Bioacoustics
The International Journal of Animal Sound and its Recording, 2008, Vol. 18, pp. 213–226
© 2008 AB Academic Publishers

EQUIPMENT REVIEW

SEEWAVE, A FREE MODULAR TOOL FOR SOUND ANALYSIS AND SYNTHESIS

JEROME SUEUR[1*], THIERRY AUBIN[2] AND CAROLINE SIMONIS[3]

[1]Muséum National d'Histoire Naturelle, Département Systématique et Évolution, MNHN USM 601 & CNRS UMR 5202, CP 50, 45 rue Buffon, 75005 Paris, France
[2]Neurobiologie de l'Apprentissage, de la Mémoire et de la Communication, CNRS UMR 8620, Bât. 446, Université Paris-Sud, 91405 Orsay Cedex, France
[3]Muséum National d'Histoire Naturelle, Département Ecologie et Gestion de la Biodiversité, MNHN USM 301& CNRS UMR 7179, CP 55, 57 rue Buffon, 75005 Paris, France

ABSTRACT

We review Seewave, new software for analysing and synthesizing sounds. Seewave is free and works on a wide variety of operating systems as an extension of the *R* operating environment. Its current 67 functions allow the user to achieve time, amplitude and frequency analyses, to estimate quantitative differences between sounds, and to generate new sounds for playback experiments. Thanks to its implementation in the *R* environment, Seewave is fully modular. All functions can be combined for complex data acquisition and graphical output, they can be part of important scripts for batch processing and they can be modified *ad libitum*. New functions can also be written, making Seewave a truly open-source tool.

INTRODUCTION

An important part of research in bioacoustics relies on the use of software dedicated to sound description and synthesis. This software should ideally combine (1) fast and automatic data acquisition, (2) direct statistical analysis of these data on the time, amplitude and frequency domains and (3) visualization of sounds through high-resolution graphics. In addition, these tools should be free, compatible with the main operating systems, and open source, thus allowing the user to modify or add new functions *ad libitum*. All these requirements are often difficult to combine in a single free device. *R* is a free software environment for data manipulation, calculation, statistical computing and graphic display. It also achieves a wide variety of tasks thanks to around 1,300 libraries or extensions (R Development Core

*Email: sueur@mnhn.fr

Team 2004). Working on a wide variety of platforms, *R* is a universal system that facilitates data exchanges and is commonly used in all biology fields. *R* seems therefore to be an excellent candidate for the development of a free and modular bioacoustics tool. We here report the main features of an *R* library we have named Seewave, which allows sound description and sound generation.

IMPLEMENTATION AND AVAILABILITY

Seewave is an extension of *R* that is available on UNIX platforms and similar systems (including FreeBSD, Linux), Windows and MacOS. It comes with its own LaTeX-like documentation format, which is used to supply comprehensive documentation, both on-line, in a number of formats, and in hardcopy. Each function is illustrated by numerous examples. The program is free and available from the official *R* package archive at http://cran.r-project.org/src/contrib/Descriptions/seewave.html. Seewave is licensed under the GNU General Public License.

RUNNING PRINCIPLE

Seewave is a command-line driven library devoted to sound analysis and synthesis (Figure 1). Sounds can be imported as ASCII or WAV files thanks to the *sound* package (Heymann 2007). Each task is achieved by calling a function to which different options, or arguments, can be specified. These options, which have default values, refer to analysis and graphical parameters. For instance, a spectrogram can be simply obtained with the following command:

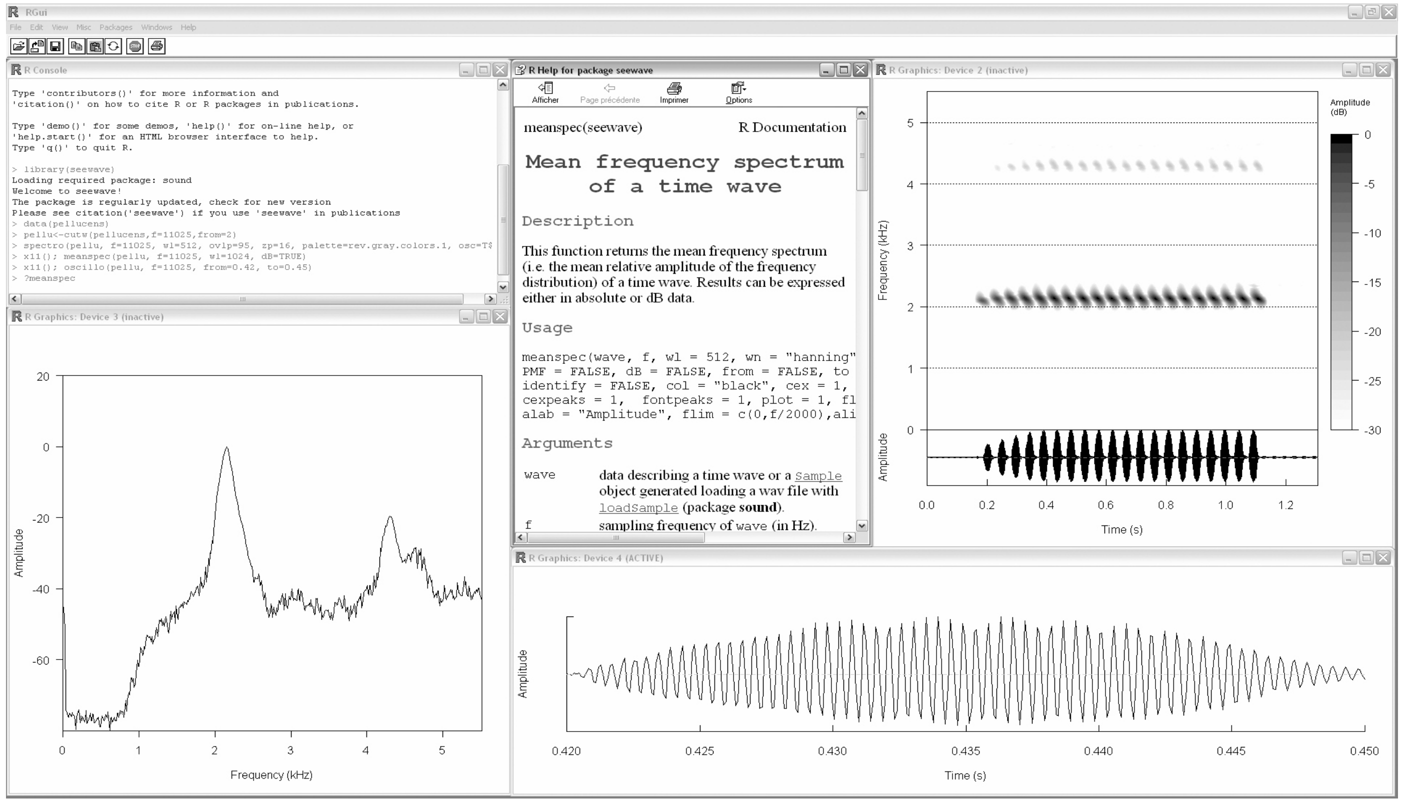
```
spectro(mysound)
```

By adding other arguments, this can be changed to:

```
spectro(mysound, wn = “Blackman”, wl = 1024, ovlp = 75,
palette = heat.colors)
```

where the Fourier window is now a Blackman window (wn) of length 1024 points (wl), there is a 75% overlap between successive windows (ovlp) and the colour palette of the spectrogram (palette) follows a heat colour scale (from light yellow to dark red).

There are several advantages to using such a command-line driven system rather than a classical GUI interface. Seewave and all *R* functions, including basic statistics functions, can be combined for fast process. In particular, all amplitude, temporal or frequency data can



```
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> library(seewave)
Loading required package: sound
Welcome to seewave!
The package is regularly updated, check for new version
Please see citation('seewave') if you use 'seewave' in publications
> data(pellucens)
> pellu<-cutw(pellucens,f=11025,from=2)
> spectro(pellu, f=11025, wl=512, ovlp=95, zp=16, palette=rev.gray.colors.1, osc=T$
> x11(); meanspec(pellu, f=11025, wl=1024, dB=TRUE)
> x11(); oscillo(pellu, f=11025, from=0.42, to=0.45)
> ?meanspec
```

meanspec(seewave) R Documentation

Mean frequency spectrum of a time wave

Description

This function returns the mean frequency spectrum (i.e. the mean relative amplitude of the frequency distribution) of a time wave. Results can be expressed either in absolute or dB data.

Usage

```
meanspec(wave, f, wl = 512, wn = "hanning"
PMF = FALSE, dB = FALSE, from = FALSE, to
identify = FALSE, col = "black", cex = 1,
cexpeaks = 1,  fontpeaks = 1, plot = 1, fl
alab = "Amplitude", flim = c(0,f/2000),ali
```

Arguments

wave	data describing a time wave or a Sample object generated loading a wav file with loadSample (package **sound**).
f	sampling frequency of wave (in Hz).

Figure 1. Screenshot of a Seewave session with the command-line window, the html help window, and three graphic windows opened (functions spectro, meanspec and oscillo). The sound is from a Tree Cricket *Oecanthus pellucens* stridulation. Main analysis parameters: sampling frequency = 11.025 kHz, FFT window length = 512 points, FFT window zero padding = 16 points, FFT window overlap = 95 %. See the Appendix for command-line details.

TABLE 1

Main functions of Seewave (version 1.4.8, 2008-04-14).

NAME	DESCRIPTION
Editing	
addsilw	Add or insert a silence section
cutw	Cut a sound section
deletew	Delete a sound section
mutew	Replace a sound section by silence
pastew	Paste a sound section
resamp	Resample
repw	Repeat a sound section
revw	Time reverse
rmoffset	Remove the offset
zapsilw	Remove silence sections
Time/Amplitude domain	
ama	Amplitude modulation analysis
corenv	Cross-correlation between two amplitude envelopes
diffenv	Difference between two amplitude envelopes
env	Amplitude envelope display
oscillo	Oscillogram display
th	Temporal entropy
timer	Automatic time measurements
Frequency domain	
autoc	Short-term autocorrelation to extract fundamental frequency
ccoh	Continuous coherence function between two sounds
ceps	Cepstrum or real cepstrum
cepstro	Cesptrogram
coh	Coherence between two sounds
corspec	Cross-correlation between two frequency spectra
covspectro	Covariance between two spectrograms
csh	Continuous spectral entropy
dfreq	Dominant frequency
diffspec	Difference between two frequency spectra
dynspec	Dynamic sliding spectrum
fund	Fundamental frequency track by cepstral analysis
ifreq	Instantaneous frequency through Hilbert transform
meanspec	Mean frequency spectrum
Q	Resonance quality factor of a frequency spectrum
sh	Spectral entropy
simspec	Similarity between two frequency spectra
spec	Frequency spectrum
specprop	Main statistical properties of a frequency spectrum
spectro	2D-spectrogram
spectro3D	3D-spectrogram
zc	Instantaneous frequency by zero-crossing
Synthesis and signal modification	
afilter	Amplitude filter
echo	Echo generator
fadew	Fade in and fade out
ffilter	Frequency filter
fir	Finite Impulse Response filter
lfs	Linear Frequency Shift
noise	Noise synthesis
pulse	Rectangle pulse synthesis
rman	Remove amplitude modulations
setenv	Set the envelope of a time wave to another wave
synth	Sound synthesis

be directly included in any statistical analysis, or any other biological analysis. For instance, the quartiles, mean, median, minimum, and maximum durations of signal periods can be automatically obtained and stored in a new object with a single command-line:

```
results.time<-summary(timer(mysound, threshold = 5,
plot = FALSE)$s)
```

Similarly 15 statistical descriptive parameters can be extracted from a frequency spectrum by simply calling:

```
results.frequency<-specprop(myspectrum)
```

Short scripts including loops or new functions can be quickly written and used for batch processing. The history of the commands can easily be saved, shared and called back for another session or for other sound analyses. Functions supplied with Seewave are open and can be modified by the user for his own needs. The user can then write new functions based on the available functions or create new functions. All these specifications ensure a full modularity of the software and a completely open source project.

FUNCTIONS

At present, Seewave (version 1.4.8, as of 2008-04-14) includes 61 functions. The main functions are summarised in Table 1. New functions and improvements are regularly added. The package also includes six sounds for illustrative purposes. Sound can be edited as oscillograms or amplitude envelopes in single or in multiple windows. An option can be set to move along the signal using a time slider. One function allows the automatic measurement of signal and silence durations in a sample by setting an amplitude threshold.

In the frequency domain, several functions based on the Fourier transform give information about spectral content. A frequency spectrum can be obtained for the whole sample, or at any location along it, by specifying a window size. The frequency content can also be dynamically tracked along the signal using a sliding window. A mean frequency spectrum can be computed with any desired window length and overlap. In particular, window length can be set to any even value (not only power of 2 values). Such spectra can be plotted in absolute or dB amplitude and spectrum amplitude peaks can be returned. For a given signal, the variations of the dominant, fundamental or instantaneous frequency according to time can also be plotted. The fundamental frequency of a harmonic series is detected by autocorrelation or cepstral methods, while the instantaneous

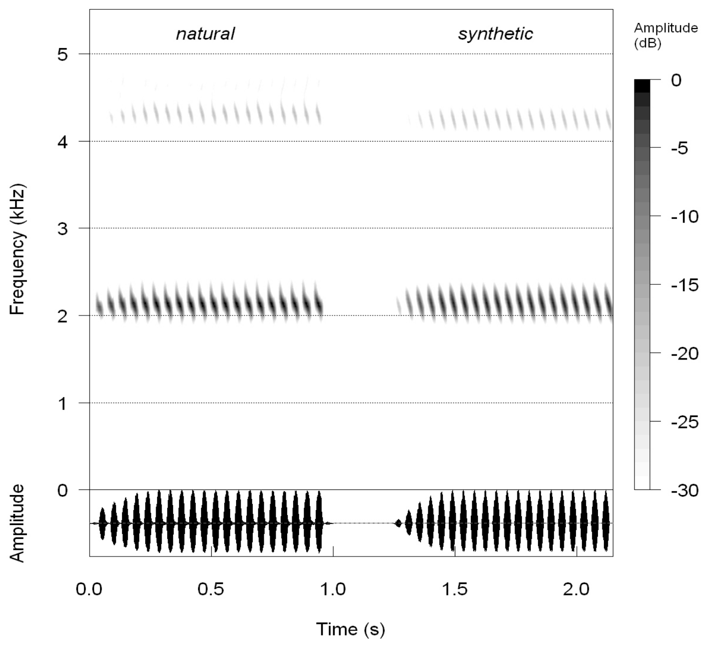


Figure 2. Natural and synthetic stridulation of the Tree Cricket *Oecanthus pellucens*. The synthesis is achieved with seven command-lines. See the Appendix for command-line details.

frequency is obtained by the zero-crossing method and Hilbert transform. 2D and 3D spectrograms can be obtained through sliding short-term Fourier transform calculations. For both representations, in addition to window analysis length and overlap of successive windows, zero-padding options allow the user to improve graphical representation and limit the uncertainty principle. Spectrograms can be viewed by applying to the amplitude any colour or grey palettes. Temporal and frequency similarity of two sounds can be assessed through different methods: cross-correlations, coherence, entropy or surface computation.

Another function of Seewave is dedicated to sound synthesis to design waves with any sinusoidal amplitude modulations and linear and/or sinusoidal frequency modulations. Any mathematical operations between different sounds can be achieved and complex sounds with or without harmonic structure can be obtained with simple additions of sine waves. Artificial white noise can be added to test specific signal-to-noise ratio in playback experiments. Slow amplitude modifications can be set following triangular or sine-like shape. It is also possible to apply the amplitude envelope of a sound to another one even if the sounds differ in duration. Effects like fade-in, fade-out and echoes

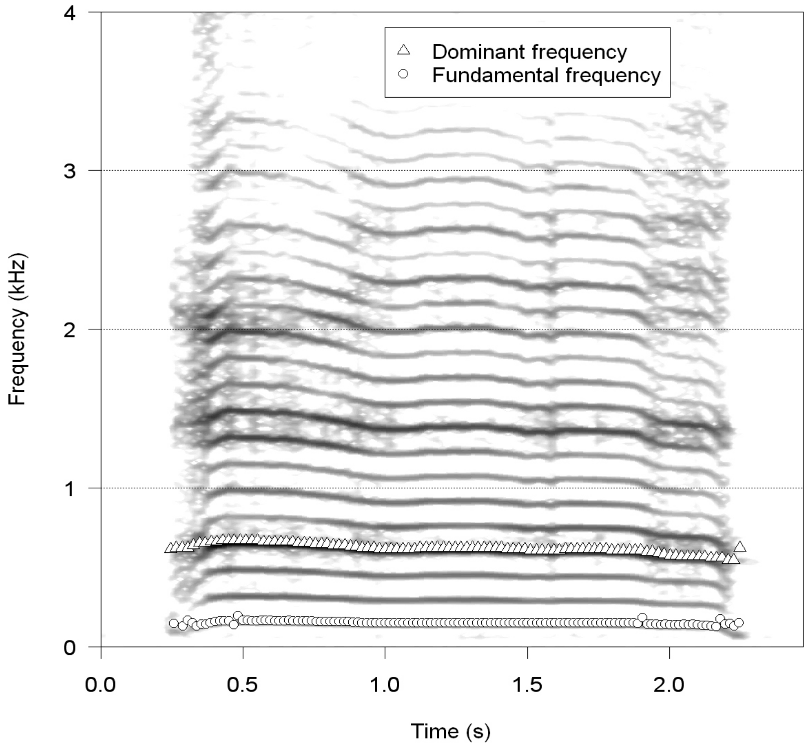


Figure 3. Graphical combination of analyses of a domestic sheep *Ovis aries* bleat on a single plot. The figure is obtained with three functions: a spectrogram (function spectro), a track of the fundamental frequency obtained by cepstral analysis (function fund) and a track of the dominant frequency obtained (function dfreq). Main analysis parameters: sampling frequency = 8 kHz, FFT window length = 512 points, FFT window overlap = 50 %. Recording by Frédéric Sebe. See the Appendix for command-line details.

can also be generated and signals can be modified through amplitude filters, FIR filters and linear frequency shifts. An example showing how to mimic a tree-cricket stridulation is given in Figure 2.

GRAPHICS

Seewave benefits from the *R* high-level graphical specifications. It is then possible to fully modify the graphics, and to combine the graphical displays of different functions in a single display with several layers overlaid or in multi-framed output. Figure 3 shows the analysis of domestic sheep *Ovis aries* bleat with three graphics overlaid whereas Figure 4 illustrates a multi-framed graphical output with the analysis of a Blue-footed Booby *Sula nebouxii* display call. Such graphics can

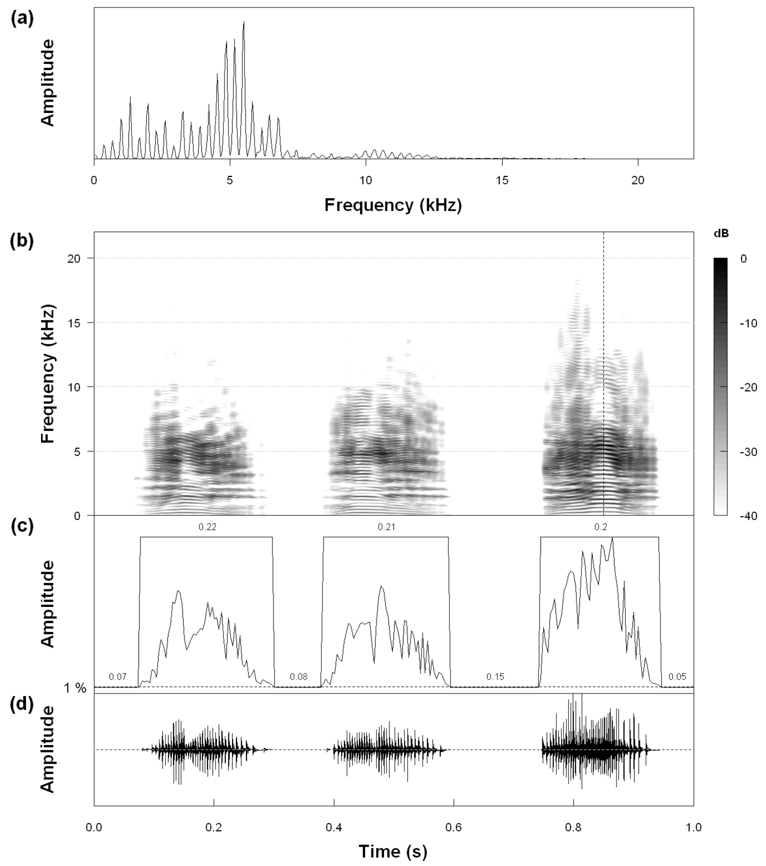


Figure 4. Graphical combination of analyses in a multi-framed output. Analysis of three display calls of Blue-footed Booby *Sula nebouxii*. (a) spectrum (function spec) calculated at the vertical dotted line position in (b), (b) spectrogram (function spectro), (c) amplitude envelope and automatic temporal measurements (function timer), (d) oscillogram (function oscillo). Main analysis parameters: sampling frequency = 44.1 kHz, FFT window length = 1024 points, FFT window overlap = 85 %. See the Appendix for command-line details.

be saved at a high resolution in several formats, including eps and pdf. All these features allow the user to produce figures ready for editing. In addition, by exploiting the real-time rendering device-driver system (Adler & Murdoch 2007), it is possible to create short movies of 3D sound representations for online or talk presentations.

LIMITATIONS

To use Seewave, it is first necessary to learn *R*-language basics. Even if not very demanding, this can be off-putting to people not familiar with command-line driven software. Seewave is also probably not the best choice to edit sound. Users might prefer GUI software where it is easier to explore the signal in the time domain. For now, Seewave can neither identify nor automatically measure temporal sub-elements such as those related to rhythm or tempo (syllables, pulses). Finally, because *R* uses the RAM memory to run, computation is fast but the RAM capacity can limit the size of the files to be analysed with Seewave. For instance, the upper limit for a computer with 1024 MB RAM is around 10 minutes sampled at 44.1 kHz.

DEVELOPMENT

Seewave is regularly updated, a new version being released about every semester. To reduce its present limitations, the next versions should include better temporal analysis, Wigner-Ville transform, LPC (Linear Predictive Coding) analysis and other tools like algorithms estimating source localization and tracking. As an open-source project, development should also directly come from users whom are warmly invited to develop and share their own functions.

ACKNOWLEDGMENTS

JS and CS are deeply indebted to Michel Baylac (Muséum national d'Histoire naturelle, Paris, France) and Emmanuel Paradis (Institut de Recherche pour le Développement, Montpellier, France) for their enthusiastic *R* lectures. We would also like to thank Martin Maechler who wrote one function argument (ETH Zurich, Switzerland). We thank Frédéric Sebe (INRA, Tours, France) for the use of his sheep bleat recording, James FC Windmill (University of Bristol, UK) for his useful corrections on the manuscript, and two anonymous referees for their helpful comments.

REFERENCES

Adler, D. & Murdoch, D. (2007). rgl: 3D visualization device system (OpenGL). R package version 0.76. http://rgl.neoscientists.org

Heymann, M. (2007). sound: a sound interface for R. R package version 1.1. http://www.MatthiasHeymann.de

R Development Core Team (2008). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. http://www.R-project.org

Received 19 November 2007, revised 28 February 2008 and accepted 12 March 2008.

APPENDIX

R scripts used to generate the figures. Comments are preceded by a sharp (#) symbol

Figure 1

# call the song of *Oecanthus pellucens* coming attached to Seewave

```
data(pellucens)
```

# cut a section and store it in a new object

```
pellu<-cutw(pellucens,f=11025,from=2)
```

# generate a spectrogram with a Fourier window of 512 points length, 95% overlap, and 16 points zero-padding

# a grey level palette is applied and an oscillographic display is requested

```
spectro(pellu, f=11025, wl=512, ovlp=95, zp=16, palette=rev.
gray.colors.1, osc=TRUE)
```

# open a second graphic window and generate a mean spectrum with a Fourier window of length 1024 points

```
x11(); meanspec(pellu, f=11025, wl=1024, dB=TRUE)
```

# open a third window and display an oscillograms of a section

```
x11(); oscillo(pellu, f=11025, from=0.42, to=0.45)
```

# open the html help for the function meanspec

```
?meanspec
```

Figure 2

# call the song of *Oecanthus pellucens* coming attached to Seewave

```
data(pellucens)
```

# store in a new object the sampling frequency to be repeatedly used

```
f<-11025
```

# cut a section and store it in a new object

```
natural<-cutw(pellucens, f=f, from=2.15, to=3.15)
```

# synthesis of a 2300 Hz chirp with a -315 Hz FM, a duration of 0.03 s and a sine-like shape amplitude envelope

```
s1<-synth(d=0.03, f=f, cf=2300, fm=c(0,0,-315), shape="sine")
```

# similar synthesis than s1 but at 2300*1.9 = 4370 Hz

```
s2<-synth(d=0.03, f=f, cf=2300*1.9, fm=c(0,0,-315),
shape="sine")
```

# addition of s1 and 0.12*s2 to follow relative amplitude between the frequency bands; amplitude normalisation

```
s3<-s1+(0.12*s2); s4<-s3/max(s3)
```

# add a short silence period and repetition of the chirp

```
s5<-addsilw(s4, f=f, d=0.015); s6<-repw(s5, f=f, times=20)
```

# add a fade in effect with cosine-like shape

```
s7<-fadew(s6, f=f, din=0.25, shape="cos")
```

# paste natural and synthetic sounds in a new object

```
result1<-pastew(s7,pellu,f=f)
```

# insert a 0.25 s silence period between natural and synthetic sounds

```
result2<-addsilw(result1, f=f, at=1, d=0.25)
```

# display a spectrogram with labels

```
spectro(result2, f=f, wl=256, ovlp=95, osc=TRUE, palette=rev.
gray.colors.1)
mtext(c("natural","synthetic"), side=3, at=c(0.2,0.7), line=1.5,
font=3)
```

Figure 3

# call a bleat of *Ovis aries* coming attached to Seewave

```
data(sheep)
```

# display a spectrogram

```
spectro(sheep, f=8000, ovlp=75, zp=16, scale=FALSE,
palette=rev.gray.colors.2, collevels=seq(-45,0,1))
```

# create a first layer

```
par(new=TRUE)
```

# dominant frequency track is computed and is overlaid onto the spectrogram

```
dfreq(sheep, f=8000, wl=1024, ovlp=85, type=”p”, pch=24,
bg=”white”, ann=FALSE)
```

# create a second layer

```
par(new=TRUE)
```

# compute fundamental frequency track and overlay it onto the spectrogram

```
fund(sheep, f=8000, wl=128, fmax=200, threshold=2, type=”p”,
pch=21, bg=”white”, ann=FALSE)
```

# add the symbol legend

```
legend(1,3.9, c(“Dominant frequency”,”Fundamental frequency”),
pch=c(24,21), bty=0)
```

Figure 4

# import a *wav* file of *Sula nebouxii* display calls and store it in a object. The file can be sent upon request

```
sula<-loadSample(“sula.wav”)
```

# divide the graphic windows in sub-frames and change the margins

```
layout(matrix(c(1,0,2,5,3,0,4,0), nc=2, byrow=T), widths=c(9,
1), heights=c(2,2.5,1.5,1.5)); par(mar=c(4.5,4,2,1),
oma=c(0,3,0,0), cex.lab=1.5)
```

# display a spectrum at 0.85 s of length 1024 points

```
spec(sula, wl=1024, at=0.85, font.lab=2)
```

# change the margins

```
par(mar=c(0,4,1,1))
```

# display a spectrogram

```
spectro(sula, wl=1024, ovlp=85, scale=F, colgrid=”grey”,
collevels=seq(-40,0,1), zp=6, plot.title=title(main = “”,
ylab=”Frequency (kHz)”, font.lab=2), axisX=F, palette=rev.
gray.colors.2)
```

# add a vertical dotted line onto the spectrogram to show where the spectrum has been computed

```
abline(v=0.85, lty=2)
```

# change the margins

```
par(mar=c(0,4,0,1))
```

# compute automatically the duration of the calls and the inter-call silences

```
timer(sula, f=44100, threshold=1, smooth=170, xlab=””,
xaxs=”i”, xaxt=”n”, font.lab=2, colval=”black”)
```

# change the margins

```
par(mar=c(5,4,0,1))
```

# display an oscillogram

```
oscillo(sula, bty=”o”, fontlab=2)
```

# change the margins

```
par(mar=c(0,0.5,3,4))
```

# add the spectrogram scale

```
dBscale(collevels=seq(-40,0,1), side=4, textlab="dB",
cexlab=0.65, fontlab=2, palette=rev.gray.colors.2)
```

# add the name of the sub-plots

```
mtext(c("(a)","(b)","(c)","(d)"), side=2, font=2, line=0.5,
at=c(0.96,0.71,0.39,0.19), outer=TRUE, las=1, cex=1.1)
```