University of
Southern Denmark

# Basic Bioacoustics using Matlab

Compendium for BB838

Magnus Wahlberg and Felix Häfele

Department of Biology
University of Southern Denmark

Odense, January 2025

# Preface

About 90% of the information in these chapters is true. It is difficult not to “cut some corners“ (freely translated Danish expression) when trying to produce distilled mathematics and physics within a few pages. Also, even though it is the third version of this compendium, there could still be typos, thoughts-of-error, and misleading instructions and barely logical ways of reasoning.

With that said, we hope the texts will be useful to spur a general discussion on how we analyse acoustic signals that can be valuable for both the newcomers and the more experienced attendants.

The text is meant to be read in parallel with practicing programming through Matlab. There will also be practical demonstrations of calibrations, sound field measurements, and much more. Acoustics is best learned by combining theory with practice – so make sure you both work on practical as well as theoretical skills.

Enjoy!

Magnus and Felix

# Contents

# 1. Basic Tools

## 1.1 Waves

Sound is local pressure and particle motion fluctuations propagating through a medium. To understand sound, we therefore need some basic tools to describe and quantify waves.

Mathematically, the simplest wave is a sinusoid. It is described by the function

$$y(x) = A \sin\left(\frac{2\pi x}{\lambda} - \theta\right) \tag{1.1}$$

As the sinus function fluctuates between $-1$ and $+1$, the factor $A$ describes the amplitude of the wave at a certain $x$ value (a spatial coordinate). The number $\lambda$ is the wave length (in meters), and $\theta$ is the phase.

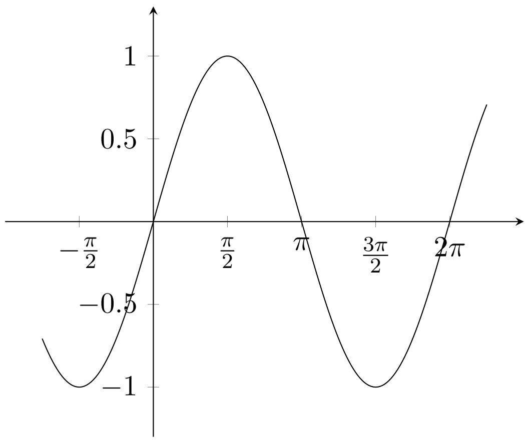


Figure 1.1: Sinusoid

Mathematicians are lazy – to avoid writing a lot of $2\pi$, you often see them abbreviate the above expression as

$$y(x) = A \sin(kx - \theta) \ , \tag{1.2}$$

where $k = \frac{2\pi}{\lambda}$ is called the wave number.

The equation above describes $y$ as a function of $x$, which here means a spatial coordinate (ask if you do not understand what this means!). In real 3D life, the wave is a function of three coordinates ($x$, $y$ and $z$). However, luckily, in most situations in bioacoustics we deal with plane waves for which the equation above is usually sufficient.

Instead of moving along the $x$-axis you can ask yourself what happens if you stand on a specific point and have the wave passing you. Then you can describe the wave as a function of time $t$:

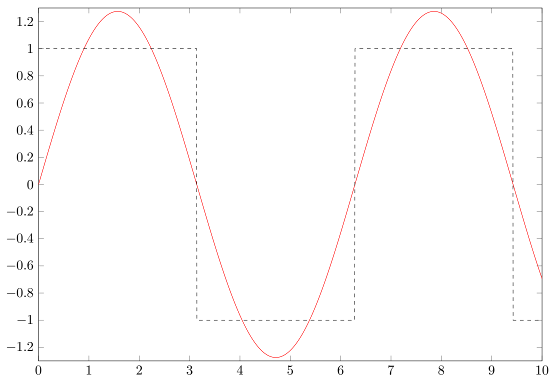
$$y(t) = A \sin\left(\frac{2\pi t}{T} - \varphi\right) \ . \tag{1.3}$$

Here $T$ is the period of the wave (the time in seconds it takes to come from one maximum to the next) and $\varphi$ is the phase angle. Again, to help the lazy mathematicians, we introduce $f = \frac{1}{T}$ and call it the frequency (the unit is cycles per second, periods per second, or Hertz [Hz]). We can also define the angular frequency $\omega = 2\pi f$, so that

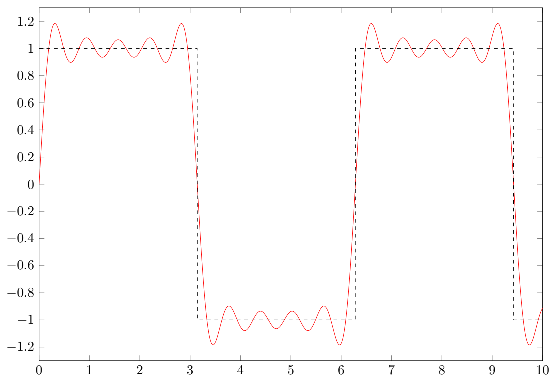
$$y(t) = A \sin(2\pi f t - \varphi) = A \sin(\omega t - \varphi) \ . \tag{1.4}$$

Actually, for acoustic waves the period $T$ and wavelength $\lambda$ are related by the speed of sound $c$ of the medium (around 340 $m/s$ in air and 1500 $m/s$ in water, depending on temperature and other factors, to be discussed later) by

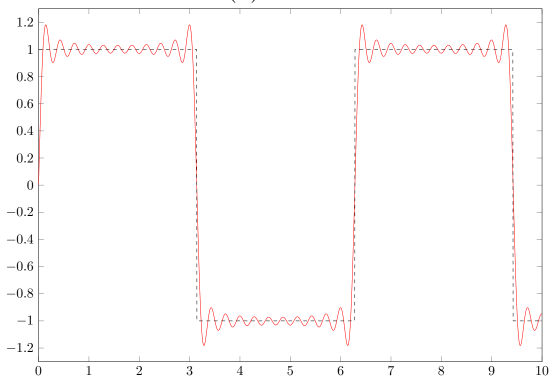
$$c = \frac{\lambda}{T} = \lambda f = k\omega \ . \tag{1.5}$$



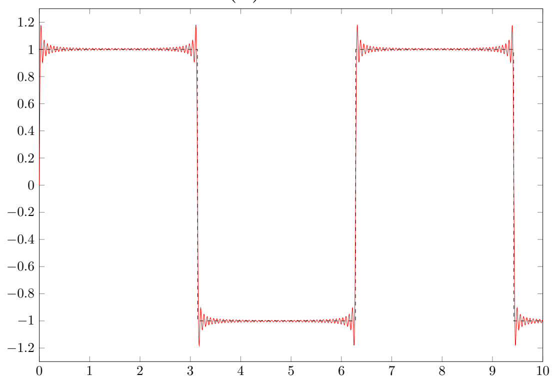
(a) $k = 1$



(b) $k = 5$



(c) $k = 11$



(d) $k = 49$

Figure 1.2: Addition of sinusoids to approximate a square wave. $k$ describes the number of added sinusoids.

For a sinusoidal acoustic signal with a frequency of 1 $kHz$, the wavelength in air is $340/1000 = 0.34$ $m$, and in water it is $1500/1000 = 1.5$ $m$. Thus, the wavelength for a given frequency is about 4.4 times longer in water than in air.

The sinusoidal wave is the basic component of any wave. You can look at them as the atoms or building blocks of any signal. Some 200 years ago, the French mathematician Joseph Fourier showed that by adding sinusoids of different amplitudes, frequencies and phase you can actually create any signal. In Figure 1.2 a square wave is approximated by adding $k = \{1, 5, 11, 49\}$ different sinusoids. The approximation, drawn in red, is approaching the black square wave as the number of sinusoid in increasing.

## 1.2 Linear Machinery

Now we have a wave - let's measure it! Before doing this, we have to agree on some common units. The most direct way to measure some thing is to use linear units. In mathematics, when something is linear, it means they add together. For example, let's say it takes 10 seconds to eat a mash mallow and 20 seconds to drink a glass of lemonade. Now, if it then takes $10 + 20 = 30$ $s$ to eat a mash mallow and drink the lemonade we say that eating and drinking the two items are linearly related. The basic linear units that we are going to use are listed in Table 1.1.

| **Magnitude** | **Abbreviation** | **Unit** | **Unit Abbreviation** |
|---|---|---|---|
| Time | $t$ | Second | $s$ |
| Period | $T$ | Second | $s$ |
| Frequency | $f$ | Hertz | $1\ Hz = 1\ \frac{1}{s}$ |
| Wavelength | $\lambda$ | Meter | $m$ |
| Force | $F$ | Newton | $N$ |
| Pressure | $p$ | Pascal | $1\ Pa = 1\ \frac{N}{m^2}$ |
| Energy | $W$ | Joule | $1\ J = 1\ Nm$ |
| Power | $P$ | Watt | $1\ W = 1\ \frac{J}{s}$ |
| Intensity | $I$ | $\frac{\text{Watt}}{m^2}$ | $\frac{W}{m^2}$ |
| Energy per area | $E$ | $\frac{\text{Joule}}{m^2}$ | $\frac{J}{m^2}$ |

Table 1.1: Overview of common units used in bioacoustics

In acoustics, and especially in bioacoustics, there are many definitions that are misleading and confusing. For example, the acoustic energy passing across 1 $m^2$ is also sometimes called energy density, but quite often also energy flux density. Both these names are confusing, as in other branches of physics, energy density means how much total acoustic energy per volume the media is containing, and energy flux density is another name for intensity. As you continue to plow the fields of bioacoustic literature you will pull up quite a few of these weedy nomenclature odd-balls.

Because we deal with linear measures of vastly different orders of magnitude, we use suffixes in front of the unit to denote exponentials: $G$ (giga) means $10^9$, $M$ (mega) means $10^6$, $k$ (kilo) means $10^3$, $m$ (milli) means $10^{-3}$, $\mu$ (micro) means $10^{-6}$, and $n$ (nano) means $10^{-9}$. So 1 $Mm$ is a million meters and 1 $mm$ is 0.001 $m$; and 100 $\mu Pa$ is the same as 0.1 $mPa$ whereas 100 $kPa$ is 100,000 pascal, or close to the atmospheric pressure at the surface of the earth.

If you have two waves $y_1$ and $y_2$ meeting somewhere in space, their amplitudes sum up. The waves are linearly related, and the resulting wave has a magnitude of one plus the other one as shown in Eq. 1.7.

$$\begin{aligned} y_1(x) &= A_1 \sin(k_1 x - \theta_1) \\ y_2(x) &= A_2 \sin(k_2 x - \theta_2) \end{aligned} \tag{1.6}$$

$$y_1(x) + y_2(x) = A_1 \sin(k_1 x - \theta_1) + A_2 \sin(k_2 x - \theta_2) \tag{1.7}$$

This is called the principle of superposition and is true for acoustic waves (at least within reasonable sound levels): if one sound wave meets another one, they sum up and the result is a linear function of the two components.

## 1.3 Decibel (logarithmic) machinery

The most common way to report the amplitude of an acoustic signal is not on a linear scale, but on a logarithmic one. As you will see later, this makes sense for many reasons: it simplifies calculations (believe it or not!), and the results are more directly related to how we or other animals perceive sound. So, logarithms are a great help, but they are of no avail if they are not thoroughly understood by the user. Therefore, take some time to dwell on the following.

Let $y$, $a$ and $x$ be any numbers. If

$$y = a^x \tag{1.8}$$

then

$$\log_a(y) = x \ . \tag{1.9}$$

We call $\log_a(y)$ the $a$-logarithm of $y$, and $a$ is the base of the logarithm. The base can be any number (except 0), but popular choices are 2, 10 or the irrational number $e$ (which is about 2.71).

When you read about logarithms or calculate them, take great care. The notation can mean very many different things. You have to make sure which base is implied – else the numbers may make no sense at all. It is quite unusual to see the base written out explicitly, such as $\log_{10}(y)$. Instead, it is generally assumed that $\ln(x) = \log_e(x)$ means natural logarithms (with base $e$), and $\log(x)$ most commonly mean logarithms of base 10. But, some calculators, programs and texts use a different notation. For example, in Matlab, natural logarithms are written as $\log(x)$, and base 10-logarithms as $\log_{10}(x)$. Confused? Never mind – you should be!

Now, consider the three logarithmic laws below:

$$\log(x \cdot z) = \log(x) + \log(z) \tag{1.10}$$

$$\log\left(\frac{x}{z}\right) = \log(x) - \log(z) \tag{1.11}$$

$$\log(a^x) = x\log(a) \tag{1.12}$$

These laws can easily be verified from the definition of logarithms above (Eq. 1.10 - 1.12). For example, let us prove the first law:

Let $x = a^u$ and $z = a^v$. We get:

$$\log_a(x \cdot z) = \log_a(a^u a^v) = \log_a(a^{u+v}) = u + v = \log_a(x) + \log_a(z). \tag{1.13}$$

With the help of the third logarithmic law (Eq. 1.12) you can easily transfer a logarithm from one base a to another base b:

$$\log_b(y) = \log_b(a^x) = x\log_b(a) = \log_a(y) \cdot \log_b(a) \tag{1.14}$$

The decibel scale, $dB$, is defined using logarithms of base 10. The ‘Bel‘ is in honour of the American scientist Graham Bell, and the ‘deci‘ means 10. The scale is defined as

$$\text{dB} = 10\log_{10}(r) \ . \tag{1.15}$$

The factor 10 is introduced to give the decibel numbers a convenient magnitude. Also, the factor 10 makes the decibels biologically relevant: in bioacoustics, 1 dB roughly corresponds to the smallest sound intensity ratio humans – or most animals for that matter – can discern. The number $r$ is a ratio between a certain measurement and a reference. For sound intensity levels, this reference is denoted $I_0$.

$$\text{dB} = 10\log_{10}\left(\frac{I}{I_0}\right) \tag{1.16}$$

Here, $I$ is the measured linear intensity (in units of power passing through an area, or $\frac{W}{m^2}$).

## 1.4 Sound pressure converted to decibel units

In reality, we rarely measure sound intensity directly. Instead we use pressure sensors (microphones or hydrophones) and measure the sound pressure. For a free-field plane acoustic wave (let's wait a while to discuss what this actually means) the intensity and sound pressure are related through the Eq. 1.17.

$$I = \frac{p^2}{\rho c} , \tag{1.17}$$

where $\rho$ is the density, and $c$ is the speed of sound, of the medium (roughly 340 $m/s$ in air and 1500 $m/s$ in water). With the help of the third logarithmic law (Eq. 1.12) we get

$$\text{dB} = 20\log_{10}\left(\frac{p}{p_0}\right) . \tag{1.18}$$

In air, we usually use $p_0 = 20\mu Pa$ RMS (which is close to the human threshold of hearing at 1 $kHz$) and in water $p_0 = 1\mu Pa$ RMS (which does not mean anything biologically relevant; probably it was chosen to simplify calculations - and to draw a line in the sand between in-air and underwater acoustics). Root-Mean-Square (RMS) means root-mean-square (you will learn later how this is calculated).

It is important to note that when expressed on a decibel scale, sound intensities are <u>not</u> linearly related. To appreciate this, consider how to calculate the linear pressure from a certain decibel level. That is, if Eq. 1.18 holds, then

$$p = p_0 10^{\frac{\text{dB}}{20}} . \tag{1.19}$$

If you want to add a sinusoidal wave with sound intensity of 94 dB re 20 $\mu Pa$ with another one in perfect phase and with the same frequency and amplitude, the result is <u>not</u> 188 dB re 20 $\mu Pa$. The result is actually 100 dB re 20 $\mu Pa$: 94 dB re 20 $\mu Pa$ corresponds to 1 $Pa$, and 2 $Pa$ is the same as 100 dB re 20 $\mu Pa$ – try to calculate this by yourself with the formula , or by using the logarithmic laws (Eq. 1.10 - 1.12).

There are some good tricks that should be learned by heart about decibels. You can try to convert the following linear units into decibel units as shown in the Matlab example below:

```
20*log10(2) = 6 dB
20*log10(10) = 20 dB
20*log10(0.5) = - 6 dB
20*log10(3) = 10 dB
20*log10(5) = 14 dB
```

Algorithm 1.1: Short Matlab example

To do so, write command prompt '20*log10(2)', and then press return. With the help of these tricks and the logarithmic laws above you can transfer from linear units to decibels quite easily, without using a calculator. For example, let's say you want to know how many underwater decibels is an acoustic pressure of 500 $Pa$. Well, this is half of 1 $kPa$, which is 103 $Pa$. Thus

$$\begin{aligned} \text{dB} &= 20\log_{10}\left(\frac{500}{10^{-6}}\right) = 20\log_{10}\left(\frac{0.5\cdot 10^3}{10^{-6}}\right) && (1.20)\\ &= 20log(0.5) + 20\log_{10}\left(\frac{10^3}{10^{-6}}\right) && (1.21)\\ &= -6 + 20\log_{10}(10^9) = -6 + 9\cdot 20\log_{10}(10) && (1.22)\\ &= -6 + 9\cdot 20 = -6 + 180 && (1.23)\\ &= 174\text{dB re}1\mu Pa. && (1.24) \end{aligned}$$

or, let's say you want to know many Pascals are 150 dB re 1 $\mu Pa$. Because $20\log_{10}(3) =$ 10 dB (see Alg. 1.1), adding 10 dB corresponds to multiplying the linear unit with 3. We can thus first calculate what 140 dB re 1 $\mu Pa$ is, and then multiply the answer with three. According to the formulas in equations above, $p = 3\cdot 10^{-6}\cdot 10^{140/20} = 3\cdot 10^{-6}\cdot 10^7 = 3\cdot 10^1 = 30Pa$.

## 1.5 Exercises

1. Complete the following exercises without any electronic help!
   a) How many dB re 1 $\mu Pa$ are?
      i. 1 $\mu Pa$
      ii. 1 $mPa$
      iii. 1 $Pa$
      iv. 1 $kPa$
      v. 1 $atm = 101.3\ kPa$ (or approximately 100 $kPa$)
      vi. 1 $nPa$
   b) How many $Pa$ are?
      i. 0 dB re 20 $\mu Pa$
      ii. 94 dB re 20 $\mu Pa$
      iii. 0 dB re 1 $\mu Pa$
      iv. 120 dB re 1 $\mu Pa$
      v. 126 dB re 1 $\mu Pa$
      vi. 240 dB re 1 $\mu Pa$
2. Try the following without a calculator:
   a) What is the resulting sound level (in dB re 1 $\mu Pa$ pp) if a sinusoid of amplitude 120 dB re 1 $\mu Pa$ pp (pp means peak-to-peak, measured from the maximum to the minimum level) is mixed with another sinusoid of same frequency and phase with the same amplitude 120 dB re 1 $\mu Pa$?

b) Same as above, but now there is a sinusoid of 120 dB re 1 $\mu Pa$ pp mixed with a sinusoid of 126 dB re 1 $\mu Pa$ pp.

c) Same as above but now there is a sinusoid of 120 dB re 1 $\mu Pa$ pp mixed with a sinusoid of 120 dB re 1 $\mu Pa$, perfectly out of phase (180 degrees) with the first one.

3. Try the following using Matlab in the Matlab Command Window:

```
% Take the attached file 'eigil.wav' and put it in your class folder (the folder where you keep the stuff for this class). In Matlab, set path to the same folder (see Mann's book Chapter 2).
[sig, fs] = audioread('eigil.wav');

% The signal is stored in the vector sig. You can check how long it is or how many channels there are:
length(sig)
size(sig)

% The sampling rate is stored in the variable fs. Try
fs

% Also try
audioinfo('eigil.wav')

% Which will give you some details about the file. What does 'BitsPerSample' mean you think?
% Later on, we will discuss what all these things mean. But already now we can make a nice plot of the signal, as a function of time. Define a time vector by
t = [0:length(sig)-1]*1e6/fs;

% Try to figure out what this means, and then type
plot(t,sig)

% You can also measure some basic features of your signal with the commands:
max(sig)
min(sig)
mean(sig)
var(sig)
sqrt(mean(sig.^2))
```

Algorithm 1.2: Execute the black lines in the Matlab Command Window

# 2. Acoustic signals in the time domain

## 2.1 A few notes on vectors and matrices in Matlab

Matlab is showing its almost unlimited strengths when it comes to vector and matrix algebra. The great thing – and perhaps the most important thing – to realize in this compendium is that almost any digitized bioacoustic measurement or manipulation that you may want to perform can be boiled down to vector and matrix algebra. Therefore, Matlab is a very powerful tool to perform (probably) any kind of bioacoustical analysis.

First, you need to know some essentials abut linear algebra, and how vectors and matrices are used. A vector is just a bunch of numbers located in different elements. A row vector can be written as [ 2 5 3 3 1 -7 0 9 ]. This vector is said to have 1 row and 8 columns. You can write a column vector as

$$\begin{bmatrix} 2 \\ 3.2 \\ -1 \\ 0 \\ -0.001 \\ 5 \end{bmatrix} \tag{2.1}$$

This vector has 6 rows and 1 column. A matrix is a bunch of horizontal vectors piled below each other, or likewise a bunch of column vectors stacked side by side. This is an example of a matrix with 4 rows and 4 columns:

$$\begin{bmatrix} 5 & -2 & 0.4 & 2 \\ 0.1 & -1.1 & 2 & 1 \\ -\frac{1}{3} & 9 & -1 & 0 \\ 2 & 1 & 0 & 0 \end{bmatrix} \tag{2.2}$$

You can add, subtract, multiply and divide matrices and vectors. However, there are certain rules that need to be followed. Two vectors or matrices can only be added or subtracted if they have the same number of rows and columns. For example, if a=[2 3 1 5] and b=[5 1 2 3], then a+b=[7 4 3 8].

In Matlab, write

```
a=[2 3 1 5]; b=[5 1 2 3];a+b
```

and see what happens! Also, try

```
a(1)
b(3:4)
a(2:end)
```

This is the way we can tease out a few or all elements of a vector. While at it, you can also try

```
sum(a)
```

and you sum up all the elements in the vector. With

```
a'
```

your row vector is suddenly a column vector.

The matrix given in Eq. 2.2 can be assigned the variable D in Matlab as

```
D=[5 -2 0.4 2;0.01 -1.1 2 1; -1/3 9 -1 0; 2 1 0 0];
```

Note that semicolons are used to jump from one row to the next. Now, let's try

```
D(2,4)
D(:,3)
D(end,:)
D'
```

Rows and column numbers are separated with a comma. The colon (:) indicates "all elements in that column/row". You can multiply or divide a vector or a matrix with a certain number (a so-called scalar); this corresponds to multiplying all the elements in the vector/matrix with that number. For example, 2a=[4 6 2 10] and b/(-10)= [-0.5 -0.1 -0.2 -0.3] – try this out in Matlab!

You can also multiply vectors with vectors, and vectors with matrices. However, now you really enter the world of linear algebra and have to keep track on certain rules that you may not be familiar with from ordinary calculus. Let's leave this from now. It suffices to say that if you want to multiply each element in vector a with the corresponding element in b, you need to use the so-called dot product in Matlab: If you write

```
a.*b
```

then Matlab will answer [ 10 3 2 15]. Likewise, if you write

```
a.^2
```

this will give you each squared element in a, or [4 9 1 25].

## 2.2 Acoustic measures in time and frequency

Acoustic signals are usually measured at a certain point in space as a local pressure disturbance varying with time. The signal can either be characterized in the time domain, or equally well in the frequency domain. These two domains are mirror images of each other – they are both equally valid descriptions of the same signal. Think of light: You may either describe electromagnetic waves in the time domain by measuring the intensity, or you may break up the light beam into its colour components using a prisma and measure each colour (which is identical to frequency) component separately. The light is the same, the two methods just describe its different features (or, in other words, they are measured using different units and magnitudes). In signal analysis, we usually jump between the time and frequency domains back and forth, both to conveniently address certain points of a discussion, or to analyse in the most efficient and least time-consuming manner. In the rest of this chapter, we will remain in the time domain.

## 2.3 Analog and digital signals

The "real" acoustic pressure signal is a continuous wave fluctuating around the ambient pressure. This is of course the one we are eventually interested in analyzing.

In the good old days, the signals were picked up by a sensor and stored on magnetic tapes as analog electric (or rather magnetic) signals. Then you could make a Fourier transform and get the spectrum of the analog signal. All this was very well – but the equipment was cumbersome, heavy, and usually with large restrictions in bandwidth and dynamic range (that is, the ratio between the smallest and largest intensity that could be handled in the recording).

Nowadays all this is done digitally. This means we measure (or sample) our analog signal at certain time intervals and store the results in a vector. We may now perform digital signal analysis on this vector, and end up with any measures we would like to have. This makes things much cheaper and faster than in the "good old analog days", and the dynamic range can be huge. But, even though the vectors in your computer are related to the actual analog signals and their spectra, you have to follow some rules and do some thinking when you interpret the results from your digital analysis and transform the results back into the "real" analog world.

## 2.4 Microphones and hydrophones: sensitivity

Sounds can be measured using a microphone in air or a hydrophone under water. The micro/hydro-phone translates the pressure into an electric voltage. We call such a magnitude translator a transducer. The microphone often consists of a vibrating, conductive membrane in close proximity to another conductive surface. The membrane and the surface are charged with usually a very high voltage. When the membrane vibrates, the capacitance between the membrane and the surface changes, and this can be picked up as a tiny voltage signal by the microphone amplifier.

A hydrophone is a crystal made of a so-called piezoelectric material. When such a material is exposed to pressure fluctuations it produces an electric voltage between its two surfaces. By soldering cables to each side of the crystal, this tiny voltage can be picked up and fed into an amplifier. Within a certain range, the voltage is proportional to the applied pressure, so by measuring the voltage you get a direct measure of the received sound level.

Both microphones and hydrophones have a certain sensitivity, defined as the amount of voltage coming out of the transducer divided by the amount of acoustic pressure applied to it. The sensitivity is frequency dependent and is usually best at lower frequencies. Above a certain cutoff frequency the sensitivity immediately starts to drop off. The cutoff frequency depends on the size of the transducer: the larger the transducer, the lower the cutoff frequency. The sensitivity is measured in linear units as $V/Pa$. In this compendium we use lower-case symbols to denote sensitivity ($s$), voltage ($u$) and pressure ($p$) in linear units, and upper case symbols ($S$, $U$, $P$) to denote the same magnitudes in decibel units. Thus, saying that sensitivity is voltage per pressure is the same as saying that

$$s = \frac{u}{p} \tag{2.3}$$

In logarithmic units, we define $S = 20\log_{10}(s)$, $U = 20\log_{10}(u)$, and $P = 20\log_{10}\left(\frac{p}{p_0}\right) = 20\log_{10}(p) - 20\log_{10}(p_0)$. Don't be confused about the fact that $S$ and $U$ seemingly have no reference unit – as a matter of fact for those logarithms the reference unit is just chosen as 1, for reasons that should become obvious below.

Now we can use the logarithmic laws to write:

$$\begin{aligned} S &= 20\log_{10}(s) \\ &= 20\log_{10}\left(\frac{u}{p}\right) \\ &= 20\log_{10}(u) - 20\log_{10}(p) \\ &= U - P + 20\log_{10}(p_0) \end{aligned} \tag{2.4}$$

If $p_0 = 1\ \mu Pa$ then the unit of $S$ is written as dB re 1 $V/\mu Pa$. In this case, $20\log_{10}(p_0) = 20\log_{10}(10^{-6}) = -120$ dB, so the last part of the formula above becomes $S = U - P - 120$.

Let's say a hydrophone has the sensitivity s = 30 $\mu V/Pa$. How to report this sensitivity in units of dB re 1 $V/\mu Pa$? We get $S = 20\log_{10}(3\cdot 10^{-5}) - 20\log_{10}(1) - 120 = 20\log_{10}(3\cdot 10^{-5}) - 120 = 20\log_{10}(3) + 20\log_{10}(10^{-5}) - 120 = 10 - 100 - 120 = -210$ dB re 1 $V/\mu Pa$.

Similarly, let's say that the sensitivity of a microphone is $S = -146$ dB re 1 $V/\mu Pa$. What is the corresponding linear measure of the sensitivity? $S = -146$ dB re 1 $V$ corresponds to $10^{-146/20} = 5\cdot 10^{-8} V$, so $s = 5\cdot 10^{-8} V/\mu Pa = 50 mV/Pa$.

## 2.5 Amplifying and filtering the signal

After the transducer, there is usually an amplifier. An amplifier is doing just what the name indicates: it amplifies the voltage with a certain number of decibels, or a certain number of times. Let's say the amplification is 40 dB. This means the signal is multiplied by 100 (because $20\log_{10}(100) = 40$). Again we will use lower case $a = 100$ for the linear amplification, and upper-case $A = 40$ dB for the corresponding decibels. They are related by $A = 20\log_{10}(a)$, or inversely by $a = 10^{A/20}$. Not that in the linear world you multiply the amplification with the signal, whereas in the decibel world you add the two together. This is because the logarithmic laws mentioned in the last chapter:

$$20\log_{10}(a\cdot u) = 20\log_{10}(a) + 20\log_{10}(u) = A + U \tag{2.5}$$

This is one more reason why decibels are nice: multiplications become additions, and divisions likewise become subtractions.

Right after the amplifier we usually install a set of filters. There is often a high-pass filter, removing low frequency noise. There should also always be a low-pass filter, filtering unwanted high-frequency components away (a so-called anti-aliasing filter, described more in detail below).

## 2.6 Analog-to-digital conversion

After the amplifier and filter, the signal is fed into an Analog to Digital Converter, ADC. This unit samples the signal so that it becomes ready to be stored on a computer. The sampling process is characterized by three numbers:

- the sampling frequency (or, how often samples are taken),
- the number of bits (indicating how detailed each sample should be measured), and

- the clipping level of the ADC, that is the maximum voltage that can be supplied to the ADC before the digitized signal is clipped.

How high should the sampling frequency be? According to the Swede-American Harry Nyquist (1889-1976), you should always sample at least twice as fast as the highest frequency component of your signal. If you do that, you can always reconstruct the analog signal from the digital signal. This is called the sampling theorem.

This is why you need the lowpass filter before the ADC (that is, the antialiasing filter mentioned above). By choosing this filter so that it filters out any frequency components above half the sampling rate, you make sure you do not violate the sampling theorem. When we discuss signal analysis, you will see how important this number is: $\frac{f_s}{2}$. In honour of this great Swede it is commonly known as the Nyquist frequency.

What happens if you do not live up to the sample theorem? If higher frequency components than $\frac{f_s}{2}$ exist in the signal that you sample, these components are mixed into the sampled signal at lower frequencies. The “new“ frequency components will be mirrored or folded down from the analog frequencies across the Nyquist rate $\frac{f_s}{2}$. After sampling, you will have no way to know which frequencies are real and which ones have been folded – the folded ones are said to be aliased with the real ones. Therefore, the above-mentioned low-pass filter is known as an anti-aliasing filter.

The number of bits decides how good you measure the samples. Usual number of bits was in the early eras of ADCs 8, then 12, then 16 – and now very often 24. As a general theoretical rule, for each bit you add 6 dB in dynamic range. The better the dynamic range, the better signal to noise ratio can be achieved. The downside of using many bits is that it increases the amount of data you need to store. Let us say you sample at 100 $kHz$ and use 16 bits. We say 8 bits = 1 byte, so each second you have to store 200 $k$Byte of data. If you want to use 24 bits, you now have to store 300 $k$Byte every second. Storage capacity used to be expensive but nowadays it is very cheap, so we are not as picky in our choice of neither sampling frequencies nor number of bits as we used to be. But, in some extreme bioacoustics applications (such as small acoustics tags on animals) you still have to think carefully on how you choose your sampling to make data storage possible.

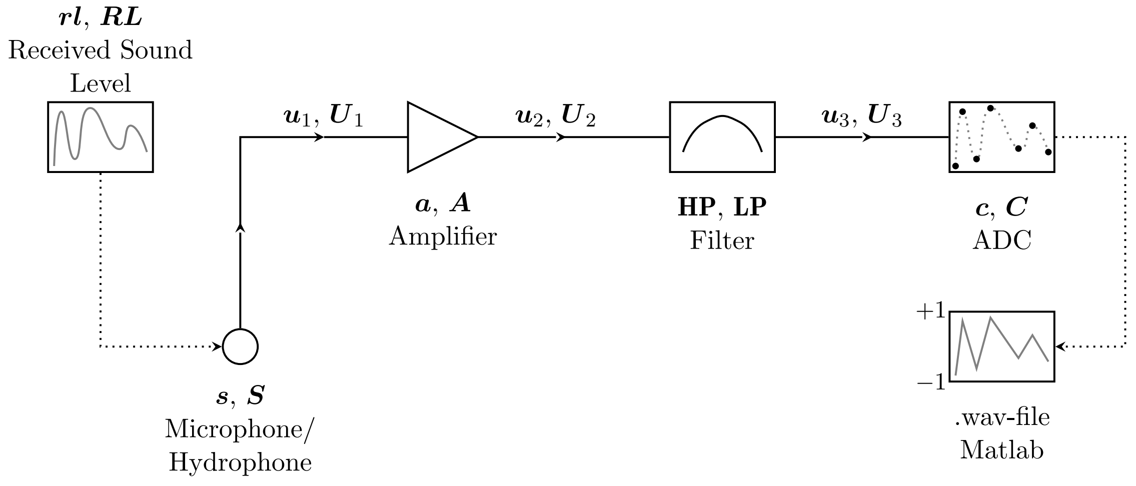
## 2.7 Tying together a recording system

Now we are ready to put everything together. Let's consider the figure below. A microphone or hydrophone of linear sensitivity $s$ (in $V/Pa$; or in decibels the $s$ sensitivity is $S$) is connected to an amplifier (amplifying a times, or $A$ dB), filter and an ADC (clipping level $c$ Volts, or $C$ dB re 1 $V$). A recording is made of a signal impinging on the micro/hydrophone with $rl$ Pa, corresponding to $RL$ dB re 1 or 20 $\mu Pa$. When you record this file with this recording system and open the WAV file in Matlab, the file will be automatically scaled between +1 and -1, corresponding to the clipping level $+c$ and $-c$ of the ADC.

## 2.8 The forward problem

First we will address questions such as: Given an acoustic signal of so many Pascals (or dB re 1 or 20 $\mu Pa$, for that matter), what is the amplitude of the signal in the Matlab display?

Let us assume that the received sound level has a peak pressure of $rl$ Volts, or $RL$ dB re 1/20 $\mu Pa$ p (where p stands for peak, and where I use 1/20 to denote the in air and underwater case, respectively). What will the corresponding peak level of the signal be in the WAV file that you open in Matlab?



$\boldsymbol{rl}$, $\boldsymbol{RL}$
Received Sound Level
$\boldsymbol{u}_1$, $\boldsymbol{U}_1$
$\boldsymbol{u}_2$, $\boldsymbol{U}_2$
$\boldsymbol{u}_3$, $\boldsymbol{U}_3$
$\boldsymbol{a}$, $\boldsymbol{A}$
Amplifier
**HP**, **LP**
Filter
$\boldsymbol{c}$, $\boldsymbol{C}$
ADC
+1
−1
.wav-file
Matlab
$\boldsymbol{s}$, $\boldsymbol{S}$
Microphone/
Hydrophone

Figure 2.1: Simplified recording chain

Let us first work through the example using linear units. The sensitivity of the micro/hydrophone is $s\ V/Pa$, so the voltage $V_1$ running through the hydrophone cable is $u_1 = s \cdot rl$. This voltage is amplified a times, giving us a voltage after the amplifier of $u_2 = a \cdot u_1 = a \cdot s \cdot rl$. Assume that the filter is designed so it doesn't affect the amplitude of the signal we want to record, but only remove low and high frequency noise (this is of course the appropriate way of filtering). Then $u_3 = u_2 = a \cdot u_1 = a \cdot s \cdot rl$. This signal is now digitized with an ADC having a clipping level of $c$ volts, which will correspond to the clipping level 1 in your Matlab WAV file. So, the peak value of your signal in your Matlab file is peak $= \frac{a \cdot s \cdot rl}{c}$.

The same calculation can also be done using decibels: start out with a received level of $RL$ dB re 1 $\mu Pa$ p. After passing the transducer this corresponds to a voltage of $U_1 = RL + S$ dB re 1 $V$. Amplify this by $A$ dB gives $U_2 = U_1 + A = RL + S + A$ dB re 1 $V$. Again, filtering is not assumed to change the signal level, so $U_3 = U_2 = RL + S + A$ dB re 1 $V$ as well. This is now run through an ADC with a clipping level $C$ dB re 1 $V$, so the signal opened in Matlab will have a peak level of Peak $= U_3 - C = RL + S + A - C$ dB. The unit of this expression is something like 'dB re peak', but it looks a bit silly so usually we just write it as "dB" and assume the reader know that the reference we talk about in this case is the peak of the Matlab file.

You can easily reassure yourself that Peak $= 20\log_{10}(\text{peak})$, using the first logarithmic law – just as you would expect.

Confused? Let's feed some numbers into this theory and see if that helps.

### 2.8.1 Exercises

1. Your microphone has a sensitivity of 50 $mV/Pa$, your amplifier amplifies $a = 100$ times and your ADC has a clipping level of 2 $V$ p. You record a bat with a received level of 74 dB re 20 $\mu Pa$ p. What is the signal peak in your Matlab WAV file? Try to perform the calculation both using linear and logarithmic units, and both by hand and by using a small Matlab script − all these four ways to Rome should give the correct answer, which is peak = 0.25 or Peak = −12 dB.

2. You receive a blue whale call of 160 dB re 1 $\mu Pa$ p with your hydrophone having a sensitivity of -200 dB re 1 $V/\mu Pa$. You amplify by 60 dB and record with an ADC having a clipping level of 10 $V$ p. What is the peak value of your blue whale signal when you open the WAV file in Matlab? Again, use your head trying to calculate

both with linear and dB units, and then do the same calculation in a Matlab script. The correct answer this time is peak = 1 or Peak = 0 dB – that is, you are right about to clip your signal; Better turn the amplification down a bit!

## 2.9 The inverse problem

Even though it is instructive to solve some forward problems given in Exercise 1 and 2, the inverse problem is much more useful in practice. Let us assume you measure the peak level of the signal in Matlab, peak is a number between 0 and 1. How many Pascals of received sound level does p correspond to?

Now you have to walk backwards in the diagram above. Because the clipping level in the Matlab WAV file is just 1, the signal voltage before the ADC must have been $V_3 = p \cdot C$. We pass straight over the filter to $V_2 = V_3 = p \cdot C$. The amplifier amplifies a times, so the level before the amplifier must have been $V_1 = \frac{V_2}{a} = \frac{p \cdot C}{a}$. As the sensitivity of the micro/hydrophone is defined by $s = \frac{V_1}{rl}$, we finally get $rl = \frac{V_1}{s} = \frac{p \cdot c}{a \cdot s}$. In logarithmic units, you can easily run through the same calculation and end with $RL = P + C - A - S$.

OK? – Time for some more examples!

### 2.9.1 Exercises

1. You come home to the camp after an exciting day in the jungle where you have recorded a howling monkey at very close distance. You used a microphone with a sensitivity of 100 $mV/Pa$, amplified with 20 dB, and after filtering the signal was fed into an ADC with clipping level 1 $V$ p. You open a WAV file in Matlab with a great howl, which has a peak level of p = 0.8. What was the received sound level? Try to do the calculation both in linear and decibel units – first using only your head and a pen and paper, then using Matlab. The correct answer is 0.8 $Pa$ p, or 92 dB re 20 $\mu Pa$ p.
2. The Eigil click you can find on Blackboard was recorded using a TC4014 hydrophone (sensitivity -187 dB re 1 $V/\mu Pa$), 20 dB amplification and an ADC with a clipping level of 5 $V$ p. What was the peak level of the recorded signal? The correct answer is 178 dB re 1 $\mu Pa$ p.
3. When you design a recording system, it is crucial to calculate its dynamic range (the lowest and loudest sound levels you can record with it). Estimate the dynamic range of the recording systems used in Exercises 2.9.1.1 and 2.9.1.2, assuming they are recorded with 16 bits and that you can actually record down to the lowest theoretical limit (often the lower limit is restricted by internal electric noise created by the recording gear, but let us leave that for now). We will discuss the correct answer to this exercise in the class.

## 2.10 A calibrated y-scale

You now have the appropriate know-how to create a calibrated y scale in your time series plots of acoustic data. It is of course not only the peak sample value that can be calculated by the formula given in the inverse problem – all your samples in your Matlab file can be transformed into pascals readily by adding these lines to your script plotting the Eigil click:

```
S = –187; % Sensitivity of hydrophone, in dB re 1 V/muPa
A = 20; % Amplification, in dB
c = 5; % ADC clipping level, in Volts
```

```
a = 10^(A/20); % Transforming amplification to linear units
s = 10^(S/20+6); %Transforming sensitivity to V/Pa
WAVClippingLevel = c / (a*s); %Clipping level of WAV file
sig = WAVClippingLevel * sig; % samples in units of pascals
```

Algorithm 2.1: Acoustic Parameters for 'eigil.wav'

Your plot will now have a y axis in pascals. The reason why the sensitivity and amplification is given in dBs whereas the ADC clipping level is given in volts, is more tradition than anything else. As a contrast to this underwater example: in in-air applications, the sensitivity is often given in linear units. Anyhow, it is good to always be prepared to switch between linear and dB units so you can handle information given in either way.

## 2.11 Calibration with a known sound source

To perform calibrated sound recordings, there are a lot of assumptions that must be valid: you must have used the amplification and ADC clipping level you thought you used, and the sensitivity of your micro/hydrophone must be correct. One easy way to make sure everything is correct is to record a sound source of known sound level with your recording system, using the same settings as in your actual recordings. This recording can be used to check that the clipping level of your Matlab WAV files is exactly what you would expect.

Calibrated sound sources are available for many types of professional hydrophones or microphones; if not, adapters can quite easily be built to fit any special model that you may have. The sound is a sinusoid of a certain frequency (for hydrophones usually 250 $Hz$, for microphones usually 1 $kHz$). The sound level given the calibrator is always in root-mean-square units (RMS) – we will later find out what this means, but for now it suffices to know that for a sinusoid (which is always what comes out of the calibrator) the peak level is 3 dB higher (or, in linear units, $\sqrt{2}$ times higher) than the given RMS level. In-air calibrators usually give a sound pressure of 1 $Pa$ RMS (or 94 dB re 20 $\mu Pa$ RMS), whereas hydrophone calibrators usually give something between 153 and 170 dB re 1 $\mu Pa$ RMS, depending on which specific adaptor is being used.

Assume you have recorded a calibration signal with a known received level of $cal$ $Pa$ RMS, and you open this file in Matlab. If the recorded calibration signal in Matlab has a peak of pcal, you can now immediately tell that the clipping level in your Matlab file corresponds to a received level of a $\sqrt{2} \cdot \frac{cal}{pcal}$ dB re 1/20 $\mu Pa$ p. You can use this information in two ways: You can compare it to the theoretically calculated clipping level (see for example exercise 2.9.1.3 above) to make sure everything is working properly in your recording chain. Once you are sure everything is fine, you can also use the calibration signal to directly calculating the clipping level and the calibrated sound levels in your recording, without bothering about the amplification, et cetera, of your recording system, as we did above.

### 2.11.1 Exercise

1. Open the file 'Cal163dB.WAV' into Matlab and measure its peak; then calculate the clipping level in the Matlab WAV file, given that the signal has a received level of 163 dB re 1 $\mu Pa$ RMS. Compare this to the theoretical clipping level calculated from the recording chain given in Exercise 2.9.1.2.

## 2.12 Pressure, Sound Pressure Level, Intensity

Once the signal has been transformed into calibrated levels of Pascals, it is easy to perform any measurements on them. We will use the Eigil file as an example here, but you are encouraged to bring in your own recordings and analyse those with the same techniques given below.

```
% The peak pressure is
p=max(sig)

% and the peak to peak pressure is
pp=max(sig)—min(sig)

%When we put this on a decibel intensity scale we get
% dBp=20*log10(max(sig)/20e—6)
%and
dBpp=20*log10(max(sig)—min(sig))/20e—6)
```

Algorithm 2.2: Pressure, Sound Pressure Level and Intensity in Matlab

This was all good for air dBs. For underwater dBs you would use $10^{-6}$ (or, in Matlab lingo: 1e-6) as the reference pressure.

Now, these are instantaneous peak measures only; what about averages? If you just take the mean value of the pressure signal, it is usually just giving a number very close to 0 (as the pressure fluctuates more or less an equal amount on the positive and negative side of the ambient pressure). So, a plain average is not so interesting. What is more interesting is to square the signal first, then average it and take the square root of that. This is called the RMS and is calculated as

$$p_{rms} = \sqrt{\overline{p^2}} \tag{2.6}$$

Here $\overline{p^2}$ denotes the mean of $p^2$. In decibels the notation is: $20\log_{10}\left(\frac{p_{rms}}{p_0}\right)$ dB re 1/20 $\mu Pa$ RMS.

In the free acoustic field, sound intensity is directly proportional to the squared sound pressure. Therefore the square of $p_{rms}$ is directly related to the average sound intensity. Sound intensity (or rather sound energy density, discussed below) is directly related to how we perceive sound. These are some of the reasons to why the RMS calculation is so powerful in not only acoustics in general but not the least in bioacoustics in particular.

There is a problem, though. The RMS level will depend on the length of the signal being averaged. The more of the "tails" of the signal that is included, the lower will the RMS level be. We therefore need good ways to define and calculate the duration of the signal before we can perform a solid RMS measure.

### 2.12.1 Exercise

1. For sinusoid signals, it can be shown that the p, pp and RMS levels are related as:

$$pp = 2p = 2\sqrt{2} \cdot p_{rms} \tag{2.7}$$
$$dBpp = dB_p + 6 = dB_{rms} + 9 \tag{2.8}$$

   Generate a sinusoid signal in Matlab, and measure the pp, p and RMS both in linear and dB units – can you verify these relationships? Can you figure out how they can be proven mathematically?

   - Hint 1: you can generate a nice sinusoid with the frequency 100 $Hz$ and duration 1 $s$ using:

```
fs=44100;
t=[0:fs−1]/fs;
y=sin(2*pi*1e2*t);
```

- Hint 2: you can calculate the RMS from the formula above, or you can use the ready-made rms()-function of Matlab.

## 2.13 The duration of a signal

There are many ways to define the duration of the signal, and the final choice depends on what signal we are working with, and what the signal to noise ratio of the recording is. Let's give some examples that can be tried out on the Eigil click.

One way to give an unambiguous measure of duration is to say: let's find the peak, then go a certain number of dBs down on both sides of the peak, and define the signal duration as the interval between these two limits. But, the signal is constantly oscillating between positive and negative values, so just going straight downhill from the peak following the signal shape would not make any sense at all – this would generate an extremely short duration! Instead, we want to find some function that follow the general shape of the signal waveform, not taking the erratic oscillations back and forth into account.

An easy, well-defined way to do this is calculating the envelope of the signal. It is done by a beautiful theoretical theory that we can discuss once we have been through spectral analysis. For now it suffices to say that in Matlab this 'trick' is performed with the one-liner

```
env = abs(hilbert(sig));
```

This generates a vector *env* having the same length as *sig*, and containing the envelope. abs is the Matlab function for the absolute number (in this case of an array of complex numbers generated by the function hilbert). If you are burning to know more about the intriguing and extremely useful properties of the Hilbert transform and the envelope function, there is a bit more information if you type help hilbert or doc hilbert. By the way, David Hilbert (1862-1943) was a German mathematician who had a huge influence in many areas of mathematics. It is interesting to note that some of the most essential theoretical findings that our present-day digital technology is based upon were created by scientists (such as Hilbert and Nyquist) at a time where computers were still only mind-boggling thought experiments.

Let's say you want to define the -10dB duration of the signal Eigil. This can be done with the following steps.

```
env = abs(hilbert(sig)); % calculate envelope function
thr = −10; % Threshold for duration estimation, in dB re envelope peak
linthr = max(env)*10^(thr/20); % Making the threshold into Pascals
duration = length( find(env>linthr) ) * 1e6/fs % Find length of envelope
    above threshold, in us
```

Algorithm 2.3: Calculation of the duration of a signal in Matlab

### 2.13.1 Exercise

1. Calculate and plot the envelope of Eigil.wav. Calculate the -3 dB and -10 dB duration of the click according to the instructions given above. Can you estimate the RMS sound intensity (in dB re 1 $\mu Pa$ RMS) for the part of the signal which is within the -3 and -10 dB durations? How do these two RMS levels compare?

## 2.14 Cumulative energy function

Another way to estimate the signal duration is to use the cumulative energy function. Consider the function cumsum() in Matlab. If you define vector a=[ 3 5 4 9 0 2], then use the cumsum()-function below:

```
cumsum(a) = [3 8 12 21 21 23];
```

The first element in cumsum(a) is just a(1), the second one is a(1)+a(2), and so on (the *i*-th element is sum(a(1:*i*))). This is called a cumulative summed array. Try to do this for the Eigil signal squared vector $sig.^2$; type:

```
csig = cumsum(sig.^2);
```

As the signal squares are proportional to the sound intensity, the *i*-th element of the *csig* vector is proportional to the amount of energy passing from the first sample until the *i*-th sample of the signal. Let's normalize this function by

```
csig = csig/max(csig);
```

Use the subplot command to create a plot of csig underneath the plot of the signal:

```
close all; figure(1)
subplot(2,1,1); plot(t,sig)
subplot(2,1,2); plot(t,csig)
```

Now we can define the duration of the signal as the interval containing 95% of the energy (say, starting at 2.5% and ending at 97.5% of the total energy):

```
durationE95= (min(find(csig>0.975))−max(find(csig<0.025)))*1e6/fs;
```

There are many more ways to measure duration: RMS duration, et cetera – but let's bother about those at some other time...

The different measures of duration all have their pros and cons. Estimating the duration from the signal envelope is very useful for signals with a well-defined peak. However, it is useless if the signal has multiple peaks. The duration given by the well-defined energy cumulative functions works fine if the signal to noise ratio is great. If there is too much noise in the recordings, the duration calculated in this way starts to depend on the duration of the original chunk of data used to estimate the cumulative function.

### 2.14.1 Exercise

1. Calculate the 95% cumulative energy duration for the Eigil click and compare to the -10 dB and -3 dB durations given above. Is it lower or higher?

2. Calulcate the dB RMS for a 95% energy duration window of the Eigil click and compare to the -10 dB and -3 dB RMS values that you calculated in Exercise 2.13.1.1. How does the RMS values compare to the p and pp values of a porpoise click, compared to the relationships you found for a sinusoid signal in Exercise 2.12.1?

## 2.15 The energy content of a signal

A final measure of a signal that is very useful and that is made in the time domain is the energy content:

$$E = \int I \, dt \tag{2.9}$$

As intensity has units of $W/m^2$, this becomes $J/m^2$, or the number of Joules passing a square meter. In the bioacoustics literature, this measure is also called Energy flux density or just Energy – both of which have other definitions in most acoustic text books. You could calculate $E$ by summing up the digitized samples of the calibrated signal intensities:

$$E = \frac{\sum sig^2}{f_s \cdot \rho \cdot c} \tag{2.10}$$

Once you have calculated the signal duration above, you can however get this done easier by noting that $I_{rms} = \frac{p_{rms}^2}{\rho c}$ is the average Intensity, and therefore the total energy density must be $E = I \cdot duration$, or in decibel units: $E = dB_{rms} + 10\log_{10}(duration)$. The final formula can be derived by:

$$E = 10\log_{10}\left(\frac{I \cdot t}{I_0 \cdot t_0}\right) = 10\log_{10}\left(\frac{I}{I_0}\right) + 10\log_{10}\left(\frac{t}{t_0}\right) = \mathrm{dB}_{rms} + 10\log_{10}(t) \tag{2.11}$$

Here we call the duration $t$, and $t_0$ is the reference duration for energy dBs, defined as $t_0 = 1\ s$. The unit of the energy density dB scale is dB re 1 $\mu Pa^2 s$. This is a rather strange unit. But it makes sense when looking at the expression behind the first equal sign in the equation above: The reference is $I_0 t_0$, and because intensity is proportional to pressure squared, the unit should be proportional to $Pa^2 s$. There is also a constant (the $\rho \cdot c$ product), but because this product is found both above and below the divisor it cancels out and is disregarded in the unit of the reference energy density.

### 2.15.1 Exercise

1. Estimate the energy content of the Eigil click inside a -3 dB, -10 dB and a 95% energy time window. How much do the measures differ?

# 3. Basic Measurements of Acoustic Signals in the Frequency Domain

## 3.1 Basics of the Frequency Domain

Usually when working in the field of bioacoustics we record animals, but we do not necessarily know how the signal looks like. Bats for example emit ultrasonic calls meaning they are using mainly frequencies which are higher than the human hearing threshold. In the recording process hydrophones and microphones sample the analog signal (for example a bat call) and digitize it (compare Figure 2.1).

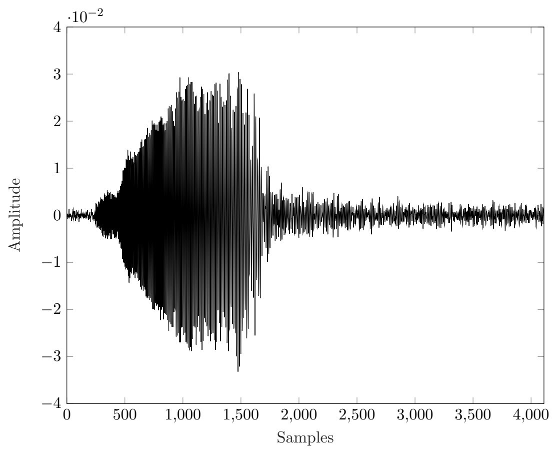
As a short recap; we talked in Section 2.3 and 2.6 already how sampling describes the conversion of a continuous to a discrete variable. For a proper reconstruction of the original analog signal from the digitized signal, we have to have the proper ratio of sampling rate to the frequency content. In the 1940s the two authors Nyquist and Shannon proposed their theorem. According to the *sampling theorem* (also referred to as Nyquist-Shannon sampling theorem), to properly sample an audio signal, the sampling rate needs to be at least two times the highest frequency which is tried to be captured:

$$f_{\text{sampling}} \geq 2 \cdot f_{\text{signal}} \ , \tag{3.1}$$

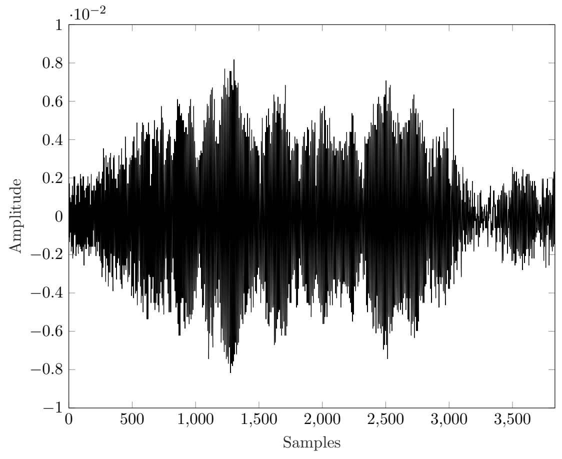
with $f_{\text{sampling}}$ the sampling frequency (usually in the literature just $f_s$) and $f_{\text{signal}}$ being the highest frequency component to be sampled. Half the sampling rate is called the *Nyquist Frequency* and is the highest frequency that can be accurately mapped in the digitization process (in Eq. 3.1 $f_{signal} = f_{Nyquist}$). If the sampling rate is too low in relation to the content of the analog signal the higher frequencies get mapped into lower frequencies. This effect of signals changing because the sampling rate is too low is called “aliasing“.

Figure 3.1 shows two different echolocation calls from bats plotted in the time domain. Figure 3.1a displays the echolocation call of a *Myotis daubentonii* and Figure 3.1b the call of a *Pipistrellus pipistrellus* bat. Obviously, we can tell that there are differences in amplitude or duration of the signal, but as we can not necessarily distinguish these two bat calls solely based on their oscillograms[1]. However, the calls of these bats are very distinctive and can easily be classified correctly when looking at the spectrum of the call the frequency domain.

[1]The in Figure 3.1 plotted oscillograms do not have a calibrated y-scale. However with the calibration process described in section 2.10 and 2.11 the scale could easily be derived.



(a) *Myotis daubentonii*



(b) *Pipistrellus pipistrellus*

Figure 3.1: Audio signal of two bats plotted in the time domain (oscillogram). Remark: the plotted oscillograms do not have a calibrated y-scale.

Any signal can either be described in the time domain (Chapter 2) or the frequency domain (Chapter 3). When we plot a signal as pressure or voltage over time, we call it an oscillogram (see Figure 3.1). When we plot it in the frequency domain, we have two plots: a magnitude spectrum showing how the signal energy is split into different frequencies, and a phase spectrum showing how the different frequencies should be delayed relative one another, again as a function of frequency. The two descriptions (oscillogram and spectrum) are complimentary – each contains all information available from the signal, and you can swap from one representation to the other without any loss of information (in theory at least).

## 3.2 Fourier Transform and common Signal Types

As mentioned above, when analysing signals we often are interested in the frequency content of the signals which is especially helpful in distinguishing proper signal content from noise. The Fourier analysis (or Fourier Transform (FT)) is a mathematical technique which allows us to break signals down into sinusoids and therefore view its spectral content. The shape of the time domain waveform (Figure 3.1) does not really provide too much valuable information which we could use to classify the bat species who originally emitted those signals. The interesting components are the frequency, phase and amplitudes of the sinusoids which are embedded in the signal.

In 1807 the French mathematician Joseph Fourier claimed that every continuous periodic signal can be represented as a collection of sinusoidal waves. Fourier's aim was to solve differential equations concerning heat conductivity – but his theory turned out to have an incredible number of other applications.

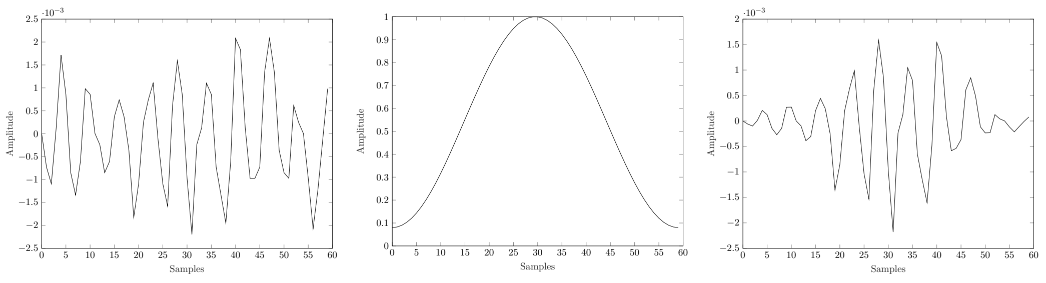
The FT has different mathematical formulation depending on what type of signal is present. The original theory was developed for continuous and periodic signals, but it was later expanded to include transients and random noise. The different mathematical descriptions in the time domain demand also different routes to their FTs.

Signals can either be *deterministic* or *random.* Deterministic signals can be described in the time domain as an amplitude over time, whereas random signal can not. The incomplete list below describe briefly the three most common types of signals and their FT:

- A continuous signal is an infinitely long time series and it can either be periodic or non-periodic (aperiodic). The FT is composed by a set of discrete frequencies with a constant frequency interval between them.
- A transient is a continuous signal with a limited extent in time. The FT of such a signal becomes a continuous curve ranging from a minimum to a maximum frequency.
- A random signal is defined by statistical parameters both in the time and in the frequency domain and can not be written as an amplitude as a function of time. At each time instant, there is a certain probability for the signal having a certain value a bit later. To estimate the frequency spectrum statistical techniques are used.

## 3.3 The Computation of Fourier Transform using the Discrete Fourier Transform and the Fast Fourier Transform

The Discrete Fourier Transform (DFT) is a technique which allows us to decompose digitized signals. The DFT can also derive the frequency spectrum of an aperiodic-discrete signal like an audio recording. The theory for the FT only applies to periodic signals. By “viewing“ a signal as a single segment which is “copied“ and repeated again after the end of the previous segment, we can assume periodicity for the whole signal. The “copies“ of the signal segments need to be smoothly connected. The smooth connection is done by using a window function. The window function ensures that there are no jumps between the start and end of the signal in each segment. A common used window function is the Hamming window. This process of smoothing a segment is shown in Figure 3.2. More on windowing and the effects in Section 3.7. If we do not use a window function, the discontinuities will result in sinusoids at all frequencies. [1]



(a) 60 samples of a bat call (b) Hamming Window (c) smoothed bat call

Figure 3.2: Multiplication of a 60 sample bat call with a hamming window function to smooth out the begin and end of the audio segment.

However, the DFT is a slow and computational expensive method, which is the reason that most of the time the Fast Fourier Transform (FFT) algorithm [2] is used in practice. For a more detailed summary and explanation of underlying mathematics of the DFT and FFT, the reader is directed to [1, 3, 4, 5]. The following paragraphs refer to Smith's [1] explanation of the concepts of a spectral analysis using the DFT.

The DFT takes an input signal $x[i]$ of $N$ samples and changes it into $2 \cdot \left(\frac{N}{2}+1\right)$ point output signals. The input signal is in the time domain. The output signal $X[i]$ is in the so-called frequency domain, containing the amplitudes of the sine and cosine waves. The first $\frac{N}{2}+1$ samples of the output signal are the Real part of $X[\cdot]$ (cosine wave amplitudes), whereas the part from $X[\frac{N}{2}+1]$ and on-wards is called the Imaginary part of $X[\cdot]$ (sine wave amplitudes).

$$x[i] = \sum_{k=0}^{N/2} \mathrm{Re}\overline{X}[k] \cdot \cos\left(2\pi \frac{ki}{N}\right) + \sum_{k=0}^{N/2} \mathrm{Im}\overline{X}[k] \cdot \sin\left(2\pi \frac{ki}{N}\right) \ , \tag{3.2}$$

Eq. 3.2 synthesises a signal in the time domain and describes it by its respective amplitude of the basis functions $\cos\left(2\pi\frac{ki}{N}\right)$ and $\sin\left(2\pi\frac{ki}{N}\right)$. $\mathrm{Re}\overline{X}[k] = \frac{\mathrm{Re}X[k]}{N/2}$ and $\mathrm{Im}\overline{X}[k] = -\frac{\mathrm{Im}X[k]}{N/2}$ are scaled amplitudes for every frequency in the signal. In other words, $x[i]$ can be represented as $\frac{N}{2}+1$ cosine plus $\frac{N}{2}+1$ sine waves. The amplitudes are stored in the two vectors $\mathrm{Re}\overline{X}[k]$ and $\mathrm{Im}\overline{X}[k]$.

But how can we compute the amplitudes of the sine and cosine waves? We use the concept of correlation. Correlating a signal with another signal returns a single measure reflecting their similarity. In simpler terms: with correlation we can detect a known waveform embedded in our input signal. To calculate this "similarity measure" we multiply two signals and sum up all the points. Eq. 3.3 and 3.4 correlates each sample $i$ of the input signal $x[i]$ with the basis functions $\cos\left(2\pi\frac{ki}{N}\right)$ and $\sin\left(2\pi\frac{ki}{N}\right)$. The index $i$ represents the samples of the input signal and runs from 0 to $N-1$ ($N$ is the total number of samples in $x$). On the other hand, $k$ runs from 0 to $\frac{N}{2}$ representing the discrete frequencies contained in our signal $x$.

$$\mathrm{Re}X[k] = \sum_{i=0}^{N-1} x[i]\cos\left(2\pi\frac{ki}{N}\right), \text{ for } k = 0, 1, \ldots, \frac{N}{2} \tag{3.3}$$

$$\mathrm{Im}X[k] = -\sum_{i=0}^{N-1} x[i]\sin\left(2\pi\frac{ki}{N}\right), \text{ for } k = 0, 1, \ldots, \frac{N}{2} \tag{3.4}$$

Eq. 3.3 and 3.4 can be used to represent the frequency domain. However, another way of viewing the frequency domain is through the use of magnitude $M$ and phase $\theta$ of a combination of sine and cosine. This is called the polar notation[2].

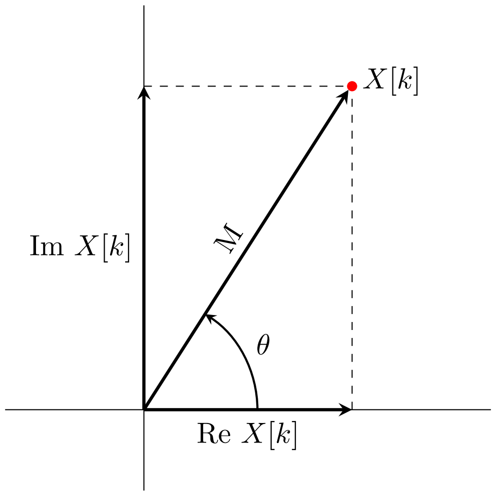


Figure 3.3: Rectangular-to-polar conversion using the complex plane, modified from [1, p. 162].

Adding a sine and cosine will result in a new cosine, following

$$A\cos(x) + B\sin(x) = M\cos(x+\theta)\ . \tag{3.5}$$

Consequently, as visualized in Figure 3.3, the conversion in magnitude and phase follows as

$$\mathrm{Mag}X[k] = \sqrt{\mathrm{Re}X[k]^2 + \mathrm{Im}X[k]^2}\ , \tag{3.6}$$

$$\mathrm{Phase}X[k] = \arctan\left(\frac{\mathrm{Im}X[k]}{\mathrm{Re}X[k]}\right)\ . \tag{3.7}$$

[2]On a small side note: the polar notation is also used to visualize complex numbers in the *complex plane*.

The difference between the rectangular notation and the polar notation of the DFT is that, for the first, the DFT composes the signal in $\frac{N}{2}+1$ sine and $\frac{N}{2}+1$ cosine waves with a specified amplitude, whereas for the polar notation the decomposition is into $\frac{N}{2}+1$ cosine waves of a magnitude and a phase shift. The polar notation is usually used for visualization of the frequency domain because it is easier for humans to interpret.

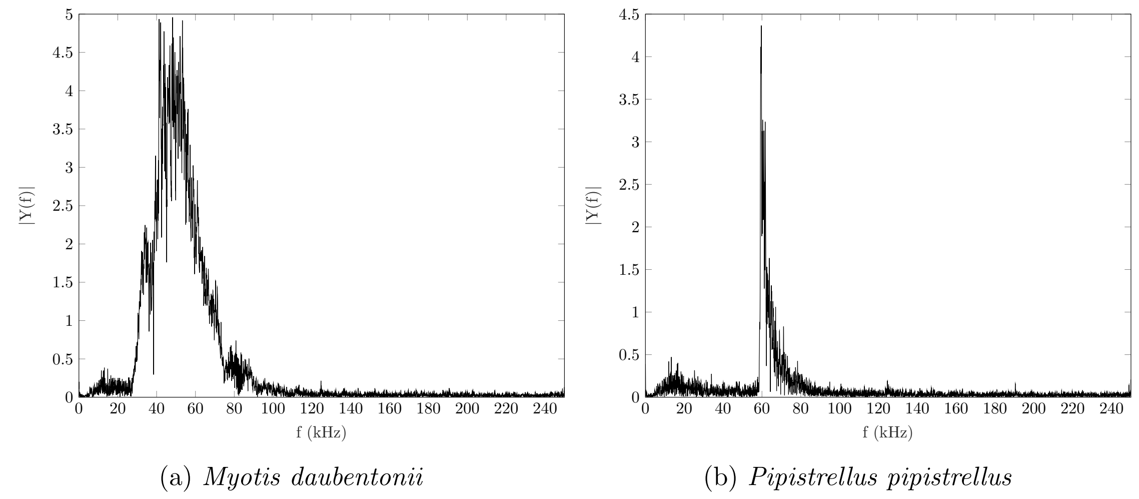


Figure 3.4: Frequency Spectrum of two bat calls derived using a FFT of $N = 20000$ length with one frequency bin representing 25 $Hz$.

As mentioned the DFT is not really used in practice. The FFT is an algorithm for computing the DFT efficiently. To make use of the computation efficiency, the length of the FFT is usually chosen as a size of $n_{\text{FFT}} = 2^k$. In case the length $N$ of the signal $x[i]$ is shorter than the length of the FFT $n_{\text{FFT}}$ the signal is padded with trailing zeros to match $N$. Applying a $n_{\text{FFT}} = 20000$ FFT to both bat calls (Figure 3.1a and 3.1b) results in the frequency spectra displayed in Figure 3.4. We are using the length of $n_{\text{FFT}} = 20000$ for the FFT so that one frequency bin is equal to $\frac{f_s}{N} = 25Hz$, with the sample frequency of $f_s = 500kHz$ (consider the sampling theorem again Eq. (3.1) for reconstructing a original signal of the bandwidth $0 - 250$ $kHz$). Note, we have to multiply the resulting amplitudes by the correction factor 2, because the FFT computes a double-sided frequency spectrum with positive and negative frequencies. Since we only view the positive half of the frequencies (one-sided spectrum) and energy conservation must be true, we have to scale the amplitudes of every positive frequency bin with the scalar factor of 2.

Moreover, since we are working with a sampled signal (a discrete signal) we have to specify: the frequency spectrum is not a continuous function but instead consists of discrete frequency bins. A frequency bin is a short segment $[f_l, f_h]$ on the frequency axis that integrates the energy over a range of frequencies resulting from a DFT (or FFT). Due to sampling it is not possible assign a precise amplitude to every possible frequency. Frequencies in-between the samples are summed up into a frequency bin. The spectral bin width can be derived for instance from the sampling frequency $f_s$ and the length of the FT $n$ as $\Delta f_{bin} = \frac{f_s}{n} = f_h - f_l$. Each of the frequency bins contains the amplitudes of all frequencies within the short segment $[f_l, f_h]$ and is often referred to by a single frequency. Popular choices are: the mid-frequency $\frac{f_l+f_h}{2}$, the lowest frequency $f_l$ or the highest frequency $f_h$ within the segment. [6]

Finally, some of the important conclusions that can be proven, but are not proven here are[3]:

[3]For more details can be found in the additional and more specific literature in [1, 3, 4, 5].

- To *any* oscillogram there is exactly <u>one</u> unique amplitude and phase spectrum and vice versa.
- A digital signal consisting of $N$ samples with the sample rate of $f_s$ has a DFT which is perfectly represented by $\frac{N}{2}+1$ points distributed between the frequencies 0 and $\frac{f_s}{2}$. Each of those points consists of a magnitude and a phase.

## 3.4 Frequency Spectra of different Signal Types

Leading up to this section we have discussed the different signal types (short list in Section 3.2) as well as how we can derive the FT of a signal. The following paragraphs describe the frequency spectrum which can be expected when we handle the three main types of signals: continuous, transient and random signals.

#### Continuous Signals

A continuous signal is an infinitely long time series. Periodic signals have a certain period after which they repeat themselves[4]. Such signals have a spectrum consisting of a fundamental frequency, overlayed with a number of harmonics (signals with a frequency which is a multiple of the fundamental frequency). Thus, the infinitely long continuous signal has a spectrum that has a number of well-defined, discrete frequency components.

#### Transient Signals

A transient signal consists of a continuum of frequencies, which are an infinite number of sinusoids with frequencies arbitrarily close to each other. These frequencies are in many cases confined within a certain bandwidth. In most cases, the shorter the signal, the broader is the bandwidth. In the extreme case of a very long signal, the bandwidth becomes narrow so that the spectrum may start to look a lot like the discrete spectrum of a continuous signal. As the signal becomes shorter (more well-defined in time) the spectrum broadens and becomes less defined in the spectral domain. In the most extreme case, a signal which solely consists of a single impulse at a one point in time and is zero elsewhere (a so-called Dirac pulse), the amplitude spectrum is perfectly flat, starting at frequency zero and lasting until an infinite high frequency.

#### Random Signals

These signals are typically noise such as ambient noise or background noise of a recording. But, it is important to mention, that noise may have tonal or transient components. As an example, the “noise“ could consist of someone playing a trumpet or drums in the background of a recording. Then the noise is not completely random anymore but has deterministic (predictable) components.

#### Summary

To sum it up, if the signal is undefined in time (like a continuous signal) its frequency content is very well defined. Further, if the signal has a short on- and off-set (that is, it is well-defined in time; a transient), its spectrum becomes very blurry. This is a fundamental principle in spectral analysis: you cannot have a signal both well-defined in time and frequency at the same time – it is either one or the other[5].

There are some other properties (some discussed earlier in Chapter 3) of the FT that are important to know about:

[4]Non-periodic continuous signals are not considered in the following.

[5]In quantum physics this is exactly the relation leading to Heisenberg’s uncertainty principle, that you may have heard about: it is impossible to determine a particle’s coordinates and velocity at the same time.

- Fourier's theory only works if you let the frequency scale run from $-\infty$ to $+\infty$.
  - What do negative frequencies mean?
    - The negative frequencies do not mean anything physically, but they are given to you as an unwanted gift as a part of the analysis (compare Eq. 3.2 - 3.7).
    - For real signals they are just mirror images of the positive frequencies and don't contain any new (or additional) information.
  - What is an infinitely high frequency?
    - The fact that frequencies can become infinitely high is just the mathematician's way to leave the highest possible frequency content in the analysis without any bounds.
- Infinitely long continuous signals have discrete spectral components.
- Transients consist of a continuum of frequencies with an infinitely number of frequency components. Infinitely short transients have an infinitely long, continuous frequency distribution.
- Two signals that look very different in the time domain may look very similar in their magnitude spectrum.
  - For example a Dirac pulse and so-called white noise. They have the same frequency distribution (ranging over all frequencies). The Dirac pulse is infinitely short, whereas the white noise is completely randomly forever. The way they differ is in the phase spectrum: for the Dirac pulse all frequencies have the same phase, whereas for white noise all phase components are completely random.

## 3.5 Frequency Analysis in Practice using Matlab

This section provides some practical examples on how to use Matlab to perform a frequency analysis. For this we use the Eigil signal (`'eivil.wav'`) as an example. Conveniently we will denote an input signal with small letters and its FT with capital letters. The signal is $N = 256$ samples long. Since the signal is discrete and the length of the signal is a multiple of $N = 2^k$ with $k = 16$ we can use the FFT algorithm to transform the signal from the time to the frequency domain. In Matlab the function `Y = fft(y,n)` computes a DFT using the FFT algorithm. Hereby is $y$ the signal with $y = \{y_1, y_2, \ldots, y_N\}$ and $n_{\text{FFT}} = n$ specifies the number of points used for the DFT[6]. If $n > N$ then the signal gets padded with zeros. The so-called *zero-padding* describes the addition of zeros at the end of the signal. Zero-padding is an easy way to increase the number of points in the FT without changing the signal's content or introducing non-existent frequencies. But be aware when $n < N$ the signal $y$ gets truncated to match the length $n$. Furthermore, you have to divide the output $Y$ always by the length $N$ of the signal[7].

The output $Y$ is a column vector of complex numbers[8] having $N = n$ samples[9] (in case $y$ is the `'eigil.wav'` signal $N = 256$). The amplitude is derived by taking the absolute value of each[10]. After taking the absolute values of the FT we get a real-valued output

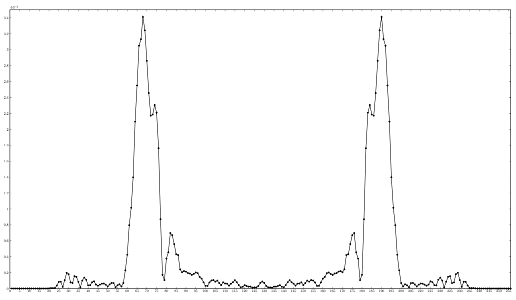
[6]Important Remark: From here on we will denote the length of a FFT always with the small letter $n$. It can be assume that $n_{\text{FFT}} = n$ for the rest of this chapter.

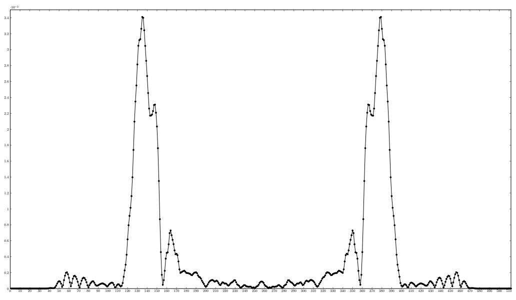
[7]The reason is the way Matlab calculates the FFT; you always have to divide by the number of samples $N$ to get the true amplitude of each frequency component.

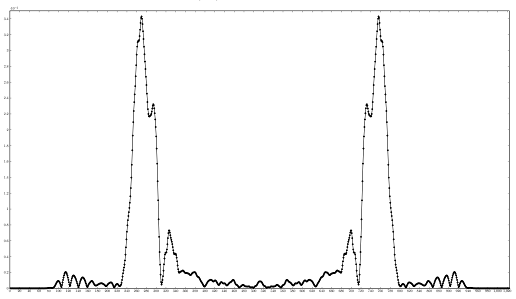
[8]Look at the output `whos Y`.

[9]$\text{length}(Y) = \text{length}(y)$

[10]Calculating the absolute value of a complex number $c$ is done through $C = |c| = abs(c) = \sqrt{\text{Re}(c)^2 + \text{Im}(c)^2}$.


(a) $n = 256$


(b) $n = 512$


(c) $n = 1024$

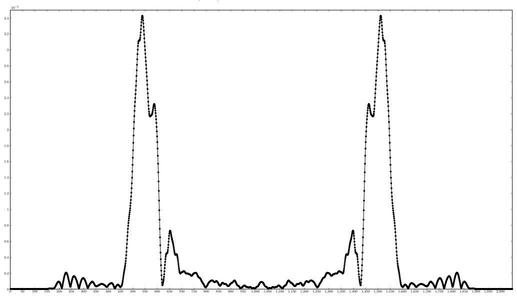

(d) $n = 2048$

Figure 3.5: Two-sided FTs of `eigil.wav` with various lengths $n_{\text{FFT}} = n = \{256, 512, 1024, 2048\}$ (using the FFT algorithm). In case $n > 256$ the signal gets padded with zeros in the time domain and therefore the frequency spectrum contains more points (see (b) - (d)). However, since the the overall number of frequencies that the signal can contain does not change (remember sampling theorem Eq. 3.1) we have more points available to cover the same frequency range. Thus resulting in a more accurate (and smooth) representation of the frequency spectrum. (Note: the graphs do not have a calibrated x- or y-scale.)

signal $Y$ with the length $n = N$, the same length as the original input signal $y$.

```
% Compute the real—valued FT of the input signal y
Y = fft(y)/length(y);
Y = abs(Y);

% Compute different lengths n of the FT Y of the signal y
Y_512 = abs(fft(y, 512)/512);
Y_1024 = abs(fft(y, 1024)/1024);
Y_2048 = abs(fft(y, 2048)/2048);

% Plot the two—sided frequency spectrum for n = 1024
figure
plot(Y_1024, '.k—')
```

Algorithm 3.1: Calculating and plotting a FT in Matlab.

Algorithm 3.1 converts a signal `y` from the time to the frequency domain. Figure 3.5 plots all the FTs of the `eigil.wav` that we computed in Algorithm 3.1. Looking in more detail in the figures we can see that all graphs (Figure 3.5a - 3.5d) display the exact same spectrum. Moreover, it is apparent that the graphs are mirrored exactly in the middle of the x-axis. This mirrored frequency spectrum is called a *two-sided* frequency spectrum. The reason is founded in of the computation of the frequency spectrum. The DFT will always result in a plot which covers the frequency range of $[0, \pm\frac{f_s}{2}]$ with $\frac{f_s}{2} = f_{Nyquist}$ or in case of `eigil.wav` $[0, \pm 250]\ kHz$ (more on that in section 3.3).

Additionally, the "smoothness" (and accuracy) of the FT representation increases from Figure 3.5a - 3.5d because the amount of available points in the time domain[11] increases from $n = 256$ to $n = 2048$ due to *zero-padding*. Because only zeros are added at the end of the signal (increasing the length of the input signal) there are more points available to represent the FT as the DFT decomposes a discrete signal with the length $N$ in time into a $2 \cdot \left(\frac{N}{2} + 1\right)$ output signal in the frequency space. And since the frequency range contained in the signal does not change, the added points have to fall in the range of $[0, \pm\frac{f_s}{2}]$ thus generating a more "smooth" representation of the FT.

Using Matlab to illustrate the paragraph above on how Matlab extends the signals with zeros in case $n > N$ (*zero-padding*) we assume `y` is the vector representing the `eigil.wav` signal with $N = 256$ samples and execute the code in Algorithm 3.2. Algorithm 3.2 results in two real-valued vectors representing the FT `Y` and `Y2` of the input signal `y`. Before computing the FFT we added a column vector with 256 zeros at the end of the signal (denoted as `y2`). Keep in mind to divide by the number of samples of the original signal to retain the true amplitude of each frequency bin. Subtracting `Y` from `Y2` and summing everything up results in zero (`a = sum(Y2 - Y) = 0`[12]).

```
% Compute the real—valued FT Y of the input signal y
Y = abs(fft(y, 512)/length(y));

% Add zero—padding by hand and compute the real—valued FT Y2
y2 = [y; zeros(256,1)];
Y2 = abs(fft(y2)/length(y));

```

[11] Read section 3.3 or [1, 3, 4, 5] for more details.
[12] If $a \neq 0$, then double check your code. $a$ has to be zero!

```
% check whether both output signals are exactly the same
a = sum(Y2-Y);
```

Algorithm 3.2: Illustration of how the `fft()`-function in Matlab extends the signals with zeros.

Coming back to the four plots in Figure 3.5 and the fact that each plot is its own mirror at $f = \frac{f_s}{2}$. To get rid of this we "exclude" the frequencies in the range of $[-\frac{f_s}{2}, 0]$ by cutting the output vector in half and then adding the energy of the frequency bins of the second part to the first part of the FT vector by multiplying with the factor 2 (compare end of section 3.3). Because the first and last elements of the signal are not iterated twice (the first element is the DC-component, and the last one is the one over which the top elements are folded), you need to divide those elements again by the factor 2. This approach results in a *one-sided* frequency spectrum and is given in Algorithm 3.3 below.

```
% Compute the real-valued FT Y of the input signal y
Y = abs(fft(y)/length(y));

% Cutting the output vector in half and "excluding" the second half
Y = Y(1:length(y)/2+1);

% For the conservation of energy multiplication by 2
Y = 2*Y;
Y(1) = Y(1)/2;
Y(end) = Y(end)/2;
```

Algorithm 3.3: Removing the mirror at $f = \frac{f_s}{2}$ and conserving the total energy in the FT of the signal.

In the beginning of this section we denoted the length of the input signal $y$ as $N$ ($N$ is the number of discrete samples taking during the recording process). Further, $n$ is the length of the FFT (often also in literature or Matlab denoted as `nfft`, or here: $n = n_{\text{FFT}}$) and represents the number of frequency bins. The output of the FFT is $Y$. The *one-sided* spectrum $Y(f)$ has the length of $\frac{N}{2}+1$ (see Algorithm 3.3). The x-axis displays the frequencies from 0 to $\frac{f_s}{2}$. As a consequence we create an evenly spaced vector representing the frequencies 0 to $\frac{f_s}{2}$. This can either be done using the Matlab command `linspace(a, b, c)` or by manually creating a vector that runs from 0 to $\frac{f_s}{2}$ with a step width of exactly the difference between each of the $n = n_{\text{FFT}}$ points of the FT meaning the width of one frequency bin in the spectrum using the notation `start:step_width:end`. This vector represents our x-axis and can be used when plotting the frequency spectrum.

After calibrating the x-axis we need to calibrate the y-axis. In all the previous figures we have used the `eigil` signal in an non-calibrated version. However, if the signal we want to transform to the frequency space has a calibrated y-axis in $Pa$ (Pascals) (for more details on the calibration of the y-axis in the time domain see Section 2.10 and 2.11), the calibration is straight forward.

$$Y(f) = 20 \cdot \log_{10}\left(\frac{Y(f)}{p_0}\right) \quad , \tag{3.8}$$

where $Y(f)$ is the FT of an input signal $y$ and $p_0$ the reference pressure. Remember: the reference pressure $p_0$ is different in water and in air. In water we are using a reference pressure of $p_{0,water} = 1\ \mu Pa = 1 \cdot 10^{-6}\ Pa$ whereas for signals recorded in air $p_{0,air} =$

$20\ \mu Pa = 20 \cdot 10^{-6}\ Pa$. Eq. 3.8 calibrates the amplitudes of the FT with the unit [dB re $p_0$].

Lastly, applying everything we talked about in this chapter we can plot the frequency spectrum of the `eigil.wav` signal by calibrating the y-axis in the time domain. Then transforming the signal to the frequency spectrum using a FFT with the length of $n = 2048$. Next we need to retrieve the *one-sided* version of the spectrum while keeping the conservation of energy in mind. And finally calibrating our x- and y-axis. After following all the listed steps we plot our signal and should find ourselves with the graph in Figure 3.6.

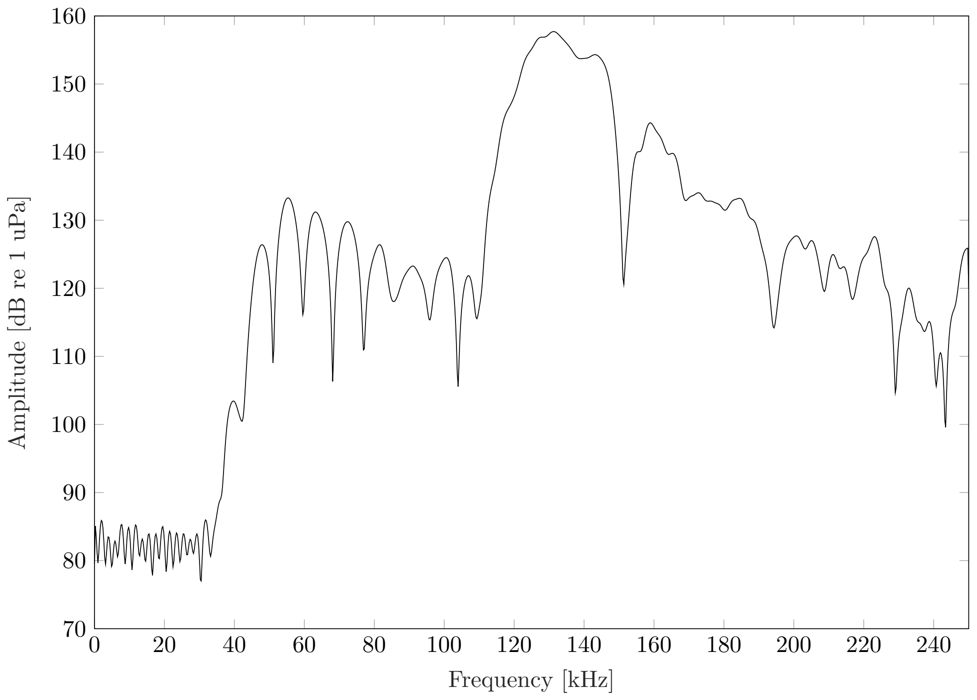


Figure 3.6: Calibrated frequency spectrum of `eigil.wav` using a $n = 2048$ FFT.

### 3.5.1 Exercise

1. Write a small script that plots both the oscillogram (amplitude over time) and spectrum (amplitude over frequency) of the supplied `sinusoid.wav`, `square.wav`, `sawtooth.wav` signals as well as the porpoise signal `eigil.wav`.

## 3.6 Measures in the Frequency Domain

It is important to derive specific and distinctive features which describe the signal accurate and unbiased. It is not viable to always compare the whole signal as such. Different circumstances or environmental factors might result in completely different recordings of the same animal or species. Issues with surrounding noise or electrical noise in the recording chain might lead to a more “hidden“ signal content. In other recordings an animal might be much closer to the microphone or hydrophone than in others. However, we still want to compare those signals and interpret the meaning of certain call structures in a signal etc.

Furthermore, it is important to agree among researchers on how to derive these features. Throughout the analysis you calculate the features in a certain way. It is necessary to be transparent in how you extracted those measure. Moreover, if everyone follows the same guidelines on how to report these features it is very easy for you to later compare and evaluate values that you arrived on with values that others have published. In the following paragraphs in section will introduce the measures peak frequency $f_p$, centroid frequency $f_c$ and bandwidth $bw$ and follow the standard set by Madsen and Wahlberg in [7].

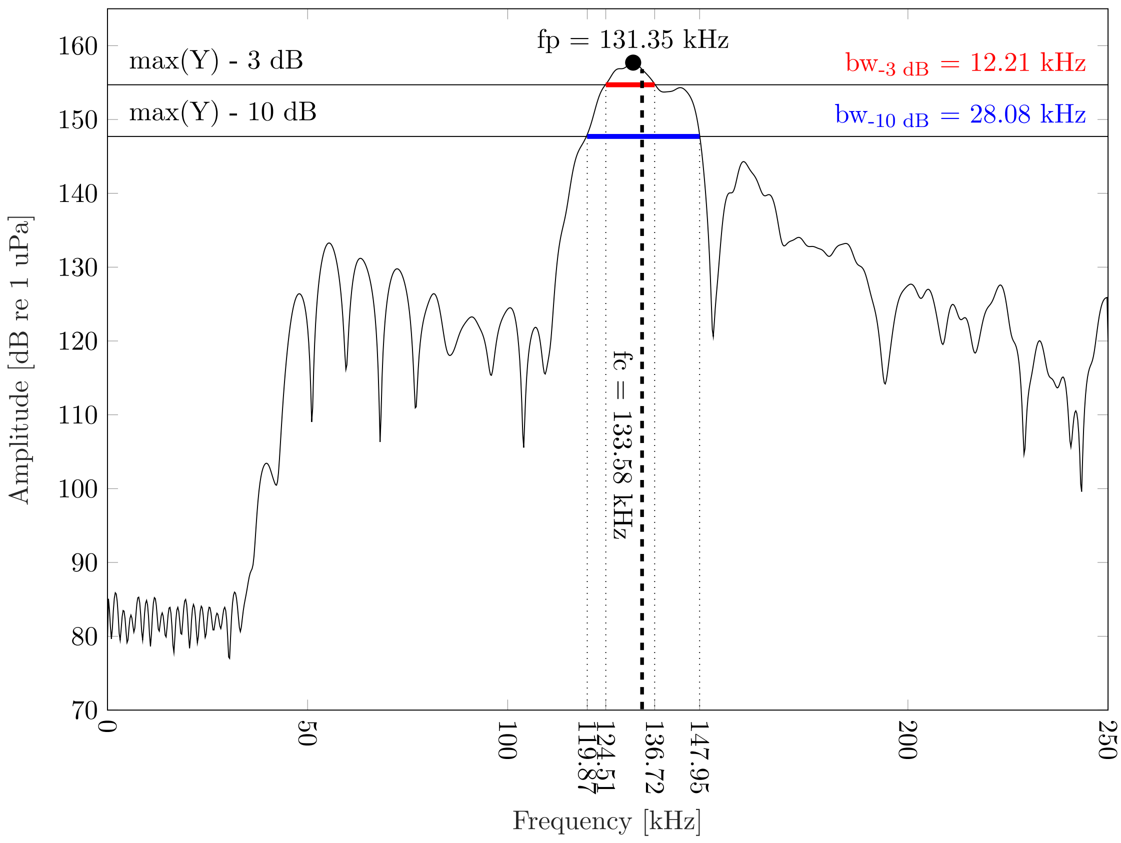


Figure 3.7: Peak frequency, centroid frequency and bandwidth for the calibrated frequency spectrum of `eigil.wav` using a $n = 2048$ FFT.

## Peak Frequency

The peak frequency $f_p$ is defined as the frequency with highest amplitude in the frequency spectrum $Y(f)$.

$$f_p = Y^{-1}(max\{Y(f)\}) \tag{3.9}$$

Extracting the peak frequency in Matlab can be done for example by making use of the `find()` function or the extended output of the `max()` function. In either way we paste the output in the vector $f_x$ (our calibrated x-axis, see Section 3.5) representing the evenly spaced the frequencies from 0 to $\frac{f_s}{2}$ .

```
% If Y is the one-sided, calibrated frequency spectrum
% and fx the vector representing the evenly spaced
% frequencies from 0 to fs/2

% then the peak frequency is given by
fp = fx(find(Y  == max(Y),1));
% or
```

```
[Y_max, i] = max(Y);
fp = fx(i);
```

Algorithm 3.4: Extracting the peak frequency

### Frequency Bandwidth

Another important feature describing the frequency spectrum is the bandwidth $bw$. The bandwidth is usually reported either as $-3$ dB or $-10$ dB below the peak frequency $f_p$ (or centroid frequency $f_c$, see paragraph below) and is denoted respectively as $bw_{-3\text{ dB}}$ or $bw_{-10\text{ dB}}$. However it is equally necessary when reporting the bandwidth to mention the band boundaries or the center frequency ($f_p$ or $f_c$) around which the band spans. The band boundaries are called minimum frequency $f_{\min}$ and maximum frequency $f_{\max}$ and should not be confused with the frequency of maximum energy (which is the peak frequency $f_p$). The bandwidth can easily be extracted as shown in Eq. 3.10 - 3.12, when $Y(f)$ is the calibrated frequency spectrum and $a = [-3\text{ dB}, -10\text{ dB}]$.

$$f_{\min} = \min\left\{Y^{-1}\left(\max\left\{Y(f)\right\} - a\right)\right\} \tag{3.10}$$

$$f_{\max} = \max\left\{Y^{-1}\left(\max\left\{Y(f)\right\} - a\right)\right\} \tag{3.11}$$

$$bw_{-a\text{ dB}} = f_{\max} - f_{\min} \tag{3.12}$$

Additionally, to stress the importance of correctly reporting the bandwidth; there is a big difference whether an animal's acoustic bandwidth spans over one or several octaves. If we have two signals with the same bandwidth whereas one has a lower center frequency than the other, it will translate into more available octaves for the signal producer at the lower centroid.

$$n_{\text{Octaves}} = \log_2\left(\frac{f_2}{f_1}\right) \tag{3.13}$$

To reiterate, lets look at Eq. 3.13. We have two signals $y_1$, $y_2$ and their FT $Y_1$, $Y_2$, which both have the same acoustic bandwidth of $bw_1 = bw_2 = 60\ kHz$. $Y_1$ has a centroid frequency of $f_{1,c} = 50\ kHz$. $Y_2$ on the other hand spans the frequency band from $f_{\min} = 120\ kHz$ to $f_{\max} = 180\ kHz$. It is clear to see that the number of available octaves $n_{1,\text{Octaves}}$ for the emitter of signal $y_1$ is greater than $n_{2,\text{Octaves}}$ for $y_2$.

$$n_{1,\text{Octaves}} = \log_2\left(\frac{f_{1,c} + \frac{bw}{2}}{f_{1,c} - \frac{bw}{2}}\right) = \log_2\left(\frac{80}{20}\right) = 2 \tag{3.14}$$

$$n_{2,\text{Octaves}} = \log_2\left(\frac{f_{2,\max}}{f_{2,\min}}\right) = \log_2\left(\frac{180}{120}\right) = 0.5850 \tag{3.15}$$

In Matlab we can derive the bandwidth again by making use of `find()`.

```
% If Y is the one-sided, calibrated frequency spectrum
% and fx the vector representing the evenly spaced
% frequencies from 0 to fs/2

% Lets define the -10 dB threshold
a = 10;

% then minimum and maximum frequency are given as
f_max = fx(max(find(Y>max(Y)-a)));
f_min = fx(min(find(Y>max(Y)-a)));
```

```
% and the bandwidth follows
bw = f_max - f_min;
```

Algorithm 3.5: Extracting the bandwidth of the frequency spectrum

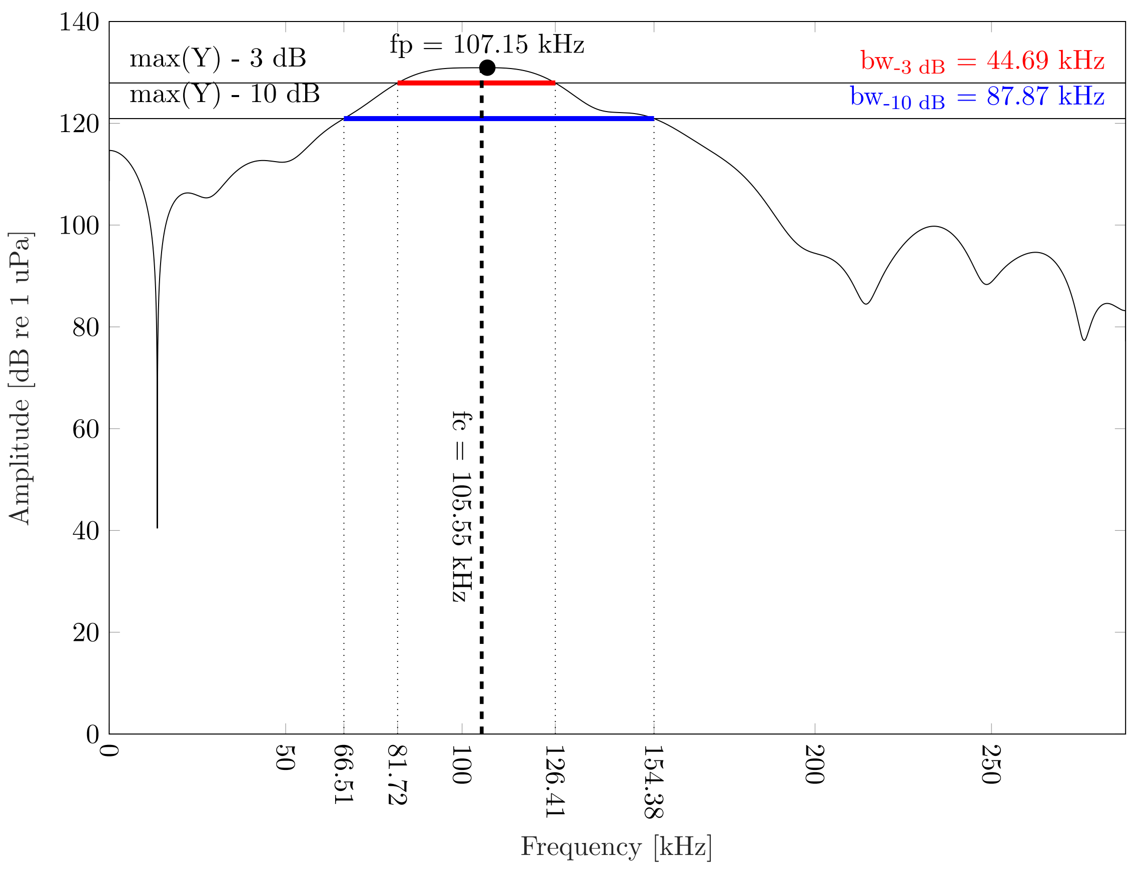


Figure 3.8: Peak frequency, centroid frequency and bandwidth for the calibrated frequency spectrum of a dolphin click using a $n = 2048$ FFT.

## Centroid Frequency

Almost similar to the peak frequency $f_p$, the centroid frequency $f_c$ returns information on the distribution of energy concentration across the frequency spectrum. $f_c$ marks the point which divides the FT in two halves of equal energy. If $y$ is a discrete signal with $N$ number of samples and $Y$ is its FT then the centroid frequency $f_c$ is derived as shown in Eq. 3.16. Keep in mind, that $Y$ has to be in linear units and <u>not</u> log-transformed.

$$f_c = \frac{\sum\limits_{i=0}^{N-1} f(i) \cdot Y^2(i)}{\sum\limits_{i=0}^{N-1} Y^2(i)} , \tag{3.16}$$

where $\sum_{i=0}^{N-1} f(i) \cdot Y^2(i)$ describes the weighted sum of the energy content of the frequency bins $f(i)$. $Y^2(i)$ is the square of the magnitude of non-log-transformed FT of $y$.

The centroid frequency $f_c$ is especially useful when handling broadband signals. In narrow band signals the centroid frequency $f_c$ and the peak frequency $f_p$ are very similar. For the porpoise click (a narrow band signal, Figure 3.7) is $f_p = 131.35\ kHz$ and $f_c = 133.58$

$kHz$. Furthermore, narrow band signals have a well defined frequency peak and because it easy to derive we report the peak frequency $f_p$.

Now lets look at a more broad band signal such as a dolphin click (see Figure 3.8). If we were to rely solely only $f_p$ then we would not represent the signal correctly. We would run into problems as the peak frequency in a broad band signal is not well defined. The peak frequency $f_p$ in the case of Figure 3.8 could be located anywhere on the plateau between roughly 90 and 115 $kHz$. Moreover, when comparing several broad band signals of the same emitter, $f_p$ would vary lot across recordings because of not having a distinctive peak compared to a more narrow band signal such as porpoises (e.g. Figure 3.7).

Even though the complexity of Eq. 3.16 in Matlab, it is just a single line of code to calculate the centroid frequency $f_c$ as shown in Algorithm 3.6.

```
% If Y is the one−sided, calibrated frequency spectrum
% and fx the vector representing the evenly spaced
% frequencies from 0 to fs/2

% then the cendroid frequency follows as
fc = sum(fx'.*Y.^2)./sum(Y.^2);

% REMEMBER: Y is not in dB but in linear units!
```

Algorithm 3.6: Extracting the centroid frequency of the frequency spectrum

### 3.6.1 Exercise

1. Calculate peak and centroid frequencies as well as the $-3$ dB and -10 dB bandwidth for the `eigil.wav` signal and visualize it appropriately.

## 3.7 Windowing

When we transform a signal from time to frequency domain we use a DFT. However, as mentioned in the beginning of this chapter, the DFT assumes periodicity and continuity. Periodicity and continuity means in this context that the presented signal (our input signal) repeats itself and stretches in time from $-\infty$ until $+\infty$. You can think of the signal being a segment which is then copied and connected to the end of previous segment until infinity.

This representation however has problems when there is no “smooth“ connection between each of the segments. So far we did not have any problems because all the signals we looked at were nicely tapered off at both ends towards zero. However, for example in a recording or noise this is often not the case. Such signals, when “connected“ to perform a FT, have discontinuity between the end of one segment and the start of another. When the FFT then computes the spectrum of a such a signal, the algorithm tries to model the “jump“ and introduces energy-rich frequency bins which are actually non-existent in our original signal.

Lets assume we modify our `eigil.wav` in such a way that the first 30 samples are equal to 150 $Pa$. Figure 3.9 shows the modified signal vector plotted as a dashed, red graph. The signal now has an initial jump of 150 $Pa$ before returning to zero where at roughly 150 $ms$ the porpoise click starts. Now when we compute the FT of this signal using a FFT with the resolution of $n = 2048$ samples we get the spectrum plotted as a red graph in Figure 3.10. Immediately it is apparent that the frequency band from $0 - 50$ $kHz$ has a higher magnitude, resulting in a biased spectrum. In-turn this means that through the

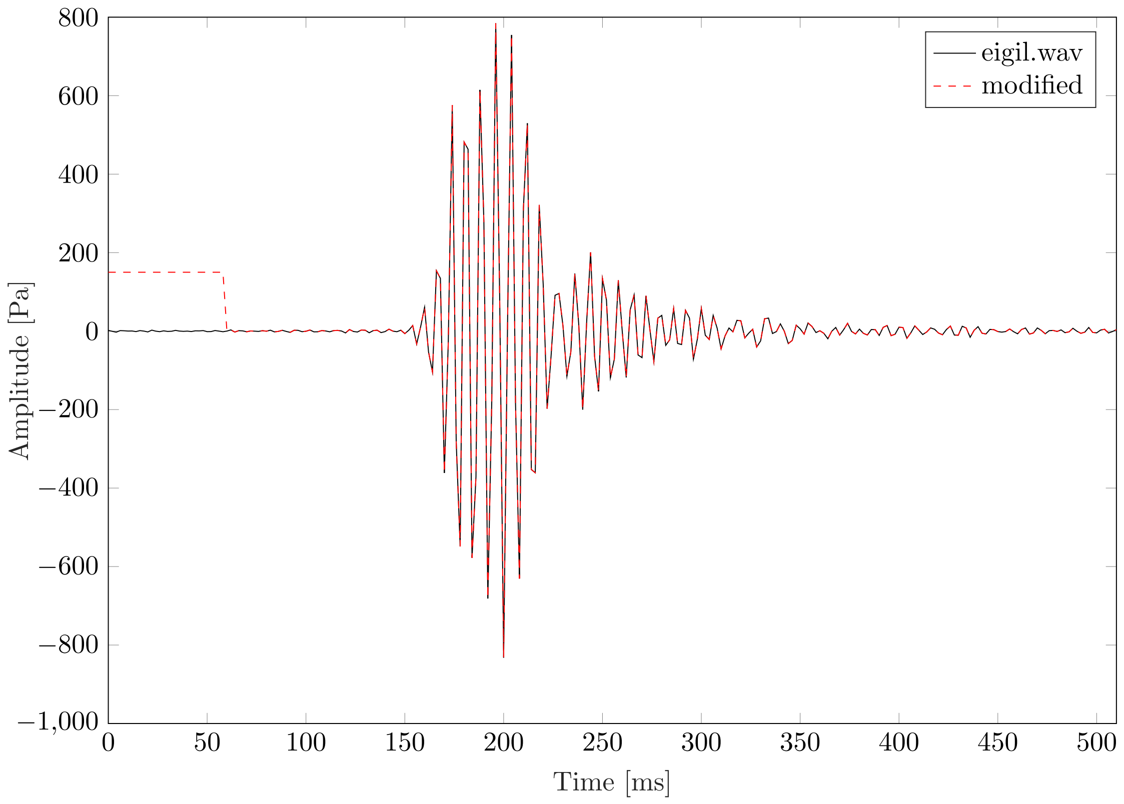


Figure 3.9: Oscillogram of `eigil.wav` and a modified version of it. For the modified version (red, dashed graph) the first 30 samples were set to be $y(i) = 150\ Pa$.

computation of the FT, the algorithm has added sinusoids in with frequencies between 0 and 50 $kHz$ in order to compensate the initial jump in the time domain.

To reduce this discontinuity (no jumps between start and end of a signal) we can use so-called window functions. A Window function tapers off both ends of the signal (or signal segment) to minimize any "jump". There is a wide variety of window functions available which have different properties and frequency responses. Figure 3.11 shows some of the more common window functions such as the Hann window, Hamming window, Blackman-Harris window and Kaiser window.

Applying a window function means an element-wise multiplication of the signal with the window function. Using a a $n_w = 256$ length Hamming window and multiplying it with the modified `eigil.wav` signal of Figure 3.9 results in the green spectrum plotted in Figure 3.10. If we compare the "real" (black), the modified (red) and the windowed (green) spectrum in Figure 3.10 we can see that the magnitude of the frequency band between 0 and 40 $kHz$ is reduced. Obviously the frequency bins from 0 to 45 $kHz$ are still bias and wrong but that is due to the severity of the modification of the `eigil.wav`.

Using and applying a window before computing the FT in Matlab is straight forward and shown in Algortihm 3.7. The `.*` operation results in an element-wise multiplication of the signal `y` with the window vector `hamming(length(y))`. Afterwards we ensure energy conservation and end with the one-sided frequency spectrum `Y`.

```
% If y is a calibrated signal sampled at fs
% then we can compute a one−sided FFT using a Hamming window as

nfft = 2048;
```

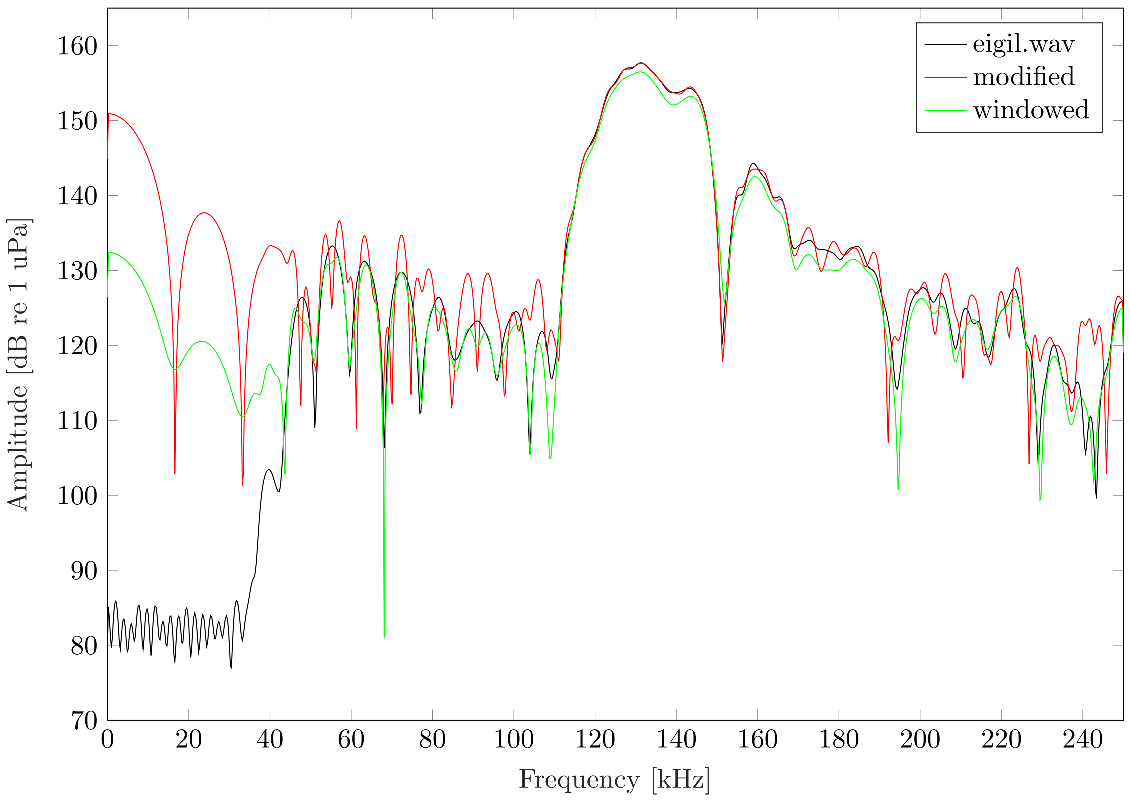


Figure 3.10: Frequency spectrum of `eigil.wav` and its modified version (see respective oscillograms in Figure 3.9). For the modified version the first 30 samples of the signal were set to be $y(i) = 150\ Pa\ (i = 0 \ldots 29)$. The FFT is computed using a $n = 2048$ sample resolution. The red colored plot displays the FT of the modified version whereas the green plot shows the spectrum after multiplying the modified signal with a $n_w = 256$ length Hamming window function. As mentioned before, the DFT assumes periodicity and continuity for the signal. For the modified version this is not the case and therefore non-existent frequencies get introduced into the spectrum. Using a window function can reduce the impact (and therefore magnitude) of through the computation introduced frequencies without affecting the rest of the signal and its frequency bins. The measures of Section 3.6 are the same for the modified, windowed variant. However, if we had not used a window function (red plot) perhaps the energy-rich frequency band of $0 - 17\ kHz$ might have had an impact when deriving the frequency measures.

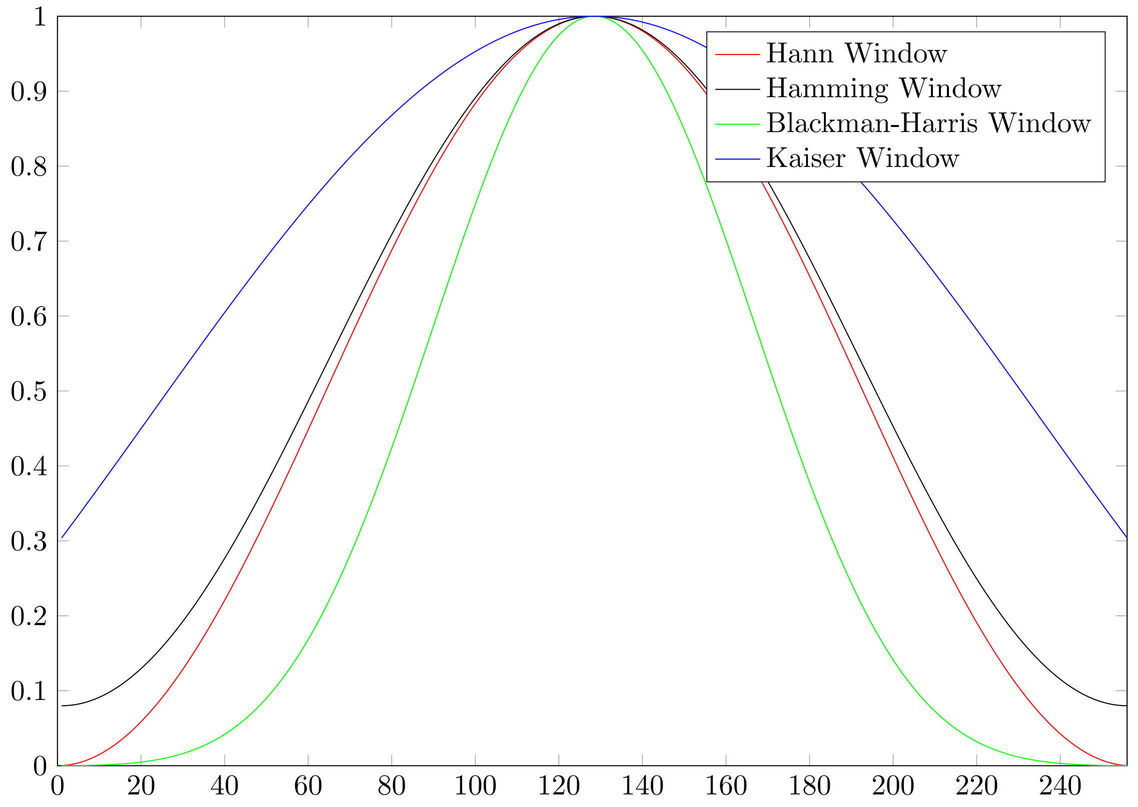


Figure 3.11: Plots of different window functions in the time domain: Hann window, Hamming window, Blackman-Harris window and Kaiser window with the length of $n = 256$ samples and in case of the Kaiser window with $\beta = 2.5$.

```

Y = abs(fft(y.*hamming(length(y)), nfft)/length(y));
Y = 2*Y(1:length(Y)/2+1);
Y(1) = Y(1)/2;
Y(end) = Y(end)/2;
```

Algorithm 3.7: Applying a Hamming window to the input signal before computing the FFT.

Lastly it is important to point out the impact of applying and not applying a window function to our signal. Lets assume we want derive the measures introduced Section 3.6 to describe our signals frequency behaviour. We can see that in case of the modified signal if we had not used a window function (red plot) perhaps the energy-rich frequency band of $0 - 17$ $kHz$ might have had an impact when deriving the frequency measures. This is especially critical when we have scripts deriving measures for us without looking at the frequency plot. It can easily happen that "discontinuities" in the time domain result in wrong (or biased) measures for the frequency domain.

Which window to use depends on the application. In many cases the differences in windowing on the spectrum are very subtle (except for using the "rectangular window"[13]). The drawback with non-rectangular windowing is that it is slightly affecting the actual signal you want to analyse. In the example displayed in Figure 3.10 we can see that the total amplitude of the peak frequency $f_p$ differs as well as the centroid frequency $f_c$ would be higher than in the "original modified" version. To conclude: windowing should be used

[13] A "rectangular window" is a vector containing only ones over the whole window length.

with care to not cause large modification of the signal before calculating the spectrum and deriving various frequency measures.

### 3.7.1 Exercise

1. Make a spectrum of the signal `tonal.wav` both with and without a Hann window (in Matlab `hanning(a)` to create a Hann window vector with the length `a`) and notice how they differ. Also apply a window over the longer “1kHz“ signal. What is the effect on windowing this one before calculating a spectrum?

## 3.8 Spectral Resolution

It has been mentioned in various sections in Chapter 3. The DFT is a periodic, continuous function which repeats itself with a “period“ $f_s$. The continuous DFT is sampled at $\frac{n}{2}$ where $n =$ `nfft` specifies the number of available samples for the FFT representing the points running from 0 to $\frac{f_s}{2}$. Each period of the DFT contains a double-sided spectrum which is converted to a one-sided spectrum[14].

Adding more points to the DFT by increasing the number of samples $n$ will result in a higher resolution for the FT. In case the signal length $N$ is smaller than the required $n$ for the FFT then the algorithm resolves this by adding zeros at the end of the signal in the time domain to reach $N = n$. This process is called *zero-padding*. The DFT size $n =$ `nfft` is closely related to the resolution of your spectrum. The more points, the finer details can be resolved. We may define the resolution of a spectrum as $\frac{f_s}{n}$. The resolution defines the frequency bin width. By increasing $n$ the resolution $\frac{f_s}{n}$ becomes smaller (the bin width is smaller) and therefore less frequency amplitudes get integrated and combined in each bin.

### 3.8.1 Exercise

1. Plot spectra for `1kHz.wav` and `white.wav` (which contains random noise) using $n = [2^8, 2^{10}, 2^{12}, 2^{16}]$ FFT (Hint: the function `subplot(m,n,a)` might be helpful.). Notice that for `1kHz.wav` and `white.wav`, you add spectral resolution by adding more samples; for Eigil, there is no a big difference. For the transient Eigil, there is little new information added by zero-padding; for the continuous and random signals, more information is constantly added by increasing n, the number of samples for the DFT (until the end of the file is reach and zero padding takes over).

## 3.9 Spectral Units

The amplitude of each spectral component is changing depending on the number of samples $n$ used in the FFT. The finer the frequency bins on the frequency axis are, the less intensity is funneled into each bin (when integrating over the whole spectrum, the total amount of energy should always be the same and equal the total amount of energy in the signal). Therefore, the magnitude of the spectrum is only interesting to know if you also know which frequency resolution (sampling rate $f_s$ and FFT size $n$) was used to create the spectrum. Usually both parameters, $f_s$ and $n$, should be reported together with the spectrum. Otherwise an interpretation and comparison with other spectra is not possible.

There are ways to standardize the amplitude scale of a spectrum. One obvious way is to say we do not report the sound intensity within each frequency bin, but we report the average energy per Herz in each frequency bin. This means we need to divide the intensity

[14]Remember to conserve the energy across the DFT by multiplying with the factor 2 when transforming a double-sided into a one-sided spectrum.

in each bin with the bin width $b_W = \frac{f_s}{n}$. Remember that the original decibel scale is $\mathrm{dB} = 20\log\left(\frac{I}{I_0}\right)$. By dividing the intensity $I$ with the bin width $b_W$ we get

$$\left[\frac{\mathrm{dB}}{Hz}\right] = 10\log_{10}\left(\frac{\frac{I}{b_W}}{I_0}\right) \tag{3.17}$$

$$= 10\log_{10}\left(\frac{I}{I_0}\right) - 10\log_{10}(b_W) \tag{3.18}$$

$$= 20\log_{10}\left(\frac{p}{p_0}\right) - 10\log_{10}\left(\frac{f_s}{n}\right) \tag{3.19}$$

The unit for this becomes a bit cryptic. Because the intensity $I$ is proportional to the pressure $p$ squared ($I \propto p^2$) you can write the unit as $\left[\mathrm{dB\ re\ 1}\ \frac{\mu Pa^2}{Hz}\right] = \left[\mathrm{dB\ re\ 1}\ \frac{\mu Pa}{\sqrt{Hz}}\right] = \left[\mathrm{dB\ re\ 1}\ \frac{\mu Pa}{Hz^{1/2}}\right] = \left[\mathrm{dB\ re\ 1}\ \mu Pa^2 Hz^{-1}\right]$. As terrible as the unit may look – it means your spectrum was measured in so-called spectral units which can immediately be compared to other measurements in the literature[15].

There are some caveats: windowing actually takes some power away from the signal (if it is e.g. white noise). This must be compensated for when estimating the spectral level (for most windows it is of the order of 3 dB). Also, the spectral analysis above assumes the signal is either random or a transient – if it contains tonal components, these have according to Fourier infinitely narrow bandwidths, so those component should not be smeared out over the frequency bins when estimating their spectral levels. This makes it really complicated analyzing spectral levels, especially if they are containing random noise, transients and tones (which many do). Suffices to say that you should always treat the levels of spectral analysis with care.

## 3.10 Interpolation

Interpolation describes a mathematical technique that increases the total number of samples of a digitized signal by "interpreting" between the discrete samples of a signal. For this, new samples are constructed in-between the samples to better represent the original analog signal. We already covered in Section 3.8 the increase in spectral resolution. Mainly interpolation is used to increase the accuracy of the signal in the time domain[16]. Increasing the resolution in the time domain can be very helpful when for example dealing with acoustic localization. Here we are especially interested in determining the exact point in time when a certain part of a signal arrived at your microphone.

In Matlab there are various ways of re-sampling the a digital signal to increase the total number of samples. For example the functions `resample()` or `interp1()`. However, using these functions will changed FT of the signal! To perform an interpolation that maintains the spectral properties of the original signal we can make use of the FFT and *zero-padding*. If we add zeros to the signal in the frequency domain and compute the inverse-FFT, then we get a signal in the time domain that is equal to the original $n$ samples of the FFT plus $k$ amount of points that we added with *zero-padding*.

As we said in the beginning of this chapter you can transform a signal from time to frequency domain and back. Because we did not change anything but adding frequency bins with zero energy and then transforming the spectrum back from frequency to time domain

[15] Given that the same window length and function was used etc.

[16] A smoother representation of the time domain signal means we are closer to the analog signal that was originally recorded.

(inverse FT) the new samples fall neatly between the original samples, thus interpolate our signal. However, note that by adding zero-energy frequency bins we increase the sample rate by $k$ frequency bins. To compute the new sample rate $f_s^*$ we can use Eq. 3.20. The idea behind it stems from the assumption that the duration $t$ of the signal before and after the interpolation stays the same. The duration is computed as $t = \frac{N}{f_s}$ with $N$ number of samples. Thus we can solve the equation $t = \frac{N}{f_s} = \frac{N+k}{f_s^*}$ and derive our new sample rate $f_s^*$ (Eq. 3.20). Keep in mind that with change of the sample rate the Nyquist-frequency also changes accordingly.

$$f_s^* = \left(1 + \frac{k}{N}\right) \cdot f_s \tag{3.20}$$

Now lets look at the the function `interpft()` in Matlab who does exactly what we just described in the paragraph above. `interpft()` interpolates an input vector or matrix through the use of zero-padding in the frequency domain. Alg. 3.8 is a snippet of the implemented function and shows exactly the process described above. The input signal is `x` having `m` number of samples. `ny` is the total amount of samples we want to have in our signal once done interpolating. In Line 9 we compute the FFT of our input signal `x` and in Line 11 we add a vector of zeros with $k =$ `ny - m` to our FT. Finally in Line 16 the signal is transformed back to the time domain. Keep in mind we have to scale our interpolated signal `y` at the end before returning it with the fraction $\frac{N+k}{N}$ (Line 18, Alg. 3.8). The reason for it is the way Matlab has implemented the `fft()`-function. When transforming to the frequency domain we divide the FT by the number of samples in our input vector. Hence when returning to the time domain we need to multiply with the number of samples. Because the number of samples increased when returning to the time domain we multiply with $\frac{N+k}{N} = \frac{N^*}{N}$ where $N^* = N+k$ is the total number of samples in the interpolated signal.

```
 1  %  If necessary, increase ny by an integer multiple to make ny > m.
 2  if ny > m
 3      incr = 1;
 4  else
 5      if ny==0, y=[]; return, end
 6      incr = floor(m/ny) + 1;
 7      ny = incr*ny;
 8  end
 9  a = fft(x,[],1);
10  nyqst = ceil((m+1)/2);
11  b = [a(1:nyqst,:) ; zeros(ny-m,n) ; a(nyqst+1:m,:)];
12  if rem(m,2) == 0
13      b(nyqst,:) = b(nyqst,:)/2;
14      b(nyqst+ny-m,:) = b(nyqst,:);
15  end
16  y = ifft(b,[],1);
17  if isreal(x), y = real(y); end
18  y = y * ny / m;
19  y = y(1:incr:ny,:);  % Skip over extra points when oldny <= m.
20
21  if nargin==2
22      y = reshape(y,[ones(1,nshifts) size(y,1) siz(2:end)]);
23  else
24      y = reshape(y,[size(y,1) siz(2:end)]);
25      y = ipermute(y,perm);
```

Algorithm 3.8: Snippet of the Matlab function `interpft()`

### 3.10.1 Exercise

1. Implement your own function that interpolates a signal in the time domain using its FT and *zero-padding*. Show your function in action by using `eigil.wav` as input and plot the result of a 10 times interpolated `eigil.wav` in the same figure as the original.

## 3.11 Filters

Another very important application in the frequency analysis is filtering. Usually when we record a signal, there is unwanted noise that we would like to remove during the analysis. A classic way to do this is to apply a filter that removes the unwanted frequency components. In “real analog life“ this is made by constructing electric circuits consisting of a set of resistor, capacitances and coils. An electronic part like this will let some frequencies pass through and filter others out.

Figure 3.12 plots the magnitude spectrum of the most common types of filters. In Figure 3.12a are all important terms visualized. Frequencies in the *pass band* are let through whereas all frequencies in the *stop band* are filtered out. The frequency at which the filter starts to drop is called the cutoff frequency $\omega_c$. It is usually defined at the frequency where the response has dropped by $-3$ dB relative the *pass band*.

The most common types of filters are:

- A low-pass filter (Figure 3.12a) allows lower frequencies to pass whereas frequencies higher than the cutoff frequency $\omega_c$ get filtered.
- The opposite is a high-pass filter (Figure 3.12b). Here, lower frequencies get filtered out and higher frequencies can pass.
- There are a lot of interesting applications for band-pass filters (Figure 3.12c). Band-pass filters have a frequency band that can pass whereas every frequency outside of this band gets blocked.
- Exactly opposite to band-pass filters filters works the band-reject filter (Figure 3.12d). Frequencies in the reject-band get filtered out and every other frequency is let through.

The *transition band* is part of the filter design. There are different designs where the magnitude of the frequency respond drops sharper than in others. In the ideal world, we would like the filter to be infinitely steep between pass and stop bands. Unfortunately, this is not possible. Any real filter has a certain slope from the pass to stop band (the *transition band*). This slope is usually defined by its steepness (for example -6 dB/octave). Sometimes filters are referred to by the number of different poles. The more poles, the steeper the slope: an $N$-pole filter has a slope of $N \cdot -6$ dB/octave. An existing transition band is not the only problem real filters have. It is also difficult to design a filter with a completely flat frequency respond in the pass band and no response at all in the stop band.

Different filter applications have different properties. The choice is again a trade-off that has to be considered when having thought carefully of what you would like to achieve. The Butterworth filter for example (called `butter()` in Matlab) has a very flat pass band and a relatively flat stop band. However, in some situation other filters can be a better

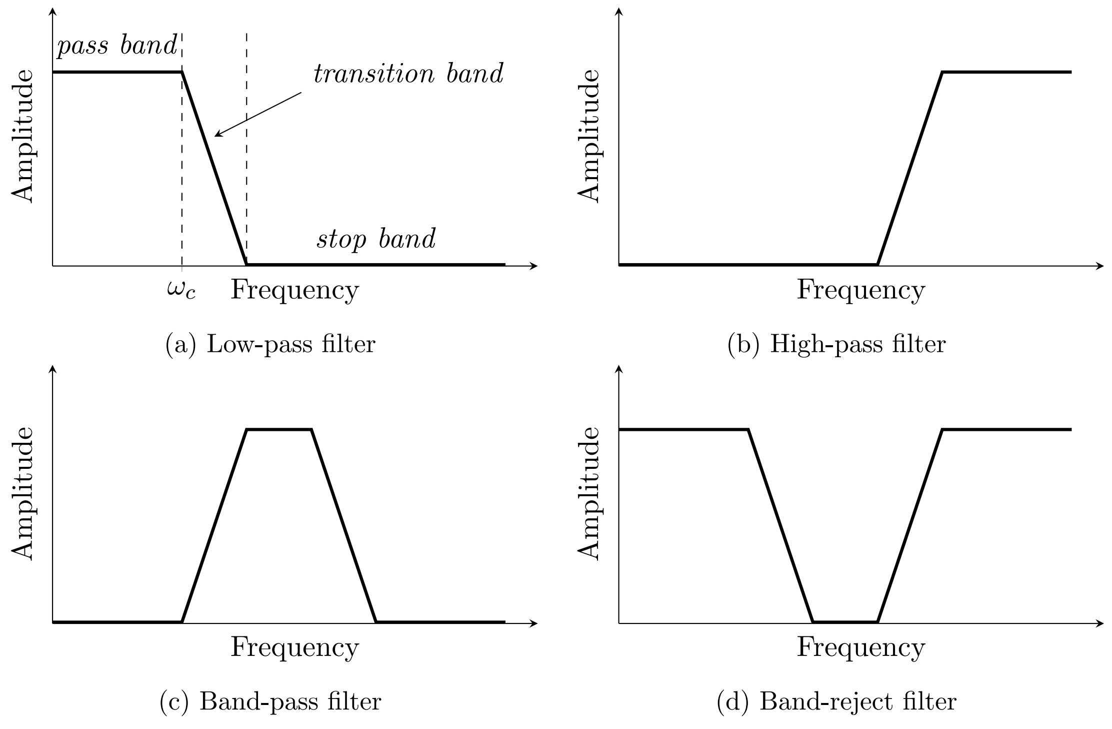


Figure 3.12: Four common types of frequency responses when applying a filter, modified from [1, p. 268].

for choice certain applications. Again, it is a trade-off – a steep filter means you have a well-fined border between the pass and stop band, but the steeper the filter, the more affects the phase of the signal in the transition from pass to stop band. Using the Matlab function `filtfilt()` can be a solution. `filtfilt()` filters the input signal digitally with zero-phase distortion. This is done by processing the input data $y$ in the forward and reverse direction. After filtering the data in the forward direction, `filtfilt()` reverses the filtered sequence and runs it back through the filter. Still, one has to be careful when designing an appropriate filter and the design strongly depends on the signal, the noise and the desired effect.

In the digital domain, the signal is represented by vector of numbers. Using a digital filter is done by arithmetic's (i.e. “plus-minus-times-divide“) meaning the use of filter components (which are also actually a vector of numbers) applied to the original signal. A very simple example is

```
y = y — mean(y);
```

The mean is an arithmetic operation finding the average value of all samples in the input signal $y$. By subtracting the mean from the signal we have actually created a high-pass filter, filtering out the frequency component 0 $Hz$ (the average value of the signal). In Matlab, you can design a filter by first defining two vectors, usually named $a$ and $b$. If you want to design a Butterworth 4-pole low-pass filter with cutoff frequency $\omega_c = 700\ Hz$, you write: `[b, a] = butter(4,700/(fs/2))`. In Algorithmn 3.9 there are some examples for low, high and band pass filters in Matlab. More filters can be found in the Matlab documentation.

```
% If y is a signal that is sampled with the sample rate fs
% then we can design different Butterworth filters as:

```

```
% low-pass filter with cutoff at 700 Hz:
[b, a] = butter(4,700/(fs/2));

% high-pass filter with cutoff at 700 Hz:
[b, a] = butter(4,700/(fs/2),'high');

% band-pass filter from 700 to 6000 Hz:
[b, a] = butter(4,[700 6000]/(fs/2));

% NOTE: frequencies are always given normalized with
% the Nyquist rate (divided with fs/2)

% to filter the signal use:
y_filt = filter(b,a,y);

% or using filtfilt() for zero-phase distortion
y_filt = filtfilt(b,a,y);
```

Algorithm 3.9: Filter design and application in Matlab

### 3.11.1 Exercise

1. Import `white.wav` (which contains random noise) into Matlab and make a plot where you have filtered the time series and spectrum using a 4-pole Butterworth low-pass filter and a 4-pole Butterworth high-pass filter, both with a cutoff frequency of 1 $kHz$.

## 3.12 Spectrograms – Visualization of the Frequency Domain in Time

A common way to display the frequency domain over time are so-called spectrograms. To generate spectrograms the audio signal is segmented using window functions (see Section 3.7) with a window length of $n_w$. For each of these segments, the frequency spectrum is computed using a FFT with the resolution $n = n_{\mathrm{FFT}}$. After placing these spectra side by side they are converted into color scale. The spectrogram usually plots time on the x-axis and frequency on the y-axis. The color scale (z-axis) represents the audio signal's frequency bin magnitude.

This allows us to view graphically how the frequency changes with time within the signal. Nevertheless, there is a trade-off between frequency resolution and time resolution. Frequency resolution favours longer segments, whereas for higher time resolution one should choose a shorter window length. Using a spectrogram we can see the frequency modulation which Myotis daubentonii is using for their echolocation calls (see Figure 3.13).

Matlab has a build-in `spectrogram()`-function that takes care of computing FFTs for the windowed segments. You can execute the function with various parameters (even without any additional parameters) but it works best if you define the optimal parameters yourself. An example for a spectrogram plot can be found in Algorithm 3.10. The signal $y$ is imported as a row or column vector. The window function and length as well as the window overlap is defined. Additionally you can set the resolution of the FFT. The last parameters `fs` and `'yaxis'` are optional. `'yaxis'` defines the y-axis as the axis displaying the frequency. Finally by setting `fs` as the fifth parameter the spectrogram is not normalized but displays the frequencies up to $\frac{f_s}{2}$.

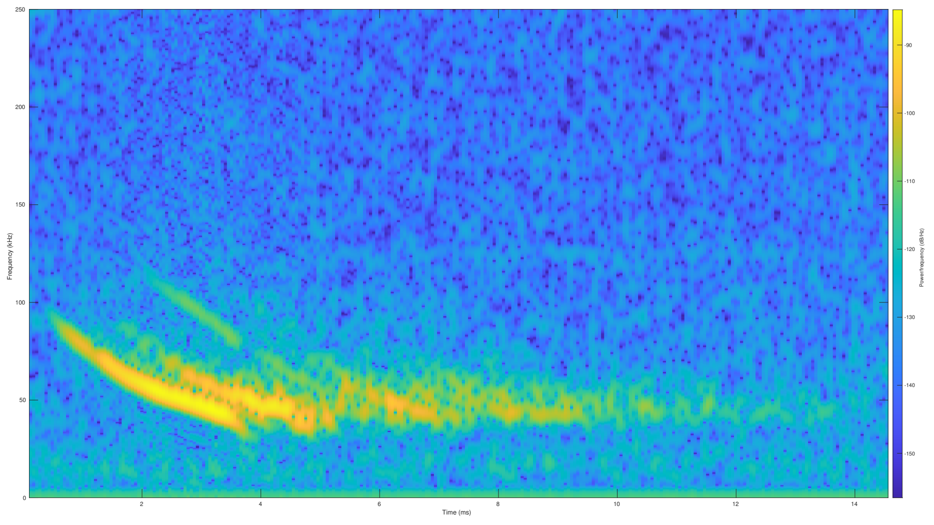


Figure 3.13: Spectrogram of a bat call using a Hamming window with the length of $n_w = 2^7 = 128$ Samples, 80% window overlap and an FFT size of $n_{\text{FFT}} = 2^{16} = 65536$. The sampling frequency of this recording is $f_s = 500\ kHz$.

```
% If y is a signal that is sampled with the sample rate fs
% then the spectrogram can be computed and plotted as
window_length = 2^7;
window_function = hamming(window_length);
overlap = .8;
nfft = 2^16;
spectrogram(y,window_function,round(window_length*overlap,0),nfft,fs,'yaxis'
    );
```

Algorithm 3.10: Plotting the spectrogram of a signal $y$

# List of Figures

# Algorithms

# List of Tables

# Bibliography

[1] S. Smith, *The Scientist & Engineer's Guide to Digital Signal Processing*, 1st. California Technical Pub, 1997, ISBN: 978-0-966-01763-2.

[2] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex fourier series," *Mathematics of computation*, vol. 19, no. 90, pp. 297–301, 1965.

[3] S. Broughton and K. Bryan, *Discrete Fourier Analysis and Wavelets : Applications to Signal and Image Processing.* Wiley, 2018, vol. Second edition, ISBN: 9781119258223. [Online]. Available: http://www.redi-bw.de/db/ebsco.php/search.ebscohost.com/login.aspx%3fdirect%3dtrue%26db%3dnlebk%26AN%3d1736760%26site%3dehost-live.

[4] K. Thyagarajan, *Introduction to Digital Signal Processing Using MATLAB with Application to Digital Communications* (SpringerLink). Springer, 2019, ISBN: 9783319760292. [Online]. Available: http://swbplus.bsz-bw.de/bsz505936747cov.htmhttps://doi.org/10.1007/978-3-319-76029-2.

[5] Ø. Ryan, *Linear Algebra, Signal Processing, and Wavelets - A Unified Approach : Python Version* (Springer Undergraduate Texts in Mathematics and TechnologySpringer-Link). Springer, 2019, ISBN: 9783030029401. [Online]. Available: https://doi.org/10.1007/978-3-030-02940-1%20;%20http://dx.doi.org/10.1007/978-3-030-02940-1.

[6] L. D. (https://dsp.stackexchange.com/a/26931), *What is a frequency bin?* Signal Processing Stack Exchange, URL:https://dsp.stackexchange.com/a/26931, accessed: 2020-01-15). eprint: https://dsp.stackexchange.com/a/26931. [Online]. Available: https://dsp.stackexchange.com/a/26931.

[7] P. Madsen and M. Wahlberg, "Recording and quantification of ultrasonic echolocation clicks from free-ranging toothed whales," *Deep Sea Research Part I: Oceanographic Research Papers*, vol. 54, no. 8, pp. 1421–1444, 2007, ISSN: 0967-0637. DOI: https://doi.org/10.1016/j.dsr.2007.04.020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0967063707001124.