UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA ELÉTRICA
GRADUAÇÃO EM ENGENHARIA BIOMÉDICA

Heitor Pereira Nunes Fernandes Cunha

Processamento de Sinais Biomédicos: Módulo 6

Uberlândia, MG
2025

Questão 1

Considerando os dados disponíveis no arquivo MODULO5.xlsx, disponível na plataforma Moodle, responda às questões: Considerando os dados disponíveis no arquivo MODULO5.xlsx, disponível na plataforma Moodle, responda às questões:

Gerar um sinal sintético resultante da concatenação de quatro sinais básicos: (i) sinal sinoidal oscilando a 40 Hz com amplitude igual a 1; (ii) sinal senoidal oscilando a 40 Hz com amplitude igual a 1/2; (iii) sinal senoidal oscilando a 1 Hz com amplitude igual a 1; (iv) sinal do tipo dente de serra (triangular) oscilando a 10 Hz e de amplitude igual a 1. A duração de cada sinal deve ser de 5 segundos e a frequência de amostragem deve ser de 1000 Hz.

```
library(dygraphs)
```

```
## Warning: package 'dygraphs' was built under R version 4.3.3
```

```
library(tuneR)
```

```
## Warning: package 'tuneR' was built under R version 4.3.3
```

```
library(signal)
```

```
## Warning: package 'signal' was built under R version 4.3.3
```

```
##
## Attaching package: 'signal'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, poly
```

```
library(patchwork)
```

```
## Warning: package 'patchwork' was built under R version 4.3.3
```

```
library(tidyverse)
```

```
## Warning: package 'tidyverse' was built under R version 4.3.3
```

```
## Warning: package 'ggplot2' was built under R version 4.3.3
```

```
## Warning: package 'tibble' was built under R version 4.3.3
```

```
## Warning: package 'tidyr' was built under R version 4.3.3
```

```
## Warning: package 'readr' was built under R version 4.3.3
```

```
## Warning: package 'purrr' was built under R version 4.3.3
```

```
## Warning: package 'dplyr' was built under R version 4.3.3

## Warning: package 'stringr' was built under R version 4.3.3

## Warning: package 'forcats' was built under R version 4.3.3

## Warning: package 'lubridate' was built under R version 4.3.3

## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.0     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.1
## v purrr     1.0.2

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks signal::filter(), stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```r
# Definições básicas
taxa_amostragem <- 1000
intervalo_tempo <- 1 / taxa_amostragem
duracao <- 5
tempo <- seq(0, duracao, by = intervalo_tempo)

# Frequências dos sinais
freq_seno1 <- 40
freq_seno2 <- freq_seno1
freq_seno3 <- 1
freq_dente_serra <- 10

# Função para gerar sinais senoidais
gerar_senoide <- function(freq, tempo, amplitude = 1) {
  amplitude * sin(2 * pi * freq * tempo)
}

# Geração dos sinais
sinal_seno1 <- gerar_senoide(freq_seno1, tempo)
sinal_seno2 <- gerar_senoide(freq_seno2, tempo, amplitude = 0.5)
sinal_seno3 <- gerar_senoide(freq_seno3, tempo)
sinal_st <- sawtooth(freq_dente_serra,
                     samp.rate = taxa_amostragem,
                     duration = duracao + intervalo_tempo,
                     xunit = "time")@left
```

Cada um dos sinais gerados deve ser normalizado considerando o uso de um conversor analógico digital de 16 bits (nbits), tal que o valor de amplitude seja um número do tipo inteiro x E S (Equação 1).

```r
library(BBmisc)
```

```
## Warning: package 'BBmisc' was built under R version 4.3.3
```

```
##
## Attaching package: 'BBmisc'
```

```
## The following objects are masked from 'package:dplyr':
##
##     coalesce, collapse, symdiff
```

```
## The following object is masked from 'package:tuneR':
##
##     normalize
```

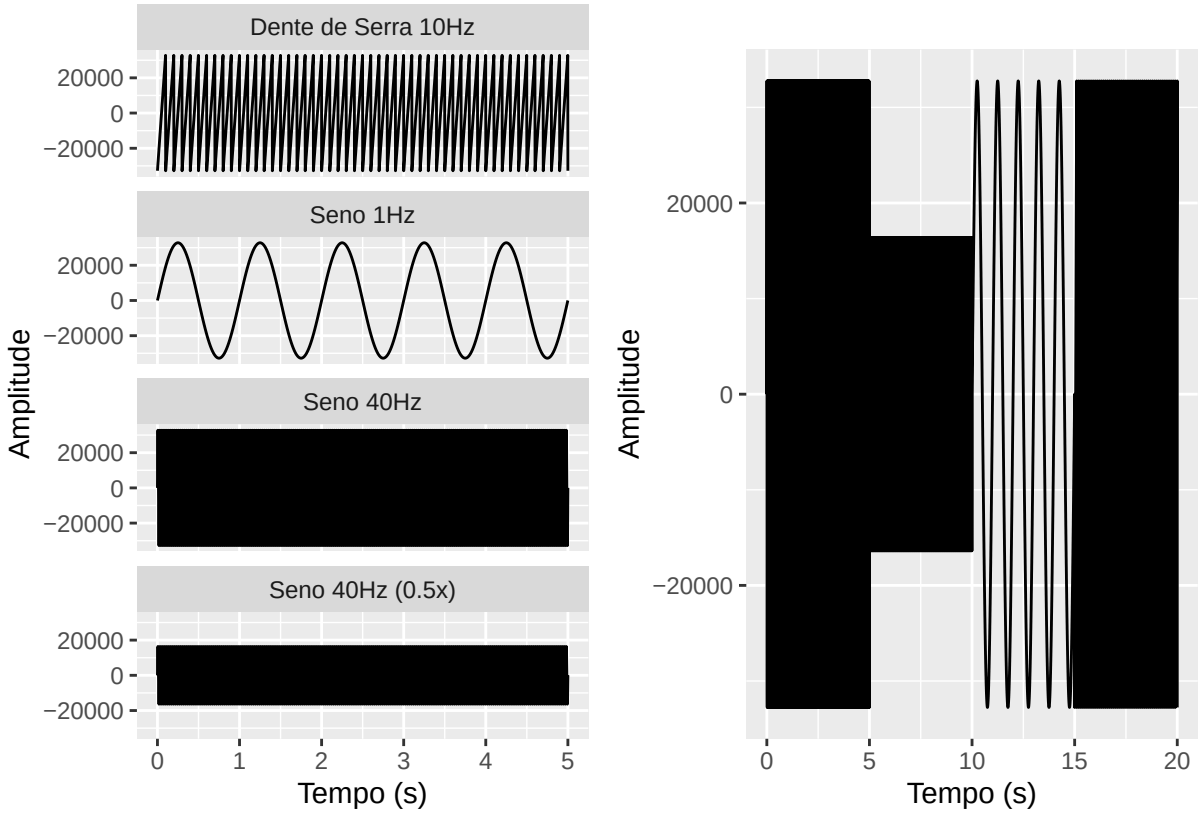
```
## The following object is masked from 'package:base':
##
##     isFALSE
```

```r
limite_inferior <- -(2^16) / 2
limite_superior <- ((2^16)/2) - 1


sinal_seno1_norm<- normalize(sinal_seno1, method = "range",range = c(limite_inferior,limite_superior))
sinal_seno2_norm<- sinal_seno1_norm/2
sinal_seno3_norm<- normalize(sinal_seno3, method = "range",range = c(limite_inferior,limite_superior))
sinal_st_norm<- normalize(sinal_st, method = "range",range = c(limite_inferior,limite_superior))
```

O sinais resultantes devem ser plotados utilizando-se o ggplot e o patchwork (veja o exemplo de resultado na Figura 1)

```r
# Criando dataframes para visualização
df_seno1 <- data.frame(tempo, amplitude = sinal_seno1_norm, tipo = "Seno 40Hz")
df_seno2 <- data.frame(tempo, amplitude = sinal_seno2_norm, tipo = "Seno 40Hz (0.5x)")
df_seno3 <- data.frame(tempo, amplitude = sinal_seno3_norm, tipo = "Seno 1Hz")
df_dente_serra <- data.frame(tempo, amplitude = sinal_st_norm, tipo = "Dente de Serra 10Hz")

df_sinais <- rbind(df_seno1, df_seno2, df_seno3, df_dente_serra)

# Plotagem dos sinais individuais
plot_sinais <- ggplot(df_sinais, aes(x = tempo, y = amplitude)) +
  geom_line() +
  facet_wrap(~ tipo, ncol = 1) +
  xlab("Tempo (s)") +
  ylab("Amplitude")

# Criação do sinal concatenado
sinal_concatenado <- c(sinal_seno1_norm, sinal_seno2_norm, sinal_seno3_norm, sinal_st_norm)
tempo_concatenado <- c(tempo, tempo + 5, tempo + 10, tempo + 15)
df_concatenado <- data.frame(tempo = tempo_concatenado, amplitude = sinal_concatenado)
```

```
# Plotagem do sinal concatenado
plot_concatenado <- ggplot(df_concatenado, aes(x = tempo, y = amplitude)) +
  geom_line() +
  xlab("Tempo (s)") +
  ylab("Amplitude")

# Exibição dos gráficos
plot_sinais | plot_concatenado
```



O sinal resultante deve ser salvo no formato wav. Utilize para isso o pacote tune R. Deve-se salvar o sinal apenas esquerdo.

```
# Conversão para formato WAV
sinal_wav <- Wave(left = as.integer(sinal_concatenado),
                  samp.rate = taxa_amostragem,
                  bit = 16,
                  pcm = TRUE)

# Salvando o arquivo de áudio
writeWave(sinal_wav, "sinal_processado_16bits.wav")
```

O sinal resultante deve ser aberto no software Audacity e tocado. Grave a tela do seu PC de forma a ilustrar o som gerado. Inclua um link do vídeo que você gerou de fora a comprovar a execução desta etapa.

Vídeo disponível em: https://drive.google.com/drive/folders/1gaCO4d0i3ongkWHvBHKrx2MUgXsEpVSz?usp=sharing

Por que não é possível ouvir o sinal senoidal oscilando a 1 Hz e é possível ouvir o sinal triangular oscilando a 10 Hz?

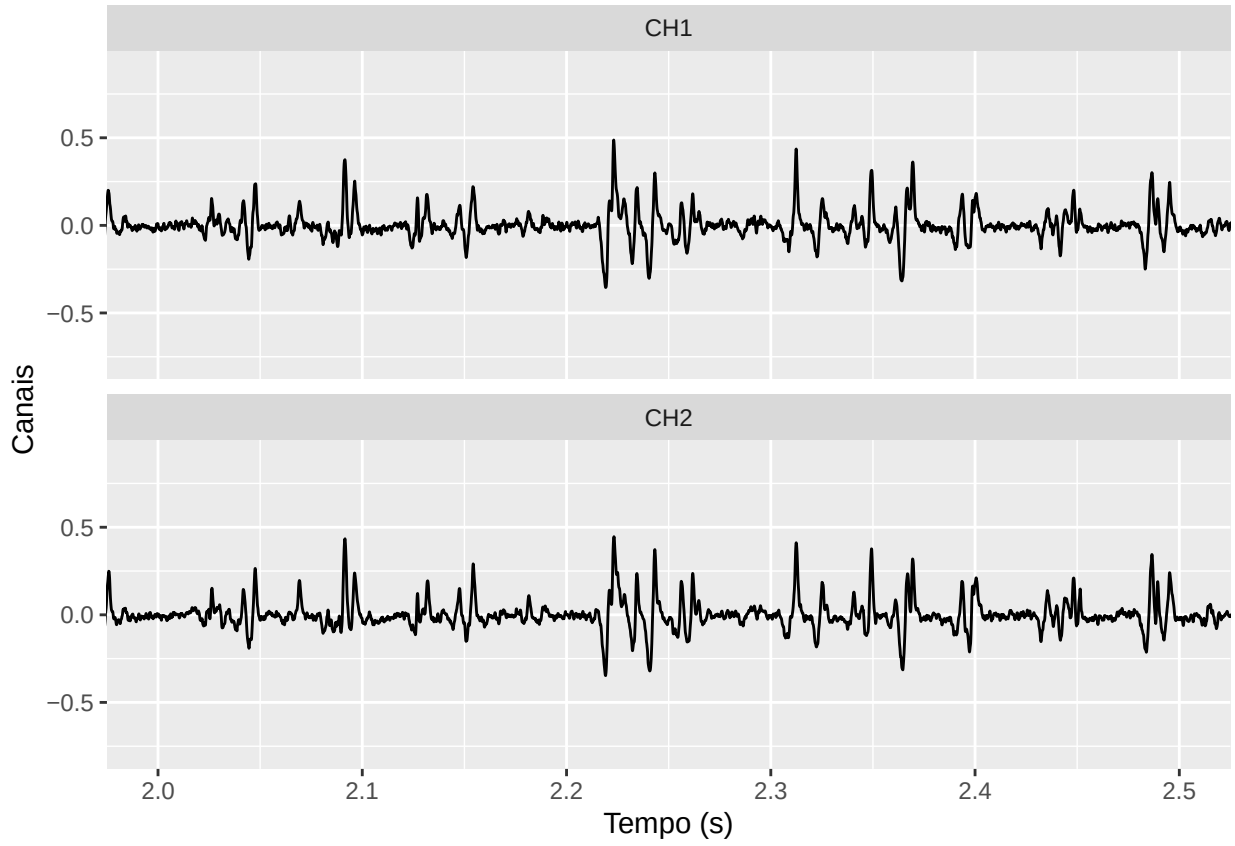
Isso ocorre porque a onda triangular é composta por múltiplas ondas senoidais que, juntas, resultam no seu formato característico. Assim, mesmo que sua frequência fundamental esteja abaixo do limiar de audição humana, algumas de suas componentes podem se situar dentro do espectro audível. Isso a diferencia de uma onda senoidal pura, que não apresenta essa combinação de frequências.

Questão 2

Escutar sinais biomédicos é parte do processamento de sinais. Esta etapa pode ser tão importante quanto visualizar os sinais. Nesta atividade você deverá:

Abrir o arquivo de sinais eletromiográficos (link) disponível na plataforma moodle e gerar um gráfico (usando o ggplot) dos dois canais de EMG com janela de visualização de tempo de 2 a 2.5 segundos.

```r
# Carregar as bibliotecas necessárias
library(ggplot2)
library(dplyr)

# Carregar os dados do arquivo EMG.txt
q2 <- read.table("E:/GitHub/PSB/Módulo 6 - PSB/EMG.txt", header = FALSE, sep = "\t")

# Renomear as colunas corretamente
colnames(q2) <- c("t", "CH1", "CH2")

# Criar dataframes para cada canal
CH1 <- data.frame(x = q2$t, y = q2$CH1, categoria = "CH1")
CH2 <- data.frame(x = q2$t, y = q2$CH2, categoria = "CH2") # Corrigido para CH2

# Unir os dataframes
canais <- rbind(CH1, CH2)

# Gerar o gráfico com janela de visualização entre 2 e 2.5 segundos
plot <- ggplot(canais, aes(x = x, y = y)) +
  geom_line() +
  facet_wrap(~ categoria, ncol = 1) +
  xlab("Tempo (s)") +
  ylab("Canais") +
  coord_cartesian(xlim = c(2, 2.5)) # Define os limites do eixo x

plot
```



Normalizar o sinal EMG dos dois canais considerando um conversor analógico digital de 16 bits (nbits), tal que o valor de amplitude seja um número do tipo inteiro x E S (Equação 1).

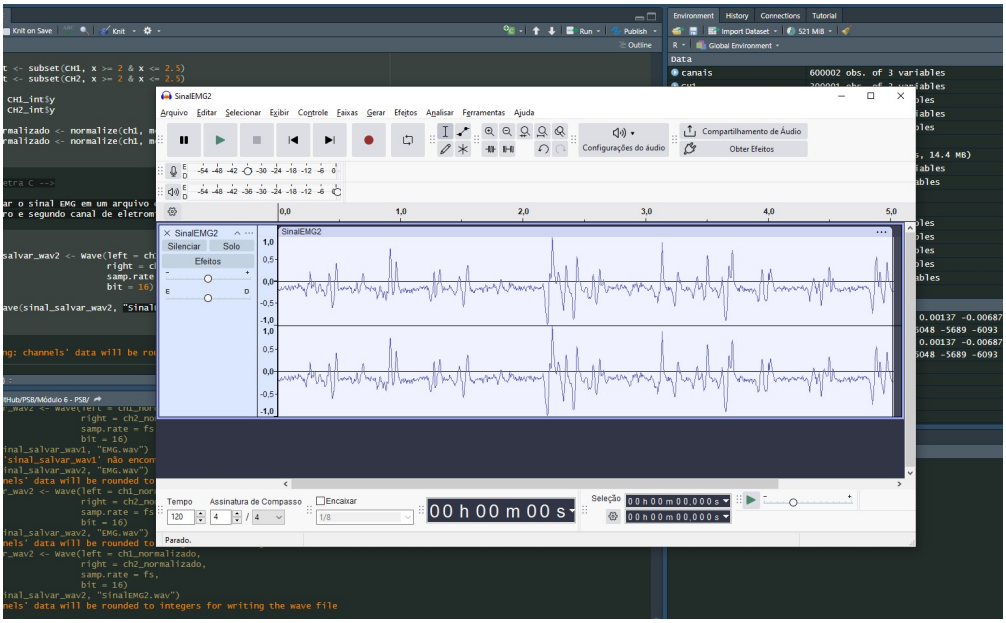
```
# Carregar a biblioteca necessária
library(BBmisc)

# Definir os limites de normalização para 16 bits
n0 <- -(2^16)/2
n1 <- ((2^16)/2) - 1

# Filtrar os dados entre 2 e 2.5 segundos
CH1_int <- subset(CH1, x >= 2 & x <= 2.5)
CH2_int <- subset(CH2, x >= 2 & x <= 2.5)

# Obter os valores de amplitude dos canais
ch1 <- CH1_int$y
ch2 <- CH2_int$y

# Normalizar os sinais de EMG dentro do intervalo de 16 bits
ch1_normalizado <- normalize(ch1, method = "range", range = c(n0, n1))
ch2_normalizado <- normalize(ch2, method = "range", range = c(n0, n1)) # Corrigido para ch2
```

Salvar o sinal EMG em um arquivo do tipo wav. Os canais esquerdo e direito do áudio devem corresponder, respectivamente, ao primeiro e segundo canal de eletromiografia.

```r
# Carregar a biblioteca necessária
library(tuneR)

fs <- 1000

# Salvar o sinal EMG em formato wav
sinal_salvar_wav2 <- Wave(left = ch1_normalizado,
                          right = ch2_normalizado,
                          samp.rate = fs,
                          bit = 16)

# Salvar o arquivo wav
writeWave(sinal_salvar_wav2, "SinalEMG2.wav")
```

```
## Warning in writeWave(sinal_salvar_wav2, "SinalEMG2.wav"): channels' data will
## be rounded to integers for writing the wave file
```

Abrir o arquivo no software Audacity para visualizar o sinal EMG no formato wav e salvar uma imagem da tela que mostre os sinais, tal como ilustrado na Figura 2. A imagem salva deve ser apresentada no relatório.

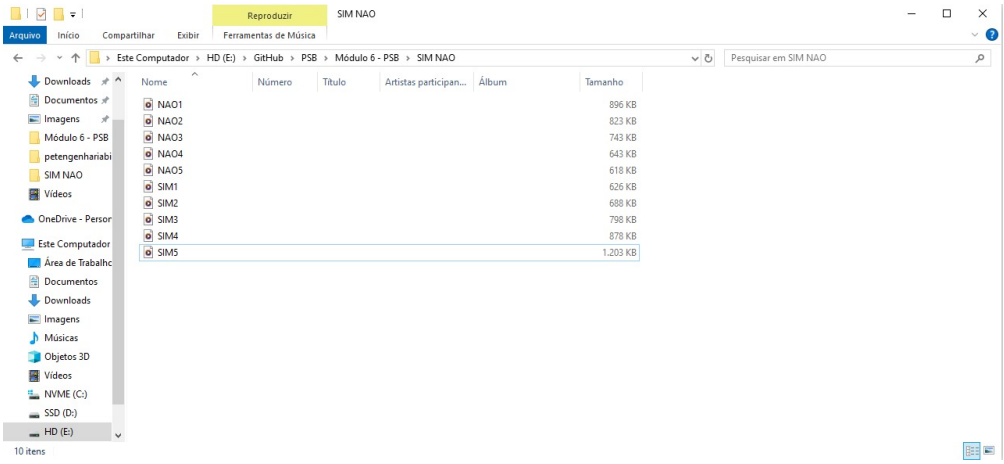


Escute os disparos dos potenciais de ação. Para isso execute os passos de 1 a 8 indicados na Figura 3. Você deve gravar a tela com o som e incluir o link do vídeo para avaliação

Vídeo disponível em: https://drive.google.com/drive/folders/1LnJtnUDFfZldMk9eshlQJuSz_x_THZTr?usp=sharing

Questão 3

Utilize o software Audacity para gravar cinco sinais de voz com o comando "sim" e outros cinco sinais com o comando "não". Faça a rotulação de cada segmento dos comandos de voz, salvando-os em arquivos distintos. No total, dez arquivos devem ser gerados.



Para cada sinal calcule as estatísticas: média, variância, coeficiente de assimetria e curtose. Organize os resultados em uma tabela (no R) e estime a variância de cada estatística.

O comando "sim" é estacionário? Justifique. O comando "sim" é ergódico? Justifique. O comando "não" é estacionário? Justifique. O comando "não" é ergódico? Justifique.

```
# Carregando o diretorio

dir <- "E:/GitHub/PSB/Módulo 6 - PSB/SIM NAO"

# SIM1
tfile_SIM1 <- file.path(dir, "SIM1.wav")
SIM1 <- readWave(tfile_SIM1)
fs_SIM1 = SIM1@samp.rate
dt_SIM1 = 1/fs_SIM1 #resolution time in seconds
yl_SIM1 <- SIM1@left
yr_SIM1 <- SIM1@right
tf_SIM1 <- (length(yl_SIM1)-1)*dt_SIM1 # tempo final em segundos
t_SIM1 <- seq(from=0, to=tf_SIM1, by=dt_SIM1)
df_SIM1 <- data.frame(time=t_SIM1, yl_SIM1, yr_SIM1)

# SIM2
tfile_SIM2 <- file.path(dir, "SIM2.wav")
SIM2 <- readWave(tfile_SIM2)
fs_SIM2 = SIM2@samp.rate # sampling frequency in Hz
dt_SIM2 = 1/fs_SIM2 #resolution time in seconds
yl_SIM2 <- SIM2@left
yr_SIM2 <- SIM2@right
```

```
tf_SIM2 <- (length(yl_SIM2)-1)*dt_SIM2 # tempo final em segundos
t_SIM2 <- seq(from=0, to=tf_SIM2, by=dt_SIM2)
df_SIM2 <- data.frame(time=t_SIM2, yl_SIM2, yr_SIM2)

# SIM3
tfile_SIM3 <- file.path(dir, "SIM3.wav")
SIM3 <- readWave(tfile_SIM3)
fs_SIM3 = SIM3@samp.rate # sampling frequency in Hz
dt_SIM3 = 1/fs_SIM3 #resolution time in seconds
yl_SIM3 <- SIM3@left
yr_SIM3 <- SIM3@right
tf_SIM3 <- (length(yl_SIM3)-1)*dt_SIM3 # tempo final em segundos
t_SIM3 <- seq(from=0, to=tf_SIM3, by=dt_SIM3)
df_SIM3 <- data.frame(time=t_SIM3, yl_SIM3, yr_SIM3)

# SIM4
tfile_SIM4 <- file.path(dir, "SIM4.wav")
SIM4 <- readWave(tfile_SIM4)
fs_SIM4 = SIM4@samp.rate # sampling frequency in Hz
dt_SIM4 = 1/fs_SIM4 #resolution time in seconds
yl_SIM4 <- SIM4@left
yr_SIM4 <- SIM4@right
tf_SIM4 <- (length(yl_SIM4)-1)*dt_SIM4 # tempo final em segundos
t_SIM4 <- seq(from=0, to=tf_SIM4, by=dt_SIM4)
df_SIM4 <- data.frame(time=t_SIM4, yl_SIM4, yr_SIM4)

# SIM5
tfile_SIM5 <- file.path(dir, "SIM5.wav")
SIM5 <- readWave(tfile_SIM5)
fs_SIM5 = SIM5@samp.rate # sampling frequency in Hz
dt_SIM5 = 1/fs_SIM5 #resolution time in seconds
yl_SIM5 <- SIM5@left
yr_SIM5 <- SIM5@right
tf_SIM5 <- (length(yl_SIM5)-1)*dt_SIM5 # tempo final em segundos
t_SIM5 <- seq(from=0, to=tf_SIM5, by=dt_SIM5)
df_SIM5 <- data.frame(time=t_SIM5, yl_SIM5, yr_SIM5)

# NAO1
tfile_NAO1 <- file.path(dir, "NAO1.wav")
NAO1 <- readWave(tfile_NAO1)
fs_NAO1 = NAO1@samp.rate # sampling frequency in Hz
dt_NAO1 = 1/fs_NAO1 #resolution time in seconds
yl_NAO1 <- NAO1@left
yr_NAO1 <- NAO1@right
tf_NAO1 <- (length(yl_NAO1)-1)*dt_NAO1 # tempo final em segundos
t_NAO1 <- seq(from=0, to=tf_NAO1, by=dt_NAO1)
df_NAO1 <- data.frame(time=t_NAO1, yl_NAO1, yr_NAO1)

# NAO2
tfile_NAO2 <- file.path(dir, "NAO2.wav")
NAO2 <- readWave(tfile_NAO2)
fs_NAO2 = NAO2@samp.rate # sampling frequency in Hz
dt_NAO2 = 1/fs_NAO2 #resolution time in seconds
```

```
yl_NAO2 <- NAO2@left
yr_NAO2 <- NAO2@right
tf_NAO2 <- (length(yl_NAO2)-1)*dt_NAO2 # tempo final em segundos
t_NAO2 <- seq(from=0, to=tf_NAO2, by=dt_NAO2)
df_NAO2 <- data.frame(time=t_NAO2, yl_NAO2, yr_NAO2)

# NAO3
tfile_NAO3 <- file.path(dir, "NAO3.wav")
NAO3 <- readWave(tfile_NAO3)
fs_NAO3 = NAO3@samp.rate # sampling frequency in Hz
dt_NAO3 = 1/fs_NAO3 #resolution time in seconds
yl_NAO3 <- NAO3@left
yr_NAO3 <- NAO3@right
tf_NAO3 <- (length(yl_NAO3)-1)*dt_NAO3 # tempo final em segundos
t_NAO3 <- seq(from=0, to=tf_NAO3, by=dt_NAO3)
df_NAO3 <- data.frame(time=t_NAO3, yl_NAO3, yr_NAO3)

# NAO4
tfile_NAO4 <- file.path(dir, "NAO4.wav")
NAO4 <- readWave(tfile_NAO4)
fs_NAO4 = NAO4@samp.rate # sampling frequency in Hz
dt_NAO4 = 1/fs_NAO4 #resolution time in seconds
yl_NAO4 <- NAO4@left
yr_NAO4 <- NAO4@right
tf_NAO4 <- (length(yl_NAO4)-1)*dt_NAO4 # tempo final em segundos
t_NAO4 <- seq(from=0, to=tf_NAO4, by=dt_NAO4)
df_NAO4 <- data.frame(time=t_NAO4, yl_NAO4, yr_NAO4)

# NAO5
tfile_NAO5 <- file.path(dir, "NAO5.wav")
NAO5 <- readWave(tfile_NAO5)
fs_NAO5 = NAO5@samp.rate # sampling frequency in Hz
dt_NAO5 = 1/fs_NAO5 #resolution time in seconds
yl_NAO5 <- NAO5@left
yr_NAO5 <- NAO5@right
tf_NAO5 <- (length(yl_NAO5)-1)*dt_NAO5 # tempo final em segundos
t_NAO5 <- seq(from=0, to=tf_NAO5, by=dt_NAO5)
df_NAO5 <- data.frame(time=t_NAO5, yl_NAO5, yr_NAO5)

# Calculando as carecteristicas

# Calculando as médias com a função mean()
media_SIM1 <- mean(df_SIM1$yl_SIM1)
media_SIM2 <- mean(df_SIM2$yl_SIM2)
media_SIM3 <- mean(df_SIM3$yl_SIM3)
media_SIM4 <- mean(df_SIM4$yl_SIM4)
media_SIM5 <- mean(df_SIM5$yl_SIM5)
media_NAO1<- mean(df_NAO1$yl_NAO1)
media_NAO2<- mean(df_NAO2$yl_NAO2)
media_NAO3<- mean(df_NAO3$yl_NAO3)
media_NAO4<- mean(df_NAO4$yl_NAO4)
media_NAO5<- mean(df_NAO5$yl_NAO5)
```

```r
# Calculando a variancia com a função var()
var_SIM1 <- var(df_SIM1$yl_SIM1)
var_SIM2 <- var(df_SIM2$yl_SIM2)
var_SIM3 <- var(df_SIM3$yl_SIM3)
var_SIM4 <- var(df_SIM4$yl_SIM4)
var_SIM5 <- var(df_SIM5$yl_SIM5)
var_NAO1<- var(df_NAO1$yl_NAO1)
var_NAO2<- var(df_NAO2$yl_NAO2)
var_NAO3<- var(df_NAO3$yl_NAO3)
var_NAO4<- var(df_NAO4$yl_NAO4)
var_NAO5<- var(df_NAO5$yl_NAO5)

# Calculando o coeficiente de assimetria através da função skewness()
library(e1071)

coef_assim_SIM1 <- skewness(df_SIM1$yl_SIM1)
coef_assim_SIM2 <- skewness(df_SIM2$yl_SIM2)
coef_assim_SIM3 <- skewness(df_SIM3$yl_SIM3)
coef_assim_SIM4 <- skewness(df_SIM4$yl_SIM4)
coef_assim_SIM5 <- skewness(df_SIM5$yl_SIM5)
coef_assim_NAO1 <- skewness(df_NAO1$yl_NAO1)
coef_assim_NAO2 <- skewness(df_NAO2$yl_NAO2)
coef_assim_NAO3 <- skewness(df_NAO3$yl_NAO3)
coef_assim_NAO4 <- skewness(df_NAO4$yl_NAO4)

coef_assim_NAO5 <- skewness(df_NAO5$yl_NAO5)

# Calculando a curtose com a função kurtosis()
curt_SIM1 <- kurtosis(df_SIM1$yl_SIM1)
curt_SIM2 <- kurtosis(df_SIM2$yl_SIM2)
curt_SIM3 <- kurtosis(df_SIM3$yl_SIM3)
curt_SIM4 <- kurtosis(df_SIM4$yl_SIM4)
curt_SIM5 <- kurtosis(df_SIM5$yl_SIM5)
curt_NAO1 <- kurtosis(df_NAO1$yl_NAO1)
curt_NAO2 <- kurtosis(df_NAO2$yl_NAO2)
curt_NAO3 <- kurtosis(df_NAO3$yl_NAO3)
curt_NAO4 <- kurtosis(df_NAO4$yl_NAO4)
curt_NAO5 <- kurtosis(df_NAO5$yl_NAO5)

# variancia de cada estatistica
var_media <- var(c(media_SIM1, media_SIM2, media_SIM3, media_SIM4, media_SIM5, media_NAO1, media_NAO2, m

var_var <- var(c(var_SIM1 , var_SIM2 , var_SIM3, var_SIM4, var_SIM5, var_NAO1, var_NAO2, var_NAO3, var_

var_ca <- var(c(coef_assim_SIM1,coef_assim_SIM2, coef_assim_SIM3, coef_assim_SIM4, coef_assim_SIM5, coe

var_kurt <- var(c(curt_SIM1, curt_SIM2, curt_SIM3, curt_SIM4, curt_SIM5, curt_NAO1, curt_NAO2, curt_NAO

# colocando tudo em um mesmo data frame
dd <- data.frame( sinais = c("SIM1", "SIM2", "SIM3", "SIM4","SIM5","NAO1","NAO2","NAO3","NAO4","NAO5",
                  mean = c(media_SIM1, media_SIM2, media_SIM3, media_SIM4, media_SIM5, media_NAO1, media
                  var = c(var_SIM1 , var_SIM2 , var_SIM3, var_SIM4, var_SIM5, var_NAO1, var_NAO2, var_NA
                  ca = c(coef_assim_SIM1,coef_assim_SIM2, coef_assim_SIM3, coef_assim_SIM4, coef_assim_S
```

sinais	mean	var	ca	curtose
SIM1	-9.387816	8.689946e+05	0.6539489	8.167734
SIM2	7.478677	1.019778e+06	0.6105369	8.350573
SIM3	-6.885380	8.722521e+05	0.4279417	12.664761
SIM4	-11.711667	3.098798e+05	0.5333970	20.202952
SIM5	-8.027155	5.053436e+05	0.9567269	10.793850
NAO1	-3.840517	5.028324e+05	0.9074686	7.156847
NAO2	-5.337675	6.370142e+05	1.1149777	7.387476
NAO3	-13.910445	6.936453e+05	1.4452374	8.664364
NAO4	-21.568876	8.686988e+05	1.1721594	6.529297
NAO5	-16.357030	8.556247e+05	1.2422134	7.016167
variâncias	62.299534	4.937076e+10	0.1154559	17.178992

```
                curtose = c(curt_SIM1, curt_SIM2, curt_SIM3, curt_SIM4, curt_SIM5, curt_NAO1, curt_NA
```

```
# organizando em uma tabela utilizando o pacote kableExtra
library(dplyr)
library(kableExtra)
```

```
## Warning: package 'kableExtra' was built under R version 4.3.3
```

```
##
## Attaching package: 'kableExtra'
```

```
## The following object is masked from 'package:dplyr':
##
##     group_rows
```

```
# mostra tabela final
dd %>% kbl() %>% kable_styling()
```

Nota-se que tanto o comando SIM quanto o comando NÃO não apresentam estacionariedade, uma vez que métricas como média, variância, coeficiente de variação e curtose variam entre as amostras. Isso ocorre porque seria necessário que o usuário mantivesse completamente constantes a entonação, o volume, a expressão e outros aspectos da fala. Quanto à ergodicidade, pode-se afirmar que os comandos SIM e NÃO não são ergódicos, pois, conforme demonstrado na tabela, essas métricas variam entre valores negativos.

> Utilize o valor máximo da correlação cruzada para realizar a comparação entre comandos de voz. Anote estes valores em uma tabela, conforme padrão abaixo. Utilize um lag de 10.000 amostras no cálculo.
>
> Baseado nos resultados apresentados na tabela, como você poderia desenvolver um sistema de reconhecimento automático de voz, que realiza a distinção entre os comandos "sim" e "não"?

```
calcular_ccf_max <- function(sinais) {
  resultados <- matrix(0, nrow = length(sinais), ncol = length(sinais))
  nomes <- names(sinais)

  for (i in 1:length(sinais)) {
    for (j in i:length(sinais)) {
```

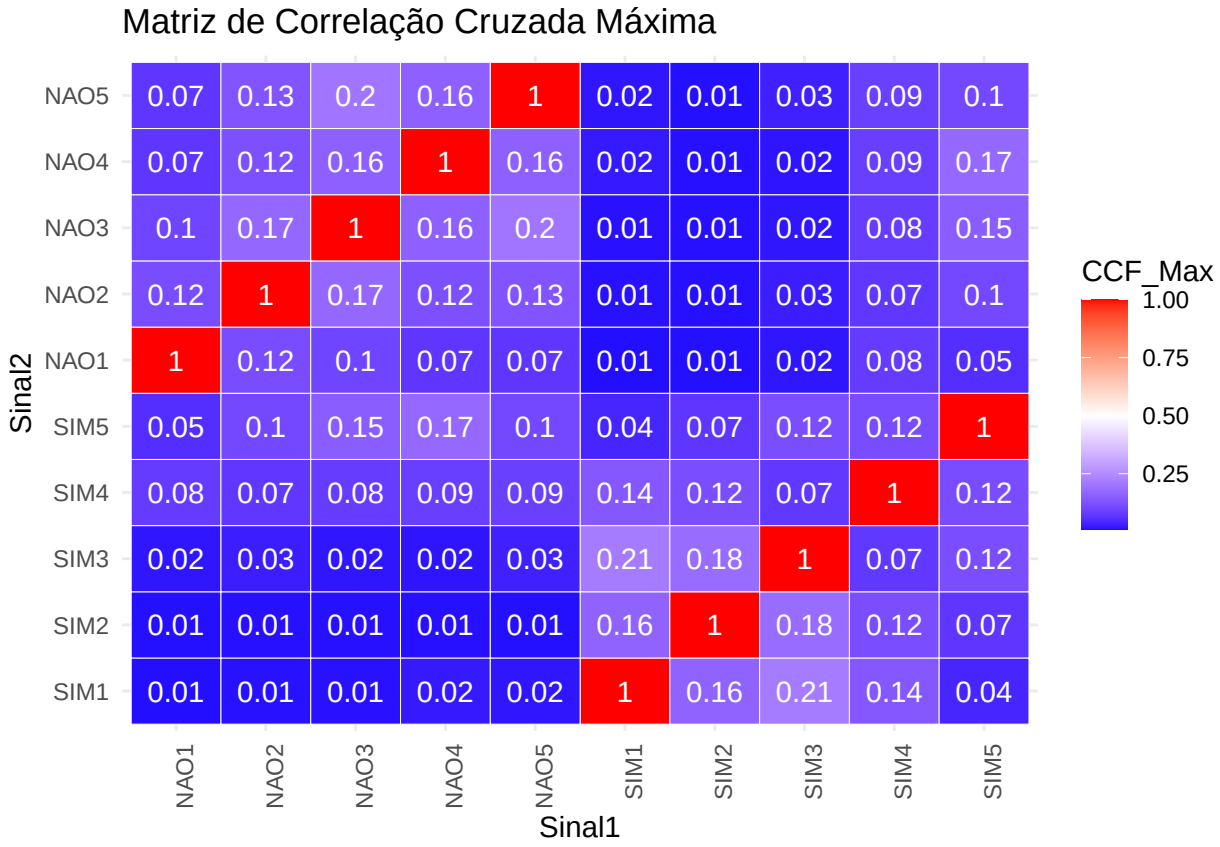
```
      ccf_res <- ccf(sinais[[i]], sinais[[j]], lag.max = 10000, plot = FALSE)
      resultados[i, j] <- max(ccf_res$acf)
      resultados[j, i] <- resultados[i, j]  # Matriz simétrica
    }
  }

  colnames(resultados) <- nomes
  rownames(resultados) <- nomes
  return(resultados)
}

# Lista de sinais
sinais <- list(
  SIM1 = yl_SIM1, SIM2 = yl_SIM2, SIM3 = yl_SIM3, SIM4 = yl_SIM4, SIM5 = yl_SIM5,
  NAO1 = yl_NAO1, NAO2 = yl_NAO2, NAO3 = yl_NAO3, NAO4 = yl_NAO4, NAO5 = yl_NAO5
)

# Calcula a matriz de correlação cruzada máxima
resultado_ccf <- calcular_ccf_max(sinais)

library(ggplot2)
library(reshape2)
```

```
## Warning: package 'reshape2' was built under R version 4.3.3
```

```
##
## Attaching package: 'reshape2'
```

```
## The following object is masked from 'package:tidyr':
##
##     smiths
```

```
# Converte a matriz de correlação cruzada em um dataframe
df_ccf <- as.data.frame(resultado_ccf)
df_ccf$Sinal1 <- rownames(df_ccf)

# Transforma a matriz em formato longo
df_long <- melt(df_ccf, id.vars = "Sinal1", variable.name = "Sinal2", value.name = "CCF_Max")

# Cria o gráfico de heatmap
ggplot(df_long, aes(x = Sinal1, y = Sinal2, fill = CCF_Max)) +
  geom_tile(color = "white") +  # Adiciona os quadrados
  geom_text(aes(label = round(CCF_Max, 2)), color = "white", size = 4) +  # Coloca os valores dentro do
  scale_fill_gradient2(low = "blue", high = "red", mid = "white", midpoint = 0.5) +  # Definindo a pale
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1),  # Gira os rótulos no eixo X
        axis.text.y = element_text(hjust = 1)) +  # Ajusta os rótulos no eixo Y
  labs(title = "Matriz de Correlação Cruzada Máxima",
       x = "Sinal1", y = "Sinal2")
```


Matriz de Correlação Cruzada Máxima
NAO5 0.07 0.13 0.2 0.16 1 0.02 0.01 0.03 0.09 0.1
NAO4 0.07 0.12 0.16 1 0.16 0.02 0.01 0.02 0.09 0.17
NAO3 0.1 0.17 1 0.16 0.2 0.01 0.01 0.02 0.08 0.15
NAO2 0.12 1 0.17 0.12 0.13 0.01 0.01 0.03 0.07 0.1
NAO1 1 0.12 0.1 0.07 0.07 0.01 0.01 0.02 0.08 0.05
SIM5 0.05 0.1 0.15 0.17 0.1 0.04 0.07 0.12 0.12 1
SIM4 0.08 0.07 0.08 0.09 0.09 0.14 0.12 0.07 1 0.12
SIM3 0.02 0.03 0.02 0.02 0.03 0.21 0.18 1 0.07 0.12
SIM2 0.01 0.01 0.01 0.01 0.01 0.16 1 0.18 0.12 0.07
SIM1 0.01 0.01 0.01 0.02 0.02 1 0.16 0.21 0.14 0.04
Sinal2
NAO1 NAO2 NAO3 NAO4 NAO5 SIM1 SIM2 SIM3 SIM4 SIM5
Sinal1
CCF_Max
1.00
0.75
0.50
0.25

Questão 4

Considerando a avaliação de três grupos neurônios, com as características abaixo:

E1: média de disparo = 53 ms; desvio padrão = 50 ms E2: média de disparo = 100 ms; desvio padrão = 35 ms E3: média de disparo = 53 ms; desvio padrão = 15 ms

Dica: o intervalo entre disparos é um processo pontual com distribuição gaussina (Utilize a função rnorm do R para gerar os tempos de disparo)

Plote o diagrama de disparos dos três grupos de neurônios, durante 10 segundos, de acordo com as regras dadas na tabela, cujo valores numéricos indicam os tempos de início e término (em segundos) dos disparos dos grupos E1, E2 e E3. Exemplo: O grupo E1 dispara entre 1 e 2 s, entre 5 e 7 s, e permanence em repouso durante todo o intervalo de simulação (de 0 a 10 segundos). A simulação deverá ser feita com uma taxa de amostragem de 10 kHz.

```r
# Função para gerar disparos com restrições temporais
gerar_disparos <- function(media_ms, dp_ms, intervalos_ativos, duracao_total = 10) {
  tempo_atual <- 0
  registros <- numeric(0)

  # Converte parâmetros para segundos
  media <- media_ms / 1000
  desvio <- dp_ms / 1000

  while(tempo_atual < duracao_total) {
    intervalo <- abs(rnorm(1, media, desvio))
    tempo_atual <- tempo_atual + intervalo

    # Verifica intervalos de atividade
    for(periodo in intervalos_ativos) {
      if(tempo_atual >= periodo[1] && tempo_atual <= periodo[2]) {
        registros <- c(registros, tempo_atual)
        break
      }
    }
  }
  return(round(registros, 4))
}

# Parâmetros de disparo (intervalos em segundos)
disparos_E1 <- gerar_disparos(53, 50, list(c(1,2), c(5,7)))
disparos_E2 <- gerar_disparos(100, 35, list(c(2,4), c(7,9)))
disparos_E3 <- gerar_disparos(53, 15, list(c(1,3), c(9,10)))

# Cria grade temporal de 10 kHz
grade_temporal <- seq(0, 10, 1/10000)

# Função para criar matriz binária de eventos
criar_matriz_eventos <- function(disparos) {
  sinais <- numeric(length(grade_temporal))
  indices <- match(round(disparos, 4), round(grade_temporal, 4))
  sinais[indices[!is.na(indices)]] <- 1
```

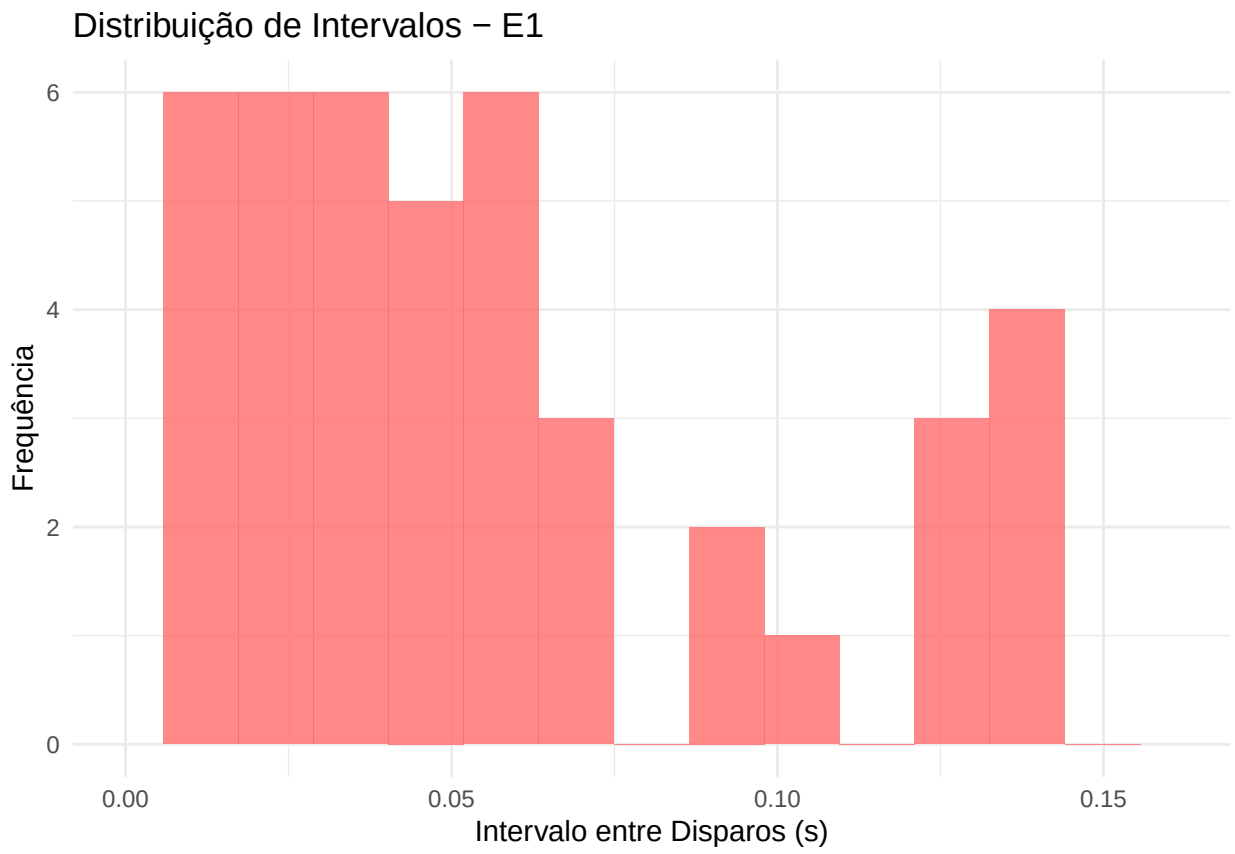
```r
  return(data.frame(tempo = grade_temporal, sinal = sinais))
}

# Gera os dados plotáveis
dados_plot_E1 <- criar_matriz_eventos(disparos_E1)
dados_plot_E2 <- criar_matriz_eventos(disparos_E2)
dados_plot_E3 <- criar_matriz_eventos(disparos_E3)
```

Gere os histogramas dos intervalos entre disparos para os grupos de neurônios E1, E2 e E3. Devem ser gerados três histogramas (um para cada grupo).

```r
# Função para calcular intervalos entre disparos
calcular_intervalos <- function(disparos) {
  if(length(disparos) < 2) return(numeric(0))
  diff(disparos)
}

# Gera dados para histogramas
intervalos_E1 <- calcular_intervalos(disparos_E1)
intervalos_E2 <- calcular_intervalos(disparos_E2)
intervalos_E3 <- calcular_intervalos(disparos_E3)

# Função unificada para plotagem
plotar_histograma <- function(dados, titulo, cor = "steelblue") {
  ggplot(data.frame(Intervalo = dados), aes(x = Intervalo)) +
    geom_histogram(fill = cor, bins = 15, alpha = 0.8) +
    labs(title = titulo, x = "Intervalo entre Disparos (s)", y = "Frequência") +
    theme_minimal() +
    xlim(0, quantile(dados, 0.95)) # Remove outliers extremos
}

# Cria e exibe os gráficos
library(ggplot2)
print(plotar_histograma(intervalos_E1, "Distribuição de Intervalos - E1", "#FF6B6B"))
```

```
## Warning: Removed 3 rows containing non-finite outside the scale range
## (‘stat_bin()‘).
```

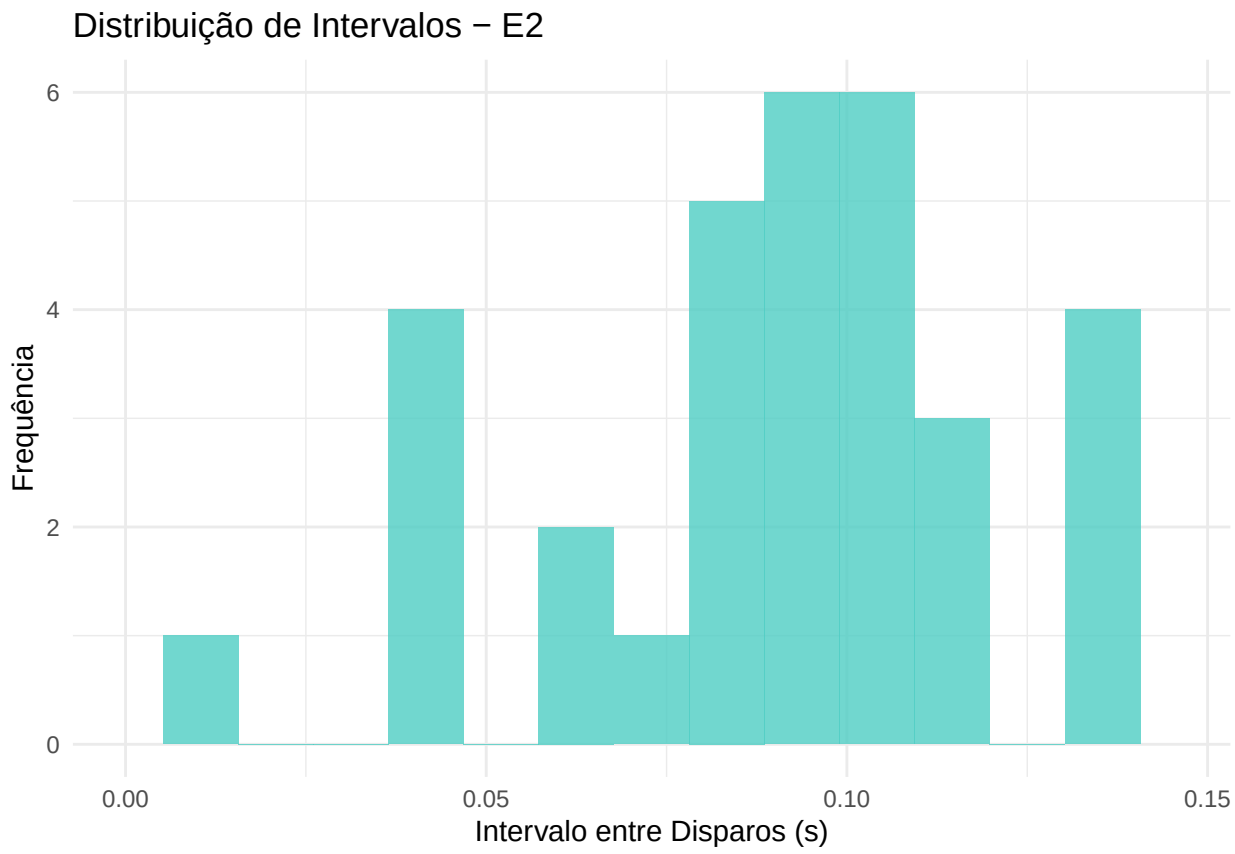
```
## Warning: Removed 2 rows containing missing values or values outside the scale range
## (‘geom_bar()‘).
```




```
print(plotar_histograma(intervalos_E2, "Distribuição de Intervalos - E2", "#4ECDC4"))
```

```
## Warning: Removed 2 rows containing non-finite outside the scale range
## (`stat_bin()`).
```

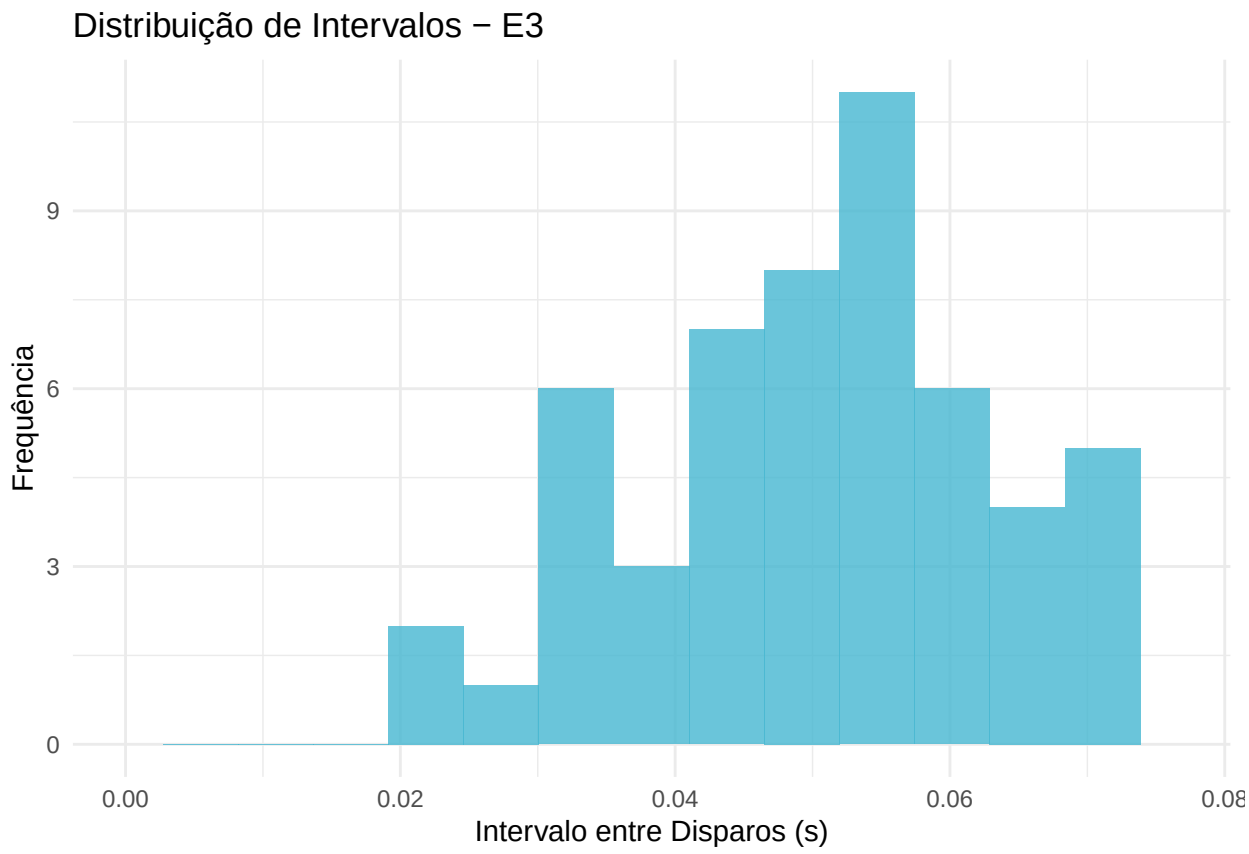
```
## Warning: Removed 2 rows containing missing values or values outside the scale range
## (`geom_bar()`).
```




```
print(plotar_histograma(intervalos_E3, "Distribuição de Intervalos - E3", "#45B7D1"))
```

```
## Warning: Removed 3 rows containing non-finite outside the scale range
## (`stat_bin()`).
```

```
## Warning: Removed 2 rows containing missing values or values outside the scale range
## (`geom_bar()`).
```


Distribuição de Intervalos − E3
Frequência
0
3
6
9
0.00
0.02
0.04
0.06
0.08
Intervalo entre Disparos (s)