Seminario 6. Manejo del sonido con R

Duración: 1 sesión

1. Objetivos del seminario

Los objetivos concretos de este seminario son:

- Identificar y representar gráficamente la forma de onda de señales de sonido.
- Conocer la estructura de un fichero típico de sonido (ficheros WAV).
- Entender y operar con los parámetros principales de una señal de sonido.
- Utilizar el entorno de programación RStudio para el manejo de sonido.

2. El formato de sonido WAV

El formato WAV o WAVE ("onda"), diseñado por Microsoft, es uno de los formatos de sonido más sencillos que existen. Consta básicamente de tres partes o bloques: el fichero comienza con un bloque de identificación, a continuación sigue un bloque que especifica los parámetros del formato y, por último, finaliza con el bloque que contiene las muestras.

La siguiente tabla refleja la estructura interna de un fichero WAV, indicando para cada uno de los bloques, los campos de que consta. El formato en que se almacenan los datos mayores que 1 bytes es little-endian (primero el más pequeño). Por ejemplo, el dato binario de 16 bits 0A10, correspondería como número entero a H'100A = D'4106

Bloque	Número de bytes	Descripción del campo	Contenido
Identificación (12 bytes)	4	Caracteres de identificación	RIFF
	4	Longitud del resto del fichero	
	4	Caracteres de identificación	WAVE
Parámetros (24 bytes)	4	Caracteres de inicio del bloque de formato	fmt_
	4	Longitud del resto del bloque de formato	0x10 (para PCM)
	2	Formato de audio	0x01 (para PCM)
	2	Número de canales	0x01 (mono), 0x02 (estéreo)
	4	Frecuencia de muestreo en herzios	
	4	Bytes por segundo	(Frecuencia*Bytes_por_muestra)
	2	Bytes por muestra	(Num_canales*Resolución/8)
	2	Resolución en bits	(8 ó 16)
Muestras	4	Caracteres de inicio de bloque	data
	4	Número de bytes de muestras	
	resto	Bytes de muestras	

La siguiente captura de pantalla muestra el contenido en hexadecimal de un fichero WAV en sus primeros bytes:

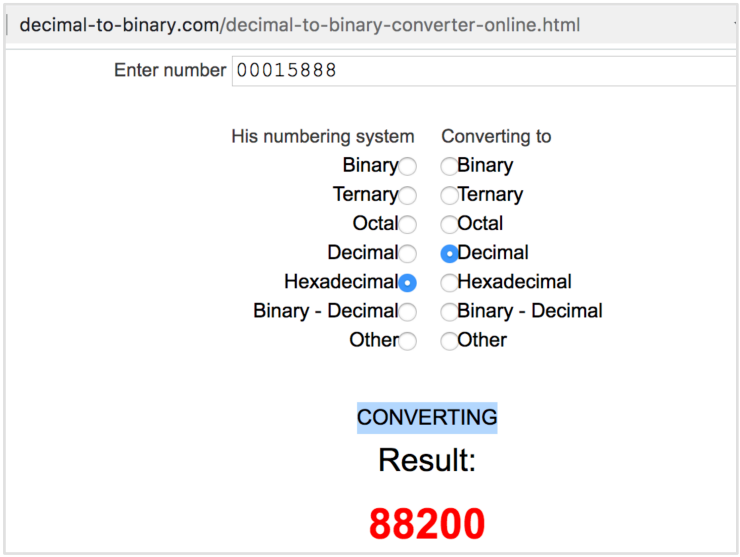
```
          0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00000000h: 52 49 46 46 8A 02 01 00 57 41 56 45 66 6D 74 20
00000010h: 10 00 00 00 01 00 01 00 44 AC 00 00 88 58 01 00
00000020h: 02 00 10 00 64 61 74 61 66 02 01 00 00 00 00 00
```

Vamos a identificar los bytes de cada campo y su valor decimal de la cabecera en este fichero para determinar las características de ese sonido:

Campo	Bytes	Valor decimal o caracteres
Caracteres de identificación (4B)	52 49 46 46	R I F F
Longitud del resto del fichero (4B)	8A 02 01 00	66186 B
Caracteres de identificación (4B)	57 41 56 45	W A V E
Caracteres de inicio del bloque de formato (4B)	66 6D 74 20	f m t _
Longitud del resto del bloque de formato (4B)	10 00 00 00	0x10 (PCM)
Formato de audio (2B)	01 00	0x01 (PCM)
Número de canales (2B)	01 00	1 Canal
Frecuencia de muestreo en herzios (4B)	44 AC 00 00	44100 Hz
Bytes por segundo (4B)	88 58 01 00	88200 B/s
Bytes por muestra (2B)	02 00	2 B/muestra
Resolución en bits (2B)	10 00	16 bits
Caracteres de inicio de bloque (4B)	64 61 74 61	d a t a
Número de bytes de muestras (4B)	66 02 01 00	66150 muestras
Bytes de muestras (resto)	...	...

Para transformar fácilmente los números en hexadecimal a decimal y entender cómo se han obtenido esos valores, podemos usar la siguiente calculadora:

http://decimal-to-binary.com/decimal-to-binary-converter-online.html



3. Reproducción y operaciones con sonidos en R

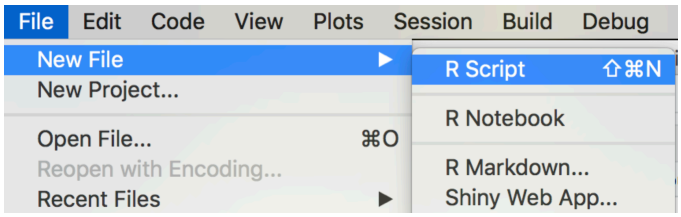
En R disponemos de varias librerías que nos permitirán trabajar con archivos de sonido, tanto WAV como MP3. Las más conocidas y aquellas para las que más documentación podemos encontrar son tuneR, seewave y soundgen.

En este seminario (y la práctica correspondiente) usaremos las dos primeras.

Una vez instalado RStudio, pasaremos a instalar ambas librerías:

```
install.packages('tuneR', dep=TRUE)
install.packages('seewave', dep=TRUE)
```

y crearemos un script de lenguaje R:



en el que al principio cargaremos las librerías necesarias:

```
library(tuneR)
library(seewave)
library(audio)
```

Cargar, reproducir y visualizar sonidos

Podemos cargar archivos de sonido (MP3 o WAV) e inspeccionar las características de dicho sonido:

```
perro <- readWave('perro.wav')
perro
gato  <- readMP3('gato.mp3')
gato
oveja <- readWave('oveja.wav')
oveja
str(perro)
str(gato)
str(oveja)
```

Podemos calcular la duración exacta del sonido dividiendo el número de muestras entre la frecuencia (podemos mostrarlo redondeando con tres decimales):

```
round(length(gato@left) / gato@samp.rate, 3)
```

Para escuchar el sonido, usaremos la función listen(), y para mostrar su forma de onda, la función plot() indicando desde qué muestra hasta qué otra queremos mostrar:

```
listen(gato)
plot( extractWave(gato, from = 1, to = 393984) )
```

Si sólo deseamos escuchar una porción del sonido original, especificaremos desde qué segundo y hasta cuál:

```
listen(gato, f=44100, from=6.5, to=8.9)
```

Trocear, unir y guardar sonidos

Si un sonido es muy largo, o sólo nos interesa una porción, podemos usar la función cutw() para quedarnos justo con la porción que queremos:

```
s2 <- cutw(gato, from=6.5,to=8.9, output="Wave")
s2
plot( extractWave(s2, from = 1, to =  105841) )
listen(s2)
```

Ahora nos hemos quedado con el último maullido del sonido original. Si queremos añadirle a este maullido el sonido de los ladridos del perro, usaremos la función pastew():

```
s3 <- pastew(perro, s2, output="Wave")
s3
plot( extractWave(s3, from = 1, to=265573) )
listen(s3)
```

Finalmente, si queremos eliminar alguna porción de sonido, usaremos la función deletew(), y si queremos añadir algún silencio a un sonido, usaremos addsilw().

```
s4 <- deletew(s3, output="Wave", from = duration(s2), to =
duration(perro))
duration(s4)
s4
s5 <- addsilw(s4, at = "end", d = 1, output = "Wave")
duration(s5)
```

Una vez hemos terminado de trabajar con los sonidos, podemos guardarlos en disco en el formato deseado:

```
writeWave(s3, file.path("snd-gato-perro.wav") )
```

Hasta ahora hemos trabajado con dos sonidos que tienen la misma frecuencia (44100Hz), pero si queremos hacer operaciones con otro sonido de diferente frecuencia, debemos ajustar la frecuencia de uno de ellos para igualarla a la del otro (downsampling / upsampling). A continuación cambiaremos la frecuencia del sonido “gato” desde los 44100Hz hasta los 8000Hz (para poder concatenarlo con el sonido de la oveja):

```
gato8000<-resamp(s2,f=44100,g=8000, output="Wave")
gato8000
s6 <- pastew(sheep, gato8000, output="Wave")
s6
plot( extractWave(s6, from = 1, to=38965) )
listen(s6)
writeWave(s6, file.path("snd-gato-perro.wav") )
```

Otras formas de modificar sonidos

Además, estas librerías nos permiten aplicar diferentes filtros: amplitud y frecuencia.

La función afilter() nos permite aplicar un filtro de amplitud, eliminando las muestras que sean inferiores a cierto nivel (threshold), visualizando el sonido resultante:

```
sndF1 <- afilter(gato,f=f,threshold=50,colwave="red")
listen(sndF1,f=f)

sndF2 <- afilter(gato,f=f,threshold=5,colwave="blue")
listen(sndF2,f=f)
```

Por otro lado, la función bwfilter() aplica un filtro de frecuencia, eliminando las muestras entre las frecuencias (en Hz) indicadas:

```
sndF3 <- bwfilter(gato,f=f, channel=1, n=1, from=3000, to=22000,
bandpass=TRUE)
listen(sndF3,f=f)
```

Debido a que estas funciones devuelven arrays con las muestras, para visualizarlas debemos acceder al array indicando los elementos a representar:

```
plot( sndF1[1:393984] , type = "l")
plot( sndF2[1:393984] , type = "l")
plot( sndF3[1:393984] , type = "l")
```

Para aplicar efectos de eco, disponemos de la función echo()

```
hola <- readWave('hola.wav')
str(hola)
holaECO <- echo(hola,f=22050,amp=c(0.8,0.4,0.2),delay=c(1,2,3))
listen(holaECO , f=22050)
str(holaECO)
plot( holaECO[1:77959] , type = "l")
```

La función revw() de la librería seewave le da la vuelta al sonido, de forma que la última muestra pasa a ser la primera:

```
alreves <- revw(hola, output="Wave")
listen(alreves)
```

Grabar sonidos

Desde R se pueden grabar sonidos directamente de la tarjeta de sonido. Para ello hay que crear un objeto de tipo “record” especificando cuántas muestras grabaremos y a qué frecuencia. En el siguiente ejemplo, si grabamos 24.000 muestras a 8.000 Hz, tendremos un sonido de 3 segundos de duración:

```
a <- record(24000, 8000, 1)
wait(a)           # wait for the recording to finish
x <- a$data       # get the result
x[1:10]           # show first ten samples
close(a); rm(a)   # you can close the instance at this point

listen(x,f=8000)

save.wave(x, file.path("tmpRECORDsavewave.wav") )  # guardarlo
xtmp <- readWave('tmpRECORDsavewave.wav')          # leerlo
xtmp

plot( extractWave(xtmp, from = 1, to = 24000) )
listen(xtmp,f=44100)
```

4. Script para probar las funciones implementadas

Para comprobar las funciones anteriores deberá crear un script de R para realizar las siguientes acciones:

1. Leer dos ficheros de sonido (WAV o MP3) de unos pocos segundos de duración cada uno.
2. Dibujar la forma de onda de ambos sonidos.
3. Obtener la información de las cabeceras de ambos sonidos.

4. Unir ambos sonidos en uno nuevo.
5. Dibujar la forma de onda de la señal resultante.
6. Pasarle un filtro de frecuencia para eliminar las frecuencias por debajo de 3.000Hz
7. Almacenar la señal obtenida como un fichero WAV denominado “mezcla.wav”.
8. Grabar un archivo de sonido de 2 segundos desde la tarjeta de sonido, reproducirlo y mostrar la señal de onda.

Como último ejercicio, extraer los valores de los campos de cabecera del fichero WAV “hola.wav” (facilitado por el profesor de la asignatura) a través de un editor hexadecimal (p.ej. “*HxD Hex Editor*”).

Cuestiones a resolver

El objetivo principal es aprender a reproducir, crear, modificar y manejar sonidos usando el lenguaje R.

El estudiante debe estudiar la estructura de un archivo de sonido WAV y aprender a manejar sonido desde R. A continuación realizará el script propuesto en la sección anterior de este guión y comprobará su correcto funcionamiento ejecutándolo en el entorno de RStudio.

Como resultado **se mostrará** al profesor el funcionamiento del script desarrollado (comprobar que los sonidos se representan y suenan como deben).

En el documento a entregar se describirá cómo se ha creado el script R y se mostrarán capturas de pantalla de las formas de onda obtenidas.

Normas de entrega

La práctica o seminario podrá realizarse de manera individual o por grupos de hasta 2 personas.

Se entregará como un archivo de texto en el que se muestre la información requerida. También se puede utilizar la sintaxis de Markdown para conseguir una mejor presentación e incluso integrar imágenes o capturas de pantalla. La entrega se realizará subiendo los archivos necesarios al repositorio “**PDIH**” en la cuenta de GitHub del estudiante, a una carpeta llamada “**S6**”.

Toda la documentación y material exigidos se entregarán en la fecha indicada por el profesor. No se recogerá ni admitirá la entrega posterior de las prácticas/seminarios ni de parte de los mismos.

La detección de copias implicará el suspenso inmediato de todos los implicados en la copia (tanto de quien realizó el trabajo como de quien lo copió).

Las faltas de ortografía se penalizarán con hasta 1 punto de la nota de la práctica o seminario.

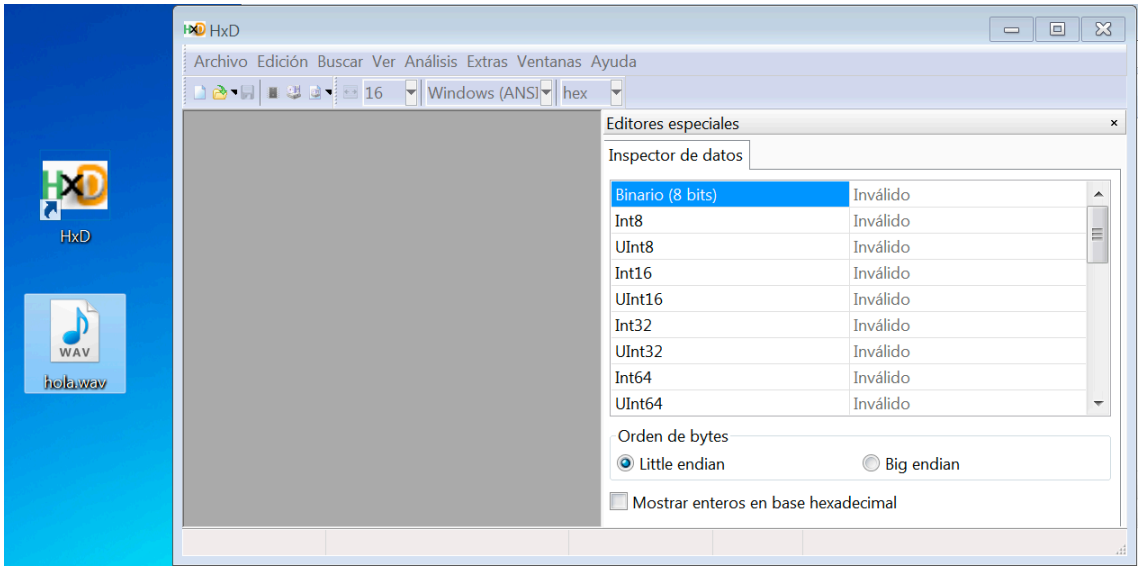
Referencias

https://www.r-bloggers.com/intro-to-sound-analysis-with-r/

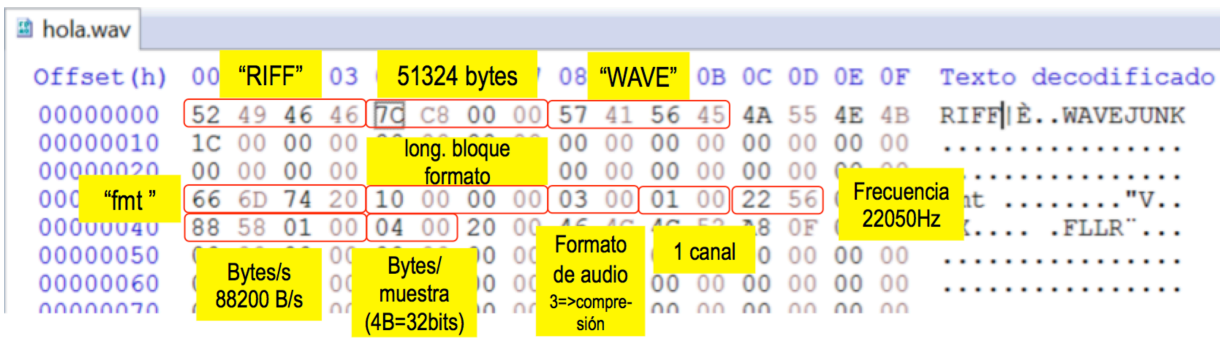
https://hansenjohnson.org/post/spectrograms-in-r/
http://samcarcagno.altervista.org/blog/basic-sound-processing-r/
https://rstudio-pubs-static.s3.amazonaws.com/92070_d44c3e320fae4b21b12c6101f347b28c.html#3
https://towardsdatascience.com/music-genre-classification-with-python-c714d032f0d8
https://cran.r-project.org/web/packages/soundgen/vignettes/acoustic_analysis.html
http://rug.mnhn.fr/seewave/
https://en.wikipedia.org/wiki/HxD
https://mh-nexus.de/en/hxd/

Apéndice

Utilizando el editor hexadecimal HxD podemos inspeccionar cualquier archivo a nivel binario. En nuestro caso, nos interesa examinar la cabecera de un archivo WAV. Para ello, tras instalar el editor, lo ejecutamos y arrastramos el archivo WAV a la ventana:



En la parte de la izquierda veremos el contenido del archivo abierto, en hexadecimal y en texto decodificado. En la parte de la derecha podemos ver en detalle el valor de cada byte del archivo (sólo tenemos que apuntar con el ratón el byte a inspeccionar). Para inspeccionar la cabecera del archivo WAV nos fijaremos en la parte de la izquierda de la ventana del editor para anotar los valores de los bytes que nos interesa:

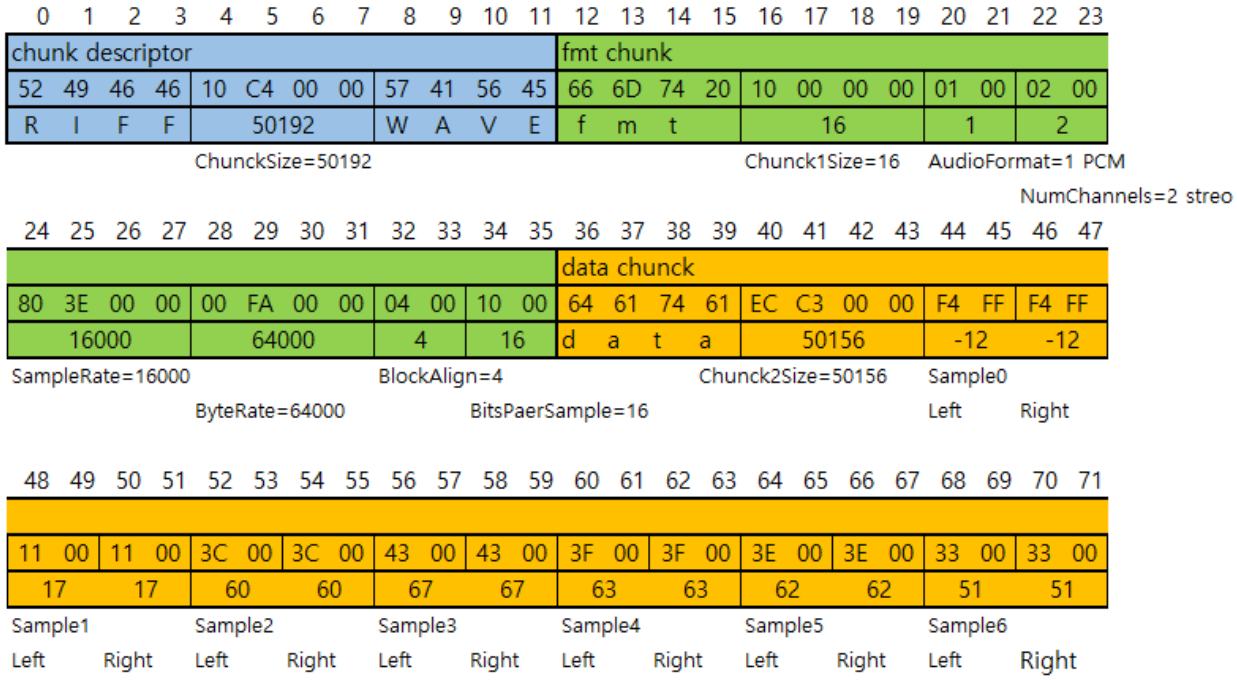


Como vemos, el programa que creó el archivo WAV "hola.wav" no usó el formato estándar descrito más arriba (el formato de audio tiene el valor 3, lo que indica que se ha usado compresión).

Si abrimos el archivo de sonido “daum.wav”, que sí sigue el formato estándar, veremos el siguiente contenido (se muestran los 72 primeros bytes del archivo):

```
52 49 46 46 10 C400 00 57 41 56 45 66 6D74 20 10 00 00 00 01 00 02 00
80 3E 00 00 00 FA 00 00 04 00 10 00 64 61 74 61 ECC3 00 00 F4 FF F4 FF
11 00 11 00 3C00 3C00 43 00 43 00 3F 00 3F 00 3E 00 3E 00 33 00 33 00
```

A continuación se analiza cada campo del archivo:



numsample=Chunck2Size/BlockAlign=50156/4=12539

datachunk base=20+chunck1Size=20+16=36

ChunkSize=12+Chunk1Size+8+Chunk2Size=12+16+8+50156=50192

Chunk2Size=numsample*BlockAlign=12589*4=50156

BlockAlign=NumChannels*BitsPerSample/8=2*16/8=4