ugr | Universidad de Granada

PRÁCTICA 5

Experimentación con el sistema de salida de sonido

Periféricos y Dispositivos de Interfaz Humana

Autor:
Clara Sola Ruiz

Índice

1. Introducción

En esta práctica se ha trabajado con señales de sonido utilizando programación en R con el objetivo de familiarizarse con los archivos de audio. Los objetivos principales de esta práctica son identificar y representar gráficamente la forma de la onda de distintas señales sonoras, conocer la estructura interna de un archivo de sonido en formato WAW, y comprender y manipular los parámetros fundamentales que definen a una señal.

Para ello se ha desarrollado la práctica utilizando el lenguaje de programación *R,* con paquetes como *tuneR* y *seewave.*

2. Ejercicios mínimos

A continuación pasamos a resolver las cuestiones relacionadas con la práctica.

2.1. Crear dos ficheros de sonido WAW

En este primer ejercicio se nos pide que creemos dos ficheros de sonido, uno con nuestro nombre y otro con el apellido. Para ello hemos utilizado la página web https://speechgen.io/ que nos permite crear esos audios.

En la imagen se puede observar que poniendo nuestro nombre, se generará el audio que se puede encontrar en la carpeta de github. Para cargar los sonidos que hemos creado en nuestro programa en R, utilizamos los siguiente comandos:

```
nombre <- readWave("nombre.wav")
apellido <- readWave("apellido.wav")
```

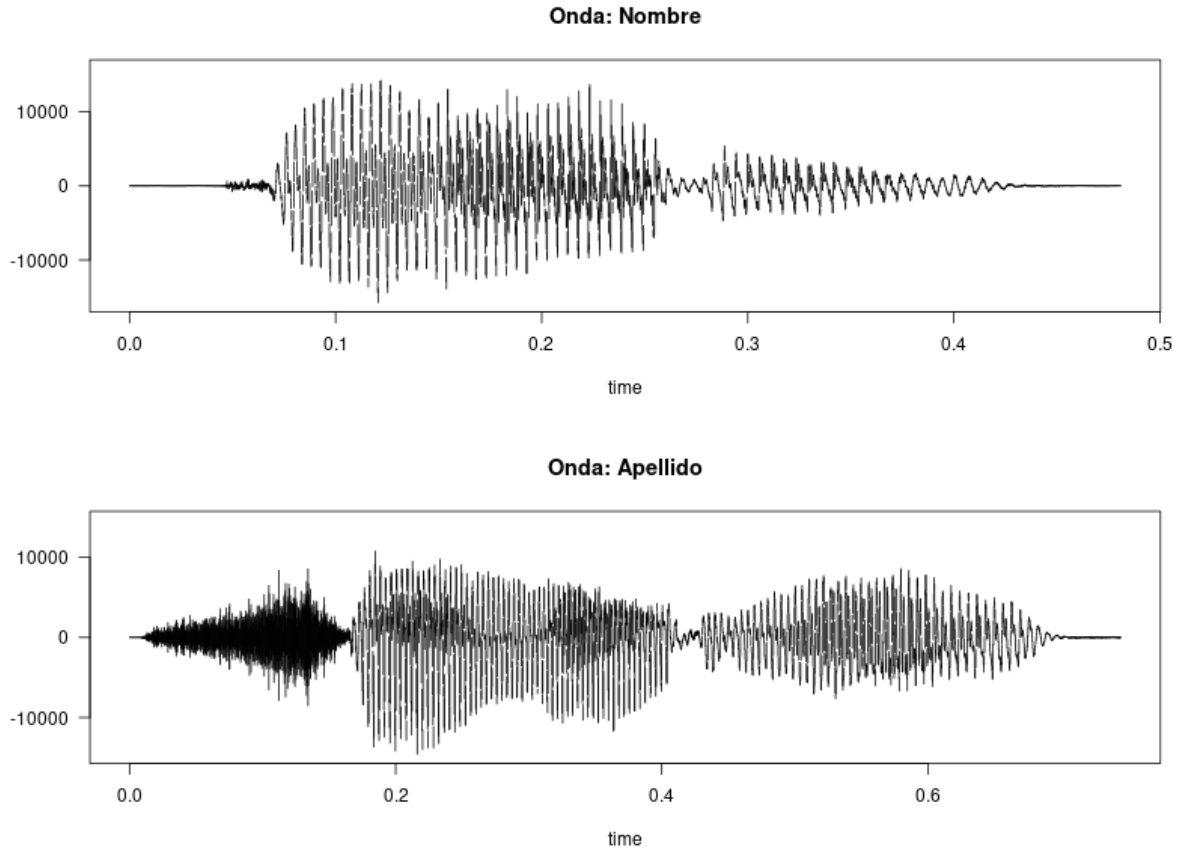
readwave sirve para leer archivos de sonido en formato WAV y cargar su contenido en R como un objeto de clase Wave.

2.2. Leer los dos ficheros de sonido creados y dibujar la forma de onda de ambos

En este ejercicio tenemos que dibujar la onda que generan los dos archivos de sonido que hemos creado con nuestro nombre y apellido. Para ello utilizamos el siguiente código:

```
# Guardar las formas de onda en un archivo PNG
png("formas_de_onda.png", width = 800, height = 600)

# Dibujar las formas de onda en el archivo
par(mfrow=c(2,1))
plot(nombre, main = "Onda: Nombre")
plot(apellido, main = "Onda: Apellido")

# Cerrar el archivo
dev.off()
```

png("formas_de_onda.png", width = 800, height = 600), este comando nos permite crear un archivo en donde se va a guardar la imagen de las ondas creadas. Con *par(mfrow=c(2,1))*, dividimos la imagen en dos para que se puedan ver las dos ondas una encima de la otra. Y con el comando *plot*, podemos pintar nuestra gráfica. El resultado es el siguiente:



El nombre es Clara y apellidos Sola Ruiz. Quizás la gráfica en el apellido se ve con más ondas al principio debido a la pronunciación de la “s”.

2.3. Obtener la información de las cabeceras de ambos sonidos.

En el tercer ejercicio se nos pide obtener la información de las cabeceras de los archivos de sonido. Para ello utilizamos el comando *str.*

```
# --------- Paso 3: Cabeceras ---------
str(nombre)
str(apellido)
```

La función *str()* en R sirve para mostrar de forma resumida y estructurada el contenido interno de un objeto. Se puede ver la siguiente salida una vez ejecutamos el archivo:

```
> source("practica_5.r")
Formal class 'Wave' [package "tuneR"] with 6 slots
  ..@ left     : int [1:23087] 0 -2 -3 -2 -1 -1 -2 -4 -4 -5 ...
  ..@ right    : num(0)
  ..@ stereo   : logi FALSE
  ..@ samp.rate: int 48000
  ..@ bit      : int 16
  ..@ pcm      : logi TRUE
Formal class 'Wave' [package "tuneR"] with 6 slots
  ..@ left     : int [1:35759] 1 2 2 1 0 0 -1 -2 -2 -3 ...
  ..@ right    : num(0)
  ..@ stereo   : logi FALSE
  ..@ samp.rate: int 48000
  ..@ bit      : int 16
  ..@ pcm      : logi TRUE
>
```

Se muestra que nombre y apellido son objetos de tipo “Wave” con 6 campos (slots).

- left contiene 23.087 (35759) valores enteros que demuestran la amplitud de la onda sonora del canal izquierdo o único en este caso porque right está a 0, lo que nos indica también que el audio es mono
- stereo está a FALSE, por lo que es sonido monoaural
- samp.rate es el número de muestras por segundo, en este caso 48000 en ambos casos.
- bit a 16, significa una resolución de 16 bits por muestra
- pcm : TRUE, significa que el archivo usa codificación PCM (Pulse Code Modulation), un formato estándar sin compresión.

2.4. Unir ambos sonidos en uno nuevo para escuchar el nombre y apellido correctamente

En este ejercicio se nos pide que unamos los dos audios que hemos creado anteriormente para poder escuchar nombre y apellidos juntos. Para ello vamos a utilizar el siguiente comando:

```
# --------- Paso 4: Unir sonidos ---------
completo <- bind(nombre, apellido)
```

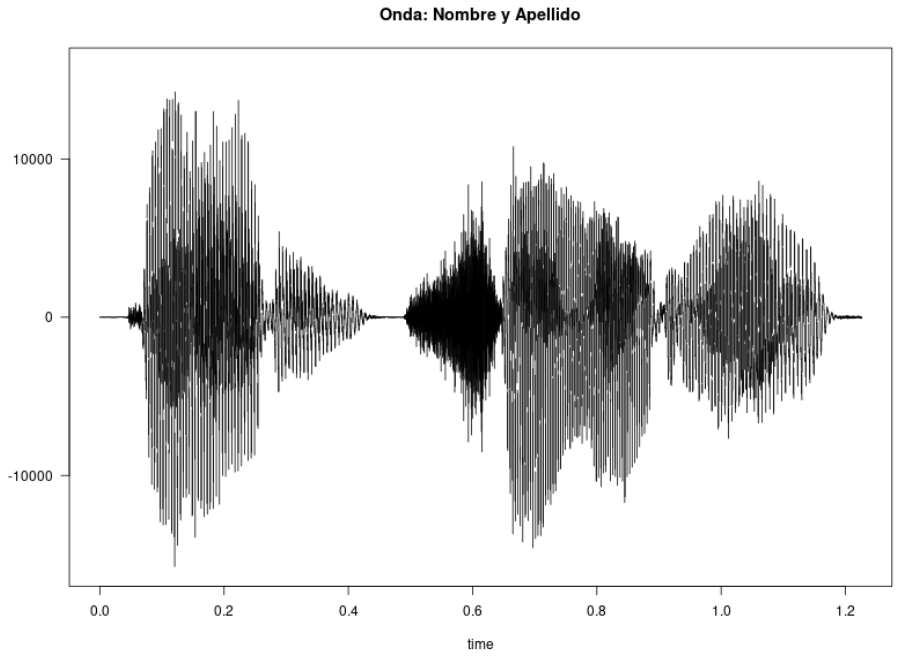
Utilizando *bind()* concatenamos dos objetos Wave con las mismas características.

2.5. Dibujar la forma de onda de la señal y reproducir el sonido resultante (una vez unidos)

En este apartado, se nos pide dibujar la forma de la onda que genera el nombre y el apellido, para ello siguiendo los mismos pasos que en el apartado 2.2 generamos nuestra onda. Después para poder escuchar el audio hemos utilizado el comando *system* que nos permite reproducir el audio dentro de R.

```
# --------- Paso 5: Dibujar forma de onda y escuchar el audio ---------
png("onda_nombre_apellido", width = 800, height = 600)
plot(completo, main = "Onda: Nombre y Apellido")
dev.off()

system("aplay basico.wav")
```

Conseguimos la siguiente onda:

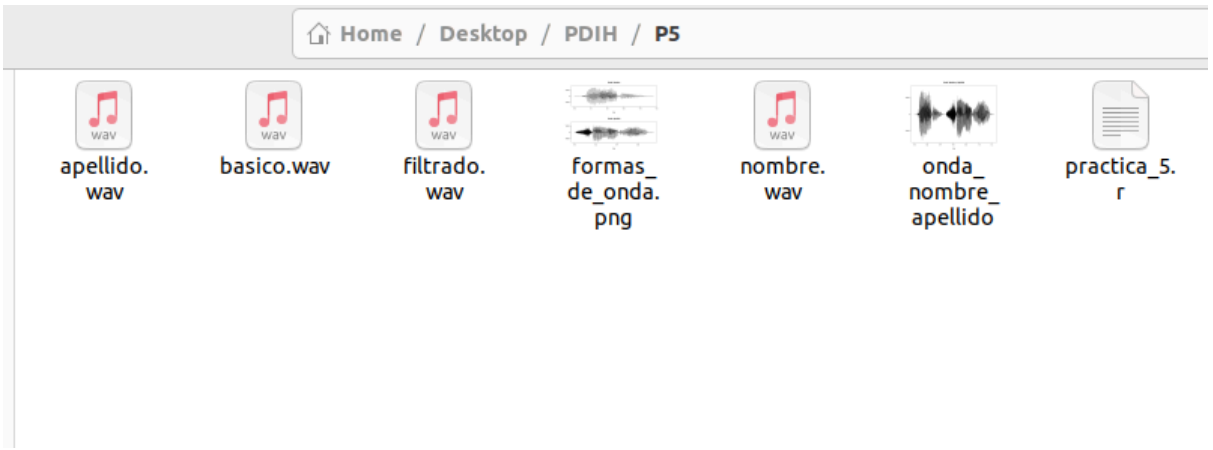


2.6. Guardar el archivo en un “basico.waw”

Para ello hemos utilizado el comando siguiente:

```
writeWave(completo, "basico.wav")
```

writewave nos permite guardar este archivo con el nombre que deseemos. Podemos ver en la carpeta que se ha guardado:



3. Ejercicios opcionales

A continuación se pasará a resolver los ejercicios opcionales:

3.1. Pasar un filtro para eliminar frecuencias entre 10.000Hz y 20.000Hz

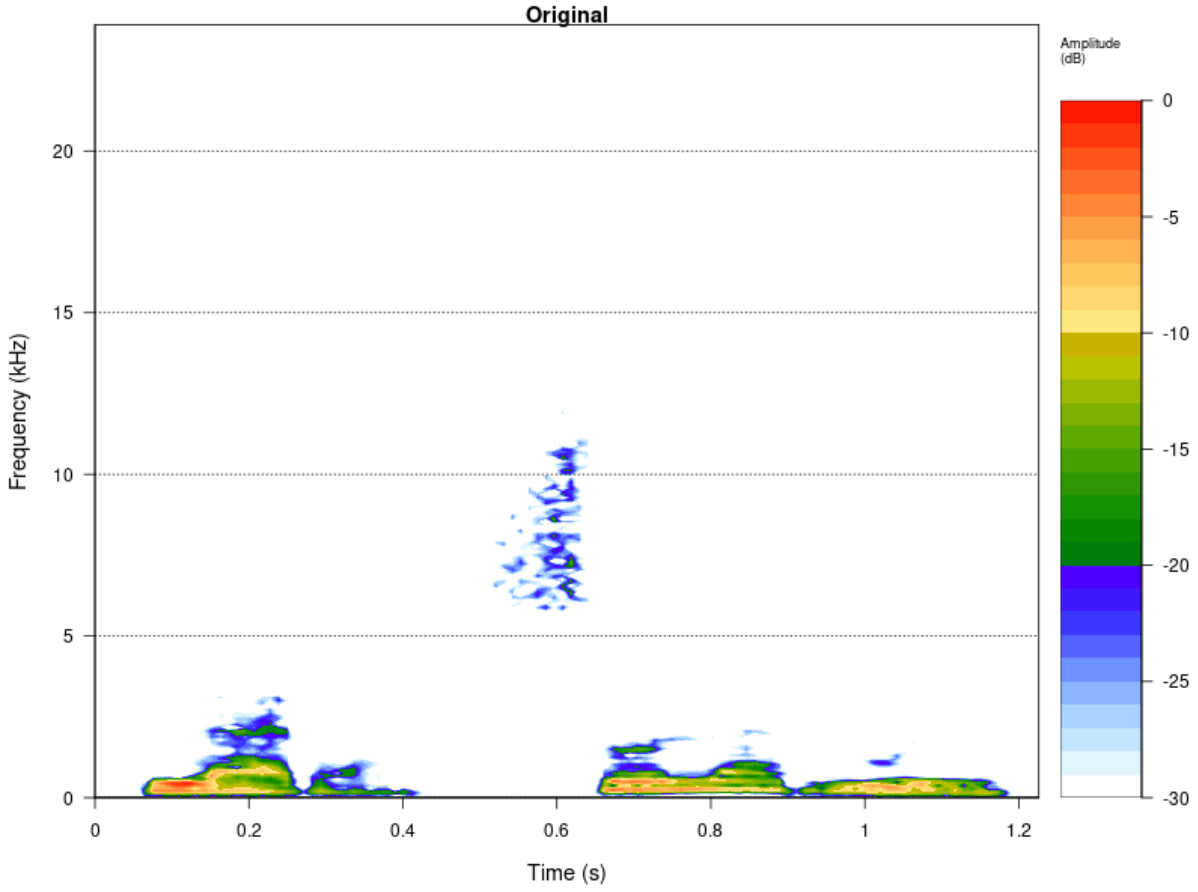
En este ejercicio se nos pide eliminar las frecuencias entre 10.000Hz y 20.000Hz. Para ello hemos utilizado el siguiente código:

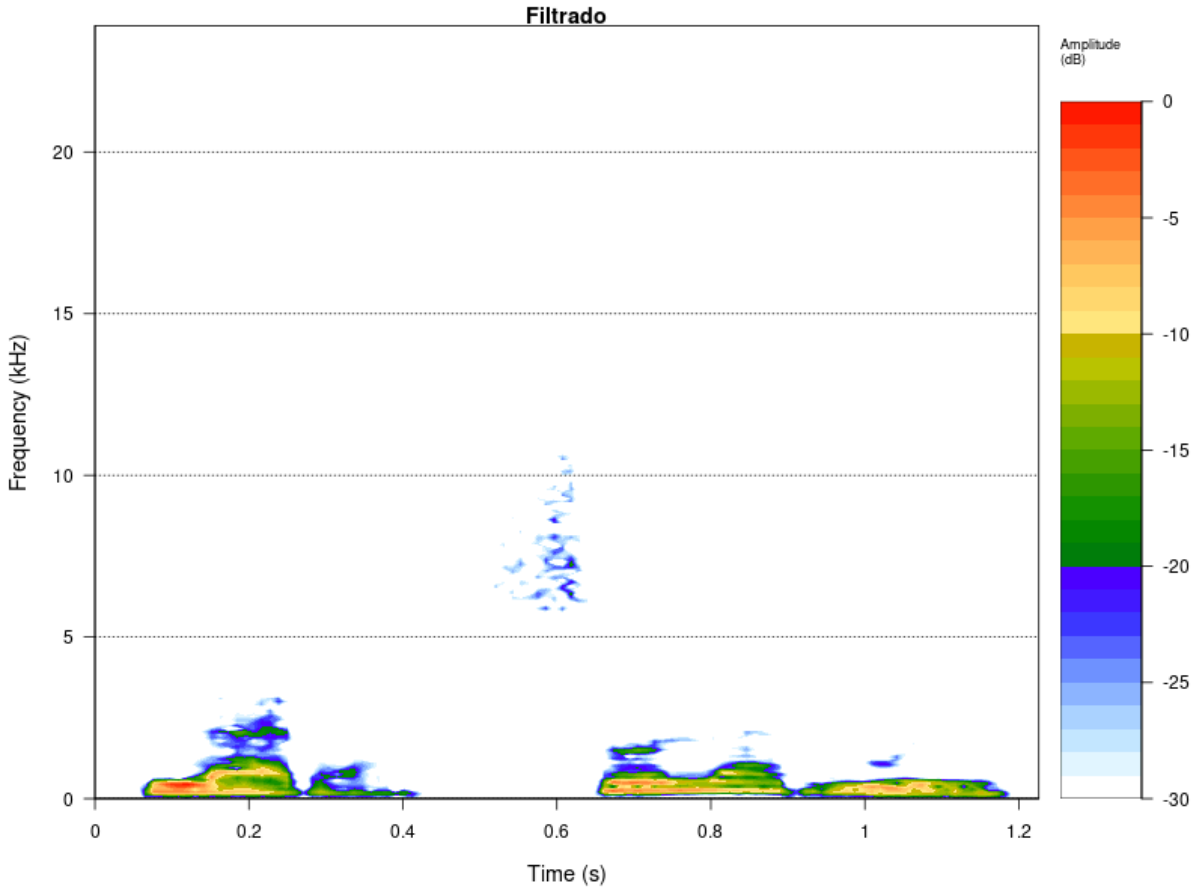
```
# --------- Paso 7: Filtro (eliminar entre 10kHz y 20kHz) ---------
filtrado <- bwfilter(completo, f = completo@samp.rate, n = 1, from =
10000, to = 20000, bandpass = FALSE, listen = TRUE)
filtrado_wave <- Wave(left = filtrado, samp.rate = completo@samp.rate,
bit = 16)
writeWave(filtrado_wave, "filtrado.wav")
```

Con la función *bwfilter()* aplicamos un filtro de frecuencia y con los parámetros *from* y *to* podemos indicar cual es el rango, y con *bandpass* igual al FALSE indicamos que queremos eliminar ese rango. Con esto conseguimos un vector filtrado que contiene la señal sin frecuencias entre 10 kHz y 20 kKHz.

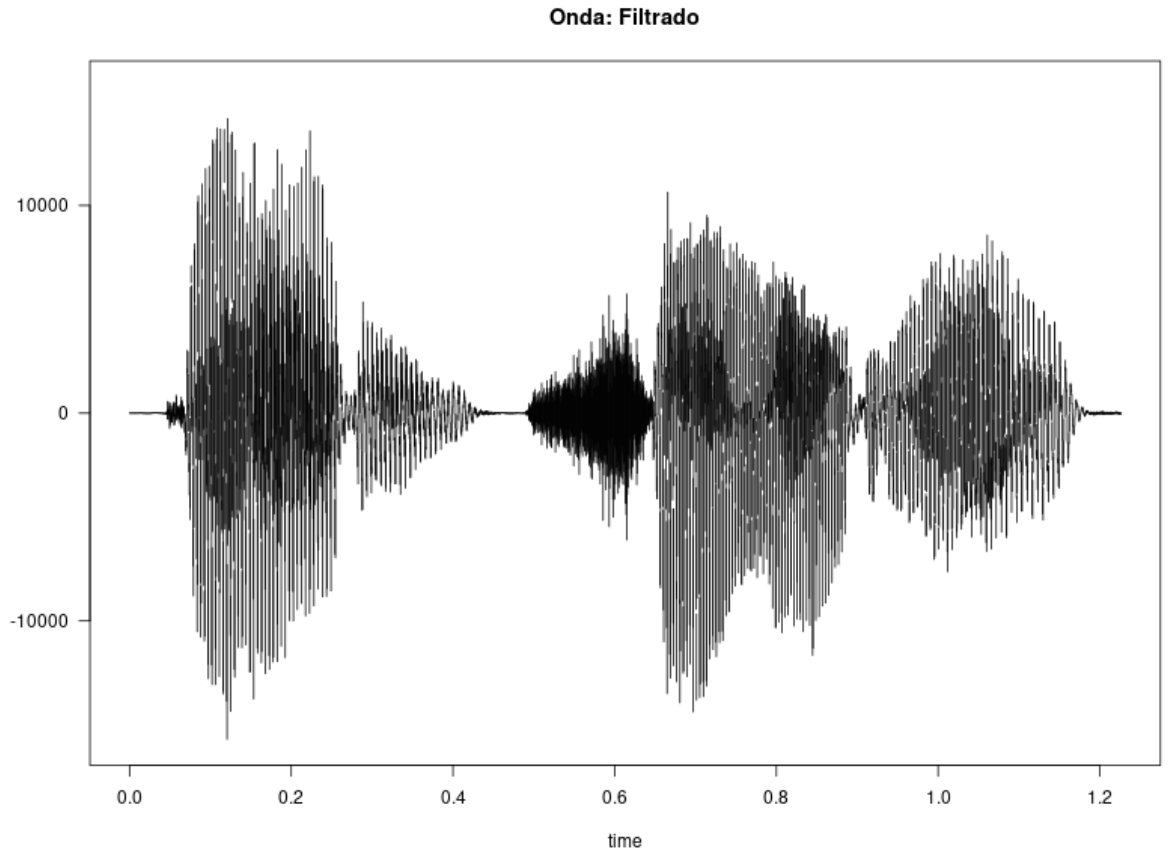
Con la segunda línea de código, convertimos el vector de filtrado en un objeto para poder después guardarlo como un archivo .waw o representarlo gráficamente, como ahora que lo he representado en dos espectrogramas para poder ver la diferencia entre el original y el filtrado.

En el original se puede observar una zona azulada entre 10 kHz y 12 kHz. Mientras que en el filtrado, se ve mucho más tenue. Aunque esta modificación no provoca una gran diferencia audible (ya que la voz humana se concentra por debajo de 5 kHz), sí se puede comprobar visualmente que el filtro se ha aplicado correctamente.


Original
Frequency (kHz)
Time (s)
Amplitude (dB)
0
-5
-10
-15
-20
-25
-30
0
5
10
15
20
0
0.2
0.4
0.6
0.8
1
1.2


Filtrado
Frequency (kHz)
Time (s)
Amplitude (dB)
0
-5
-10
-15
-20
-25
-30
0
5
10
15
20
0
0.2
0.4
0.6
0.8
1
1.2

La onda que se genera es la siguiente:



3.2. Aplicar un efecto de eco y dar la vuelta al sonido

En este apartado se nos pide que generemos un efecto de eco y que guardemos ese sonido, así como después generar el reverso de ese audio. Para ello hemos utilizado el siguiente código:

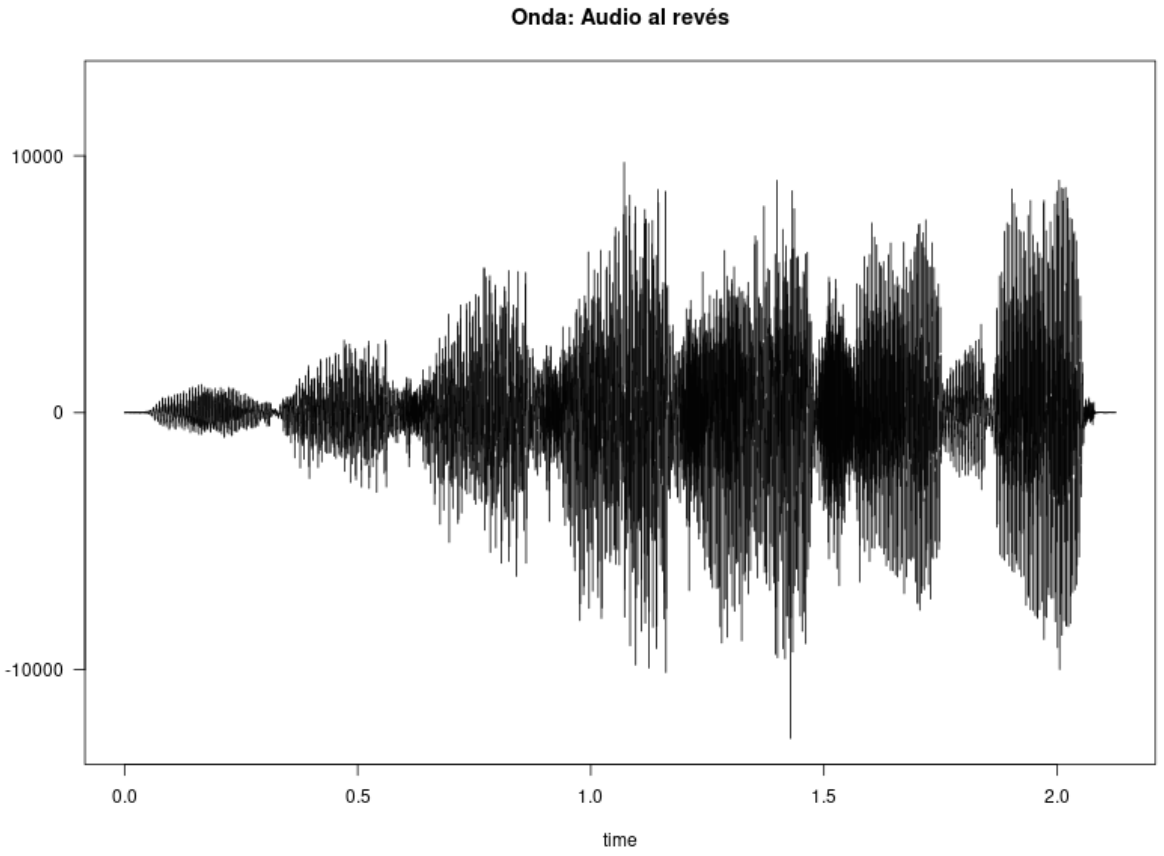
```
# --------- Paso 8: Eco + reverso ---------

# Aplicar eco
eco_wave <- echo(
  completo,
  f = completo@samp.rate,
  amp = c(0.8, 0.4, 0.2),
  delay = c(0.3, 0.6, 0.9),
  output = "Wave"
)

eco_wave@left <- 10000 * eco_wave@left

# Guardar el resultado con eco
writeWave(eco_wave, "eco.wav")
system("aplay eco.wav")
```

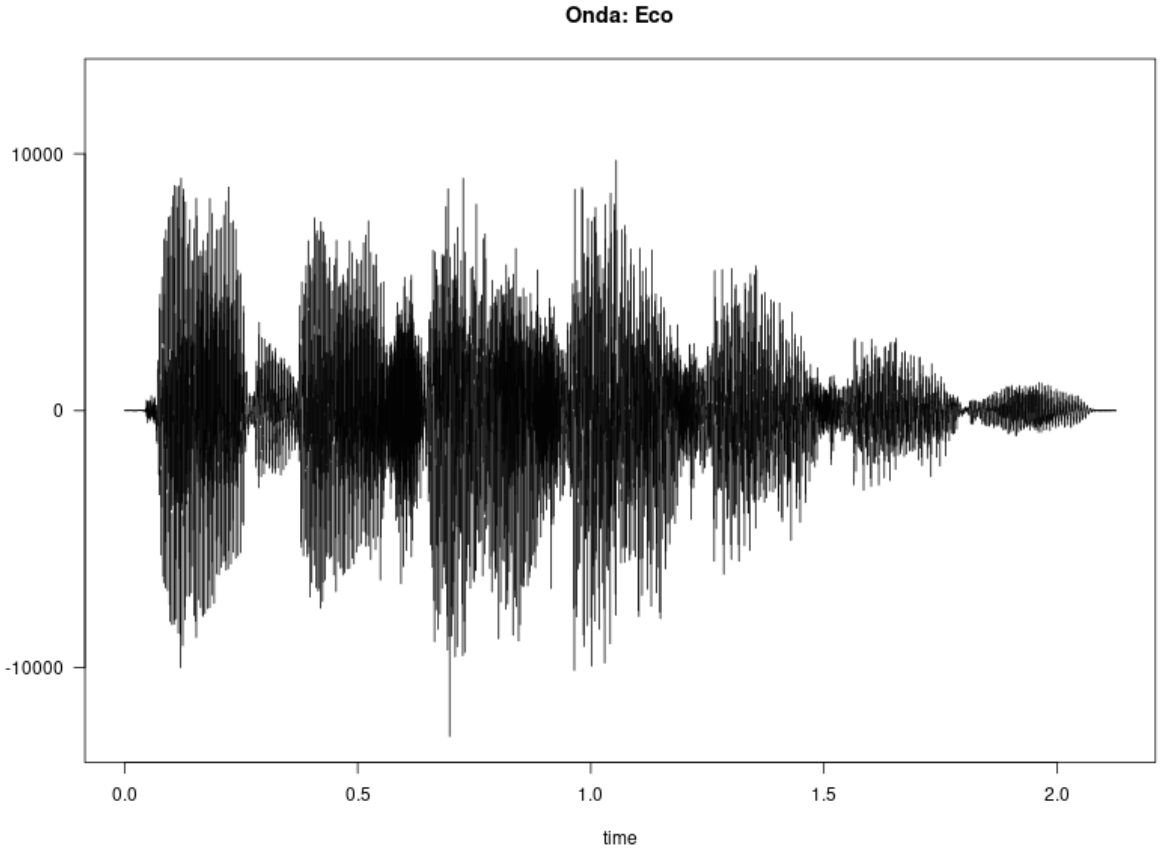
```
# Reverso del audio con eco
alreves <- revw(eco_wave, output = "Wave")

# Guardar el audio al revés
writeWave(alreves, "alreves.wav")
system("aplay alreves.wav")
```

Con la función *echo()* del paquete *seewave*, aplicamos tres ecos con retrasos de 0.3s, 0.6s y 0.9s, y con amplitudes decrecientes de 0.8, 0.4, 0.2 (cada eco suena más bajo). Tras aplicar el eco, el sonido se quedaba muy bajo así que multiplicamos las muestras del canal izquierdo para amplificar el sonido. Finalmente guardamos el resultado con *writeWave* y lo reproducimos con *system("aplay eco.wav").*

Para conseguir el audio al revés, utilizamos *revw()* que invierte el audio siendo la última muestra la primera. Y guardamos el audio igual que antes.

A continuación podemos ver las gráficas que se generan:


Onda: Audio al revés
10000
0
-10000
0.0
0.5
1.0
1.5
2.0
time


Onda: Eco
10000
0
-10000
0.0
0.5
1.0
1.5
2.0
time

4. Bibliografía

Universidad de Granada. (Mayo, 2025). Práctica 5: Sonido [PDF]. Recuperado de https://swad.ugr.es/swad/tmp/gW/dTzNbyL7gTsRYPWsri_WVbdJ1Y8VO-wObjH6_otww/P5-Sonido.pdf

Universidad de Granada. (Mayo, 2025). Material de apoyo – Sonido [PDF]. Recuperado de https://swad.ugr.es/swad/tmp/aT/zSfeTeMmSWjJTUE5n3yMzAd7hXr1SflvzumfNeTcE/S-sonido.pdf

R Core Team. (2024). R: A language and environment for statistical computing. R Foundation for Statistical Computing. Recuperado de https://www.r-project.org/

SpeechGen. (Mayo, 2025). Generador de voz por IA – Conversión de texto a voz online. Recuperado de https://speechgen.io/