PDIH-P5

León Corbacho Rodríguez 4º 21/05/2024

1. **Leer dos ficheros de sonido (WAV o MP3) de unos pocos segundos de duración cada uno. En el primero debe escucharse el nombre de la persona que realiza la práctica. En el segundo debe escucharse el apellido.**

Dando uso de los conocimientos del seminario que habla sobre el R-studio y las librerías con sus métodos podemos realizar los ejercicios. Para este en concreto he usado el "espeak" de Linux para grabar por texto mi nombre y apellidos con los archivos pasados a mí maquina:

Nombre.wav

Apellido.wav

Tras este proceso, debemos cargar los audios en R, esto se realiza con el uso de la función readWAV(nombre.wav) y readWAV(apellido.wav) y le damos los nombres de sonido1 y sonido2 respectivamente.

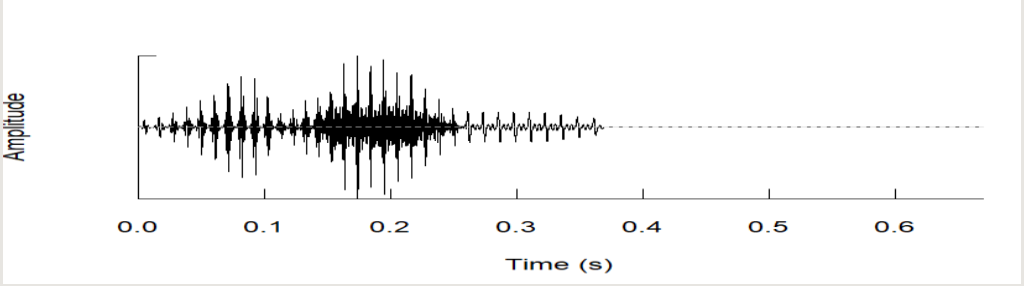
```
library(tuneR)
library(seewave)
#library(soundgen)
library(audio)

# establecer el path concreto en cada caso a la carpeta de trabajo
setwd("/Users/Leon/OneDrive/Escritorio/2023-2024/PDIH/S5-sonidos/")

#Ejercicios de funciones:

# Leer dos ficheros de sonido (WAV o MP3) de unos pocos segundos de duración
sonido1 <- readWave('nombre.wav')
sonido2 <- readWave('apellido.wav')
```

2. **Dibujar la forma de onda de ambos sonidos.**

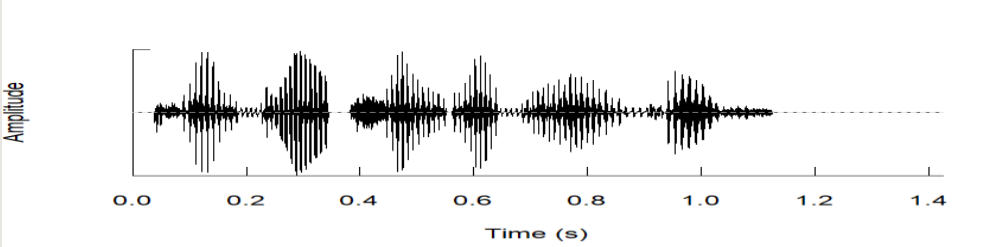
Usando la función oscillo() podemos tener el grafo de las ondas de los audios del sonido1 y sonido2:

```
# Dibujar la forma de onda de ambos sonidos
oscillo(sonido1,f=f)
oscillo(sonido2,f=f)
```

León;



Apellido:



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

Para obtener la información usamos el método str() que nos devuelve la información de la cabecera del archivo.wav. Quedaría así en el script:

```
# Obtener la información de las cabeceras de ambos sonidos
str(sonido1)
str(sonido2)
```

```
> str(sonido1)
Formal class 'Wave' [package "tuneR"] with 6 slots
  ..@ left     : int [1:14769] -2 3 14 24 28 28 28 31 38 43 ...
  ..@ right    : num(0)
  ..@ stereo   : logi FALSE
  ..@ samp.rate: int 22050
  ..@ bit      : int 16
  ..@ pcm      : logi TRUE
>
```

```
> str(sonido2)
Formal class 'Wave' [package "tuneR"] with 6 slots
  ..@ left     : int [1:31418] 0 0 0 0 0 0 0 0 0 0 ...
  ..@ right    : num(0)
  ..@ stereo   : logi FALSE
  ..@ samp.rate: int 22050
  ..@ bit      : int 16
  ..@ pcm      : logi TRUE
>
```

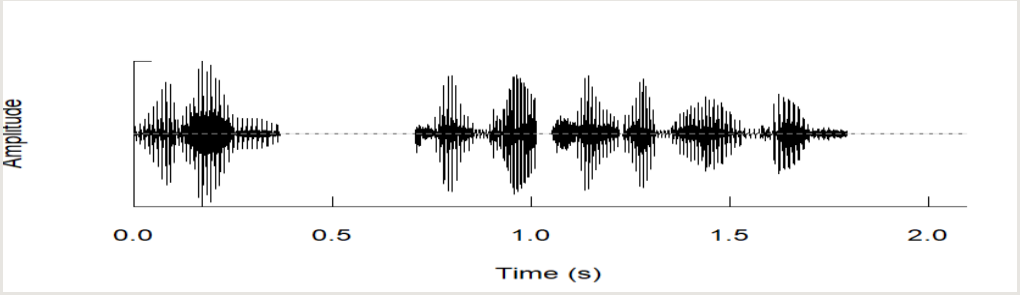
4. Unir ambos sonidos en uno nuevo.

Se puede optar de varias maneras pero la mejor opción para poder pronunciar nuestro nombre completo es simplemente usar el método pastew(s1,s2).
Dándonos como resultado el audio conjunto
(He puesto en los parámetros primero el sonido 2 y luego el sonido 1, porque al parecer pone primero el segundo parámetro y el primero el último, así que para decir mí nombre y después mi apellido lo he tenido que configurar así)

```
# Unir ambos sonidos en uno nuevo
sonido_combinado <- pastew(sonido2, sonido1, output="Wave")
```

5. Dibujar la forma de onda de la señal resultante.

Con el anterior sonido se nos pide que enseñemos las ondas del "sonido_combinado". Usando el método oscillo() tenemos el siguiente dibujo.



Como podemos ver es una fusión entre las dos ondas una detrás de otra, siendo primero el nombre y el segundo el apellido.

6. Pasarle un filtro de frecuencia para eliminar las frecuencias entre 10000Hz y 20000Hz.

Usando un método que se nos ha explicado en clase es el uso del método bwfilter(). Y con las frecuencias que nos dan, los parámetros dentro de este método quedarían una cosa tal que así:

```
#Filtro de frecuencia a sonido combinado de 10000Hz-20000Hz
sonido_combinado_filtrado <- bwfilter(sonido_combinado,f=f, channel=1, n=1, from=10000/(f/50),
                  to=20000/(f/50), bandpass=TRUE, listen = TRUE, output = "Wave")

oscillo(sonido_combinado_filtrado,f=f)

listen(sonido_combinado_filtrado, f=f)

# Asegurarse de que los datos sean redondeados al alza
sonido_combinado_filtrado@left <- ceiling(sonido_combinado_filtrado@left)
if (!is.null(sonido_combinado_filtrado@right)) {
  sonido_combinado_filtrado@right <- ceiling(sonido_combinado_filtrado@right)
}
```

El apartado más abajo es porque siendo tipo wav no sé por qué pero no me deja poner los valores 10000 y 20000 y me pide que ponga (f/2), sin embargo el resultado es que esta no se escucha nada por eso esos valores. Además, al ser un poco rara la salida de los valores de frecuencia hacemos que se redondee al por mayor los valores de frecuencia.

Sin embargo tras la última sesión de prácticas hemos podido resolver todos los problemas ocasionados por el bwfilter() como:

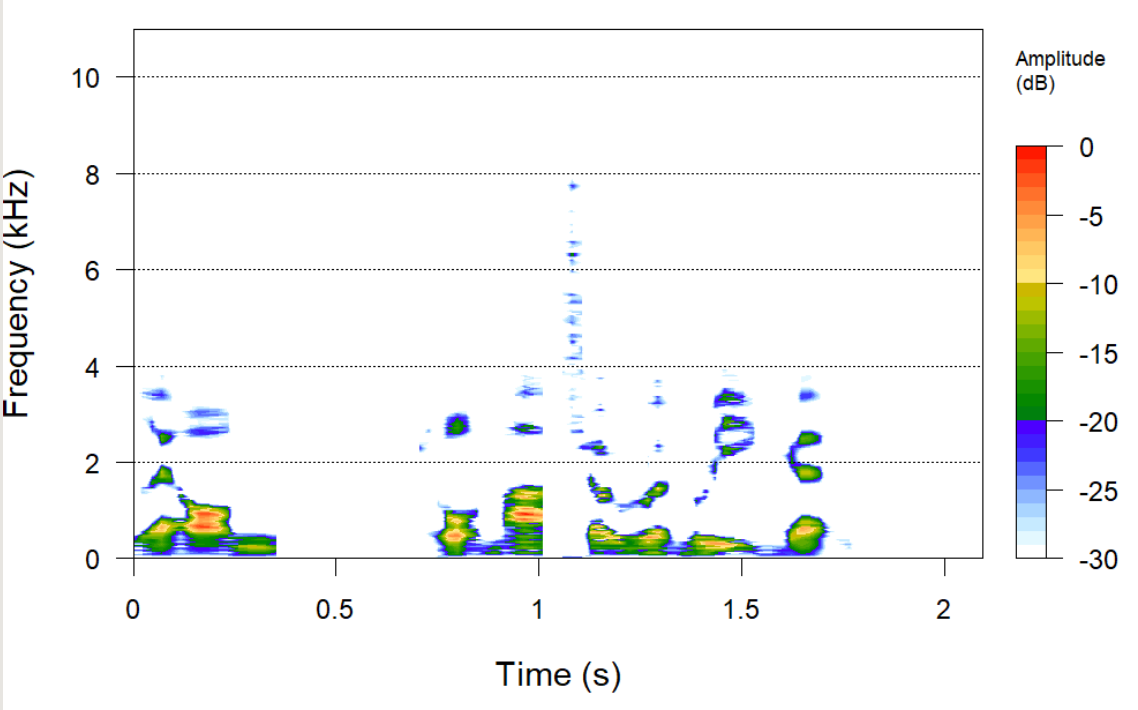
- Tener que normalizarlo
- No realizar los cortes correspondientes de las frecuencias pedidas.
- Escucharse mal el resultado

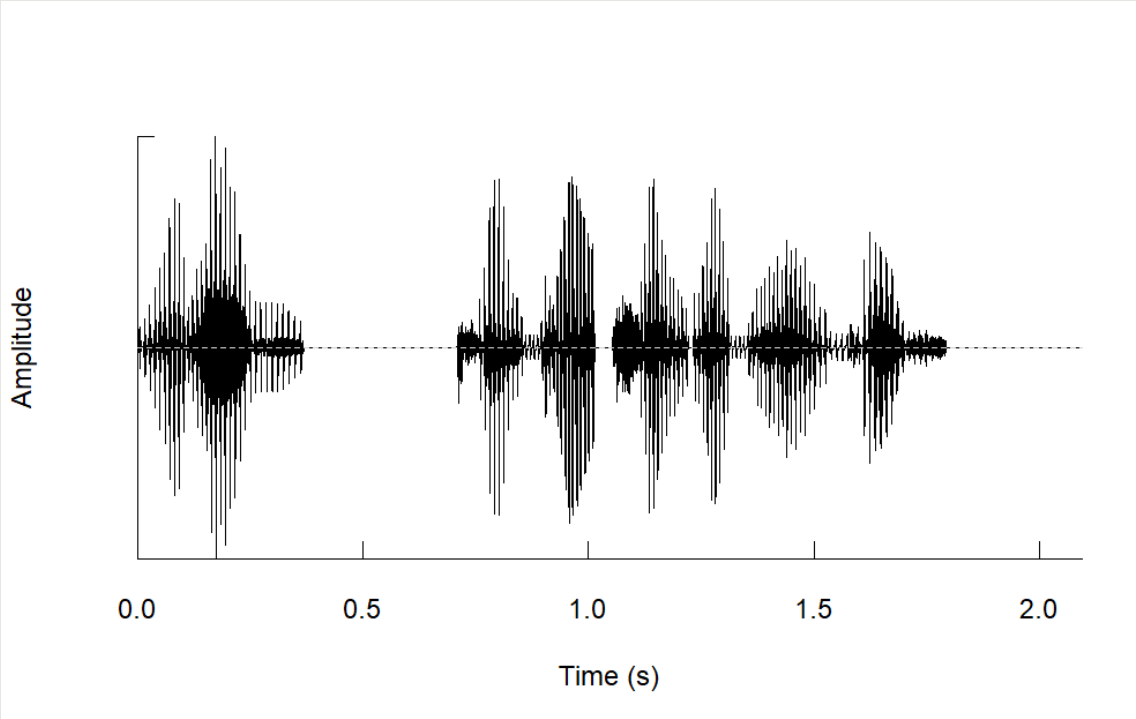
La solución, el método fir(). Este método nos da también la opción de filtrar las frecuencias de un audio según los parámetros que le pongamos. En la práctica que nos atañe lo usaremos con los siguientes parámetros:

```
sonido_combinado_f <-fir(wave = sonido_combinado,
    from = 10000, # lower bound frequency in Hz
    to = 20000, # upper bound frequency in Hz
    bandpass = TRUE,
    output = "Wave")


spectro(wave = sonido_combinado, flim = c(0,8),scale = FALSE)
spectro(wave = sonido_combinado_f, flim = c(0,8))
```

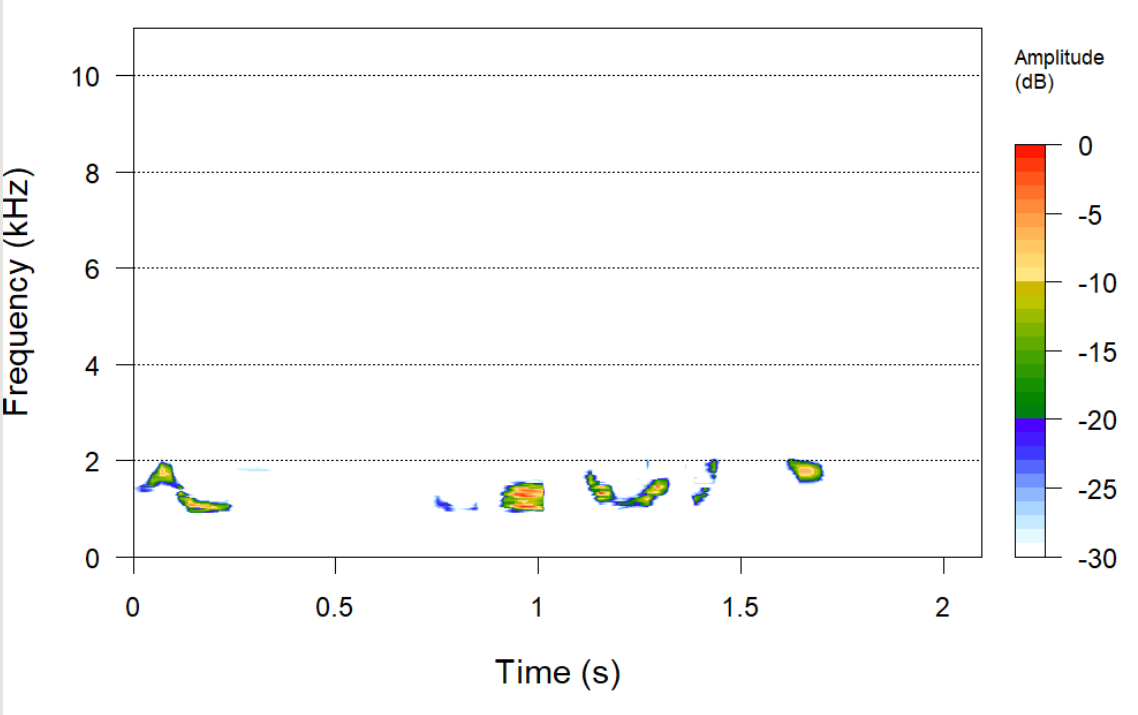
Para poder comprobar que se ha efectuado el filtro tenemos la función spectro(). Esta función nos permite crear un espectrograma del audio y comparando la muestra original del audio y el nuevo con el filtrado podemos ver las diferencias:

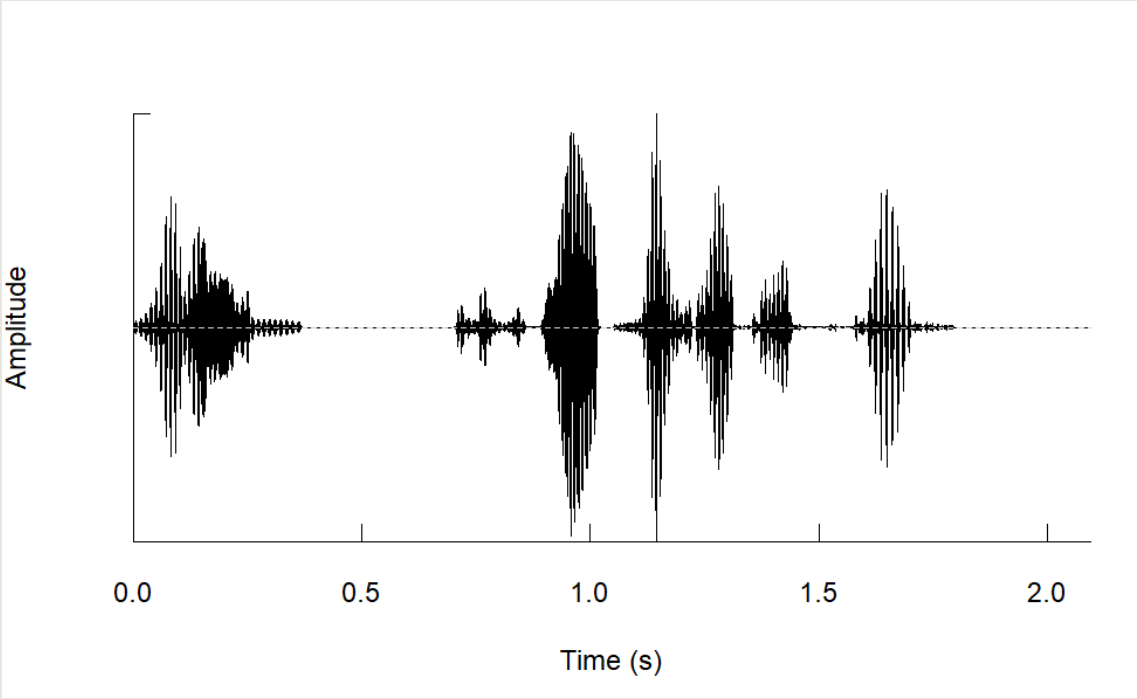
Audio sin filtrar:





Audio filtrado:





Se puede apreciar cómo se ha reducido la frecuencia del sonido y además de que al escucharlo suena como si estuviera tapándose con algo.

7. Almacenar la señal obtenida como un fichero WAV denominado "mezcla.wav".

En el apartado anterior explique el problema del sonido filtrado, tras escribirlas en un archivo wav.

```
# Almacenar la señal combinada como un nuevo fichero WAV
writeWave(sonido_combinado_f, file.path("mezcla.wav"))
```

Sin embargo, nos encontramos con un problema, el objeto "sonido_combinado_f" (El sonido filtrado) se pasa del rango que un archivo wav puede guardar [-32768, 32767], siendo el archivo que compete rangos hasta los (46187).

```
Error en writeWave(sonido_combinado_f, file.path("mezcla.wav")):
  for 16-bit Wave files, data range is supposed to be in [-32768, 3276
7], see ?normalize
```

Para solventar este error como nos indica debemos normalizarlo para que el archivo sea de 16bits. No está en este rango porque la función fir() hace que los números de las frecuencias sean números reales, cosa que la extensión wav no puede leer ya que solo lee muestras de números enteros. Es por eso que recurrimos a la función normalize(). Con esta función normalizamos los valores de las muestras a las de rango de 16 bits y ya no habría problema de escritura

```
sonido_combinado_f <-normalize(sonido_combinado_f, unit="16", center = TRUE, level = 1)
```

Volviendo a ejecutar el primer comando ya tenemos guardado nuestro archivo mezcla.wav.

8. Cargar un nuevo archivo de sonido, aplicarle eco y a continuación darle la vuelta al sonido. Almacenar la señal obtenida como un fichero WAVdenominado “alreves.wav”

Usando como ejemplo el archivo de sonido “oveja.wav”. Cargamos el archivo como ya antes se ha mencionado (readWave()). Lo importantes es añadirle el eco. Esto se consigue con el método echo(sonido,frecuencia,amplitud,delay).

```
# Cargar un nuevo archivo de sonido
nuevo_sonido <- readWave('oveja.wav')

f <- nuevo_sonido@samp.rate

# Aplicarle eco
nuevo_sonido_eco <- echo(nuevo_sonido, f=f, amp=c(0.8, 0.5, 0.1), delay=c(0.5,1,1.5))
```

Tras esto nos pide que invirtamos el sonido. Usando la función revw(), obtenemos una versión del audio dado la vuelta.

```
# Darle la vuelta al sonido (invertir el sonido)
nuevo_sonido_eco_invertido <- revw(nuevo_sonido_eco, f=nuevo_sonido@samp.rate, output="Wave")
```

Sin embargo antes de escribir el archivo nuevo hay que hacer un paso extra. Antes daba error porque no se detectaba las muestras y estas pasaban a ser 0 en el audio resultante si no hacíamos este paso. El paso es multiplicar por 100 el número de las muestras del archivo y así darnos valores más redondeados y no tan cercanos al 0.

```
nuevo_sonido_eco_invertido@left <- 10000 * nuevo_sonido_eco_invertido@left
```

Ya al fin guardamos el archivo como nos indican, “alreves.wav”.

```
# Almacenar la señal obtenida como un fichero WAV denominado “alreves.wav”
writeWave(nuevo_sonido_eco_invertido, file.path("alreves.wav"))
```