I/O of sound with R

Jérôme Sueur [*]

June 28, 2012

This document shortly details how to import and export sound with R using the packages seewave, sound, tuneR and audio.

Contents

[*]Muséum national d'Histoire naturelle, Paris, France – http://sueur.jerome.perso.neuf.fr

1 In

The main functions of `seewave (>1.5.0)` can use different classes of objects to analyse sound:

- usual classes (numeric `vector`, numeric `matrix`),
- time series classes (`ts`, `mts`),
- sound-specific classes (`Wave`, `Sample`, `audioSample`).

1.1 Non specific classes

1.1.1 Vector

Any muneric vector can be treated as a sound if a sampling frequency is provided in the `f` argument of seewave functions. For instance, a 440 Hz sine sound (A note) sampled at 8000 Hz during one second can be generated and plot following:

```
> s1<-sin(2*pi*440*seq(0,1,length.out=8000))
> is.vector(s1)
```

```
[1] TRUE
```

```
> mode(s1)
```

```
[1] "numeric"
```

```
> library(seewave)
> oscillo(s1,f=8000)
```

1.1.2 Matrix

Any single column matrix can be read but the sampling frequency has to be specified in the seewave functions.

```
> s2<-as.matrix(s1)
> is.matrix(s2)
```

```
[1] TRUE
```

```
> dim(s2)
```

```
[1] 8000    1
```

```
> oscillo(s2,f=8000)
```

If the matrix has more than one column, then the first column only will be considered.

```
> x<-rnorm(8000)
> s3<-cbind(s2,x)
> is.matrix(s3)
```

```
[1] TRUE
```

```
> dim(s3)
```

```
[1] 8000    2
```

```
> oscillo(s3,f=8000)
```

1.2 Time series

The class `ts` and the related functions `ts`, `as.ts`, `is.ts` can be used also for sound. Here follows the command to similarly generate a time series corresponding to a 440 Hz sine sound sampled at 8000 Hz during one second:

```
> s4<-ts(data=s1, start=0, frequency=8000)
> str(s4)
```

```
 Time-Series [1:8000] from 0 to 1: 0 0.339 0.637 0.861 0.982 ...
```

To generate a 0.5 second random noise:

```
> s4<-ts(data=runif(4000), start=0, end=0.5, frequency=8000)
> str(s4)
```

```
 Time-Series [1:4001] from 0 to 0.5: 0.442 0.641 0.655 0.219 0.714 ...
```

The length of `s4` is not 4000 but 4001. Data are actually recycled, `s4[4001]` being the same as `s4[1]`. The functions `frequency` and or `deltat` return the sampling frequency (f) and the time resolution (Δt) respectively:

```
> frequency(s4)
```

```
[1] 8000
```

```
> deltat(s4)
```

```
[1] 0.000125
```

As the frequency is embedded in `ts` objects, there is no need to specify it when using seewave functions:

```
> oscillo(s4)
```

In the case of multiple time series, seewave functions will consider the first series only:

```
> s5<-ts(data=s3,f=8000)
> class(s5)
```

```
[1] "mts" "ts"
```

```
> oscillo(s5)
```

1.3 Specific sound classes

There are three object classes corresponding to the binary `wav` format or to the compressed `mp3` format:

- the class `Wave` of the package tuneR,
- the class `Sample` of the package sound,
- the class `audioSample` of the package audio

1.3.1 Wave class (package tuneR)

The class `Wave` comes with the package tuneRmanaged by Uwe Ligges. This S4 class includes different slots with the data (left or right channel), the sampling frequency (or rate), the number of bits (8 /16 /24 /32) and the type of sound (mono /stereo). High sampled sound (*i.e.* > 44100 Hz) can be read.
The function to import `.wav` files from the hard-disk is `readWave`:

```
> s6<-readWave("mysong.wav")
> s6
```

```
Wave Object
        Number of Samples:      480000
        Duration (seconds):     60
        Samplingrate (Hertz):   8000
        Channels (Mono/Stereo): Mono
        Bit (8/16/24/32):       16
```

The other advantage of using `readWave` is for reading part of long files. It is indeed possible to import only a section of the `.wav` file using the arguments `from` and `to` and by specifying the time units with the arguments `units`. The units can be turned to ”samples”, ”minutes” or ”hours”. For instance, to read only the section starting at 1s and ending at 5s of the file ”mysong.wav” :

```
> s7<-readWave("mysong.wav", from = 1, to = 5, units = "seconds")
> s7
```

```
Wave Object
        Number of Samples:      32000
        Duration (seconds):     4
        Samplingrate (Hertz):   8000
        Channels (Mono/Stereo): Mono
        Bit (8/16/24/32):       16
```

Note that `.mp3` files can be imported as a `Wave` object with the function `readMP3`.
To get information regarding the object (sampling frequency, number of bits, mono /stereo), it is necessary to use the indexing of S4 object classes:

```
> s7@samp.rate
```

```
[1] 8000
```

```
> s7@bit
```

```
[1] 16
```

```
> s7@stereo
```

```
[1] FALSE
```

A property not apparent in these call is that `readWave` does not normalise the sound. Values describing the sound will be included between $\pm 2^{bit-1}$:

```
> range(s7@left)
```

```
[1] -32764  32764
```

1.3.2 Sample class (package sound)

The class `Sample` and the related `as.Sample` and `is.Sample` functions belong to the package `sound`written by Matthias Heymann. Like `Wave`, the `Sample` class is a list containing information regarding data, sampling frequency, bits and the type of sound (mono/stereo).
To read a `.wav` file stored on the hard-disk using `loadSample`:

```
> s8<-loadSample("mysong.wav")
> s8
```

```
type      : mono
rate      : 8000 samples / second
quality   : 16 bits / sample
length    : 480000 samples
R memory  : 1920000 bytes
HD memory : 960044 bytes
duration  : 60 seconds
```

All kinds of .wav files are supported: mono, stereo, 8 or 16 bits per sample, at any sampling frequency above 1000 Hz. The duration and the sampling frequency of a `Sample` object can be obtained using dedicated functions or list indexing:

```
> rate(s8)

[1] 8000

> s8$rate

[1] 8000

> duration(s8)

[1] 60

> s8$duration

NULL
```

Unlike `readWave`, `loadSample` changes the limits of the sound between ±1:

```
> range(s8$sound[1,])

[1] -0.9999695  0.9999695
```

seewave functions which return a value describing a sound have a special argument named `output` that can be set to `"matrix"`, `"Sample"`, `"audioSample"`, `"Wave"` or `"ts"`. This allows to control the class of the returned object.

```
> s9<-repw(s2,f=8000,times=2,output="Sample")
> s9

type      : mono
rate      : 8000 samples / second
quality   : 16 bits / sample
length    : 16000 samples
R memory  : 64000 bytes
HD memory : 32044 bytes
duration  : 2 seconds
```

However, matrix objects are easier to handle and can be used with functions not dedicated to sound. It might be then better to let the arguments `Sample` and `Wave` set to `FALSE` and to store the sampling frequency `f` in an object:

```
> f<-8000
> oscillo(s2,f)
```

1.3.3 audioSample class (package audio)

The package audio, developed by Simon Urbanek, is another option to handle `.wav` files. Sound can be imported using the function `load.wave`. The class of the resulting object is `audioSample` which is essentially a numeric vector (for mono) or numeric matrix with two rows (for stereo). The sampling frequency and the resolution can be called as attributes :

```
> s10<-load.wave("mysong.wav")
> head(s10)

sample rate: 8000Hz, mono, 16-bits
[1]  0.0000000  0.7070923  0.9999695  0.7070923  0.0000000 -0.7071139
```

```
> s10$rate
```

```
[1] 8000
```

```
> s10$bits
```

```
[1] 16
```

The main advantage of the package audio is that sound can be directly acquired within an R session. This is achieved by first preparing a vector of NA and then using the function `record`. For instance, to get a mono sound of 5 seconds sampled at 16 kHz :

```
> s11 <- rep(NA_real_, 16000*5)
> record(s11, 16000, 1)
```

A recording session can be controled using three complementary functions : `pause`, `rewind`, and `resume` (see 4.1.3). See the documentation of audio for details regarding the control of audio drivers: http://www.rforge.net/audio/.

2 Out

2.1 .txt format

For a maximal compatibility with other sound softwares, it can be useful to save a sound as a simple `.txt` file. This can be done using the function `export` with the argument `header=FALSE`. By default, the name of the object is used to name the `.txt` file. The following commands will write a file ”tico.txt” on the hard-disk.

```
> data(tico)
> export(tico, f=22050, header=FALSE)
```

For Windows users, the software Goldwave © can be helpful when handling long sound files or large number of files. To export a sound as a `.txt` file that can be directly read by Goldwave ©, the same function can be used but with the default argument `header=TRUE`. seewavewill automatically add the header needed. Hereafter the name of the exported file is changed using the argument `filename`:

```
> export(tico, f=22050, filename="tico_Gold.txt")
```

Any header can be specified for a connection with other softwares. For instance, if an external software needs the header ”f=sampling frequency; ch=left”:

```
> export(tico, f=22050, filename="tico_ext.txt",
+ header="f=22050; ch=left")
```

2.2 .wav format

tuneR, soundand audiohave a function to write `.wav` files: `writeWave`, `saveSample`, and `save.wave` respectively. Within seewave, the function `savewav`, which is based on `writeWAve`, can be used to save data as `.wav`. By default, the name of the object will be used for the name of the `.wav` file:

```
> savewav(tico, f=22050)
```

As seen before, if the object to be saved is of class `ts`, `Sample` or `Wave`, there is no need to specify the argument `f`. Here we use the argument `filename` to change the name of the `wav` file:

```
> ticofirst<-cutw(tico, f=22050, to=0.5, output="Wave")
> savewav(ticofirst, filename = "tico_firstnote.wav")
```

2.3 .flac format

Free Lossless Audio Codec (FLAC) is a file format by Josh Coalson for lossless audio data compression. FLAC reduces bandwidth and storage requirements without sacrificing the integrity of the audio source. Audio sources encoded to FLAC are typically reduced in size 40 to 50 percent. See the flac webpage for details .
.flac format cannot be used as such with R. However, the function wav2flac allows to call FLAC software directly from the console. FLAC has therefore to be installed on your OS. If you have a .wav file you wish to compress into .flac, call:

```
> wav2flac("tico_firstnote.wav", overwrite=TRUE)
```

To compress a .wav file into .flac, the argument reverse has to be set to TRUE:

```
> wav2flac("tico_firstnote.flac", reverse=TRUE)
```

3 Mono and stereo

Both Sample and Wave classes can handle stereo files. There are some specific functions regarding mono/stereo type. Both libraries include functions with the same name.

3.1 Wave class

To generate a stereo sound, two mono sounds are first created using sine, a function that returns a Wave object, and then combined using stereo:

```
> left<-sine(440, bit = 16)
> right<-sine(2000, bit = 16)
> s12<-stereo(left,right)
> s12
```

```
Wave Object
        Number of Samples:      44100
        Duration (seconds):     1
        Samplingrate (Hertz):   44100
        Channels (Mono/Stereo): Stereo
        Bit (8/16/24/32):       16
```

To go back to a mono file taking the left channel only:

```
> s13<-mono(s12,"left")
```

The function channel do roughly the same as it extracts one or more channels. To get this time the right channel:

```
> s14<-channel(s12,"right")
```

And eventually, the S4 indexing can be used to do it "manually". In this particular case, the returned object will be of class vector.

```
> s13<-s12@left
> is.vector(s13)
```

```
[1] TRUE
```

```
> s14<-s12@right
> is.vector(s14)
```

```
[1] TRUE
```

3.2 Sample class

With the `Sample` objects, the syntax is almost similar. To generate stereo sound, the function `synth` with the argument `Sample=TRUE` is called twice:

```
> left<-synth(cf=440,d=5,f=8000,output="Sample")
> right<-synth(cf=2000,d=5,f=8000,output="Sample")
> s13<-stereo(left,right)
> s13
```

```
type      : stereo
rate      : 8000 samples / second
quality   : 16 bits / sample
length    : 40000 samples
R memory  : 320000 bytes
HD memory : 160044 bytes
duration  : 5 seconds
```

To get either the left or right channel:

```
> s14<-left(s13)
> s14
```

```
type      : mono
rate      : 8000 samples / second
quality   : 16 bits / sample
length    : 40000 samples
R memory  : 160000 bytes
HD memory : 80044 bytes
duration  : 5 seconds
```

```
> s15<-right(s13)
> s15
```

```
type      : mono
rate      : 8000 samples / second
quality   : 16 bits / sample
length    : 40000 samples
R memory  : 160000 bytes
HD memory : 80044 bytes
duration  : 5 seconds
```

List indexing allows extracting manually the data:

```
> s16<-s13$sound[1, ]
> is.vector(s16)
```

```
[1] TRUE
```

```
> s17<-s13$sound[2, ]
> is.vector(s17)
```

```
[1] TRUE
```

The `mirror` function interchanges left and right channels and `panorama` control the panaroma through a `pan` argument which varies between -50 and +50.

```
> s13<-mirror(s13)
> s14<-panorama(s13, pan=30)
```

4 Play sound

4.1 Specific functions

4.1.1 Wave class

`Wave` objects can be played with `play` of `tuneR`:

```
> play(s6)
```

It may happen that the default players of the function `play` are not installed on the OS. Three functions can help in setting the media player: `findWavPlayer` returns the most common system commands on the OS, `WavPlayer` returns the command that is currently used by `play`, `setWavPlayer` is used to define the command to be used by `play`. For instance, if Audacious is the player to use (Linux OS):

```
> setWavPlayer("audacious")
```

4.1.2 Sample class

For the class `Sample` of the package `tuneR` the functions are basically the same. The function `play`:

```
> play(ticofirst)
> play("mysong.wav")
```

and the functions `WavPlayer`, `setWavPlayer`, `getWavPlayer` to set the sound player of your OS.

4.1.3 audioSample class

The package `audio` has similarly a function `play` but also have three useful functions to control recording and playback:

- `pause` that stops audio recording or playback,
- `rewind` that rewinds audio recording or playback, *i.e.*, makes the beginning of the source (or target) object the current audio position,
- `resume` that resumes previously paused audio recording or playback.

4.1.4 Other classes

The package `seewave`includes listen a function based on `play` of `soundR` but accepting all specific and non-specific classes and with two arguments (`from` and `to`) to listen only a section of a sound object:

```
> listen(s1, f=8000, from=0.3, to=7)
> listen(s13, from=0.3, to=4)
```

4.2 System command

The call of an external sound player can also be achieved using directly `system` that allows invoking directly the system command. For instance, to play a sound with Audacity (Linux OS):

```
> system("audacity mysong.wav")
```

To run a sound player with Windows is slightly more tricky as the complete path to the .exe file has to be specified and paster has to be invoked to combine both program and file names:

```
> system(paste('"C:/Program Files/GoldWave/GoldWave.exe"', 'mysong.wav'))
```

5 Summary

Here is a temptative of summary of main `R` functions used for sound input and output:

	Input	**Output**	**Mono/Stereo**	**Play**	**Object**
tuneR	readWave	writeWave	mono, stereo	play	`Wave`
sound	loadSample	saveSample	mono, stereo	play	`Sample`
audio	load.wave, record	save.wave	mono, stereo	play, pause, resume, rewind	`audioSample`
seewave	–	export, savewav	–	listen	`vector`, `matrix`, `ts`, `mts`, `Wave`, `Sample`, `audioSample`