, 2, \ldots, P-1, P \tag{3.17}$$

Unlike the autocorrelation method, the covariance method computes the LPC coefficients without applying an "explicit" window on the speech signal. The algorithm incorporates a pitch synchronous analysis approach where the analysis is conducted by starting at the beginnings of the signal's pitch cycle for voiced speech. The LPC update rate is higher for the covariance method compared to the autocorrelation method and the algorithm better adapts to changes in the vocal tract transfer function and is therefore generally more accurate. However, the covariance method does usually require more horsepower. The algorithm is summarized below in Eq. (3.18) and (3.19) where c_{km} is the covariance matrix. The *Choleski* (Rabiner and Schafer 1978) decomposition approach is often used in solving for the coefficients.

$$c_{km} = \sum_{n=P}^{n=N-1} x[n-k] \cdot x[n-m] \tag{3.18}$$

$$\sum_{k=1}^{k=P} a_k \cdot c_{km} = -c_{0m} \quad m = 1, 2, \ldots, P-1, P \tag{3.19}$$

For musical applications, due to the source-filter model configuration, we can change the fundamental frequency information to a desired frequency and/or change the excitation signal with something other than the original signal's residual signal (it can be anything!). Thus, viewed from a musician's perspective, the LPC system is a very powerful and flexible tool for audio applications not only limited to voice signals but other generic audio signals as well. The source-filter model is essentially the same in concept used in the talk box introduced in Chap. 7, Sec. 4. In the talk box, a source such as the electric guitar is fed via the mouth into the oral cavity, whereby the resonant structure of the oral cavity produces the talking guitar effect. We can certainly implement a basic talk box digitally by having a number of presets of LPC coefficients that reflect certain vocal sounds . . . potential musical fun indeed.

3.4 *The phase vocoder*

In a nutshell, the *phase vocoder* is one of the most powerful, if not the ultimate vocoder among its many peers and is the younger, more sophisticated, and flexible version of older sibling channel vocoder. The work has been pioneered by Flanagan and Golden (Flanagan and Golden 1966) while working at Bell Labs (as you can tell, there were some amazing folks at Bell Labs throughout the years) and although the initial research was targeted towards application in speech, it has found explosive popularity in the computer/electronic music world.

The phase vocoder is very similar to the channel vocoder as it is also built on the filter-bank idea but on a much larger scale. It does not, however, conform to the model of using an excitation signal at the decoder per se (voiced/unvoiced excitation) but rather uses the instantaneous amplitude information along with narrowly tuned decoder filter-banks to construct an additive synthesis setup — summing sinusoids with specific amplitude, frequency, and phase parameters. For the channel vocoder, we observed that the filter-banks tracked the amplitude as a function of time (instantaneous amplitude) and there was not much attention given to the frequency itself — the focus on the frequency/phase is what sets the two vocoders apart. The phase vocoder includes the notion of *instantaneous frequency* which is computed via *instantaneous phase* values of two consecutive time-frames. Let's look at Figs. 3.18, 3.19, and 3.20 to get a better idea why instantaneous frequency is important in producing "high-quality" synthesized versions of the original input.

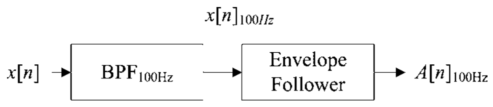


Fig. 3.18. Single channel: center frequency at 100 Hz strip of a channel vocoder (analysis).

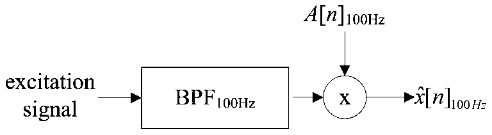


Fig. 3.19. Single channel: center frequency at 100 Hz strip of a channel vocoder (synthesis).

As previously mentioned, the channel vocoder's filter-banks have fixed center frequencies (bands-pass filter's frequencies) and the input is divided into a number of hard-wired sub-bands for which only the energy level (instantaneous amplitude $A[n]$) is computed at each band. Like the encoder, the decoder also has a fixed filter-bank configuration with center frequencies identical to the encoder. For the phase vocoder this is not the case.

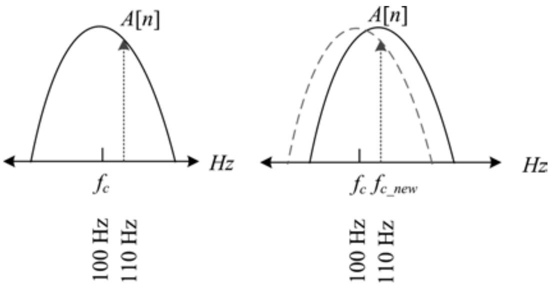


Fig. 3.20. Actual frequency location of 2nd harmonic and channel for center frequency $f_c = 100\,\text{Hz}$ in analysis part (left). Center frequency f_{c_new} shifted via instantaneous frequency in re-synthesis (right).

Let's consider a scenario where the input is a complex harmonic signal with fundamental frequency 55 Hz and 3 harmonics total. For this particular example, the octave (equivalent to the second harmonic) would be at 110 Hz. Let's also assume that for whatever reason, the filter-banks have been tuned so that the second harmonic (110 Hz) falls into the filter-bank corresponding to the channel with center frequency 100 Hz as shown in Fig. 3.20(left). As we can see, the second harmonic at 110 Hz, which is the input to this particular sub-band, is not exactly at the filter bank's hard-wired center frequency of 100 Hz. This is a problem. The problem arises in the synthesis part (decoder) as the decoder has no way of knowing (or does not care in the case of channel vocoder) what the original frequency of the sinusoid was. Hence, during synthesis, that particular sub-band with center frequency 100 Hz will resonate at none other than 100 Hz and not 110 Hz, although with the appropriate energy level controlled via $A[n]_{100\,Hz}$: the instantaneous amplitude we computed with the envelope follower.

So, what does the phase vocoder do better than the channel vocoder? It helps compute a more accurate frequency for synthesis, through estimation of the instantaneous frequency for a particular channel in the encoder module. During the synthesis stage, the decoder re-synthesizes the output signal with the analyzed instantaneous frequency for the center frequency using additive synthesis techniques. Thus, unlike the channel vocoder, the phase vocoder synthesizes the output by directly combining sinusoids with appropriate instantaneous amplitude and instantaneous frequency values and is not based on a source-filter model. How do we compute the instantaneous frequency? Read on ...

3.4.1 *Estimation of instantaneous frequency*

We already have been briefly introduced to the instantaneous frequency in Chap. 4, Sec. 2. The instantaneous frequency is shown in Eq. (3.20) where $\Theta(t)$ is the phase ($\omega t + \phi$, not to be confused with the initial phase ϕ) depicted in (3.22) for a sine wave (3.21). The initial phase ϕ in Eq. (3.22) can also be a function of time in which case it becomes $\phi(t)$. When ϕ is a constant (initial phase) then $\phi(t)$ will of course be a constant as well.

$$f(t) = \frac{1}{2\cdot\pi}\frac{d\Theta(t)}{dt} \tag{3.20}$$

$$y(t) = \sin\Theta(t) \tag{3.21}$$

$$\Theta(t) = 2\cdot\pi\cdot f\cdot t + \phi = \omega\cdot t + \phi \tag{3.22}$$

In a digital world, the instantaneous frequency is commonly computed via two consecutive STFT frames (when using the Fourier transform) by first calculating the phase value at bin k (3.24), followed by computing the difference of respective unwrapped phase angles, and finally dividing the difference by the duration (R in samples) between two consecutive STFT frames ($X^d[\cdot]$) as shown in Eq. (3.26). The 2π divisor acts as a normalizer and gets rid of the radian unit so that the result f is in Hertz. The result is the instantaneous frequency f_{m+1} at the next time instance $m+1$.

$$X^d[k] = a + jb \tag{3.23}$$

$$\angle X^d[k] = \varphi[k] = \tan^{-1}\left(\frac{b}{a}\right) \tag{3.24}$$

$$\left|X^d[k]\right| = \sqrt{a^2 + b^2} \tag{3.25}$$

$$f_{m+1} = \frac{1}{2\pi} \cdot \frac{\varphi_{m+1} - \varphi_m}{R} \cdot f_s \tag{3.26}$$

It is sometimes helpful to better understand instantaneous frequency if we think about it in terms of the phase change on a circle and how frequency itself is defined within that circle. If we look at a sinusoid as shown in Fig. 3.21 for example, we would see the "movement" of a "point" along the unit circle in a counterclockwise direction at every tick of the digital "word clock" $(1/f_s)$. The sine function's waveform motion can be represented on the unit circle in terms of the points on the x and y axes or the angle that results between the x and y coordinates. We know that the definition of frequency is resolutions per second and thus instantaneous frequency can be regarded as the change in phase divided by the time interval between two consecutive phase values: $T = 1/f_s$ seconds in this particular example and the phase is computed for each sample period via Eq. (3.24). This idea is shown in Fig. 3.21.

So, if we again take the 100 Hz center frequency band-pass filter as an example and structure the band-passed output to be in the form of $a+jb$, we can compute the instantaneous phase and instantaneous frequency

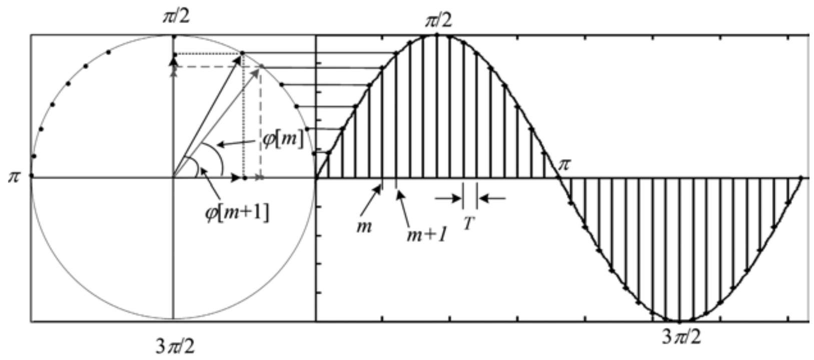


Fig. 3.21. Instantaneous frequency and phase change rate.

(if we consider the y-axis to be the imaginary axis as in Fig. 3.21). Thus, for two consecutive points in discrete time, say, between sample $n = m$ and $n = m + 1$, we can compute the instantaneous frequency as shown in Eq. (3.26), where $R = T$. However, in order to compute a correct instantaneous frequency value, we have to address phase unwrapping issues as radian phase has a modulo 2π characteristic (there will be discontinuities when computing the phase difference whenever the phase exceeds a full cycle). This is discussed in the next section and in more detail in Sec. 3.4.4.

3.4.2 *Phase unwrapping*

Before we can compute the instantaneous frequency we need to do one more thing, namely, phase unwrapping followed by instantaneous frequency estimation. We already had the fortune of getting acquainted with phase unwrapping in Chap. 6, Sec. 3.3 and remember that it is basically a process whereby we add 2π radians whenever a phase increment passes the 2π point (modulo 2π system). Thus, the reason we need to perform phase unwrapping is due to the fact that when we take the difference between two consecutive phases $\varphi_2 - \varphi_1$, there is the potential of getting the incorrect instantaneous phase difference due to the modulo 2π characteristic (we need the change in phase to compute the instantaneous frequency). To clarify this, let's say we are computing the instantaneous phase of a particular channel with $f_c = 100\,\text{Hz}$ and the computed phase ϕ_{nT} results in a sequence as shown in Eq. (3.27) where n is the sample index for period $T = 1$ sec (for convenience we use degrees rather than radians in this example).

$$\varphi_1 = 0, \quad \varphi_2 = 95, \quad \varphi_3 = 190, \quad \varphi_4 = 285, \quad \varphi_5 = 10, \quad \varphi_6 = 105 \tag{3.27}$$

If we compute the instantaneous frequency for the above six phase values, we arrive at the following:

$$\begin{aligned}
\frac{\varphi_2 - \varphi_1}{T} &= \frac{95 - 0}{1} = 95 \\
\frac{\varphi_3 - \varphi_2}{T} &= \frac{190 - 95}{1} = 95 \\
\frac{\varphi_4 - \varphi_3}{T} &= \frac{285 - 190}{1} = 95 \\
\frac{\varphi_5 - \varphi_4}{T} &= \frac{10 - 285}{1} = -275 \\
\frac{\varphi_6 - \varphi_5}{T} &= \frac{105 - 10}{1} = 95
\end{aligned} \tag{3.28}$$

We see that the instantaneous frequency seems to be quite constant at 95 Hz until we reach frame 4 and 5: it is −275 Hz and not 95 Hz. There seems to be something wrong here as one would normally not expect such a drastic change in frequency or no change in frequency at all if it is an ideal constant sinusoid. This is a typical case of wrapped phase due to modulo 360° (or 2π) — the phase gets initialized much like our watch gets reinitialized whenever the 12 hour mark is passed. Thus, with phase unwrapping after frame 5 — adding 360° (or 2π) we will be able to compute the correct instantaneous frequency:

$$\begin{aligned} \frac{\varphi_5 - \varphi_4}{T} &= \frac{(10+360) - 285}{1} = 95 \\ \frac{\varphi_6 - \varphi_5}{T} &= \frac{(105+360) - (10+360)}{1} = 95 \end{aligned} \tag{3.29}$$

Figure 3.22 shows phase unwrapping for a sinusoid where the straight line is the unwrapped phase and the saw-tooth like figure depicts the phase before unwrapping.

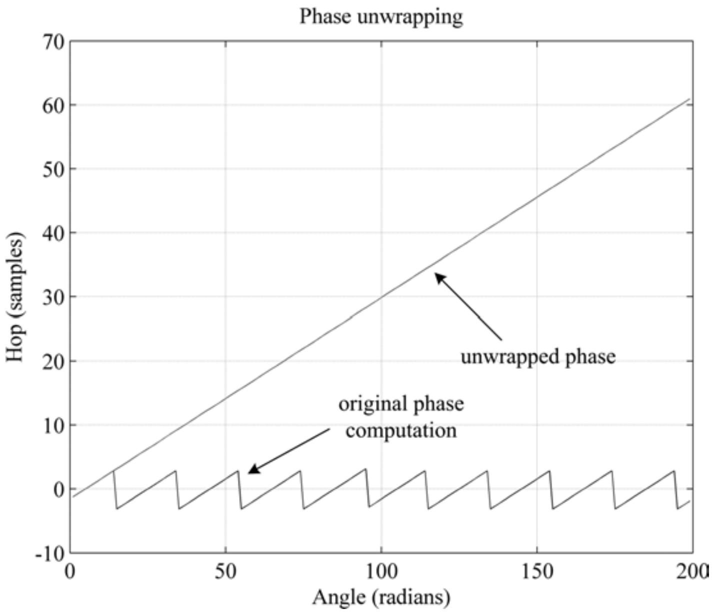


Fig. 3.22. Phase unwrapping.

3.4.3 *Phase vocoder: Filter-bank interpretation*

The explanation of the phase vocoder up until now pretty much falls under the category of the "filter-bank interpretation" (Dolson 1986) as we have basically viewed the system as a bank of band-pass filters on the encoder and decoder side. In this section, we will formally tackle the task of interpreting the phase vocoder from this perspective and walk you through the process and steps for obtaining the instantaneous frequency. As mentioned earlier, in the analysis part (encoder), the instantaneous amplitude and the instantaneous frequency are estimated, whereas in the synthesis part (decoder), those two parameters are used for re-synthesis via additive synthesis techniques — literally summing of sinusoids with their respective frequencies components. This idea is shown in Figs. 3.23 and 3.24 where $A[m]$ and $f[m]$ are the instantaneous magnitude and instantaneous frequencies for a particular filter-bank and time-frame m.

Let's now take a more detailed look at how to compute the instantaneous phase with the objective of estimating the instantaneous frequency from the filter-bank interpretation's perspective. We do this by first simplifying things a bit: concentrate on the encoder's instantaneous

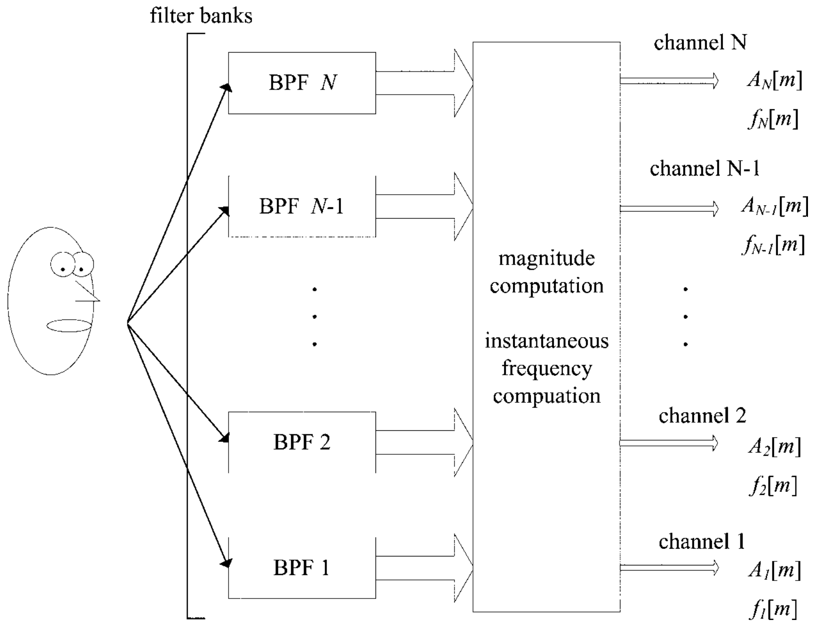


Fig. 3.23. Filter-bank structure of phase vocoder: encoder.

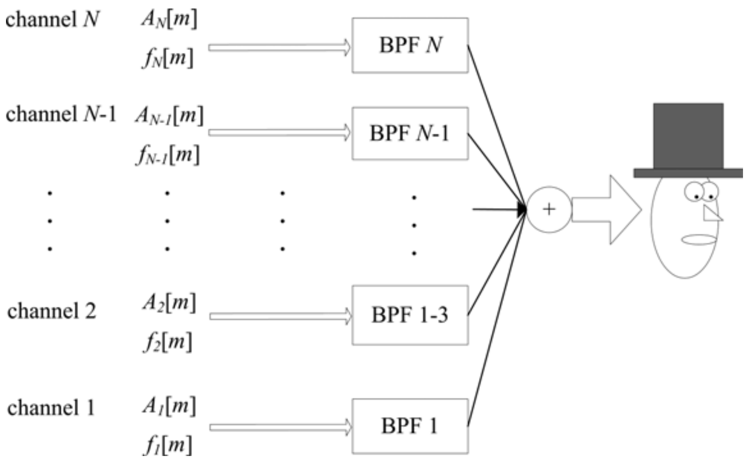


Fig. 3.24. Filter-bank structure of phase vocoder: decoder.

frequency computation block and the signal flow for a single channel of the phase vocoder (each channel will be identical except the center frequency) as shown in Fig. 3.25. This diagram may initially seem quite complex and to be fair, it may actually be not that simple at first glance, but if we break up the overall process and follow the procedure step by step for each sub-block at a time it will become clear that the whole process is actually less formidable than meets the eye. We will keep our eye on the ball by remembering that our objective is to estimate the instantaneous frequency, which in turn is accomplished by computing the instantaneous phase of two successive time-frames.

Consider an example where the "filter-bank" channel k with center frequency f_k is tuned to 100 Hz and one sinusoidal component (such as a harmonic) is near the center frequency of this particular channel (we have put the "filter-bank" in quotations as there is no actual filter-bank per se in the phase vocoder and band-pass filtering is accomplished differently as will be explained shortly). As shown in Fig. 3.26 (beginning section of Fig. 3.25), the input $x(t)$, viewed from in the frequency-domain, will be just a vertical line (ideally speaking) as it is a single sinusoid. Next, we notice that the input is divided into two parallel paths and each path is multiplied by a sine and cosine — in our example $\omega_k = 2\pi f = 2\pi \cdot 100$. The result is then low-pass filtered at ω_k as shown in Fig. 3.27. Now, these two processes serve one purpose — enable us to compute the phase. How does this get accomplished? It gets accomplished by a process called *heterodyning.*

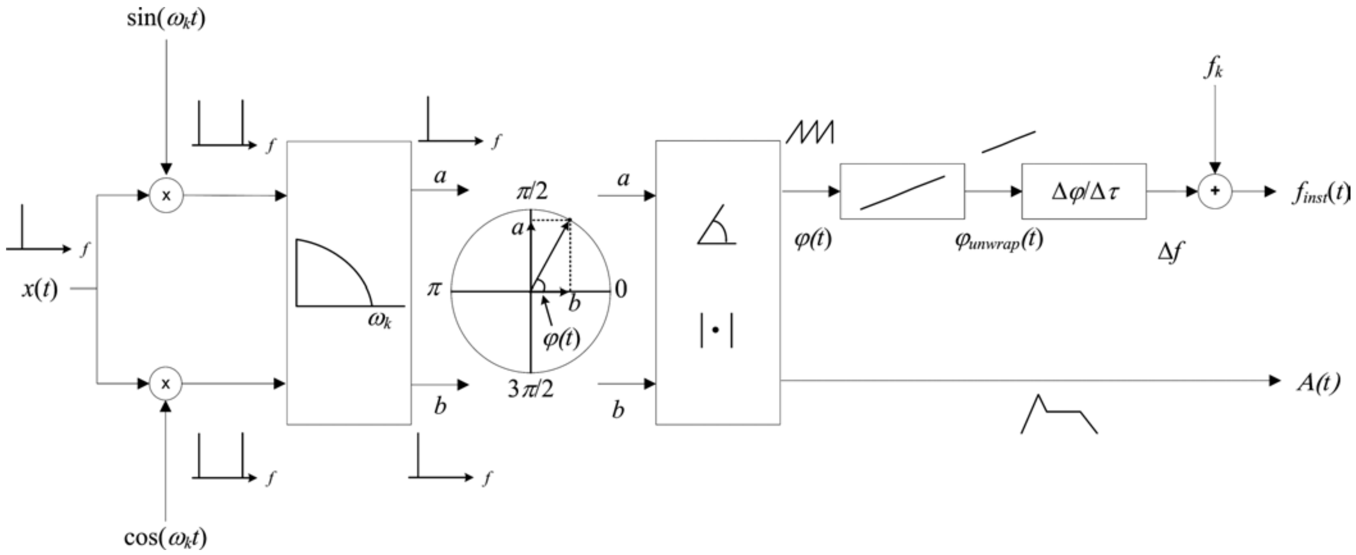


Fig. 3.25. Overview of phase vocoder encoder: filter bank interpretation.

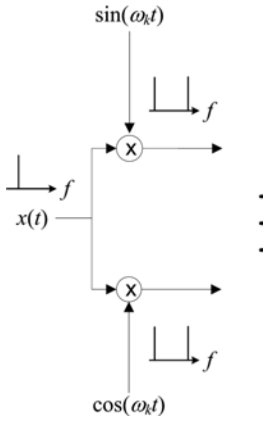


Fig. 3.26. Heterodyning.

Why do we need to compute to phase? The answer is of course: to estimate the instantaneous frequency.

You will remember that in the DFT, if the output of a bin is as shown in Eq. (3.30), we can compute the phase as well as the magnitude using (3.31) and (3.32). But if we are not using the Fourier transform, how do we compute the phase? The answer is heterodyning.

$$c = a + jb \tag{3.30}$$

$$\angle c = \varphi = \tan^{-1}\left(\frac{b}{a}\right) \tag{3.31}$$

$$|c| = \sqrt{a^2 + b^2} \tag{3.32}$$

Heterodyning serves to facilitate the computation of the phase by formatting the input $x(t)$ so that the output of the sine and cosine multiplication results in a rectangular coordinate system (much like the real plus imaginary system in the Fourier transform) as shown in Fig. 3.26. How does heterodyning enable us to jump to a rectangular coordinate system? In order to answer this, we will need to go back a few chapters and recall from amplitude modulation and beating (Chap. 4, Sec. 4.1), that the following relationship holds true:

$$\cos(A) \cdot \cos(B) = \frac{\cos(A - B) + \cos(A + B)}{2} \tag{3.33}$$

We recollect that this basic relationship is the *sum and difference* ($A + B$ and $A - B$) which forms the basis for heterodyning. In heterodyning, the multiplication of the input is by $\sin(\omega t)$ and $\cos(\omega t)$ where each results in a sum and difference pair. The *sum* part is removed via the low-pass filter, allowing the difference part $(A - B)$ only to remain. We also know that the cosine and sine will always be 90° out of phase, with the cosine being ahead by 90° at all times. Thus, the output of heterodyning — multiplication and low-pass filtering, will result in two sinusoidal components (sine and cosine) shifted in frequency $(A-B)$ towards the DC, where each component will be 90° out of phase with respect to each other. The shifting in our example is by $f_k = 100\,\text{Hz}$. Due to the 90° phase difference between the two sinusoids (output of heterodyning cosine and sine with input), a rectangular coordinate system results after low-pass filtering as illustrated in Fig. 3.27.

The output components resulting from this transformation of the input signal (the vertical and horizontal components) are referred to as the *in-phase* and *quadrature* components. Since we have the input in a rectangular coordinate configuration, it is now easy to compute the phase using (3.31) and the magnitude according to (3.32) as shown in Fig. 3.28.

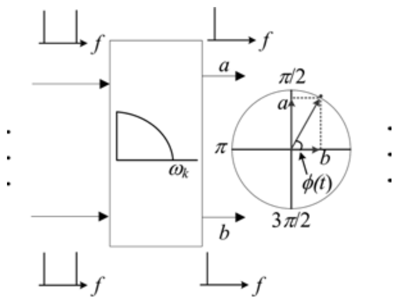


Fig. 3.27. Low-pass filtering of the heterodyned output.

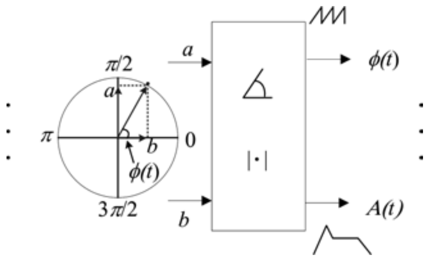


Fig. 3.28. Phase and amplitude envelope computation.

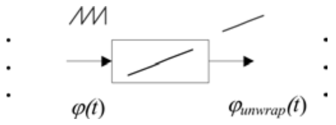


Fig. 3.29. Phase unwrapping.

Thus, as mentioned earlier, the phase vocoder does not use "filter-banks" per se (traditional band-pass filter) but utilizes the heterodyning process which behaves somewhat like a band-pass filter. In any case, as far as the instantaneous amplitude and phase computation is concerned, we're done. The next step is to compute the instantaneous frequency or more specifically the "heterodyned" instantaneous frequency, since the input carrier signal has been shifted by f_k towards DC during the heterodyning process. This is achieved via the two consecutive instantaneous phase values as further described below. We know that in order to compute the instantaneous frequency, we need to perform phase unwrapping first (see Sec. 3.4.2 as to why we need phase unwrapping) and this is precisely what the next step is as shown in Fig. 3.29. The result after phase unwrapping is $\varphi(t)_{unwrap}$.

This process of phase unwrapping is performed for two consecutive frames where the duration between the two frames is $\Delta\tau$. Thus, the instantaneous frequency is estimated simply by computing the phase difference between those two frames ($\Delta\varphi$) and dividing it by $\Delta\tau$. One last thing we need to do before we ship the results off to the decoder, is to bring back the instantaneous frequency to the proper frequency range. Due to heterodyning, the input was shifted down toward the DC by f_k Hertz and thus the last process requires us to add back f_k to the heterodyned instantaneous frequency. For example, if the input sinusoid was 102.34 Hz, heterodyning would have shifted it down to 2.34 Hz, requiring us to add back f_k to Δf to obtain the actual instantaneous frequency. This is shown in Fig. 3.30.

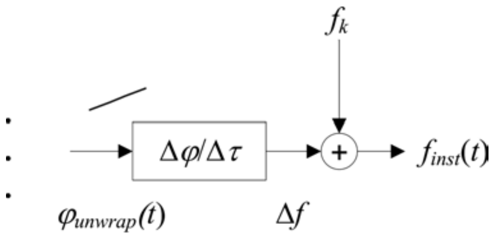


Fig. 3.30. Adding back center frequency f_k to computed instantaneous frequency.

This basically concludes the concept and process behind of the encoder side of the phase vocoder viewed from the filter-bank interpretation's perspective as summarized in Fig. 3.25. I hope that Fig. 3.25 at this point does not look as daunting as before. On the decoder side, life is easier (it usually is) as we only need to plug in the frequency and magnitude values and re-synthesize the output by summing all of the sinusoids with their respective parameters transmitted from the encoder.

3.4.4 *Phase vocoder: Fourier transform interpretation*

The guts of the phase vocoder as seen from Mr. Fourier's point of view is based on the STFT with overlapping windowed portions of the input signal. In essence, the Fourier transform interpretation of the phase vocoder is actually very similar to the filter-bank interpretation (as it should be). The STFT version also includes a collection of "band-pass-filters" conforming to a "filter-bank" structure in the form of evenly spaced discrete frequency bins — each bin k (f_s/N Hz where N is the DFT length) can be loosely regarded to correspond to one channel. The number of channels is controlled via the length of the DFT window size. For example, if N is 1,024 samples long with sampling frequency $f_s = 8$ kHz, the frequency resolution will be 7.8125 Hz. For bin 1 to bin 3, this would be equivalent to "center frequencies" corresponding to 7.8125 Hz, 15.625 Hz, and 23.4375 Hz so-to-speak.

$$X^d[k] = \sum_{n=0}^{n=N-1} x[n] \cdot e^{-j\frac{2\pi kn}{N}} \tag{3.34}$$

In the filter-bank model of the phase vocoder, we used heterodyning to render the in-phase and quadrature components which were used as an intermediate step towards obtaining the phase information. In the Fourier transform version, the phase is computed directly from the frequency bins themselves and the instantaneous phase and instantaneous frequency from two successive STFT frames. As we can see in Eq. (3.34), the DFT also does a heterodyning-like multiplication — computing the product of the input $x[n]$ and the Euler identity which is of course made up of a cosine and sine component.

The phase vocoder is very flexible for applications such as time-stretching and pitch-shifting (without the munchkin effect that we encountered with down-sampling or the "inverse-munchkin" effect when up-sampling). As mentioned before, due to the use of the Fourier transform

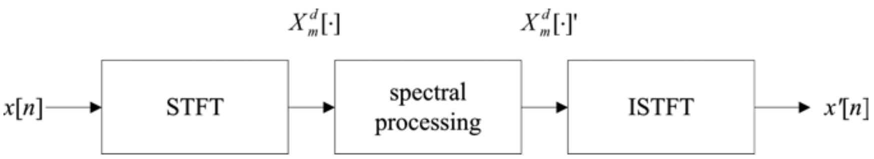


Fig. 3.31. Basic structure of phase vocoding.

for phase vocoding, the output of the analysis part will entail a frequency-domain view of the filter-banks — each bank represented by its individual frequency, magnitude, and phase components. The synthesis part takes these three components (probably modulated in some way) and re-synthesizes the final output, ultimately rendering a time-domain signal via inverse-STFT: summing of overlapping STFT frames. This basic idea is depicted in Fig. 3.31 where the STFT part takes on the analysis task, spectral processing the job of modulation/transformation/modification of the spectra, and the ISTFT is given the responsibility to safely bring us back home to the time-domain.

We have already talked about phase unwrapping and instantaneous frequency before and unsurprisingly these two concepts are also critical for the Fourier-based phase vocoder where for each frequency bin k, phase unwrapping needs to be performed. Computing the instantaneous frequency entails devising a phase unwrapping algorithm in order to get the unwrapped instantaneous phase change between two consecutive STFT frames, the ultimate goal being the preservation of the instantaneous frequency. When using the STFT for the phase vocoder, each bin k corresponds to one channel with each discrete frequency slot spinning in a counterclockwise direction at a particular angular velocity, much like a rotating wheel. The lower frequency bins spin slower (DC does not spin at all) and the higher bins spin faster — for bin k the "radian distance" travelled (the phase when uncoiled) in one sample corresponds to Eq. (3.35) where N refers to the divisor of the frequency axis (window size/DFT size), k to the bin number, and φ the phase for bin k.

$$\varphi[k] = \frac{2\pi k}{N} \tag{3.35}$$

This means that for higher frequency bins, the number of cycles (full rotations) the "wheel" goes through is greater compared to lower ones for the same period of time. Think of a race between a rabbit, raccoon, and turtle racing around a circular racetrack and measuring how much distance is covered by each one in, say, 5 minutes . . . huh? . . . oh! The instantaneous unwrapped phase for the next frame is computed by figuring out what the projected phase value should be (number of 2π turns) with respect to the current frame and fine tuning it by adding the actual phase that is obtained at the next frame. This is depicted in Fig. 3.32 where we use degrees (°) instead of radians to facilitate the grasping of the concept for phase unwrapping in the STFT-based phase vocoder.

For example, let's say that the current frame s and next frame $s+1$ have an analysis STFT hop size of R_a samples. If the bin k in question has an angular frequency of 45° per sample, in R_a samples, it will travel R_a times 45° which will be the theoretical instantaneous phase value at the next frame $s+1$. That is, if it were a perfect sinusoid fitting into the analysis window without getting cut off midstream (full sinusoidal waveforms), the result will be as shown in Eq. (3.36) (projected instantaneous unwrapped phase in Fig. 3.32).

$$\varphi^p_{(s+1)R_a}[k] = \frac{360 \cdot k}{N} R_a + \varphi_{sR_a}[k] \tag{3.36}$$

However, if the sinusoid is quasi-stable and changes as shown in Fig. 3.32 (wiggly line opposed to dotted line), the instantaneous phase cannot be computed with merely (3.36) as we do not have enough information

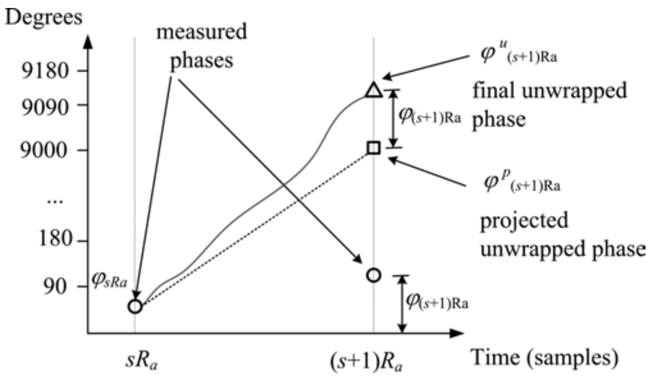


Fig. 3.32. Phase unwrapping between two consecutive STFT frames (after Zölner 2002).

regarding the behavior of the phase between frame $s+1$ and s. Hence, for the general case in computing the instantaneous phase, we would need to find out how many cycles (m integer multiples of 360 or 2π) a particular bin k has gone through (in our example 45°) and add the measured phase (obtained via the DFT) at frame $s+1$ to the result according to Eq. (3.37). Thus, the problem now becomes finding this integer m.

$$\varphi^u_{(s+1)R_a}[k] = 360 \cdot m + \varphi_{(s+1)R_a}[k] \tag{3.37}$$

Finding m can be simply achieved by using Eq. (3.38). We compute the integer number of 360° resolutions found in Eq. (3.36) via Eq. (3.38).

$$m = \text{int}\left(\frac{\varphi^p_{(s+1)R_a}[k]}{360}\right) \tag{3.38}$$

If we take the 45° as an example for the angular update rate per sample for a given frequency bin $k = 32$, window size $N = 256$, $\varphi_s = 47°$, and hop size $R_a = 200$ samples, we will get the following results for the projected unwrapped instantaneous phase and modulo 360 multiplier m:

$$\begin{aligned}\varphi^p_{(s+1)R_a}[k] &= \frac{360 \cdot k}{N} R_a + \varphi_s[k] \\ &= \frac{360 \cdot 32}{226} 200 + 47 \\ &= 45 \cdot 200 + 47 \\ &= 9047°\end{aligned} \tag{3.39}$$

$$\begin{aligned}m = \text{int}\left(\frac{\varphi^p_{(s+1)R_a}[k]}{360}\right) &= \text{int}\left(\frac{9047}{360}\right) \\ &= \text{int}(25.13056) \\ &= 25\end{aligned} \tag{3.40}$$

From Eq. (3.40), we can compute the final unwrapped instantaneous phase by using Eq. (3.37) and the measured phase $\varphi_{(s+1)Ra}$ via the complex number representation for bin $k = 32$ which happened to be 120° in our example. This results in 9, 120° as shown in (3.41). The results of computing the phases are shown in Fig. 3.32 and again in Fig. 3.33 from the circular rotational perspective — the inner most circle denotes the measured phase locations ($m = 0$) and the outer circles the subsequent 360° added phases with the outermost circle corresponding to $m = 25$ as computed from Eq. (3.37).

$$\begin{aligned} \varphi^u_{(s+1)R_a}[k]|_{k=32} &= 360 \cdot m + \varphi_{(s+1)R_a}[k] \\ &= 360 \cdot 25 + 120 \\ &= 9120 \end{aligned} \tag{3.41}$$

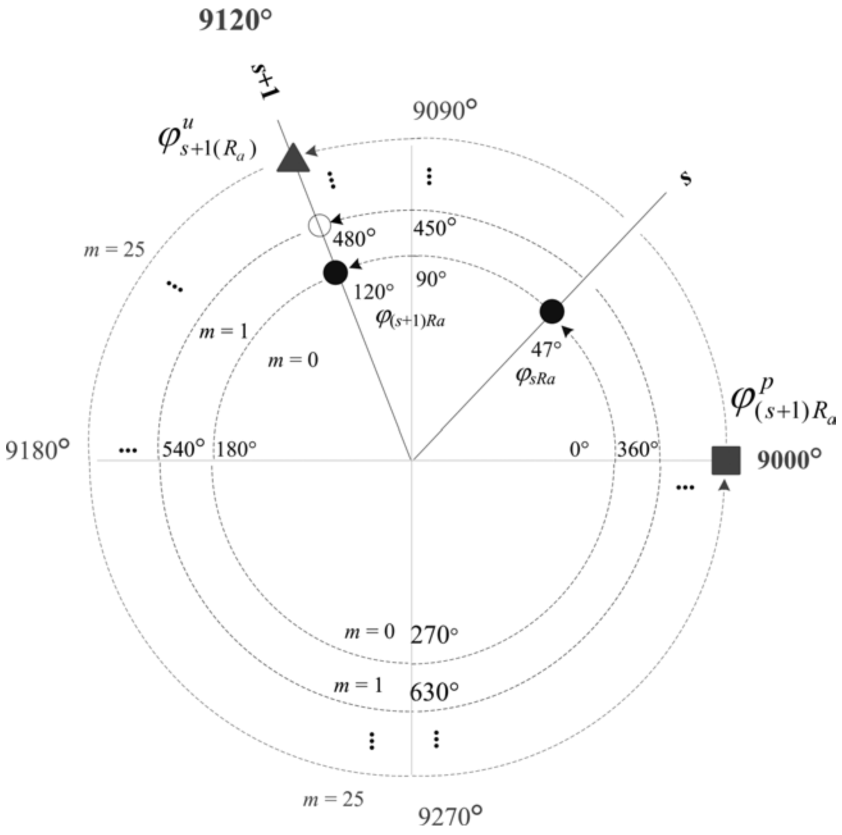


Fig. 3.33. Phase unwrapping via and computation of instantaneous phase.

We have used degrees instead of radians to make things a bit clearer but the equations for phase unwrapping can of course be written in radians as well. If radians are used, Eqs. (3.36), (3.37), and (3.38) become (3.42), (3.43), (3.44), and the final unwrapped phase will be equal to $9120 \cdot \pi / 180 = 50.67 \cdot \pi$ radians.

$$\varphi^{p}_{(s+1)R_a}[k] = \frac{2\pi \cdot k}{N} R_a + \varphi_s[k] \tag{3.42}$$

$$\varphi^{u}_{(s+1)R_a}[k] = 2\pi \cdot m + \varphi_{(s+1)R_a}[k] \tag{3.43}$$

$$m = \text{int}\left(\frac{\varphi^{p}_{(s+1)R_a}[k]}{2\pi} \right) \tag{3.44}$$

From Eq. (3.44) we can compute the unwrapped phase difference between two consecutive frames s and $s+1$ using Eq. (3.45) and finally, since we now have the delta phase, we can compute the instantaneous frequency with Eq. (3.26) resulting in Eq. (3.46) where we multiply the results by the sampling frequency to obtain the instantaneous frequency in Hertz.

$$\Delta\varphi_{(s+1)R_a}[k] = \varphi^{u}_{(s+1)R_a}[k] - \varphi_{sR_a}[k] \tag{3.45}$$

$$f_{s+1}[k] = \frac{1}{2\pi} \cdot \frac{\Delta\varphi_{(s+1)R_a}[k]}{R_a} \cdot f_s \tag{3.46}$$

3.4.5 *Phase vocoder basics: Time and pitch-shifting*

In Chap. 2, we briefly saw how we could perform time-stretching and compression in the time-domain using different hop sizes for the analysis and synthesis part via the OLA method and windowing (without caring about instantaneous phase/frequency). That is, an analysis hop size in samples R_a, smaller than the synthesis hop size R_s, would render a longer signal than the original, resulting in time-expansion and vice-versa. We also were able to implement a pitch-shifting algorithm in the same chapter, albeit a pretty bad one, as pitch-shifting meant changing the duration of a signal inadvertently producing the munchkin effect as a byproduct. However, as we are now equipped with the phase vocoder, we can produce much better results in both time-shifting and pitch-shifting situations with "little" added artifacts (if we are mindful). In this section, we will discuss some basic approaches and issues involved in pitch-shifting and time-shifting audio signals.

3.4.5.1 Time-shifting

There are two main approaches for both time-shifting and pitch-shifting via phase vocoding and they both usually entail using the same STFT-based analysis block but different synthesis block. When using the STFT in the analysis block, we can opt for the filter-bank model (sum of sinusoids) or the ISTFT model for the synthesis side. We first start with the filter-bank synthesis version (while using the STFT for the analysis part).

Time-shifting: Filter-bank model. We have already covered the analysis part and also discussed computing the instantaneous frequency in the previous section. For the synthesis part, using the filter-bank model, the result for time-shifting is implemented by appropriately sustaining or shortening the sinusoidal components via interpolation and decimation respectively. That is, for time-stretching we will need to interpolate the amplitude values of each sinusoid (frequency bins k) between the current frame and the next *new* frame's location, whereas for time-compression, decimation takes place. A time-stretching example is shown in Fig. 3.34 with synthesis hop size $R_s = 3 \cdot R_a$ where the general time-shift ratio is expressed as Eq. (3.47).

$$time\_shift\_ratio = \frac{R_s}{R_a} \tag{3.47}$$

The important difference between time-domain time-stretching (introduced in Chap. 2) and the frequency-domain is that in frequency-domain time-stretching, we have the privilege of controlling the sinusoidal components and their respective phase characteristics that make up a given sample on the discrete-time axis while at the same time maintaining the instantaneous frequency characteristics. The up-sampling/down-sampling approach in Chap. 2 did not allow us to do that. Since we know the instantaneous phase difference $\Delta\varphi_{s+1}[k]$ between frames s and $s+1$ at their original locations before time-stretching, we can determine the *phase increment* ($\psi_{(s+1)R_a}[k]$) per sample n that is required to drive the additional sinusoids that are needed to fill the space between the current frame and the *new* location of the next frame $\varphi'_{(s+1)R_a}[k]$ using Eq. (3.48).

$$\psi_{(s+1)R_a}[k] = \frac{\Delta\varphi_{(s+1)R_a}[k]}{R_a} \tag{3.48}$$

The new interpolated phase values (denoted by superscript i) between the original frame and the time-stretched frame (n begins at $s \cdot R_a + 1$ and ends at $3 \cdot s \cdot R_a$ samples) is computed with Eq. (3.49). The synthesized output

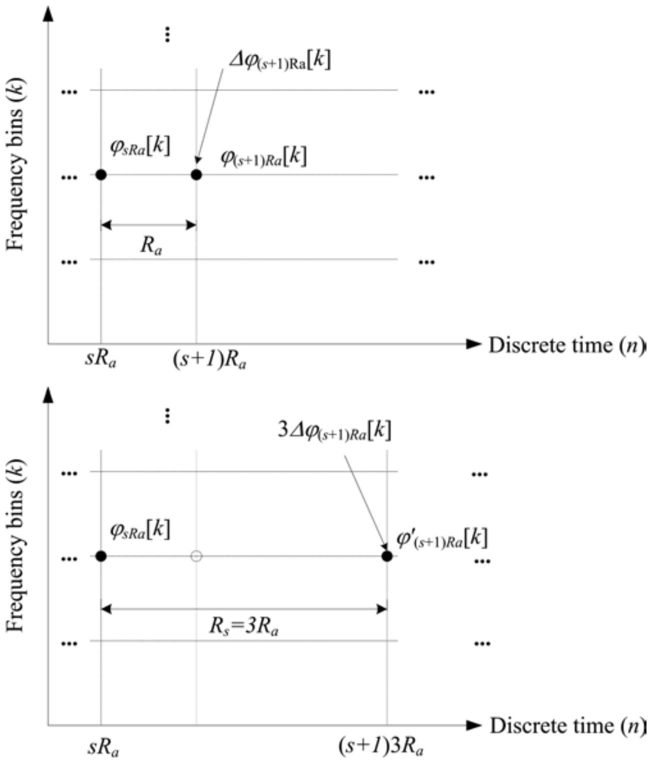


Fig. 3.34. Time-stretching: 3 fold $R_s = 3 \cdot R_a$. Top figure plot shows the analysis configuration and the bottom plot the time-stretched configuration.

sample $y[n]$ is then finally reconstructed (synthesized) via the sum of $N/2$ sinusoids (N is the window/FFT size and k the bin index) at each sample n with their corresponding amplitude values and instantaneous phase values as shown in Eq. (3.50).

$$\varphi^i_{n+1}[k] = \varphi_n[k] + \psi_{(s+1)R_a}[k] \tag{3.49}$$

$$y[n] = \sum_{k=0}^{N/2} A_n[k] \cdot \cos(\varphi^i_n[k]) \tag{3.50}$$

Let's look at Fig. 3.35 to help us better understand how the amplitude and phase values are updated and why the instantaneous frequency plays such a critical role in contributing to a sinusoid's behavior and ultimately the quality of sound.

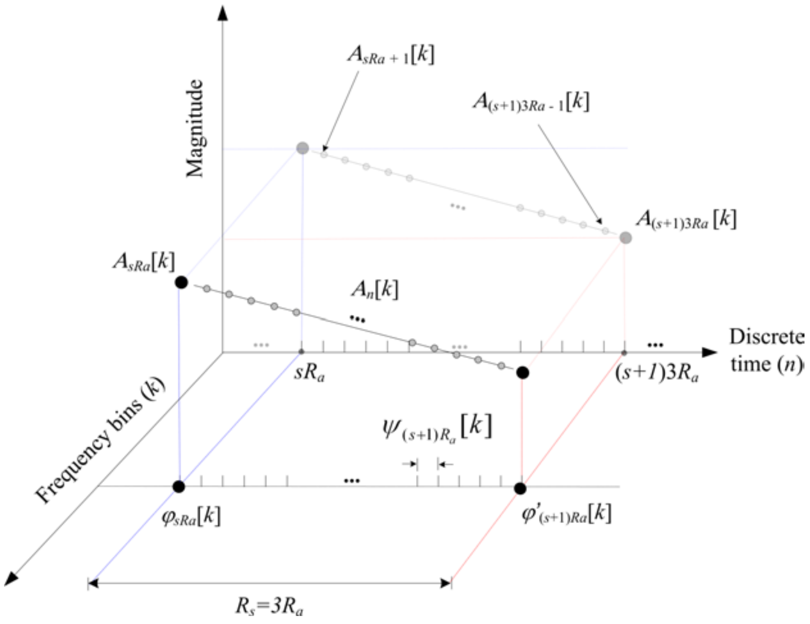


Fig. 3.35. Time-stretching $R_s = 3 \cdot R_a$ and interpolation of magnitude and phase.

Figure 3.35 shows the same bottom plot of Fig. 3.34 with the added magnitude interpolation dimension along with details on the phase increment/interpolation between the current frame and new next frame at $(s + 1) \cdot 3 \cdot R_a$ sample time. As we can see, the magnitude component of the sinusoid located at bin k, is linearly interpolated from $A_{sRa}[k]$ to $A'_{(s+1)3Ra}[k]$, providing the appropriate magnitude values for the sinusoids at each point in time (n) between $s \cdot R_a$ and $(s + 1) \cdot 3 \cdot R_a$. For interpolation of the phase between the current and next frame, we know that the phase increment should follow $\psi_{(s+1)Ra}$ in Eq. (3.48) which allows the instantaneous frequency to be maintained — the radian increment ("radian distance" travelled per sample) will be sustained during this region according to (3.48). Thus, since we have the appropriately interpolated magnitude values $A_n[k]$ and the interpolated instantaneous phase φ^i_{n+1} at each sample, we can use Eq. (3.50) to compute the synthesized output at each sample n via the sum of $N/2$ sinusoids (DC to Nyquist limit). Each point in time n will be represented by $N/2$ sinusoids.

For time-compression, we need only to employ a decimation scheme since the number of samples between the current frame s and next frame

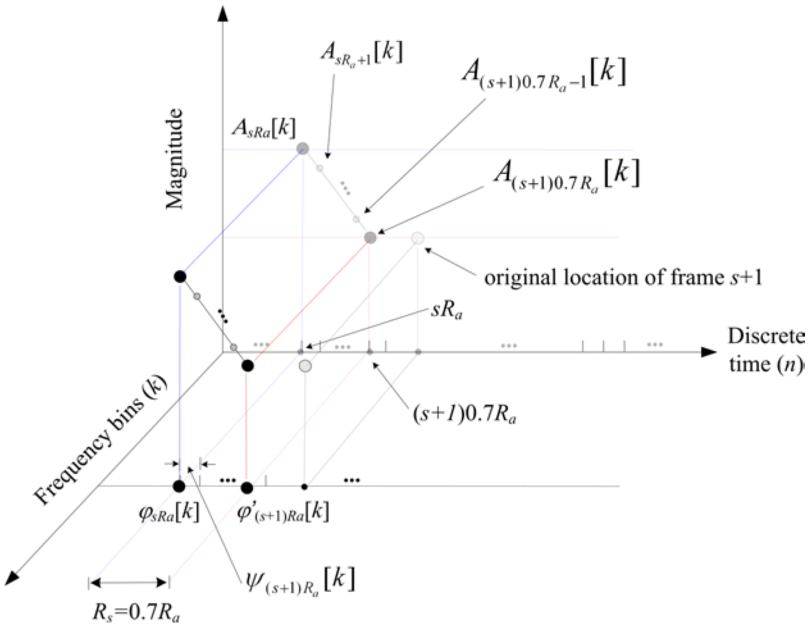


Fig. 3.36. Time-compression $R_s = 0.7 \cdot R_a$ and decimation of magnitude and phase.

s+1 will eventually have to decrease. However, all of the equations employed for time-stretching also apply here as shown in Fig. 3.36 where we used an example of 30% decrease in the length of the signal.

Time-shifting: ISTFT model. A similar approach is taken for time-shifting via ISTFT (overlapping DFTs) and the phase vocoder. In the Fourier transform method, the ratio of R_s/R_a (synthesis hope size vs. analysis hop size) is used to determine the stretching or contraction factor while preserving the instantaneous frequency characteristics computed in the analysis part. However, unlike the filter-bank model, the main difference lies in the frame-by-frame synthesis rather than sample-by-sample synthesis as each IDFT in the context of ISTFTs spans a *region* of time dictated by the synthesis hop size R_s and the DFT size as depicted in Fig. 3.37. Contrasting the filter-bank model, where at each time n, a new output sample is produced via the sum of sinusoids, in the ISTFT model the DFT frame (with its $N/2$ sinusoids) is sustained for the duration of the DFT window length and added in the usual OLA method on the synthesis side producing the time-shifted output.

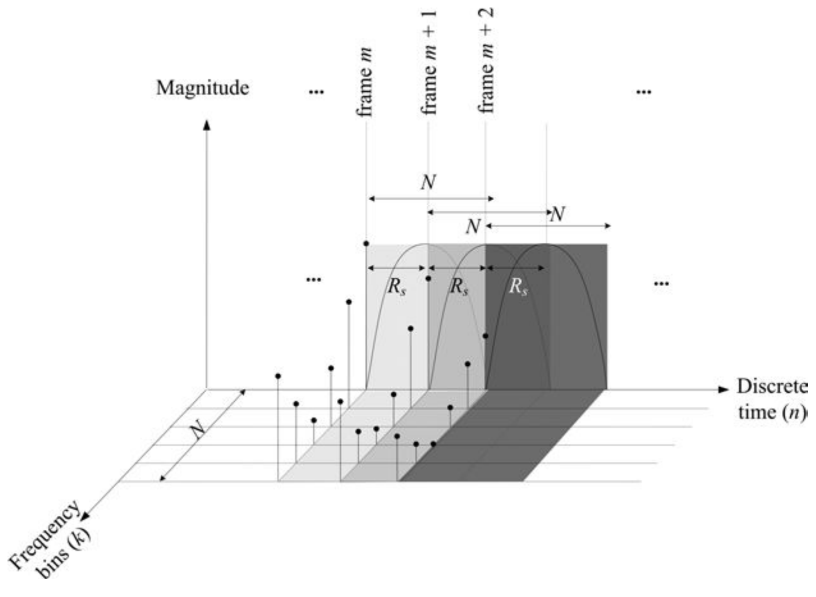


Fig. 3.37. ISTFT model for synthesis in time-shifting.

As mentioned before, the main issue is in preserving the instantaneous phase by scaling the new delta phase $\Delta\varphi'_{(s+1)Rs}$ (unwrapped) in the synthesis part using Eq. (3.51).

$$\Delta\varphi'_{(s+1)Rs}[k] = \frac{R_s}{R_a} \cdot \Delta\varphi_{(s+1)Rs}[k] \tag{3.51}$$

The instantaneous frequency is preserved as the new delta phase increases, so will the hop size according to the ratio R_s/R_a. The delta phase, however, also needs to be added to the previous unwrapped phase φ'_{sRs} to maintain the phase increment as shown in Eq. (3.52).

$$\varphi'_{(s+1)R_s}[k] = \varphi'_{sR_s}[k] + \Delta\varphi'_{(s+1)Rs}[k] \tag{3.52}$$

We can thus finally use the results of Eq. (3.52) to compute the modified DFT frame's bin k component $X^d[k]'$ with the updated phase values via Eq. (3.53) and construct the time-domain signal through OLA of successive ISTFT frames.

$$X^d[k]'_{(s+1)R_s} = \left(|X[k]_{(s+1)Rs}|\right) \cdot e^{j\varphi'_{(s+1)Rs}} \tag{3.53}$$

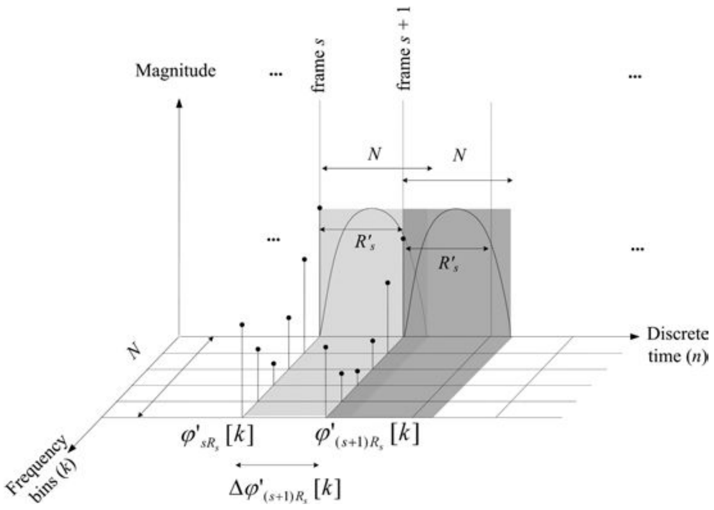


Fig. 3.38. ISTFT model for synthesis in time-stretching.

For the time-compression and the ISTFT, the procedure is the same as before, but instead of an increase of the delta phase, a decrease of delta phase results.

3.4.5.2 Pitch-shifting

Pitch shifting, which refers to the multiplication of every frequency component (rather than adding an offset value) by a certain transposition factor, can be quite straightforwardly achieved exploiting a time-shifting methodology followed by re-sampling as depicted in Fig. 3.39 (Dolson 1986; Zölner 2002). For example, if we were to octave shift a signal upwards in frequency, we would simply proceed by time-expanding (via the phase vocoder) the signal by a factor of two and resample the same signal at a ratio of $\frac{1}{2}$. In short, we do time-expansion by a factor of two followed decimation by two. The reason this works is because when we time-expand the signal, no pitch shifting takes place, yet, we have twice as many samples. And when we down-sample by two, pitch is increased by an octave but we return to the original signal length as time-shifting by two followed by down-sampling by the same ratio takes us back to the original signal size.

There are many enhancements and tweaks that can be done to improve the phase vocoder, including refinements such as ensuring that the sum of the squares of the overlapping windows is equal to one to get perfect reconstruction if no modification takes place at $R_a = R_s$. We will not cover

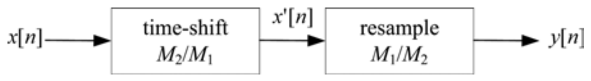


Fig. 3.39. Pitch-shifting via time-stretching and re-sampling.

the more eclectic intricacies and details regarding the phase vocoder here, but hope that the basic concepts outlined in this book will help the reader get started on a longer journey towards "phase-voco-lightenment."

4 Research Topics in Computer Music

As one can imagine there is a myriad of on-going research in computer music today, especially with the ever more powerful personal computers hitting the streets with multi-core processors facilitating simulation of models, testing of concepts, implementing old algorithms for real-time systems, and all sorts of explorations in sound generation and analysis. We will conclude the book with the introduction of a few recent interesting and exciting topics in computer music research.

4.1 *Salient feature extraction*

One area in computer music research that has lately attracted much attention is music information retrieval (MIR). This area of research extends over an interdisciplinary body of researchers focusing on topics concerning the retrieval of information from music and audio. Examples of such research include automatic genre classification, query (song) by humming, beat detection, and automatic musical instrument classification to name a few. We will introduce some of the MIR topics in the next section and discuss *salient feature extraction* in this section as it is a very important sub-research field in MIR and musical pattern recognition in general.

Salient feature extraction concerns with the study and development of algorithms to extract "important" information, characteristics, and features from a signal — in our case, audio signals. For example, if I were to flash my pen quickly in front of you for about a second and asked you to identify this object, you would probably be able to immediately tell me that it was some sort of pen. This is actually quite amazing, considering that you only saw the object for a very short time but were nevertheless able to identify the object — albeit maybe not being capable of telling me if it was a mechanical pencil, a fountain pen, or ball pen. The same holds true in situations for recognizing familiar musical instrument sounds such as the trumpet, or strumming of a guitar — even if we were to hear it very fleetingly, there is probably a good chance that we would be able to name the instrument and maybe even the tune. One of the reasons this is possible, is due to the salient features objects exhibit. For the pen example, we know from experience that a pen is of a certain size, has a cylindrical and lank shape and depending on the characteristics of the size itself (fatter or slimmer), we would also be able to guess whether it is a fine pointed pen or a thicker marker-type pen. Hence, by extracting salient features from objects, it is possible to identify or at least make an informed guess as to what we are seeing or hearing. In a sense, this approach may be viewed as a dimensionality reduction scheme or the reduction of the vast information to what is considered salient. One way of looking at salient feature extraction is by answering the question — "what are the fundamental qualities and features that help and enable us to identify a certain object?" and "what feature sets or feature vectors are sufficient in representing a certain object?" In this section, we will introduce some salient feature extraction techniques relevant to *sound objects*.

4.1.1 *Spectral envelope*

The *spectral envelope* embodies a wealth of information and can be seen as a zoomed-out view of the power spectrum (DFT magnitude values). What heavily determines the shape of the envelope is often the location of dominant peaks in the spectrum. One surprising finding by Grey was that through *line-segment approximation* of the harmonic amplitude envelopes (Grey 1977), he was able to get very good re-synthesis results of the original tones with a 100:1 data reduction — only by using partial information of each harmonic's amplitude trajectory (essentially decimation of the harmonic amplitude envelope) he was able to re-synthesize the original signal's timbre to a degree of it being recognizable as the original upon listening. On the distribution of the harmonics, it has been suggested that no harmonics higher than the 5th to 7th, regardless of the fundamental frequency, are resolved individually. Studies have also shown that the upper harmonics rather than being perceived independently are heard as a group (Howard, Angus 2001). Further support for this phenomena is made by Hartman who, according to Puterbaugh (Puterbaugh 1999), suggested that for a signal with fundamental frequency below 400 Hz, only the first 10 harmonics play an individual role: harmonics greater than 10 affect the timbre *en masse*. Numerous methods exist in determining the spectral envelope. One method is by salient peak detection of the power spectrum, another is taking the RMS of a STFT frame, yet another one is simply low-pass filtering of a DFT frame, or using LPC to describe it in terms of resonant pole structures as we have seen in Sec. 3.2.

4.1.2 *Spectral centroid*

The *spectral centroid* corresponds to a timbral feature that describes the *brightness* of a sound. This important feature has been elicited in the past (Helmholtz 1954; Lichte 1941) and experimentally observed in MDS-based (multidimensional scaling) investigations (Krumhansl 1989; McAdams *et al.* 1995; Lakatos 2000). The spectral centroid can be thought of as the center of gravity for the frequency components in a spectrum. It exists in many variations including its mean, standard deviation, square amplitudes, log amplitudes, use of bark frequency scale (Sekey, Hanson 1984), and the harmonic centroid (see appendix for details about bark frequency scale). During the lifetime of a sound, the centroid does not stay static but rather changes and furthermore, as one might expect, varies characteristically with intensity. For example, a trumpet blown in middle C with pianissimo and forte dynamics result in different spectral centroid characteristics. This feature is actually exploited indirectly in wavetable synthesis and other sound synthesis algorithms where filters are often used to make the same sample sound brighter or more dampened at various points in the waveform, mimicking the dynamicity of real instrument sounds. The spectral centroid is currently one of the MPEG-7 (Motion Picture Experts Group) timbre descriptors and is defined as shown in Eq. (4.1).

$$SC_{Hz} = \frac{\sum_{k=1}^{N-1} k \cdot X^d[k]}{\sum_{k=1}^{N-1} X^d[k]} \tag{4.1}$$

$X^d[k]$ is the magnitude corresponding to frequency bin k, N is the length of the DFT, and SC is the spectral centroid in Hertz. Generally, it has been observed that sounds with "dark" qualities tend to have more low frequency content and those with brighter sound dominance in high frequency (Backus 1976) which can be inferred by the value of the centroid.

It has also been suggested (Kendall, Carterette 1996) that the centroid be normalized in pitch, hence making the spectral centroid a unit-less and relative measure, since it is normalized by the fundamental frequency f_0. Some researchers have therefore utilized both the normalized and absolute versions of the centroid (Eronen, Klapuri 2000) in their research such as automatic instrument recognition (Park 2004).

$$SC_{\text{relative}} = \frac{\sum_{k=1}^{N-1} kX[k]}{f_0 \sum_{k=1}^{N-1} X[k]} \tag{4.2}$$

However, with the addition of f_o, it is only useful in the context of pitched sounds as purely percussive or inharmonic sounds usually are absent of a fundamental frequency. Furthermore, for obvious reasons, a precise pitch-extraction algorithm has to be devised for accurate centroid measurement which is not a trivial problem to solve.

4.1.3 *Shimmer and Jitter*

Shimmer and *jitter* refer to short-time, subtle irregular variations in amplitude and frequency of harmonics respectively. Shimmer is characterized by random amplitude modulation which is especially present for higher harmonics of instrument sounds. Jitter is a strong characteristic in instruments such as the string instrument family. For bowed strings, the unstable interaction between the bow and string — constant action of micro-level pulling and releasing, results in frequency jitter. Computing shimmer and jitter entails following the magnitude and frequency of harmonics over time which can be achieved by designing a robust harmonic tracker. This is, however, not trivial. An alternative approach for computing jitter and shimmer is using a 24 sub-band bark scale. In this case, each band generally corresponds to one harmonic, but if more than one harmonic is found in one of the 24 bands, the average of the harmonics and center frequency of that particular band is taken. Shimmer and jitter, which have characteristics of noise, are believed to have Gaussian normal distribution.

4.1.4 *Spectral flux*

The *spectral flux* defines the amount of frame-to-frame fluctuation in time. It is computed via the energy difference between consecutive STFT frames.

$$SF = |X_f^d[\cdot] - X_{f-1}^d[\cdot]| \tag{4.3}$$

$X[\cdot]$ denotes the magnitude components and superscript f and $f-1$ current and previous frames respectively. *SF* also sometimes referred to as the *delta magnitude spectrum*, has been used to discriminate speech and musical signals. It exploits the fact that speech signals generally change faster than musical signals, noting that in human speech there is a constant game of ping-pong being played between consonants and vowel sounds as we have seen in Sec. 3. In musical signals however, drastic changes tend to vary on a lesser degree.

4.1.5 *Log spectral spread*

The log spectral spread obtained as a salient dimension from multidimensional scaling experiments (Tucker 2001) is defined as the energy found between bounded pairs of upper and lower frequency boundaries. The lower and upper frequency bounds are found by applying a threshold value with respect to the maximum magnitude of the spectrum (e.g. −10 dB off the maximum) followed by locating the upper and lower frequency bounds, and finally taking the log of the result. The spread is somewhat similar to the spectral distribution found by Grey (Grey 1977) and is also compared to the richness of a sound. However, no attempts have been made to quantify this factor. Spectral spread, along with envelopes, may be helpful in observing timbral qualities of instruments such as the trombone, French horn, and tuba which generally lack in high frequency content, whereas the trumpet, primarily due to its brightness, is rich in upper harmonics (Howard, Angus 2001). The basic algorithm is shown in Fig. 4.1 with the final output representing the spectral spread between upper (u) and lower (w) frequency boundaries. The spectral spread has also been specified as one of the MPEG-7 audio descriptors.

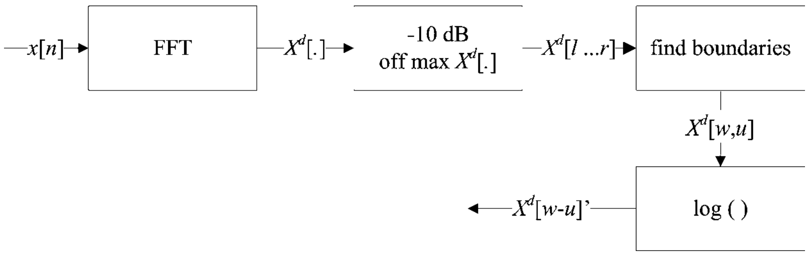


Fig. 4.1. Log spectral spread flow chart.

4.1.6 *Roll-off*

The *roll-off* point in Hertz is defined as the frequency boundary where 85% of the total power spectrum energy resides. It is commonly referred to as the *skew* of the spectral shape and is sometimes used in differentiating percussive and highly transient sounds (which exhibit higher frequency components) from more constant sounds such as vowels.

$$SR = R \tag{4.4}$$

$$\sum_{k=0}^{R} X^d[k] = 0.85 \sum_{k=0}^{N-1} X^d[k] \tag{4.5}$$

$X^d[k]$ are the magnitude components, k frequency index and R the frequency roll-off point with 85% of the energy.

4.1.7 *Attack time (rise time)*

The traditional *attack time* or *rise time* of a sound is simply defined as shown in Eq. (4.6) and is also used without the log operator. It computes the time between a start-point (determined via a threshold value) and the time location of the maximum amplitude of a sound (see also ADSR in Chap. 2). The log rise time has been found to be an important dimension in MDS and other timbral studies where it is often found as one of the three dimensions of a given MDS timbre space (Saldanha, Corso 1964; Scholes 1970; Krimphoff 1993; McAdams 1995; Lakatos 2000).

$$LRT = \log(t_{\mathrm{max}} - t_{\mathrm{thresh}}) \tag{4.6}$$

A slight variation for Eq. (6.27) also exists in the form of Eq. (4.7).

$$LRT = \log(t_{\mathrm{threshMax}} - t_{\mathrm{threshMin}}) \tag{4.7}$$

Here, the only difference is setting a threshold value for the maximum magnitude of a signal with a maximum threshold coefficient, sometimes set to 2% of the maximum magnitude (Misdariis, Smith, Pressnitzer, Susini, McAdams 1998). However, this approach, although sometimes better for some envelopes, is a static approach to a dynamic problem and may cause errors and inconsistencies when computing the attack time.

4.1.8 *Amplitude modulation* (*Tremolo*)

We have already seen amplitude modulation synthesis and recall that it basically consists of the product of two sinusoids. Amplitude modulation is omnipresent in musical tones, especially in performance situations and is often accompanied by frequency modulation, both being strongly coupled to each other (Backus 1976). The amplitude modulation frequency typically ranges from 4–8 Hz and is usually found in the steady-state portion of a

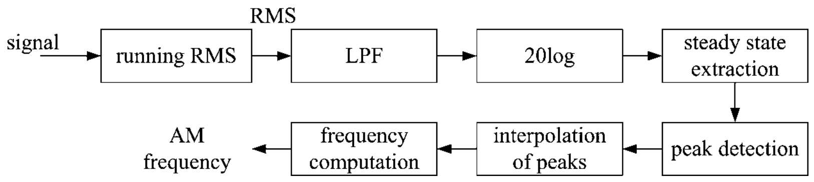


Fig. 4.2. Block diagram of AM extraction (Park 2000).

sound. It can be computed with fundamental frequency estimation methods by analyzing the sustain portion of the sound. For modulation frequencies between $9 \sim 40\,\text{Hz}$, a *roughness* and *graininess* perception is noted, which also contributes to the resulting timbral quality.

4.1.9 *Temporal centroid*

The *temporal centroid* has been found important as signal descriptors for highly transient and percussive sounds and basically denotes the center of gravity of the time-domain signal. It has been used along with log-attack time and spectral centroid in MDS experiments by Lakatos (Lakatos 2000). It is defined as the energy weighted mean of the time signal given in Eq. (4.8) and is also a part of the MPEG-7 audio descriptors.

$$TC = \frac{\sum_{n=1}^{L-1} nx[n]}{\sum_{n=1}^{L-1} x[n]} \tag{4.8}$$

$x[n]$ is the input signal, n time index and L the length of the signal. The algorithm is virtually the same as spectral centroid as both compute the center of gravity — spectral centroid in the frequency-domain and temporal centroid in the time-domain.

4.2 *MIR (Music information retrieval)*

Music information retrieval (MIR) has been a steadily growing research field accelerated by the digital revolution, changing forever the way we hear, store, interface, retrieve, access, and manage music whether it is audio recordings, music sheets, music articles or reviews. In a way, although digital technology has brought the music to each individual via the simple pressing of a button, it has also made the sharing, managing, and retrieving of music a nightmare due to the shear overflow of information and data. In 1999 10,000 new albums were released with registration of 100,000 new works during that year alone (Uitdenbogerd 1999). With basic home digital audio workstations becoming more and more ubiquitous (whether one is aware that one has it or not) and with MP3 players (MPEG-1 Audio Layer 3), MP3-enabled cell-phones, and smart phones becoming a common and even necessary accessory, it is not difficult to imagine what the numbers for newly released music is at present time! Navigating through this deluge of information is not a trivial matter and thus the interest in MIR has been growing since the late 1990s. Research in MIR is on-going and includes examples such as query-by-humming, automatic beat detection, automatic musical instrument recognition, automatic genre classification, automatic artist identification, automatic music analysis, and optical music recognition to name a few. In most, if not all of these research areas, salient feature extraction plays a critical role and many of the aforementioned algorithms are used in the research topics discussed in this section.

4.2.1 *Query-by-humming* (*QbH*)

Query-by-humming pertains to the automatic identification of a melody inputted into a machine (such as a computer) via *humming* and comparing the input query to an existing database with the artificial system providing the user with a ranked list of possible matches closest to the inquiry. A typical approach in designing such a system is via a bottom-up model structured around transduction of the audio signal (hummed melody by singer), extraction pitch information, and temporal information (onset, time events, segmentation). The next step is to pass these features to the melody analysis module whereby a melodic similarity measurement and classification algorithm against a database takes place. Typically, in the early stages of MIR, MIDI (Musical Instrument Digital Interface) files have been used to compute the melodic similarity (Kurth 2002; Pardo 2004; Ghias 1995). MIDI files are advantageous as they embed the exact melody (if correctly inputted of course), accurate pitches, rhythmic and temporal information as well as velocity information (comparable to dynamics) in digital format. In such a system, the query input is monophonic (single voice) as is the database itself. The importance of rhythm in QbH is a critical one and is a central parameter in determining the melody as published in a research paper where only rhythmic information was used (Jang *et al.* 2001) to determine the target melody. However, the best results seemingly are achieved by combining both temporal and frequency analysis. Once pitch extraction and segmentation is complete, the feature vectors are subject to pattern recognition modules such as hidden Markov models (Jang 2005), Gaussian mixture models (Marolt 2004), and adaptive boosting (Parker 2005). The main issues in QbH are, however, not only designing an accurate pitch extraction algorithm and rhythmic analyzer but also designing a system that is flexible enough in allowing errors by the singer (Kurth 2002). This addresses missing note and wrong note problems as well as tempo variations and rhythmic inexactitudes resulting from the humming of the melodies in the form of error tolerances.

4.2.2 *Automatic beat detection and rhythm analysis*

Another interesting area in MIR is the research in beat detection and rhythm analysis. Beat, rhythm, and tempo are all very important parameters in music and are often used to segment, group, and structure music, especially in the form of musical patterns from the perspective of time events, rather than pitch events. Interestingly, notated music, including tempo as well as rhythmic values of notes, are seemingly clear on paper but are in fact interpreted (sometimes very) differently by the individual performer or particular ensemble. Even though they may be reading from the exact same score, at times the interpretation and hence the music generated may be very different on a number of levels, especially in the area of temporal events — its tempo, rhythm, and beat. Additionally, even if one ensemble plays a piece in a set tempo to an audience of just two people, the perception of rhythm and tempo and metrical levels may actually be subtly different for the listener — the difference often depends on the knowledge of the piece beforehand (McKinney 2004). This is one of the main reasons it is a difficult task to design an artificial system that will robustly detect beat, rhythm and tempo, as human factors cause ambiguities to the resulting musical experience. There are numerous techniques currently being used to approach this problem. Some of the techniques include utilizing a single oscillator based on phase and period locking for piano pieces (Pardo 2004); utilizing frequency-domain analysis and spectral energy flux in combination with onset detection, periodicity detection, and temporal estimation of beat locations (Alonso 2004); applying Gaussian models in conjunction with correlation techniques, further combined with concepts from the area of psychology and gestalt theory (Frieler 2004); and distance-based algorithms such as k-means (Takeda 2004), Euclidian interval vector distance (Coyle 1998), Hamming distance (Toussaint 2004) and interval-difference distance measure of Coyle and Shmulevich (Coyle 1998), and the so-called swap distance (Orpen 1992).

4.2.3 *Automatic timbre recognition*

Musical timbre has always been a subject surrounded by ambiguity and mystery. As a matter of fact, it is still a term that experts in fields such as psychology, music, and computer science have a somewhat difficult time defining clearly. Numerous descriptions of timbre, especially in the past, have dealt with semantic descriptions such as cold, warm, cool, dull, short, long, smooth, rough ... etc., which, however, do not really reveal concrete and tangible information about its structure, dimension, or mechanism. One might be tempted to ask "we identify, recognize, and differentiate musical timbre quite easily, so why is it so hard to understand and design a machine to do it?" This is a valid point but in many cases, the things that humans do with little effort, such as smelling, touching, seeing, and recognizing objects, are the most difficult aspects to understand, and especially difficult to model on a computer system. As a matter of fact, it seems that in many cases, machines and humans have an inverse relationship when it comes to accomplishing perceptual tasks. Tasks that machines do effortlessly are done with greater difficulty by humans and vice-versa. Through continued research, we have witnessed much advancement bringing us a little closer towards understanding the complexities of timbre. So, what makes timbre so difficult to understand? The reasons of course are far too complex to put into one sentence, but the main problems underlying the complexity may be attributed to the following:

Timbre is a perceptual and subjective attribute of sound, rather than a purely physical one, making it a somewhat non-tangible entity.
Timbre is multidimensional in nature, where the qualities and importance of features, and the number of dimensions are not fully understood (we have already seen a number of timbral dimensions in Sec. 4.1).
There are no current existing subjective scales to make judgments about timbre.
There are no standard sets of sound examples against which the researcher can test their developed models.

The artificial recognition models for instruments and timbre in machines may be generally divided into two categories, the *top-down* model and the *bottom-up model.* The majority of the recognition research pertaining to

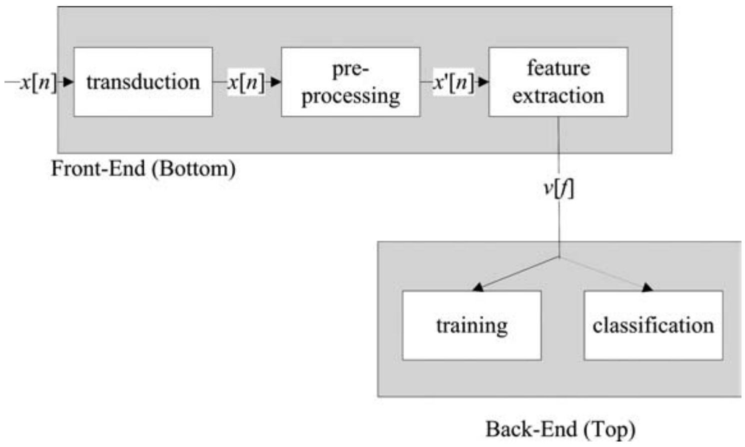


Fig. 4.3. Bottom-up model in timbre-classification.

timbre has used bottom-up models also known as *data-driven models* as shown in Fig. 4.3. Computer systems that implement machine recognition often apply some sort of machine learning technique categorized into *supervised* and *unsupervised learning*. There are various models and initial studies in automatic musical instrument timbre recognition including using k-nearest neighbors (Fujinaga 2000), Bayesian classifiers (Martin and Kim 1998), binary trees (Jensen and Arnspang 1999), artificial neural networks using radial and elliptical basis functions (Park 2004), independent subspace analysis (Vincent 2004), Gaussian mixture models (Essid 2004), as well as support vector machines (Marques 1999). As is true for most MIR systems, timbre recognition highly depends on the garbage-in-garbage-out (GIGO) paradigm and performance depends on robust feature extraction.

4.3 *FMS (feature modulation synthesis)*

The motivation for *feature modulation synthesis* research can be traced to the lack of synthesis/re-synthesis tools for composers who wish to directly deal with perceptually relevant dimensions of sound and be able to control and manipulate such dimensions in the form of modification of salient features (Park 2004; Park *et al.* 2007, 2008). Classic synthesis algorithms commonly generate sound from scratch or by using sonic templates such as sine tones or noise via control parameters that are often

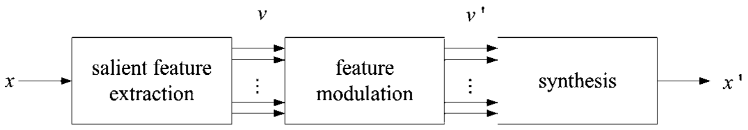


Fig. 4.4. Basic FMS architecture.

abstract and seemingly have little relationship to perceptually relevant nomenclatures. FMS tries to alleviate some of this disconnect between using abstract control parameters and the sound being produced by enabling control over timbre via salient timbral feature modulation. The architecture follows a synthesis-by-analysis model with salient feature extraction, feature modulation, and synthesis modules as shown in Fig. 4.4. x is the input sound object, x' synthesized result, v and v' the extracted and modulated feature vectors respectively. The fundamental approach to sound synthesis in FMS is to analyze a sound object for salient features, modulate a select number of these feature vectors, and synthesize a new altered sound object while keeping all other timbral features untouched. Although modulating one specific timbral dimension ideally should not affect any other features, in practice, artifacts occur. The first block in Fig. 4.4 (salient feature extraction) utilizes various time and frequency-domain-based analysis algorithms (Park *et al.* 2007, 2008). The second feature modulation block changes the desired features and corresponds to an important module in FMS as it contains algorithms that alter the analyzed feature profiles v to desired new feature profiles v'. The final block takes the modulated feature vector v' and synthesizes the modulated sound object via time and frequency-domain processes. Some examples of FMS include modulation of the analyzed spectral centroid, modulation of the RMS amplitude envelope, spectral spread alteration, harmonic expansion and compression, LPC-based noise (residual) modulation, as well as 3-D spectral envelope alteration.

To better understand how the system works, let's consider an example where we have a sound object like the electric bass guitar and want to change the brightness factor. It is usually the case that guitar strings sound brightest at the attack portion of the waveform and lose energy as time passes resulting in dampening — high frequency energy decays at a faster rate as we have seen in our physical modeling example in Chap. 7. By analyzing the signal for its spectral centroid, we can measure the amount of brightness of the waveform as a function time. Thus, by modifying each spectral frame via multiplication of the spectrum by an appropriate

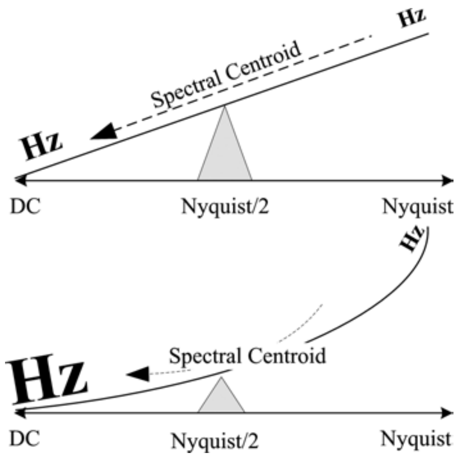


Fig. 4.5. Spectral multiplication via "see-saw".

envelope, we can alter the spectral centroid and change the brightness factor of the synthesized signal. The approach for modulating the spectral centroid follows the mechanics of a see-saw. That is, more energy towards the lower frequencies will result in the see-saw being tilted towards the DC and more energy in the high frequency area will result in tilting towards the Nyquist. The see-saw is thus initially balanced at Nyquist/2 frequency and tilts either towards the DC or Nyquist depending on the desired centroid location during modulation as shown in Fig. 4.5. The shape of the see-saw thus forms the multiplying envelope used to shift the spectral centroid either towards DC or Nyquist. Another example of FMS is found when performing noise analysis of a sound object by computing the residual signal using LPC (see Sec. 3.2). The residual signal is ideally noise-based and hence can be used to represent the noisiness factor of a waveform. Since we have the noise portion of a sound object, we can modulate the noise level by exploiting this source-filter model and control aspects of noise content of a particular signal.

5 Musical Examples

We can probably imagine why vocoders have had such popularity in musical applications — not only due to the wide possibilities affording the musician to flexibly manipulate sound but perhaps more so because the voice itself is a sound that most of us are in one way or another very familiar with.

Numerous composers and artists have used the vocoder spanning from *Kraftwerk* with *Die Roboter* (1978), Laurie Anderson and *O Superman* (1981), Herbie Hancock in *Rock It* (1983), Cher's *Believe* (1999), *Daft Punk's One More Time* (2000), and *Snoop Dog's Sensual Seduction* (2007). Some of the pioneers in computer music who have utilized vocoders extensively include Charles Dodge and Paul Lansky. Charles Dodge's work with LPC started at Bell Labs in the 1960s and cumulated to the release of *Speech Songs* in 1972. Working at the lab when the speech researchers had left for the night, Dodge would experiment with state-of-the-art speech technology (at that time) and utilize it for his compositional ideas — it was not meant to be used in that context. *A Man Sitting in the Cafeteria* is an example of a collection of 4 pieces from *Speech Songs.* One can clearly hear the manipulation of voiced and unvoiced parts of the speech that is inputted to the computer; in essence making the normally non-musical perceived speech into a musical one — albeit not in the traditional sense of music for voice. It is undeniably synthetic and perhaps even pleasantly humorous at times as the composer modulates various aspects of the text/voice including the tempo, rhythm, pitch, and noise parts in an unexpected manner. The text goes something like this:

A man sitting in the cafeteria
had one enormous ear
and one tiny one
Which was fake?

Paul Lansky also conducted a wealth of research in the area of LPC and explored its application in musical compositions, the most notable ones being the *Idle Chatter* collection of pieces. The first one of these pieces is called *Idle Chatter* (1985) and takes the listener on a journey of the world of *sound poetry* and speech. The composer uses speech and obscures the informational part of its multidimensional structure — the listener knows that the source is speech but is left to grapple with what is actually being said. On one level, when viewed from a phonetic rather than a semantic angle of using the voice in a musical context, the piece can be regarded as an experience of sonic sequences of highly rhythmic voice patterns, engulfed in a rich harmonic language. On another level, the voice materials, both used as percussive and melodic devices, walk the grey area between speech as timbre and speech as dialect. Other pieces that are on the opposite end of the pole are pieces like my own called *Omoni* (2000) and *Aboji* (2001) which deal with the narrative aspect of speech. In those types of works, excerpts of interviews, often abstract, incomplete, and modulated in various ways, make up the sound material as well as the informational material which comprise the complex gestalt of words, poetry, and musical composition. In a way, at least from my own perspective, the composer carefully negotiates his role as composer vs. reporter and creates something that walks the fine line between composition and documentation. The last piece I'll introduce before closing this chapter and this book is Andy Moorer's *Lions are Growing* (1978). This piece is based on poems from Richard Brautigan's *Rommel Drives on into Egypt.* The work is available on CD entitled *The Digital Domain: A Demonstration* (1983). Unlike the Paul Lansky and Charles Dodge pieces which used LPC, Moorer utilized the phase vocoder to manipulate and transform textures, formant structures, pitch, and aspects of time. You will probably hear the resemblance to Charles Dodge's *Speech Songs* as the material in Moorer's piece is also solely made from the synthesized voice injected with an updated speech synthesis algorithm six years later — any resemblance is not purely coincidental (at least as far as the fundamental role of the vocoder is concerned). The idea itself, generally speaking, is somewhat similar to Dodge's piece, but the sound quality is remarkably on a higher level and of course synthesized via a very different method. One of the most humorous and enjoyable moments (there are many from smooth glissandi to slick time-shifting), is at the end of piece — as Elvis would say, "thank you, thank you (very much)." Thank you, thank you very much indeed!

References and Further Reading

Alonso, M., B. David and G. Richard 2004. "Tempo and Beat Estimation of Musical Signals," in ISMIR 2004 Fifth International Conference on Music Information Retrieval Proceedings, pp. 158–163.

Backus, J. 1976. "Input Impedance Curves for the Brass Instruments". Journal of the Acoustical Society of America, 60, pp. 470–480.

Bisnhu, Atal, M. R. Schroeder 1967. "Predictive coding of speech," in Proc. Conf. Communications and Proc., Nov. 1967, pp. 360–361.

Bisnhu, Atal, M. R. Schroeder 1970. "Adaptive predictive coding of speech," Bell Syst. Tech. J., Vol. 49 No. 8, pp. 1973–1986, Oct. 1970.

Bogert, B. P., M. Healy and J. Tukey 1963. "The quefrency alanysis of time series for echoes: Cepstrum, pseudo-autocovariance, cross-cepstrum, and saphe cracking," in Time Series Analysis, M. Rosenblatt (Ed.) 1963, Ch. 15, pp. 209–243.

Coyle, E. J. and I. Shmulevich 1998. "A system for machine recognition of music patterns". In Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing, Seattle, Washington.

Childers, D., D. Skinner and R. Kemerait 1977. "The Cepstrum: A Guide to Processing", Proceedings of the IEEE, Vol. 65, No. 10, pp. 1428–1443.

Cuadra, P., A. Master and C. Sapp 2001. "Efficient Pitch Detection Techniques for Interactive Music", Proceedings of the 2001 ICMC, Havana

Dolson, M. 1986. "The phase vocoder: a tutorial," Computer Music Journal, Spring, Vol. 10, No., 14–27.

Doyle, M. "Private communication." October 13, 2000. The extent of actual use by Roosevelt and Churchill is still unclear and needs further research to clarify; however, there is evidence of a call between Truman and Churchill on VE-Day (Victory in Europe).

Dudley, H. 1938. "System for the Artificial Production of Vocal and Other Sounds (Voder)," U.S. patent 2,121,142.

Dudley, H. 1939. "The Vocoder", Bell Labs. Rec., Vol. 17, pp. 122–126.

Dudley, H. 1939. "The Vocoder", Journal of Acoustical. Society of America., Vol. 11, No. 2, p. 169

Eronen and A. Klapuri 2000. "Musical Instrument Recognition Using Cepstral Coefficients and Temporal Features". In Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2000, pp. 753–756.

Essid, S., G. Richard and B. David 2004. "Musical instrument recognition based on class pairwise feature selection" Proceedings of ISIMIR 2004, Barcelona, Spain.

Flanagan, J. and R. Golden 1966. "Phase vocoder". Bell System Technical Journal, 45, 1493–1509.

Flanagan, J. 1972. "Voices of men and machines",' Journal of the Acoustical Society of America, Vol. 51, pp. 1375–1387

Frieler K. 2004. "Beat and meter extraction using Gaussified onsets". Proceedings of ISIMIR 2004, Barcelona, Spain.

Gihas A. 1995. "Query By Humming — Musical Information Retrieval in an Audio Database". ACM Multimedia 95, Electronic Proceedings November 5–9, 1995 San Francisco, California

Grey, J. M. 1977. "Multidimensional Perceptual Scaling of Musical Timbre". Journal of the Acoustical Society of America Vol. 61, pp. 1270–1277.

Helmholtz, H. L. 1954. On the Sensation of Tone as a Physiological Basis for the Theory of Music (translation of original text 1877), New York: Dover Publications.

Herrera, P., X. Amatriain, E. Batlle and X. Serra 2000. "Towards instrument segmentation for music content description: a critical review of instrument classification techniques", in International Symposium on Music Information Retrieval, pp. 23-25, Plymouth, Mass, USA.

Howard, D. M. and J. Angus, 2001. Acoustics and Psychoacoustics. Oxford, Boston: Focal Press.

Jensen, K. and J. Arnspang 1999. "Binary Decision Tree Classification of Musical Sounds". In Proceedings of the 1999 International Computer Music Conference.

Jang, J., C. Hsu and C. Yeh 2001. "Query by Tapping: A New Paradigm for Content-Based Music Retrieval from Acoustic Input", Proceedings of the Second IEEE Pacific Rim Conference on Multimedia: Advances in Multimedia Information Processing, pp. 590–597, October 24–26.

Jang, J., C. Hsu and H. Lee 2005. "Continuous HMM and Its Enhancement for Singing/Humming Query Retrieval". Proceedings of the ISMIR 2005 Conference, pp. 546–551.

Kondoz, Ahmed 2004. Digital Speech: Coding for Low Bit Rate Communication Systems, Wiley.

Krimphoff, J., S. McAdams and S. Winsberg 1994. "Characterisation du Timbre des Sons Complexes. II Analyses acoustiques et quantification psychophysique". Journal de Physique IV, Vol. 4, pp. 625–628.

Kurth, F. 2002. "Efficient Fault Tolerant Search Techniques for Full-text Audio Retrieval". Proc. 112th AES Convention, Munich, Germany.

Lakatos, S. 2000. "A Common Perceptual Space for Harmonic and Percussive Timbres". Perception & Psychophysics 62(2), pp. 1426–1439.

Lichte, W. H. 1941. "Attributes of Complex Tones". Journal of Experimental Psychology, 28, pp. 455–480.

Marolt, M. 2004. "Gaussian Mixture Models for Extraction of Melodic Lines From Audio Recordings", Proceedings of ISIMIR 2004, Barcelona, Spain.

Martin, K. and Y. Kim 1998. "Musical Instrument Identification: A Pattern-Recognition Approach", 136th meeting of the JASA.

McAdams, S., S. Winsberg, S. Donnadieu, G. De Soete and J. Krimphoff, 1995. "Perceptual Scaling of Synthesized Musical Timbres: Common Dimensions, Specificities, and Latent Subject Classes", Psychological Research 58, 177–192.

McKinney, M. 2004. "Extracting the perceptual tempo from music", Proceedings of ISIMIR 2004, Barcelona, Spain.

Misdariis, N., B. K. Smith, D. Pressnitzer, P. Susini and S. McAdams 1998. "Validation of a Multidimensional Distance Model for Perceptual Dissimilarities among Musical Timbres", 16th International Congress on Acoustics and 135th Meeting Acoustical Society of America, Seattle, Washington.

Moorer, J. A. 1975. "On the Segmentation and Analysis of Continuous Musical Sound by Digital Computer",' Doctoral Dissertation, Department of Computer Science, Stanford University.

Kendall, R. A. and E. C. Carterette 1996. "Difference Threshold for Timbre Related to Spectral Centroid", Proceedings of the 4th International Conference on Music Perception and Cognition, pp. 91–95.

Orpen K. and D. Huron 1992. "Measurement of similarity in music: A quantitative approach for nonparametric Representations", In Computers in Music Research, Vol. 4, pp. 1–44.

Pardo, B. 2004. "Tempo Tracking with a Single Oscillator". Proceedings of ISIMIR 2004, Barcelona, Spain.

Park, T. H. 2000. "Salient Feature Extraction of Musical Signals", Master's Thesis. Dartmouth College, Electro-acoustic Music Program.

Park, T. H. 2004. "Towards Automatic Musical Instrument Timbre Recognition", Ph.D. dissertation, Princeton University.

Park, T. H., Z. Li and J. Biguenet 2008. "Not Just More FMS: Taking it to the Next Level", In Proceedings of the 2008 ICMC Conference, Belfast, Ireland.

Parker, C. 2005. “Applications of Binary Classification and Adaptive Boosting to the Query-by-Humming Problem”. Proceedings of the ISMIR 2005 Conference.

Puterbaugh, J. D. 1999. “Between a Place and Some Location, A View of Timbre through Auditory Models and Sonopoietic Space”, Ph.D. Dissertation. Princeton University Music Department.

Rabiner, L. R. and R. W. Schafer 1978. “Digital Signal Processing of Speech Signals”, Prentice Hall, Signal Processing Series.

Scholes, P. A. 1970. The Oxford Companion to Music. London: Oxford University Press.

Schroeder, M. R. 1968. “Period Histogram and Product Spectrum: New Methods for Fundamental Frequency Measurements”, Journal of Acoustical Society of America, Vol. 43, No. 4, pp. 829–834.

Sekey, A. and B. A. Hanson, 1984. “Improved 1-bark Bandwidth Auditory Filter”, JASA Vol. 75, No. 6.

Takeda, H. 2004. “Rhythm and Tempo Recognition of Music Performance from a Probabilistic Approach”, Proceedings of ISIMIR 2004, Barcelona, Spain.

Toussaint, G. 2004. “A Comparison of Rhythmic Similarity Measures”, Proceedings of ISIMIR 2004, Barcelona, Spain.

Truax, Barry 1978. Handbook for Acoustic Ecology. A.R.C. Publications, Vancouver,

Tucker, S. 2001. “Auditory Analysis of Sonar Signals”, Ph.D. Transfer Report. Department of Computer Science. University of Sheffield.

Uitdenbogerd, A. and J. Zobel, 1999. “Matching techniques for large music databases”, Proceedings of the 7th ACM International Multimedia Conference, pp. 57–66.

Vincent, E. and X. Rodet 2004. “Instrument identification in solo and ensemble music using Independent Subspace Analysis”, Proceedings of ISIMIR 2004, Barcelona, Spain.

Zölner, Udo 2002. DAFX — Digital Audio Effects. Editor, John Wiley and Sons.

Zwicker, E. and H. Fastl 1999. Psychoacoustics Facts and Models. Springer Verlag.

APPENDIX

1 To Scale or Not to Scale: That is the Question

When we hear frequencies and tones, the perception of those frequency components is usually not linear as we have already learned in Chapter 1. Thus, a number of different types of scales exist that address the issue of nonlinearity of frequency and its perception. We will present a number of representative scales here.

1.1 *Equal temperament scale*

The most common type of frequency scale is probably the 12-tone equal temperament scale. It is the tuning system adopted by the majority of musicians today, especially in the Western world which has the advantage of permitting modulation (key change) without affecting the relative relationship between tones. In short, the distance between each note (interval) is structured so that each pair of adjacent notes forms an equal frequency ratio according to Eq. (1.1).

$$f_x = f_r \cdot 2^{n/12} \tag{1.1}$$

In essence, what Eq. (1.1) tells us is that the octave is divided into 12 "equal" parts where the integer n determines the frequency (notes) between a note and its octave equivalent. If you look at n and start with $n = 0$ and increment to 12, you will find that the reference note f_r is multiplied by $1, 2^{1/12}, \ldots, 2^{11/12}$, 2 where the last multiplication refers to the octave above f_r. The ratio between two semitones (adjacent integer values of n) is equal to one cent, where 100 cents make up one semitone interval as shown in Eq. (1.2).

$$cent = 2^{1/12} \tag{1.2}$$

A common reference frequency is 440 Hz which denotes the A4 note or the 49th key when counted in a pitch ascending manner on a standard piano. Thus, finding/computing equal temperament notes is quite straightforward. For example, the middle C4 (261 Hz) is computed as shown in (1.3) as the C4 note is 9 "notes" below A4:

$$\begin{aligned} f_x &= f_r \cdot 2^{n/12} \\ &= 440 \cdot 2^{\frac{-9}{12}} \\ &= 261.6\,\text{Hz} \end{aligned} \tag{1.3}$$

1.2 *Just intonation*

Just intonation is a scale/tuning system that relies on ideal harmonic relationships where the frequencies of the notes are a ratio of integer numbers (rational numbers). Unlike the equal temperament scale, with just intonation, it is necessary to retune the instrument for the scale that you wish to use — an *A Major* tuning setup will not be applicable for a *B Major* setup which makes changing keys problematic for instruments that have fixed notes such as a piano or fretted stringed instrument. Thus, in a sense, just intonation is actually more accurate than the equal temperament system as it adheres to the harmonic frequency structure though the equal temperament system is now used widely around the world in musical and non-musical communities alike (you will rarely hear any other tuning system in popular music other than the equal temperament system).

1.3 *Bark scale*

The *bark frequency scale* is a nonlinear frequency scale which maps frequencies in Hertz to 24 bands of a special psychoacoustic unit called the bark. The bark scale adheres close to the human nonlinear hearing system with one bark corresponding to the width of one critical band. 24 critical bands with center frequencies and their corresponding band edges are shown in Table 1.1. A number of formulas exist in computing the bark

Table 1.1. Bark frequency scale, center frequency and band edges.

Bark Unit	Bark Center Frequencies	Band edges
1	50	0, 100
2	150	100, 200
3	250	200, 300
4	350	300, 400
5	450	400, 510
6	570	510, 630
7	700	630, 770
8	840	770, 920
9	1000	920, 1080
10	1170	1080, 1270
11	1370	1270, 1480
12	1600	1480, 1720
13	1850	1720, 2000
14	2150	2000, 2320
15	2500	2320, 2700
16	2900	2700, 3150
17	3400	3150, 3700
18	4000	3700, 4400
19	4800	4400, 5300
20	5800	5300, 6400
21	7000	6400, 7700
22	8500	7700, 9500
23	10500	9500, 12000
24	13500	12000, 15500

scale and one of them is shown in Eq. (1.4) (Zwicker and Terhardt 1980).

$$B = 13\tan^{-1}\left(\frac{0.76f}{1000}\right) + 3.5\tan^{-1}\left(\frac{f}{7500}\right)^2 \tag{1.4}$$

1.4 *Mel scale*

Like the bark scale, the *mel frequency scale* also follows a perceptual model for frequency mapping where the results reflect a scale that follows perceptually equidistant frequencies. Mel describes a nonlinear mapping of pitch perception as a function of frequency in Hertz according to Eq. (1.5) where m is the mel frequency and f frequency in Hertz (Stevens and Volkman 1937, 1940)

$$m = 1127.01048\log_e\left(1 + \frac{f}{700}\right) \tag{1.5}$$

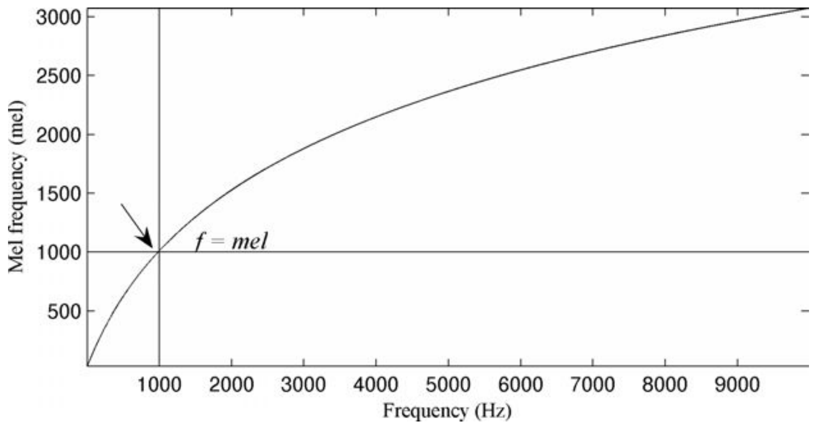


Fig. 1.1. Mel frequency to Hertz for $f = 20$ to 10 kHz.

As we can see from Eq. (1.5), f is equal to m when $f = 1\,\text{kHz}$ — the reference point between mel frequency and frequency in Hertz occurs at 1 kHz where 1 kHz is equal to 1000 mels (the reference dB is 40 dB above the threshold of hearing). The name mel comes from the musical word melody and addresses the frequency part of melody and its pitch related perceptual distance measurement.

We can of course also go from the mel scale back to the frequency (Hz) scale by performing an exponential power operation on both sides of Eq. (1.5) resulting in (1.6).

$$f = 700 \cdot e^{\frac{m}{1127.01048}} - 1 \tag{1.6}$$

2 MATLAB® Programs Used in This Book

In this section, I have included information regarding the MATLAB® programs used to make the majority of the plots, algorithms, and examples throughout the book. I hope that the source code will be helpful in further digging into the concepts presented in this book through implementation and hacking. The programs are downloadable from http://music.princeton.edu/∼park/dsp-book.

Table 2.1. MATLAB® programs used in this book.

Chapter	Section	Description	Program name (.m file)
1	2	Sine wave Fig. 2.2	sineWave.m
	2	Make sine wave sound Code Example 2.1	sineWaveSound.m
	4	Sampling Figs. 4.1–4.4	sineWave.m
	4	Sampling and aliasing Figs. 4.5 $\sim$ 4.11	samplingNAliasing.m makeSine.m
	5	Quantization of amplitude Fig. 5.2	quantizationPlot.m
	6	DC signal Fig. 6.1	dcPlot.m
	6	DC offset removal Code Example 6.1	dcOffsetRemoval.m
	7	Distortion and clipping Fig. 7.1	distortionPlot.m
	7	Quantization and dithering Fig. 7.2	ditherPlot.m
	7	Harmonic distortion Fig. 7.3	ditherHarmonicDistortion1.m
	7	Harmonic distortion Fig. 7.4	ditherHarmonicDistortion2.m
2	2	Waveform and envelope Figs. 2.2, 2.3	rmsPlots.m rms.m
	2	Reverse playback Code Example 2.1	reverseWave.m
	3	Wavform views Fig. 3.1	waveformSubplots.m
	3	Looping Fig. 3.2	waveformLoop.m
	4	Windows Figs. 4.1, 4.3	windowingPlots.m
	5	Zero crossings Fig. 5.1	zcrPlot.m
	5	Zero crossings Figs. 5.2, 5.4	zcrDemo.m
	6	Up-sampling Fig. 6.1	upSampling.m
	6	Down-sampling Fig. 6.2	downSampling.m
	6	Down/up sampling Code Example 6.1	upDownSamplingEx.m
	7	OLA Figs. 7.1, 7.2	olaPlots.m
3	2	Grain and windowing Figs. 2.1, 2.3	granularWindows.m granularSynthesis.m
	3	Dynamic compressor Code Example 3.1	compressor.m
	5	Equal loudness panning Fig. 4.1	panning.m

(*Continued*)

Table 2.1. (*Continued*)

Chapter	Section	Description	Program name (.m file)
4	4	Beating Fig. 4.1	beating.m
	4	Amplitude modulation Fig. 4.4	amplitudeModulation.m
	5	Bessel function Fig. 5.2	bessel.m
5	4	Impulse Fig. 4.1	impulsePlot.m
6	2	Frequency response types Fig. 2.4	freqLinearLogBode.m
	2	Frequency response LPF Fig. 2.5	freqIirLpf.m
	3	Magnitude/phase response Fig. 3.1	freqFirLpf.m
	3	Magnitude/phase response Fig. 3.3	phaseResponse.m
	3	Phase distortion Fig. 3.5	linearVsNonLinearPhase.m
	3	Group delay Fig. 3.6	groupDelay.m
7	2	Filter characteristics Figs. 2.5 ∼ 2.14, 2.16 ∼ 2.27	filterDesign.m
	3	Subtractive synthesis Code Example 3.1	subtractiveSynthesis.m
	3	Biquad BPF Code Example 3.2 Figs. 3.3, 3.4	biquadBpf.m
	3	Comb-filter coefficients Fig. 3.7	combFilterCoeff.m
	3	Comb-filer z-plane Figs. 3.8 ∼ 3.10	combFilterZplane.m
	3	Comb-filtering 1 Code Example 3.3	combFilterExample1.m
	3	Comb-filtering 2 Code Example 3.4	combFilterExample2.m
	3	Comb-filtering 3 Code Example 3.5	combFilterExample3.m
	3	Plucked string model Code Example 3.6 Figs. 3.22, 3.23	pluckedStringKS0.m
	3	Direct implementation Code Example 3.7	pluckedStringKS1.m
	3	Code Example 3.8	chorus.m
	3	All-pass filter z-plane Fig. 3.33	allpassFreq.m
	3	All-pass impulse Fig. 3.38	allpassImpulse.m

(*Continued*)

Table 2.1. (*Continued*)

Chapter	Section	Description	Program name (.m file)
	4	All-pass impulse examples Figs. 3.39 ∼ 3.41	allpassImpulses.m
	4	Basic all-pass reverb Code Example 3.10	allpassReverb.m
8	2	Additive synthesis Figs. 2.1 ∼ 2.3, 4.2	squareWave.m
	4	Fourier basics Figs. 4.2, 4.4, 4.6 ∼ 4.8,	fourierBasics.m
	6	Zero padding Figs. 6.1, 6.2	zeroPadding.m
	7	Aliasing Figs. 7.2 ∼ 7.4	samplingNAliasing2.m
	8	Upsampling Fig. 8.6	upsamplingFreqDomain.m
	9	Windowing rectangular Figs. 9.1 ∼ 9.7	windowingDftRectSinusoids.m
	9	Window types Figs. 9.8 ∼ 9.15	windowTypesDft.m
	9	Sinc functions Figs. 9.16 ∼ 9.19	sincPlots.m
	11	Circular convolution plots Figs. 11.1 ∼ 11.3	convolutionPlots.m
	12	Dithering Figs. 12.1, 12.2	ditherHarmonicDistortion2.m
	13	Spectrogram plot Fig. 13.1	specgramPlot.m
	15	MATLAB® and FFT Code Example 14.1	fftExample.m
9	2	LTAS Fig. 2.2	ltas.m
	2	FFT log vs. linear Figs. 2.4, 2.5	fftLogVsLinear.m
	2	Peak valley analysis Fig. 2.7	peakValleyAnalysisMain.m peakValleyAnalysis.m
	2	Peak valley RMS Fig. 2.8	peakValleyRmsMain.m peakValleyAnalysis.m
	2	Salient peak picking Fig. 2.9	findPeaks hillClimbing.m
	2	Inverse comb filtering Figs. 2.11, 2.12	inverseCombFilterPlots.m
	2	Inverse comb filter pitch Fig. 2.13	inverseCombFilterPitch.m
	2	Cepstral pitch analysis Fig. 2.15	cepstrum.m
	3	Envelope follower Fig. 3.2	envFollower.m

(*Continued*)

Table 2.1. (*Continued*)

Chapter	Section	Description	Program name (.m file)
	3	Bark scale bands Fig. 3.3	channelVocoder.m
	3	Enveloper follower results Fig. 3.4	channelVocoder.m
	3	ZCR voiced/unvoiced Figs. 3.8 ∼ 3.10	zcrAnalysis.m zcrComputation.m
	3	Pre-emphasis voiced/unvoiced Fig. 3.11	preEmphasisVuv.m
	3	Low-band full energy voiced/unvoiced Fig. 3.12	lowFullEnergyRatioVuv.m
	3	SFM analysis voiced/unvoiced	sfmVuv.m sfm.m
	3	LPC Fig. 3.17	lpcAnalysis.m
	3	Phase unwrapping Fig. 3.22	phaseUnwrapping.m
Appendix	1	Fig. 1.1 mel frequency vs. Hz plot	melVsHzPlot.m
		Book Cover	bookCover.m

References and Further Reading

Stevens, S. S., Volkman, J. and Newman, E. B. 1937. "A Scale for the Measurement of the Psychological Magnitude of Pitch", Journal of the Acoustical Society of America, 8.

Stevens, S. S., Volkman, J. and Newman, E. B. 1940. "The Relation of Pitch to Frequency, A Revisited Scale", Journal of the Acoustical Society of America, 53.

Zwicker, E. and Terhardt E. 1980. "Analytical Expressions for Critical-band Rate and Critical Bandwidth as a Function of Frequency", Journal of the Acoustical Society of America 68, 1523–1525.

Index