

# Use R!

## Series editors

Robert Gentleman   Kurt Hornik   Giovanni Parmigiani

More information about this series at <http://www.springer.com/series/6991>

Jérôme Sueur

# Sound Analysis and Synthesis with R

 Springer

Jérôme Sueur  
Muséum National d'Histoire naturelle  
Paris, France

**Electronic Supplementary Material** The online version of this article (<https://doi.org/10.1007/978-3-319-77647-7>) contains supplementary material, which is available to authorized users.

ISSN 2197-5736

ISSN 2197-5744 (electronic)

Use R!

ISBN 978-3-319-77645-3

ISBN 978-3-319-77647-7 (eBook)

<https://doi.org/10.1007/978-3-319-77647-7>

Library of Congress Control Number: 2018939906

© Springer International Publishing AG, part of Springer Nature 2018

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, express or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Printed on acid-free paper

This Springer imprint is published by the registered company Springer International Publishing AG part of Springer Nature.

The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

*Un livre sans histoires ni paroles  
mais écrit avec passion pour*

*Chloé  
Julia  
r  
o  
l  
i  
n  
e*

# Preface

Sound is virtually always around us, everywhere, all the time. This morning, my day started on a rather unpleasant one: that of the repeated buzzing of my alarm clock. The night had been quiet despite the purr of the central heating, some motorbikes racing down the street, and blackbirds singing in one of our garden's trees at dawn. Now that I go down to the kitchen I can hear my clothes rub against my body and the wooden steps crack under my feet. As soon as I move, I realize that I myself generate sound. Quickly the house wakes up in an explosion of surrounding sounds seeping from the flush, the kettle, the toaster, the fridge, ventilation, and other domestic appliances. Music plays on the radio, but it does not cover the call of a hungry cat and family voices that soon invest and dominate in the acoustic space. My working day is a long, and sometimes exhausting, suite of sounds: metallic train screeches, mobile phone ringtones, office babbles, siren blares, street work roar, and radio tunes but also some amazing tropical sounds I recorded in a remote forest that I play back on my computer to escape that city soundscape. Sound is ubiquitous. It constantly reaches my body, being absorbed or bounced back, received and processed through my ears. But my body is also a sound source. My heart, my blood, my breath, my bones, and my vocal chords generate sound. I am sound in a world of sounds. No air, no life, no sound.

The soundscape I go through any given workday is mainly a city soundscape with very little enjoyable sound. Most of this sound can be considered as noise, which is actually sound conveying no information or overlapping other meaningful sounds. Nice soundscapes are certainly to be found in nature, in the middle of a dark forest or in the depth of an even darker ocean. Wildlife sound can be a bird song, a frog call, an insect hummer, a deer grunting, the exploding sound of a small pistol shrimp, or the amazing whistle of a giant whale. We often refer to the extraordinary diversity of life forms and colours, but life diversity is also to be found in animal vocalizations. Animals can produce rhythmic or continuous, pure-tone or polyphonic, harmonic or dysharmornic, synchronized or cacophonous sounds. Animals can almost play any instrument in any orchestra. Whatever their properties, the sounds emanating from animals are never exactly the same from one song bout to another one, from one individual to another one, and from one species to another one. The variety of animal

sound is so high that audio robots that can identify and interpret a human voice and chat with you on the phone can hardly discriminate the sound of a dolphin from a whale's. This animal acoustic diversity may be a challenge for sound analysis and synthesis, but more importantly, they are a living treasure that has to be enjoyed and preserved.

Animal sound variety is the matter of bioacoustics and ecoacoustics, two closely related life sciences disciplines. As a bioacoustician or ecoacoustician I often have to face the naive but essential question about my research: "Come on, Jérôme, what's your job all about? Is there really a point in listening to cicadas?". Addressing this question is almost the same issue as wondering why we need to name insects and flowers, scrutinize the sky to discover new stars, analyse the old century playwrights style, or understand the physics of a golf ball. Such fundamental research participates in the world's knowledge and bioacoustics. Ecoacoustics are no exception. It is essential to describe and to understand the patterns and processing that determine natural acoustic environments. To me, it is as important a thing to know how a pigeon call is produced as it is to know how a financial index is computed. However, it would be unfair to say that bioacoustics and ecoacoustics have no application in our daily lives. Next time you fly, think that the engines of the plane you are comfortably seated in will not fail after sucking up flying birds owing to the loudspeakers at the end of the runway that play specific alarm sounds and scare them away. These specific sounds were designed by Thierry Aubin, a renowned bioacoustician.

One of the most important soundmark of my city working day is the subtle noise of my fingers on the computer keyboard. As soon as I have settled administration, teaching, curation, and supervision duties I open R and play with. But why does R has such an important role in my professional life?

I have been reluctant to programming for ages. As a schoolboy and later as a university student I always disliked programming courses as I was confused with FOR, WHILE, IF, THEN, DO, and other mysterious instructions. I surely have always been a software user for work, but I never thought that one day, I would have written a command rather than just clicked on a mouse. It actually took me a long time to get into R and, eventually, to love R. I was introduced to R by a colleague of mine, Michel Baylac, who is an expert in morphometrics. This was at a morning lab coffee break and here follows the discussion we had some ten years ago. Consider that the original dialogue was in French:

Jérôme: Michel, how did you do your elliptic Fourier analysis in your last paper?

I could not find any statistical software that does it.

Michel: I used R.

Jérôme: Sorry?

Michel: I programmed the EFA with R.

Jérôme: Air? I do not know this software. What is it?

Michel: Well it is a programming language deriving from S.

Jérôme: Ace? I do not know that one either. So you can do your own analysis.  
Sounds great!

Michel: It is. And it is free. No licence to purchase.

A few coffee breaks and some explanations about software names later, I successfully installed R and got started with it but I quickly gave up so used I was to graphical user interface. Fortunately, Michel gave an R-based statistics Master course a few weeks later so I went back to school and followed his instructions to run R multivariate analysis. But I had no data to analyse at that time and a few months later I had forgotten almost everything. A year after, I joined my future wife Caroline Simonis, who is a co-author of *seewave*, for a second session of Michel's course and I joined her again for a course on linear models with R organized by Emmanuel Paradis, who wrote the best-seller *R for beginners* and the wonderful phylogenetics *ape* package. I was probably more motivated to be with Caroline than to learn R, but Caroline understood the interest of R much faster than I did and she talked me into starting to write more code rather than simply  $\text{lm}(Y \sim X)$ . This time I had data to look into at hand and free time to play with R. I can still remember that winter evening when I could plot my very first pure-tone spectrogram. The image was incomplete and inaccurate, but to me it was a wonderful and shiny plot that motivated me to keep on with R. I was so amazed at being able to run such an analysis by myself knowing all the production steps perfectly that in the following weeks I could not stop writing basic sound-analysis dedicated functions. I was lucky to be greatly helped by Caroline and also by Thierry Aubin, my mentor in bioacoustics and the author of the *Syntana* software that inspired the main *seewave* functions.

R definitely changed my research. I was no more limited to the utilities provided by prohibitive closed source softwares that my department could not afford. I could do almost everything by myself: I could draw fieldwork observation maps, read, analyse, and change my sound samples. I could collect qualitative and quantitative data, run batch processes, apply multivariate analysis, plot high-quality graphics, and eventually produce a paper combining R and  $\text{\LaTeX}$  in a nice layout. The most important thing was that I was not only a user but also a designer. I was able to create, imagine, and share new tools with others.

I'm telling this very personal story as I think this could be the future of anyone who is a bit afraid of software programming. Learning R is not that difficult with a little help—which a nice woman like Caroline can do but this is not absolutely necessary. Just keep in mind that it is worth an effort and that the reward will be tremendous.

This book was written for students who are interested in bioacoustics and ecoacoustics, but I really hope that it can help anyone who is willing to dive into



the fantastic area of acoustics and into the endless land of  $\mathbb{R}$ . A book and a cake are not that different: they both require time and energy to be made, but they are consumed apace. I really hope this book has a nice taste, and I wish you very nice reading and programming nights!

Paris, France

Jérôme Sueur

# Acknowledgements

I would like first to thank Andreas Wessel who, some years ago, initiated this book by whispering my name in Lars Koerner's ear.

I am deeply indebted to Michel Baylac and Emmanuel Paradis for having taught me R at several occasions. Without their help, I would still be using spreadsheet applications to compute an arithmetic mean.

I was extremely lucky to be supervised during my research training by Thierry Aubin and Daniel Robert, my mentors for ever in bioacoustics.

The core of this book is the `seewave` package which was initiated with Thierry Aubin and Caroline Simonis. `seewave` has been growing up during the last 11 years thanks to the contribution of Ethan C. Brown, Marion Depraetere, Camille Desjonquères, François Fabianek, Amandine Gasc, Eric Kasten, Stefanie LaZerte, Laurent Lellouch, Jonathan Lees, Jean Marchal, Sandrine Pavoine, David Pinaud, Alicia Stotz, Luis J. Villanueva-Rivera, Zev Ross, Carl G. Witthoft, and Hristo Zhivomirov. `seewave` and related analyses have been improving thanks to ideas, comments, checks, or bug reports by Andrey Anikin, Charlotte Curé, Stéphane Dray, Denis Dupeyron, Almo Farina, Arnold Fertin, Kurt Hornik, Emiliano A. Laca, Nadia Pieretti, Daniel Ridley-Ellis, Jesse Ross, Pavel Senin, and Arvind Sowmyan. `seewave` has been maintained on CRAN thanks to the crucial help of Kurt Hornik, Uwe Ligges, Brian Ripley, and Simon Urbanek, all members of the R core team.

My motivation to complete this book mostly came from the imaginary students I had in mind when coding and writing. I also receive significant support from the students or junior researchers I was lucky to supervise their research in bioacoustics or ecoacoustics: Pablo Bolaños, Marion Depraetere, Camille Desjonquères, Manon Ducrettet, Amandine Gasc, Alexandre Kempf, Laurent Lellouch, Diego Llusia, Felipe Moreno, Alexandra Rodriguez, Alexandra Stotz, and Juan Sebastian Ulloa.

This book is acoustically and visually illustrated thanks to several people who shared sounds and/or images: Laurent Arthur, Thierry Aubin, Renaud Boistel, David Cartmell, Emmanuel Delfosse, Amandine Gasc, Joël Gilbert, Jean-François Julien, Diego Llusia, Ladislav Nagy, Christian Roesti, Frédéric Sèbe, and Andreas Trepte.

Readers of beta versions of the manuscript kindly took their precious time to check and improve the text: Andrey Anikin, Thierry Aubin, Stéphane Dray, Amandine Gasc, Jonathan Katz, Laurent Lellouch, Nathan Merchant, Benoît Obled, and Loïc Ponger.

# Contents

<b>1</b>	<b>Introduction</b> .....	1
1.1	Sound as a Science Material.....	1
1.2	Layout .....	2
1.3	Convention for Notation and Code.....	5
1.4	Book Compilation .....	6
<b>2</b>	<b>What Is Sound?</b> .....	7
2.1	A Debate Under a Dangerous Tree.....	7
2.2	Sound as a Mechanical Wave.....	9
2.2.1	Air Particle Motion.....	9
2.2.2	Air Pressure Variation.....	10
2.2.3	Amplitude .....	12
2.2.4	Phase .....	20
2.2.5	Duration .....	20
2.2.6	Frequency.....	21
2.2.7	Writing Sound with a Simple Equation .....	25
2.2.8	Amplitude and Frequency Modulations.....	26
2.3	Sound as a Time Series .....	29
2.4	Sound as a Digital Object.....	30
2.4.1	Sampling.....	30
2.4.2	Quantization .....	30
2.4.3	Issues in Sampling and Quantization.....	32
2.4.4	File Format .....	33
2.5	Sound as a Support of Information.....	34
<b>3</b>	<b>What Is R?</b> .....	37
3.1	A Brief Introduction to an Ocean of Tools .....	37
3.2	How to Get R.....	39
3.3	Do You Speak R? .....	40
3.3.1	Where Am I? .....	40
3.3.2	Objects.....	41
3.3.3	Operators .....	46

3.3.4	Functions .....	47
3.3.5	Controlling Flow .....	50
3.3.6	Manipulating Objects .....	54
3.3.7	Vectorization and Recycling .....	62
3.3.8	Handling Character Strings .....	64
3.3.9	Drawing a Graphic .....	65
3.3.10	Scripting .....	73
3.3.11	Calling External Software .....	74
3.4	R and Sound .....	75
3.4.1	To Use or Not to Use R for Sound Analysis? .....	75
3.4.2	Main Packages .....	76
3.4.3	How to Install seewave .....	79
<b>4</b>	<b>Playing with Sound</b> .....	<b>81</b>
4.1	Object Classes .....	81
4.1.1	vector, matrix, data.frame Classes .....	81
4.1.2	ts and mts Classes .....	82
4.1.3	audioSample Class of the Package audio .....	85
4.1.4	sound Class of the Package phonTools .....	86
4.1.5	Wave Class of the Package tunerR .....	87
4.2	How to Read (Load) a Sound .....	90
4.2.1	.wav Files .....	90
4.2.2	.mp3 Files .....	92
4.2.3	From .mp3 to .wav Files .....	93
4.2.4	.flac Files .....	94
4.2.5	Local Files .....	94
4.2.6	Online Files .....	95
4.2.7	Song Meter <sup>®</sup> Files .....	99
4.3	How to Listen to a Sound .....	100
4.3.1	With the Package audio .....	101
4.3.2	With the Package phonTools .....	105
4.3.3	With the Package tunerR .....	105
4.3.4	With the Package seewave .....	106
4.4	How to Record a Sound .....	107
4.5	How to Write (Save) a Sound .....	108
4.6	Tuning R .....	110
<b>5</b>	<b>Display of the Wave</b> .....	<b>111</b>
5.1	Oscillogram .....	112
5.1.1	Simple Oscillogram .....	112
5.1.2	Axes .....	114
5.1.3	Colors .....	116
5.1.4	Decoration and Annotation .....	119
5.1.5	Zoom In .....	121
5.1.6	A Bit of Interactivity .....	123
5.1.7	Multiple Oscillogram .....	123

- 5.2 Amplitude Envelope ..... 125
  - 5.2.1 Principle ..... 125
  - 5.2.2 In Practice with seewave ..... 128
  - 5.2.3 Smoothing ..... 129
  - 5.2.4 In Practice with phonTools ..... 136
- 5.3 Combining Oscillogram and Envelope ..... 138
- 6 Edition ..... 139**
  - 6.1 Resampling ..... 139
  - 6.2 Channels Managing ..... 142
  - 6.3 Manipulating Sound Sections ..... 146
    - 6.3.1 Extract ..... 146
    - 6.3.2 Delete ..... 149
    - 6.3.3 Paste ..... 150
    - 6.3.4 Repeat ..... 154
    - 6.3.5 Reverse ..... 155
  - 6.4 Removing and Inserting Silence Sections ..... 155
  - 6.5 Changing Amplitude ..... 159
    - 6.5.1 Offset ..... 159
    - 6.5.2 Amplitude Level ..... 161
    - 6.5.3 Fade-In and Fade-Out ..... 163
- 7 Amplitude Parametrization ..... 167**
  - 7.1 Linear Relative Scale ..... 167
  - 7.2 Logarithm Relative Scale ..... 173
    - 7.2.1 Signal-to-Noise Ratio ..... 173
    - 7.2.2 dB Weightings ..... 174
    - 7.2.3 dB Arithmetic ..... 175
    - 7.2.4 Sound Attenuation Through Spreading Losses ..... 177
  - 7.3 Absolute Scale ..... 181
- 8 Time-Amplitude Parametrization ..... 185**
  - 8.1 What and How to Measure? ..... 185
  - 8.2 Manual Measurements ..... 186
  - 8.3 Automatic Measurements ..... 191
    - 8.3.1 The Cicada Case ..... 193
    - 8.3.2 The Frog Case ..... 198
  - 8.4 Amplitude Modulation Analysis ..... 205
    - 8.4.1 The Cicada Case ..... 205
    - 8.4.2 The Frog Case ..... 209
- 9 Introduction to Frequency Analysis: The Fourier Transformation ... 213**
  - 9.1 From Time to Frequency and Back ..... 213
  - 9.2 Fourier Series ..... 214
    - 9.2.1 Periodicity ..... 214
    - 9.2.2 Trigonometric Fourier Series ..... 217

9.2.3	Compact Fourier Series .....	219
9.2.4	Exponential Fourier Series .....	222
9.3	Fourier Transform .....	224
9.4	Frequency Scales .....	229
9.4.1	Bark and Mel Scales .....	229
9.4.2	Musical Scale .....	231
9.5	Amplitude Scales .....	235
9.6	Fourier Windows .....	236
9.7	Inverse Fourier Transform .....	240
9.8	Cepstrum .....	241
<b>10</b>	<b>Frequency, Quefrequency, and Phase in Practice .....</b>	<b>247</b>
10.1	Frequency Spectrum .....	247
10.1.1	Functions of the Package <code>tuneR</code> .....	248
10.1.2	Functions of the Package <code>seewave</code> .....	249
10.1.3	Identification of Peaks .....	265
10.1.4	Profile Analysis .....	275
10.1.5	Symbolic Analysis .....	286
10.1.6	Parametrization .....	293
10.2	Quefrequency Cepstrum .....	302
10.3	Phase Portrait .....	303
<b>11</b>	<b>Spectrographic Visualization .....</b>	<b>309</b>
11.1	Short-Time Fourier Transform .....	309
11.1.1	Principle .....	309
11.1.2	The Uncertainty Principle .....	312
11.2	Computation and Display of the Spectrogram .....	315
11.3	Function of the Package <code>signal</code> .....	319
11.4	Functions of the Package <code>tuneR</code> .....	320
11.5	Function of the Package <code>phonTools</code> .....	324
11.6	Function of the Package <code>soundgen</code> .....	325
11.7	Functions of the Package <code>seewave</code> .....	326
11.7.1	2D Spectrogram .....	326
11.7.2	External Computing of the Short-Time Fourier Transform .....	349
11.7.3	Inverse Short-Time Fourier Transform .....	351
11.8	Measurements and Annotations on the Spectrogram .....	353
11.8.1	Simple Measure .....	353
11.8.2	Fancy Measure and Annotation .....	353
11.8.3	Automatic Parametrization .....	358
11.9	Complex Display and Printing .....	362
11.9.1	Multi-Spectrogram Graphic .....	362
11.9.2	Printing in a File .....	364
11.9.3	Long Spectrogram Graphic .....	365
11.10	Dynamic Spectrogram .....	366
11.11	Movie .....	368

- 11.12 Waterfall Display ..... 370
- 11.13 3D Spectrogram..... 372
- 11.14 Mean Spectrum ..... 375
- 11.15 Soundscape Spectrum ..... 377
- 12 Mel-Frequency Cepstral and Linear Predictive Coefficients** ..... 381
- 12.1 Mel-Frequency Cepstral Coefficients (MFCCs)..... 381
  - 12.1.1 Theory ..... 381
  - 12.1.2 Practice ..... 385
- 12.2 Linear Predictive Coefficients (LPCs) ..... 394
  - 12.2.1 Theory ..... 394
  - 12.2.2 Practice ..... 395
- 13 Frequency and Energy Tracking**..... 399
- 13.1 Frequency Tracking..... 400
  - 13.1.1 Dominant Frequency ..... 400
  - 13.1.2 Fundamental Frequency ..... 405
  - 13.1.3 Formants ..... 416
  - 13.1.4 Instantaneous Frequency ..... 418
- 13.2 Energy Tracking ..... 427
- 14 Frequency Filters** ..... 435
- 14.1 Preemphasis Filter ..... 440
- 14.2 Comb Filter ..... 443
- 14.3 Butterworth Filter ..... 445
- 14.4 Wave Smoothing Filter ..... 449
- 14.5 DFT and STDFT Filter ..... 451
  - 14.5.1 Principle ..... 451
  - 14.5.2 `ffilter()` Function..... 451
  - 14.5.3 Examples ..... 452
- 14.6 FIR Filter ..... 455
  - 14.6.1 Principle ..... 455
  - 14.6.2 `fir()` Function ..... 455
  - 14.6.3 Examples ..... 456
  - 14.6.4 Setting the Transfer Function ..... 459
- 15 Other Modifications** ..... 465
- 15.1 Setting the Amplitude Envelope ..... 465
- 15.2 Echoes and Reverberation ..... 467
- 15.3 Amplitude Filtering ..... 468
- 15.4 Modifications Using the ISTDFT ..... 470
- 15.5 Modifications Using the Hilbert Transform ..... 474
- 16 Indices for Ecoacoustics**..... 479
- 16.1  $\alpha$  Indices..... 482
  - 16.1.1 Functions ..... 482
  - 16.1.2 Batch Processing: How to Obtain a List of  $\alpha$  Indices for a Set of Sounds ..... 492



- 16.2  $\beta$  Indices..... 494
  - 16.2.1 Functions ..... 494
  - 16.2.2 Batch Processing: How to Obtain and Analyze a Matrix of  $\beta$  Indices..... 505
- 17 Comparison and Automatic Detection ..... 521**
  - 17.1 Cross-Correlation ..... 521
  - 17.2 Frequency Coherence ..... 528
  - 17.3 Dynamic Time Warping ..... 530
  - 17.4 Automatic Identification ..... 534
    - 17.4.1 Principle ..... 534
    - 17.4.2 In Practice with the Package `monitor` ..... 538
- 18 Synthesis ..... 555**
  - 18.1 Silence ..... 555
  - 18.2 Noise..... 557
  - 18.3 Non-sinusoidal Sound ..... 558
    - 18.3.1 Pulse Wave ..... 558
    - 18.3.2 Square Wave ..... 560
    - 18.3.3 Triangle and Sawtooth Waves ..... 561
  - 18.4 Sinusoidal Sound: Additive Synthesis ..... 564
    - 18.4.1 Principle ..... 564
    - 18.4.2 In Practice with `tuneR`..... 566
    - 18.4.3 In Practice with `seewave` ..... 569
  - 18.5 Sinusoidal Sound: Modulation Synthesis ..... 574
    - 18.5.1 Principle ..... 574
    - 18.5.2 In Practice with `signal` ..... 574
    - 18.5.3 In Practice with `seewave` ..... 574
    - 18.5.4 Examples ..... 578
  - 18.6 Tonal Synthesis ..... 598
    - 18.6.1 Principle ..... 598
    - 18.6.2 In Practice with `seewave` ..... 598
    - 18.6.3 Examples ..... 600
  - 18.7 Speech ..... 604
    - 18.7.1 Solution with the Package `phonTools`..... 604
    - 18.7.2 Solution with the Package `soundgen` ..... 605
- A List of R Functions..... 611**
- B Sound Samples ..... 619**
- References..... 627**
- Index ..... 633**

# Acronyms

A	Maximum amplitude
AM	Amplitude modulation
DC	Direct current voltage
DFT	Discrete Fourier transform
E	Energy (J)
F	Force (N)
FFT	Fast Fourier transform
FM	Frequency modulation (Hz)
FT	Fourier transform
I	Intensity ( $\text{W m}^{-2}$ )
IDFT	Inverse discrete Fourier transform
IFT	Inverse Fourier transform
ISTDFT	Inverse discrete short-time Fourier transform
ISTFT	Inverse short-time Fourier transform
P	Power (W)
Q	Quality factor
RMS	Root-mean-square
S	Area ( $\text{m}^2$ )
SIL	Sound intensity level (dB)
SPL	Sound pressure level (dB)
STDFT	Short-time discrete Fourier transform
STFT	Short-time Fourier transform
SVL	Sound velocity level (dB)
T	Period (s)
TKEO	Teager-Kaiser energy operator
Z	Acoustic impedance ( $\text{N s m}^{-3}$ )
ZCR	Zero crossing rate
<i>a</i>	Instantaneous amplitude
<i>a</i>	Acceleration ( $\text{m s}^{-2}$ )
<i>c</i>	Sound celerity ( $\text{m s}^{-1}$ )
<i>d</i>	Duration (s)

$f$	Ordinary frequency (Hz)
$f_c$	Carrier frequency (Hz)
$f_d$	Dominant frequency (Hz)
$f_r$	Resonant frequency (Hz)
$f_s$	Sampling frequency (Hz)
$f_N$	Nyquist frequency (Hz)
$p$	Pressure (Pa)
$p_0$	Reference air pressure at 0 s.l.m ( $1.1013 \times 10^5$ Pa)
$p_{\text{ref}}$	Human auditory threshold in air ( $2 \times 10^{-5}$ Pa = $20 \mu\text{Pa}$ = 0 dB)
$v$	Particle velocity ( $\text{m s}^{-1}$ )
$t$	Time (s)
$\omega$	Angular frequency (rad)
$\lambda$	Wavelength (m)
$\rho$	Volumetric mass density ( $\text{kg m}^{-3}$ )
$\varphi$	Angular phase (rad)

# List of Figures

Fig. 2.1      Sound emanating from a tuning fork. The two tuning fork hinges are represented from above with two blue squares. Their vibrations generate a sound that propagates as a longitudinal wave in air. Sound is represented along a single direction with an alternation of air rarefaction ( $r$ ) and compression ( $c$ ) with a wavelength  $\lambda$ . A simple framed elastic membrane at a fixed position in the  $(x, y)$  space vibrates sympathetically with sound. This is an oversimplified representation of sound propagation around a tuning fork; see Russell et al. (2013) and Russell (2000) for a complete description ..... 8

Fig. 2.2      Sound pressure ( $p$ ) and amplitude variations. The sound was recorded at time  $t = 0$  and at distance  $d_1$  from the source with a  $-\pi \div 4$  rad or  $-45^\circ$  phase shift  $\varphi$ . The bottom  $x$ -axis shows the time  $t$  in seconds, the top  $x$ -axis shows the distance in meter and the  $y$ -axis is the instantaneous pressure  $p$  in Pascal. In this ideal case, air pressure oscillates cyclically as a sinusoidal function around  $p_0$ . The gray rectangle delimits one cycle. In the time domain, the interval between two compression peaks is the period ( $T$ ). In the space dimension, the distance between two compression peaks is the wavelength ( $\lambda$ ). The red vertical bars on the top  $x$ -axis represent the density of air particles. Low and high air particle density corresponds to air rarefaction ( $r$ ) and compression ( $c$ ), respectively ..... 11

Fig. 2.3      Amplitude ( $A$ ). The three main amplitude quantities of a sound: the instantaneous, the maximum, the peak-to-peak, and the average (root-mean-square, rms) amplitude ..... 13

Fig. 2.4      dB scale. Top: relation between the ratio of two pressures and the corresponding dB value. Doubling the pressure is equivalent to an addition of 6 dB. Bottom: from pressure in

Pa to sound pressure level (SPL) in dB. Values are given for every 10 dB ..... 15

Fig. 2.5 dB weighting curves. The weightings curves of dB(A), dB(B), dB(C), and dB(D) according to frequency. The code used to produce this figure is given in Sect. 7.2.2 ..... 17

Fig. 2.6 Sound attenuation for a spherical source. Curves of dB attenuation with distance due to spreading losses in a free and unbounded medium (model) and of what could be measured in the medium (measurements). The difference between the two curves due to medium absorption and scattering is named excess of attenuation (EA). The measurement curve is here still idealized as scattering effects will produce an irregular curve ..... 19

Fig. 2.7 Phase ( $\varphi$ ). Two sounds with similar amplitude and frequency but different phase. There is a  $\pi \div 4$  rad or  $45^\circ$  shift between the two waves ..... 20

Fig. 2.8 Duration ( $d$ ). Two sounds of different duration, the red sound being a third shorter than the blue one ( $d_1 = 2 \div 3 \times d_2$ ). The amplitudes of the two sounds were set to different values to allow comparison ..... 21

Fig. 2.9 Frequency ( $f$ ). Two sounds with different frequencies: the red sound has a frequency four times higher than the blue one. In other words, there are three blue cycles and twelve red cycles, or there are four red cycles for a single blue cycle. If  $t = 1$  s, then the frequency of the blue wave is 3 Hz, and the frequency of the blue wave is 12 Hz..... 22

Fig. 2.10 Harmonics. Sound made of three tones with a harmonic ratio: the fundamental ( $f_0$ ), the first harmonic ( $f_1$ ), and the second harmonic ( $f_2$ ). The light gray lines correspond to these three tones isolated..... 23

Fig. 2.11 Square (top), triangle (middle), and sawtooth (bottom) waves. These periodic functions consist of harmonics series ..... 24

Fig. 2.12 Noise (top) and Dirac pulse (bottom) waves. These functions do not produce either harmonics or inharmonics overtones ..... 25

Fig. 2.13 Amplitude and frequency modulations (AM, FM). The instantaneous amplitude (blue plain line) is modulated according to an amplitude exponential decay  $a(t)$  (black dashed line) (top) or according to a frequency exponential increase  $f(t)$  (bottom) ..... 27

Fig. 2.14 Sinusoidal amplitude modulation. Two examples of instantaneous amplitude (blue plain line) modulated according to a sinusoidal amplitude modulation  $a(t)$  (black dashed line). The frequency of the amplitude modulation  $f_{am}$  of the above example is half the one in the example

below. The amplitude depth  $m$  is 1 (or 100%) in the example above and 0.5 (or 50%) in the example below ..... 27

Fig. 2.15 Sinusoidal frequency modulation. Three examples of sinusoidal frequency modulations  $f(t)$ : a frequency modulation with a frequency of 2 and a modulation index of 50 (top), a frequency modulation of 4 with a similar modulation index of 50 (middle), and a frequency modulation of 2 with a modulation index of 100 ..... 28

Fig. 2.16 Example of a time series. The atmospheric concentrations of CO<sub>2</sub> expressed in parts per million (ppm) from 1960 to 1997. This dataset could be transformed into a sound. Data from the package `datasets` ..... 29

Fig. 2.17 Sampling. Digital sound is a discrete process along the time scale. The same wave is sampled at two different rates: the wave above is sampled four times more than the bottom wave. Each point is a sample; the line is original continuous sound ..... 31

Fig. 2.18 Quantization. Digital sound is a discrete process along the amplitude scale: a 3 bit ( $= 2^3 = 8$ ) quantization (gray bars) gives a rough representation of a continuous sine wave (blue line) ..... 31

Fig. 2.19 Aliasing on a sine wave. In blue, the original sine wave was sampled at an appropriate rate representing well the cycle period or frequency. In red, the same sine wave sampled at a too low rate generating *aliasing* at a lower wrong frequency ... 32

Fig. 2.20 Aliasing on a complex wave. The original blue wave is a complex wave including several frequency components. When sampled at an appropriate rate, the wave can be properly represented with all small amplitude changes. However, when sampled at a low rate, the main amplitude features are lost (red dots and red segments)..... 33

Fig. 2.21 Clipping. This wave was not properly acquired. The amplitude exceeds the limits of the quantization scale leading to a squared or flat waveform (arrow). Such waveform cannot be studied properly as amplitude, time, and frequency features are distorted ..... 34

Fig. 2.22 Shannon diagram of a communication as published in Shannon (1949) and Shannon and Weaver (1949) ..... 35

Fig. 2.23 Shannon diagram adapted to animal communication system. Drawn with the package `diagram` (Soetaert 2014) ..... 35

Fig. 3.1 Vectorization and recycling. This graphic uses data recycling (argument `color`) and vectorization (argument `cex`)..... 63

Fig. 3.2 Scatter plot. A simple X–Y scatter plot with the `Sepal.Length` and `Sepal.Width` variables of the dataset `iris` ..... 65

Fig. 3.3 Graphic tuning. A meaningless example of graphic changes using low-level plot functions ..... 69

Fig. 3.4 Layout plate scheme by a 5-year-old hand. The first step of composing an R graphic plate is to take a pen and piece of paper and to draw it! Colors are not necessary... ..... 70

Fig. 3.5 Layout plate scheme with `layout()`. We first prepare the layout by generating an appropriate matrix. The size of the graphic numbers is increased with the function `par()` ..... 71

Fig. 3.6 Directed network of CRAN packages dedicated to sound. The network was constructed based on the main directed relationships between CRAN packages dedicated to sound. The size, or degree, of each node corresponds to the number of connections. This highlights the central position of `tuneR` and `seewave`. Built with the package `network` (Butts 2008) and drawn with the package `GGally` (Schloerke et al. 2017)..... 77

Fig. 3.7 Flowchart of `seewave` dependencies. R packages are in rounded boxes. External tools are in framed rounded boxes. Mandatory items are labeled with a star (\*). Drawn with the package `diagram` (Soetaert 2014) ..... 79

Fig. 4.1 Sound as a time series. This is a 0.05 s sound with a carrier frequency of 440 Hz and a sampling frequency of 8000 Hz. The plot was created with the function `plot()` applied to a `ts` object..... 84

Fig. 4.2 Geographical map of Xeno-Canto recordings. The function `xcmaps()` of `warbleR` can return a map of a species recordings, here for the rufous-collared sparrow, or tico-tico, *Zonotrichia capensis*, recorded in Brazil ..... 98

Fig. 5.1 The rufous-collared sparrow *Zonotrichia capensis* also named tico-tico in Portuguese. Reproduced with the kind permission of Ladislav Nagy ..... 112

Fig. 5.2 A simple oscillogram. The waveform of the tico sound obtained with `oscillo(tico)` ..... 113

Fig. 5.3 Oscillogram with a calibrated amplitude. The default blank y-axis is tuned to display absolute values, here along a Pascal scale ..... 115

Fig. 5.4 Oscillogram axes. The axes were removed, and a time scale bar was added ..... 116

Fig. 5.5 Oscillogram colors. The colors of most graphical items can be changed to tune the oscillogram plot..... 117

Fig. 5.6	Oscillogram decoration. Example of necessary and useless annotations on an oscillogram .....	119
Fig. 5.7	Oscillogram highlight with a rectangle. The yellow background was added, thanks to the function <code>polygon()</code> .....	121
Fig. 5.8	Oscillogram time zoom in. The plate was built with four calls to the function <code>oscillo()</code> using different values for the arguments <code>from</code> and <code>to</code> .....	122
Fig. 5.9	Multi-line oscillogram. Using the argument <code>k</code> , the oscillogram is split in four sections of equal duration over four lines. The argument <code>j</code> can also be used to divide the oscillogram in columns .....	124
Fig. 5.10	Overplotting oscillograms. This figure demonstrates the overplot of two oscillograms, a noisy and a clean version of the dataset <code>tico</code> .....	125
Fig. 5.11	Absolute and analytic (or Hilbert) amplitude envelope. The figure shows a 0.05 s signal with a triangular shape sampled at 22,050 Hz. Both absolute and analytic (or Hilbert) envelopes are overplotted to show their different behavior in the following amplitude modulations .....	127
Fig. 5.12	Analytic envelope of <code>tico</code> . The envelope was obtained with the simple command <code>env(tico)</code> .....	129
Fig. 5.13	Tuning of an amplitude envelope. The envelope of <code>tico</code> was zoomed in on the second syllable, the color of the envelope was changed, and a title was added .....	130
Fig. 5.14	Sliding window. Graphical representation of a window sliding along the time axis. The sound is sampled at 22,050 Hz; the window length is made of 512 samples which is equivalent to 0.0232 s. The overlap is 0% (top), 50% (middle), and 75% (bottom). The height of the window was artificially increased for a sake of clarity .....	131
Fig. 5.15	Amplitude envelope smoothing. Example of the <code>tico</code> amplitude analytic envelope smoothed with different sliding window lengths and overlaps .....	133
Fig. 5.16	Amplitude envelope types and smoothing with a sliding average. The plate shows the shape of the <code>tico</code> envelope either as an absolute amplitude envelope ( <code>envt='abs'</code> ) or as an analytic envelope ( <code>envt='hil'</code> ) for different average sliding window lengths. The difference by subtraction between the two envelopes is also shown .....	134
Fig. 5.17	Amplitude envelope smoothing by moving sum. The envelope is smoothed by computing the sum of neighbor values within a window containing 8, 512, or 1024 samples .....	135
Fig. 5.18	Amplitude envelope smoothing with a kernel function. The envelope is smoothed by applying a kernel function parametrized with a smoothing parameter <code>m</code> .....	136



Fig. 5.19 Envelope following `powertrack()` function. The envelope of `tico` was obtained with the function `powertrack()` of `phonTools`. The envelope is obtained through a smoothing average on the square of the sound ..... 137

Fig. 5.20 Oscillogram and envelope. The analytic amplitude (or Hilbert) envelope is plotted in red over the oscillogram ..... 137

Fig. 6.1 Aliasing and downsampling. The original file (top) is a 5000 Hz pure tone sampled at 22,050 Hz. The same sound downsampled at 11,025 Hz clearly shows time and frequency artifacts (bottom) ..... 141

Fig. 6.2 Oscillogram of a stereo `Wave` object. The object `tico` was converted into a stereo `Wave` object with `stereo()` and plotted as an oscillogram with the function `oscilloST()`. The left channel is on the top and the right channel is at the bottom of the plot ..... 143

Fig. 6.3 Clicks when concatenating (pasting) waves. The concatenation of two waves with different phases might generate unwanted clicks. There is a  $3\pi \div 2$  rad or  $270^\circ$  shift between the two waves ..... 151

Fig. 6.4 Click removing by `prepComb()`. The click at the junction between `wave1` and `wave2` was removed thanks to the function `prepComb()` of the package `tuner` ..... 151

Fig. 6.5 Pasting sounds with `pastew()`. The second syllable is pasted (inserted) into `tico` at 0.6 sand the result is plotted ..... 153

Fig. 6.6 Click removing by `pastew()`. The click at the junction between `wave1` and `wave2` was removed thanks to the function argument `tjunction` of `pastew()` of the package `seewave` ..... 154

Fig. 6.7 Histogram of `tico` absolute amplitude envelope. Distribution of the absolute values (absolute amplitude envelope) of the `tico` samples. The first cell counts the numbers of samples between 0 and 1000, the vertical red bar indicates the center of the first cell at 500..... 156

Fig. 6.8 Removing silence. The figure shows the results of both `noSilence()` and `zapsilw()` functions. The first function works at start and end of the signal operating as a trim function when the second function removes every silence sections. Sections modified are highlighted with red arrows drawn with `arrows()` ..... 158

Fig. 6.9 Muting. The second syllable of `tico`, which starts at 0.6 sand stops at 0.87 s is muted by replacing original samples values with 0 values. The new silence section is highlighted with a red arrow drawn `arrows()` ..... 159

Fig. 6.10 Adding silence. Silence sections can be added with the function `addwilw()` as demonstrated here by adding 0.2 s bouts at both start and end of `tic0`. The new silence sections are highlighted with red arrows drawn with `arrows()` ..... 160

Fig. 6.11 Amplitude offset. This wave is shifted toward high amplitude values, departing from the  $p_0$  reference value ..... 160

Fig. 6.12 Fade-in and fade-out. Fade-in and fade-out are applied to the tuning fork sound with three different amplitude shapes: linear, exponential and cosine..... 165

Fig. 7.1 Attenuation due to spreading losses. The curve of attenuation due to spreading losses for a sound source of 80 dB measured at 1 m is shown up to 150 m. This curve was obtained using the function `attenuation()` ..... 178

Fig. 7.2 Signal path and calibration sequence. The recording chain goes through several stages from the initial sound source to the terminal digital file passing through processes of transduction (microphone, hydrophone, accelerometer, or other), amplification (pre-amplifier), digitization (analogue-digital converter), and file conversion (computer algorithm). The arguments of the function `PAMGuide()` are indicated below the process they are related to. The argument `Si` covers the chain from transduction to digitization. Modified from Merchant et al. (2015) ..... 181

Fig. 8.1 Pictures of soniferous animals: the Mediterranean cicada *Cicada orni* (Jérôme Sueur) and the Martinique Robber frog *Eleutherodactylus martinicensis* (reproduced with the kind permission of Renaud Boistel) ..... 186

Fig. 8.2 Calling song of *Cicada orni* saved in the dataset `orni`. The song is made by the regular repetition of five syllables or echemes (*e-i*) (first panel). Each echeme is made of about ten pulses (*p-i*) as shown here by zooming in on the third echeme (*e-3*) (second panel). The start of echeme 3 (*e-3*) can be identified clearly (third and fourth panels). The end of the echeme 3 (*e-3*) is more difficult to localize due to echoes (bottom, upward arrows with question marks)..... 187

Fig. 8.3 Automatic time measurement of the `orni` sound. The five echemes (signal) and the inter-echeme (pause) separating them are automatically detected with the function `timer()`. The Hilbert amplitude envelope (`envt="hil"`) was smoothed with a moving average (`msmooth=c(50,0)`)..... 194

Fig. 8.4 Automatic measurement of the `orni` sound with amplitude and time thresholds. The figure is the graphical

output of `timer()` with a smoothing parameter (`msmooth=c(30,0)`), an amplitude threshold (`threshold=5`), and a time threshold (`dmin=0.04`)..... 196

Fig. 8.5 Automatic measurement of the `orni` sound with a moving sum. The figure is the graphical output of `timer()` with a smoothing parameter using `sum(ssmooth=100)` and an amplitude threshold (`threshold=6`)..... 198

Fig. 8.6 Oscillogram of the frog *Eleutherodactylus martinicensis*. The recording made Renaud Boistel is a succession of 17 two-note calls of a focal recorded male, with important background sound due to other vocalizing males ..... 200

Fig. 8.7 Automatic time measurement of the frog *Eleutherodactylus martinicensis*. The 17 two-note vocalizations (signals) and the pauses separating them are automatically detected with the function `timer()`. The Hilbert amplitude envelope (`envt="hil"`) was squared (`power=2`) and smoothed with a moving average (`msmooth=c(100,90)`). The results were filtered with a 0.2s time threshold (`dmin=0.2`).... 201

Fig. 8.8 Graphical use of `timer()` results. The results returned by `timer()` are used to zoom on the first four vocalizations, to label and to frame these vocalizations ..... 202

Fig. 8.9 Comparison of manual and automatic measurements. The plot shows against time the duration the 17 vocalizations (signal) and pauses of the calling sequence of the frog *E. martinicensis* obtained manually using the argument `identify` of `oscillo()` (manual) and the estimation returned by the function `timer()` (automatic)..... 203

Fig. 8.10 Distribution of the automatic measurements according to different `timer()` settings on the 17 vocalizations (signal) and pauses of the calling sequence of the frog *Eleutherodactylus martinicensis* ..... 204

Fig. 8.11 Amplitude modulation analysis of the `orni` sound: fast amplitude modulations. The function `ama()` shows two peaks corresponding to the pulse repetition rate (0.237 kHz) and the carrier frequency (2.347 kHz) ..... 206

Fig. 8.12 Amplitude modulation analysis of the `orni` sound: slow amplitude modulations. The function `ama()` set with a large window shows a dominant peak corresponding to the echeme repetition rate (0.007 kHz) ..... 207

Fig. 8.13 Amplitude modulation analysis of the frog *Eleutherodactylus martinicensis*: fast amplitude modulations. The function `ama()` shows three peaks corresponding to the fundamental frequency of the first note (1.938 kHz), the fundamental frequency (3.141 kHz) and the beating between these two frequencies (1.219 kHz) ..... 210

Fig. 8.14 Amplitude modulation analysis of the frog *Eleutherodactylus martinicensis*: slow amplitude modulation. The function `ama()` set with a large window shows a dominant peak corresponding to the vocalization repetition rate (0.001 kHz) ..... 211

Fig. 9.1 Jean-Baptiste Joseph Fourier (1768–1830). Engraving by Jules Boilly, around 1823 (Public Domain) ..... 214

Fig. 9.2 Fourier transformation principle. Any complex waveform can be decomposed into a sum of simple waveforms. Here the top waveform with a period  $T$  is decomposed into the addition of three simple waveforms ( $n = 3$ ) related by a fundamental frequency  $f_0$  ..... 215

Fig. 9.3 A periodic waveform. The waveform, possibly a sound, is made of five repetitions of the same pattern. The waveform follows the equation  $s(t + mT) = s(t)$ , with  $T$  the period and  $m = \{1, 2, 3, 4, 5\}$  ..... 216

Fig. 9.4 Frequency decomposition and signal reconstruction. The original signal (O) is decomposed into a series of ten functions written as  $[A_n \cos(\omega_n t) + B_n \sin(\omega_n t)]$  with  $n = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$ . The last signal (R) is the reconstruction of the original signal (O) using the coefficients  $A_n$  and  $B_n$  and the angular frequencies  $\omega_n$  ..... 220

Fig. 9.5 Frequency spectrum. The frequency spectrum is a barplot of the Fourier coefficients  $C_n$  against the  $n$  angular frequency indices. The top frequency scale in Hz was manually added with the graphical function `axis()` and `mtext()` ..... 222

Fig. 9.6 Phase spectrum. The phase spectrum is a barplot of the phase coefficients  $\varphi_n$  against the  $n$  angular frequencies ..... 223

Fig. 9.7 Mirrored frequency spectrum of the FFT. The modulus of the FFT is a symmetric (mirrored) function of the angular (or regular) frequency around the Nyquist frequency  $f_N$  ..... 227

Fig. 9.8 Frequency spectrum of the FFT. This spectrum includes all the Fourier coefficients from  $C_0$  to the Nyquist frequency  $f_N$  .... 228

Fig. 9.9 Hertz ( $x$ -axis), mel (left  $y$ -axis), and Bark (right  $y$ -axis) scales. Bark and mel scale are closely related even if defined differently and evolving on different ranges ..... 230

Fig. 9.10 Frequency of Western musical notes. The frequency in Hertz and mel of the 12 Western musical notes is plotted over the first 6 octaves. The mel scale, through its logarithm properties, spaces more equally the notes than the Hertz scale along the octaves ..... 233

Fig. 9.11 Amplitude scale of the frequency spectrum. Seven examples of amplitude scales used to show a frequency

spectrum, from raw data directly returned by the FFT to linear and scaled scales and logarithmic scales based on the dB unit. A zoom between 0 and 5000 Hz was operated on the frequency axis..... 236

Fig. 9.12 FFT window shape. Shapes of the six FFT windows implemented in *seewave*. The windows includes here  $N = 512$  samples..... 237

Fig. 9.13 FFT window effects on the frequency spectrum. A 2 kHz sound lasting 0.02 s is windowed (syn. tapered) with a function that reduces errors in the frequency spectrum. Basically the rectangular window (first line of graphics) has no effect when the remaining (Bartlett, Blackman, flattop, Hamming, and Hanning) changes the shape of the spectrum. The effects are less visible on a spectrum with a linear amplitude scale (second column) than on a spectrum with a dB amplitude scale (third column)..... 239

Fig. 9.14 Cepstrum: echo detection. The original signal is a 45 Hz signal affected by an echo arriving with a delay of 0.2 s and an increase of 50% of amplitude (upper panel). Applying the complex cepstral transform returns a graphic with a quefrequency  $x$ -axis and an amplitude  $y$ -axis. A peak appears at 0.2 s (bottom  $x$ -axis scale) corresponding to 5 Hz (top  $x$ -axis scale) (bottom panel) corresponding to the echo delay..... 243

Fig. 9.15 Cepstrum of a harmonic series. The original signal is a 0.1 s harmonic series with a 440 Hz fundamental frequency and nine harmonics regularly and linearly decreasing in amplitude. The 440 Hz fundamental frequency can be seen as a regular amplitude modulation (gray area) (first panel). The spectrum is therefore made of ten frequency peaks spaced by 440 Hz (gray area) (second panel). The logarithm of the frequency spectrum shows the same profile with the same distance between peaks, but frequency peaks are compressed (third panel). The cepstrum shows a peak at a quefrequency of 0.002 s equivalent to 440 Hz (fourth panel)..... 244

Fig. 9.16 Cepstrum of an amplitude modulated signal. The original signal is a 2500 Hz pure tone signal with an amplitude modulation of 440 Hz (gray area) lasting 0.1 s (first panel). The spectrum is made of three frequency peaks, a dominant frequency peak at 2500 Hz and two lateral frequency peaks at  $2500 - 440 = 2060$  Hz and  $2500 + 440 = 2990$  Hz (gray area) (second panel). The logarithm of the frequency spectrum shows the same profile with the same distance between peaks, but frequency peaks are compressed (third panel). The cepstrum shows a peak at a quefrequency of

0.0024 s equivalent to 417 Hz, slightly departing from the 440 Hz modulation frequency (fourth panel)..... 245

Fig. 10.1 Pictures of soniferous animals: the northern lapwing *Vanellus vanellus* (reproduced with the kind permission of Andreas Trepte, <http://www.photo-natur.de>) and the Italian tree cricket *Oecanthus pellucens* (reproduced with the kind permission of Christian Roesti, <http://www.orthoptera.ch>)..... 248

Fig. 10.2 Frequency spectrum with `periodogram()` of `tuneR`. The frequency spectrum returned by `periodogram()` is a power spectral density, that is, a frequency spectrum squared and scaled by its sum ..... 249

Fig. 10.3 Frequency spectrum with `spec()` of `seewave` ..... 250

Fig. 10.4 Size of the frequency spectrum—1. The frequency spectrum is computed with `spec()` over the complete `peewit` dataset (top), on a section between 0.3 and 0.4 s (middle) and on a 512 sample window selected in the middle of the sound (bottom) ..... 253

Fig. 10.5 Size of the frequency spectrum—2. The frequency spectrum is computed with `spec()` at the center of `peewit` dataset with different DFT sizes (128, 256, 512, 1024). The spectrum is displayed with a line and points to highlight the frequency resolution ..... 255

Fig. 10.6 dB frequency spectrum. Frequency spectrum computed at the center of `peewit` with a window of 512 samples. The amplitude scale is expressed in dB in reference to a maximum value set to 0 ..... 258

Fig. 10.7 High-level plot modifications of the frequency spectrum. The main graphical parameters of `spec()` were used to change the appearance of the frequency spectrum, including its orientation ..... 259

Fig. 10.8 Decoration of the frequency spectrum. This plot results from the use of low-level plot functions—`par()`, `polygon()`, `axis()`, `grid()`, `title()`, `points()`, `rect()`, `rect()`, `box()`—to change the visual output of `spec()` ..... 259

Fig. 10.9 Multifrequency spectrum plot. Thirteen frequency spectra computed regularly along `peewit` are plotted on a single graph ..... 261

Fig. 10.10 Frequency band plot. Four displays of the function `fbands()`: ten regular frequency bands in a usual vertical orientation (top-left), ten regular frequency bands with color and orientation modifications (top-right), eight regular frequency bands defined by hand (bottom-left), and

eight frequency bands defined following music octaves (bottom-right)..... 264

Fig. 10.11 Peak detection of frequency spectrum. Plot output of the basic use of the function `fpeaks`: all peaks, here 56, even if tenuous, are detected..... 267

Fig. 10.12 Parameters for frequency spectrum peak detection. The function `fpeaks()` has four arguments to help in selecting the peaks of a frequency spectrum. The argument `amp` is an amplitude threshold working on the slopes of the peaks (top-left), the argument `freq` acts as a frequency threshold (top-right), the argument `threshold` is an overall amplitude threshold (bottom-left), and the argument `nmax` selects the most prominent  $n$  peaks (bottom-right). The illustration is based on schematized frequency spectra with frequency resolution of  $\Delta_f = 43$  Hz.  $S$  selected peak,  $NS$  nonselected peak ..... 268

Fig. 10.13 Example of frequency spectrum peak detection. Frequency peak detection is here tested on the a frequency spectrum computed at the center of the dataset `peewit`. Each setting (arguments `amp`, `freq`, `threshold`, and `nmax`) returns a different number of peaks detected..... 269

Fig. 10.14 Example of frequency spectrum peak detection with combined parameters. The figure shows peak detection on a spectrum computed for the second note of `tico` without any selection (circle), using the argument `amp` only (triangle), and the arguments `amp` and `freq` together (disk). A frequency zoom in was operated between 3.5 and 5.5 kHz ..... 270

Fig. 10.15 Local peak detection on the frequency spectrum. The peak of maximum energy is identified for specific frequency regions defined with the argument `bands` of the function `localpeaks()`. Detection over ten regular frequency regions (top-left), over 500 Hz wide regions (top-right), seven irregular regions (bottom-left), and octave-based regions (bottom-right)..... 272

Fig. 10.16 Frequency spectrum and quefrequency cepstrum of a sheep bleat. The plots were obtained with `spec()` and `ceps()`, respectively ..... 275

Fig. 10.17 Frequency spectrum of periodic signals—part 1. Pure harmonic series with a dominant fundamental frequency (top-left), harmonic series with a dominant frequency different from the fundamental frequency (top-right), inharmonic series (bottom-left) and two harmonics series mixed (bottom-right).  $f_d$ : dominant frequency.  $f_0$  and  $g_0$ : fundamental frequencies ..... 277

Fig. 10.18 Frequency spectrum of periodic signals—part 2. Pure sine wave with a DC component (top-left), pure sine wave with a sinusoidal amplitude modulation beating at  $f_{am}$  and with low ( $m=10\%$ ) modulation index (top-right), pure sine wave with a sinusoidal amplitude modulation beating at  $f_{am}$  with a maximum ( $m=100\%$ ) modulation index (middle-left), a harmonic series with a sinusoidal amplitude modulation beating at  $f_{am}$  with a maximum (100%) modulation index (middle-right), squared pure sine wave repeated at the frequency  $f_{am}$  (bottom-left), spectrum of `orni` which can be considered as a AM signal with periodic pauses. DC: direct current.  $f_c$ : carrier frequency.  $f_0$ : fundamental frequency ..... 279

Fig. 10.19 Frequency spectrum of periodic signals—part 3. 5 kHz pure sine wave linearly increasing in frequency up to 7 kHz (top-left), 5 kHz pure sine wave affected by a sinusoidal frequency modulation with  $f_{fm} = 0.5$  kHz and  $\beta = 1$  (top-right), 5 kHz pure sine wave affected by a sinusoidal frequency modulation with  $f_{fm} = 0.5$  kHz and  $\beta = 2$  (middle-left), 5 kHz pure sine wave affected by a sinusoidal frequency modulation with  $f_{fm} = 0.5$  kHz and  $\beta = 4$  (middle-right), 0.44 kHz pure sine wave affected by a sinusoidal frequency modulation with  $f_{fm} = 0.2$  kHz and  $\beta = 8$  generating sidebands reflected around 0 (bottom-left), 5 kHz pure sine wave increasing in frequency from 5 to 5.5 kHz affected by an additional sinusoidal frequency modulation with  $f_{fm} = 0.5$  kHz and  $\beta = 1$  (bottom-right)..... 282

Fig. 10.20 Theoretical frequency spectrum of a FM signal. The spectrum is obtained by applying Carson’s rule and Bessel functions to estimate the number, the frequency position, and the relative amplitude of a pure tone sound with a carrier frequency at 5000 Hz and a frequency modulation with a frequency of 500 Hz and a frequency peak deviation of 500 Hz equivalent to a modulation index  $\beta = 1$  ..... 284

Fig. 10.21 Frequency spectrum shape of brief signals. Frequency spectrum of a pure sine wave with a duration of 0.1, 0.01, 0.001, and 0.0001 s showing the appearance of side lobes that increase in importance up to a totally flat spectrum profile... 286

Fig. 10.22 Symbolic analysis. The symbolic analysis consists in translating each amplitude values into a letter according to the shape of the numeric series, here a frequency spectrum of `peewit` ..... 289



Fig. 10.23 SAX principle. The figure shows how the Z-transformed data are converted into letters in reference to a Gaussian distribution. The data come from the example given in the DIY box 10.2. The SAX series of symbols, or word, would be here *eecbaabc*. They correspond to monthly number of sun spots from 1750 to 1760. Inspired from Lin et al. (2003)..... 292

Fig. 10.24 Resonance quality factor  $Q$ . The  $Q_{-6\text{dB}}$  factor of *pellucens* was computed with a dB frequency spectrum over 1024 samples at the position 1s. Specifying axis limits allows to zoom in around the frequency peak where  $Q$  is computed ..... 296

Fig. 10.25 Statistic parameters of the frequency spectrum. The frequency spectrum of a segment of *orni* is here displayed as cumulative distribution function by setting `plot=2`..... 301

Fig. 10.26 Quefrequency cepstrum. The first rahmonic, or quefrequency peak, was estimated by using the argument `tidentify` and then highlighted with `points`. The graph has two  $x$  scale, one at the bottom expressed in time (s) and the other (top) expressed in Hz ..... 302

Fig. 10.27 Phase-space plots of pure tone and noise. The figure shows the phase-space plots obtained with `phaseplot()` (top) and `phaseplot2()` (bottom) applied to a pure tone (left) and to noise (right). Pure tone has a periodic shape when noise has an unstructured an aperiodic shape ..... 304

Fig. 10.28 Phase portrait of pipe and elephant sounds. Oscillogram, frequency spectrum, and phase portrait of (from top to bottom) a pipe sound (line 1), a “brassy” pipe sound (line 2), an elephant trumpet call (line 3), and a “brassy” elephant trumpet call (line 4) ..... 306

Fig. 11.1 Illustration of the short-time discrete Fourier transform. The function `dynspec()` can be used to better understand the principle of the short-time discrete Fourier transform. A series of frequency spectra are computed along the signal, here the dataset *sheep*, for a given Fourier window. The screenshot here shows the frequency spectrum computed for the eleventh window located at 0.672 s along the sound. The Fourier window has a length of 512 samples and is tapered by a Hanning window (default values of the arguments `w1` and `wn` respectively). Moving along the signal is made possible, thanks to the small control pop-up window entitled “Position.” Operating system: Ubuntu ..... 310

Fig. 11.2 Heisenberg box. The principle of the short-time discrete Fourier transform is based on a division of the time-frequency plane into an array of atoms. A unity atom

is named a Heisenberg box represented as a quadrilateral with a width  $\sigma_t$  and a height  $\sigma_f$ . The window function applied on the frequency domain applies as well on the frequency domain. Inspired from Mallat (2009) ..... 312

Fig. 11.3 Short-time discrete Fourier transform: atom shape. The figure shows the shape of the atoms (or Heisenberg boxes) for different window sizes. Four time width  $\sigma_t$  are considered: 128, 256, 512 and 1024 samples for a 0.2 s sound sampled at 44,100 Hz. A zoom is operated along the frequency y-axis from 0 to 2000 Hz. To facilitate the comparison, one central atom is highlighted in blue..... 313

Fig. 11.4 Short-time Fourier discrete transform: atom shape with overlapping. The figure shows the shape of the atoms (or Heisenberg boxes) obtained with a window made of 512 samples. Four overlaps between successive windows are considered: 0%, 50%, 75%, and 87.5% for a 0.2 s sound sampled at 44,100 Hz. A zoom is operated along the frequency y-axis from 0 to 2000 Hz. To facilitate the comparison, one central atom is highlighted in blue..... 314

Fig. 11.5 Short-time Fourier transform: atom shape with zero-padding. The figure shows the shape of the atoms (or Heisenberg boxes) obtained with a window made of 512 samples. Four zero-padding settings are considered: 0, 32, 64, and 128 for a 0.2 s sound sampled at 44,100 Hz. A zoom is operated along the frequency y-axis from 0 to 2000 Hz. To facilitate the comparison, one central atom is highlighted in blue ..... 316

Fig. 11.6 Spectrogram with `specgram()` of signal. The spectrogram is computed and displayed with the function `specgram()` of the package `signal`. Fourier window size = 512 samples, overlap = 75% = 383 samples, Hanning window..... 320

Fig. 11.7 Spectrogram with `periodogram()` of `tuneR`. The spectrogram is computed with the function `periodogram()` of the package `tuneR` and displayed with the function `image()`. Fourier window size = 512 samples, overlap = 75%, split cosine bell window..... 322

Fig. 11.8 Spectrogram with `powspec()` of `tuneR`. The spectrogram is computed with the function `powspec()` of the package `tuneR` and displayed with the function `image()`. Fourier window size = 512 samples, overlap = 75%, Hamming window ..... 323

Fig. 11.9 Spectrogram with `spectrogram()` of `phonTools`. Fourier window size = 512 samples, overlap = 75%, Hamming window ..... 324

Fig. 11.10 Spectrogram with `spectrogram()` of `soundgen`. Fourier window size = 512 samples, overlap = 75%, Hamming window ..... 325

Fig. 11.11 Spectrogram with `spectro()` of `seewave`. Fourier window size = 512 samples, overlap = 75%, Hanning window ... 326

Fig. 11.12 Pictures of soniferous animals: the hissing cockroach of Madagascar *Elliptorhina chopardi* (reproduced with the kind permission of Emmanuel Delfosse) and the Kuhl’s pipistrelle *Pipistrellus kuhlii*, a bat commonly found in Europe (reproduced with the kind permission of Laurent Arthur) ..... 329

Fig. 11.13 Different Fourier window length with `spectro()`. The spectrogram of `cockroach` was obtained with `wl = {128, 256, 512, 1024}` samples. Other STDFT parameters: Hanning window, 0% of overlap, no zero-padding ..... 330

Fig. 11.14 Different Fourier window overlaps with `spectro()`. The spectrogram of `cockroach` was obtained with `ovlp = {25, 50, 75, 87.5}` samples. Other STDFT parameters: Hanning window, 512 samples, no zero-padding .... 331

Fig. 11.15 The spectrogram is computed with the function `spectro()` of the package `seewave` and displayed with the function `image()`. Fourier window size = 512 samples, overlap = 75%, Hanning window ..... 335

Fig. 11.16 Spectrogram, oscillogram and amplitude scale display with `spectro()`. STDFT parameters: Hanning window, 512 samples, 87.5% of overlap, no zero-padding..... 336

Fig. 11.17 Contour plot with `spectro()`. The contours shows iso-dB lines from -30 to 0 dB regularly spaced by 4dB. STDFT parameters: Hanning window, 512 samples, 87.5% of overlap, no zero-padding ..... 339

Fig. 11.18 Spectrogram with a logarithmic frequency scale. The logarithmic scale obtained the argument `flog=TRUE` ..... 340

Fig. 11.19 Different color levels with `spectro()`. The spectrogram of `cockroach` was obtained with four different series of color levels: a linear series going from -30 to 0 by step of 1 (`collevels=seq(-30, 0, 1)`), a linear series going from -60 to 0 by step of 4 (`collevels=seq(-60, 0, 4)`), a linear series going from -30 to 0 by step of 15 creating a two-color scale (`collevels=seq(-30, 0, 15)`), and a logarithmic series from -30 to 0 (`collevels=c(-exp(seq(log(30), 0, length=30)))`). Other STDFT parameters: Hanning window, 512 samples, 87.5% of overlap, no zero-padding ..... 341

Fig. 11.20	Color palettes to be used with <code>spectro()</code> . Examples of different colour palettes for the amplitude scale of a spectrogram. The <code>jet.colors</code> and <code>green.colors</code> palettes were obtained with <code>colorRampPalette()</code> . See text for details .....	342
Fig. 11.21	Change of colour palette with the function <code>choose_palette()</code> of the package <code>colorspace</code> . This screenshot shows the interactive tool to select a colour palette according to several parameters and the result on the face spectrogram. Operating system: Ubuntu .....	342
Fig. 11.22	Different colour palettes with <code>spectro()</code> . The spectrogram of <code>cockroach</code> was obtained with the palettes <code>temp.colors</code> , <code>jet.colors</code> , <code>green.colors</code> , and <code>reverse.gray.colors</code> . STDFT parameters: Hanning window, 512 samples, 87.5% of overlap, no zero-padding .....	343
Fig. 11.23	Color changes with <code>spectro()</code> . The colors of the grid, the axes, the labels, and oscillogram are set to white when the background is turned to black. The palette was also changed for a better contrast with the background .....	344
Fig. 11.24	Zoom-in and axes changes with <code>spectro()</code> . The spectrogram of <code>cockroach</code> is zoomed in in time and frequency, and changes are applied to the axes: the size of the labels and values are changed, and the unit of the frequency axis is changed to Hz .....	347
Fig. 11.25	Spectrogram decoration. The spectrogram of <code>cockroach</code> obtained with <code>spectro()</code> is decorated with the low level plot functions <code>arrows</code> , <code>text</code> , <code>points</code> , and <code>rect</code> .....	349
Fig. 11.26	Spectrogram selections with <code>manualoc()</code> . Manual annotations were added by clicking on the spectrographic display. Here eight regions of interest were delimited .....	355
Fig. 11.27	Spectrogram annotations with <code>viewSpec()</code> . Three regions of interest were delimited, saved, and read back with <code>viewSpec()</code> .....	358
Fig. 11.28	The main principle of <code>acoustat</code> . One of the most important stages in the process is to estimate a time and a frequency contour through an aggregation of the columns and rows of the STDFT matrix. The example, here based on <code>cockroach</code> , shows the spectrogram and the contours. The contours are drawn with a line and points to show the discretization due to the STDFT. STDFT parameters: Hanning window, 512 samples, 87.5% of overlap, no zero-padding .....	360
Fig. 11.29	Parametrization of the spectrogram with <code>acoustat()</code> . Visual display of the function <code>acoustat()</code> with the time	

envelope (top) and the frequency contour (bottom). The median and quartiles are indicated with vertical red segments.... 361

Fig. 11.30 Several spectrograms in a single graphic display. The spectrogram of `tico`, `orni`, `peewit`, and `cockroach` are arranged to be all plotted in a single graphic display. The amplitude color scale is added with the function `dBscale()` ..... 363

Fig. 11.31 Saving a spectrogram in a raster file. This image was produced using the function `png()` to print the spectrogram of `forest` into a `.png` file. The settings of `png()` and `spectro()` were adjusted to widen the spectrogram..... 365

Fig. 11.32 Saving a long spectrogram in a series of raster files. These two images saved into two separated `jpeg` files were produced using the function `lspec()` of `warbleR` to split and print the 60 s spectrogram of `forest` ..... 367

Fig. 11.33 Dynamic spectrogram. The function `dynspectro()` can be used to navigate along a long sound. A series of STDFT are computed along the signal, here the sound `forest`, for a given number of frames set with the argument `slidframe`. The screenshot here shows the STDFT computed for the frame between 11.05 and 20.04 s. Moving along the signal is made possible, thanks to the small control pop-up window entitled "Position." Operating system: Ubuntu ..... 368

Fig. 11.34 Waterfall display. The figure shows four examples of waterfall display obtained by applying the function `wf()` on `cockroach`. STDFT parameters: Hanning window, 512 samples, 50% of overlap, no zero-padding ..... 371

Fig. 11.35 3D animation of the `cockroach` spectrogram. Animation around the 3D spectrogram of `cockroach` based on a series of 100 `.png` images. Animated on electronic version only ..... 373

Fig. 11.36 Mean frequency spectrum with `meanspec()`. The plot shows the mean frequency spectrum of `peewit`, a sound with few frequency modulations. STFT parameters: Hanning window, 512 samples, 87.5% of overlap, no zero-padding ..... 376

Fig. 11.37 Issues with the mean spectrum. The mean spectrum can returned counterintuitive results as illustrated with three synthetic samples (top-left, top-right, bottom-left) and the natural `cockroach` whistle (bottom-right). For each case the spectrogram is shown on the left and the mean spectrum on the right..... 378

Fig. 11.38 Soundscape frequency spectrum. The soundscape frequency spectrum, here computed and displayed for the recording *forest* consists in a Welch frequency spectrum binned into 1 kHz frequency bands. The graphic is based on the high-level plot graphic function `barplot()` ..... 379

Fig. 12.1 Mel-frequency filter bank. A bank of mel-frequency triangular filters is generated and displayed with the `seewave` function `melfilterbank()`. The bank includes 26 filters starting from 0.3 to 22.05 kHz..... 383

Fig. 12.2 Auditory spectrum. The result of the function `audspec()` is displayed with the function `image()`. The left y-axis refers to frequency expressed in mel and the right y-axis indicates the index of the 26 mel-frequency filters used. Time was divided into 74 windows by the STDFT ..... 387

Fig. 12.3 Lifters on 13 MFCCs that are all equal to 1. The blue and dashed line displays the 13 MFCCs. The plain black lines show the weighting function of seven lifters differing in their length, from 9 to 15. The lifter of length 12, that is, the number of MFCCs-1, applies a perfect sine function between 0 and  $\pi$  ..... 389

Fig. 12.4 Display of the MFCCs. The 13 MFCCs selected are displayed according to time that was divided into 74 windows by the STDFT ..... 393

Fig. 12.5 Filter frequency response deriving from LPC. The function `lpc()` returns the LPC coefficients of a sound, here *hello*, and plots the resulting filter frequency response (black line). The original frequency spectrum obtained after a pre-emphasis filter is also shown (blue line) ..... 396

Fig. 12.6 Formant analysis based on LPC. The function `findformants()` can estimate the resonant frequency  $f_r$  and  $-3$  dB bandwidth  $\Delta_{-3\text{dB}}f$  of each formant. A pole-zero diagram (right) completes the spectral display (left) to show the position of the formants in the complex unit circle ..... 397

Fig. 13.1 Dominant frequency tracking with `dfreq()`. The dominant frequency of *sheep* is tracked along time calling the function `dfreq()` which computes in background a STDFT, here with a Fourier window length of 512 samples (`wl=512`) and an overlap between successive Fourier windows of 87.5% (`wl=87.5`) ..... 401

Fig. 13.2 Dominant frequency tracking with different settings of `dfreq()`. The graphic displays the results obtained with the function `dfreq()` using five different settings ..... 402

Fig. 13.3 Fundamental frequency tracking with `autoc()`. The graphic displays the results obtained with the function `autoc()` on `sheep` using four different settings. The figure was manually obtained with `plot()`, `points()`, and `legend()` ..... 406

Fig. 13.4 Fundamental frequency tracking with `fund()`. The graphic displays the results obtained with the function `fund()` on `sheep` using four different settings. The figure was manually obtained with `plot()`, `points()`, and `legend()` ..... 409

Fig. 13.5 Fundamental frequency tracking with `FF()`. The graphic displays the results obtained with the function `FF()` on `sheep` using default and tuned settings. The figure was manually obtained with `plot()`, `points()`, and `legend()` ..... 410

Fig. 13.6 Melody plot. The `tuner` function `melodyplot()` displays the notes estimated from the fundamental frequency, here the fundamental frequency of the `theremin` sound ..... 412

Fig. 13.7 Melody quantization plot. The `tuner` function `quantplot()` displays the notes estimated from the fundamental frequency after having binned the time scale, here for the `theremin` sound ..... 413

Fig. 13.8 Fundamental frequency tracking with `pitchtrack()`. The fundamental frequency of the voice data `hello` is detected and tracked with the function `pitchtrack()` of `phonTools`. The result is plotted over a spectrogram obtained with `spectro()` of `seewave` ..... 414

Fig. 13.9 Fundamental frequency tracking with `analyze()`. The fundamental frequency of the voice data `hello` is detected and tracked with the function `analyze()` of `soundgen` following four methods which, here, return the same results. The legend was added manually with `legend()` ..... 415

Fig. 13.10 Formant tracking with `formanttrack()`. The formants of the voice data `hello` are detected and tracked with the function `formanttrack()` of `phonTools`. The results, here for three formants, are plotted over a spectrogram obtained with `spectro()` of `seewave` ..... 417

Fig. 13.11 Instantaneous frequency tracking with `ifreq()`. The instantaneous frequency is computed and plotted with the function `ifreq()` on `tico`. An amplitude threshold of 6% was applied to select the notes ..... 419

Fig. 13.12 Artifact of instantaneous frequency tracking. The instantaneous frequency is computed and plotted with the

function `ifreq()` on `bat`. An amplitude threshold of 5% was applied to select the call. The function can properly estimate the instantaneous frequency when the sound is monotonal but not when an harmonic appears making the sound bitonal ..... 421

Fig. 13.13 Frequency modulation analysis of the theremin sound. The function `fma()` shows a first peak at 0.006 kHz. This peak was here identified using `identify=TRUE` and then added on the graphic with the low-level plot functions `points()` and `text()` as in Fig. 8.11. Note that the peak can also be automatically identified using `fpeaks()` ..... 422

Fig. 13.14 Zero-crossing principle. Positions where the signal crosses the zero line are identified (red points) and used to estimate the instantaneous period  $T_{zc}$  and therefore the instantaneous frequency  $f_{zc}$  ..... 422

Fig. 13.15 Zero-crossing with a multi-tonal sound. A sound made of different frequencies, here a fundamental and its first harmonic, crosses the zero line several times such that the instantaneous frequency varies around four values ..... 423

Fig. 13.16 Zero-crossing limitation and interpolation solution. The figure is based on the analysis of a 0.1 s sound sampled at 44,100 Hz with a linear frequency increasing from 0 to 22,050 Hz. Without interpolation the ZC is very inaccurate when getting close to the Nyquist frequency (top). This error can be reduced by interpolating the original signal, here with a  $\times 10$  factor (bottom) ..... 424

Fig. 13.17 Instantaneous frequency tracking with `zc()`. The instantaneous frequency of the `bat` call is estimated using the zero-crossing principle without (top) and with a tenfold interpolation (bottom) ..... 425

Fig. 13.18 Zero-crossing rate. The zero-crossing rate method is used on `bat` sound by dividing the signal in 53 successive windows by setting the arguments `wl=512` and `wl=87.5` ..... 426

Fig. 13.19 Teager-Kaiser energy operator. Examples of TKEO applied to amplitude modulated (AM) and/or frequency modulated (FM) sounds ..... 429

Fig. 13.20 Teager-Kaiser energy operator with multi-tonal sound. The TKEO does not return appropriate results with a multi-tonal sound, as illustrated here with a sound with a carrier frequency at 2000 Hz and four harmonics. Spectrogram (top) and TKEO (bottom) ..... 430

Fig. 13.21 Teager-Kaiser energy operator with high-frequency content. The TKEO does not return appropriate results for frequencies above  $f_s \div 4$ , as illustrated here with a frequency modulated sound starting at 0 Hz and ending at



$f_s \div 2 = 22,050$  Hz. The vertical (frequency) or horizontal (vertical) blue line indicates where the TKEO is no more operational. Spectrogram (top) and TKEO (bottom) ..... 431

Fig. 13.22 Teager-Kaiser energy operator with noise. The TKEO does not return appropriate results when the system, that is the recording, includes noise as illustrated here with a frequency modulated sound starting at 0 Hz and ending at  $f_s \div 2 = 22,050$  Hz mixed with white noise. Spectrogram (top) and TKEO (bottom) ..... 432

Fig. 13.23 Teager-Kaiser applied on `tico` and `sheep`. The TKEO can be applied directly on `tico` as the conditions of application are met (top). However, the TKEO does not return relevant results if applied on `sheep` that does not meet all conditions of application (middle). A band-pass filter between 500 and 700 Hz can solve the problem by focusing on a single and low-frequency band (bottom) ..... 433

Fig. 14.1 Pictures of soniferous animals: the South-American poison frog *Allobates femoralis* and the European midwife toad *Alytes obstetricans* (Reproduced with the kind permission of pictures by Andrius Pasukonis and Diego Llusia) ..... 436

Fig. 14.2 Spectrogram and oscillogram of the vocalization of the dart poison frog *Allobates femoralis*. The recording includes two sequences of four notes and background noise due to wind, distant individuals, and insects. Fourier window size = 512 samples, overlap = 0%, Hanning window ..... 437

Fig. 14.3 Spectrogram and oscillogram of the vocalization of the European midwife toad *Alytes obstetricans*. The recording includes three notes, wind, and insects. Fourier window size = 512 samples, 0% of overlap, Hanning window ..... 438

Fig. 14.4 Principle of a frequency filter. The figure sketches how a frequency filter can change the frequency content of a sound. The input sound is a white noise with a flat frequency spectrum (left), the filter is characterized by a transfer function  $H(f)$  with a bell-like shape (middle), and the output has a frequency spectrum with a shape similar to the filter transfer function (right). Note that the frequency  $x$ -axis follows a logarithmic scale. Inspired from Speaks (1999)..... 439

Fig. 14.5 Transfer function of preemphasis filter. The figure shows the Bode plot of the transfer function of preemphasis filters with values of  $\alpha$  varying between 0 and 1 ..... 441

Fig. 14.6 Example of a preemphasis filter. Graphical display of the `seewave` function `preemphasis()` showing side-by-side the spectrogram of the filtered signal, here

hello, and the frequency response of the filter along a linear amplitude scale ..... 442

Fig. 14.7 Effect of varying the  $\alpha$  time constant of the preemphasis filter. The mean spectra of the original signal ( $\alpha = 0$ ) and filtered signals ( $\alpha = \{0.1, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$ ) using the `seewave` function `preemphasis` are plotted on the same graph. This illustrates how much high-frequency content is enhanced depending on the value of  $\alpha$ . Mean spectra parameters: Hanning window, 1024 samples, 87.5% of overlap, no zero-padding..... 443

Fig. 14.8 Transfer function of comb filter. The top graphic shows the transfer function  $H$  of five comb filters differing in  $\alpha$  but not in  $K$  ( $K = 0.001$ ). The sharpness of the peak increases with  $\alpha$ . The bottom graphic shows the transfer function  $H$  of four comb filters differing in  $K$  but not in  $\alpha$ . The number and position of peaks changes with  $K$  ..... 444

Fig. 14.9 Example of a comb filter. Graphical display of the `seewave` function `comb` showing side-by-side the spectrogram of the filtered signal, here `hello`, and the frequency response of the filter along a linear amplitude scale ... 445

Fig. 14.10 Transfer function of Butterworth filter. The figure shows the Bode plot of the transfer function of a 100 Hz high-pass, a 1000 Hz low-pass, a 100–1000 Hz band-pass, and a 100–1000 Hz band-stop of a 1–5th Butterworth filter. The vertical black dashed-line show the cutoff frequency(ies) and the gray grid underlines the  $-20$  dB roll-off per decade ..... 446

Fig. 14.11 Filter through wave smoothing with `smoothw()`. The original `femo` recording (left) is passed through a wave smoothing a first time (middle) and a second time (right)..... 450

Fig. 14.12 Filter through wave smoothing with `rmnoise()`. The original `femo` recording (left) is passed through a cubic smoothing spline with a smoothing parameter `spar=0.4` (middle) and `spar=0.6` (right) ..... 451

Fig. 14.13 Principle of DFT filter. A DFT filter is based on a return travel between the time and frequency domains: the frequency signal spectrum  $F[n]$  of the original signal  $s[n]$  is multiplied by the transfer function of the filter  $H[n]$ , here a low-pass filter, and the filtered signal is obtained through the inverse Fourier transform. Each function is made of  $n$  samples..... 452

Fig. 14.14 Example of STDFT filter. Two examples of DFT filter based on the function `istfft()`. The second harmonic of the first harmonic of the fourth note of `femo` was

removed with a band-stop filter (top) or selected with band-pass filter (bottom). The red square was added using the low-level plot function `rect()` ..... 454

Fig. 14.15 Principle of FIR filter. A FIR filter is based on a convolution (`*` sign) between the original signal  $s[n]$  and the transfer function of the filter  $H[n]$  expressed in the time domain. This latter can be obtained from the transfer function in the frequency domain using the inverse of the Fourier transform (IDFT) ..... 456

Fig. 14.16 Mean frequency spectrum of `toad`. The recording not only includes vocalizations produced by a male of *Alytes obstetricans* but also wind and nocturnal insect stridulations ..... 457

Fig. 14.17 Oscillogram of `toad` before and after FIR filtration. The oscillogram of `toad`, a recording including three vocalizations of *Alytes obstetricans*, wind, and stridulation of nocturnal insects, is shown before (top) and after filtration (bottom) ..... 458

Fig. 14.18 Antialiasing FIR filter. The figure shows the original signal `peewit`, the downsampled and distorted version without any filter process, and the downsampled version with a low-pass FIR filter ..... 460

Fig. 14.19 Band-pass frequency transfer functions. Five band-pass transfer functions are displayed on a plot with linear scales. These functions were built manually with basic numeric vectors or with the help of the functions `squarefilter()` and `drawfilter()` ..... 461

Fig. 14.20 Correction FIR filter for a loudspeaker. Plot of the frequency filter of the original noise (input) given to the loudspeaker, of the noise as recorded after being broadcast by the loudspeaker, and of the noise corrected by the FIR filter ..... 464

Fig. 15.1 Changing the amplitude envelope with `setenv()`. The amplitude envelope of `tico` was applied to `tuningfork`. Fourier window size = 512 samples, overlap = 0%, Hanning window ..... 466

Fig. 15.2 Changing the amplitude envelope with `drawenv()`. The amplitude envelope of `tico` was modified graphically using the mouse cursor ..... 467

Fig. 15.3 Amplitude filter with `afilter()`. The original `femo` recording (left) was passed through an amplitude filter with a threshold of 3% (middle) and 5% (right). Fourier window size = 512 samples, overlap = 0%, Hanning windows ..... 469

Fig. 15.4 Use of `afilter()` on dominant frequency tracking. The graphic shows the results of tracking the dominant

	frequency of <code>femo</code> after having filtered the sound using <code>afilter()</code> with different settings.....	470
Fig. 15.5	Modifications using the ISTDFT. Four examples of sound modifications on <code>femo</code> based on the function <code>istfft()</code> . The second harmonic of the first harmonic of the fourth note was amplified (top-left), reversed in frequency (top-right), replaced by a pure tone (bottom-left), and replaced by noise (bottom-right). Fourier window size = 512 samples, overlap = 0%, Hanning windows .....	471
Fig. 15.6	Linear frequency shift using the ISTDFT. The song of <code>orni</code> was shifted toward low or high frequencies with the function <code>lfs()</code> that uses the ISTDFT in background. Fourier window size = 512 samples, overlap = 0%, Hanning window .....	475
Fig. 15.7	Modifications using the Hilbert transform. Three examples of sound modifications on <code>tico</code> based on the function <code>synth2()</code> . The frequency modulation was inverted according to time (left), the frequencies were multiplied by 2 (middle), and the frequency modulation was replaced by 4000 Hz pure tone (right). Fourier window size = 512 samples, overlap = 0%, Hanning window .....	477
Fig. 16.1	Recording the French Guiana tropical acoustic communities. Twelve autonomous recorder SM2 of the company Wildlife Acoustics <sup>®</sup> were settled in the Nouragues reserve in French Guiana to record both understory and canopy acoustic communities. For each recorder, one microphone was installed at 1.5 m (understory recording), and another one was set at a height of 20 m (canopy recording). The hanging microphone on the right of the picture is ready to be sent up to the canopy. Picture by Jérôme Sueur and Amandine Gasc .....	480
Fig. 16.2	Barplot of the values used by the acoustic diversity index ( <i>ADI</i> ). The relative amplitude values of the frequency bins used to compute <i>ADI</i> are plotted as a barplot. The values were obtained with a maximum frequency of 22,050 Hz and a frequency step of 500 Hz .....	488
Fig. 16.3	Visualization of three $\beta$ indices. Graphical output of the function <code>diffspec()</code> (top), <code>diffcumspec()</code> (middle), and <code>ks.dist()</code> (bottom) for the indices $D_f$ , $D_{cf}$ , and $D_{KS}$ , respectively. In each case the mean spectra of the two sounds <code>night</code> and <code>day</code> were provided to the functions. The gray area or the segment indicates the dissimilarity index .....	498

Fig. 16.4 Comparison between spectral dissimilarity index and cumulative spectral dissimilarity index. The indices  $D_f$  and  $D_{cf}$  return different values for spectra of similar shapes but with different frequency. The examples are here for pure-tone theoretic sounds. The index  $D_f$  returns the same value in the two cases (probability mass functions of two frequency spectra, top-left and top-right) when  $D_{cf}$  returns expected low and high values (cumulated probability mass functions of two frequency spectra, bottom-left and bottom-right)..... 499

Fig. 16.5 Visualization of a  $\beta$  index matrix with a heatmap. The dissimilarity matrix obtained with the cumulative spectral difference  $D_{cf}$  index was plotted as a heatmap using the function `image()`. The scale on the left was produced by taking advantage of the function `dBscale()` used to add a dB scale to a spectrogram. Gray lines were added manually with `abline()` ..... 510

Fig. 16.6 Visualization of a  $\beta$  index matrix with a hierarchical cluster analysis dendrogram. The dissimilarity matrix obtained with the cumulative spectral difference  $D_{cf}$  was treated with hierarchical cluster analysis, the result being plotted as a dendrogram. The color rectangles show how to cut the dendrogram in 2, 3, or 4 clusters ..... 512

Fig. 16.7 Visualization of a  $\beta$  index matrix with a db-RDA projection according to hour. The  $\beta$  index matrix was treated with a distance-based redundancy analysis, and the observations were projected with `s.class()` in the space defined by the two first axes of the ordination process. Each observation is one factor level, i.e., there is a single observation per factor level ..... 516

Fig. 16.8 Visualization of a  $\beta$  index matrix with a db-RDA projection according to time periods. The  $\beta$  index matrix was treated with a distance-based redundancy analysis, and the observations were projected with `s.class()` in the space defined by the two first axes of the ordination process. Each observation is grouped according to a factor with the three levels: morning, day, and night. Ellipses would include 67.5% of the observations ..... 517

Fig. 16.9 Tuned visualization of the  $\beta$  index matrix with a db-RDA projection according to time periods. This is another version of the graphic displayed in Fig. 16.8 but tuned by modifying some arguments of `s.class()`. In particular the ellipses would here cover 95% of the observations..... 518

Fig. 16.10 Visualization of the db-RDA permutation test. The histogram shows the distribution of the statistic obtained

by permutation when  $H_0$  is true. The statistic observed for the data tested is depicted as a diamond placed on the top of a segment. The  $p$ -value of the test is the probability to obtain a statistic greater than the statistic observed, that is, the surface of the histogram on the right of the diamond, here  $p \approx 0.0009$  ..... 519

Fig. 17.1 Cross-correlation principle. Cross-correlation mainly consists in moving forward and backward a series along another series and in computing a correlation coefficient at each  $m$  lag step. In this graphic, the blue  $x$  series is moved forward ( $m > 0$ ) along the red series  $y$  (top) generating a time series of the correlation coefficient  $r_{xy}$  (bottom). The correlation time series shows a peak for a lag of 0.1 s indicating that the two series are shifted by 0.1 s. The backward movement ( $m < 0$ ) is not shown for a sake of clarity ..... 522

Fig. 17.2 Waveform cross-correlation. The waveform of the second and third notes of `tico` was cross-correlated with the base function `ccf()`. The figure shows the oscillogram of the two notes and the time series of the correlation coefficient  $r_{xy}(m)$ , where  $m$  is the lag in s ..... 525

Fig. 17.3 Hilbert amplitude envelope cross-correlation. The Hilbert amplitude envelopes of the second and third note of `tico` were cross-correlated with the function `corenv()`. The cross-correlation indicates a frequency shift, or offset, of 0.014 s ..... 526

Fig. 17.4 Frequency spectrum cross-correlation. The mean frequency spectra of the second and third note of `tico` were cross-correlated with the function `corspec()`. The cross-correlation indicates a frequency shift, or offset, of 0.26 kHz ..... 527

Fig. 17.5 STDFT cross-correlation of STDFT matrices. The STDFT matrices of the second and third note of `tico` were cross-correlated with the function `covspectro()`. The cross-correlation indicates a time shift, or offset, of 0.02 s ..... 528

Fig. 17.6 Frequency coherence. Frequency coherence between the left and right channel of a recording achieved at tea time in French Guiana. A value of 1 indicates a pure coherence. Here the coherence is maximum between 10 and 15 kHz ..... 530

Fig. 17.7 Continuous frequency coherence. The frequency coherence is computed along time using `ccoh()`, a short-term version of `coh()`. Here the function is applied between the left and right channel of the a recording achieved at tea time in French Guiana ..... 531

Fig. 17.8 Dynamic time warping on Hilbert amplitude envelope. The smoothed Hilbert amplitude envelopes of `note2` and `note3` of `tico` are compared using dynamic time warping alignment. Note that the envelopes here have the same length (176 samples) but that their length could differ. The dotted gray lines connect the samples that match following the best alignment found by the algorithm..... 532

Fig. 17.9 Dynamic time warping on frequency spectra. The mean frequency spectra of `note2` and `note3` of `tico` are compared using dynamic time warping alignment. Note that the frequency spectra have here the same length (256 bins) but that their length could differ. The dotted gray lines connect the frequency bins that match following the best alignment found by the algorithm ..... 533

Fig. 17.10 Dynamic time warping on dominant frequency tracking. The dominant frequency of `note2` and `note3` of `tico` was obtained with `dfreq()` and then compared using dynamic time warping alignment. Note that the frequency tracks have not the same length (11 and 14 measurements, respectively). The dotted gray lines connect the dominant frequency measurements that match following the best alignment found by the algorithm ..... 534

Fig. 17.11 Automatic identification system workflow. An automatic identification system can be divided into two major components: a first phase of development where the system is built and trained based on one or several templates, one or several training datasets, and a second phase of application on one or several test datasets. The plain arrows indicate the basic way of the workflow, and the dashed arrows indicate feedback to optimize the system. See text for further details ..... 535

Fig. 17.12 Receiver operating characteristic (ROC). The false positive rate (FPR) and the true positive rate (TPR) define the ROC space. The plain curves indicate the ROC curves for an efficient system (blue), a non-efficient system (red), and a system returning random predictions (pink). Areas under the curve (AUC) are colored accordingly and specified in the legend ..... 537

Fig. 17.13 Visualization of manual annotations with `viewSpec()`. The 28 SOI of the `Allobates_femorialis.wav` recording were delimited and overlaid on a spectrographic display with `viewSpec()` ..... 541

Fig. 17.14 Cross-correlation with the package `monitor`. The time series of the correlation coefficient as stored in the result of the function `corMatch()`. The function

was applied between four templates and a training file `Allobates_femoralis.wav`. Only the score for the template `t1` is here displayed..... 544

Fig. 17.15 Automatic detection with the package `monitor`. The two-panel figure obtained with `plot()` on an object obtained with `findPeaks()` on the template 1. The top panel is a spectrogram with detections indicated with red rectangles. The bottom panel shows the time series of the correlation coefficient, here named `Score`. In this case, no selection (threshold  $\theta = -0.1$ ) was applied so that all peaks were considered as positive or true detections ..... 546

Fig. 17.16 ROC curve for *Allobates femoralis* vocalization identification. The curve was built by varying the output threshold  $\theta$  from 0 to 1 by step of 0.01. The size of the points is relative to  $\theta$ . The point 67 was chosen as the best output threshold  $\theta$  with a good TPR and a null FPR ..... 550

Fig. 17.17 Variation of the area under the curve (AUC) according to time tolerance ( $\tau$ ). The AUC was computed for a series of time tolerances between 0 and 0.2. The area reaches a maximum when  $\tau = 0.09$ ..... 552

Fig. 17.18 Automatic detection with the package `monitor`: final check. The final results of the automatic detection system applied on the training dataset, here a single file containing 28 vocalizations of *Allobates femoralis*. The plot shows the detections of all four templates. Only the sixth vocalization is missed ..... 554

Fig. 18.1 Frequency spectrum of white and colored noises. The noises were obtained with the function `noise()` of `tuneR`. The frequency spectra were built calling `spec()` with a log frequency  $x$ -axis and a dB  $y$ -axis ..... 557

Fig. 18.2 Synthesis of pulse waves. Four series of pulses were generated with `pulsew()` of `seewave` and `pulse()` of `tuneR`. The waveforms were plot with `oscillo()` ..... 559

Fig. 18.3 Synthesis of square waves. Four series of squares were generated with `square()` of `tuneR`. The waveforms were plot with `oscillo()` ..... 561

Fig. 18.4 Synthesis of sawtooth waves. Four series of sawtooth were generated with `sawtooth()` of `tuneR`. The waveforms were plot with `oscillo()` ..... 562

Fig. 18.5 Frequency beating. Beating can arise when adding pure tones closely related in frequency. The addition of two pure tones with carrier frequencies of 50 and 55 Hz generates a sound with an amplitude modulation of 5 Hz ..... 565



Fig. 18.6 Constructive and destructive interference. The pure tones  $s_1$  and  $s_2$  have a similar frequency of 3 Hz and are in phase, whereas  $s_1$  and  $s_3$  have also a frequency of 3 Hz but are out of phase that is an absolute phase shift of  $\pi$  rad. The sum of  $s_1$  and  $s_2$  returns a reinforced sound due to constructive interference. The sum of  $s_1$  and  $s_3$  leads to a null sound due to destructive interference..... 566

Fig. 18.7 Synthesis of an harmonic series. This series leads to a waveform with a square-like shape. The figure was produced calling `spectro()` using the arguments `tlim` and `flim` to zoom in time and frequency. Fourier window size = 512 samples, overlap = 0%, Hanning window ..... 569

Fig. 18.8 Synthesis of a sine wave with amplitude envelop changes. A 440 Hz sine sound was synthesized using `sine()` and multiplied with an amplitude envelope following a linear (top), exponential (middle) and sinusoid (bottom) increase ..... 570

Fig. 18.9 Synthesis of harmonic series. Four examples of use of the argument `harmonics` of `synth()`. See text for details. Fourier window size = 1024 samples, overlap = 0%, Hanning window, frequency zooming between 0 and 5 kHz ..... 572

Fig. 18.10 Synthesis of chirps. Linear, quadratic and logarithmic chirps were synthesized with `chirp()` and visualized with `spectro()`. Fourier window size = 1024 samples, overlap = 87.5%, Hanning window ..... 575

Fig. 18.11 Modulation synthesis: parameters of `synth()`. The arguments `am` and `fm` control the amplitude modulation (AM) and frequency modulation (FM) parameters. Each parameter is labeled according to the element position in the argument. For instance, `fm[2]` indicates the second element of the argument `fm`, that is, the frequency deviation of the sinusoid FM. The sound used as an example combines a sinusoid AM, a positive linear FM, and a sinusoid FM. The sound was synthesized with `synth(f=44100, d=1, cf=5000, fm=c(2000, 10, 10000, pi/2), am=c(80, 5, pi/2))`. Fourier window size = 1024 samples, overlap = 87.5%, Hanning window ..... 576

Fig. 18.12 Modulation synthesis full example with `synth()`. The sound was generated using most of the arguments of `synth()`. The display was directly produced with `plot=TRUE`. Fourier window size = 1024 samples, overlap = 87.5%, Hanning window ..... 578

Fig. 18.13 Synthesis of an exponential chirp with harmonics. The sound was generated using the arguments `fm` and

harmonics of `synth()`. Fourier window size = 1024 samples, overlap = 87.5%, Hanning window ..... 579

Fig. 18.14 Synthesis of a combination of exponential chirps. The sound was generated using the argument `fm` of `synth()` and the addition of two synthetic sounds. Fourier window size = 512 samples, overlap = 0%, Hanning window ..... 580

Fig. 18.15 Synthesis of AM waves. Four AM waves differing in the depth ( $m$ ) and frequency ( $f_{am}$ ) of the AM. These AM waves are characterized by frequency sidebands. Fourier window size = 512 samples, overlap = 0%, Hanning window, dynamic range = 60 dB ..... 581

Fig. 18.16 Synthesis of FM waves. Four FM waves differing in their modulation index  $\beta = \Delta f_c \div f_{fm}$  where  $\Delta f_c$  is the carrier frequency and ( $f_{fm}$ ) is the frequency of the FM. These FM waves are characterized by complex frequency sidebands. Fourier window size = 512 samples, overlap = 0%, Hanning window ..... 583

Fig. 18.17 Synthetic sound based on a numeric vector. The sound was generated using the handmade function `numsound()`. Fourier window size = 512 samples, overlap = 0%, Hanning window ..... 586

Fig. 18.18 Synthesis of C major scale notes. Synthesis of the 12 notes of the C major scale following Western music. Fourier window size = 4096 samples, overlap = 87.5%, Hanning window ..... 588

Fig. 18.19 Frequency spectrum of a Shepard scale tone. The bands are equally spaced along a log frequency scale ..... 591

Fig. 18.20 Synthesis of a Shepard scale. Six tones, or notes, composed, ordered to create an illusion of endlessly ascending pitch when repeated. Frequency zoom in between 0 and 5 kHz. Fourier window size = 4096 samples, overlap = 87.5%, Hanning window ..... 592

Fig. 18.21 Synthesis of a Risset glissando. Fourier window size = 4096 samples, overlap = 87.5%, Hanning window, dynamic range = 60 dB ..... 595

Fig. 18.22 Synthesis of the call of the tree cricket *Oecanthus pellucens*. Original (left) and synthesis (right) of one stridulation of the Italian tree cricket *Oecanthus pellucens*. Fourier window size = 512 samples, overlap = 87.5%, Hanning window ..... 596

Fig. 18.23 Synthesis of the call of the frog *Eleutherodactylus martinicensis*. Original (left) and synthesis (right) of four two-note vocalizations of the Martinique Robber frog *Eleutherodactylus martinicensis*. Fourier window size = 512 samples, overlap = 0%, Hanning window ..... 597

Fig. 18.24 Synthetic sound with AM and FM following a normal density function. The sound was generated using tonal principle with the function `synth2()`. Fourier window size = 1024 samples, overlap = 87.5%, Hanning window ..... 600

Fig. 18.25 Tonal synthesis based on a pre-existing sound. The pre-existing sound of `peewit` (left) was used to synthesize a new sound (right) with several frequency bands of equal energy. Fourier window size = 512 samples, overlap = 0%, Hanning window ..... 601

Fig. 18.26 Synthesis of a face-like sound. This smiling face was synthesized using additive synthesis with `synth()` and tonal synthesis `synth2()`. Fourier window size = 512 samples, overlap = 75%, Hanning window ..... 603

Fig. 18.27 Synthesis of an English speaker vowels with `phonTools`. The five vowels were synthesized with `vowelsynth()` of the package `phonTools`. Fourier window size = 512 samples, overlap = 87.5%, Hanning window, dynamic range = 60 dB ..... 606

Fig. 18.28 Synthesis of an English speaker vowels with `soundgen`. The five vowels were synthesized with `generateBout()`. Fourier window size = 512 samples, overlap = 87.5%, Hanning window, dynamic range = 60 dB .... 607

Fig. 18.29 `soundgen` Shiny application. A web Shiny application linked to the package `soundgen` ..... 608

# List of Tables

Table 2.1	dB ratios .....	14
Table 3.1	Type, mode, class and, dimensions of R objects .....	44
Table 3.2	R operators .....	46
Table 3.3	Fundamental R arithmetic and statistic functions .....	49
Table 3.4	Import and export of R data .....	61
Table 4.1	Equivalence between <code>audio</code> , <code>phonTools</code> , and <code>tuneR</code> functions dedicated to sound import and export .....	101
Table 5.1	Time resolution of a sliding window .....	132
Table 7.1	Main calibration arguments of <code>PAMGuide()</code> function .....	183
Table 8.1	Precision of manual time measurements on the <code>orni</code> sound ....	190
Table 8.2	Comparison of automatic time measurements on the <code>orni</code> sound .....	198
Table 9.1	The Fourier transformation family .....	216
Table 9.2	Spectral-cepstral dictionary .....	242
Table 10.1	Frequency and time resolution .....	254
Table 11.1	Time and frequency resolution of the STFT .....	317
Table 11.2	Correspondence between the main arguments of spectrographic functions found in several packages .....	318
Table 11.3	Default values of the arguments of the <code>seewave</code> function <code>spectro()</code> .....	328
Table 14.1	Types of frequency filters: short description of the frequency filters found in <code>seewave</code> , sorted by alphabetic order .....	439
Table 16.1	$\alpha$ acoustic indices: name, function, package, and main literature reference .....	483

Table 16.2  $\beta$  acoustic indices: name, function, package, and main literature reference ..... 495

Table 17.1 Confusion matrix in automatic identification process ..... 536

# List of DIY Boxes

DIY 4.1	How to read a single channel of a stereo file.....	92
DIY 5.1	How to draw your own oscillogram .....	114
DIY 5.2	How to highlight a part of an oscillogram with a different color ..	118
DIY 5.3	How to compute and draw the absolute amplitude envelope.....	126
DIY 6.1	How to apply mono conversion and to mix channels.....	145
DIY 6.2	How to split a sound into several sound bouts .....	147
DIY 7.1	How to estimate a distance of attenuation .....	179
DIY 8.1	How to take manually time measurements on a group of .wav files .....	191
DIY 10.1	How to plot two frequency spectra with the <code>ggplot2</code> style .....	262
DIY 10.2	How to code the piecewise aggregate approximation (PAA) .....	291
DIY 10.3	How to compute several spectral features on several sounds .....	298
DIY 11.1	How to change the position of the amplitude scale and plot a spectrum on the side of the spectrogram .....	336
DIY 11.2	How to print in 3D a spectrogram .....	374
DIY 12.1	How to obtain MFCCs step by step .....	390
DIY 13.1	How to plot the dominant frequency and fundamental frequency tracks on a single spectrogram .....	407
DIY 13.2	How to derive the instantaneous frequency using zero-crossing rate .....	426
DIY 14.1	How to produce the Bode plot of a Butterworth low-pass or high-pass filter.....	447
DIY 15.1	How to generate a series of sounds with different linear frequency shifts .....	473
DIY 16.1	How to tune the visualization of a db-RDA projection.....	515
DIY 18.1	How to a generate a symmetric triangle wave .....	563