UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN

FACULTAD DE INGENIERÍA MECÁNICA Y ELÉCTRICA

MANUAL TÉCNICO

Nombre	Matrícula	Carrera
Gaston Tejeda Villaseñor	1889295	ITS

Materia: TÓPICOS SELECTOS DE CIENCIAS DE LA INGENIERÍA II

Maestro: Raymundo Said Zamora Pequeño

Fecha: 20 de Noviembre de 2021

Índice

1. Introducción

En este manual se explicará el funcionamiento del proyecto y de sus componentes, para crear un mejor entendimiento del por que y el como funciona. Se comenzará por explicar el objetivo del proyecto, con el propósito de que el lector entienda que es lo que se trato de lograr, luego explicaremos su estructura, la cual está dividida en módulos y submódulos, para dar lugar a una explicación mas detallada de los componentes que se encuentran en el proyecto. Se dará una breve definición de las funciones utilizadas conforme vayan apareciendo y se hará lo posible por explicar el funcionamiento general de cada módulo y submódulo.

2. Objetivo

El objetivo de este proyecto es realizar un programa que permita analizar mediante índices acústicos los llamados de distintas especies de ranas, para poder estudiar las características de sus poblaciones de manera no invasiva. Se planteo un programa que reciba un archivo de audio en el que se presenten llamados de las especies estudiadas y muestre un grafico que represente los datos obtenidos a partir del audio.

En el documento se utilizan ciertos archivos de audio a los que lamentablemente no tenemos acceso (han sido borrados o ya no se permite el acceso público a estos. En realidad, desconocemos porque no se encuentran en la web indicada en el documento) por lo que utilizaremos una serie de audios recopilados de una biblioteca virtual (https://sounds.bl.uk/Environment/Amphibians) de archivos de audios referentes a distintas especies de anfibios, entre estos, ranas y sapos. No nos fue posible encontrar audios de las especies mencionadas en el documento original, pero pudimos recrear la situación por medio de 2 especies que se encuentran en el mismo ecosistema (hecho demostrado por los audios en que aparecen estas 2 especies simultáneamente) y comparten características similares en sus llamados con los utilizados en el documento original, por ejemplo, en el documento original se menciona que la especie *Litoria rothii* tiene un llamado de pulso múltiple y constante, tal y como nuestra Bufo americanus : *American Toad – Bufonidae*, mientras que el resto de las especies utilizadas en el documento original y en nuestro proyecto cuentan con llamados de pulso múltiple (véase ***imágenes 1.1.1, 1.1.2 y 1.1.3***). Las especies que se analizarán en este proyecto son Bufo americanus : *American Toad – Bufonidae, Hyla crucifer : Spring Peeper – Hylidae* y *Hyla versicolor : Grey Tree Frog - Hylidae* (a quienes nos referiremos únicamente como American Toad, Spring Peeper y Grey Treefrog de aquí en adelante por motivos de practicidad) cuyos llamados se encuentran entre los 1200 Hz y 1800 Hz para la American Toad, entre los 2100 Hz y 3000 Hz para la Spring Peeper y entre los 1500 Hz – 2200 Hz para la Grey Treefrog, datos determinados mediante investigación previa (véase el reporte de investigación anexado junto al proyecto), y para cualquier llamado de otra especie o fuera de estos rangos de frecuencia no se pueden garantizar resultados óptimos.

Los índices utilizados para analizar los llamados de las especies seleccionadas son el índice de complejidad acústica (ACI) y el índice de entropía temporal (th) Estos fueron determinados a partir de los resultados de la investigación previa.

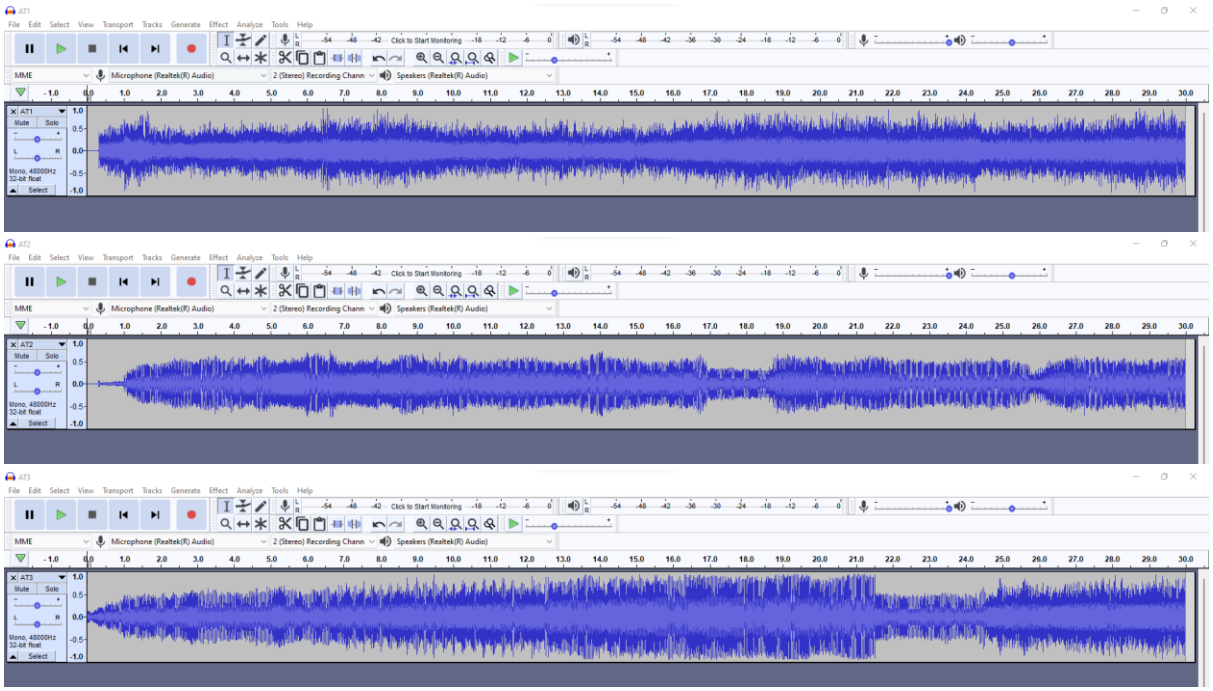


Imagen 1.1.1 – *Fragmentos de audio de la American Toad.*

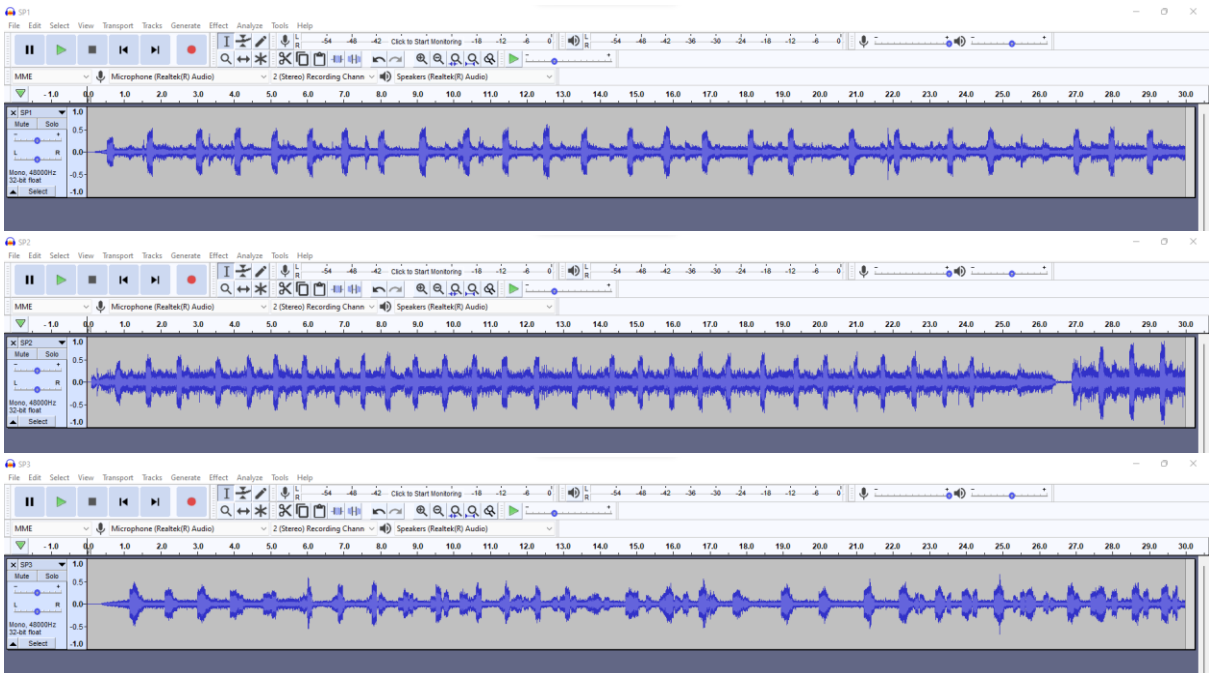


Imagen 1.1.2 – *Fragmentos de audio de la Spring Peeper.*

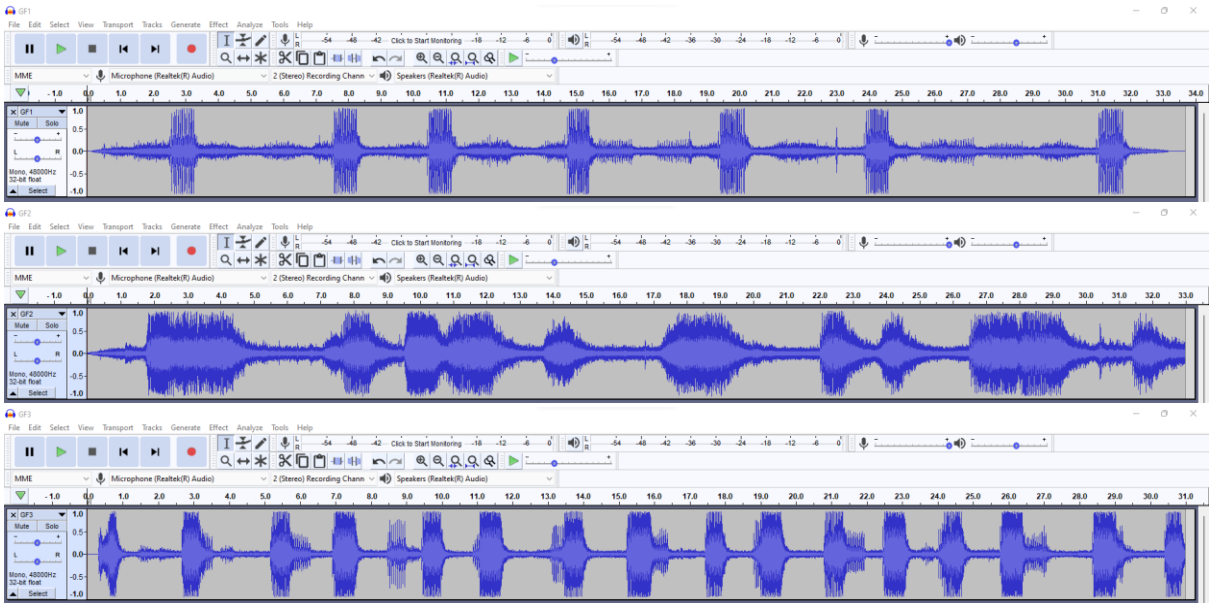


Imagen 1.1.3 – *Fragmentos de audio de la Grey Treefrog.*

3. Estructura

Este proyecto fue desarrollado en RStudio con encoding recomendado de UTF-8.

3.1. Librerías

Las librerías utilizadas para el proyecto son shiny, seewave, audio, tuneR y vegan.

```
1  library(shiny)
2  library(seewave)
3  library(audio)
4  library(tuneR)
5  library(vegan)
```

Imagen 2.1.1 – *Librerías de R utilizadas en el proyecto.*

- Shiny: es un paquete que facilita construir aplicaciones interactivas directamente desde R.
- Seewave: paquete que contiene funciones para analizar, manipular, mostrar, editar y sintetizar ondas de tiempo (sonidos particularmente).
- Audio: Interfaces para dispositivos de audio (principalmente basados en muestras) de R para permitir la grabación y reproducción de audio.
- TuneR: Analiza música y discursos, extrae características como MFCC's, manipula archivos de onda y sus representaciones de varias maneras, lee mp3, lee midi, realiza pasos de transcripción, etc.
- Vegan: Métodos de ordinación, análisis de diversidad y otras funciones para ecologistas.

3.2. Interfaz de Usuario

Para la interfaz de usuario se dividió el display en 3 secciones:

- 1 *side panel* o panel de entradas para los inputs del proyecto.
- 1 *main panel* o panel de salidas para los resultados u outputs del proyecto.
- 1 panel de titulo para mostrar el nombre del proyecto.

```
 9  ui<-fluidPage(
10
11    tags$head(tags$style(
12      HTML('
13                 #sidebar {
14                    background-color: #67b5db;
15                }
16
17                #Panel {
18                    background-color: #f8a8ff;
19                }
20
21                body, label, input, button, select {
22                font-family: "Arial";
23                }')
24    )),
```

Imagen 2.2.1 – *Declaración de estilos de interfaz de usuario.*

3.2.1. Panel de titulo

Para el panel de titulo solamente declaramos su contenido con 1 sola instrucción.

```
titlePanel(
  h1("Proyecto Integrador - Analizdor de llamados de anfiios", align = "center")
),
```

Imagen 2.2.2 – *Declaración de Título para el panel de título.*

3.2.2. Panel de entradas

En el panel de entradas se declaran todas las entradas de datos que utilizará nuestro proyecto.

```
sidebarPanel(id="sidebar",

             fluidRow(
               selectInput(inputId = "ModeSelect",
                           h3("Seleccionar modo"),
                           choices=list("Carga de datos","Ejecutar Analisis"))
             ),

             uiOutput("mode"),

),

```

Imagen 2.2.3 – *Declaración de entradas para el panel de entradas.*

3.2.3. Panel de salidas

Para el panel de salida se declararon los siguientes campos:

- Título del panel: en este campo se determina el titulo que se mostrara en el panel.
- Tabla de archivo de audio: objeto de tipo tabla en que se mostraran los datos del archivo de audio seleccionado para analizar.
- Descripción de grafico de resultados: este en realidad son 2 campos, que muestran un alinea de texto que funciona como descripción de los valores que se mostrarán en el resultado del análisis.
- Gráfico resultado: objeto tipo plot que muestra el grafico del resultado del análisis.

```
56      mainPanel(
57        id="Panel",
58        h3("Rangos de Frecuencia de las especies analizadas"),
59        textOutput("rangeAT"),
60        textOutput("rangeSP"),
61        textOutput("rangeGF"),
62        h3("Resultados graficos"),
63        tableOutput("name1"),
64        textOutput("colorAT"),
65        textOutput("colorSP"),
66        textOutput("colorGF"),
67        plotOutput("plot1"),
68        textOutput("TDUR_AT"),
69        textOutput("TDUR_SP"),
70        textOutput("TDUR_GF"),
71        textOutput("LOUD_AT"),
72        textOutput("LOUD_SP"),
73        textOutput("LOUD_GF"),
74        textOutput("CALL_AT"),
75        textOutput("CALL_SP"),
76        textOutput("CALL_GF"),
77        textOutput("STNR_AT"),
78        textOutput("STNR_SP"),
79        textOutput("STNR_GF"),
80        tableOutput("name2"),
81        plotOutput("plot2"),
82        tableOutput("name3"),
83        plotOutput("plot3"),
84      )
```

Imagen 2.2.4 – *Declaración de salidas para el panel de salidas.*

3.3. Funciones de servidor

En este módulo del proyecto se encuentran las funciones que se ejecutaran durante la ejecución del proyecto. Es declarado por la línea de código de la ***imagen 1.3.1***.

```
68 server <- function(input,output){
```

Imagen 2.3.1 – *Declaración de módulo de funciones de servidor.*

3.3.1. Lectura de archivo de audio

La lectura del archivo de audio se declara como una función reactiva para que se active al seleccionar un archivo en el campo de carga de archivo.

```
70   data<-reactive({
71     readWave(input$Audio$datapath)
72   })
```

Imagen 2.3.2 – *Función de lectura de archivo.*

Se utiliza la función readWave() que recibe como parámetro la dirección de un archivo de audio y lo transforma en un objeto de R tipo wave.

3.3.2. Inicio de análisis

Este modulo se activa en función al botón de Inicio de Análisis (como se demuestra en la ***imagen 2.3.3***).

```
74   observeEvent(input$btnIniciar,{
```

Imagen 2.3.3 – *Declaración de función dependiente accionada por el botón Inicio de Análisis.*

3.3.3. Demostración de información de archivo de audio.

Se muestran los datos del archivo de audio cargado.

```
76     output$name<-renderTable({
77       input$Audio
78     })
```

Imagen 2.3.4 – *Carga de datos del archivo de audio al objeto tipo tabla del panel de salidas.*

3.3.4. Declaración de constantes

Se declaran las constantes que se utilizarán a lo largo del proyecto:

- duration: variable que contiene la duración de cada segmento en que se dividirá el archivo original de audio para su posterior análisis en segundos. Valor = 30.
- minfAT: frecuencia mínima en que se encuentran los llamados de la American Toad en Hz. Valor = 1200.
- maxfAT: frecuencia mínima en que se encuentran los llamados de la American Toad en Hz. Valor = 1800.
- minfASP: frecuencia mínima en que se encuentran los llamados de la Spring Peeper en Hz. Valor = 2100.
- maxfSP: frecuencia mínima en que se encuentran los llamados de la Spring Peeper en Hz. Valor = 3000.
- minfAGF: frecuencia mínima en que se encuentran los llamados de la Grey Treefrog en Hz. Valor = 1500.
- maxfGF: frecuencia mínima en que se encuentran los llamados de la Grey Treefrog en Hz. Valor = 2200.

```
 9  ##CONSTANTES
10  duration <- 30
11  minfAT <- 1200
12  maxfAT <- 1800
13  minfSP <- 2100
14  maxfSP <- 3000
15  minfGF <- 1500
16  maxfGF <- 2200
17
```

Imagen 2.3.5 – *Declaración de constantes para el análisis.*

3.3.5. Segmentación de Audio

Para la segmentación del audio primero creamos una variable de tipo lista llamada *segmentos* que contendrá los segmentos del audio como objetos tipo wave. Para determinar la cantidad

de segmentos se divide la duración del audio en segundos (almacenada en la variable *sound_lenght*) entre la duración de los segmentos (constante *duration*).

Se crea la variable de tipo secuencia *start_times*, que guarda una secuencia desde 0 hasta el tiempo máximo de segmentación (almacenado en *mxtime*), avanzando en intervalos de valor igual a la duración de cada segmento. Esta variable nos servirá para declarar los valores de tiempo de inicio y fin a la hora de segmentar el audio mediante un ciclo for como se muestra en la ***imagen 2.3.6***.

```
87      ##CODIGO DE ANALISIS
88        #Segmentado de audio
89      segmentos = list()
90      sound_length <- round(length(data()@left) / data()@samp.rate, 2)
91      cantsegm <- round(sound_length / duration)
92      mxtime <- duration * cantsegm
93      start_times = seq(0, mxtime, by = duration)
94
95      for (i in seq_along(start_times)) {
96        segmentos[[i]] = readWave(input$Audio$datapath,
97                                  from = start_times[i],
98                                  to = start_times[i] + duration,
99                                  units = "seconds")
100     }
```

Imagen 2.3.6 – *Submódulo de segmentación de audio.*

3.3.6. Calcular características de los bins

Los bins son fragmentos de estudio sacados a partir de los segmentos. Estos se determinan de la siguiente manera:

La cantidad total de bins se obtiene al dividir el frame máximo del espectrograma del audio a analizar entre dos. Después se obtiene el rango de frecuencia en Hz que abarcará cada bin (almacenado en *fbins*) dividiendo la frecuencia máxima del audio a analizar, obtenida mediante la función fpeaks que rescata los picos de frecuencia del audio objetivo a partir de su espectrograma.

Después determinamos el rango de bins en que se encuentran los llamados de cada especie a analizar (almacenados en maxbinAT, minbinAT, binrangeAT, maxbinSP, minbinSP, binrangeSP, maxbinGF, minbinGF, binrangeGF) dividiendo su frecuencia máxima y mínima entre el rango de frecuencia de cada bin (maxbinXX y minbinXX) y restando el máximo al mínimo (binrangeXX).

```
146         #Calcular caracteristicas de los bins
147       spectrograma <- spec(data(), plot = FALSE)
148       idmax <- which.max(spectrograma[,1])
149       totframes <- spectrograma[idmax, 1]
150       cantbins <- ceiling(totframes * 1000) / 2
151       maxf <- max(fpeaks(meanspec(data(), plot = FALSE), plot = FALSE)) * 1000
152       fbins <- maxf / cantbins
153       maxbinAT <- ceiling(maxfAT/fbins)
154       maxbinSP <- ceiling(maxfSP/fbins)
155       maxbinGF <- ceiling(maxfGF/fbins)
156       minbinAT <- floor(minfAT/fbins)
157       minbinSP <- floor(minfSP/fbins)
158       minbinGF <- floor(minfGF/fbins)
159       binrangeAT <- maxbinAT - minbinAT
160       binrangeSP <- maxbinSP - minbinSP
161       binrangeGF <- maxbinGF - minbinGF
```

Imagen 2.3.7 – *Submódulo de Calcular Características de los bins.*

3.3.7. Limpiar segmentos

Ahora debemos aislar los llamados de cada especie a partir de sus rangos de frecuencia de llamado. Para esto declaramos las variables cleansegmAT, cleansegmSP y cleansegmGF (a partir de ahora, los fragmentos de audio estarán separados según el rango de frecuencia de llamado de la especie a la que se enfocan) como objetos tipo lista y los llenamos utilizando un ciclo for que se repite tantas veces como hay segmentos de audio y la función ffilter(), la cual permite filtrar un objeto tipo wave según un rango de frecuencia determinado (en este caso, los rangos de frecuencia de llamado de cada especie).

```
64  #limpiar segmentos
65  cleansegmAT <- list()
66  cleansegmSP <- list()
67  cleansegmGF <- list()
68  dirtysegmAT <- list()
69  dirtysegmSP <- list()
70  dirtysegmGF <- list()
71  for (i in 1:cantsegm) {
72    cleansegmAT[[i]] = ffilter(segmentos[[i]], from = minfAT, to = maxfAT, bandpass = TRUE, output = "Wave", listen = FALSE)
73    cleansegmSP[[i]] = ffilter(segmentos[[i]], from = minfSP, to = maxfSP, bandpass = TRUE, output = "Wave", listen = FALSE)
74    cleansegmGF[[i]] = ffilter(segmentos[[i]], from = minfGF, to = maxfGF, bandpass = TRUE, output = "Wave", listen = FALSE)
75    dirtysegmAT[[i]] = ffilter(segmentos[[i]], from = minfAT, to = maxfAT, bandpass = FALSE, output = "Wave", listen = FALSE)
76    dirtysegmSP[[i]] = ffilter(segmentos[[i]], from = minfSP, to = maxfSP, bandpass = FALSE, output = "Wave", listen = FALSE)
77    dirtysegmGF[[i]] = ffilter(segmentos[[i]], from = minfGF, to = maxfGF, bandpass = FALSE, output = "Wave", listen = FALSE)
78  }
79
```

Imagen 2.3.8 – *Submódulo de Limpiar Segmentos.*

3.3.8. Separar bins

Ahora vamos a separar los segmentos de audio por bins. Para esto declaramos 2 variables de tipo lista (binsAT, binsSP y binsGF) que contendrán los bins de cada especie analizada. Para saber cuántos bins habrá en total para cada especie multiplicamos la cantidad de segmentos por el rango de bins de su respectiva especie.

Para generar los bins utilizaremos un ciclo for que se repite tantas veces como hay bins por especie, apoyándonos de 2 variables contadores para el bin por segmento y para el audio limpiado original, además de la función ffilter().

```
#Separar bins
binsAT <- list()
binsSP <- list()
binsGF <- list()
cantbinsAT <- cantsegm * binrangeAT
cantbinsSP <- cantsegm * binrangeSP
cantbinsGF <- cantsegm * binrangeGF
#AT
cont <- minbinAT
contcleans <- 1
for (i in 1:cantbinsAT) {
  binsAT[[i]] = ffilter(cleansegmAT[[contcleans]], from = cont * fbins, to = (cont + 1) * fbins, bandpass = TRUE, output = "
  if(cont < maxbinSP){
    cont <- cont + 1
  } else {
    cont <- minbinAT
    contcleans <- contcleans + 1
  }
}
```

Imagen 2.3.9 – *Submódulo de Separar bins (Solo se muestra hasta los de American Toad, pero el proceso se repite para las otras dos especies).*

3.3.9. Calcular índices

Para calcular los índices primero declaramos tres variables tipo lista (INDEX_AT, INDEX_SP e INDEX_GF) de tipo lista que contendrán los índices de cada especie. El proceso siguiente va a depender del índice (o combinación de estos) seleccionado en las entradas.

3.3.9.1. Combinación de índices ACI-th

Para esta combinación se determina primero la cantidad de índices totales por especie (cantidad total de bins por especie, multiplicada por la cantidad de índices, ya que cada se calcularán índices por cada bin) y se almacenan en las variables cantindexAT, cantindexSP y cantindexGF.

Después se realiza un ciclo for para cada especie, que se repite tantas veces como índices por especie (cantindexXX) y se calculan los índices ACI y th mediante las funciones ACI() y th() que calculan los índices de complejidad acústica y de amplitud respectivamente para un objeto tipo wave.

```
#Calculamos indices
INDEX_AT <- list()
INDEX_SP <- list()
INDEX_GF <- list()
if(indx == "ACI-th"){
  cantindexAT <- cantbinsAT * 2
  cantindexSP <- cantbinsSP * 2
  cantindexGF <- cantbinsGF * 2
  #AT
  for(i in 1:cantindexAT){
    if(i %% 2 == 0){
      INDEX_AT[[i]] <- th(env(binsAT[[i/2]], plot = FALSE))
    } else {
      INDEX_AT[[i]] <- ACI(binsAT[[(i + 1)/2]])
    }
  }
  #SP
  for(i in 1:cantindexSP){
    if(i %% 2 == 0){
      INDEX_SP[[i]] <- th(env(binsSP[[i/2]], plot = FALSE))
    } else {
      INDEX_SP[[i]] <- ACI(binsSP[[(i + 1)/2]])
    }
  }
  #GF
```

Imagen 2.3.10 – *Fragmento del submódulo de Calcular índices / índices ACI-th.*

3.3.9.2. Índice ACI

Se sigue el mismo proceso que para la combinación de ACI-th (véase apartado 3.3.9.1) pero las variables cantindexXX ya no serán necesarias (debido a que su valor será el mismo a la cantidad de bins ya que solo se calcula un índice por bin) y solamente se calcula el índice ACI mediante la función ACI() que calcula el índice de complejidad para un objeto tipo wave.

```
156   } else if(indx == "ACI"){
157     #AT
158     for(i in 1:cantbinsAT){
159       INDEX_AT[[i]] <- ACI(binsAT[[(i + 1)/2]])
160     }
161     #SP
162     for(i in 1:cantbinsSP){
163       INDEX_SP[[i]] <- ACI(binsSP[[(i + 1)/2]])
164     }
165     #GF
166     for(i in 1:cantbinsGF){
167       INDEX_GF[[i]] <- ACI(binsGF[[(i + 1)/2]])
168     }
```

Imagen 2.3.11 – *Submódulo de Calcular índices / índice ACI.*

3.3.9.3. Índice th

Se sigue el mismo proceso que para la combinación de ACI-th (véase apartado 3.3.9.1) pero las variables cantindexXX ya no serán necesarias (debido a que su valor será el mismo a la cantidad de bins ya que solo se calcula un índice por bin) y solamente se calcula el índice M mediante la función th() que calcula el índice amplitud para un objeto tipo wave.

```
169   } else {
170     #AT
171     for(i in 1:cantbinsAT){
172       INDEX_AT[[i]] <- th(env(binsAT[[(i + 1)/2]], plot = FALSE))
173     }
174     #SP
175     for(i in 1:cantbinsSP){
176       INDEX_SP[[i]] <- th(env(binsSP[[(i + 1)/2]], plot = FALSE))
177     }
178     #GF
179     for(i in 1:cantbinsGF){
180       INDEX_GF[[i]] <- th(env(binsGF[[(i + 1)/2]], plot = FALSE))
181     }
182   }
```

Imagen 2.3.12 – *Submódulo de Calcular índices / índice M.*

3.3.10. Graficación

Se crea la variable cantindx, cuyo valor es 1 a menos que se haya elegido una combinación de índices en las entradas, y se generan matrices a partir de las listas de índices generadas en los módulos anteriores (INDEX_AT, INDEX_SP e INDEX_GF) con cantidad de filas iguales a la cantidad de segmentos generados a partir del audio a analizar y cantidad de columnas igual al rango de bins de cada especie (binrangeXX) multiplicado por la cantidad de índices (cantindx), ordenada por filas, utilizando la función matrix() que sirve para generar matrices a partir de objetos tipo lista o vectores (es necesario utilizar la función unlist() en nuestra lista

de índices para convertirla en un vector, ya que la función metaMDS usada a continuación no maneja de manera correcta los objetos de tipo lista).

Después utilizamos la función metaMDS(), que realiza escalados multidimensionales no paramétricos e intenta encontrar una solución estable usando múltiples inicios aleatorios, para realizar un escalado multidimensional no paramétrico de los índices calculados para cada especie.

Por último, utilizamos ordiplot() (Grafico ordinal, alternativa a la función plot e identifica funciones por ordinación) y par() para generar una comparación de los escalados generados por la función metaMDS() de manera gráfica y la enviamos al objeto plot en el panel de salidas.

```
##GRAFICACION
output$colorAT <- renderText({
  outputAT <- "American Toad -> Verde"
})
output$colorSP <- renderText({
  outputSP <- "Spring Peeper -> Rojo"
})
output$colorGF <- renderText({
  outputGF <- "Grey Treefrog -> Azul"
})

cantindx <- 1
if(indx == "ACI-th"){
  cantindx <-2
}
mxAT <- matrix(unlist(INDEX_AT), nrow = cantsegm, ncol = binrangeAT * cantindx , byrow = TRUE)
MDS_AT <- metaMDS(mxAT)
mxSP <- matrix(unlist(INDEX_SP), nrow = cantsegm, ncol = binrangeSP * cantindx , byrow = TRUE)
MDS_SP <- metaMDS(mxSP)
mxGF <- matrix(unlist(INDEX_GF), nrow = cantsegm, ncol = binrangeSP * cantindx , byrow = TRUE)
MDS_GF <- metaMDS(mxGF)

ordiplot(MDS_AT, type = "n", main = indx)
orditorp(MDS_AT, display = "species", labels = F, pch = c(16), col = c("green"), cex = 1)
par(new=TRUE)
ordiplot(MDS_SP, type = "n", main = indx)
orditorp(MDS_SP, display = "species", labels = F, pch = c(16), col = c("red"), cex = 1)
par(new=TRUE)
ordiplot(MDS_GF, type = "n", main = indx)
orditorp(MDS_GF, display = "species", labels = F, pch = c(16), col = c("blue"), cex = 1)
```

Imagen 2.3.13 – *Submódulo de Graficación.*

3.3.11. Extracción de características de los llamados

Para extraer las características de los llamados nos apoyaremos de la función env() la cual produce un vector con valores que representan la envoltura de amplitud de un objeto tipo wave. Las características que se extraerán son Duración Total, Volumen, Calling Rate y Signal to Noise Ratio (SNR). Se obtendrá la máxima amplitud del llamado limpiado previamente (véase apartado 3.3.7) se recorerá en orden cronológico los valores de amplitud registrados. Secrea una variable booleana llamada lastStatus y currStatus las cuales indicaran si la iteración anterior se encontraba registrando una vocalización o si se encontraba en silencio o ruido. La primera vez que se registre un valor mayor al %60 de la amplitud máxima y se haya registrado un estado de llamado falso en el registro anterior se considerará el inicio de una vocalización, mientras que la primera vez que se registre una amplitud menor al %50 de la amplitud máxima y se haya registrado un estado de llamado verdadero en el registro anterior se considerará el final de una vocalización.

```
##EXTRACCION DE CARACTERISTICAS
envAT <- list()
envSP <- list()
envGF <- list()
envAT_NOISE <- list()
envSP_NOISE <- list()
envGF_NOISE <- list()
for(i in 1:cantsegm){
  envAT[[i]] <- env(cleansegmAT[[i]], msmooth = c(4785,90), plot = FALSE)
  envSP[[i]] <- env(cleansegmSP[[i]], msmooth = c(4785,90), plot = FALSE)
  envGF[[i]] <- env(cleansegmGF[[i]], msmooth = c(4785,90), plot = FALSE)
  envAT_NOISE[[i]] <- env(dirtysegmAT[[i]], plot = FALSE)
  envSP_NOISE[[i]] <- env(dirtysegmSP[[i]], plot = FALSE)
  envGF_NOISE[[i]] <- env(dirtysegmGF[[i]], plot = FALSE)
}
```

Imagen 2.3.14 – *Obtención de envolturas para ruido y llamados para extracción de características.*

3.3.11.1. Duración Total

La duración total es el tiempo total desde la primera vocalización hasta la última. Para obtener la duración total, creamos un objeto tipo lista, ya que se debe recuperar para cada segmento y luego se sacará un promedio de los resultados. Se guarda el valor de tiempo en el que se registra el inicio de la primera vocalización y el valor de tiempo del final de la ultima vocalización, luego estos valores se restan y el resultado es la duración total.

```
for(i in 1:cantsegm){
  #AT
  #Loudness
  loudnessAT[[i]] <- colMeans(envAT[[i]])
  #Total Duration #Call Rate
  callrateAT[[i]] <- 0
  mxAmpAT[[i]] <- max(envAT[[i]])
  mxAmpAT_NOISE[[i]] <- max(envAT_NOISE[[i]])
  currState <- FALSE
  lastState <- FALSE
  ini <- nrow(envAT[[i]]) / 100
  fin <- nrow(envAT[[i]]) / 100
  for(j in 1:nrow(envAT[[i]])){
    lastState <- currState
    if(envAT[[i]][[j]] > mxAmpAT[[i]] * lim){
      currState <- TRUE
      lim <- 0.5
      if(!lastState){
        callrateAT[[i]] <- callrateAT[[i]] + 1
        if(j/100 < ini){
          ini <- j/100
        }
      }
    } else {
      currState = FALSE
      lim <- 0.6
      if(lastState){
        fin <- j/100
      }
    }
```

Imagen 2.3.15 – *Fragmento de la extracción de características para la American Toad.*

3.3.11.2. Volumen

Es el promedio de la apmlidos en la actividad de llamado de la especie. Para ello simplemente se calculó un promedio de los valores de amplitud del segmento previamente limpiado (véase ***imagen 2.3.15*** bajo el comentario "#Loudness").

3.3.11.3. Call Rate

Es el número de vocalizaciones dentro de un periodo de 30 segundos. Se utilizaron las vocalizaciones calculadas para generar esta información.

3.3.11.4. Signal to Noise Ratio (SNR)

Es la diferencia en decibeles entre la máxima envoltura de amplitud y de el ruido de fondo en un segmento de 30 segundos. Para esto se utilizaron filtros para obtener fragmentos de audio con solamente ruido y se compararon las amplitudes máximas.

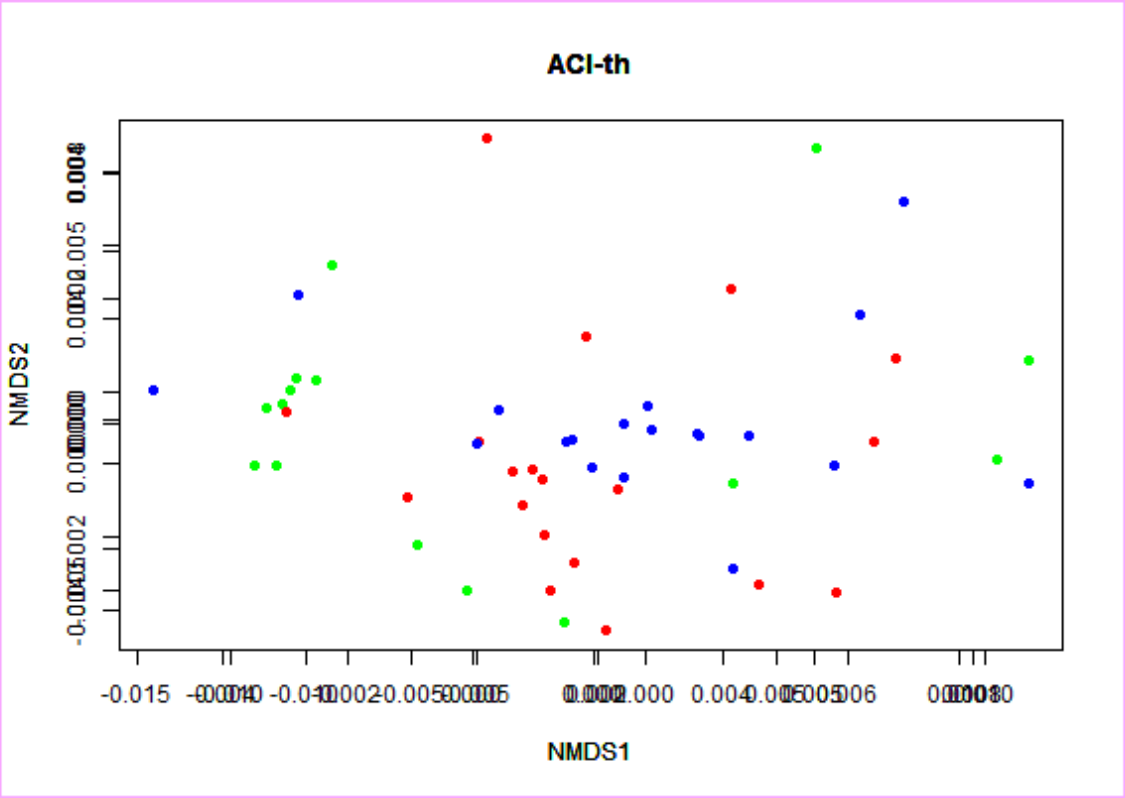
4. Resultados

Los resultados obtenidos distan de los generados en el documento original, principalmente en los gráficos NMDS, pero esto se atribuye a que las especies analizadas, a pesar de que se hizo todo lo posible por que fueran lo más similares posibles a los originales, en cuanto a características de llamado se refiere, siguen sin ser las mismas especies ni los mismos llamados (en el documento originar los llamados llegan a tener frecuencias de hasta 6kHz, mientras que nuestras especies en ningún momento lograron superar los 3 kHz). A pesar de esto no creemos que nuestros resultados puedan ser desacreditados, pues si siguió el proceso correctamente y estamos contentos con los resultados. Al final podemos seguir comprobando que el índice ACI() es mas eficaz a la hora de identificar a las especies que el índice th(), tal y como se concluyó en el documento original.

En cuanto a las características de los llamados (tales como duración, call rate, etc.) no hay mucho que decir, se obtuvieron los datos de manera eficaz y nos fue posible identificar sus distintas características.

Resultados graficos

American Toad -> Verde
Spring Peeper -> Rojo
Grey Treefrog -> Azul

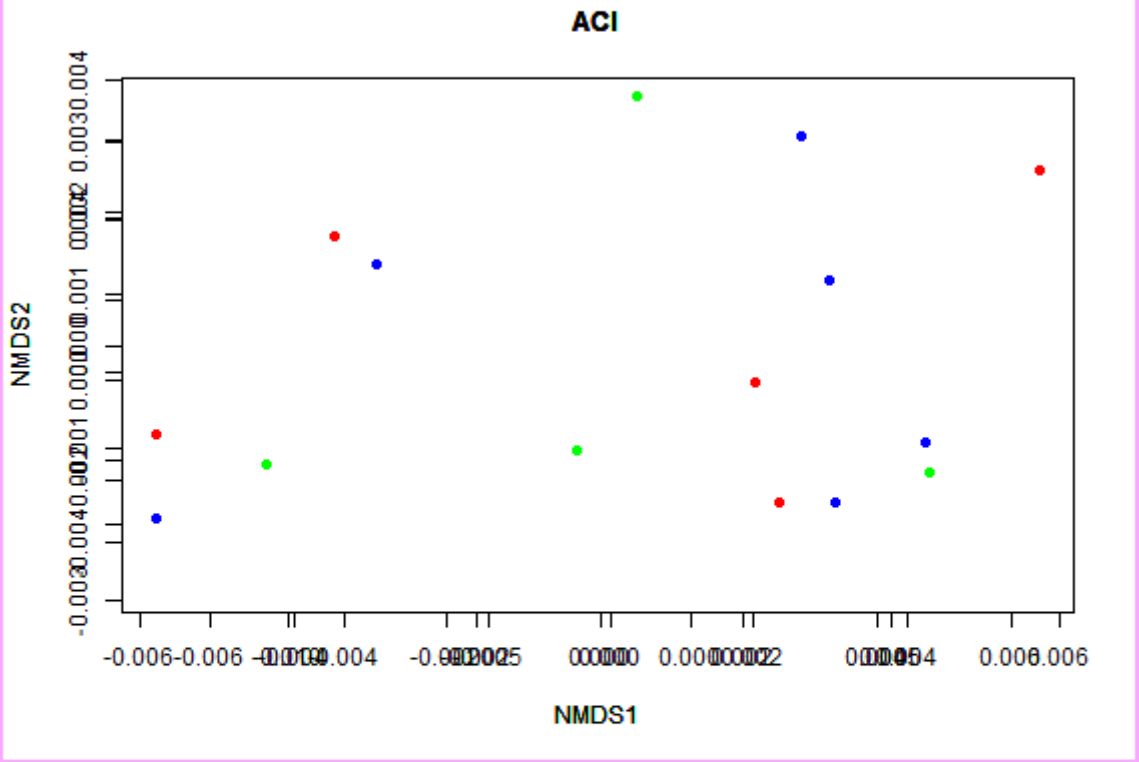


Duracion total de American Toad: 17.8 segundos
Duracion total de Spring Peeper: 28.28 segundos
Duracion total de Grey Treefrog: 26.25 segundos
Volumen de American Toad: 7.22 dB
Volumen de Spring Peeper: 4.63 dB
Volumen de Grey Treefrog: 8.88 dB
Call Rate de American Toad: 8.33 llamados/30 segundos
Call Rate de Spring Peeper: 30 llamados/30 segundos
Call Rate de Grey Treefrog: 2.67 llamados/30 segundos
Signar to Noise Ratio de American Toad: 15.9 dB
Signar to Noise Rate de Spring Peeper: 12.52 dB
Signar to Noise Rate de Grey Treefrog: 18.79 dB

Imagen 3.1 – *Resultado del análisis con la combinación de índices ACI-th.*

Resultados graficos

American Toad -> Verde
Spring Peeper -> Rojo
Grey Treefrog -> Azul

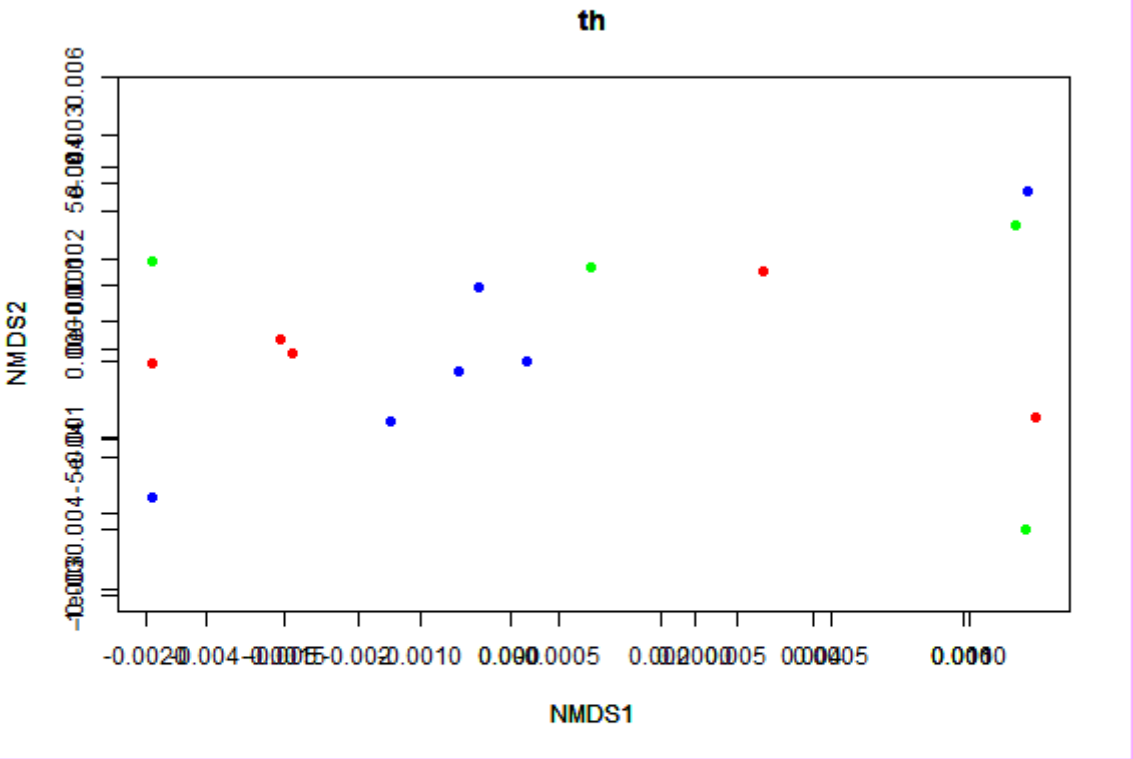


Duracion total de American Toad: 17.8 segundos
Duracion total de Spring Peeper: 28.28 segundos
Duracion total de Grey Treefrog: 26.25 segundos
Volumen de American Toad: 7.22 dB
Volumen de Spring Peeper: 4.63 dB
Volumen de Grey Treefrog: 8.88 dB
Call Rate de American Toad: 8.33 llamados/30 segundos
Call Rate de Spring Peeper: 30 llamados/30 segundos
Call Rate de Grey Treefrog: 2.67 llamados/30 segundos
Signar to Noise Ratio de American Toad: 15.9 dB
Signar to Noise Rate de Spring Peeper: 12.52 dB
Signar to Noise Rate de Grey Treefrog: 18.79 dB

Imagen 3.2 – *Resultado del análisis el índice ACI.*

Resultados graficos

American Toad -> Verde
Spring Peeper -> Rojo
Grey Treefrog -> Azul



Duracion total de American Toad: 17.8 segundos
Duracion total de Spring Peeper: 28.28 segundos
Duracion total de Grey Treefrog: 26.25 segundos
Volumen de American Toad: 7.22 dB
Volumen de Spring Peeper: 4.63 dB
Volumen de Grey Treefrog: 8.88 dB
Call Rate de American Toad: 8.33 llamados/30 segundos
Call Rate de Spring Peeper: 30 llamados/30 segundos
Call Rate de Grey Treefrog: 2.67 llamados/30 segundos
Signar to Noise Ratio de American Toad: 15.9 dB
Signar to Noise Rate de Spring Peeper: 12.52 dB
Signar to Noise Rate de Grey Treefrog: 18.79 dB

Imagen 3.3 – *Resultado del análisis con el índice th.*

5. Bibliografía

Jerome. (n.d.). *Documentación seewave*. Seewave - an r package for sound analysis and synthesis. Retrieved September 18, 2021, from https://rug.mnhn.fr/seewave/.

NMDS Ordination. RPubs. (n.d.). Retrieved November 21, 2021, from https://www.rpubs.com/RGrieger/545184.

Indraswari, K., Bower, D. S., Tucker, D., Schwarzkopf, L., Towsey, M., & Roe, P. (2018). Assessing the value of acoustic indices to distinguish species and quantify activity: A case study using frogs. *Freshwater Biology*, *65*(1), 142–152. https://doi.org/10.1111/fwb.13222

Shiny. (n.d.). Retrieved November 26, 2021, from https://shiny.rstudio.com/.

Comprehensive R Archive Network (CRAN). (n.d.). *Package audio*. CRAN. Retrieved November 26, 2021, from https://cran.r-project.org/web/packages/audio/.

Comprehensive R Archive Network (CRAN). (n.d.). *Package tuner*. CRAN. Retrieved November 26, 2021, from https://cran.r-project.org/web/packages/tuneR/.

Package 'vegan' - cran.r-project.org. (n.d.). Retrieved November 26, 2021, from https://cran.r-project.org/web/packages/vegan/vegan.pdf.