Amanda Martinelli de Oliveira

Mestranda em Oceanografia Biológica pelo Instituto Oceanográfico (IO-USP).

Desenvolve projeto sobre **paisagens acústicas submarinas** no litoral norte de São Paulo, pelo laboratório de biologia e conservação de mamíferos aquáticos - LABCMA http://sotalia.com.br/.

exec

Trabalho Final

Proposta A

Investigações sobre a paisagem acústica de um local são possíveis de serem feitas através de gravações sonoras de diversas quantidades de tempo. Dependendo das questões logísticas do equipamento de gravação (bateria, espaço de armazenamento) as gravações podem ser contínuas, e então geralmente por um período mais curto, ou podem ser programadas em intervalos pré-estabelecidos (gravações de 1 minuto a cada 10 minutos, por exemplo), possibilitando o registro de um período maior de tempo. No último caso, com o fim da autonomia do equipamento, o produto são vários registros de gravação de período curto, que representam um período maior (para o exemplo anterior = 144 arquivos por dia). Esses arquivos curtos se iniciam com uma sequência de notas como sinalização do início da gravação pelo equipamento. Como as investigações consideram medidas de frequência do espectro total, essas sequências de sinalização devem ser deletadas de cada arquivo, para que não enviesem os resultados.

Dessa forma, a função deverá:

1. Encontrar os arquivos de som em determinada pasta (Argumento= Pasta)
2. Cortar todos esses arquivos (Argumento = Tempo de início e fim do corte)
3. Unir todos os arquivos então cortados em um único arquivo de som, que será a representação do período maior.

Outros possíveis argumentos seriam os referentes as configurações das gravações, como por exemplo a taxa de amostragem.

A função deverá retornar um arquivo de som composto por todos os trechos de gravações menores presentes na pasta indicada, sem a presença das sequências sinalizadoras de início (Arquivo com a soma dos 144 arquivos da pasta).

Proposta B

É comum que as informações coletadas em uma pesquisa resulte em uma grande tabela de dados. Para investigar esses dados, e muitas vezes também para sua apresentação, geramos gráficos através dessa tabela. Porém, geralmente geramos um gráfico por vez, o que demanda muito trabalho e tempo. Assim, com o intuito de minimizar os esforços, essa função deverá:

À partir de um dataframe (Argumento = dataframe), gerar n gráficos (Argumento = tipo de gráfico) referentes às colunas pré estabelecidas (Argumento = colunas) em função das linhas desse dataframe. Outros possíveis argumentos seriam os referentes as configurações dos gráficos.

A função deverá então retornar a quantidade de gráficos demandada, das variáveis das colunas pela variável da linha.

Comentário Melina

Olá Amanda, me pareceu que a proposta A é mais desafiadora e interessante que a B. Na verdade, achei a proposta B bem bobinha e só deveria ser feito caso você não consiga de jeito nenhum fazer algo parecido com a proposta A. Então, eu proponho que você faça a A!

Na proposta A, eu não tenho conhecimento das ferramentas (pacotes e funções) existentes para edição de áudios, mas fazendo uma busca rápida vi que há pacotes que podem te ajudar. Então, pode ser que seja fácil fazer essa função usando funções de pacotes existentes (não esqueça de deixar isso explícito no help da função). Nesse caso, sugiro que você complemente sua função com outras funcionalidades para arquivos de áudio.

Uma sugestão: Se os arquivos de áudio a serem trabalhados forem muito grandes, acho que você pode delimitar um tamanho máximo do arquivo de áudio final (junção dos pequenos arquivos cortados), pois senão é capaz que o R não suporte (seria bom você buscar estas informações de capacidade máxima de memória que o R usa). Assim, sua função, antes de juntar as partes poderia coletar a informação do tamanho de cada arquivo e calcular quanto de memória seria necessário para juntar todos os arquivos e, se o tamanho calculado ultrapassar o limite posto poderia sair uma mensagem de erro ou aviso (warning) informando o tamanho calculado e excedente. A função também poderia retornar, além do tamanho do arquivo, o tempo total do arquivo gerado.

FUNÇÃO

Código da função addsound()

```
addsound = function(pasta, t0=NULL, t1=NULL, sf=NULL, nome.arq=NULL )
```

```
{
    # Verificando se todos os argumentos estão presentes#
    if(missing(pasta))  # Caso o argumento pasta estiver ausente, uma
mensagem de erro sera exibida avisando o usuário
    {
    stop("Argumento pasta faltante")
    }

    if(missing(t0))  # Caso o argumento t0 estiver ausente, uma mensagem de
erro sera exibida avisando o usuário
    {
    stop("Argumento tempo inicial do corte faltante")
    }

    if(missing(t1))  # Caso o argumento t1 estiver ausente, uma mensagem de
erro sera exibida avisando o usuário
    {
    stop("Argumento tempo final do corte faltante")
    }

    if(missing(sf))  # Caso o argumento sf estiver ausente, uma mensagem de
erro sera exibida avisando o usuário
    {
    stop("Argumento taxa de amostragem faltante")
    }

    if(missing(nome.arq))  # Caso o argumento nome.arq estiver ausente, uma
mensagem de erro sera exibida avisando o usuário
    {
    stop("Argumento nome do arquivo final faltante")
    }

############################################################
# 1. Carregando os pacotes necessários
############################################################

    library(tuneR)  #Funções: readWave
    library(seewave) #Funções: deletew, pastew


############################################################
#2. ENCONTRANDO OS ARQUIVOS .WAV NA PASTA INDICADA
############################################################

    setwd(pasta) #Entrando no diretório indicado

    sons = list.files(path=pasta, pattern = ".wav", all.files = TRUE ) #
Lista os arquivos com a extensão .wav presentes na pasta

    narq <- length(sons) # Contagem do número total de arquivos .wav
presentes na pasta
```

```
#############################################################
#3. VALIDAÇÃO DA MEMÓRIA
#############################################################

    # memory.limit retorna Megabits
    # file.size retorna bytes

    mem.byte <- (memory.limit()*(10^6))/8 #Transformar memory.limit em bytes

    files.tamanho <- sapply(dir((pasta), pattern=".wav"), file.size)  #
Tamanho de cada arquivo da pasta


    list.mem.total <- vector() #Criação de um vetor

    for (r in 1:narq) # Lista com as somas dos tamanhos dos arquivos de som.
Tamanho necessário para o objeto lista.w da etapa de união dos sons.
    {
    list.mem.total[[r]] <- sum(files.tamanho[1:r])
    }

    total.files.size <- sum(list.mem.total[1:narq])


    if ( total.files.size > mem.byte )  #Caso o tamanho total dos arquivos
seja maior que a memória do R do usuário, a função para.
    {
    stop(
        paste("Memória disponível insuficiente para gerar o arquivo final.
Memória necessária =", total.files.size, ">", mem.byte, "= Memória
disponível")
        )
    }


#############################################################
#4. LER TODOS OS ARQUIVOS DE SOM
#############################################################


    list.arq <- list() #Criar uma lista

    for(i in 1:(narq))  # Criar uma lista com cada arquivo de som lido
    {
    list.arq[[i]]= readWave(sons[i]) # Uso da função readWave para ler os
arquivos .wav
    }


#############################################################
```

```
#5. APAGAR OS PRIMEIROS SEGUNDOS DE CADA ARQUIVO
############################################################

    list.apagado <- list() # Criar uma lista

    for(k in 1:(narq)) # Criar uma lista onde estarão todos os arquivos de
som já apagados
    {
    list.apagado[[k]]= deletew(list.arq[[k]], f=sf, from= t0, to=t1,
output="Wave")  # Uso da função deletew para apagar o período determinado
pelos argumentos da minha função
    }



############################################################
#6. UNINDO TODOS OS ARQUIVOS
############################################################

    lista.w <- list() # Criar uma lista

    lista.w[[1]] <- pastew(list.apagado[[1]], list.apagado[[2]],f=sf, at=0,
plot=FALSE, output="Wave") # União dos 2 primeiros arquivos e sua indexação
como primeiro objeto da lista


    if (narq > 2)  # Caso existam mais que 2 arquivos
    {
        for(h in 3:(narq)) # União dos arquivos restantes
        {
        lista.w[[h-1]] <- pastew(lista.w[[h-2]], list.apagado[[h]], f=sf,
at=0, plot=FALSE, output="Wave")
        }
    }

############################################################
#7. SALVAR O ARQUIVO DE WAVE
############################################################

    nome.wav <- paste(nome.arq, ".wav", sep="") #Adicionando a extensão .wav
no nome do arquivo de saída indicado
    nlista.w <- length(lista.w) # Contagem do número de objetos na lista.w

    savewav(lista.w[[nlista.w]], filename = nome.wav) # Salvar o arquivo
final, que é o último aruquivo de "lista.w"


############################################################
#8. LISTA PARAMETROS
############################################################
```

```
    lista.w[[nlista.w]] # Arquivo de som final.

    lista.parametros <- list(lista.w[[nlista.w]],
object.size(lista.w[[nlista.w]])) # Lista com 2 itens : Parâmetros total do
arquivo final, e Tamanho do arquivo final
    names(lista.parametros) <- c("Parâmetros sonoros:", "Tamanho do arquivo
final:") # Nomeando os objetos da lista

    return (lista.parametros) # Objeto de retorno para o usuário é a lista
com os itens descritos acima.

############################################################
# FIM #
############################################################
}
```

Página de ajuda

```
addsound()                package: nenhum                    R Documentation


                              Corte e soma de arquivos de áudio


Description:

    Essa função corta os mesmos trechos de todos os arquivos de som
presentes em uma pasta, e os soma em um último arquivo final.

Usage:

addsound = (pasta="NULL", t0=NULL, t1=NULL, sf=NULL, nome.arq="NULL" )


Arguments:

pasta
      caracteres, caminho para o diretorio onde estao os arquivos de som a
serem tratados

t0
      numerico, tempo do inicio do corte (em segundos).

t1
      numerico, tempo do fim do corte (em segundos).

sf
```

```
      numerico, taxa de amostragem do arquivo (em Hertz).

nome.arq
      caracteres, nome do arquivo final gerado.

Details:

Pasta deve ser um caminho válido para o diretório onde encontram-se os
arquivos de som a serem cortados e adicionados.

Somente arquivos de som em formato ".wav" podem ser tratados pela função.

Caso a memória disponível no programa não seja suficiente para tratar a
quantidade de dados demandada, uma mensagem de erro aparecerá indicando o
problema.

Os argumentos pasta e nome.arq devem ser indicados entre aspas. (e.g.:
pasta= "C:/Users/Documents/Audios", nomes.arq="Final")

Values:

O arquivo de som final gerado estará em formato ".wav" e será salvo no mesmo
diretório dos sons analisádos.
Além do arquivo final exportado, uma lista com detalhes do mesmo será
exibida.
    Componente 1: Parâmetros sonoros.
    Componente 2: Tamanha do arquivo.

A comparação entre memória necessária e memória disponível está em valores
de bytes.

Note:

Os pacotes tuneR e seewave devem estar instalados.


Author(s):

Amanda Martinelli de Oliveira
amartinelli@usp.br

Sao Paulo, junho de 2017

See also:

Veja a função memory.limit() para detalhes sobre a memória do programa.


Examples:

## Baixar os arquivos na seção abaixo. Salve todos os arquivos em um mesmo
```

```
diretório. Utilize o caminho para este diretório no argumento “pasta” dos
exemplos (e.g. addsound(pasta= “C:/Users/Documents/Exemplo1”, t0=0…)
## Aplicando a função com o corte entre 0s e 2.1s. #
## Taxa de amostragem de 96000

addsound(pasta = "C:/Documents/Exemplo1", t0=0, t1=2.1, sf= 96000, nome.arq=
"Total" )
```

Arquivos para o exemplo

Arquivos da função

addsound_codigo.r help.addsound.txt