CS425 Audio and Speech Processing

Matthieu Hodgkinson
Department of Computer Science
National University of Ireland, Maynooth

April 25, 2012

Contents

Chapter 0

Introduction : Speech and Computers

As a means of introduction to the topic, I give here a list of how computers handle speech signals, and what can be done with them. We will see thereafter how this module relate to that.

1. Recording brings a digitised representation of the speech waveform in the computer. As everything in a computer, this representation can ultimately be broken down to ones and zeros, or *bits*. A certain number of bits – this number is called *bit-depth* – is used to represent the amount of displacement from the 0-axis, or *amplitude*, at a given instant in time. Audio CDs, for example, use 16 bits for that purpose, which yields a vertical scale of $2^{16} = 32,768$ different values, more than enough to bluff the human ear. This is nevertheless for one *sample* only of the overall *waveform*. To represent, say, only one second of a waveform, thousands of such samples are needed, evenly spaced at a regular interval of time, the *sampling period*. The inverse of this period – in other words, the number of samples per second – is the *sampling frequency*. To stick with the previous example, audio CDs use a sampling frequency of 44,100kHz.

   The process of taking samples of a waveform is called *sampling*, and that of rounding its values to the nearest on the "bit scale", *quantisation*. These processes are illustrated in the lower and upper plots, respectively, of Figure 1. There, the sampling frequency and bit-depth were set very low, for illustrative purposes.

   The digital medium has tremendous advantages in the recording of audio. Large quantities of data can be stored on small devices (e.g. compare the capacity and size of a USB stick nowadays with those of an analogue tape, or an old record). Also, the reproduction of digital audio is transparent, meaning that no noise is added. And in such format, of course, audio can be transferred via the internet.

2. Analysis reads and performs operations with the values of the speech waveform to yield specific pieces of information on this waveform. Low-level analysis techniques can tell whether a speech sound is *voiced* (involvement of the vocal cords) or *unvoiced* (only an air flow excites the vocal tract), what the pitch of a voiced sound is, whether anything is being spoken at a given instant in time or if it is only silence, etc. Higher-level analysis techniques (which often rely on lower-level techniques as well) can lead to the recognition of syllables, words and sentences ; a

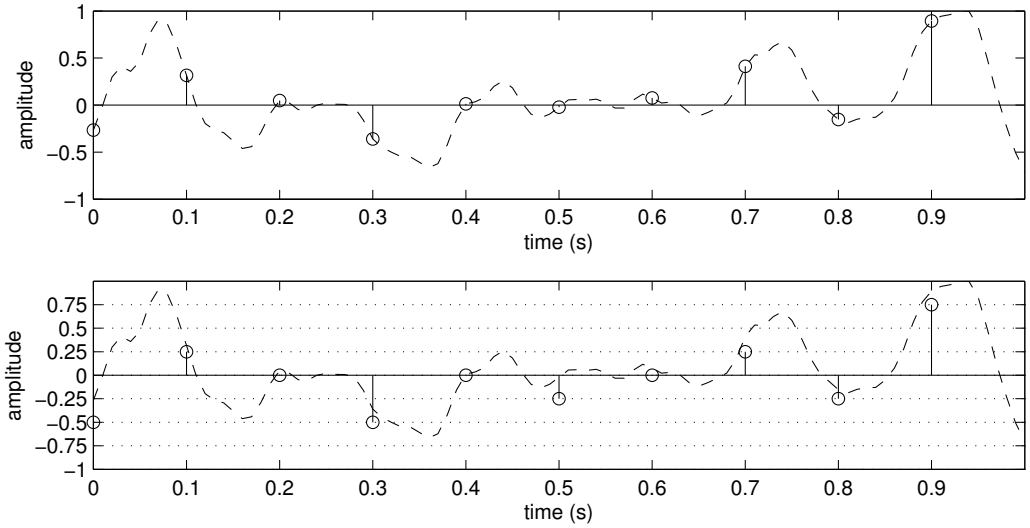

Figure 1: *Continuous waveform (dashed line) sampled (upper plot stems) and quantised (lower plot stems). The sampling period is here 0.1 second, hence the sampling frequency 10Hz, and the bit-depth, 3.*

discipline called *speech recognition*. The recognition of the *gender* of the speaker is one potential outcome of analysis [Wu and Childers, 1991], and one might foresee the retrieval of other pieces of information further down this road : age, physical condition, or anything you find you can tell about a person just from hearing this person speaking. Maybe some day computers will be able to tell just as much. Or more.

3. Processing is often assimilated to analysis, for instance in the label of this *Audio and Speech Processing* module. Other than that, processing is meant as the modification of a speech signal, to produce another speech signal of altered qualities. Here is a little sample list of examples of speech processing ; they were all performed upon this original speech waveform [1] :

   - robotisation (*play* [2]) works by phase-synchronising the frequency

[1] `http://www.cs.nuim.ie/~matthewh/CS425/2012/sf/High%20Speed%20Commercial.wav`

[2] `http://www.cs.nuim.ie/~matthewh/CS425/2012/sf/High%20Speed%20Commercial%20-%20robotisation.wav`

components of the sound to a clock, but leaving the spectral distribution untouched. Thus, the waveform becomes nearly-periodic and shows a steady pitch, but its timbre remains untouched so the words can still be made out.

- whisperisation (*play* [3]) is similar to robotisation, to the exception that the phases of the frequency components is now scambled. Still, the timbre remains, but now any sense of pitch is lost.
- cross-synthesis of the original crossed with this pitch-synchronous pulse train [4] gives *this* [5]. The phase information (and hence, the pitch quality) of the buzz waveform is used in combination with the spectral distribution (and hence, the timbre and the words) of the speech waveform, creating this musical speech.
- convolution of the original with this telephone horn's impulse response [6] gives *this* [7]. When one talks in or through an acoustic system (such as a room), one also hears the response of the system to the talking. This phenomenon is well modeled through *convolution*, a straightforward and yet efficient technique to impart the "acoustic" qualities of a system onto an input.

These are all musical application examples of speech processing, but there are applications other than musical, such as the alteration of the voice of persons who want to keep anonymity. Can you think of any other?

4. The synthesis of speech is possible – although in most cases synthesized speech can be distinguished from natural speech. There exist several ways and techniques of making the computer "talk", the foremost two being reported to be *concatenative synthesis* and *formant*

[3] `http://www.cs.nuim.ie/~matthewh/CS425/2012/sf/High%20Speed%20Commercial%20-%20whisperisation.wav`

[4] `http://www.cs.nuim.ie/~matthewh/CS425/2012/sf/High%20Speed%20Commercial%20-%20buzz.wav`

[5] `http://www.cs.nuim.ie/~matthewh/CS425/2012/sf/High%20Speed%20Commercial%20-%20buzz.wav`

[6] Direct URL `http://www.cs.nuim.ie/~matthewh/CS425/2012/sf/telephone%20IR.wav`, originally downloaded from `http://fokkie.home.xs4all.nl/IR.htm`

[7] `http://www.cs.nuim.ie/~matthewh/CS425/2012/sf/High%20Speed%20Commercial%20-%20telephone.wav`

synthesis (`http://en.wikipedia.org/wiki/Speech_synthesis`, last access : 24/01/12). In essence, concatenative synthesis concatenates together small pieces of speech that are stored in a database to create words and sentences. Formant synthesis uses a *source-filter model*, the source standing for the vocal cords, and the filter, for the vocal tract.

The applications to speech synthesis are numerous, especially in the domain of assistive technology. Examples include screen readers for visually impaired people and aids to persons with severe speech impairment.

5. Speech recognition is probably the most sought end-goal of computerised speech analysis. The idea here is to get the computer, deprived of the cognitive capabilities of humans, to recognise words or sentences through Digital Signal Processing (DSP) techniques alone. The applications to such technology is obviously speech-to-text transcription, but also speech-based navigation in computer programs. *Windows Speech Recognition* in Windows 7 is a good example of both speech-to-text transcription and speech-based navigation.

Such disciplines as speech synthesis or speech recognition require a number of advanced DSP techniques. The prerequisites to this module come down to 1st- and 2nd-year college mathematics, programming abilities sufficient to pick up the Matlab language quick, and a lot of good sense. However, it is not expected that you know in advance what auto-correlation, the Fourier transform or cepstral analysis are, to mention only a few. To get some speech synthesis or recognition working in one semester only, all these would have had to be known in advance.

So this module will only initiate you to speech processing (meant in the way of analysis), to give you an idea as to how it can be worked out if a given segment of speech is voiced or unvoiced, or what a sentence's pitch contour is. There is already a lot to these, and knowing this much will give you a good impression of what awaits you, should you wish to pursue a career in this area.

Not everything will be about DSP, though. This module is not called *Audio Processing* alone, but *Audio and Speech Processing*. Speech sounds have specific characteristics, which are helpful to be known beforehand, so as to know what types of signals we are looking at and know what kind of information it is we want to extract. The learning of these characteristics

begins with the learning of the human vocal apparatus, which we approach at the onset of the first part of this module.

Exercises

- Describe the processes of sampling and quantisation (use drawings if you'd like)
- Define sampling frequency and bit-depth.
- The *bit-rate* of a waveform is the total number of bits used in the digital representation of one second of this waveform. What is the bit-rate of audio CDs?

Chapter 1

English phonemes : linguistics and acoustics

1.1 The human vocal apparatus

We are not going to bring about a discussion on the physiology of the human vocal apparatus. What only really matters to us is to get to know the names, position and relative appearance of these organs directly involved in the production of speech sounds. Some of these organs are already well-known to you, such as the teeth, the palate, and so on. There are others, however, of which you might never have heard the name before, but that are likely to come up in our discussions on phonetics. The functionality of each really is going to be discussed in due time, in the description of the various phones of the English language. For now, let us only look at Figure 1.1, learn these new names and how to place them on a drawing of the human vocal apparatus.

1.2 The unit of speech : phonemes

Speech recognition methods do not aim for the recognition of entire words directly. These are complex sound structures, and the task of recognising them can obviously be made easier by recognising substructures such as syllables, and putting them into words thereafter. Yet even syllables can be broken down in smaller units of speech and be thereby made simpler to recognise.

The smallest unit of speech one can find is the *phone* :

> **phone**
> "A term used in phonetics to refer to the smallest perceptible discrete segment of sound in a stream of speech." [Crystal, 2009]

As such, syllables and words and sentences can be inferred by the recognition of the phones of a speech and their combinations. However, this might be an overkill, as there exists a structure which can still be seen as the building block of speech : the phoneme.

> **phoneme**
> "The minimal unit in the sound system of a language, according to phonological theories." [Crystal, 2009]
> "The smallest phonetic unit in a language that is capable of conveying a distinction of meaning, as the *m* of *mat* and the *b* of *bat* in English." [`www.thefreedictionary.com`]

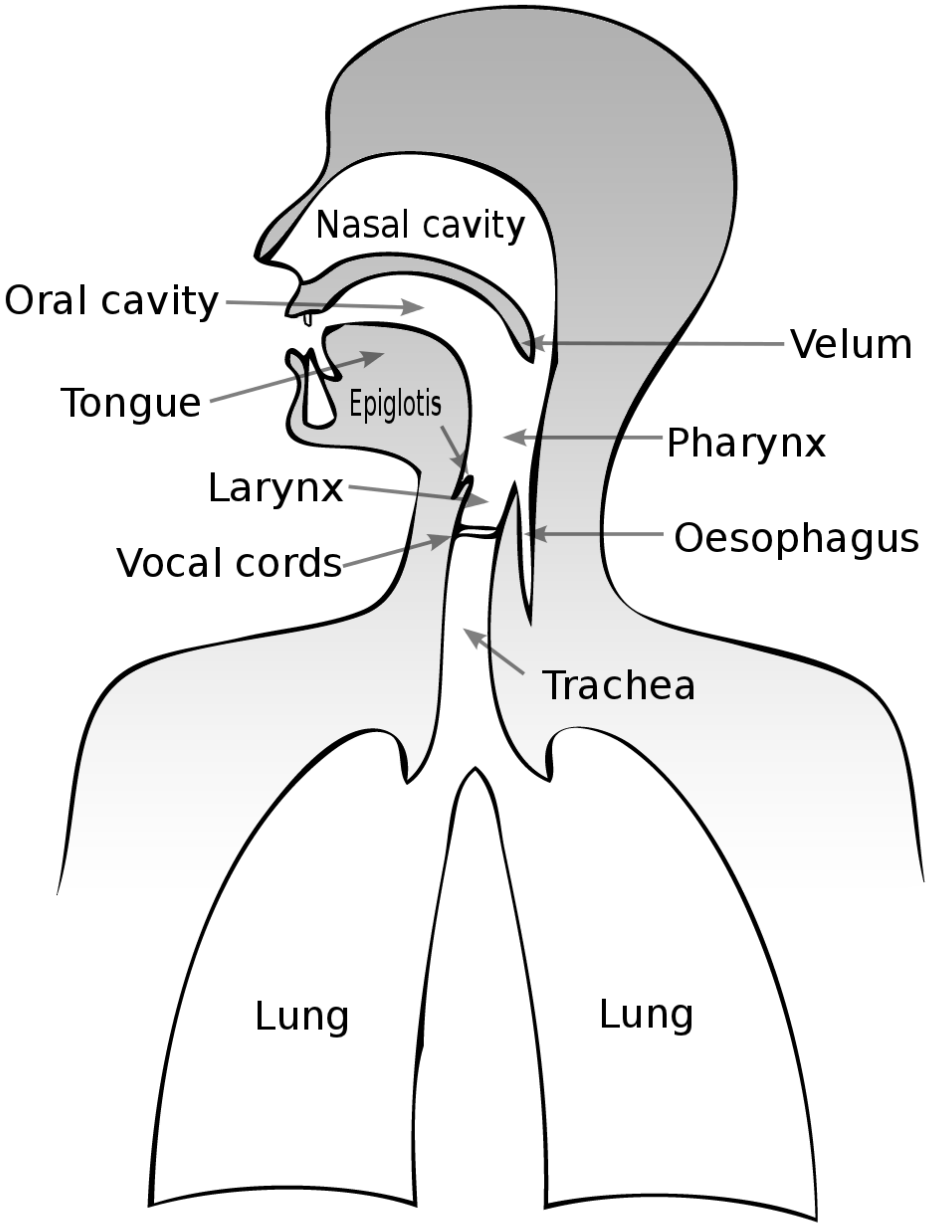

Figure 1.1: *Vocal tract [`http://ling.uta.edu/~josh/LING-3330/chapter-1/vocal-tract.png`, last access : 24-01-12]*

Although at a *phonetic* level, phonemes are more complex than phones – one phoneme can be made of one, two or more phones – at a *phonological* level, they are the simplest speech entity one can find in a language. In this regard, one can say that in English, syllables are made up with phonemes, not with phones. We will therefore work with phonemes.

Some phonemes have a great degree of similarity between them. For example, when you pronounce a /v/, you do the exact same as for the /f/, except that in the former, your vocal cords vibrate, in the latter, not. Knowing this, making the difference between the two at a signal processing level becomes much easier. It is therefore important to categorise phonemes.

Two criteria come into consideration where the classification of phonemes is concerned : the phonetic criterion, of primary importance to us, but also the phonological criterion. The definitions of *phonetics* and *phonology* follow :

> **phonetics**
> "The science which studies the characteristics of human sound-making, especially those sounds used in speech, and provides methods for their description, classification and transcription." [Crystal, 2009]

> **phonology**
> "A branch of linguistics which studies the sound systems of languages." [Crystal, 2009]

Phonetics are only concerned with the mode of production of speech sounds and their acoustic qualities. Phonology, on the other hand, brings about the context of a language, and look at phonemes in terms of their role in the construction of syllables and words. The definition of consonants in [Crystal, 2009] actually illustrates well this difference of approach :

> **consonant**
> "Phonetically, they are sounds made by a closure or narrowing in the vocal tract so that the airflow is either completely blocked, or so restricted that audible friction is produced. (...) From a phonological point of view, consonants are those units which function at the margins of syllables, either singly or in clusters." [Crystal, 2009]

In fact, the concept of *phoneme* is probably an invention of phonology, while that of *phone* could be one of phonetics. In our classification of phonemes,

we will therefore look at both criteria – otherwise, we'd have been looking for phones alone, not phonemes.

1.3 Classification of phonemes

The classification of phonemes as presented in this section was mostly drawn upon that found in the book by Rabiner and Schafer, *Digital processing of speech signals* in Table 3.1 of page 43 [Rabiner and Schafer, 1978]. The symbols use in the written transcription of phonemes were found in the online resource at the URL `http://www.antimoon.com/resources/phonchart2008.pdf`. Finally, the names and naming conventions of these phonetic characters were found in the Wikipedia article "Naming conventions of the International Phonetic Alphabet". The material inherent to these three documents was synthesized to produce the list of IPA characters of Table 1.1 as well as the classification and listing of the English phonemes found throughout this section and its tables.

Phonemes are made up of phones, which are described with characters (IPA characters) that you may have never seen and whose names are probably unknown to you. It is necessary before we start the classification of the phonemes to get to know these characters.

1.3.1 The International Phonetic Alphabet

The International Phonetic Alphabet (IPA) aims at listing all phones of all languages, and for each, assigning a unique grapheme. Here the list is restricted to the phones found in the list of English phonemes. There isn't a one-to-one relationship between the English phones and the Latin alphabet, and as a result this list includes unusual characters. Each character has a name of its own[1], which it is important for both the lecturer and students to know so as to be able to communicate over this topic. Most of these characters are named according to simple rules that are easy to learn, while a few have traditional names. Below we list these naming conventions.

[1]The naming of IPA characters is not unanimous, but mostly consistent. The conventions given here should allow you to reach over the differences you might find in other contexts where the IPA is used. See the "Naming conventions of the International Phonetic Alphabet" Wikipedia article for more details.

- The characters present in the Latin or Greek alphabets keep their name. E.g. a is "a", ʊ is "upsilon", θ is "theta"...
- There is one *script* character in this list, the ɑ, "script a". There exist others, but not known as English phones.
- Some are ligatured characters. In our list, only the æ is, and is called "a-e ligature".
- Characters that are rotated upside-down take the prefix "turned". E.g. ə ("turned e"), or ɒ ("turned script a").
- Characters that are flipped left-right take the prefix "reversed". In our list, we find ɜ ("reversed epsilon").
- Capitalised letters, albeit smaller than usual, are said "capital", as in ɪ, "capital i".

The characters that abide by neither of these rules are characters proper to the IPA, and have traditional names. Also, mention should be made of the "length mark" (ː), which, if following a vowel, indicates a stress.

Now that we are familiar with this naming scheme, we can move on to using them in the classification of English phonemes.

1.3.2 Vowels

[Crystal, 2009] gives the following definition for vowels :

> **vowel**
> "Sounds articulated without a complete closure of the mouth or a degree of narrowing which would produce audible friction." [Crystal, 2009]

To make the difference between vowels and diphthongs, a complementary definition should be introduced :

> **vowel** (bis)
> "Vowels are produced by exciting a fixed vocal tract with quasi-periodic pulses of air caused by vibration of the vocal cords." [Rabiner and Schafer, 1978]

IPA symbol	name
ʌ	turned v
ɑ	script a
æ	a-e ligature
ə	turned e
e	e
ɜ	reverse epsilon
ɪ	capital i
i	i
ɒ	turned script a
ɔ	turned c
ʊ	upsilon
u	u
a	a
o	o
b	b
d	d
f	f
ɡ	g
h	h
j	j
k	k
l	l
m	m
n	n
ŋ	eng
p	p
r	r
s	s
ʃ	esh
t	t
θ	theta
ð	eth
v	v
w	w
z	z
s	s
ʒ	ezh
ː	15length mark

Table 1.1: *The International Phonetic Alphabet (IPA)*

IPA Symbol	Examples
ʌ	cup, luck
ɑː	arm, father
æ	cat, black
ə	away, cinema
e	met, bed
ɜː	turn, learn
ɪ	hit, sitting
iː	see, heat
ɒ	hot, rock
ɔː	call, four
ʊ	put, could
uː	blue, food
eə	where, air
ɪə	near, here
ʊə	pure, tourist

Table 1.2: *Vowels*

Here, the term "fixed vocal tract" stresses the fact that vowels are steady, as opposed to diphthongs which feature an element of *glide* (see below). Yet, these two definitions together are still not enough to make the distinction between vowels and *semivowels*. To understand where the difference lies, it is necessary to bring about a *phonological* definition of vowels "

> **vowel** (ter)
> "A speech sound (...) usually forming the most prominent and central sound of a syllable" [`www.thefreedictionary.com`]

In our classification of phonemes, all three of these definitions together are necessary to pinpoint vowels. Would you be able to merge these into one, complete definition?

1.3.3 Diphthongs

The difference between vowels and diphthongs lies at a phonetic level :

> **diphthong**
> "A gliding monosyllabic speech item that starts at or near the

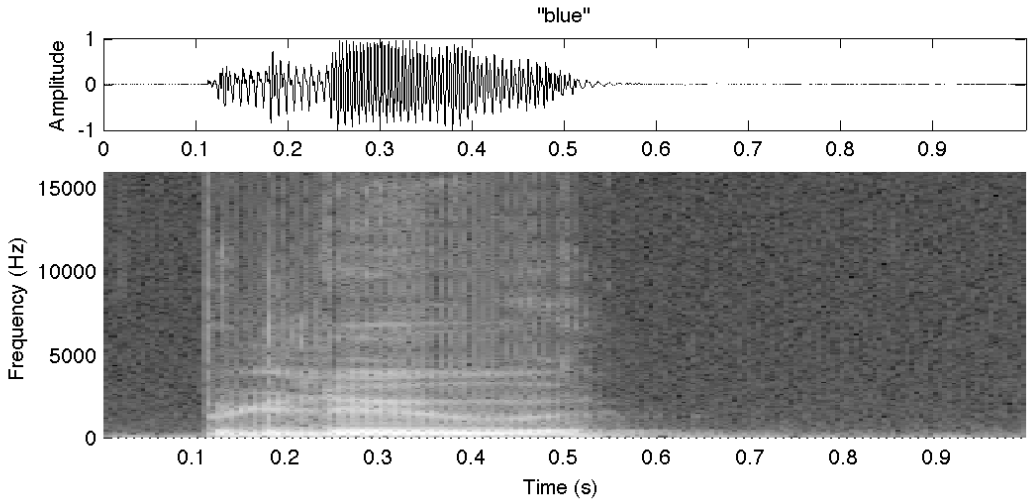

Figure 1.2: *Vowel /uː/ in "blue" (/bluː/). Notice that throughout the phoneme /uː/, the spectrogram remains approximately static, with continuous horizontal stripes (the* formants*). Such stripes are the result of the excitation of the vocal cords. The fact that they are steady is indicative of the steadiness of the vocal tract (fixed position).*

IPA Symbol	Examples
aɪ	five, eye
aʊ	now, out
oʊ	go, home
eɪ	say, eight
oɪ	boy, join
ju	new, abuse

Table 1.3: *Diphthongs*

> articulatory position for one vowel and moves to or toward the position for another." [Rabiner and Schafer, 1978]

Phonologically, diphthongs and *monophthongs* (i.e. straight vowels) are identical, constituting as they do the central, stressed elements of syllables. There exists six diphthongs in English, as listed in Table 1.3.

1.3.4 Semivowels

At a phonetic level, semivowels abide by the definition of vowels. However, they are found at the margin of syllables, and are used to articulate syllables together. They thus have the phonological role of consonants.

> **semivowel**
> "A sound functioning as a consonant but lacking the phonetic characteristics normally associated with consonants (such as friction or closure); instead, its quality is phonetically that of a vowel; though, occurring as it does at the margins of syllables, its duration is much less than that typical of vowels." [Crystal, 2009]

Interesting for us to realise that semivowels are shorter in duration... could be used as a clue to recognise them...

Anyway, the list of semivowels is given in Table 1.4. Notice that all of them are usually considered as consonants, but try them all in turn out of the context of words, and you will realise that indeed they have the acoustic characteristics of vowels. This ambiguity between phonological role and phonetic quality explains why the letter y is considered a consonant in certain languages (as in English), and a vowel in others (as in French).

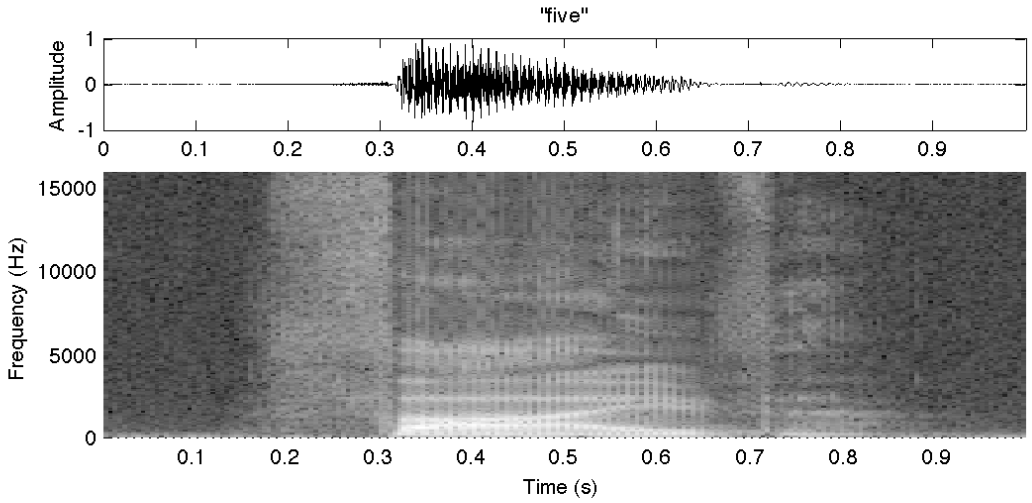

Figure 1.3: *Diphthong /aɪ/ in "five" (/faɪv/). Throughout the diphthong, the vocal cords are steadily excited, all the while the vocal tract changes from one vowel position to another, resulting in a conspicuous motion of the formants.*

IPA Symbol	Examples
w	wet, window
l	leg, little
r	red, try
y	yes, yellow

Table 1.4: *Semivowels*

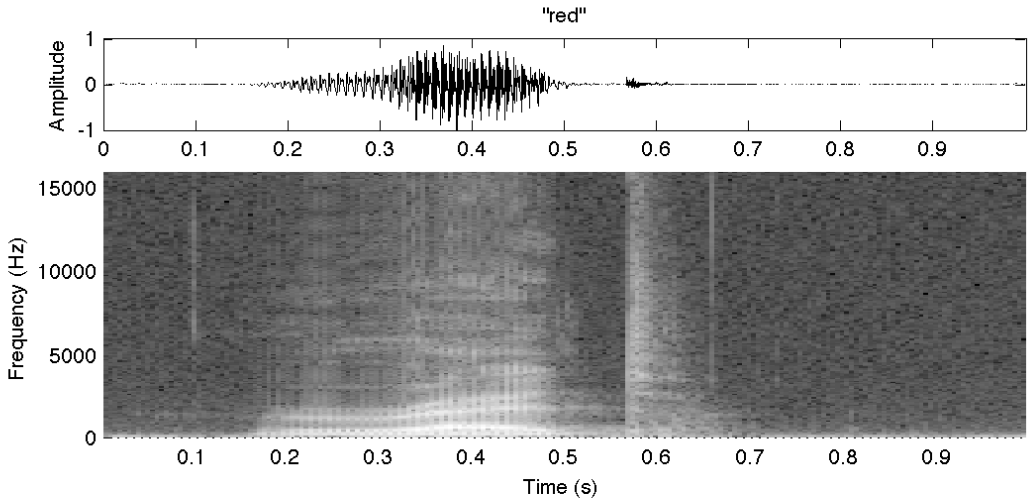

Figure 1.4: *Semivowel /r/ in "red" (/red/). The spectrogram of the phonemes /re/ could be interpreted as that of a diphthong, abiding as it does by the description we've give of the spectrogram of Figure 1.3. This will generally be the case with semivowels, as they are most often preceded and/or followed by a vowel. The distinction here is mostly phonological, although, at a phonetical level, semivowels tend to be shorter in duration than regular vowels, due to their functionality of consonants. (In this example, I actually insisted a little bit too much on the /r/, which makes it even longer than the following /e/...)*

IPA Symbol	Examples
m	man, lemon
n	no, ten
ŋ	sing, finger

Table 1.5: *Nasals*

1.3.5 Nasals

> **nasal** The nasal consonants /m/, /n/ and /ŋ/ are produced with glottal excitation and the vocal tract totally constricted at some point along the oral passageway. The velum is lowered so that air flows through the nasal tract, with sound being radiated at the nostrils. [Rabiner and Schafer, 1978]

The list of nasals is as in Table 1.5.

1.3.6 Stops

Voiced and unvoiced stops should be treated together, as

> **voiced stop**
> voiced stop consonants (/b/, /d/ and /g/) are transient, non-continuant sounds which are produced by building up pressure behind a total constriction somewhere in the oral tract, and suddenly releasing the pressure

while

> **unvoiced stop**
> unvoiced stop consonants (/p/, /t/ and /k/) are similar to their voiced counterparts with one major exception : during the period of total closure of the tract, as the pressure builds up, the vocal cords do not vibrate. [Rabiner and Schafer, 1978]

The difference is only slight and subtle, but it is there, and that seems to be the only thing which makes a difference between /b/ and /p/, /d/ and /t/, and /g/ and /k/. Voiced and voiced stops are listed together in Table 1.6.

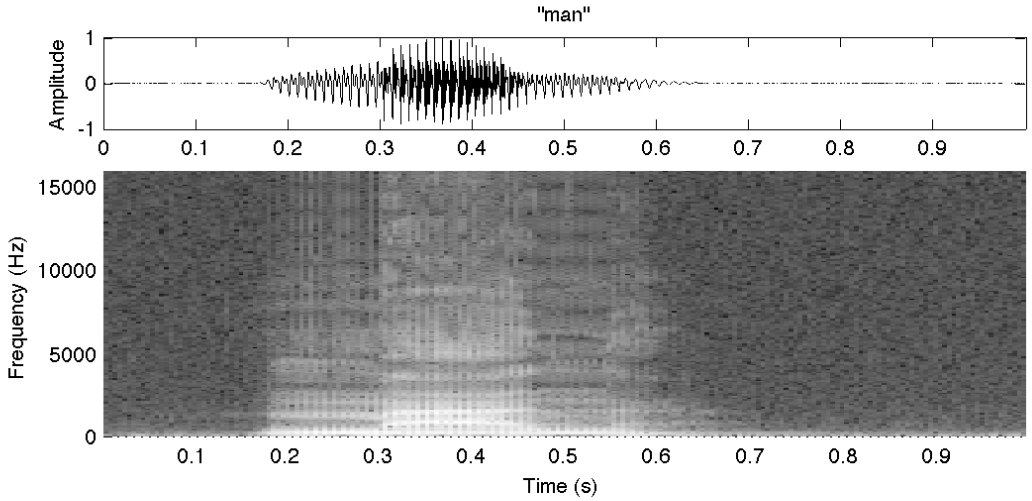

Figure 1.5: *Nasals /m/ and /n/ in "man" (/mæn/). They are not exactly like vowels in phonetical terms, as the mouth is completely shut during their production, yet the vocal cords are in motion, yielding the characteristic formants. Due to the closure of the mouth, the outgoing energy is noticeably lesser than for vowels.*

IPA Symbol	Examples
b	bad, lab
d	did, lady
g	give, flag
p	pet, map
t	tea, getting
k	cat, black

Table 1.6: *Stops*

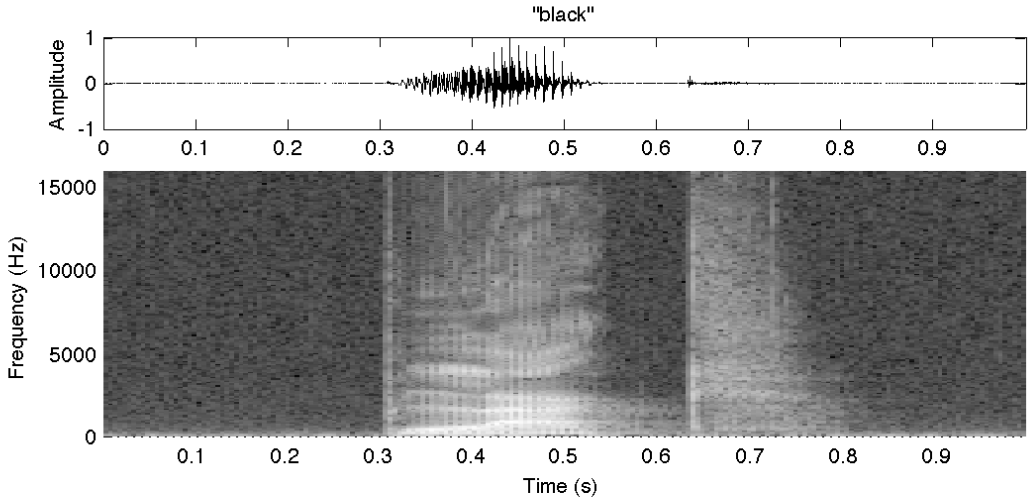

Figure 1.6: *Voiced stop /b/ and unvoiced stop /k/ in "black" (/blæk/). The distinction between voiced and unvoiced is subtle, but seems to be present in the spectrogram : the opening of the mouth for the /b/ seems to begin not at the very onset, but rather at the moment where the formants begin. The energy before that seems to be the preliminary, voiced build-up of air behind the close lips. Try it : keep your mouth and nasal cavity shut so that no air can escape, and set your vocal cords in motion. They will vibrate for a split second only, until there is not more room for air in your mouth. This happens for /b/, but not for /k/, where energy is only visible after the opening of the mouth.*

IPA Symbol	Examples
f	find, if
θ	think, both
s	sun, miss
ʃ	she, crash
v	voice, five
ð	this, mother
z	zoo, lazy
ʒ	pleasure, vision

Table 1.7: *Fricatives*

1.3.7 Fricatives

Likewise, unvoiced and voiced fricatives are treated together, as

> **unvoiced fricative**
> unvoiced fricatives (/f/, /θ/, /s/ and /ʃ/) are produced by exciting the vocal tract by a steady air flow which becomes turbulent in the region of the constriction in the vocal tract

while

> **voiced fricatives**
> voiced fricatives (/v/, /ð/, /z/ and /ʒ/) are the counterparts of unvoiced fricatives in that the place of constriction for each of the corresponding phonemes is essentially identical. However, the voiced fricatives differ markedly from their unvoiced counterparts in that two excitation sources are involved in their production. For voiced fricatives the vocal cords are vibrating, and thus one excitation source is at the glottis. [Rabiner and Schafer, 1978]

The list of voiced and unvoiced fricatives is given in Table 1.7.

1.3.8 Affricates

A straightforward definition of *affricates*,

> **affricate**
> A stop consonant followed by a fricative. [`www.thefreedictionary.com`]

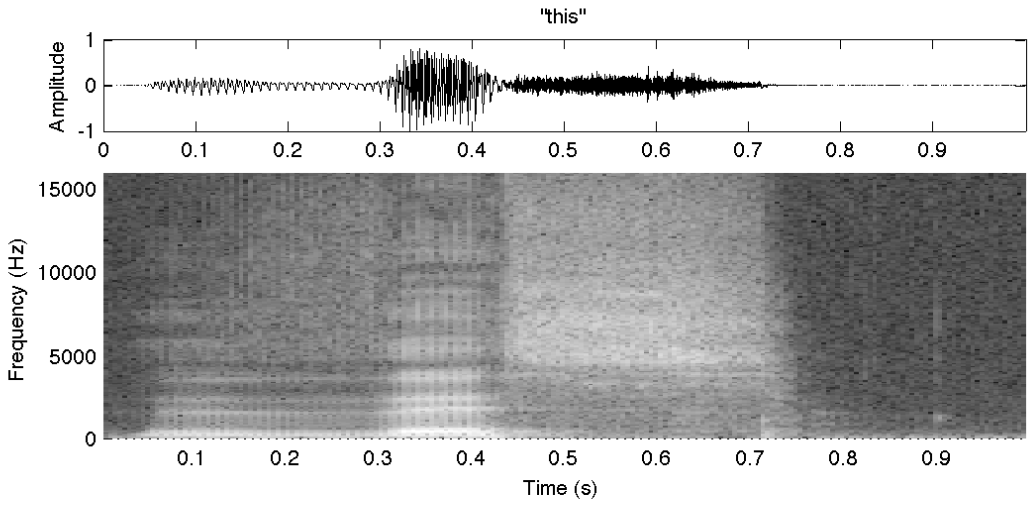

Figure 1.7: *Voice fricative /ð/ and unvoiced fricative /s/ in "this" (/ðɪs/). /ð/ is voiced, hence the presence of formants ; /s/ is unvoiced, and so the formants are gone. Phonetically, consonants involve noisy turbulence in the outgoing air flow, caused by a narrowing of the vocal tract at some place. Noise in a spectrogram is discernible as broadband energy (i.e. a smeared layer of energy spanning a wide frequency interval), which is very visible here with the /s/. Such energy is much less present over the /ð/, and you will notice that, when you pronounce this phoneme, the noisy bit coming from the constriction between your teeth and your tongue is only faint. Still, there is constriction, and /ð/ classifies as a consonant.*

IPA Symbol	Examples
dʒ	just, large
tʃ	check, church

Table 1.8: *Affricates*

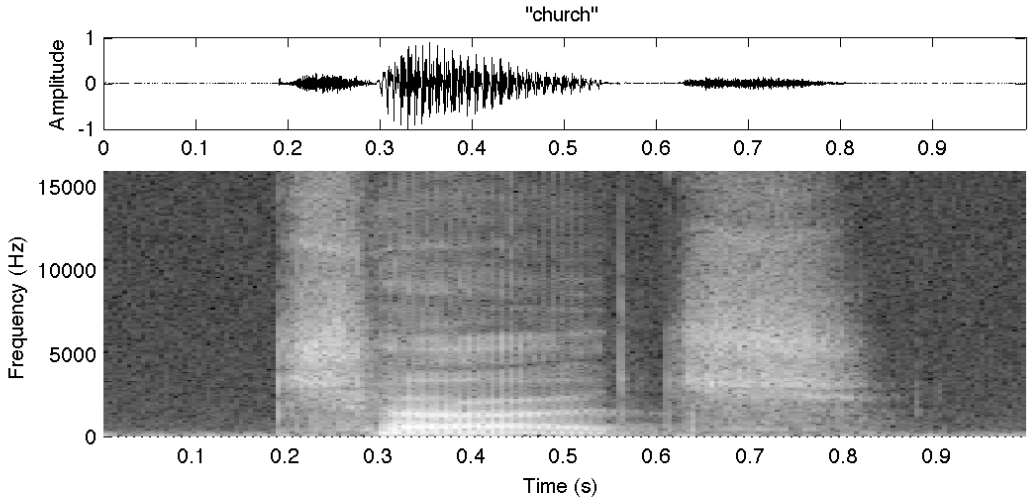

Figure 1.8: *Affricate /tʃ/ in "church" (/tʃɹtʃ/). This phoneme clearly comes out as the concatenation of a stop and an affricate.*

a more refined one,

> **affricate** (bis)
> Sound made when the air-pressure behind a complete closure in the vocal tract is gradually released; the initial release produces a plosive, but the separation which follows is sufficiently slow to produce audible friction, and there is thus a fricative element in the sound also. [Crystal, 2009]

and the (short) list of affricates (Table 1.8).

1.3.9 Whisper (or /h/)

The last phoneme we are seeing here stands alone in its own category : the /h/ :

IPA Symbol	Examples
h	how, hello

Table 1.9: *Whisper*

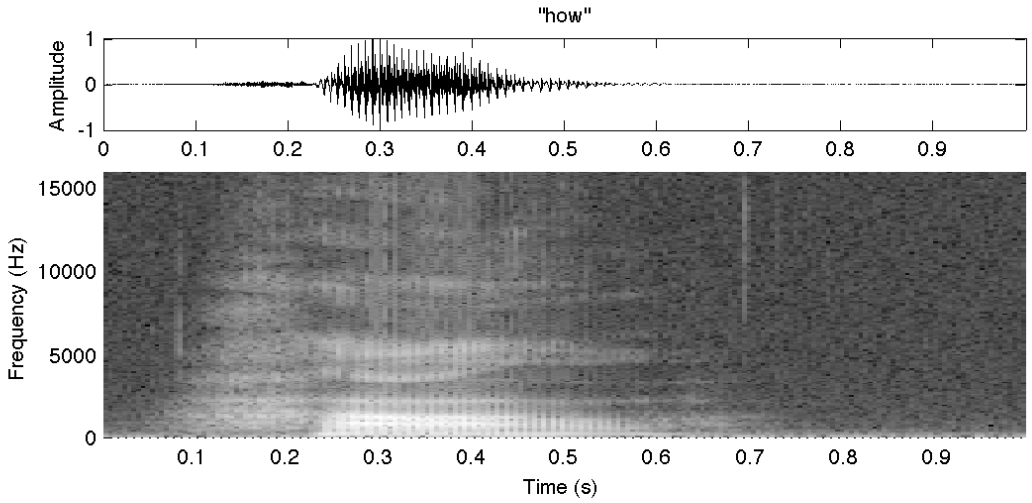

Figure 1.9: *Whisper /h/ in "how" (/haʊ/). The character of this consonant is similar to that of fricatives, in terms of noisiness. The particularity of this phoneme is probably the fact that, although the vocal cords are still, the spectral density is similar to that of the following vowel, to the point that formants are almost discernible. The energy is nevertheless smeared vertically, due to its noisy nature, and lesser than that of vowels.*

> **whisper**
> The phoneme /h/ is produced by exciting the vocal tract by a steady air flow – i.e., without the vocal cords vibrating, but with turbulent flow being produced at the glottis. The characteristics of /h/ are invariably those of the vowel which follows since the vocal tract assumes the position for the following vowel during the production of /h/ [Rabiner and Schafer, 1978]

For the sake of consistency, we present this phoneme in a table, Table 1.9.

1.3.10 Recapitulation on phonemes' definitions

It was deemed useful to put together a recapitulative table where the classes of phonemes are listed, along with the distinctive features of each, in terms of phonetics and phonology separately. The result is presented in Table 1.10.

1.4 Conclusion

exercises

- Give list of phonemes and student has to tell what phoneme class each is, and give the IPA symbol for each.
- Give blank vocal apparatus and student has to fill.
- Get student to write complete definition of vowel
- Write phonetic transcription of a piece of speech under spectrogram and waveform.
- Characterise phonemes phonetically, phonologically, in terms of waveform and spectrogram.
- Give a blank spectrogram and waveform and the student has to guess as to what class of phonemes is represented.

category	phonetic description	phonologic role
vowel	fixed vocal tract ; mouth not closed ; no audible friction	central part of a syllable
consonant	flow of air completely blocked or noisy due to constriction	margin of syllables
diphthongs	monosyllabic ; glide from a vowel to another	c.f. *vowel*
semivowels	c.f. *vowel*	c.f. *consonant*
nasals	glottal excitation ; vocal tract completely closed ; sound radiated at nostrils	c.f. *consonant*
unvoiced fricatives	steady air flow, turbulent due to constriction	c.f. *consonant*
voiced fricatives	c.f. *unvoiced fricatives* + additional excitation of vocal tract by vocal cords	c.f. *consonant*
unvoiced stops	pressure build-up behind total constriction, subsequent sudden release	c.f. *consonant*
voiced stops	c.f. *unvoiced stops* + small excitation from vocal cords before release	c.f. *consonant*
affricates	gradual release of air pressure after complete constriction ; concatenation of /t/ or /d/ with /ʃ/ or /ʒ/, respectively	c.f. *consonant*
Whisper	steady air flow, made turbulent at glottis	c.f. *consonant*

Table 1.10: *Recapitulation on phonemes categories*

Chapter 2

Voiced/Unvoiced/Silence discrimination through time-domain processing

The phonetic and phonologic classification of the English phonemes of the first part of this module might inspire to the sound engineer a variety of ideas as to how to recognise the various categories of phonemes automatically - with the ultimate goal of recognising the phonemes, and syllables and words they're in. For instance, one may be able to spot vowels looking for fixed formants. Voiced fricatives would also have fixed formants, but also a high-frequency, broadband component due to the excitation source produced by the constriction in the vocal tract. Diphthongs could be spot at their formants' glide, and so on.

The picture here is very complex, and things are easier said than done. For us, a simpler starting point should be found. Voiced/Unvoiced/Silence (VUS) discrimination is usually given for a starter in textbooks [Rabiner and Schafer, 1978, Kondoz, 2004]. This is what we aim at in this chapter. It may not seem much, but it is a start to Speech Recognition, allowing the separation of all voiced phonemes from all unvoiced phonemes. (The phonemes involving both excitations of the vocal chords and vocal tract constriction are actually both voiced and unvoiced, but let us keep it simple for now.)

In this chapter, we will stick with time-domain processing techniques, and fairly straightforward ones, too. These will provide us with higher-level pieces of information about our speech signals, on top of which we will use a little bit of statistics and rely on algorithmic decision-making to come up with our automated discrimination method. Just be aware that the approach we are taking here is one in many. An example of an approach based more on statistics and less on heuristics is that based on Hidden Markov Models [Rabiner, 1989, Plannerer, 2005, Pelton, 1993]. For a larger overview of methods used in speech recognition, see [Plannerer, 2005].

Here we are going to begin with a simple – but not reliable in all circumstances – way of automatically discriminating speech and silence. As described, the only attribute of the waveform involved is its short-time level. We will follow up with a voiced-unvoiced discrimination procedure, which is a similar approach to the speech-silence discrimination, but involving several different attributes of the waveform, as opposed to one only. The most important learning outcome of this journey will probably be the learning of some signal processing basics, applicable to areas other than speech recognition as well. A lot of the rest of the material will be more algorithmic, *ad hoc* material, which is inherent to our "hands-on" approach. The algorithmic aspect to the voiced-unvoiced discrimination is largely based and inspired on the method found in [Kondoz, 2004]. Some of it is my own, and a little part

of it will be... yours! A great opportunity to give your brain exercise, and also learn to use Matlab to implement and test your method(s).

To recapitulate before we start, there are two main sections to this chapter of the module : the speech-silence discrimination part, and the voiced-unvoiced one. The procedure found in each is similar. At first, we consider the speech waveform, and try and find attributes that differentiates the two states we want to discriminate. Then, we see how these attributes can be *functionalised*, or how it is that we can process the waveform to obtain a measurement of these discriminatory attributes. Finally, we come up with an algorithm which uses these measurements to decide which segments of the input belongs with which state.

2.1 Energy-based Speech/Silence discrimination

In order to discriminate speech from silence segments, we should find out as many and reliable attributes as possible which make a difference between speech and silence states. Unfortunately in the case of the speech-silence dichotomy, there seems to be only one that's reliable and straightforward enough : level, which is greater for speech segments. We say it is unfortunate, because this attribute alone is not completely reliable either, as silence is never really silence ; there is always a little bit of environmental noise that can rival in level with the faintest of speech sounds, such as unvoiced fricatives. If this noise showed any other time-domain characteristics that could distinguish it from speech waveform, we should use them. But it does not seem to, as the characteristics of the background noise are completely dependent on the recording environment, upon which very few assumptions might be taken. In other words, we are to find characteristics of a state that cannot be characterised, not in general terms anyway. In the frequency domain, it is possible to take a spectral print of the background noise which can help in its characterisation, but so long as we are in the time domain, we can only assume that the level of silence is lesser than that of voiced and unvoiced segments, and use this as a discrimination criterion.

2.1.1 Discriminatory attribute

Figure 2.1 presents the waveform of the utterance "crash", surrounded with a little bit of silence. If you are viewing this document in electronic format, you

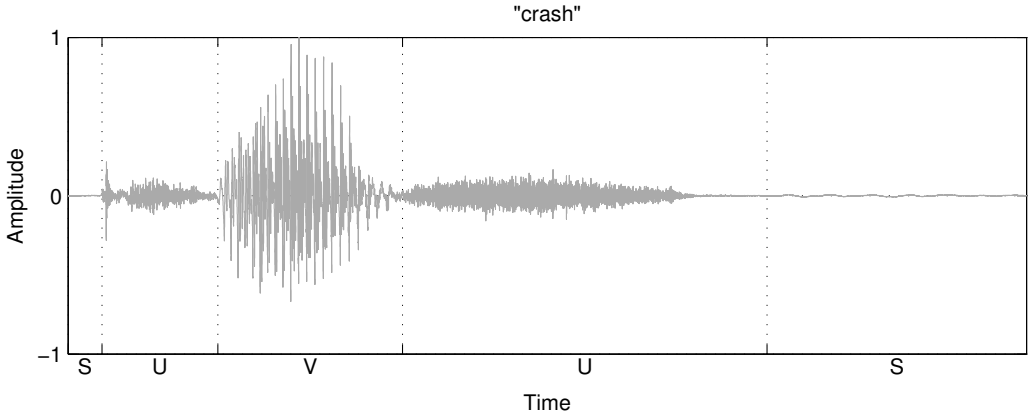

Figure 2.1: *Utterance "crash" segmented in Voiced, Unvoiced and Silence (VUS) states.*

can zoom in on the silence segments and see for yourself that even there, the waveform is not completely zero and there is a little bit of turbulence. But the recording conditions were still good enough to attain a Signal-to-Noise Ratio (SNR) – the maximal dB level of the waveform lest the dB level of the noise floor – of approximately 60 decibels. It is therefore fairly easy to spot by the eye where the silence segments lay. We can do this because our eye gets an overall vision of the envelope, or contour, of the waveform. The question we address in the following section is how, at a signal processing level, to get this envelope?

2.1.2 Candidate attribute functions

We are here going to present two formal functions : the Short-Time Energy (STE), and the Mean Average (MA). These two functions of the "crash" waveform were normalised and superimposed to the waveform in Figure 2.2. The STE is obtained by summing the square of the waveform values over a finite number of samples, N,

$$\boxed{\mathrm{STE}[n] = \sum_{m=0}^{N-1} x^2[n-m]}\ , \tag{2.1}$$

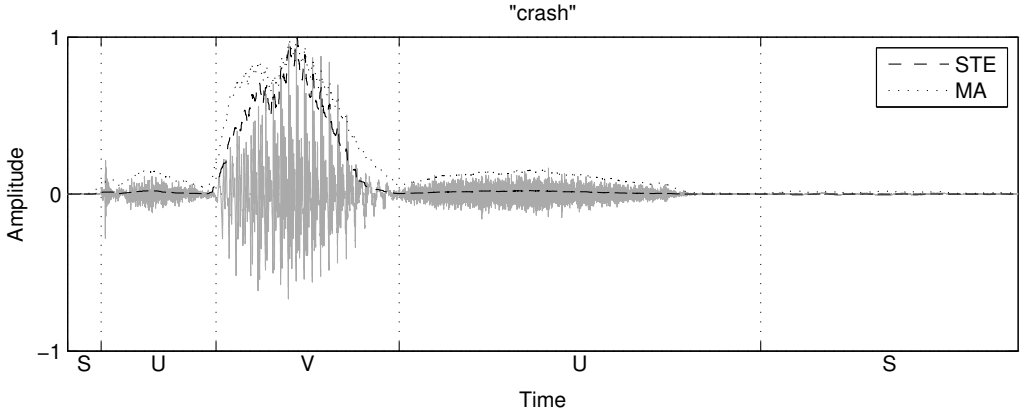

Figure 2.2: *Same utterance, with Short-Time Energy (STE) and Magnitude Average (MA) outlines. Due to the squaring process involved in STE computation, extreme waveform values produce an emphasis in the STE measurement. The MA reflects better the waveform envelope as we see it, which is why it is often preferred in level-based discrimination. [Kondoz, 2004]*

while the MA sums the absolute value (denoted by the vertical brackets $|.|$) of these same waveform values,

$$\boxed{\mathrm{MA}[n] = \sum_{m=0}^{N-1} |x[n-m]|} \,. \tag{2.2}$$

Here, notice that the STE and MA values at index n use the past N values, including the current value n, i.e. $x[n], x[n-1], ..., x[n-N+1]$. For practical uses, we rather use the N values centered around n, from $n-N/2$ to $n+N/2-1$. The formal definitions, however, must be such, so as to obtain a direct equivalence between the STE as shown in (2.1) and with the *convolution* of the square of the waveform with a *rectangular window*,

$$\mathrm{STE}[n] = \sum_{m=-\infty}^{\infty} x^2[n-m]w[m], \tag{2.3}$$

where

$$w[n] = \begin{cases} 1, & n \in [0, N[\\ 0, & n \notin [0, N[\end{cases}, \tag{2.4}$$

and the output of an FIR filter of order $N-1$, whose coefficients $b_0, b_1, ..., b_{N-1}$ are all ones,

$$\mathrm{STE}[n] = \sum_{m=0}^{N-1} b_m x^2[n-m]. \tag{2.5}$$

We will see convolution and filter theory in more detail in due time. Equations (2.3), (2.4) and (2.5) are really here to justify the backward interval of summation in (2.1) and (2.2), and may serve as references later in this handout. Note that the very same equivalences could have been drawn with the MA.

Back to our topic of STE and MA and how they reflect the contour of the waveform, the fact that in the STE it is the square of the waveform values which is used, as opposed to the absolute value, emphasizes large amplitude values. This is why the modest waveform amplitude of the unvoiced segments are not proportionally represented in the STE dashed line of Figure 2.2. For this reason, the MA is often preferred as a discrimination criterion [Kondoz, 2004], but the STE remains a standard measure, one step away from the short-time *intensity*,

$$\boxed{\mathrm{STI}[n] = \frac{1}{N} \sum_{m=0}^{N-1} x^2[n-m]}\,, \tag{2.6}$$

and two from the short-time Root-Mean-Square (RMS),

$$\boxed{\mathrm{RMS}[n] = \sqrt{\frac{1}{N} \sum_{m=0}^{N-1} x^2[n-m]}}\,. \tag{2.7}$$

We have now two ways of measuring the level of our waveform. The remaining part is to work out in detail how these measures should be used in the speech-silence discrimination.

2.1.3 Algorithm

If all STE or MA values for speech segments were greater than for silence segments, there would be a threshold T such that $\mathrm{STE}_{\mathrm{speech}} \geq T$, and $\mathrm{STE}_{\mathrm{silence}} \leq T$. Finding this threshold would then be trivial, and T could be used to determine with certainty the underlying state to any time index.

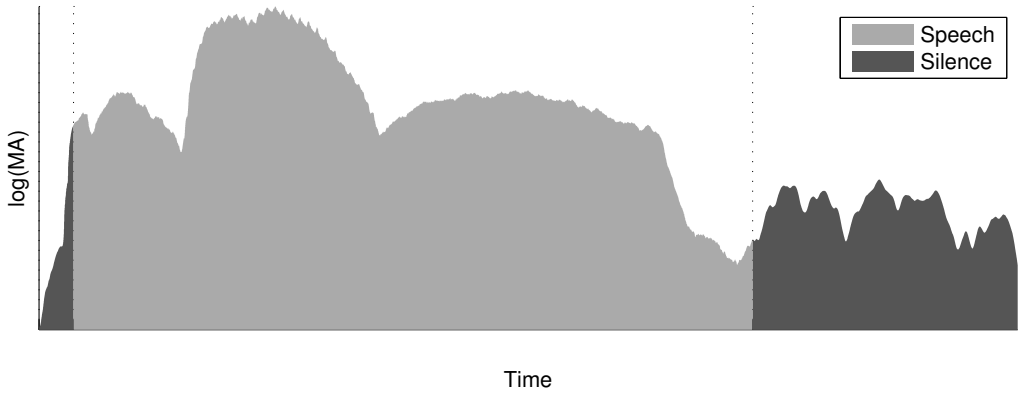

Figure 2.3: *State value overlap : it is not always true that silent-segment MAs are lesser than speech-segment MAs, which is visible here. In general, we say that the attribute function values of one state overlap to some extent with these of the other state.*

Of course, it is not as straightforward, and there is normally some *overlap*. This is exemplified in Figure 2.3, where it can be found that, in spite of the general trend, there are some speech MA values lesser than other silence MA values. The question we are to address now is how the threshold should be set, with regard to this overlap.

Threshold setting

State-discriminatory thresholds are used in the voiced-unvoiced algorithm presented in [Kondoz, 2004], but it is not said how these should be determined. The material presented in this subsection is thence original.

The method proposed here determines the threshold upon the overlapping values only, disregarding the rest of the data. The overlapping values of Figure 2.3 were sorted and presented in Figure 2.4. In this representation, we use f and g to denote the attribute function values of the lower- and upper-valued states that overlap. The lower and upper limits of this interval of overlap are situated at $\min(g)$ and $\max(f)$ – although $\min(f)$ and $\max(g)$ could have been equivalently used. N_f and N_g are the numbers of overlapping values f and g. These numbers are used to normalise the x-axes of the plots, so as to give equal weight to these conflicting sets of values.

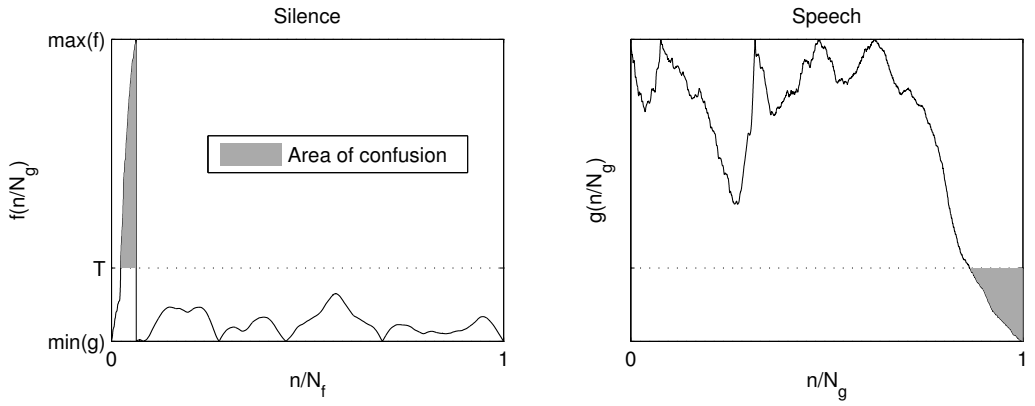

Figure 2.4: *Determination of threshold such that the area of confusion is equal between the states.*

An easy approach would be to set T as the mid-value in the region of overlap, i.e. halfway up in Figure 2.4. This, however, might not be "fair" to one of the states. In this example, you see that the largest part of the overlapping values of the silence state would be found under the threshold, while the largest part of the speech state would be found as belonging with State 1. Let us call the part of a state which is on the wrong side of the threshold the "area of confusion" of that state. To be fair to both states, the threshold should be adjusted so that the area of confusion is equal in both states. This can be formulated mathematically as

$$\frac{1}{N_f}\sum_{n=0}^{N_f-1}\max(f[n]-T,0)-\frac{1}{N_g}\sum_{n=0}^{N_g-1}\max(T-g[n],0)=0. \tag{2.8}$$

An easy and quick way of finding the value of T which satisfies 2.8) is to use a binary search. Normally this is an algorithm for finding a value within a finite ordered set of values. Here, the value we want to find is probably within neither f or g. A good time when to stop the search to stop is when, from one iteration to the next, the number of values of f and g below or above T has not changed. The algorithm can therefore be formulated as follows :

1. From the overlapping attribute functions, only keep the overlapping part, that you store in vectors f and g.

2. Set $T_{\min}$ and $T_{\max}$ as the minimum and maximum of this region of overlap.

3. Set $T = \frac{1}{2}(T_{\min} + T_{\max})$.

4. Set i and p as the number of values of f and g below and above T, respectively, i.e. $i = \sum_n f[n] < T$ and $p = \sum_n g[n] > T$, where $<$ and $>$ are to be seen as boolean operators, producing 1 if true, 0 if false.

5. Set j and q both to -1.

6. So long as $i \neq j$ or $p \neq q$, repeat the following steps.

7. Evaluate the left-hand side of (2.8). If it is positive, set $T_{\min} = T$. Else, set $T_{\max} = T$.

8. Set $T = \frac{1}{2}(T_{\min} + T_{\max})$.

9. Set $j = i$, and $q = p$.

10. Set $i = \sum_n f[n] < T$ and $p = \sum_n g[n] > T$.

Decision-making

The threshold to our sole attribute function is now revealed. In this case, taking the decision as to whether a time index corresponds to either state is simply finding out whether the attribute function at that time index is valued less or more than the threshold. The result of our algorithm as applied to our training signal "crash" is shown in Figure 2.5. Notice that here, we have determined Speech and Silence segments on the very signal that we have been using for the "training" part of algorithm. Recall that this signal, the "crash" utterance, was manually segmented in speech and silence parts, and it is from the overlapping MA values of our segmentation that the threshold T was determined. However, this threshold could now be used for the speech-silence discrimination of other utterances, so long as they were recorded in similar circumstances of environmental noise. This is demonstrated in Figure 2.6, where the threshold determined upon the utterance "crash" was used to speech-silence discriminate "hit".

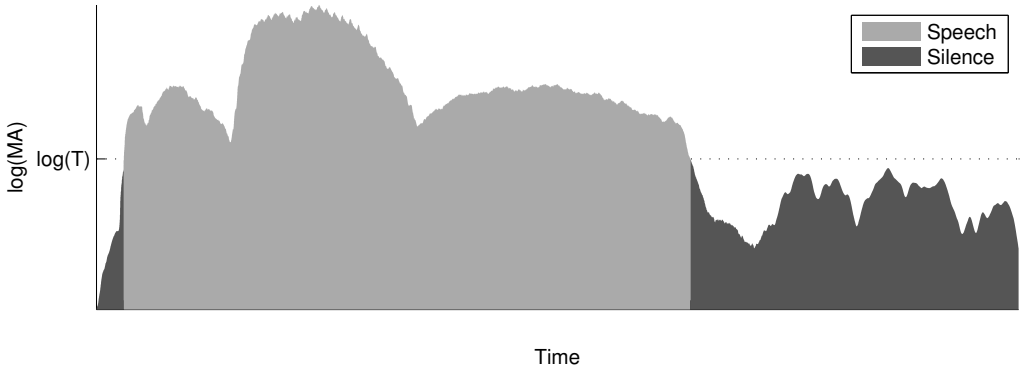

Figure 2.5: *Simple decision-making in the case of one attribute function only : make whichever values of that function that are below the threshold belong with one state, and above the threshold, with the other.*

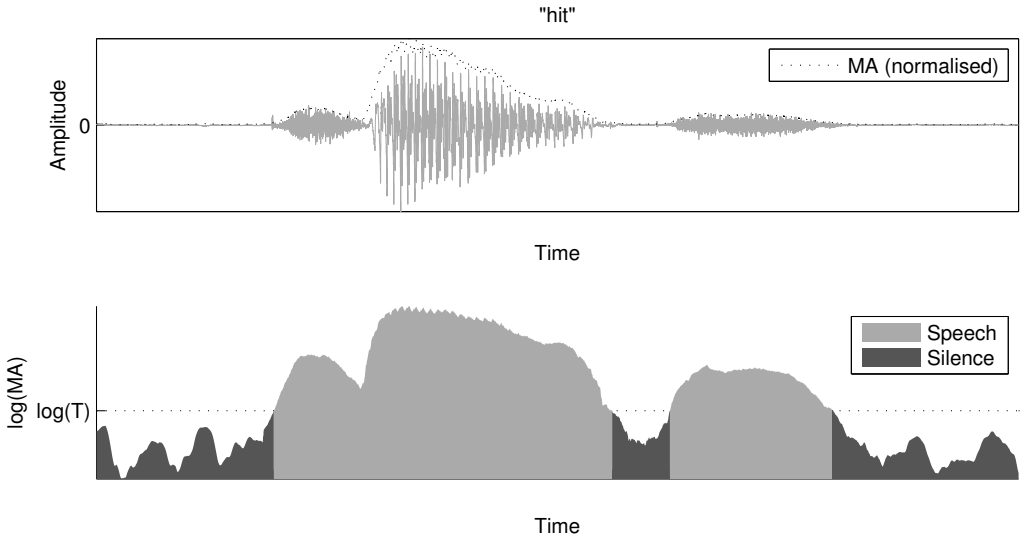

Figure 2.6: *Threshold determined upon utterance "crash" used to state-discriminate "hit". The two words were recording within the same environment.*

Are plosives silence?

"hit" features the unvoiced stop /t/. Our energy-based speech-silence discrimination has understandably categorised the stop before the burst as silence, but this is within a word, so it shouldn't be considered silence, should it? Can you design and implement a method that would fix this?

2.1.4 *A posteriori* comments

Although in its detail and implementation it might seem a little bit complicated at first, this method in its idea is extremely simple and intuitive : manually segment a signal in its speech and silence parts ; measure the MA of both ; set a threshold within the region of overlap ; measure the MA of any speech signal that was recorded in similar conditions ; categorise all time indices whose MA is below the threshold as silence. The major limitations of this approach are, first, that it requires the preliminary manual segmentation of a signal, second, that all signals this method is thereafter used upon were recorded in similar environment, and last but not least, that the SNR of the recording conditions is large enough to make the difference between silence and speech segments of low energy. More information on the characteristics of the environmental noise could be useful in overcoming this last limitation, at least to some part. But this, again, is more the resort of frequency-domain analysis.

For now we stay in the time-domain. Now that we have a method for speech-silence discrimination that works rather well, we can proceed to voiced-unvoiced discrimination within the speech segments returned by our method. We'll therefore end up with a complete VUS discrimination.

2.2 Voiced/Unvoiced discrimination

The voiced-unvoiced discrimination to follow uses the same idea as the previous speech-silence discrimination, looking as it does for attributes that characterise contrastingly the states to discriminate, and setting for each a threshold. However, there is this time more than one such attributes. The added complexity here therefore regards the way to use several attribute functions and thresholds in the final voiced-unvoiced discrimination, because, as you'd expect, not all attribute functions will be synchronously above or below their respective discriminatory threshold. These potential conflicts will

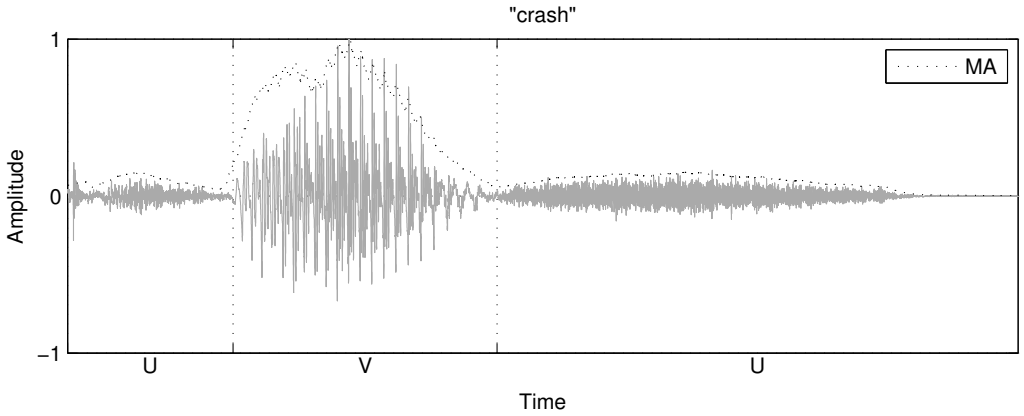

Figure 2.7: *"crash" utterance with MA outline again, but with a focus on the speech part.*

have to be resolved. This will be the subject of Section 2.2.2. But before that, let us look at the attribute functions that will serve as a basis for our discrimination.

2.2.1 Discriminatory attributes and functions

The attribute functions listed in this section are, for the first two, common to all textbooks I could come across that use algorithmic time-domain processing for voiced-unvoiced discrimination [Rabiner and Schafer, 1978, Kondoz, 2004]. The third, however, was particular to [Kondoz, 2004], in which textbook a number of other attribute functions are mentioned : periodic similarity, peakiness of speech, spectrum tilt and pre-emphasized energy ratio. However, some of these attributes are not straightforward and/or computationally cheap to compute. For this reason I have chosen the following three only, as they work well, and they are easy and computationally cheap to obtain. In fact, they can all be obtained as the output of FIR filters, which is very convenient to get in Matlab through the use of the `filter` function.

STE and MA again

Let us stick with our "crash" utterance. If you refer to Figure 2.7, you will find that the MA is generally lesser in unvoiced segments than it is in voiced

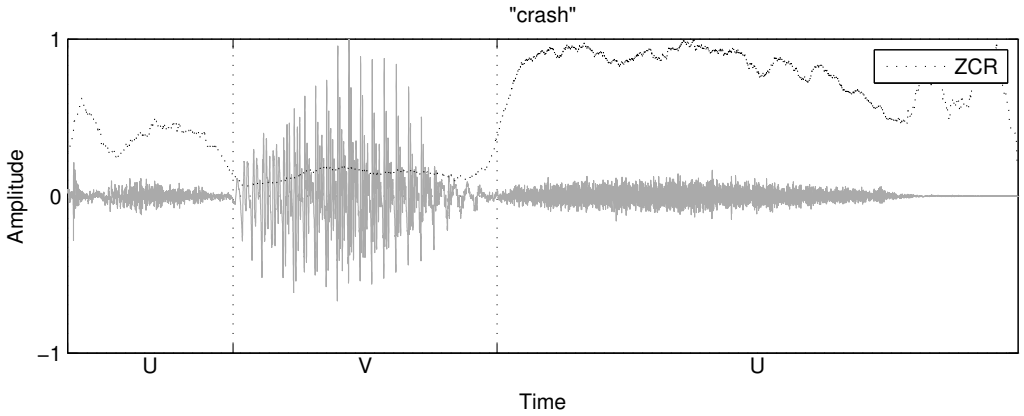

Figure 2.8: *The waveform in unvoiced segments is more "dense", more turbulent, oscillates faster than that of voiced segments. This is reflected in the Zero-Crossing Rate (ZCR).*

segments. We will therefore use the MA again, in the exact same was as described in Section 2.1.2.

Zero-Crossing Rate (ZCR)

Still looking at the "crash" waveform, you can notice that the unvoiced segments, although lower in energy, exhibit a denser waveform, more turbulent. An indicator of this "density" is the rate at which the waveform crosses the zero-axis, or *Zero-Crossing Rate* (ZCR). The ZCR of the "crash" utterance was normalised and superimposed to the waveform in Figure 2.8, and it can be seen that it is significantly higher in unvoiced segments. Formally, the ZCR function can be expressed as

$$\boxed{\text{ZCR}[n] = \sum_{m=0}^{N-1} |\text{sgn}(x[n-m]) - \text{sgn}(x[n-m-1])|} \text{ [Rabiner and Schafer, 1978],} \tag{2.9}$$

where sgn(.) is the *signum function*,

$$\text{sgn}(x) = \begin{cases} 1 & x \geq 0, \\ -1 & x < 0. \end{cases} \tag{2.10}$$

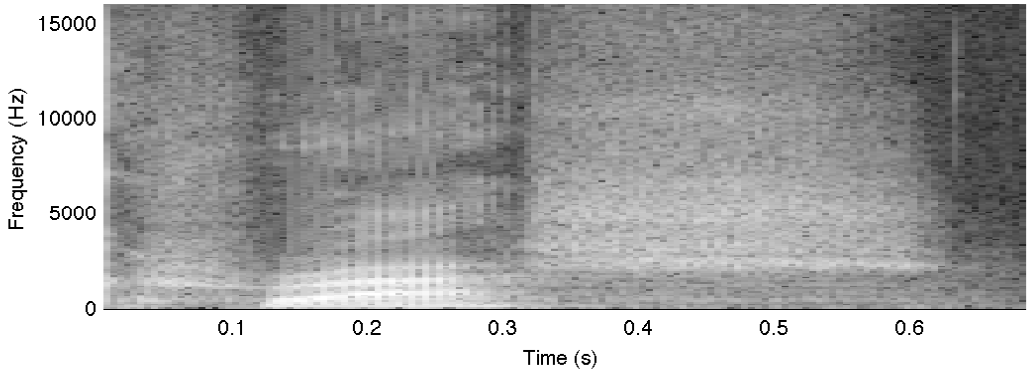

Figure 2.9: *Spectrogram of "crash". The energy density of unvoiced segments is low below 1kHz and becomes large above 1kHz, while the opposite is true for voiced segments.*

Now if you proceed to the substitution

$$y[n] = |\text{sgn}(x[n]) - \text{sgn}(x[n-1])|, \tag{2.11}$$

you will find that

$$\boxed{\text{ZCR}[n] = \sum_{m=-\infty}^{\infty} y[m]w[n-m]}, \tag{2.12}$$

i.e. that the ZCR is once again equivalent to the convolution of y with a rectangular window. Likewise, the ZCR can be equated with the output of an order $N-1$ FIR filter with ones for coefficients, fed with y.

Low-band/Full-band energy ratio (LF)

The last attribute we are going to consider here relates to the contrasting spectral distribution between voiced and unvoiced segments. Considering figure 2.9, you should notice that the unvoiced phonemes exhibit a broadband, smeared layer of energy starting from frequency 3kHz (approximately) and above. On the other hand, the magnitude spectrum of the voiced segment is seen to be denser in its lower part. These are characteristics visible in

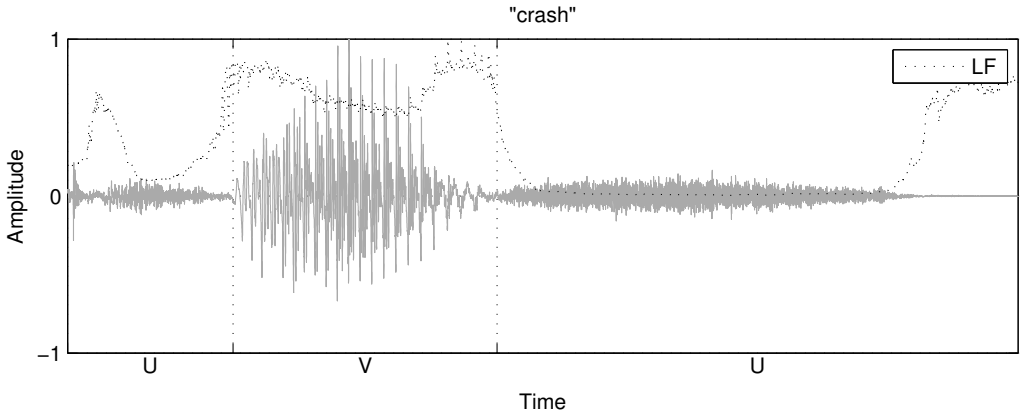

Figure 2.10: *The Low-band/Full-band energy ratio is clearly greater for voiced segments, is easy to obtain, and hence is a great candidate in voiced-unvoiced discrimination.*

the frequency domain, but which can be exploited in the time domain in a Low-band/Full-band energy ratio (LF), shown for the utterance "crash" in Figure 2.10. This is the ratio of the STE of a low-pass filtered version of the signal to that of the regular signal. The cutoff of the low-pass filter can be set at a frequency of 1kHz. Formally, this is

$$\mathrm{LF}[n] = \frac{\mathrm{STE}_{\mathrm{LP}}[n]}{\mathrm{STE}[n]}, \tag{2.13}$$

with $\mathrm{STE}_{\mathrm{LP}}$ being the STE of the low-pass filtered copy of the input x. Let us not delve into filter theory just now ; at a practical level, it will be explained how, in Matlab, you can obtain the appropriate filter coefficients and use them in the `filter` function to get this filtered signal.

2.2.2 Weight your arguments before coming to a decision

So we have chosen three attribute functions, which probably will be most of the time in agreement as to whether what time indices are voiced or unvoiced. However, it will not always be the case either, and for such situation, a decision-making algorithm should be elaborated.

Normalisation of attribute functions

The algorithm presented by Kondoz in [Kondoz, 2004] is "wise" in the sense that it takes into account not only whether an attribute function is above or below its discrimination threshold, but also how "certainly" the function claims the time indices to be voiced or unvoiced. For instance, looking back at Figure 2.6, it is obvious that the highest-reaching lobe of the MA corresponds to a speech segment, while the MA in its lowest parts denotes silence. However, as the threshold is neared, the certainty as to whether the speech is voiced or unvoiced lessens. In general, we'll agree that the higher above or lower under the threshold the attribute function is, the greater the degree of certainty.

To account for this degree of certainty, Kondoz normalises the attribute functions, such that the maximal value reaches 1, the minimal, -1, and the threshold is at 0. Thence, two (or more) normalised attribute functions can be added together, and that with the greatest degree of certainty wins, making the sum go positive if it is voiced, and negative if it is unvoiced. We now set to formulate this normalisation.

Let us call one of our attribute functions f, with $f_{\min}$ being its minimal value, and $f_{\max}$, its maximal. Let us call its normalised version g. To satisfy our criteria, g has to be a *composite* function, with a positive part to map the part above the threshold in the original function from 0 to 1, and a negative part doing the same on the negative side. Formally, we have

$$g(f) = \begin{cases} g_+(f), & f \geq T \\ g_-(f), & f \leq T \end{cases}. \tag{2.14}$$

The two components g_+ and g_- are *linear functions* of the input function f, of the form

$$g_+(f) = a_+ f + b_+, \tag{2.15}$$
$$g_-(f) = a_- f + b_-. \tag{2.16}$$

The coefficients a_+, a_-, b_+ and b_- have to be adjusted for our purpose. This is a simple matter of solving two-unknowns equations. We find that

$$g(f) = \begin{cases} \frac{f-T}{f_{\max}-T}, & f \geq T \\ \frac{f-T}{T-f_{\min}}, & f \leq T \end{cases}. \tag{2.17}$$

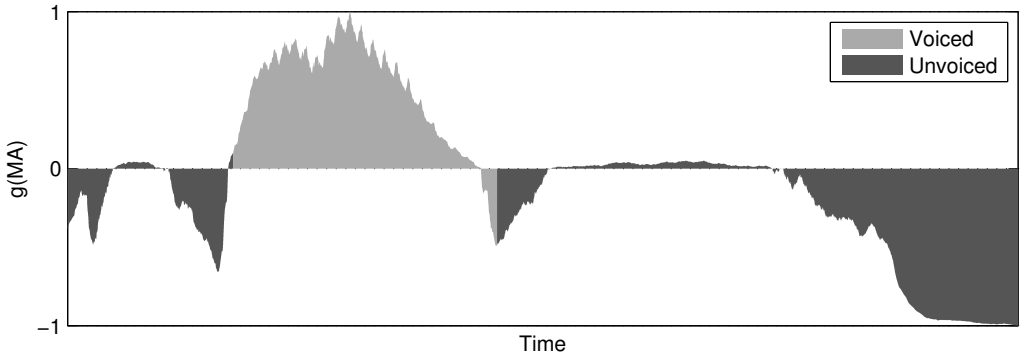

Figure 2.11: *Attribute function normalisation : the data of Figure 2.3 is linearly transformed to span* $[-1, 1]$ *and be 0-centered around the threshold.*

Just try and substitute $f_{\max}$, T, and $f_{\min}$ for f, and you will find that you get the results 1, 0 and -1, as intended. To complete this discussion, we give in Figure 2.11 a graphical example of the result of normalisation, with the normalised MA function seen already in Figure 2.3.

Decision making

The voiced-unvoiced discrimination can now be taken upon the summing of the values of our three normalised functions. We can make up a two-valued function, which is 1 everywhere this sum is positive, and 0 where negative, i.e.

$$\mathrm{VU}_{\mathrm{jit}}[n] = \begin{cases} 1 & g_{\mathrm{STE}}[n] - g_{\mathrm{ZCR}}[n] + g_{\mathrm{LF}}[n] \geq 0, \\ 0 & g_{\mathrm{STE}}[n] - g_{\mathrm{ZCR}}[n] + g_{\mathrm{LF}}[n] < 0. \end{cases} \tag{2.18}$$

The reason why, in (2.18), g_{ZCR} is negated is because, as opposed to MA and LF, values of ZCR *below* the threshold correspond to *voiced* segments. Also, the reason for the presence of the *jit* subscript in $\mathrm{VU}_{\mathrm{jit}}$ will become clear in the next subsection.

Jitter removal

A VU function such as seen in (2.18) can still exhibit *jitter*, i.e. unreasonably fast oscillations between the two states. The rate of phones in speech can be counted to about ten per second, and hence the rate of state change should be

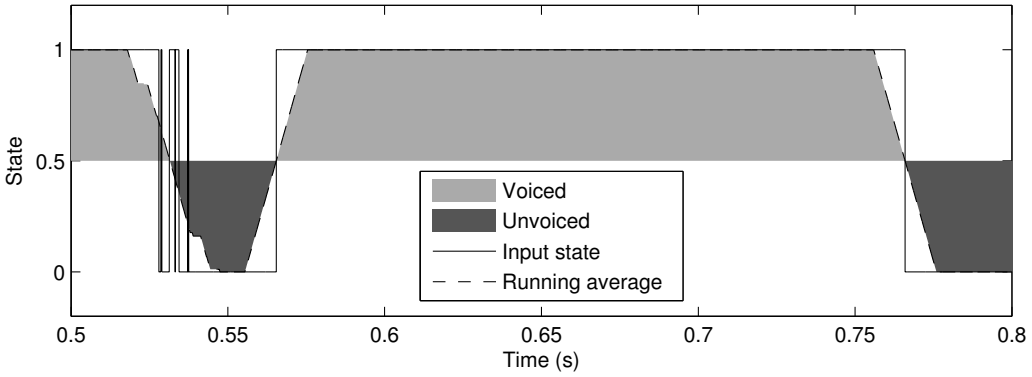

Figure 2.12: *Removing potential jitter in our VU function (solid line) using a Short-Time Average (dashed line). All time indices where the STA is above 0.5 are finally considered voiced.*

at most this. However, we plot in Figure 2.12 a snapshot of the $\mathrm{VU}_{\mathrm{jit}}$ function output by our current algorithm on an arbitrary speech signal. Between time 0.5 and 0.55 seconds, the jitter seen in the solid line is at a rate of a hundred changes per second, which is unreasonably high.

An easy and appropriate way of removing this jitter is through the use of a Short-Time Average (often called *running average*). This function is very similar to the short-time functions we have been seeing so far :

$$\mathrm{STA}_{\mathrm{VU}}[n] = \frac{1}{N}\sum_{m=0}^{N-1} \mathrm{VU}_{\mathrm{jit}}[n-m]. \tag{2.19}$$

The STA is superimposed in dashed lines to the input, jittery function in Figure 2.12. It is obvious to see there that all values for which the STA exceeds 0.5 should be considered voiced. Our final VU function therefore takes the form

$$\mathrm{VU}[n] = \begin{cases} 1 & \mathrm{STA}_{\mathrm{VU}}[n] \geq 0.5, \\ 0 & \mathrm{STA}_{\mathrm{VU}}[n] < 0.5. \end{cases} \tag{2.20}$$

2.2.3 Example result

I implemented the Kondoz algorithm, completed by our threshold-setting and jitter removal, as a Matlab program. To test the program, I used the

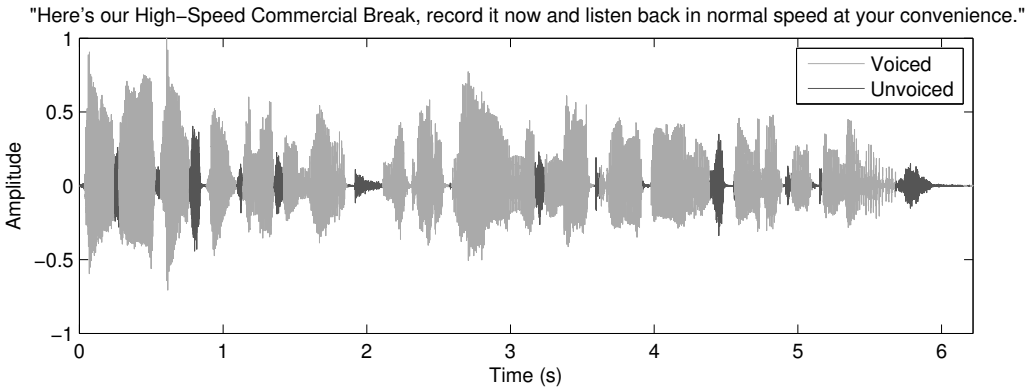

Figure 2.13: *Kondoz' (completed) algorithm run on a professional-quality speech recording. Only the "commercial break" utterance (about 1 second long) was used for training.*

speech recording heard in the track *High Speech Commercial Break* from Irish composer Roger Doyle's *Babel: KBBL* 1999 album. The entire recording says "Here's our High-Speed Commercial Break. Record it now and listen back in normal speed at your convenience." As training signal, I used "commercial break" alone. The algorithm was thereafter run on the entire recording, and the result can be visualised in Figure 2.13. The voiced-gated signal can be heard $\boxed{here}$ [1], and the unvoiced-gated, $\boxed{here}$ [2]. Most fun is to play back the unvoiced phones concatenated together, $\boxed{here}$ [3] :D

[1] `http://www.cs.nuim.ie/~matthewh/CS425/2012/sf/HSCB%20voiced.wav`

[2] `http://www.cs.nuim.ie/~matthewh/CS425/2012/sf/HSCB%20unvoiced.wav`

[3] `http://www.cs.nuim.ie/~matthewh/CS425/2012/sf/HSCB%20unvoiced%20concatenated.wav`

Chapter 3

Time-Domain pitch estimation

In this section, we are going to focus on the voiced segments of our speech signals, and examine ways of estimating their fundamental frequency, or *pitch*. [1] The fundamental frequency, inverse of the *fundamental period*, is the rate at which an oscillation starts a new cycle. Pitch, on the other hand, is the perceptual counterpart of the fundamental frequency. The American Standards Association defines it, in 1960, as "that attribute of auditory sensation in terms of which sounds may be ordered on a musical scale" [Moore, 2004]. As per its sensorial nature, it can hardly be measured, but when, in speech processing, we talk about "pitch measurement", we really are talking about measuring the fundamental frequency of a voiced speech sound.

It is well known that the intonation of speech is important to the process of conveying meaning. At first sight, it seems that intonation is based on three attributes : loudness, duration of syllables, and pitch contours. It can therefore be said that pitch contributes to the intelligibility of speech. Where speech processing is concerned, speech contours are use in [Rabiner and Schafer, 1978] for speaker recognition. It is obvious, on the other hand, that appropriate speech contours are indispensable to add realism to formant-based (i.e. purely synthetic, without the use of pre-recorded speech) speech synthesis.

Pitch comes across as an essential attribute to all paradigms in speech processing. In source-filter models, the vocal tract is modeled as resonant filters, to which is fed – for voiced sounds – a smoothened impulse train [Quatieri, 2002, Rabiner and Schafer, 1978, Kondoz, 2004]. The period of this impulse train, of course, is matched with the period of the air pressure oscillation through the glottis. In a different approach known as Sinusoidal Modeling Synthesis (SMS) [McAulay and Quatieri, 1986, III and Serra, 1987, Serra and Smith, 1990], a sum of sinusoids related harmonically in frequency (and to some extent, in amplitude as well) is seen as the major constituent of voiced speech sounds, the other being a low-level stochastic (i.e. noisy) constituent. The fact that these sinusoids are "harmonically related" means that the frequency of each is an integer multiple of the fundamental frequency. Hence, by knowing this fundamental frequency, one also gets to know, at least to some good approximation, the frequency of all the sinusoids. Getting to estimate the pitch is therefore the first step to SMS.

The time-domain method to pitch estimation that we are going to give the most attention is based on autocorrelation. The Average Magnitude Difference Function (AMDF) is also worthwhile looking at ; it is a method very

[1] We are still restricting our approaches to time-domain approaches.

close to the autocorrelation method, but which is computationally cheaper than time-domain-computed autocorrelation. However, we will see in Chapter 5 that the Fast Fourier Transform (FFT) can be used to compute autocorrelation at a significantly lower cost, beating even the computational efficiency of the AMDF. You should be aware, on the other hand, that these methods are not the only ones one could come up with for time-domain pitch estimation. In [Gold and Rabiner, 1969], for example, an algorithm which "encodes" a speech waveform as meaningful, synchronous impulse trains and aims at inferring the waveform's period from the periodicity of these impulses is developed. This method dates back from the late 60s, and is attractive for its light computational cost. The implementation is nevertheless far from being as straightforward as that of the autocorrelation and AMDF methods.

These two methods both exploit the fact that if a signal $x[n]$ is periodic, and that this period spans N_T samples, then the waveform is the same every other N_T samples, or

$$x[n] = x[n + kN_T], \quad \forall k \in \mathbb{Z}. \tag{3.1}$$

How each exploits this is the only place where the two methods differ.

3.1 Autocorrelation

The autocorrelation of a signal gives an indication of how alike itself a signal is when shifted. (What this means will become clear in the following.) It is a special case of cross-correlation, where it is two different signals that are compared as they are shifted one along the other. Correlation and autocorrelation are operations usually denoted with the *star* operator, $(x \star y)[n]$ denoting the correlation of x with y with a *lag* of n samples. For the autocorrelation of x, we can also use the xx shortcut, i.e.

$$xx[n] \doteq (x \star x)[n]. \tag{3.2}$$

The mathematical definition of autocorrelation is

$$xx[n] = \sum_{m=-\infty}^{\infty} x[m]x[m+n]. \tag{3.3}$$

In (3.3) x is seen as a theoretical signal, with a domain spanning infinity. $x[n]$ could be $\sin[n]$, or x^2, for example. However, for the infinite-interval

summation to *converge* (i.e. sum up to a finite value), x has to be a sinusoid, or a sum of them. In that case, the greatest autocorrelation value with be found at index 0, where it is simply the sum of the square of the values of x, in other words, its *energy* :

$$xx[0] = \sum_{m=-\infty}^{\infty} x^2[m]. \tag{3.4}$$

Also, if x is periodic and abides by Equation (3.1), then it is easy to see that $xx[n] = xx[n + kN_T]$ as well. Therefore, the autocorrelation maximum will also be found at index N_T as well as 0. Hence, finding this other peak is equivalent to finding the period of x, and thereby its frequency. This is what the autocorrelation method for pitch estimation aims at doing.

Equation (3.3) is nevertheless not practical for the analysis of finite-length signal. The summation shall not span the $[-\infty, \infty]$ interval, but a finite interval. Likewise, the autocorrelation indices n for which xx is non-zero will span a finite interval too. Working out the boundaries of these intervals is the purpose of sections 3.2 and 3.3.

3.2 Over what values of m?

In practice, the signals we are dealing with are of finite-length. In theory, we emulate this by multiplying our infinite-length signal x with a rectangular window (c.f. Equation (refeq:rectangular)). We denote this window-multiplied signal x_w, i.e. $x_w[n] \doteq x[n]w[n]$. It is this signal that we are now going to correlate with itself to get xx, as in

$$xx[n] = \sum_{m=-\infty}^{\infty} x_w[m]x_w[m + n]. \tag{3.5}$$

The writing of Equation (3.5) can be made more explicit, by limiting the interval of summation. Our rectangular window is 0 for all n outside the interval $[0, N[$, and consequently, so is x_w. By extension, this means that the product $x_w[m]x_w[m + n] \neq 0$ if $m \in [0, N[$ and $m + n \in [0, N[$. These two conditions can be summarised into one only, that $m \in [0, N[\cap[-n, N - n]$, or more concisely, $m \in [\max(0, -n), \min(0, -n) + N[$. We can on this basis

reformulate (3.5) using x instead of x_w as follows :

$$xx[n] = \sum_{m=\max(0,-n)}^{\min(0,-n)+N-1} x[m]x[m+n]. \tag{3.6}$$

Here we see that the autocorrelation of a signal of length N for a given autocorrelation index is the sum of at most N terms. Let us now see what indices the autocorrelation value of a finite-length signal is non-zero for.

3.3 Over what values of n?

In a sigma notation $\sum_a^b$, there are $b - a + 1$ terms added together. In other words, the lower limit a must be at most equal to the upper limit b, or $b \geq a$. This is of course valid for Equation (3.6). Our non-zero summation condition is there

$$\min(0, -n) + N - 1 \geq \max(0, -n). \tag{3.7}$$

We have to find the values of n for which (3.7) is true. For $n = 0$, it will be, so long as $N > 0$. The leftmost limit will therefore be for negative n, and the rightmost, for positive n. Where n is negative, $\min(0, -n)$ is 0 and $\max(0, -n) = -n$. We substitute these values into (3.7) and find that

$$n > -N. \tag{3.8}$$

Where n is positive, $\min(0, -n)$ is $-n$, $\max(0, -n)$ is 0, and so (3.7) becomes

$$n < N. \tag{3.9}$$

We now have the left and right boundaries of our non-zero-autocorrelation interval, $]-N, N[$.

Symmetry of autocorrelation

Realising that autocorrelation is symmetric about 0 is going to both alleviate the computation of autocorrelation signals and simplify its formulation. The symmetry property is that $xx[n] = xx[-n]$, shown by the substitution of p for $m + n$ in (3.6),

$$xx[n] = \sum_{p=\max(0,n)}^{\min(0,n)+N-1} x[p-n]x[p]. \tag{3.10}$$

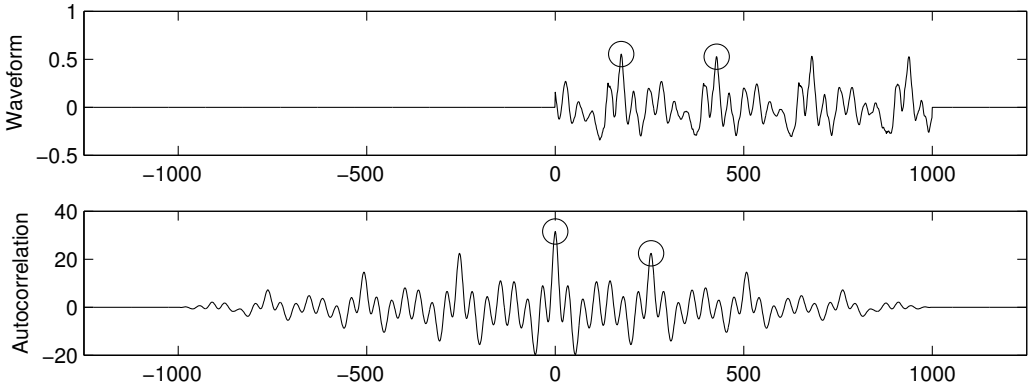

Figure 3.1: *A recangular-windowed quasi-periodic waveform and its autocorrelation.*

Because of this symmetry, (3.6) needs be computed for n in the interval $[0, N[$ only. Also, you will notice that it implies that the lower limit, $m = \max(0, -n) = 0$, and the upper limit, $\min(0, -n) + N - 1 = N - n - 1$. Our final formulation for autocorrelation is therefore simply

$$xx[n] = \sum_{m=0}^{N-1-n} x[m]x[m+n], \quad n \in [0, N[. \tag{3.11}$$

Figure 3.1 is here given as a visual support to this introduction on autocorrelation. The waveform (upper plot) is here multiplied by a rectangular window of length N. It is originally a near-periodic waveform, and the period was outline by the circling of two periodic maxima. The central peak and right-hand side largest peak were also circled in the cross-correlation (lower plot), and it is no coincidence that the time lapse between the circles in the waveform and autocorrelation are alike. Note, however, the steady decay trend in the autocorrelation, due to the windowing of the waveform. To ensure that, in spite of this decay, the autocorrelation peak corresponding to the period of the waveform is still second greatest after the central peak, the rectangular has to be long enough (at least twice the period of the correlated waveform, as we will see later). Finally, notice the symmetry of autocorrelation about 0.

3.4 Short-Time Autocorrelation

In Equation (3.11), the autocorrelation analysis of x spans the interval $[0, N[$, regardless of what the values of x outside this interval are. This analysis can be "slid", that is, taken over an interval $[m, m+N[$, for any integer m. For an index m, we can get an entire, N-length autocorrelation signal, and m can itself span an interval of length M, creating an autocorrelation *plane* of MN values. Tracking the greatest autocorrelation peak along the sliding index m has the potential of yielding the evolution of the period over time. In fact, we are making the assumption that the speech signal evolves slowly enough that its fundamental frequency can be considered constant over the N samples of the window, but we are aware that, on a longer term, this fundamental is very susceptible to change. It is these changes that we are interested in tracking, to get the aforementioned pitch contours.

We are now going from a standard autocorrelation notation to a Short-Time Autocorrelation (STAC) notation. The notation used in the following was directly inspired from that of the Short-Time Fourier Transform (STFT), a time-frequency analysis scheme that you are more than likely to encounter, should you pursue studies in signal processing.

Let

$$x[n, u] \doteq x[n]w[n - uR], \tag{3.12}$$

where R is known as the *hop size*, i.e. the number of samples between one analysis and the next. Consistently with the notation in Equation (3.12), we express the autocorrelation plane as

$$xx[n, u] = \sum_{m=-\infty}^{\infty} x[m, u]x[m + n, u] \tag{3.13}$$

$$= \sum_{m=uR}^{N-1-n+uR} x[m]x[m + n], \quad n \in [0, N[. \tag{3.14}$$

The result in (3.14) was obtained following the same development as in Section 3.2. You see that obtaining the autocorrelation signal $xx[n, u]$ is just a matter of shifting the analysis uR samples to the right. An example of an $xx[n, u]$ autocorrelation plane is given in Figure 3.2. Here, the autocorrelation is said to be *normalised* because the autocorrelation values of each frame were divided by the largest autocorrelation value, at index zero (i.e. $xx[n, u]$ were divided by $xx[0, u]$). Thus the central autocorrelation peak is always

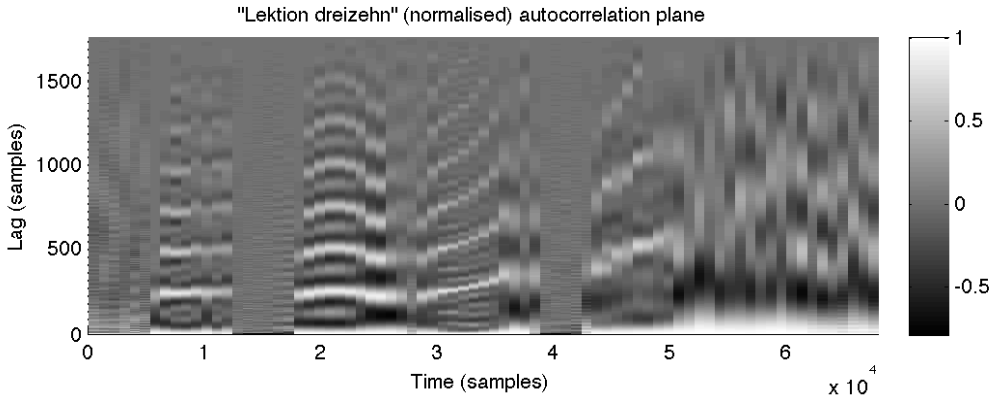

Figure 3.2: *Normalised-autocorrelation plane on German speech "Lektion dreizehn".*

1. Elsewhere, the segments where clean, bright, horizontal lines emerge are those where the waveform is the most periodic, where the vocal cords provide for most of the excitation of the vocal tract. In fact, the value to which the greatest non-central correlation peak rises can be taken as voiced-unvoiced discriminator. We just didn't use it in our VUS discrimination above because of the computational expense of autocorrelation.

Computational expense is also the reason why the hop size R is not simply 1. In this type of 2-dimensional analysis of sound signals, where the sampling frequencies can be very high, the bulk of data and calculations can easily become overwhelming even on nowadays computers. Instead of computing the autocorrelation every successive samples, we skip R of them between each analysis. This number R is known as the *hop size*. It is generally derived in terms of the length of the window, N, and a *window overlap factor*, that we here call O, as in

$$R = N/O. \tag{3.15}$$

O is generally set to 2, 4 or 8. For the autocorrelation function to be updated every $N/2$, $N/4$ or $N/8$ samples is generally sufficient to track the short-time variations of the pitch – recall the assumption that the pitch of the signal was assumed constant over a period of N samples anyway!

Autocorrelation, indeed, is a computationally expensive piece of analysis. In Equation (3.11), the computation of $xx[0]$, one sample of autocorrelation, takes N multiplications. That of $x[1]$ takes $N-1$, that of $x[2]$ takes $N-2$,

and so on. In general, you see that the computation of $x[n]$ takes $N - n$ multiplications. To see how many operations are implied in the calculation of an entire, one-dimensional autocorrelation signal, we sum up $N - n$ as n goes from 0 to $N - 1$, which gives

$$\sum_{n=0}^{N-1} N - n = \frac{1}{2}N^2 + \frac{1}{2}N. \tag{3.16}$$

In big-oh notation, you see that we have here a computational complexity of $O(N^2)$. If we were to perform this computation every single sample of an incoming signal of length M, then the computation would go $O(MN^2)$. We are better off not computing the autocorrelation for each sample index, and skip R of them each time instead, to give a total number of multiplications $\frac{M}{2R}(N^2 + N)$ for the generation of our autocorrelation plane. Strictly speaking, this is still a $O(N^2)$ complexity, and hence still not great. We will see, in Chapter 5, how it can be reduced to $O(MN \log N)$. We are not getting there just yet, however. In the meantime, we will see the Average Magnitude Difference Function, similar in computation and properties to autocorrelation, but where each term of the summation process involves some subtraction as opposed to multiplication, which is computationally cheaper.

3.5 Average Magnitude Difference Function (AMDF)

The AMDF is another popular means of estimating the period of a waveform in the time domain. For an infinite-length signal, it can be expressed as

$$d[n] = \sum_{m=-\infty}^{\infty} |x[m] - x[m + n]|. \tag{3.17}$$

The AMDF at index 0 is zero, and for a periodic signal x, so it is at every index that's an integer multiple of the period of the waveform. Hence, a method for estimating the period is simply to find the lowest AMDF value to the right of the central dip, and look up the corresponding lag index.

Similarly to autocorrelation, the AMDF beyond indices $]-N, N[$ is irrelevant (although this time, not necessarily 0), and $d[n] = d[-n]$. This brings us to formulating

$$d[n] = \sum_{m=0}^{N-1-n} |x[m] - x[m + n]|, \quad n \in [0, N[. \tag{3.18}$$

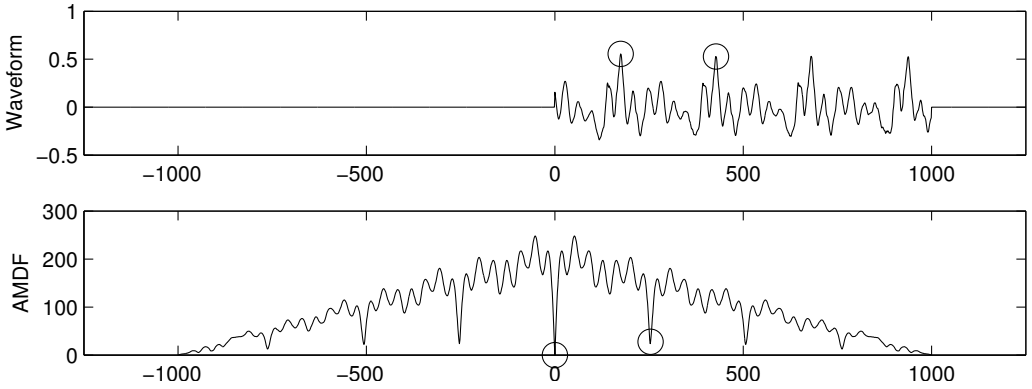

Figure 3.3: *Rectangular-windowed signal already seen in Figure 3.1 (upper plot) and its Average Magnitude Difference Function (lower plot).*

A plot of the AMDF on the windowed signal already utilised for Figure 3.1 is shown in Figure 3.3. The linear decay trend due to the windowing makes it inappropriate to look for the lowest value to the right of the central dip, as this would point at the edge of the AMDF signal rather than the period-related dip. However, the difference between the dips and the peaks on their either side can be used : the largest would still be, after the central dip, the dip corresponding to the period of the waveform.

3.6 Autocorrelation-based pitch estimate

In the autocorrelation of near-periodic, windowed signals, the peak corresponding to a time-lag of the period – which we may call thereon the *period's peak* – is the greatest after the central peak, provided that the window is sufficiently long (at least twice the fundamental period). In speech signals, this applies to the phones that, at a phonetic level, abide by the definition of vowels (c.f. Section 1.3). Some phones exhibit periodicity, but are also mixed with noise, as in, for example, voiced fricatives. The presence of this noise may compromise the prominence of the period's peak. Aperiodic sounds such as unvoiced fricatives, on the other hand, do not exhibit any autocorrelation peak, except for the zero-lag peak. In fact, the autocorrelation of a random signal spanning an infinite interval and mean 0 is 0 everywhere else but at lag 0. These three cases mentioned are illustrated in Figure 3.4, with the

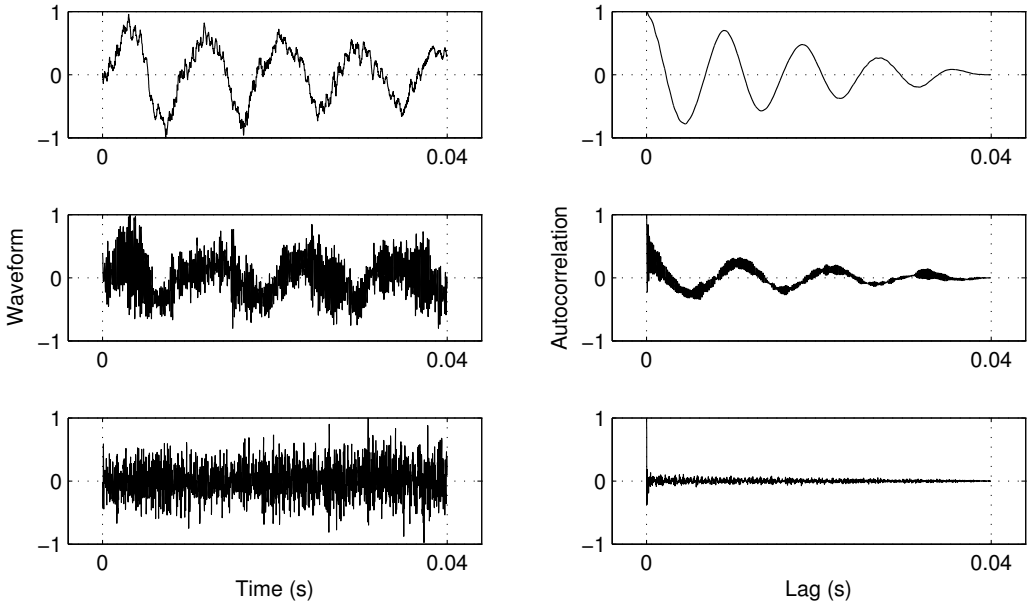

Figure 3.4: *Waveform and autocorrelation of voiced phoneme* /ʌ/ *("s<u>u</u>n"), voiced fricative* /z/ *("la<u>z</u>y"), and unvoiced fricative* /s/ *("<u>s</u>un")*

waveform and autocorrelation of voiced (upper plot), mixed (middle plot) and unvoiced phones (lower plot). See how the unvoiced excitation present in the waveform of voiced fricative /z/brings in turn noise in thee autocorrelation. This suggests that autocorrelation-based pitch estimation might benefit from preliminary low-pass filtering of the waveform, so as to clear the autocorrelation signals of spurious, noisy peaks.

This is all to say that in well-conditioned cases, the fundamental period is indeed found at the index of the second-greatest peak, but that this is not always the case. We can say that, for signals exhibiting some periodicity, the second-largest peak is most often the period's peak, but reliability in pitch tracking based on the sole detection of the largest peak is is not complete. Autocorrelation-based pitch detection might benefit from additional considerations. These will be the subject of this section, after a straight-forward maximum detection algorithm and demonstration is given in Section 3.7.

3.7 Maximum autocorrelation peak detection

In this section we describe a maximum-detection algorithm, and give a demonstration of pitch detection based exclusively on this algorithm, essentially to show its limits in the context of speech processing. The algorithm is well-known and straightforward, and most programming languages propose functions or methods of equivalent functionality. It is nevertheless shown here as an exercise in algorithm formulation. The algorithm's *signature* is

$$j = \max(\mathbf{a}), \tag{3.19}$$

i.e. given a vector $\mathbf{a}$ of length N, the function max() returns index j of largest entry in $\mathbf{a}$. If all elements in $\mathbf{a}$ are equal, i.e. $a_i = a_j \ \forall i, j \in [1, N]$, 1 is returned. Indexing in $\mathbf{a}$ starts at 1 ; consider that $a_i = 0$ if $i \notin [1, N]$. [2]

1. $i \leftarrow 1$, $j \leftarrow 1$, $A \leftarrow 0$.
2. While $i \leq N$, repeat the following.
3. If $a_{i-1} < a_i > a_{i+1}$ and $a_i > A$, $A \leftarrow a_i$ and $j \leftarrow i$.
4. Increment i.

Recall however that, in the case of autocorrelation, the largest peak is found at lag 0, while this peak is wanted out of the detection. The solution is to use as input to max() a vector of the same length as the autocorrelation vector, that is 0 at index 1 and for whichever indices the autocorrelation value is not a local maximum, and which keeps these autocorrelation values elsewhere. This vector is easier described algorithmically :

$$\mathbf{b} = \text{peaks}(\mathbf{a}) \tag{3.20}$$

For an input vector $\mathbf{a}$ of length N,

1. Create a zero-vector $\mathbf{b}$ of length N.
2. For each $i \in [2, N-1]$, repeat the following.
3. If $a_{i-1} < a_i > a_{i+1}$, $b_i \leftarrow a_i$.

[2] In this algorithm's notation, the $\leftarrow$ symbol is the *assignment operator*.

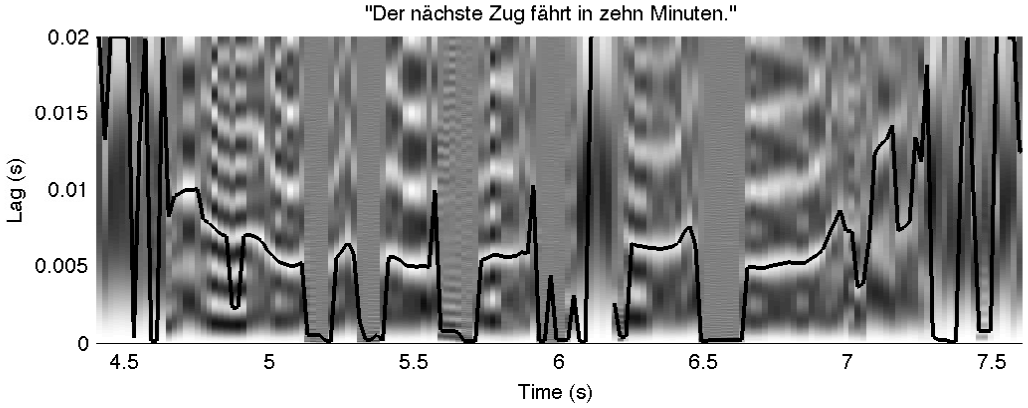

Figure 3.5: *Autocorrelation plane of a speech sequence and peak tracking (black line). The period information returned for segments that are not sufficiently periodic can be unrealistic and erratic.*

After that, for each autocorrelation frame, the index of the autocorrelation can be found as $i = \max(\text{peaks}([xx[0,u], xx[1,u], ..., xx[N-1,u]]))$, and the indices be used as a "period track". This approach was used upon the autocorrelation plane of a speech sequence available for audition $\boxed{here}$ [3]. In Figure 3.5, the autocorrelation peak tracking is highlighted with the black line. A band-limited $\boxed{sawtooth - wave}$ [4] whose pitch follows the pitch of this sequence as tracked by this method was generated. As the hop-size R of our autocorrelation plane is not 1, linear interpolation was used to determine the pitch of the sawtooth wave between the center of each autocorrelation frame. As for its amplitude, it is the normalised Short-Time Energy of the 1kHz low-passed speech signal.

The estimation of the pitch of unvoiced segments is futile and, as can be seen in Figure 3.5, gives unrealistic and/or erratic results. Above second 6.5, for example, the indicated period well below 1 millisecond corresponds to a frequency well beyond 1kHz, which is unrealistic. Where erratic jumps are concerned, median-filtering is sometimes used to smoothen the period track, i.e. the median of the peak indices surrounding the current peak index is put in place of this peak index. Outliers are thus efficiently removed, but this

[3] `http://www.cs.nuim.ie/~matthewh/CS425/2012/sf/dialogue2.wav`

[4] `http://www.cs.nuim.ie/~matthewh/CS425/2012/sf/pitch_maxonly.wav`

method has the inconvenience that the output might not correspond exactly to the indices where the peaks actually lay, especially near the ends of voiced segments. To keep transparency to an optimum, it may be preferable to exclude from the search peaks that are known not to be reasonable candidates. This is the subject of Section 3.8.

3.8 Reducing the scope of the search

It is evident that no pitch peak is to be found for autocorrelation frames where the signal is aperiodic – in unvoiced segments – or where no signal is to be found altogether – in silence segments. The end-product of our study of Voiced-Unvoiced-Silence discrimination can here be found some use already, in excluding from the pitch tracking the autocorrelation frames found in unvoiced or silence segments.

The scope can be further restricted to the time-lag regions corresponding to periods realistic to be encountered in human speech. The highest fundamental frequency a human being can be found to produce by the means of his/her vocal cords is known to be in the vicinity of 1kHz, and the highest, 40Hz. These pitches correspond to periods of 1 and 25 milliseconds. If the type of speaker is not known *a priori*, the region of search for autocorrelation lag can be restricted to this interval ; it can nevertheless be restricted more if the vocal range specific to the speaker whose speech is analysed were known.

Finally, it has been discussed already that, the more periodic the analysed speech segment was, the highest the period's peak. Below a certain level, autocorrelation peaks can be found unreliable. On this basis, an autocorrelation peak detection threshold can be set, below which peaks are excluded from the search. This value, determined heuristically (i.e. upon the observation of the data at hand), is likely to be in the range of 0.1 to 0.2, for normalised autocorrelation signals. This is nevertheless subject to adjustment depending on the length of the window used, and possibly as well on the Signal-to-Noise Ratio of the recording environment.

In Figure 3.6, the peak-detection scope was reduced to segments classified as voiced in our VUS discrimination, limited to the range 50 to 500Hz, and (normalised) autocorrelation peaks below 0.2 were excluded from the search. The principal gain here is the clean-up of estimates at irrelevant times, and in this regard, our VUS discrimination seems to be the prime contributor to the quality enhancement. On the other hand, some undesirable jumps can still

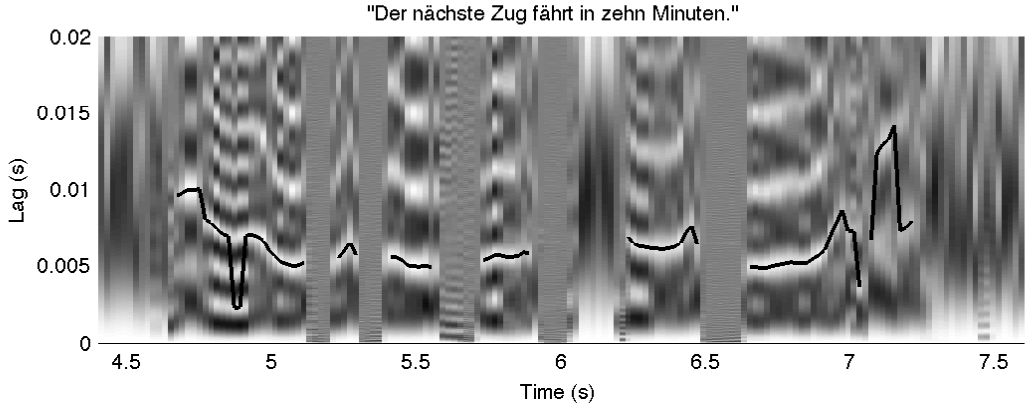

Figure 3.6: *Period tracking based on the largest-peak assumption, but restricted to voiced segments and with lag and peak thresholds. A sawtooth wave following the corresponding pitch variations is available* *here.*

be seen : that shortly before second 5, and those at and after second 7. These instances show that the law of the largest peak is not always the right one. I briefly propose here an autocorrelation-based pitch-tracking algorithm which conciliates two assumptions instead of one : that the largest peak is *generally* the period peak, and that period peaks of adjacent frames are closest.

3.9 Description of a "heavy-track" algorithm

The starting point of the autocorrelation-based pitch tracking algorithm presented in this section is the following assumption :

> If two peaks in adjacent frames are in the continuity of one another, then the peak closest to either in the other's frame is that other peak.

Peaks in the continuity of one another form *tracks*, a track being defined by the succession of two or more peaks abiding by the above rule. [5]

[5]Note, however, that the reciprocal is not necessarily true, meaning that in adjacent frames, the peaks closest to one another are not necessarily each other's continuation. For this reason it is desirable to set such peak detection thresholds in our autocorrelation plane

There remains yet to clarify what being the closest peak to another peak means. Let $\mathbf{A}_{i,j}$ be a peak in matrix $\mathbf{A}$ (i.e. $\mathbf{A}_{i-1,j} < \mathbf{A}_{i,j} > \mathbf{A}_{i+1,j}$), and $\mathbf{A}_{k,l}$ be another peak in another column. A vertex may be traced between $\mathbf{A}_{i,j}$ and $\mathbf{A}_{k,l}$ if :

- the columns are adjacent, i.e. $j = l \pm 1$,
- the peak $\mathbf{A}_{k,l}$ is strictly the closest to $\mathbf{A}_{i,j}$, i.e. there isn't a peak $\mathbf{A}_{m,l}$ such that $|m - i| \leq |k - i|$,
- reciprocally, the peak $\mathbf{A}_{i,j}$ is strictly the closest to $\mathbf{A}_{k,l}$, there isn't a peak $\mathbf{A}_{m,j}$ such that $|m - k| \leq |i - k|$.

Contiguous vertices are thereafter concatenated in tracks. As means of example, let us consider the following sparse matrix

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \tag{3.21}$$

standing as our autocorrelation plane, and whose non-zero entries stand for peaks. The result of joining the non-zero entries as per the above rules is shown in Figure 3.7. In the figure, the coordinate (6,1) is not tied to (6,3) because not they do not stand in adjacent columns (rule 1.). (6,1) is not tied to (7,2) or (5,2) because it is a distance equal to the both of them (rule 2.). Finally, the closest from (7,2) in column 3 is (5,3), but a vertex is not drawn because the closest from (5,3) in column 2 is (5,2), breaking second condition, breaching the reciprocity of conditions 2. and 3.

Tracks may be denoted as sets of tuples, the first and second entries of each tuple representing matrix row and column indices. In our above example, the set describing the topmost track would be $T = \{(1,1),(1,2),(2,3)\}$. The ensemble of the tracks draw may be collected in $\mathbf{T}$, a set of track sets, where $\mathbf{T}_i$ is the i^{th} track in the set. In our example, we would have

as those mentioned in Section 3.8, otherwise tracks might reach to peaks which shouldn't belong.

$\mathbf{T} = \{\{(1,1),(1,2),(2,3)\},\{(3,1),(4,2)\},\{(5,2),(5,3)\}\}$, and, for example, $\mathbf{T}_3 = \{(5,2),(5,3)\}$.

Figure 3.8 illustrate the process as applied to a full-size autocorrelation plane. Also, a documentation on the processing of a matrix into such tracks, with detailed algorithms, can be found in Appendix A.

Up to this point, tracks that are overlapping are likely to be seen, as in Figure 3.8. As there can be one period estimation at a time only, it has to be decided for each frame which track is to be followed. This is where we exploit two other observations :

> for a windowed near-periodic waveform, the period's peak is generally the largest,

and

> the tracks matching the actual pitch evolution tend to be longer.

Both these qualities can be combined together by measuring the *weight* of each track, or the sum of the autocorrelation values that it contains. Thus, the greater and more numerous the peaks that constitute a track, the greatest its weight. Our approach is henceforth simple : for a given frame, the selected peak will be the peak belonging to the overlapping track with the most weight.

The autocorrelation's set of tracks is thereby going to be processed into a set $\mathbf{C}$ of *pitch contours*. Contours are like tracks, sets of index-tuples, except that contours within one same set do not overlap. Also, when, from one frame to another, the selected track changes, a contour is terminated and a new one created, as it is excluded that a pitch contour spans more than one track. As a result, contours are subsets of tracks, i.e. $\mathbf{T}_i \subseteq \mathbf{C}_j$.

Here follows an informal algorithmic description of the creation of contour set $\mathbf{C}$ from track set $\mathbf{T}$:

1. For each column in the autocorrelation plane, repeat the following.

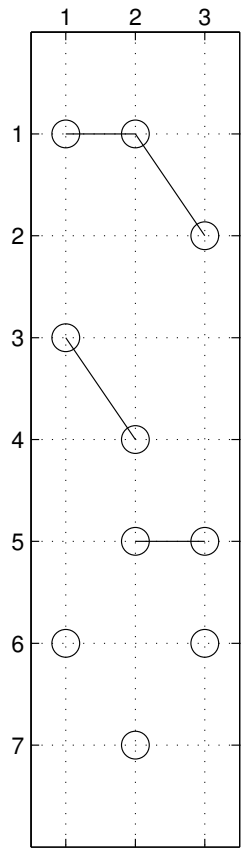

Figure 3.7: *Track-tracing on a reciprocal, strict closest-peak basis.*

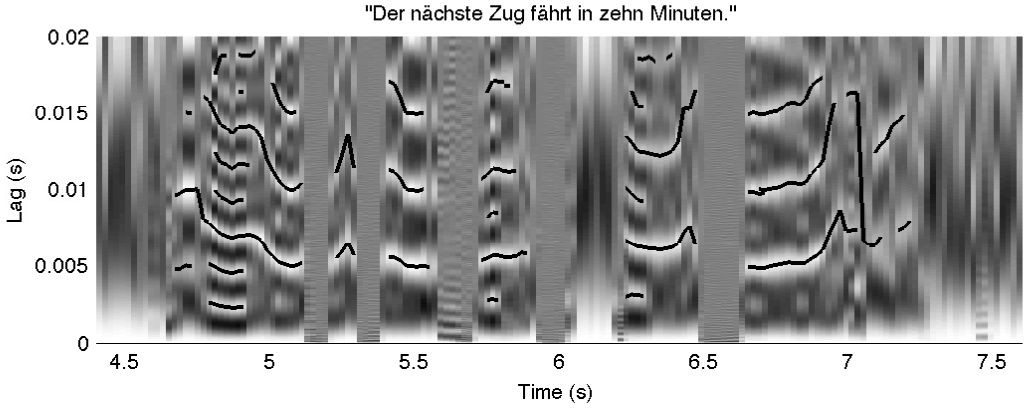

Figure 3.8: *Autocorrelation tracks traced as per the described procedure.*

2. Find the indices of the tracks which overlap with the current column. If there is any, proceed.
3. Within the selected tracks, select that which has the greatest weight.
4. If this track differs from the track selected in the previous column, create a new segment.
5. Concatenate to the tail of the newest segment the entry of the selected track whose column index matches the current column.

This procedure leaves with the steadiest tracks, which represent the period evolutions of the voiced segments in a seemingly reliable way. Demonstration is given in Figure 3.9. As per the previous examples of pitch tracking, a pitch-synchronous sawtooth wave was generated, and is available *here*. [6]

[6] `http://www.cs.nuim.ie/~matthewh/CS425/2012/sf/pitch_closest.wav`

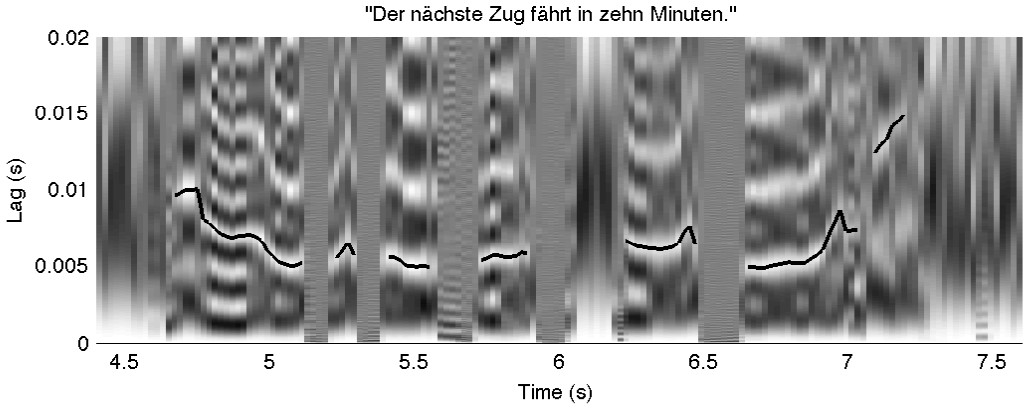

Figure 3.9: *Period contours obtained with our heavy-track algorithm.*

Chapter 4

Formants in Linear Prediction

Linear Prediction (LP) is a very smart filter design technique, whereby the resonant cavities of the vocal apparatus are modeled with an *all-pole* (this term will be explained soon) filter. The underlying model is faithful, the results, very satisfactory, and the applications, numerous. It is indeed a key technique where speech processing is concerned. However, it relies heavily on the theory of filters. A fairly comprehensive and rigorous introduction to this theory is therefore necessary. Rest assured that what you will learn in this introduction is widely applicable throughout signal processing. (In fact, it seems that all signal processing comes down to filter theory.) Also, it will give you another glimpse at the workings of the frequency domain. Filtering, indeed, might be done in the frequency domain as well as in the time domain – the equivalences are numerous. Filter specifications actually are most often relevant to the frequency domain, though not always. But enough talking, let us get started.

4.1 Introduction on filters

A filter is a digital system whereby an output waveform is to be created by the filtering of an input. There exists a number of types of filters, and they do all sorts of things. Where musical effects are concerned, for example, artificial reverberation is done by the filtering. Distortion, chorus, *wha-wha*, likewise, are all done with filtering. In fact, the Short-Time Energy (2.1), Magnitude Average (2.2), Zero-Crossing Rate (2.9) or Low-band/Full-band energy ratio (2.13) are all signals that can be seen as the output of filters. However, these are non-linear filters. In this introduction, we are going to concentrate on *Linear Time-Invariant* (LTI) filters.

LTI filters are the most popular of filters because of the nice properties which make their analysis and design relatively convenient. By *linear* is meant that, on the one hand, the amplitude of the output waveform is proportional to that of the input waveform, and on the other hand, that the filtering of the sum of two signals equals the summed filtering of each signal. Mathematically, we write

$$\mathrm{F}\{ax_1[n] + bx_2[n]\} = a\mathrm{F}\{x_1[n]\} + b\mathrm{F}\{x_2[n]\}, \tag{4.1}$$

where $y[n] = \mathrm{F}\{x[n]\}$ is the signal produced by the filtering of x by F. Aside from the STE-and-the-likes examples of non-linear filters, typical filters that

do not abide by property (4.1) are filters that emulate the overdrive on guitar amplifiers.

By *time-invariant* is meant that the output for a given waveform will be the same regardless of the moment in time when the filtering is performed. This property is expressible by

$$y[n] = \mathrm{F}\{x[n]\} \iff y[n - n_0] = \mathrm{F}\{x[n - n_0]\}. \tag{4.2}$$

An example of a filter that's not time-invariant would be a filter that implements vibrato, and all its derivations : chorus, flanger, etc. There, the input waveform is delayed by an amount of time that's dependent on the time of arrival, and hereby not time-invariant.

4.1.1 Difference Equations

The *difference equation* for an LTI is as follows :

$$\begin{aligned} y[n] = b_0 x[n] + b_1 x[n-1] + b_2 x[n-2] + ... + b_P x[n-P] \\ - a_1 y[n-1] - a_2 y[n-2] - ... - a_Q y[n-Q], \end{aligned}$$

or

$$y[n] = \sum_{k=0}^{P} b_k x[n-k] - \sum_{k=1}^{Q} a_k y[n-k]. \tag{4.3}$$

LTIs can therefore be seen as computing the current output sample as the weighted sum of the current and past input samples and the past output samples. The filters is subdivided in two "sections", the feedforward section, which uses the samples of the input x, and the feedback section, which uses the samples of the output y. The weights, or *coefficients*, to the input samples are

$$b_0, b_1, ..., b_P, \tag{4.4}$$

and P is called the *order* of the feedforward section. The coefficients of the feedback section are

$$a_1, a_2, ..., a_Q, \tag{4.5}$$

and likewise, Q is the order of the feedback section. Notice that the list of feedback coefficients begins at subscript 1, while that of feedforward coefficients, at subscript 0.

4.1.2 Impulse Responses and Transfer Functions

The impulse response of a filter is the signal obtained by feeding the filter with an impulse $\delta[n]$, the *delta* function, defined as

$$\delta[n] = \begin{cases} 1, & n = 0, \\ 0, & n \neq 0. \end{cases} \tag{4.6}$$

Substituting δ for the input x in the LTI general expression (4.3) yields the impulse response, usually denoted with the letter h :

$$h[n] = \sum_{k=0}^{P} b_k \delta[n-k] - \sum_{k=1}^{Q} a_k h[n-k]. \tag{4.7}$$

LTI filters have the characteristic that they are *completely determined* by their impulse response. This means that, knowing the impulse response for all indices of time n for which it is non-zero, or having a specific formula for this expression (e.g. such as $h[n] = a^n$), allows the complete analysis of the corresponding filter. By the frequency analysis of this special signal, one gets to know how the filter affects the magnitude and phase of the sinusoidal components of the input signal. In fact, if the filter coefficients are known, the spectrum of the impulse response is easy to derive,

$$H(\omega) = \frac{\sum_{k=0}^{P} b_k e^{-jk\omega}}{1 + \sum_{k=1}^{Q} a_k e^{-jk\omega}}. \tag{4.8}$$

In (4.8), ω is a symbol used to denote *angular frequency*, in radians per time unit (i.e. seconds if in continuous-time, samples if in discrete-time). For a sampling rate f_s, the angular frequency ω will correspond to a frequency in Herz f as per

$$\omega = 2\pi f / f_s. \tag{4.9}$$

Finally, $e^{j\phi}$ is the *complex exponential*, j being the *imaginary unit* – in an engineering context, j traditionally replaces the i found in mathematics. e is the base to the natural logarithm, a mathematical constant approximated by 2.71828. It has the special property that

$$e^{j\phi} = \cos\phi + j \sin\phi, \tag{4.10}$$

which, on the plane of imaginary numbers, situates a point at distance 1 from the origin and makes an angle ϕ with the positive side of the horizontal axis, as shown in Figure 4.1.

Equality (4.10) is known as *Euler's equality*. It might not make too much sense intuitively, but this equality can be proven with Taylor Series. Complex exponentials are used for the analytical expression of sinusoids. For instance, such complex exponential $Ae^{j(\omega n+\phi)}$ would be used to describe a sinusoid of magnitude (i.e. amplitude, sort of) A, frequency ω radians per sample, and initial phase ϕ. Although we are mostly dealing with real signals, this form might be preferred to $A\cos(\omega n + \phi)$ for the numerous nice properties that complex exponentials have, both in the time and frequency domain.

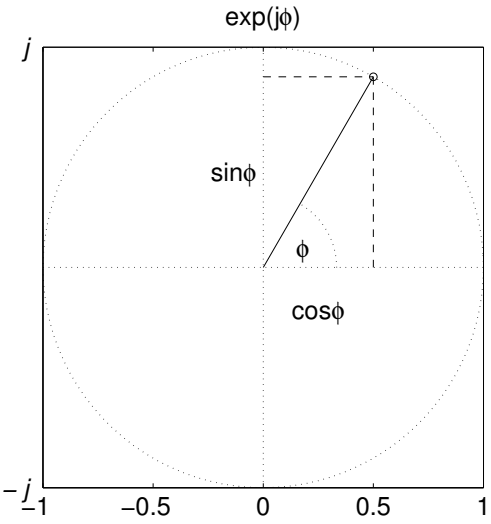

Figure 4.1: *Graphical representation of complex exponential $e^{j\phi}$, for an arbitrary angle ϕ.*

Coming back to impulse responses of filters and their spectra, the spectrum of the impulse response of a filter, (4.8), is known as the filter's *transfer function*. The transfer function of a filter has the property that it is periodic in 2π, i.e.

$$H(\omega) = H(\omega + n2\pi), \quad \forall n \in \mathbb{Z}. \tag{4.11}$$

This is easy to show. Note that in the general transfer function expression, (4.8), the argument ω of $H(\omega)$ only appears in the complex exponential, $e^{-jk\omega}$, so to demonstrate (4.11) comes down to demonstrating that

$$\begin{aligned} e^{-jk(\omega+n2\pi)} &= e^{-jk\omega}e^{-jkn2\pi} \\ &= e^{-jk\omega}(\cos(-kn2\pi) + j\sin(-kn2\pi)) \\ &= e^{-jk\omega}(1+0) \\ &= e^{-jk\omega}, \end{aligned}$$

given that $kn \in \mathbb{Z}$. Another property, that can likewise be shown, is that, provided real filter coefficients, the negative-frequency side of the transfer function is the *complex conjugate* of the positive side, i.e.

$$H(\omega) = H^*(-\omega). \tag{4.12}$$

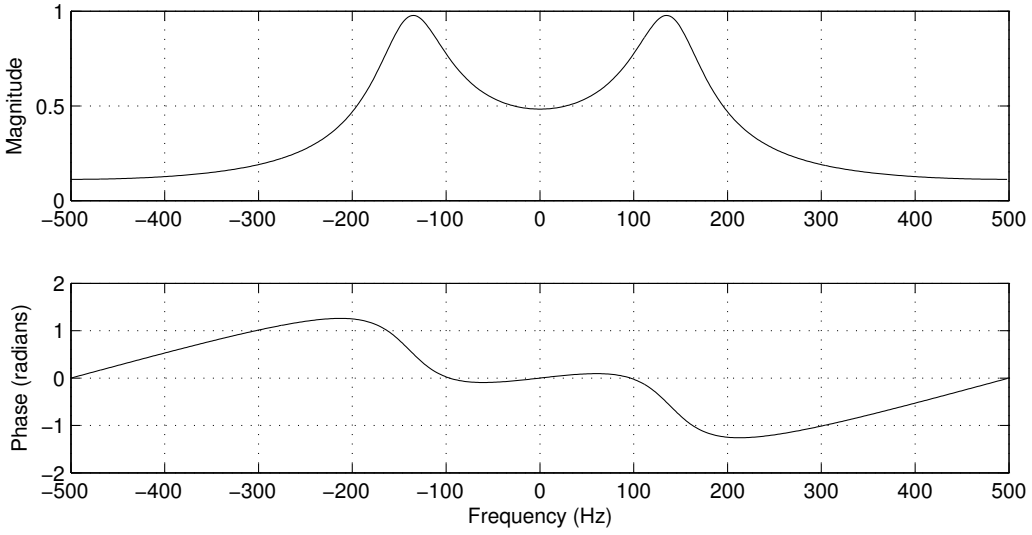

Figure 4.2: *Magnitude (top) and phase (bottom) responses of an arbitrary filter.*

The superscript asterisk denotes complex conjugation, as in $(a+ib)^* = a-ib$. The complex conjugate of a complex number is the same number, except with the imaginary part negated. With complex exponentials, it is only a matter of negating the exponent, i.e. $(e^{j\phi})^* = e^{-j\phi}$. As you can see here, The consequence of complex conjugation on a number is that the angle is negated, i.e. $\angle e^{-j\phi} = -\angle e^{j\phi}$, but the magnitude remains the same, i.e. $|e^{-j\phi}| = |e^{j\phi}|$.

Transfer functions are useful representations of filters for their analysis. For instance, the angle of a filter's transfer function,

$$\angle H(\omega) = \tan^{-1} \frac{\mathcal{I}\{H(\omega)\}}{\mathcal{R}\{H(\omega)\}}, \tag{4.13}$$

tells us what phase delay would undergo an input sinusoid of frequency ω, and whose magnitude,

$$|H(\omega)| = \sqrt{\mathcal{R}\{H(\omega)\}^2 + \mathcal{I}\{H(\omega)\}^2}, \tag{4.14}$$

tells us what amplitude scaling would undergo that same sinusoid. $\mathcal{R}\{H(\omega)\}$ and $\mathcal{I}\{H(\omega)\}$ denote the real and imaginary parts of $H(\omega)$, respectively.

Figure 4.2 gives an example of the transfer function of a filter. These graphs tell us, for instance, that a sinusoid of frequency 200Hz sent as input

would come out of the filter with its amplitude attenuated by a factor of slightly less than 0.5, and that its phase would be delayed by about 1.25 radians. Notice as well the properties of complex conjugation – same magnitude, opposite angle on either side of the zero-axis – and periodicity. (When the frequency axis is presented in hertz as opposed to radians, the periodicity is in the sampling frequency, f_s, which here is 1000, as opposed to 2π.)

4.1.3 Finite Impulse Response filters

Finite Impulse Response filters are filters whose impulse response is 0 for any time index n greater than a finite integer P, i.e.

$$h[n] = 0, \quad n > P. \tag{4.15}$$

For condition (4.15) to be met, the filter must have a feedforward section but no feedback section. In other words, the order of the feedback section must be 0. To recapitulate, here is the general form of difference equation for FIR filters,

$$y[n] = \sum_{k=0}^{P} b_k x[n-k], \tag{4.16}$$

and that of their transfer function,

$$H(\omega) = \sum_{k=0}^{P} b_k e^{-jk\omega}. \tag{4.17}$$

FIRs have the property that their impulse response can be read directly as their set of coefficients, i.e.

$$h[n] = b_n. \tag{4.18}$$

Also, know that FIR filters may be call *all-zero* filters. We won't really explain why here, only that filter transfer functions are normally not analysed only in terms of frequency, ω, but in terms of a complex exponential with arbitrary magnitude, z – hence the recurrent term of Z transform. The analysis domain therefore becomes the entire complex plane, as opposed to an interval spanning 2π where the transfer function repeats itself. On this plane, there are P zeros for a P-order FIR filter. If these zeros were found on the *unit circle*, (c.f. the dotted circle in Figure 4.1), then the transfer function as expressed in (4.8) would be 0 at the corresponding frequencies.

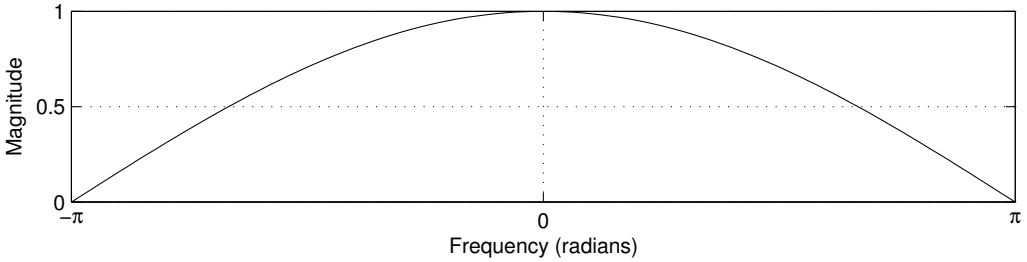

Figure 4.3: *Magnitude response of first-order averaging filter.*

Example

A most simple example of an FIR is the 1st-order averaging filter, where the output sample is computed as the average of the current and past sample, i.e.

$$y[n] = \frac{1}{2}(x[n] + x[n-1]). \tag{4.19}$$

You see that past output samples are not used in the difference equation, which makes this filter, indeed, an FIR. The two non-zero coefficients are $b_0 = b_1 = 1/2$, and the impulse response is just the reading of these coefficients, $h[n] = \{1/2, 1/2, 0, 0, 0, ...\}$. The transfer function is thus

$$H(\omega) = \frac{1}{2}(1 + e^{-j\omega}). \tag{4.20}$$

As every first-order FIR, the transfer function has a zero somewhere on the complex plane. Here I can tell you that it is on the unit circle, and that you will therefore find it in the transfer function (4.20). What is needed, here, is therefore to find the value of ω for which $e^{-j\omega} = -1$. This value is π, as $e^{-j\omega} = \cos(-\pi) + j\sin(-\pi) = -1 + j0$. The magnitude response (i.e. the magnitude of the transfer function) is shown in Figure 4.3.

4.2 Infinite Impulse Response filters

Infinite Impulse Response (IIR) filters are so called because their impulse response never settles to zero after its first non-zero value. If this value occurs at $n = 0$, like it most often does, then we can write that $h[n] \neq 0$,

$n \geq 0$. For a filter to be an IIR, it must feature a feedback section. There might be a feedforward section as well, it will still be an IIR.

Feedback coefficients are responsible for the introduction of *poles* in the complex plane, that is, places in the plane where the transfer function is infinity because divided by 0. Hence, feedback filters that do not feature a feedforward section might be called *all-pole* filters. The difference equation of these all-pole filters can be generalised as

$$y[n] = b_0 x[n] - \sum_{k=1}^{Q} a_k y[n-k], \tag{4.21}$$

and their transfer function, as

$$H(\omega) = \frac{b_0}{1 + \sum_{k=1}^{Q} a_k e^{-jk\omega}}. \tag{4.22}$$

The coefficient b_0 is an optional scaling factor. It might be 1, or any other scalar ; the feedforward section still remains of order 0.

The rule of thumb in the design of feedback filters is to keep these zeros within the unit circle. If one such zeros is either on or outside this circle, the filter becomes *unstable*, which, in practice, means that it will sooner or later blow up when fed with a non-zero input. It will become obvious why in the following example.

Example

Let us consider the IIR defined by the difference equation

$$y[n] = x[n] + ay[n-Q]. \tag{4.23}$$

The output of this filter is its input plus its output delayed by Q samples and multiplied by a. So if it were fed by an impulse (4.6), the first output sample would be 1, and then, Q samples later, it would be a, and another Q samples later, a^2, and so on. You can see that the impulse response is

$$h[n] = a^{n/Q}, \quad n = kQ, \forall k \in \mathbb{N}, \tag{4.24}$$

and 0 elsewhere. You see that so long as a is lesser than 1, $h[n]$ will converge towards 0. If a is 1, however, the impulse response will not converge, and

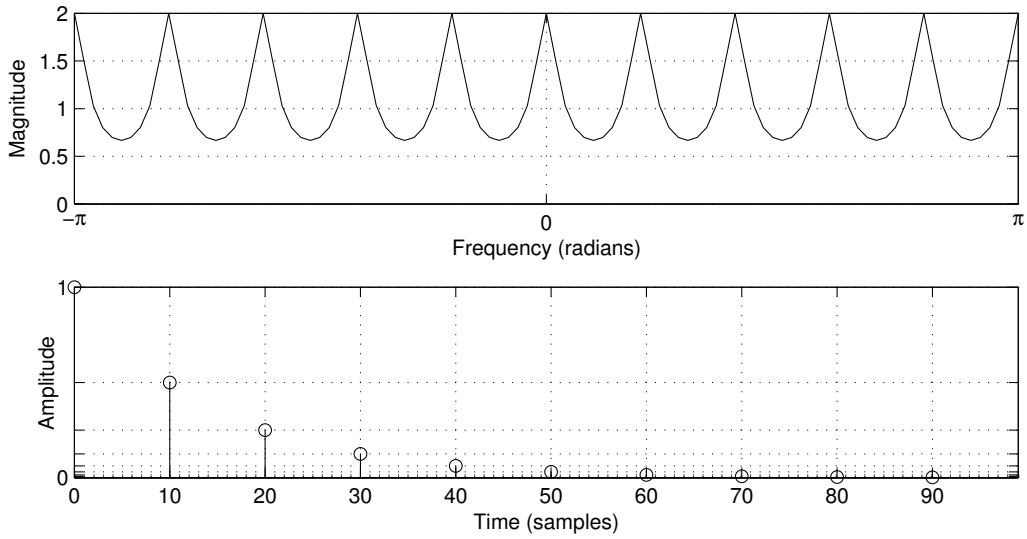

Figure 4.4: *Magnitude (top) and impulse (bottom) responses of tenth-order comb filter with feedback coefficient 0.5.*

if $a > 1$, then it will blow out of proportions. The impulse response of this filter, with coefficient a set to 0.5 and Q to 10, is shown in the lower plot of Figure 4.4. The transfer function of this filter will be

$$H(\omega) = \frac{1}{1 - ae^{-jQ\omega}}. \tag{4.25}$$

and its magnitude is plotted in the upper plot of Figure 4.4. You see that within the interval $[-\pi, \pi]$, the magnitude peaks, or *resonances*, or *poles*, count to 10, which, non-coincidentally, is the order Q of the filter. Similarly to what we saw previously with feedback filters, Q^{th}-order feedback filters feature Q poles in their transfer function. For the anecdote, this type of simple feedback filter is called a *comb filter*, because of its shape of upside-down comb. These peaks at regular frequency intervals should remind you of the harmonics in the spectra of pitched signals. In fact, this type of filter will resonate at a fundamental frequency of $2\pi/Q$. For demonstration, I set a filter order Q such that $f_s/Q = 400Hz$, the coefficient a to 0.99, so as to make the resonance nice and long, and sent as input to the filter a "noise burst" (play

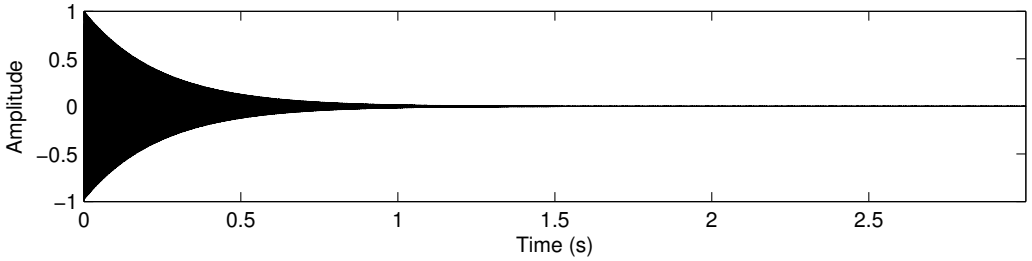

Figure 4.5: *Output to comb filter where* $f_s/Q = 400Hz$ *and* $a = 0.99$*, fed with a burst of* Q *random numbers in the interval* $[-1, 1]$*. The input burst can be played here, and the output, here.*

from here [1]), a sequence of Q random numbers. What comes out is a pitched waveform with exponential decay, which is reminiscent of tones produced by plucked and hit string instruments. The waveform is shown in Figure 4.5, and can be downloaded for playback here [2]. This may sound a little bit synthetic to you still, but Karplus and Strong [Karplus and Strong, 1983], and soon after Julius Smith [Jaffe and Smith, 1983], came up with simple additions which could make this filter sound *very much* like plucked strings.

The theoretical requisites on Linear Time-Invariant filters have been covered for us to approach Linear Prediction. On more short introduction before we tackle these proper, though, on *formants*, at the heart of the motivation for linear prediction.

4.3 Formants in Speech

We have seen already in our chapter on phonetics that the vocal apparatus consists of two potential excitation sources – the vocal cords for voiced sounds, and constrictions formed someplace along the tract – and two resonant cavities : the vocal cavity and the nasal cavity. The resonances in the vocal apparatus are responsible for the presence of *formants* in spectra. Figure 4.6 testifies of this phenomenon. : in the analysis of the vowel /u/ found in "zoo", regions where harmonics are emphasized are clearly found around

[1] `http://www.cs.nuim.ie/~matthewh/CS425/2012/sf/burst.wav`
[2] `http://www.cs.nuim.ie/~matthewh/CS425/2012/sf/comb.wav`

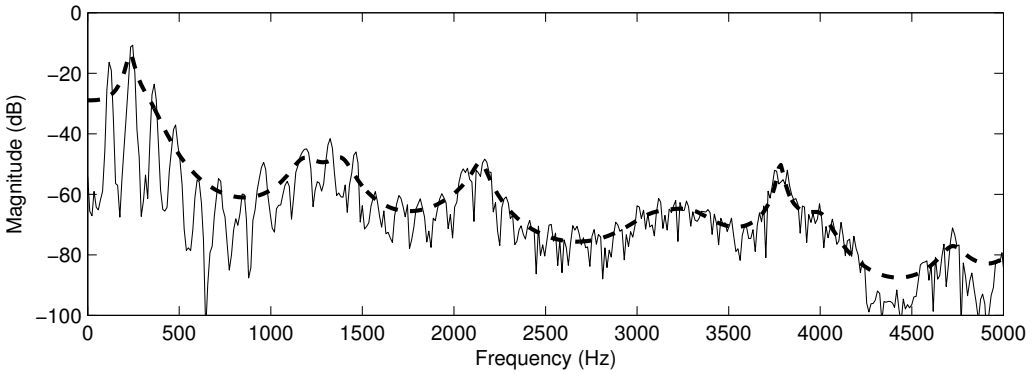

Figure 4.6: *Spectrum (solid line) of vowel phoneme* /u/ *in "z<u>oo</u>", and formantic envelope (dashed line). The envelope is the magnitude response of a Linear Prediction all-pole filter, here of order 250.*

frequencies 250, 1300, 2100, 3250 and 3800Hz. These regions correspond to the brighter horizontal stripes seen in the spectrograms of Section 1.3.

The timbre of vowels are essentially characterised by these formants, more precisely, their centre frequencies and *bandwidths*. The bandwidth is the width of a pole's *passband*, and the passband is the frequency interval over which the magnitude is at least the peak's magnitude lest 3 decibels. Figure 4.7 features the magnitude response of the famous *butterworth filter*. The center frequency there is 150Hz, the bandwidth, 100Hz, and so the passband spans the interval $[100, 200]$Hz.

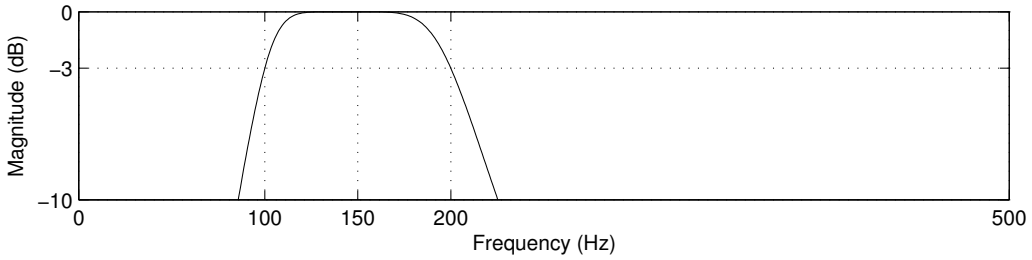

Figure 4.7: *Magnitude response of a* butterworth filter*, with center frequency 150Hz and bandwidth 100Hz.*

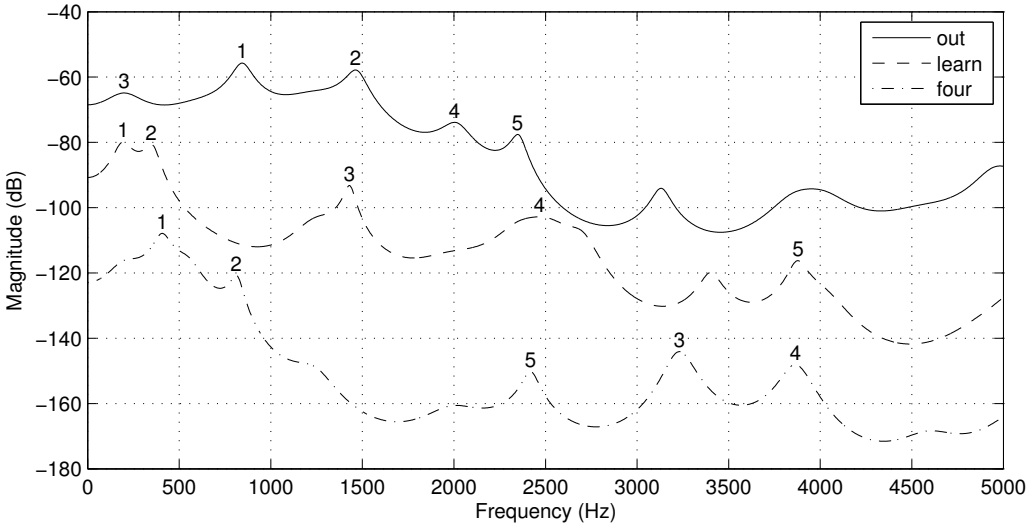

Figure 4.8: *Formants of four vowels :* /a/ *(solid),* /ɜ/ *(dashed) and* /o/ *(dash-dotted).*

Back to the business of the timbre of vowels, Figure 4.8 shows the formantic envelopes of three vowels : /a/ [3], /ɜ/ [4] and /o/ [5]. Table 4.1, in turn, shows a measure of the centre frequency and bandwidth of the five most prominent resonances found in these formantic envelopes. Such measurements are useful to the recognition and synthesis of vowels. There are several ways of measuring the formants of vowels, but the most popular approach is that of *Linear Prediction* : it is relatively simple, and it relies on a model fairly faithful to reality, which gives the method a strong potential in terms of applications. All the formantic contours seen in the figures around are derived using linear prediction, and as you can see from Figure 4.6, it is very faithful. It is this method that we are going to study now.

[3] `http://www.cs.nuim.ie/~matthewh/CS425/2012/sf/formants_out.wav`
[4] `http://www.cs.nuim.ie/~matthewh/CS425/2012/sf/formants_learn.wav`
[5] `http://www.cs.nuim.ie/~matthewh/CS425/2012/sf/formants_four.wav`

	/a/	/ɜ/	/o/
1	840 | 100	190 | 230	410 | 100
2	1,460 | 120	346 | 240	810 | 80
3	200 | 280	1,430 | 60	3,230 | 110
4	2,000 | 220	2,470 | 340	3,860 | 120
5	2,3470 | 90	3,880 |100	2,420 | 80

Table 4.1: Center frequency and bandwidth ($\pm 5Hz$) of 5 most prominent resonances found in formantic envelopes shown in Figure 4.8.

4.4 The theory of Linear Prediction

At the basis of linear prediction, there is the model according to which the vocal apparatus is a resonator excited primarily by a glottal impulse train. An impulse train is a periodic waveform, zero everywhere but every multiple of its period, T, i.e.

$$u[n] = \begin{cases} 1, & n = kT \\ 0, & n \neq kT \end{cases} \quad \forall k \in \mathbb{Z}. \tag{4.26}$$

Figure 4.9 shows a pulse train of period $T = 10$ (upper plot), and in anticipation of the coming argument, its spectrum (lower plot).

The theory of the impulse-fed resonator is supported by the observation that the spectrum of vowels, such as that seen in Figure 4.6, could be approximated with the product of the spectrum of an impulse train (Figure 4.9) and an all-pole filter (figures 4.6 and 4.8). We therefore create a synthetic speech spectrum S as the product of the spectrum of an impulse train, U, and the transfer function of an all-pole, resonant filter, R :

$$S(\omega) = R(\omega)U(\omega). \tag{4.27}$$

Our resonantor R being an all-pole filter, we express its transfer function, as per what we've learnt in Section 4.1, as

$$R(\omega) = \frac{b_0}{1 + \sum_{k=1}^{Q} a_k e^{-jk\omega}}. \tag{4.28}$$

The synthetic speech spectrum is obtained by multiplying the spectrum of the impulse train by that of the resonator, which is equivalent, in the time

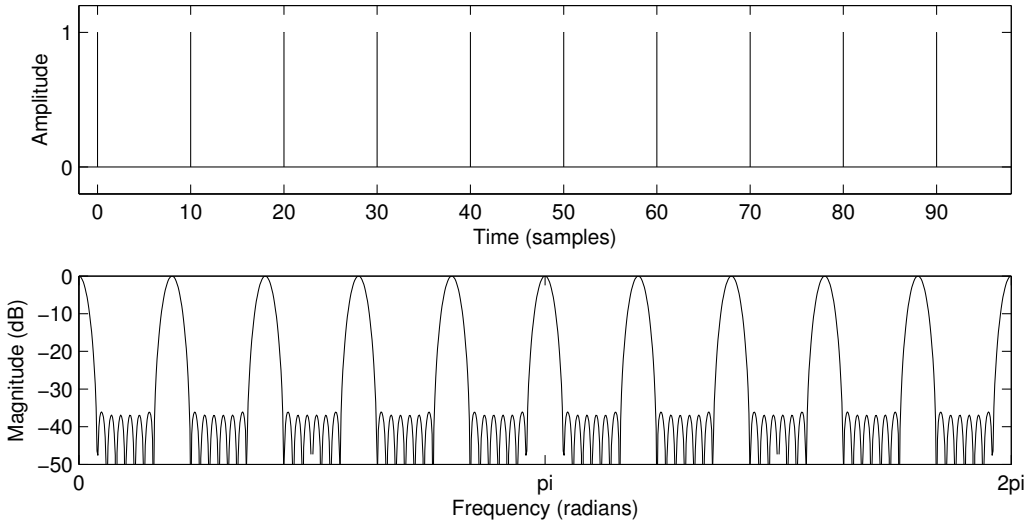

Figure 4.9: *An impulse train of period $T = 10$ (top) and its spectrum (bottom).*

domain, to feeding the digital filter corresponding to R with the pulse train, i.e.

$$s[n] = b_0 u[n] - \sum_{k=1}^{Q} a_k s[n-k]. \tag{4.29}$$

If our actual speech signal x abides by our model of an impulse train feeding a resonator, and if the coefficients a_k and filter order Q are found in an appropriate manner, then the signal s is going to approximate well x. Then we can express x in terms of its past samples, as per (4.29) : [6]

$$\tilde{x}[n] = -\sum_{k=1}^{Q} a_k x[n-k], \tag{4.30}$$

where $\tilde{x}[n]$, may be called the *prediction* of $x[n]$, an approximation of its actual value by the *linear* combination of its past samples – hence the term *linear prediction*. The prediction error $e[n]$ is the difference between the

[6]Here we remove $b_0 u[n]$ from the expression, which it will soon be seen how it is beneficial. Anyway, $u[n]$, as an impulse train, is 0 for most samples.

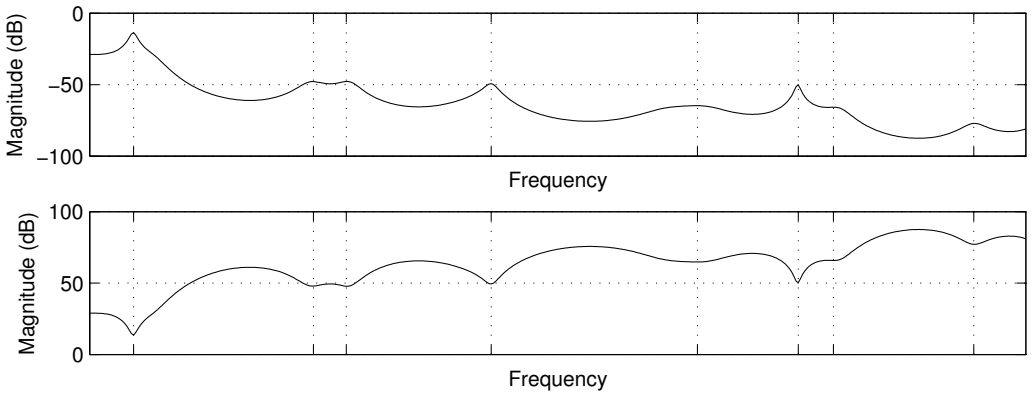

Figure 4.10: *Top : Response of formantic filter already seen in Figure 4.6 ; bottom : Response of the inverse of the filter.*

prediction $\tilde{x}[n]$ and the actual value $x[n]$, i.e.

$$\begin{aligned} e[n] &= x[n] - \tilde{x}[n] \\ &= x[n] + \sum_{k=1}^{Q} a_k x[n-k]. \end{aligned} \tag{4.31}$$

This signal, $e[n]$, may also be called the *residual*, which contains the noisy part of the signal, also called the *stochastic part*, or *non-deterministic part*. Indeed, you should realise that e is the result of sending the speech signal x through a *feedforward* filter, whose transfer function,

$$G(\omega) = 1 + \sum_{k=1}^{Q} a_k e^{-jk\omega}, \tag{4.32}$$

is the *exact inverse* of the transfer function of the resonator lest the scaling by b_0, i.e. $G(\omega) = b_0 R^{-1}(\omega)$. This is an *anti-resonator*, featuring magnitude dips wherever the resonator featured peak. Sticking with the example of the vowel whose spectrum is seen in Figure 4.6, we show in Figure 4.10 the magnitude responses of the original resonator $R(\omega)$, and that of the inverse filter $G(\omega)$. Wherever there were resonances in the original spectrum, the inverse filter features dips. The conglomerates of frequency components making up the formants (c.f. the white stripes in the spectrograms of Section 1.3) are

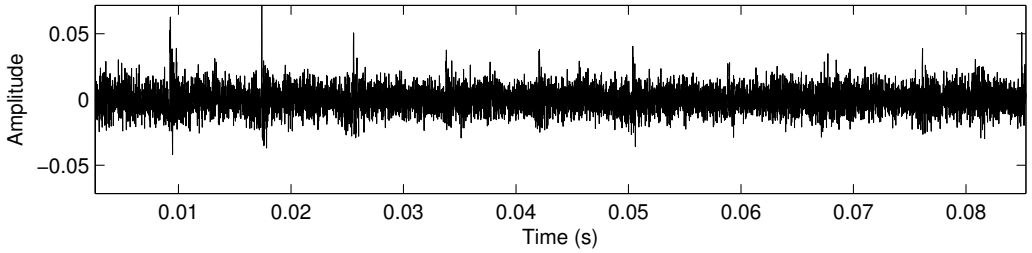

Figure 4.11: *Residual waveform, obtained from the filtering of the speech signal with the inverse filter $G(\omega)$.*

therefore going to be greatly attenuated, and the regions of the spectrum where only noise is found will be spared. In fact, the glottal excitation is also going to be spared : to see how, let us suppose that the speech signal x is completely faithful to the synthetic signal s, that is, $x[n] = s[n]$. Then we can write that

$$\begin{aligned} e[n] &= s[n] + \sum_{k=1}^{Q} a_k s[n-k], \\ &= b_0 u[n], \end{aligned}$$

by substitution of (4.29) into (4.31). In an ideal case, the residual $e[n]$ will only be an impulse train. In practice, it is also going to include the noisy part of the speech, not accounted for in the model. Figure 4.11 shows the residual obtained by the inverse filtering of the speech waveform seen in Figure 4.6 with the inverse filter seen in Figure 4.10. As per our statements, both the regular pulses and the noise of the waveform can be seen there.

All these demonstrations – these waveforms and figures – were possible because I know already how, for a given speech waveform, the coefficients can be found in an optimal sense. In the next section I explain how.

4.5 Finding the Linear Prediction coefficients

There exists several ways of finding the coefficients of a linear prediction filter. We are going to look at the autocorrelation method, for the reasons that it yields good results, that it is the most popular, and because we are already familiar with autocorrelation.

The starting point of the autocorrelation method is the intent to minimise the energy E of the error, i.e. [7]

$$E = \sum_n e^2[n] \tag{4.33}$$

$$= \sum_n \left(x[n] + \sum_{k=1}^{Q} a_k x[n-k] \right)^2 \tag{4.34}$$

Minimising E can be done by taking the partial derivatives of E in terms of each of the coefficients a_i, equating these to 0 and solving the resulting system of equations with linear algebra. We set this equality and start developing below :

$$\begin{aligned}
\frac{1}{2}\frac{\partial E}{\partial i} &= \sum_n \left(x[n] + \sum_{k=1}^{Q} a_k x[n-k] \right) x[n-i] \\
&= \sum_n \left(x[n-i]x[n] + x[n-i] \sum_{k=1}^{Q} a_k x[n-k] \right) \\
&= \sum_n x[n]x[n-i] + \sum_n x[n-i] \sum_{k=1}^{Q} a_k x[n-k] \\
&= \sum_n x[n]x[n-i] + \sum_{k=1}^{Q} a_k \sum_n x[n-k]x[n-i] \\
&= 0, \qquad i = 1, 2, ..., Q.
\end{aligned}$$

From the last two equalities we write

$$\sum_n x[n]x[n-i] = -\sum_{k=1}^{Q} a_k \sum_n x[n-k]x[n-i]. \tag{4.35}$$

Here is where autocorrelation kicks in : notice that

$$\sum_n x[n]x[n-i] = xx[i] \tag{4.36}$$

[7] We do not specify the limits of summation over n for convenience, but bear in mind that we are in short-time analysis, where the signal is windowed before being analysed, and hence non-zero only over a finite interval $[0, N[$. These would be the limits in (4.33), should we choose to mention them.

and that

$$\sum_n x[n-k]x[n-i] = \sum_n x[n-k+i]x[n] \tag{4.37}$$

$$= xx[i-k], \tag{4.38}$$

so Equation (4.35) can be re-written in terms of the autocorrelation values of x as

$$xx[i] = -\sum_{k=1}^{Q} a_k xx[i-k], \quad i = 1, 2, ..., Q. \tag{4.39}$$

Let us recall that our unknowns are the coefficients a_k which satisfies the system of equations in (4.39). We have Q such coefficients, so Q unknowns, and Q such equations (with i going from 1 to Q). This system of equations can be written in terms of vectors and a square matrix,

$$\begin{bmatrix} xx[1] \\ xx[2] \\ \vdots \\ xx[Q] \end{bmatrix} = \begin{bmatrix} xx[0] & xx[1] & \cdots & xx[Q-1] \\ xx[1] & xx[0] & \cdots & xx[Q-2] \\ \vdots & \vdots & \ddots & \vdots \\ xx[Q-1] & xx[Q-2] & \cdots & xx[0] \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_Q \end{bmatrix}, \tag{4.40}$$

or

$$\mathbf{xx} = -X\mathbf{a}, \tag{4.41}$$

where

$$\mathbf{xx} = \begin{bmatrix} xx[1],\ xx[2],\ ...,\ xx[Q] \end{bmatrix}^T, \tag{4.42}$$

$$\mathbf{a} = \begin{bmatrix} a_1,\ a_2,\ ...,\ a_Q \end{bmatrix}^T, \tag{4.43}$$

and

$$X_{i,k} = xx[i-k]. \tag{4.44}$$

The coefficients are then simply found as

$$\mathbf{a} = -X^{-1}\mathbf{xx}. \tag{4.45}$$

Before we are finished, there remains to determine the scalar b_0 seen in Equation (4.29). It can be shown that this is the square root of E [Quatieri, 2002],

$$b_0 = \sqrt{E}. \tag{4.46}$$

The computation of the inverse of the matrix X is an operation which, with the standard procedure of Gauss-Jordan elimination, is done in $O(Q^3)$ operations. In the linear prediction examples presented in this section, the order of the filter was set to 250, [8] (for a sampling rate of 96kHz,) implying over 15 millions of operations. The Levinson algorithm exploits the fact that the autocorrelation matrix X is symmetrical about its diagonal – it is thereby a *Toepliz matrix* – and is known to reduce this computation to $O(N^2)$, which is much more efficient already. The computation of these examples was done in matlab, where (4.45) was computed with the "left-matrix-divide" operator (i.e. `a=-X\xx`), and it seemed near-instantaneous.

4.6 Having fun with linear prediction

Now that we have been through the tedious theory is time to contemplate what can be done with such filters. Analysis of formants is possible, and this has already been shown in Section 4.3. Linear prediction is also very useful in *speech coding*, which allows the storage of speech signals at a lower memory cost, facilitating their transmissions throughout networks [Quatieri, 2002]. Speaker recognition can also benefit from linear prediction, as formants are specific not only of the vowel being pronounced, but also on the person pronouncing it. More generally, the technique is so satisfactory that there must be dozens of applications I have never thought or heard of before existing out there.

One I can easily imagine is the re-synthesis of speech signals. A given speech signal can be short-time analysed, and for each window the resonant filter coefficients derived. The five most prominent (or the five lowest) poles can easily be obtained, as shown in Figure 4.8 and Table 4.1. (These are really those which characterise the timbre of vowels as we experience it.) Then an impulse train may be generated, following the pitch of the analysed sound using our technique of Section 3.6, and fed through a system of five one-pole filters set in parallel, whose poles glide, from window to window, tracking the center frequencies and bandwidths measured by linear prediction. (The filters will then have to be time-variant.) This would model the voiced part

[8] There are ways of finding automatically the order Q in an optimal sense [Quatieri, 2002], but we are not going to go through them here. A fixed order of 250 for a sampling rate of 96kHz yields the satisfactory results seen in the various figures and sound files anyway.

of the sound. As for the unvoiced part, it could probably be modeled with white noise, whose amplitude interpolates between the energy values of the residual, E.

This is a mere suggestion of a project. Here I have done something simpler : re-using the speech used in the various figures and audio examples in Section 3.6, I selected manually three vowels of contrasting character and ran linear prediction on each of them. Then I tracked the pitch of the entire dialogue as per the method seen in Section 3.6, and generated an impulse train that follows this pitch. The result waveform is this impulse, filtered by each of these vowel-filters in turn. The vowels chosen for analysis were /n/ [9], /e/ [10] and /a/ [11]. The resulting waveform can be heard here [12]. This is fun, but most exciting is to *listen to the impulse responses of the vocal tract for each of these vowels!* The /n/ [13], the /e/ [14], and the /a/ [15]. Nasals are *extremely* resonant! This really sounds like a tube. The /a/, most open, is the least resonant. Linear prediction here gives us a privileged insight in the acoustics of our own, organic vocal tubes...

[9] `http://www.cs.nuim.ie/~matthewh/CS425/2012/sf/letsplay_n.wav`
[10] `http://www.cs.nuim.ie/~matthewh/CS425/2012/sf/letsplay_e.wav`
[11] `http://www.cs.nuim.ie/~matthewh/CS425/2012/sf/letsplay_a.wav`
[12] `http://www.cs.nuim.ie/~matthewh/CS425/2012/sf/letsplay.wav`
[13] `http://www.cs.nuim.ie/~matthewh/CS425/2012/sf/letsplay_n_IR.wav`
[14] `http://www.cs.nuim.ie/~matthewh/CS425/2012/sf/letsplay_e_IR.wav`
[15] `http://www.cs.nuim.ie/~matthewh/CS425/2012/sf/letsplay_a_IR.wav`

Chapter 5

The Fast Fourier Transform

The Fast Fourier Transform (FFT) is an incredibly useful transform, that allows the expression of digital signals as the sum of complex exponentials. Then visualisation of the magnitude of the spectrum allows to get an impression of the frequency distribution of the sound. For example, the noisy spectrum of a cymbal would be very strong at high frequency indices, while the engine of a Harley-Davidson would feature a concentration of sinusoidal energy in the lower end of the spectrum. More importantly, the frequency domain is often very propitious to intuitive processing, especially where filtering is concerned. To cut the high-frequency components of a signal is only a matter of setting to zero all the bins of the spectrum above the chosen cutoff frequency, and then inverse-transforming the signal to get back to the time domain. But this is only one in many, many, many applications.

The Fourier transform and Fourier series, developed by Baron Joseph Fourier in the 18$^{\text{th}}$ century, [1] were here first. They are the continuous-domain equivalent of the Discrete-Time Fourier Transform (DTFT) and the Discrete Fourier Transform (DFT), which, if memory serves well, were developed with the advent of computers. Of these four transforms – which are the same idea in slightly different contexts – we are going to see the theory of the DFT. The FFT, developed in the 70s, [2] yields the exact same result as the DFT, except that it is computed with a divide-and-conquer algorithm and makes its computation much faster. We are going to see this in more detail when the time is right. But first let us look at the DFT and its inverse transform, the Inverse DFT (IDFT).

5.1 The forward DFT

We find useful to introduce the idea of the DFT with a little bit of linear algebra. Consider the 2-dimensional vector $\vec{z} = [z_1, z_2]^T$, in a 2-dimensional vector space (a plane) with its 2 basis vectors $\vec{i} = [1, 0]^T$ and $\vec{j} = [0, 1]^T$. The *projection* of $\vec{z}$ onto $\vec{i}$ will give us a scalar that tells us what "amount" of $\vec{i}$ there is in $\vec{z}$. This projection is obtained with the *dot product* of the vectors :

$$\vec{i} \cdot \vec{z} = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = z_1.$$

Likewise, $\vec{j} \cdot \vec{z}$ tells what amount of $\vec{j}$ there is in $\vec{z}$.

[1] to be double-checked

[2] to be double-checked as well

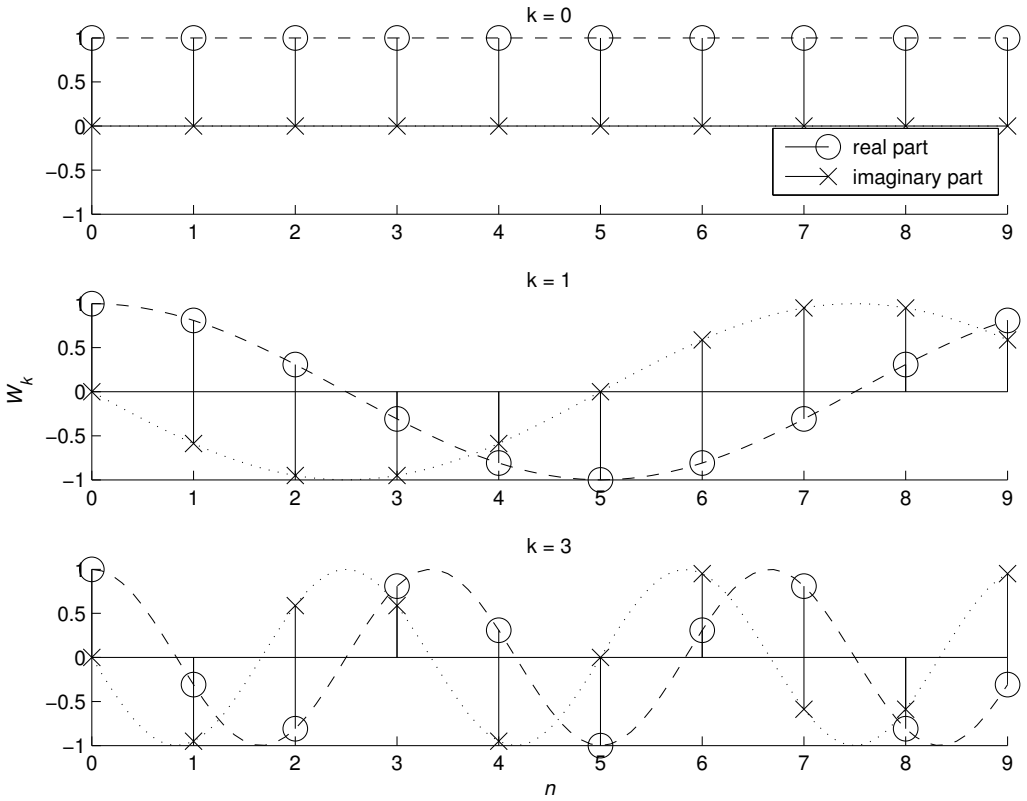

Figure 5.1: *Basis vector $\vec{W}_k = e^{-jk\vec{n}2\pi/N}$, plotted as a function of $\vec{n}$, for three values of k, and with N set to 10. The dashed and dotted lines give an idea of what the continuous, complex-exponential underlying model is for each.*

The principle of the DFT is analogous. Now we have an N-dimensional vector $\vec{x} = [x[0], x[1], ..., x[N-1]]^T$, in an N-dimensional space with its N basis vectors. The basis vectors, however, are now

$$\vec{W}_k = e^{-jk\vec{n}2\pi/N}, \quad k = 0, 1, ..., N-1,$$

where $\vec{n} = [0, 1, ..., N-1]^T$. Basis vectors for $N = 10$ and $k = 0, 1, 3$ are given in Figure 5.1. What you should notice here is they are sampled sinusoids ; it is the underlying idea of the Fourier theorem that any periodic function can be expressed as a sum of such sinusoids. Similarly to our 2-dimensional example above, the projection $\vec{W}_k \cdot \vec{x}$ of $\vec{x}$ onto the basis function $\vec{W}_k$ yields a

complex scalar $X[k]$, telling us that in the sound segment $\vec{x}$ there is a complex sinusoid of frequency $k2\pi/N$, of magnitude $|X[k]|$, and of phase $\angle X[k]$. This dot product can be written in vector notation,

$$X[k] = \left[\begin{array}{ccccc} 1 & e^{-jk2\pi/N} & e^{-jk4\pi/N} & ... & e^{-jk(N-1)2\pi/N} \end{array} \right] \left[\begin{array}{c} x[0] \\ x[1] \\ x[2] \\ .. \\ x[N-1] \end{array} \right], \quad (5.1)$$

or it can be written in its usual form,

$$\boxed{X[k] = \sum_{n=0}^{N-1} x[n] e^{-jkn2\pi/N}, \quad k = 0, 1, ..., N-1}. \quad (5.2)$$

Equation (5.2) is the most famous expression of the DFT.

Thanks to our vector-based introduction of the DFT, it is going to be easy to introduces the *inverse* DFT.

5.2 The inverse DFT

In Equation (5.1), instead of having the vector $\vec{W}_k$ multiplying $\vec{x}$ to produce only one entry of the vector $\vec{X} = [X[0], X[1], ..., X[N-1]]$, we could arrange all the vectors $\vec{W}_k$ for $k = [0, 1, ..., N-1]$ in a matrix

$$W = \left[\begin{array}{c} \vec{W}_0^T \\ \vec{W}_1^T \\ \vec{W}_2^T \\ ... \\ \vec{W}_{N-1}^T \end{array} \right]$$

such that

$$W_{k,n} = e^{-jkn2\pi/N},$$

and whose product with the vector $\vec{x}$ would yield the entire spectrum vector $\vec{X}$, i.e.

$$\vec{X} = W\vec{x}. \quad (5.3)$$

From Equation (5.3), it is easy to work out that

$$\vec{x} = W^{-1}\vec{X}, \tag{5.4}$$

and it turns out that

$$\left(W^{-1}\right)_{k,n} = \frac{1}{N}e^{jkn2\pi/N}, \tag{5.5}$$

which is saying that the inverse of the DFT matrix W is the exact same, except scaled and with negated power. On this basis we can re-write Equation (5.4) in summation form to yield the standard inverse-DFT (IDFT) expression

$$\boxed{x[n] = \frac{1}{N}\sum_{k=0}^{N-1} X[k]e^{jkn2\pi/N}, n = 0, 1, ..., N-1}\,. \tag{5.6}$$

5.3 Proof of the transparency of the DFT

Our development above is nearly sufficient as a proof that the IDFT of the DFT of a signal yields the original signal, and hence that the DFT can be said to be *transparent*. However, we have not demonstrated how the equality (5.5) was obtained. (The truth is, I checked it numerically in Matlab.) Instead we are going to use equations (5.2) and (5.6).

$$\begin{aligned} x[n] &= \frac{1}{N}\sum_{k=0}^{N-1} X[k]e^{jkn2\pi/N} \\ &= \frac{1}{N}\sum_{k=0}^{N-1}\left(\sum_{m=0}^{N-1} x[m]e^{-jkm2\pi/N}\right)e^{jkn2\pi/N}\ ; \end{aligned}$$

make sure, in this substitution, that you change the index of summation of the inner transform from n to something else (here, m).

The order of the summation can be changed without altering the result. The trick is to bring the summation with respect to k inside, which will allow

us to bring $x[m]$ outside of the summation with respect to k :

$$\begin{aligned} x[n] &= \frac{1}{N}\sum_{m=0}^{N-1}\sum_{k=0}^{N-1} x[m]e^{-jkm2\pi/N}e^{jkn2\pi/N} \\ &= \frac{1}{N}\sum_{m=0}^{N-1} x[m]\sum_{k=0}^{N-1} e^{jk(n-m)2\pi/N}. \end{aligned} \tag{5.7}$$

It is well known and can easily be verified that

$$\begin{aligned} \sum_{k=0}^{N-1} e^{jk(n-m)2\pi/N} &= \begin{cases} N, & n = m \\ 0, & n \neq m \end{cases} \\ &= N\delta[n-m]. \end{aligned}$$

We substitute this result in (5.7), and get

$$\begin{aligned} x[n] &= \sum_{m=0}^{N-1} x[m]\delta[n-m] \\ &= x[n], \end{aligned}$$

which is true.

5.4 Periodicity of the DFT and IDFT

The DFT is periodic in N, i.e.

$$X[k] = X[k+pN], \quad p \in \mathbb{Z}. \tag{5.8}$$

This can easily be verified the same way as we verified that the transfer function of a digital filter was periodic in 2π. In Equation (5.2), the only term involving k is $e^{-jkn2\pi/N}$, and so verifying (5.8) reduces to verifying that $e^{-jkn2\pi/N} = e^{-j(k+pN)n2\pi/N}$, which it is, as

$$e^{-j(k+pN)n2\pi/N} = e^{-jkn2\pi/N}e^{-jpn2\pi}$$

and $e^{-jpn2\pi} = 1$, provided that both p and n are integers.

The same goes for the inverse DFT,

$$x[n] = x[n+pN], \quad p \in \mathbb{Z}, \tag{5.9}$$

so long as $x[n]$ was produced by the inverse DFT of a spectrum, $x[n] = \mathrm{DFT}^{-1}\{X[k]\}$.

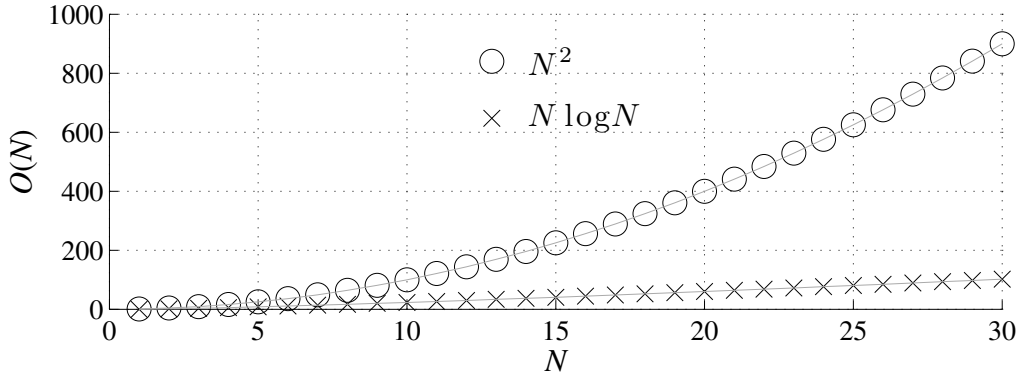

Figure 5.2: *DFT (circles) and FFT (crosses) computational complexity functions. In comparison with the quadratic shape of the DFT, the FFT's seems almost linear.*

5.5 Computational Complexity

The DFT equation, (5.2), shows that, for the computation of any given value $X[k]$, the operation $x[n]e^{-jkn2\pi/N}$ has to be calculated for N values of n. As there are N values of k as well, the computation of the entire vector $\vec{X}$ requires the calculation of $x[n]e^{-jkn2\pi/N}$ N^2 times, implying the computational complexity $O(N^2)$.

The Fast Fourier Transform (FFT) is a divide-and-conquer algorithm [Steiglitz, 1996] which reduces the computational complexity to $O(N \log N)$, which, for high values of N, can become an enormous gain. Figure 5.2 gives an idea of the savings that can be made.

You should bear in mind, however, that the operations yield the exact same results, i.e. DFT$\{x[n]\}$ = FFT$\{x[n]\}$. The exact same approach can be taken for the IDFT, and the same relation between thee IDFT and IFFT can therefore be given.

5.6 The convolution and correlation theorem

The convolution theorem is a famous theorem which states that the spectrum of the product of two functions equals the convolution of the spectrum of each

function, i.e.

$$\mathrm{FFT}\{x \cdot y\} = \mathrm{FFT}\{x\} * \mathrm{FFT}\{y\}.$$

Conversely, the spectrum of the convolution of two functions equals the product of the spectrum of each function, i.e.

$$\mathrm{FFT}\{x * y\} = \mathrm{FFT}\{x\} \cdot \mathrm{FFT}\{y\}.$$

The convolution of two signals can therefore be obtained by the inverse FFT of the product of their spectra,

$$x * y = \mathrm{FFT}^{-1}\{\mathrm{FFT}\{x\} \cdot \mathrm{FFT}\{y\}\}. \tag{5.10}$$

In terms of computation cost, this is very advantageous. In Equation (5.10), the FFT is performed three times, which traduces to a computational complexity of $O(3N \log N) = O(N \log N)$, while convolution normally is, like autocorrelation, an operation of complexity $O(N^2)$.

Similarly, it can be said of the correlation of signals x and y that

$$\mathrm{FFT}\{x \star y\} = \mathrm{FFT}^{*}\{x\} \cdot \mathrm{FFT}\{y\},$$

* denoting complex conjugation – bear in mind that our FFT spectra are complex. Inverse-FFT of each side yields

$$x \star y = \mathrm{FFT}^{-1}\{\mathrm{FFT}^{*}\{x\} \cdot \mathrm{FFT}\{y\}\}. \tag{5.11}$$

Proof of both the convolution and correlation theorems can be obtained easily, provided that the steps are known and well respected. We are here giving proof of the correlation theorem, as we have been needing autocorrelation repeatedly throughout this module. Our starting point is Equation

(5.11).

$$\begin{aligned}
(x \star y)[n] &= \mathrm{FFT}^{-1}\{\mathrm{FFT}^{*}\{x\} \cdot \mathrm{FFT}\{y\}\} \\
&= \frac{1}{N} \sum_{k} e^{jkn2\pi/N} \left(\sum_{m} x[m] e^{jkm2\pi/N} \sum_{p} y[p] e^{-jkp2\pi/m} \right) \\
&= \frac{1}{N} \sum_{k} \sum_{m} \sum_{p} x(m) y(p) e^{jk(m+n-p)2\pi/N} \\
&= \frac{1}{N} \sum_{m} x[m] \sum_{p} y[p] \sum_{k} e^{jk(m+n-p)2\pi/N} \\
&= \sum_{m} x[m] \sum_{p} y[p] \delta[m+n-p] \\
&= \sum_{m} x[m] y[m+n],
\end{aligned}$$

the last line of which we recognise, indeed, to be the correlation of signals x and y. Where convolution is concerned, the very same approach can be taken with starting point Equation (5.10) instead of (5.11), and should yield $\sum_m x[m]y[n-m]$. As for autocorrelation, it can be found that

$$\begin{aligned}
xx[n] &= \mathrm{FFT}^{-1}\{\mathrm{FFT}^{*}\{x\} \cdot \mathrm{FFT}\{x\}\} \\
&= \mathrm{FFT}^{-1}\left\{ |\mathrm{FFT}\{x\}|^{2} \right\}.
\end{aligned}$$

This holds because the product of a complex number with its complex conjugate is the magnitude squared of that number, i.e. $(a+jb)(a-jb) = a^2+b^2 = |a+jb|^2$. To get the autocorrelation of a signal is therefore simply a matter of taking the inverse FFT of the squared magnitude of the FFT of this signal, which is very conveniently done in such environment as Matlab.

Conclusion

This concludes our module on Audio and Speech Processing. To begin with, we familiarised ourselves with the workings of our vocal apparatus, the concepts of phonology and phonetics key to our module, and the time- and frequency-domain characteristics of the English phonemes.

We thereafter tackled time-domain Voice-Unvoiced-Silence (VUS) discrimination, where the method that was chosen for its relative simplicity. A frequency-domain approach at that time was excluded because of the prerequisites the frequency domain comes with. Frequency-domain characteristics were not disregarded inasmuch (e.g. Low-band/Full-band energy ratio) so long as they could be measured in the time domain.

We then moved on to time-domain pitch estimation, using a short-time autocorrelation method. The difficulties inherent to this method were identified and addressed. At that stage it was shown that the VUS discrimination of the previous chapter could be used to locate the time segments where pitch estimation made sense – the voiced segments. Also, an original, "heavy-track" algorithm was formulated to overcome the difficulty of abrupt jumps in the pitch estimate.

Chapter 2 addressed VUS discrimination, and Chapter 3, pitch estimation. Besides pitch, an essential characteristic of speech is its formants, which make the difference between vowels. We studied the widespread technique of Linear Prediction (LP) with the aim of getting the formantic envelope of input vowels, with applications in speech recognition, sound synthesis and acoustics. LP, although usually performed through time-domain filtering, was brought to the student from a frequency-domain point of view, for the reason that it was deemed more intuitive.

In the final chapter of the module, the Fast Fourier Transform (FFT) was finally approached. At that stage the student had already developed a strong familiarity with frequency-domain representations, and had been

introduced to complex exponentials, the building blocks for the DFT theory. It was shown how the FFT could be used to reduce substantially the cost of correlation, which we had been using repeatedly throughout the module, in the context of pitch estimation and Linear Prediction.

Appendix A

Algorithmic Basis for Autocorrelation-based "heavy-track" pitch tracking

Here follows a detailed description of the processes involved in the tracing of the contours as demonstrated in Section 3.9. Intuitively, the algorithm is fairly straightforward, but the implementation was found to be much more tedious, even in a scripting language such as Matlab. The intent of this appendix is therefore to keep a record of the result of hours of algorithm-writing and optimisation. It is intended to be written with mathematical rigour, independently of any specific programming language. Its Matlab implementation will nevertheless be made available by the author, but it is hoped that this appendix is found valuable for the encoding in other programming languages, such as C++.

In the body text of this document, the algorithm is described as dealing with an autocorrelation matrix $\mathbf{A}$ of M rows and N columns, as a result of which the tracks and contours came to being sets of tuples, e.g. $T = \{(1,1),(1,2),(2,3)\}$. It was found nevertheless more convenient to transform the matrix $\mathbf{A}$ in a vector $\mathbf{a}$, whose k^{th} entry was the entry of row $i = \text{mod}(k-1, M)+1$ and column $j = \lfloor (k-1)/M \rfloor + 1$ of the original matrix. [1] Among the several advantages of this representation, the most appreciable is probably the memory alleviation, reducing matrix indexation from tuples to single

[1] This is the way matrices are actually stored in Matlab. See "Linear Indexing" section in Matlab's documentation.

integers and the memory load of track and contour sets by a factor of two. Keeping tracks and contours as vectors is also more convenient than doing so as sets of tuples.

A.1 closest()

The closest() function is the lowest-level routine of the algorithm. It allows finding the closest non-zero entry of a vector from a given entry index in a strict sense.

$$j = \texttt{closest}(\mathbf{a}, i, M) \tag{A.1}$$

Given a vector $\mathbf{a}$ of I elements and an index i, return index j such that $\mathbf{a}_j \neq 0$ and for which $|i - j|$ is strictly minimal, i.e. there does not exist an index $k \neq j$ such that $\mathbf{a}_k \neq 0$ and $|i - k| = |i - j|$. The algorithm is as follows.

1. If $i \notin [1, I]$, $j \leftarrow 0$, stop.
2. If $\mathbf{a}_i \neq 0$, $j \leftarrow i$, stop.
3. $i_{\min} \leftarrow \lfloor (i-1)/M \rfloor M + 1$ and $i_{\max} \leftarrow i_{\min} + M - 1$
   $j \leftarrow 0$ and $s \leftarrow 1$
   $l \leftarrow$ `false` and $r \leftarrow$ `false`.
4. While $i - s \geq i_{\min}$ or $i + s \leq i_{\max}$, repeat the following steps.
5. If $i - s \geq i_{\min}$ and $\mathbf{a}_{i-s} \neq 0$, $l \leftarrow$ `true`.
6. If $i + s \leq i_{\max}$ and $\mathbf{a}_{i+s} \neq 0$, $r \leftarrow$ `true`.
7. If l and r are `true`, stop.
8. If l is `true`, $j \leftarrow i - s$, stop ; If r is true, $j \leftarrow i + s$, stop.
9. $l \leftarrow$ `false` and $r \leftarrow$ `false`.
10. Increment s.

A.2 closiproques()

The function closiproques() returns a vector the same size as **a** from which tracks where peaks are closest one to another in a strict and reciprocal sense can be traced.

$$\mathbf{p} = \text{closiproques}(\mathbf{a}, M) \tag{A.2}$$

For a vector **a** and an integer M, returns a vector **p** of dimensions equal to **a**'s, whose i^{th} element p_i is p if $a_i \neq 0$, $p = \text{closest}(\mathbf{a}, i + M)$ and $i = \text{closest}(\mathbf{a}, p - M)$. Otherwise, p_i is 0.

1. Create a zero vector **p** of dimensions equal to **a**'s.
2. For all values of i for which $a_i \neq 0$, repeat the following.
3. $j \leftarrow \text{closest}(\mathbf{a}, i + M, M)$
4. If $i = \text{closest}(\mathbf{a}, j - M, M)$, $p_i \leftarrow j$.

A.3 tracks()

tracks() orders the tracks inherent in input vector **p** in the set T.

$$T = \text{tracks}(\mathbf{p}) \tag{A.3}$$

For a vector $\mathbf{p} = \text{closiproques}(\mathbf{a}, N)$, returns T, a set of vectors **t**.

1. Assign the index of the first non-zero entry in **p** to i.
2. While $i \neq \emptyset$, repeat the following.
3. $t_1 \leftarrow i$, $\mathbf{t} \leftarrow t_1$, $j \leftarrow 1$
4. While $p_{t_j} \neq 0$, repeat steps 5 to 7.
5. Concatenate p_{t_j} to the tail of **t**.
6. $p_{t_j} \leftarrow 0$
7. Increment j.
8. Make **t** the last element of set T.
9. Assign the index of the first non-zero entry in **p** to i.

A.4 Summary

A contours() function may be created from the algorithmic description in Section 3.9 of the body text. This function takes the sparse matrix **A**, whose non-zero entries are the peak values of the autocorrelation plane (excluding the 0-lag peaks). Thresholding the such as that detailed in Section 3.8 is desirable. If a preliminary Voiced-Unvoiced-Silence discrimination is not available, the autocorrelation-value thresholding should compensate. The function contours() calls tracks(), which calls closiproques(), which calls closest(). The returned set of contours can be used for pitch-related analysis and processing.

Bibliography

[Crystal, 2009] Crystal, D. (2009). "Dictionary of Linguistics and Phonetics". Wiley, New Jersey, USA.

[Gold and Rabiner, 1969] Gold, B. and Rabiner, L. (1969). "Parallel Processing Techniques for Estimating Pitch Periods of Speech in the Time Domain". *The Journal of the Acoustical Society of America*, 46(2B), pp. 442–448.

[III and Serra, 1987] III, J. O. S. and Serra, X. (1987). "PARSHL: An Analysis/Synthesis Program for Non-Harmonic Sounds Based on a Sinusoidal Representation". In *Proceedings of the International Computer Music Conference (ICMC-87)*, Tokyo, Japan.

[Jaffe and Smith, 1983] Jaffe, D. A. and Smith, J. O. (1983). "Extensions of the Karplus-Strong Plucked-String Algorithm". *Computer Music Journal*, 7(2).

[Karplus and Strong, 1983] Karplus, K. and Strong, A. (1983). "Digital Synthesis of Plucked-String and Drum Timbres". *Computer Music Journal*, 7(2).

[Kondoz, 2004] Kondoz, A. M. (2004). *Digital Speech*. John Wiley & Sons, Ltd, second edition.

[McAulay and Quatieri, 1986] McAulay, R. and Quatieri, T. (1986). "Speech analysis/Synthesis based on a sinusoidal representation". *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 34(4), pp. 744 – 754.

[Moore, 2004] Moore, B. C. J. (2004). *An Introduction to the Psychology of Hearing*. Elsevier Academic Press, fourth edition.

[Pelton, 1993] Pelton, G. E. (1993). *Voice Processing*. McGraw-Hill, Inc.

[Plannerer, 2005] Plannerer, B. (2005). "An Introduction to Speech Recognition".

[Quatieri, 2002] Quatieri, T. F. (2002). *Discrete-Time Speech Signal Processing*. Prentice Hall PTR.

[Rabiner, 1989] Rabiner, L. R. (1989). "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition". *Proceedings of the IEEE*, 77(2).

[Rabiner and Schafer, 1978] Rabiner, L. R. and Schafer, R. W. (1978). "Digital Processing of Speech Signals". Prentice-Hall, New Jersey, USA.

[Serra and Smith, 1990] Serra, X. and Smith, Julius, I. (1990). "Spectral Modeling Synthesis: A Sound Analysis/Synthesis System Based on a Deterministic Plus Stochastic Decomposition". *Computer Music Journal*, 14(4), pp. pp. 12–24.

[Steiglitz, 1996] Steiglitz, K. (1996). *A Digital Signal Processing Primer*. Addison-Wesley Publishing Company, Inc., Menlo Park, California, USA.

[Wu and Childers, 1991] Wu, K. and Childers, D. G. (1991). "Gender recognition from speech. Part I: Coarse analysis". *The Journal of the Acoustical Society of America*, 90(4), pp. 1828–1840.