Introducción a la Robótica Empresarial

Trabajo Práctico Final

Profesora Dra Marcela Riccillo

Gian Luca Pecile - 41614672

Índice

Ejercicio 1 - Sonido

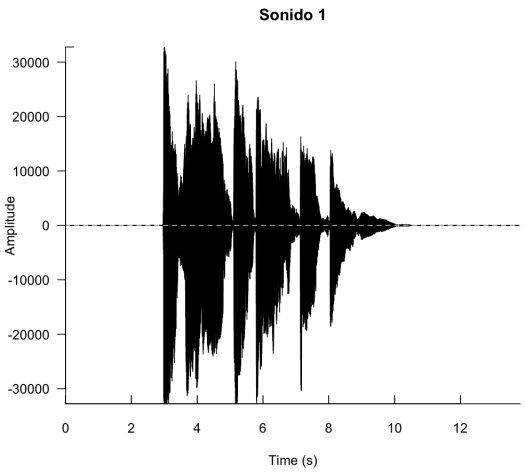
Sonidos seleccionados con hipervínculo a su fuente de pixabay:

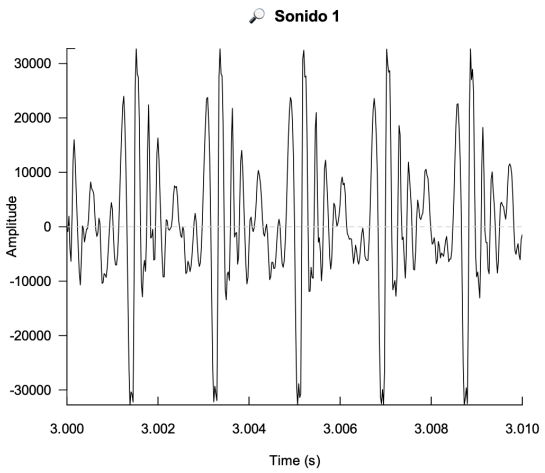
1. Instrumental musical: sax jazz.
2. Sonido genérico: brownian noise.
3. Sonido de habla: arigato.

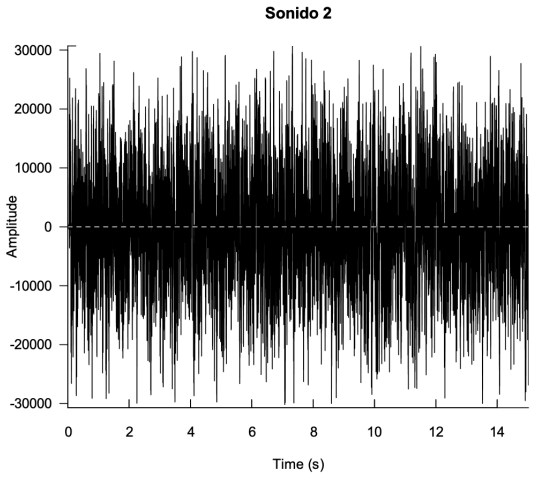
Parte A - Oscilograma

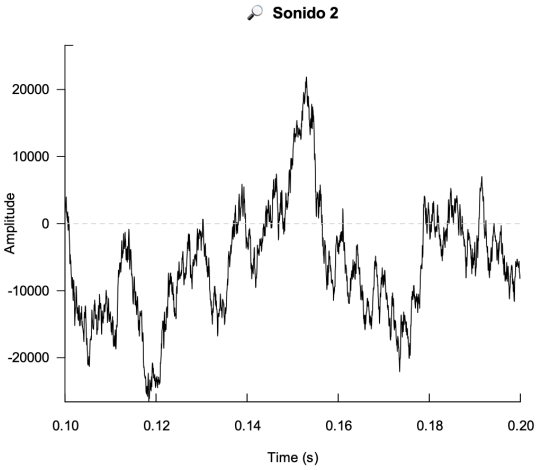
	Cantidad de muestras	Duración (segundos)	Muestreo (Hz)	Mono o stereo
Sonido 1	609408	13.82	44100	Mono
Sonido 2	360576	15.02	24000	Stereo
Sonido 3	76032	3.17	24000	Stereo

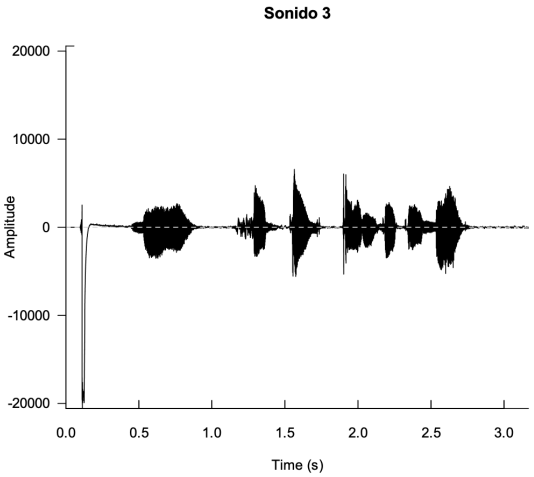
En las siguientes figuras se comparan los sonidos con su versión donde se aplica zoom (🔎) para apreciar mejor cada onda:

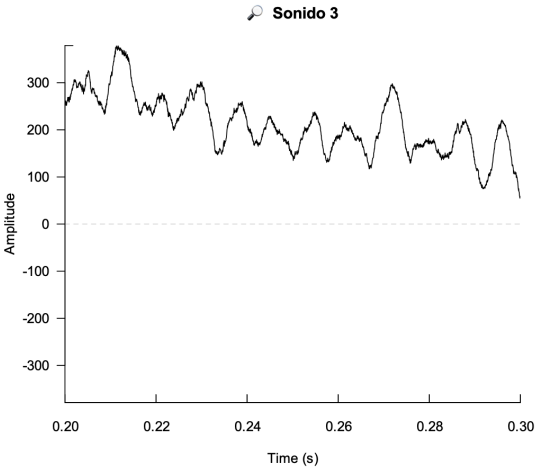


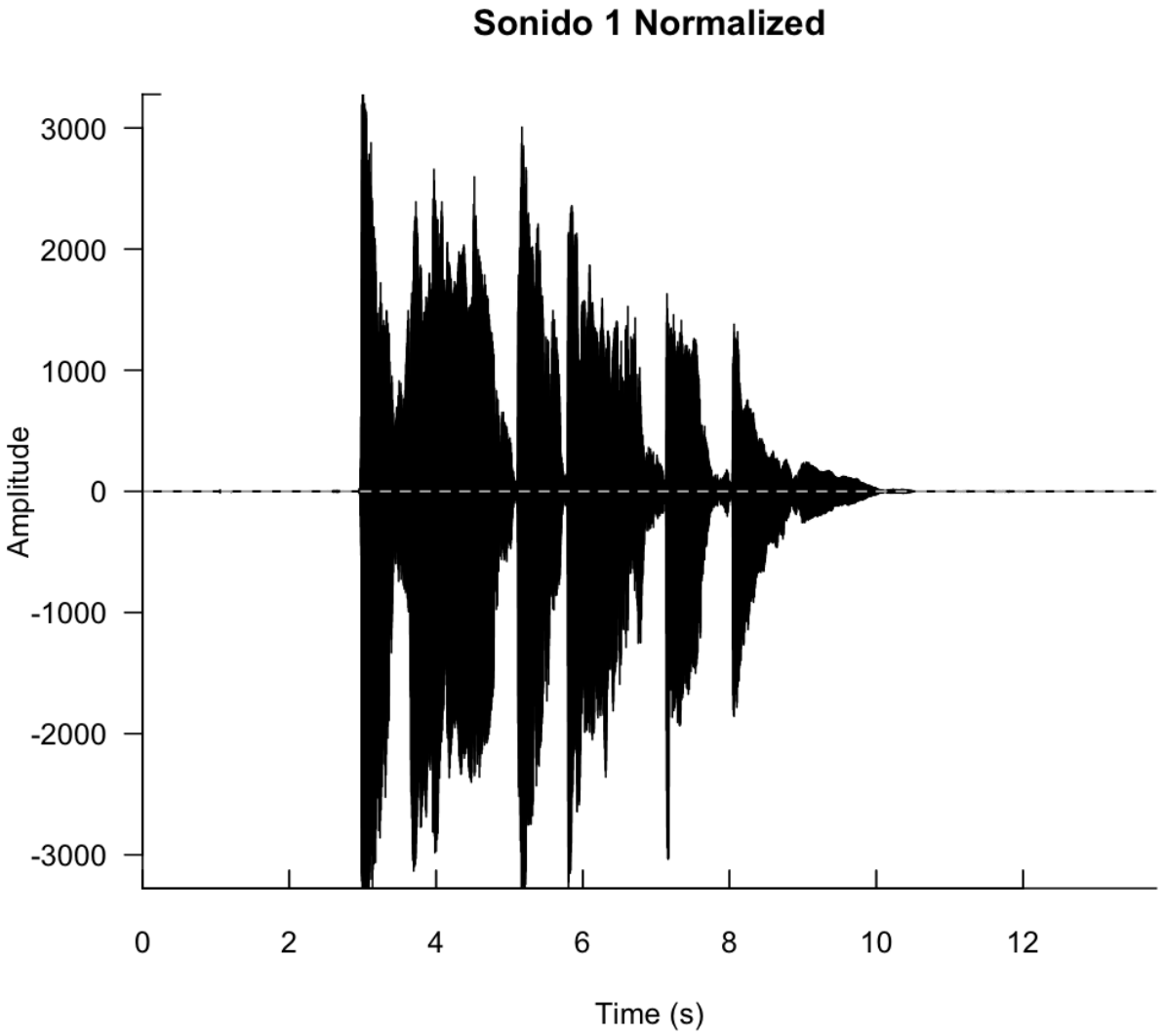



Sonido 2
Amplitude
Time (s)


Sonido 2
Amplitude
Time (s)


Sonido 3
Amplitude
Time (s)

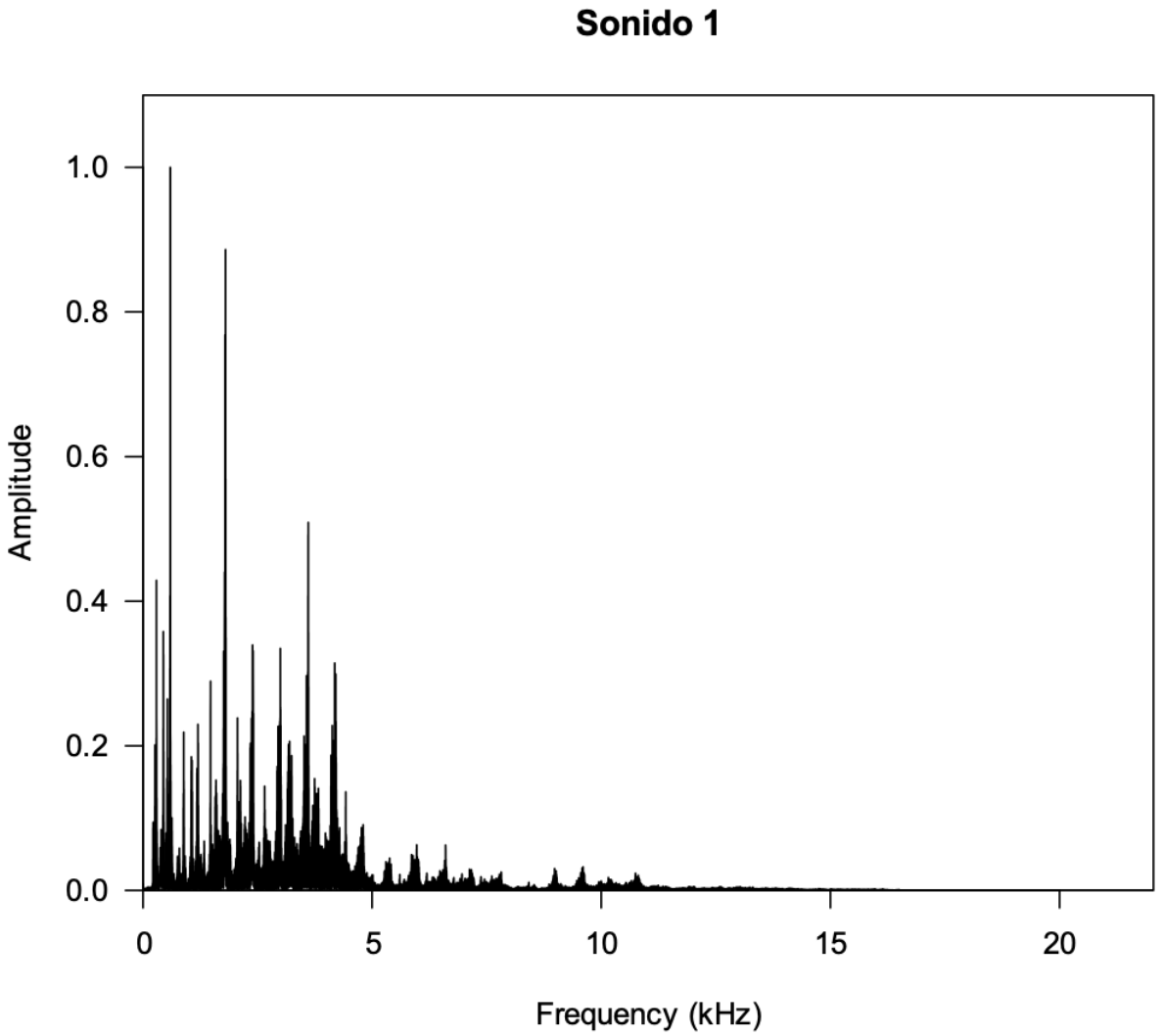

Sonido 3
Amplitude
Time (s)



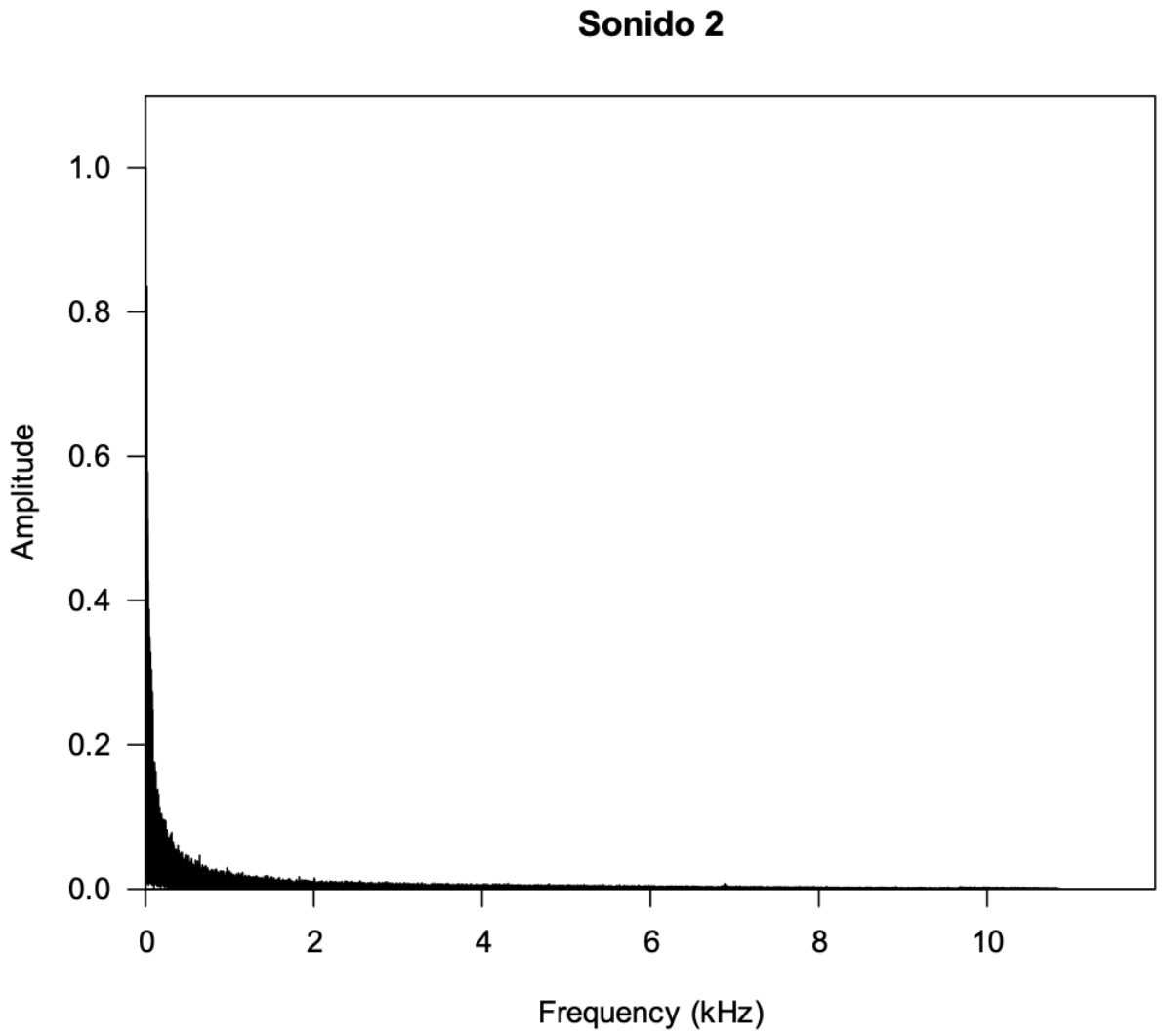
En comparación con el sonido 1, la amplitud resulta un 10% de la original preservando el tiempo y figura general. A su vez, el sonido al ser reproducido se escucha más bajo en comparación.

Parte B - Periodograma / Espectograma

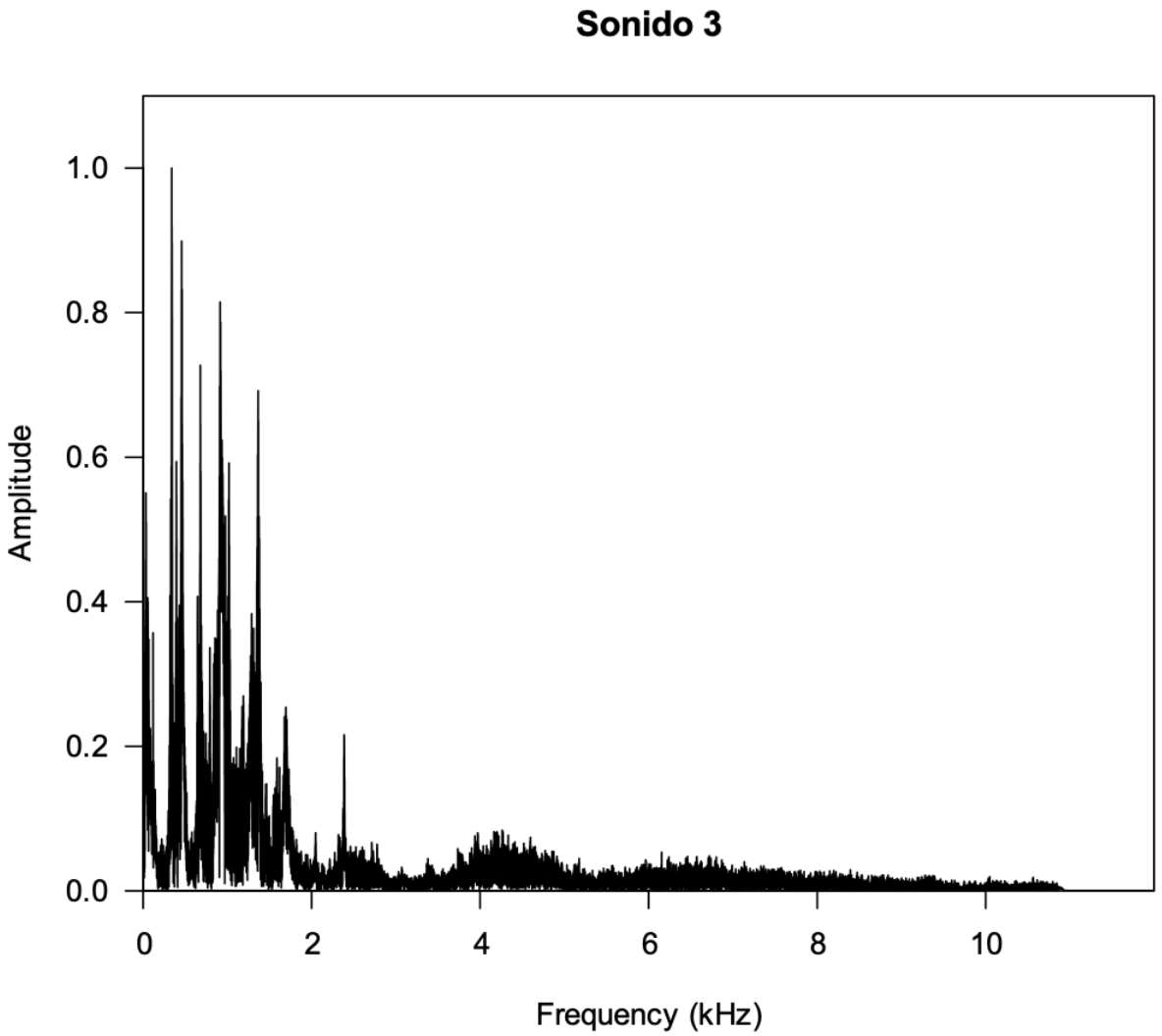
Se usa **spec** para la creación de los periodogramas de las figuras a continuación:



Rango: 0-20kHz.



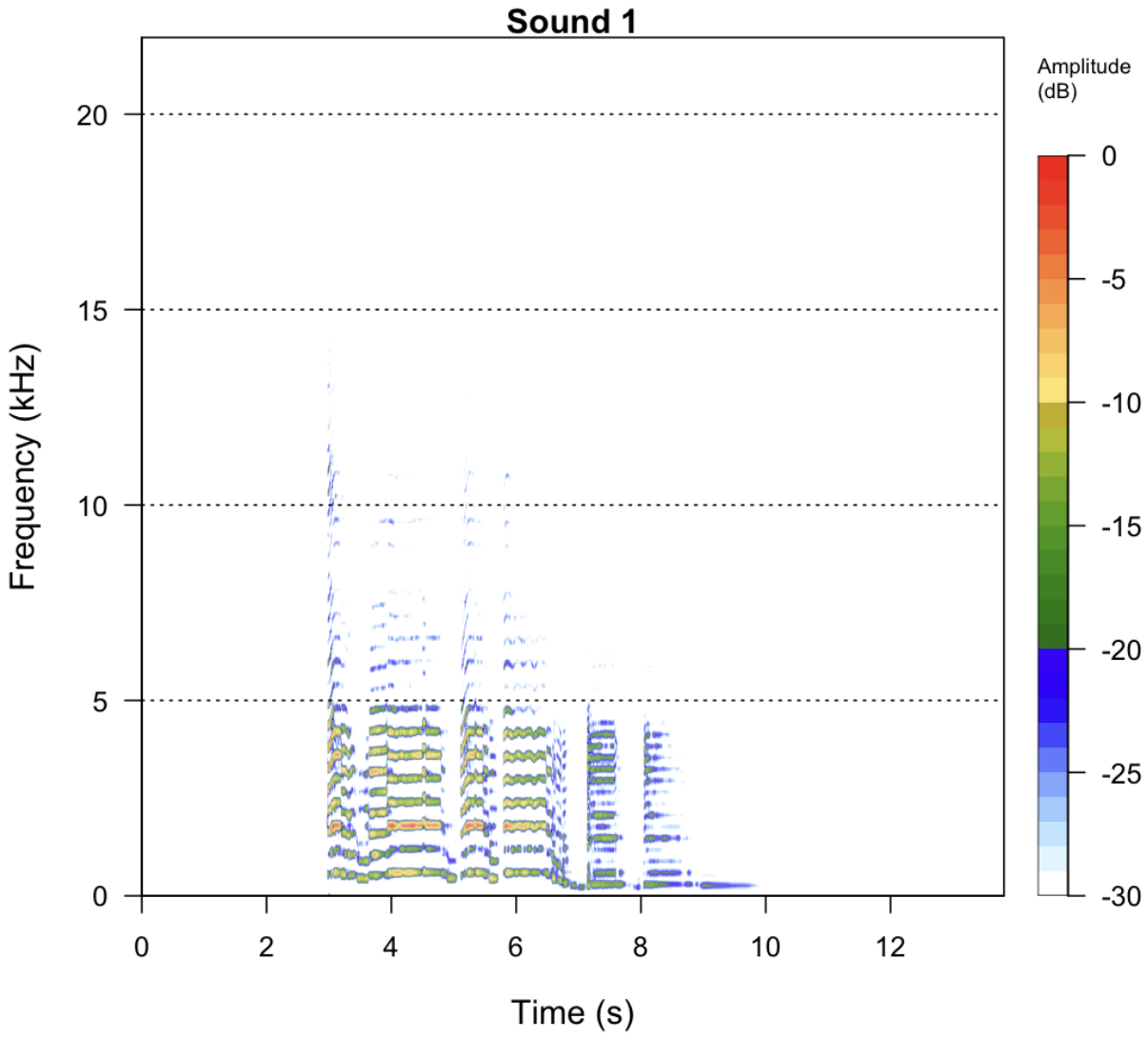
Rango: 0-12kHz.

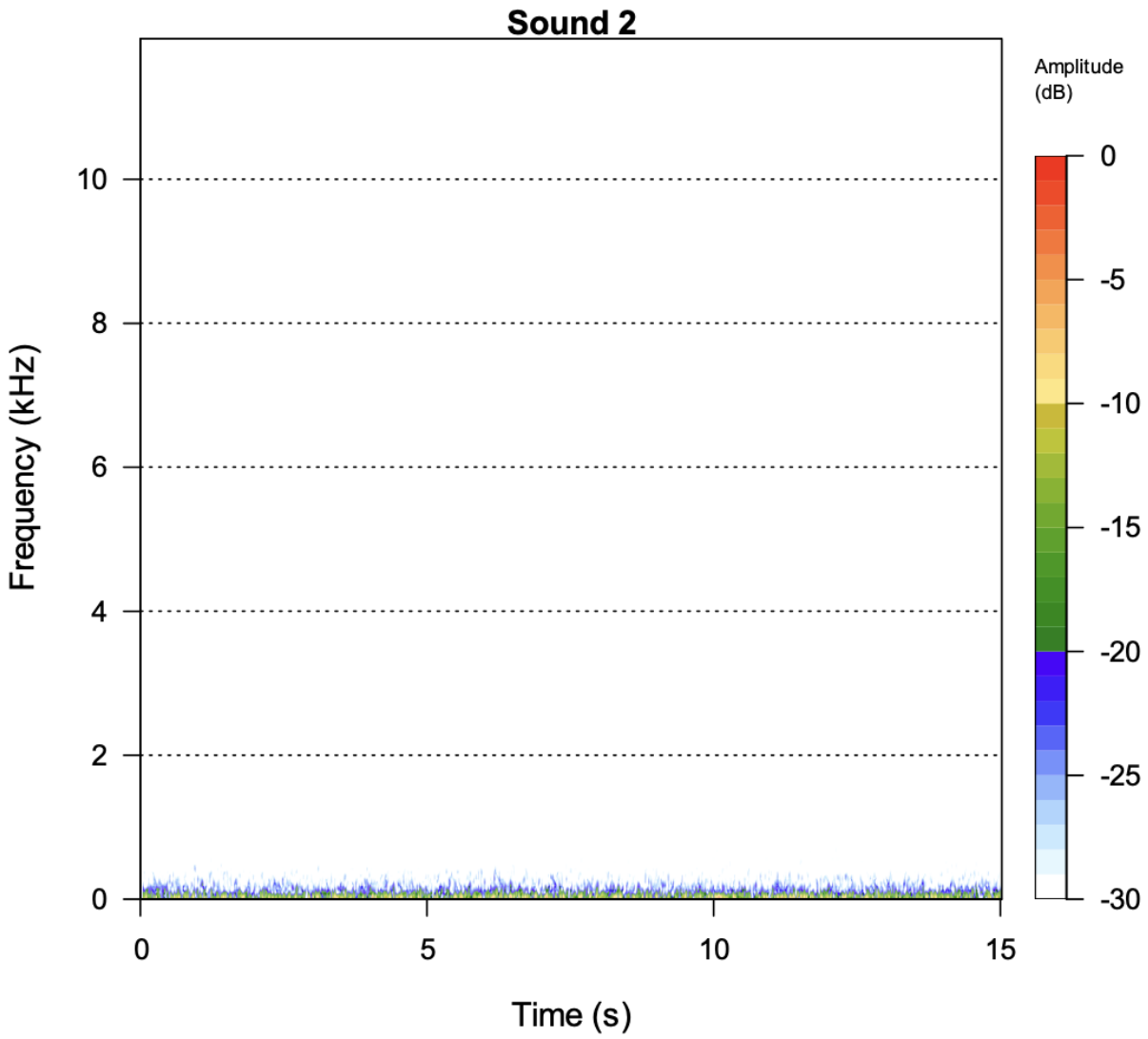


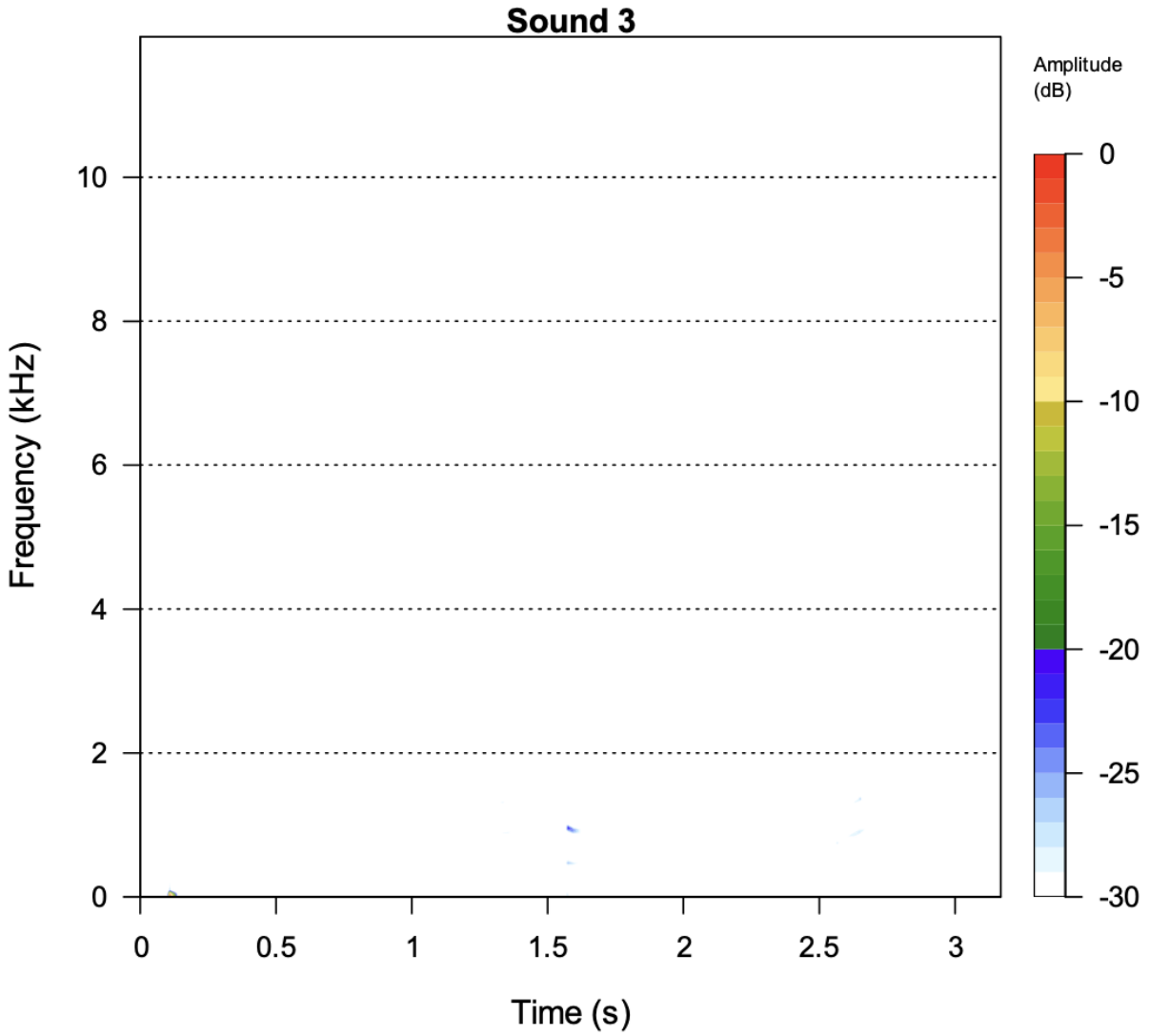
Rango: 0-12kHz.

Las mayores amplitudes respecto a la frecuencia se encuentran entre 0 y 1 kHz para todos los sonidos.

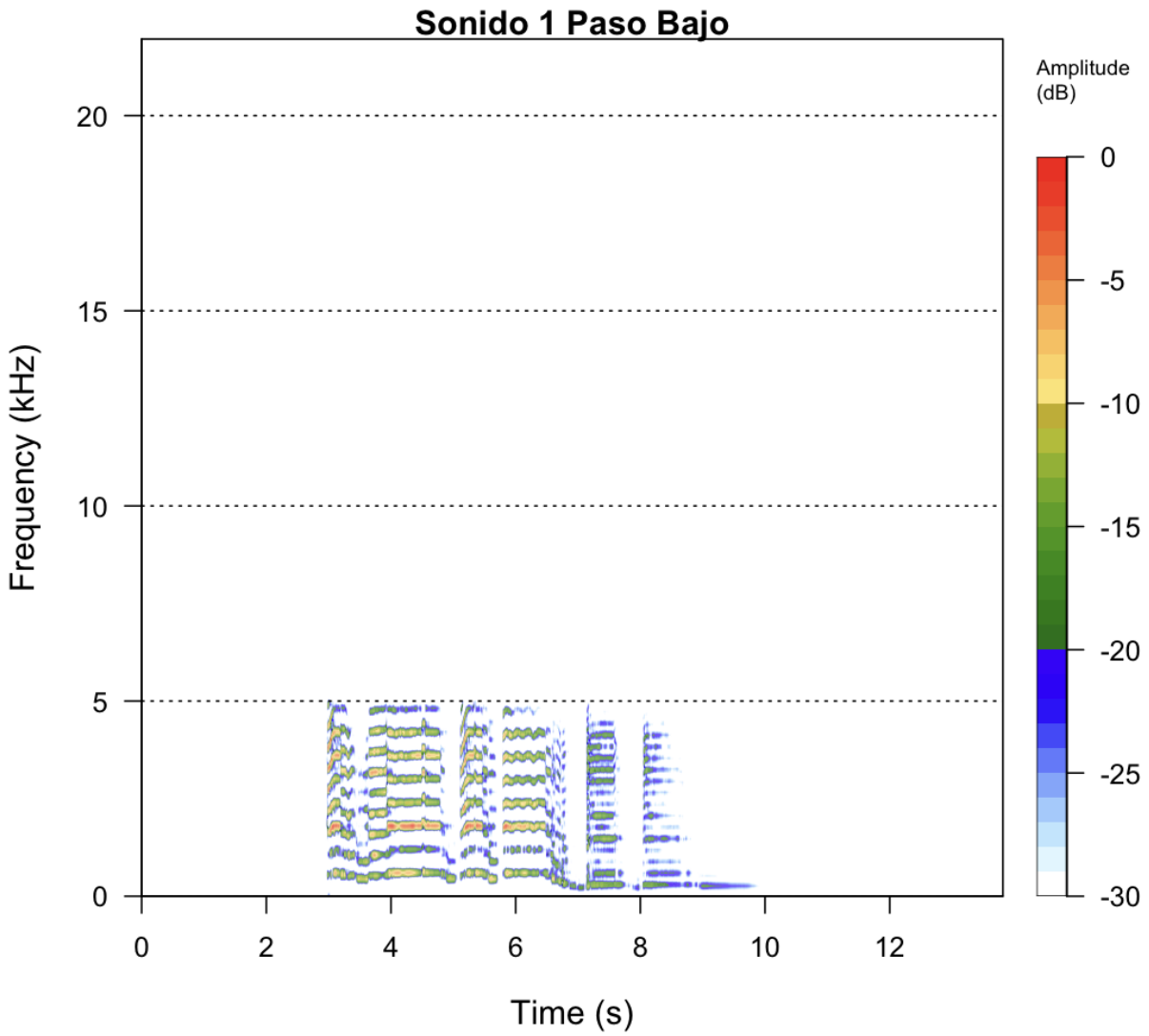
Para realizar los espectrogramas por otro lado, se hace uso de la función **spectro** y su output se presenta a continuación:

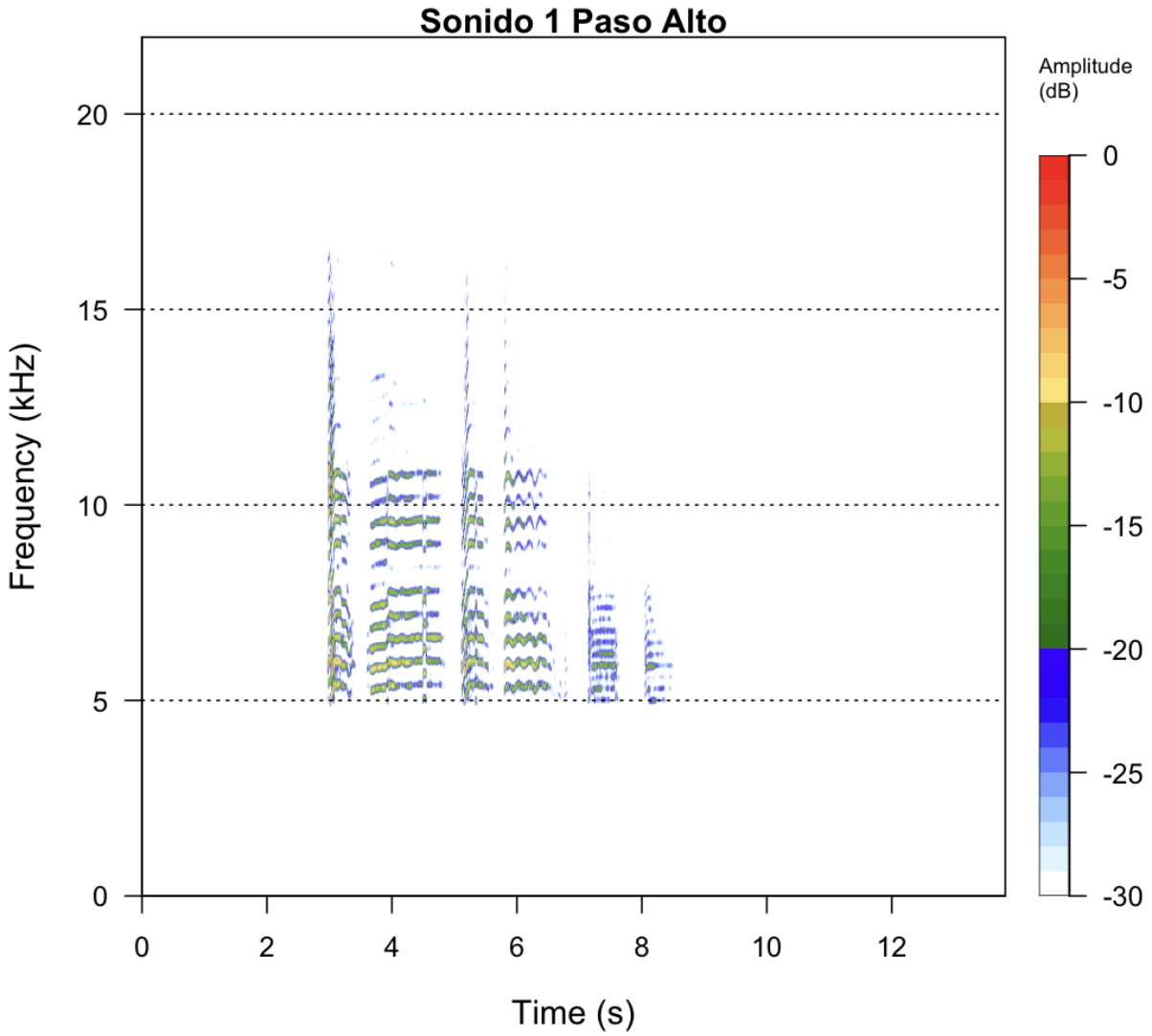

Sound 1
Frequency (kHz)
20
15
10
5
0
0
2
4
6
8
10
12
Time (s)
Amplitude (dB)
0
-5
-10
-15
-20
-25
-30


Sound 2
Frequency (kHz)
10
8
6
4
2
0
0
5
10
15
Time (s)
Amplitude (dB)
0
-5
-10
-15
-20
-25
-30



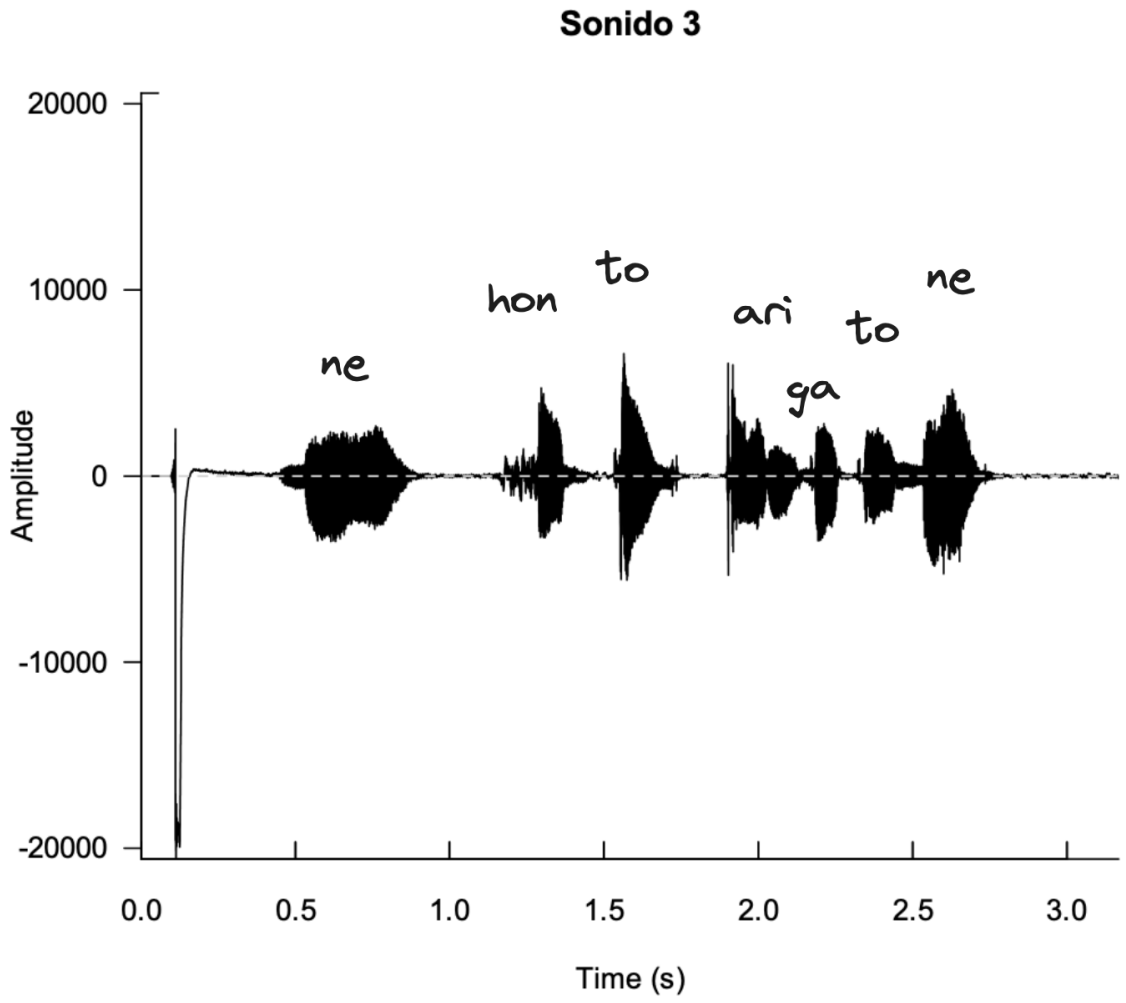
Se usa el sonido 1 para realizar pruebas con el filtro pasa alto y pasa bajo que se presenta en las figuras a continuación:


Sonido 1 Paso Bajo
Frequency (kHz)
20
15
10
5
0
0
2
4
6
8
10
12
Time (s)
Amplitude (dB)
0
-5
-10
-15
-20
-25
-30



Parte C - Análsis

Disclaimer: el sonido 3, correspondiente al habla donde se habla en japonés. Desconozco del idioma fuera de que “arigato” significa gracias, con lo cual puede haber errores en la traducción literal de lo que se escucha en el sonido y luego se representa en la figura donde se agregan las consonantes ubicadas en el oscilograma:



De las tres figuras se puede apreciar cómo se tienen mayores picos y “montañas” en los sonidos que no son un instrumento donde en comparación con el sonido 1 parece más suave, es el que más se asemeja a una función sinusoidal y por esto se podría atribuir el placer que genera la música al oído. Por otro lado, la amplitud más fuerte se encuentra en el ruido marrón, donde a su vez todo se encuentra muy puntiagudo al hacer zoom. Por último se pueden clasificar las consonantes de la siguiente manera:

Oclusivas (o plosivas):

- "t" en "honto".
- "g" en "arigato".

Fricativas:

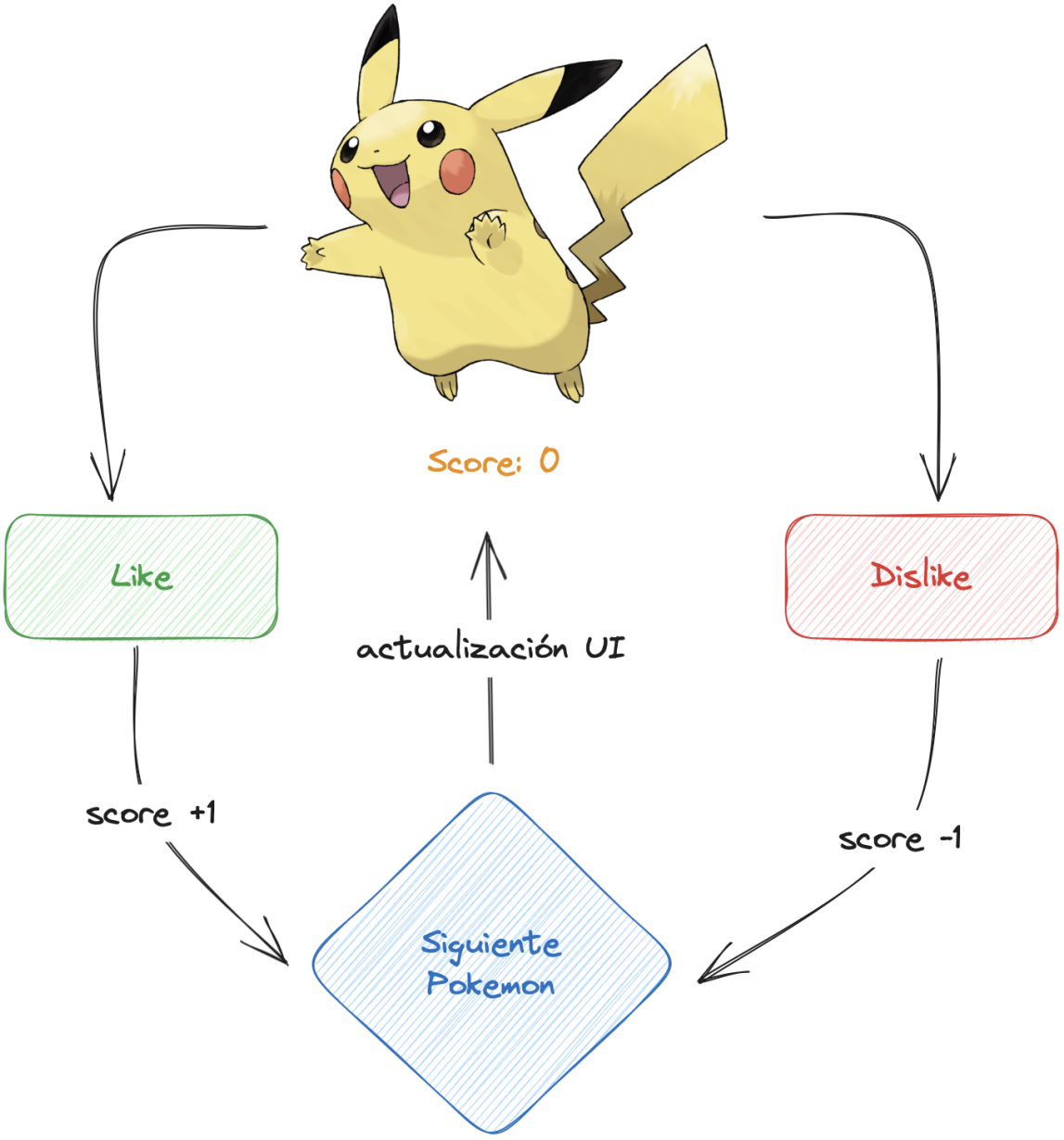
- "h" en "honto".

Nasales:

- "n" en todas las palabras ("ne," "honto," "arigato," "ne").

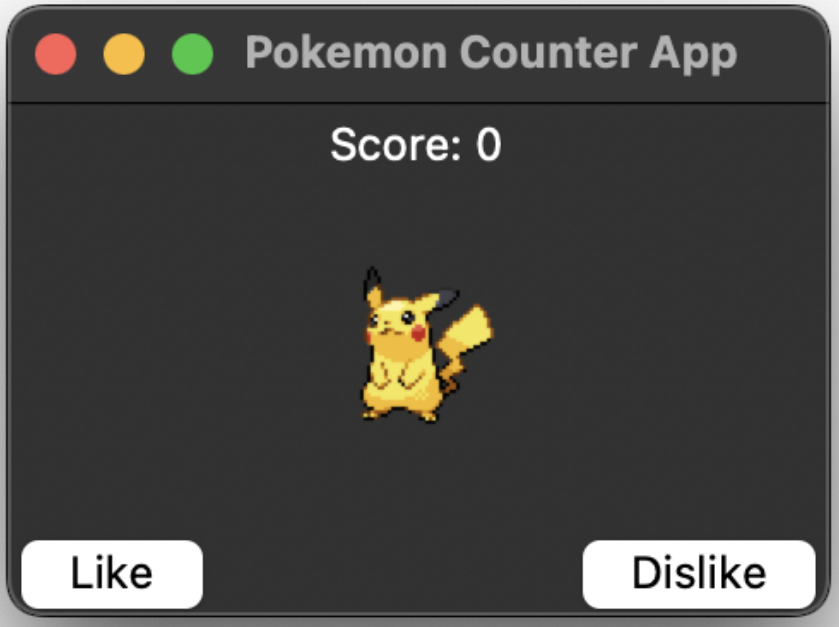
Ejercicio 2 - Interfaces

Parte A - Creación de Interfaz



La interfaz descrita en la figura muestra el principio de f(state)=UI donde en base al estado del contador se produce una actualización de la interfaz para actualizar la UI accorde.

Parte B - Implementación



En su versión actual se cambia entre los siguientes cuatro pokémon:

- Pikachu
- Bulbasaur
- Charmander
- Squirtle

Las sprites (imágenes pequeñas usadas en el juego) se obtienen dinámicamente de la pokeapi y se pueden agregar más pokémon si se lo desea ya que se preserva un array con un objeto que contiene el score y el nombre del pokémon para ser buscado realizando una request HTTP a la API de pokemones. El score es actualizado y se pide el siguiente en el orden en el cual fueron ingresados.

Anexo

- Repositorio público con código correspondiente a la resolución.

Código Ejercicio 1

```
library(seewave)
library(tuneR)
sound1 = readMP3("sax_jazz.mp3")
sound2 = readMP3("brownian_noise.mp3")
sound3 = readMP3("arigato.mp3")
# comando necesario para que funcione en mac
tuneR::setWavPlayer('/usr/bin/afplay')


# vemos la metadata
sound1
sound2
sound3


# oscilograma de los sonidos
oscillo(sound1, title="Sonido 1");axis(side=2, las=2)
oscillo(sound2, title="Sonido 2");axis(side=2, las=2)
oscillo(sound3, title="Sonido 3");axis(side=2, las=2)
```

```
# zoom
oscillo(sound1, from=3, to=3.01, title="🔎 Sonido 1");axis(side=2, las=2)
oscillo(sound2, from=0.1, to=0.2, title="🔎 Sonido 2");axis(side=2, las=2)
oscillo(sound3, from=0.2, to=0.3, title="🔎 Sonido 3");axis(side=2, las=2)

# normalize
sound1B = normalize(sound1,unit='16',level=0.1)
oscillo(sound1B, title="Sonido 1 Normalized");axis(side=2, las=2)

# sound test
play(sound1)
play(sound1B)

# spec
spec(sound1, main="Sonido 1");axis(side=2, las=2)
spec(sound2, main="Sonido 2");axis(side=2, las=2)
spec(sound3, main="Sonido 3");axis(side=2, las=2)

# spectro
spectro(sound1);title(main="Sound 1")
spectro(sound2);title(main="Sound 2")
spectro(sound3);title(main="Sound 3")

# pasa bajo
sound1PB=ffilter(sound1,to=5000,output="Wave")
sound1PB=normalize(sound1PB,unit='16')
spectro(sound1PB);title(main="Sonido 1 Paso Bajo")

# paso alto
sound1PA=ffilter(sound1,from=5000,output="Wave")
sound1PA=normalize(sound1PA,unit='16')
```

```
spectro(sound1PA);title(main="Sonido 1 Paso Alto")
```

Código Ejercicio 2

```
import tkinter as tk
from PIL import Image, ImageTk
import requests
import io

class PokemonCounterApp:
def __init__(self, master):
self.master = master
self.master.title("Pokemon Counter App")

self.pokemon_list = [
{"name": "pikachu", "score": 0},
{"name": "bulbasaur", "score": 0},
{"name": "charmander", "score": 0},
{"name": "squirtle", "score": 0},
]

self.current_index = 0
self.current_pokemon = self.pokemon_list[self.current_index]

self.score_label = tk.Label(master, text=f"Score: {self.current_pokemon['score']}")
self.score_label.pack()

self.image_label = tk.Label(master)
self.image_label.pack()

like_button = tk.Button(master, text="Like", command=self.like)
like_button.pack(side=tk.LEFT)

dislike_button = tk.Button(master, text="Dislike", command=self.dislike)
dislike_button.pack(side=tk.RIGHT)

self.update_display()

def like(self):
self.current_pokemon['score'] += 1
self.next_pokemon()
self.update_display()

def dislike(self):
```

```
self.current_pokemon['score'] -= 1
self.next_pokemon()
self.update_display()

def next_pokemon(self):
self.current_index = (self.current_index + 1) % len(self.pokemon_list)
self.current_pokemon = self.pokemon_list[self.current_index]

def update_display(self):
self.score_label.config(text=f"Score: {self.current_pokemon['score']}")

# Fetch and display Pokemon image
image_url = self.get_pokemon_image_url(self.current_pokemon['name'])
image = self.load_image_from_url(image_url)
self.image_label.config(image=image)
self.image_label.image = image

def get_pokemon_image_url(self, pokemon_name):
api_url = f"https://pokeapi.co/api/v2/pokemon/{pokemon_name}/"
response = requests.get(api_url)
data = response.json()
image_url = data['sprites']['front_default']
return image_url

def load_image_from_url(self, url):
response = requests.get(url)
img_data = response.content
img = Image.open(io.BytesIO(img_data))
img = ImageTk.PhotoImage(img)
return img

if __name__ == "__main__":
root = tk.Tk()
app = PokemonCounterApp(root)
root.mainloop()
```