

Universidade Federal de Uberlândia

## List do Módulo 3

Levy Gabriel da Silva Galvão

Uberlândia  
2021

# Módulo 3

## Exercício 1

Repete e explique a finalidade de cada um dos exemplos realizados neste tutorial.

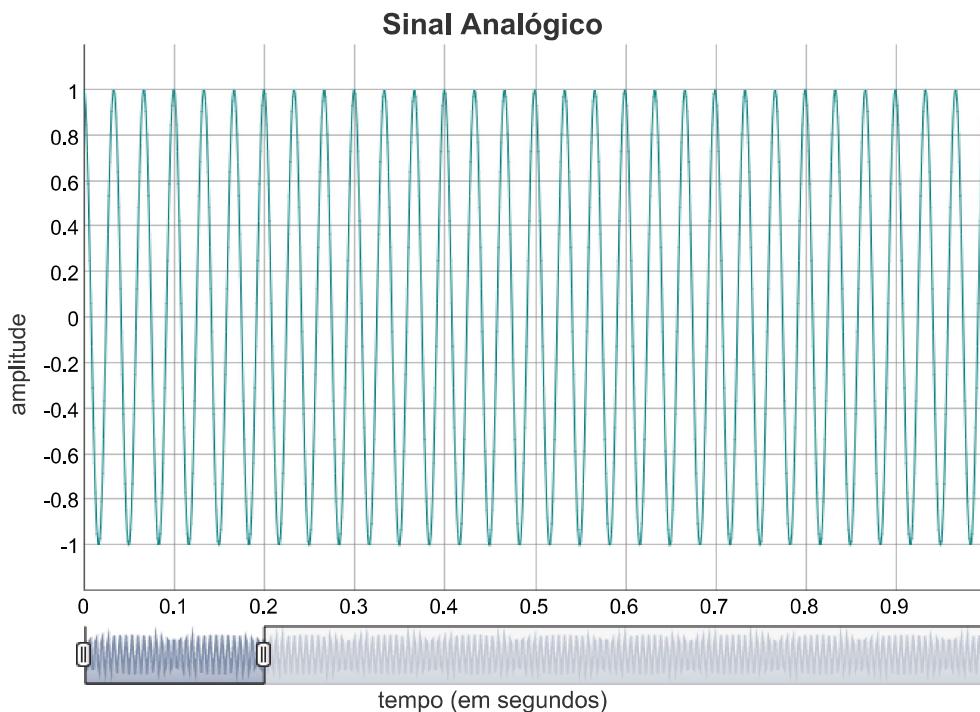
### Exemplo 1:

Este exemplo mostra como simular um sinal analógico no R, permitindo gerar um vetor temporal com período de amostragem e duração escolhidas. O vetor de tempo é usado para gerar um cosseno que é plotado pelo dygraph e de forma que lembra um sinal analógico.

```
library(dygraphs)

fs <- 1000                      # sampling frequency (Hz)
dt <- 1/fs                        # sampling period (s)
tf <- 5                           # duration (s)
t <- seq(from = 0, to = tf, by = dt) # time vector
f <- 30                           # frequency of cosine
y <- cos(2*pi*f*t)               # cosine

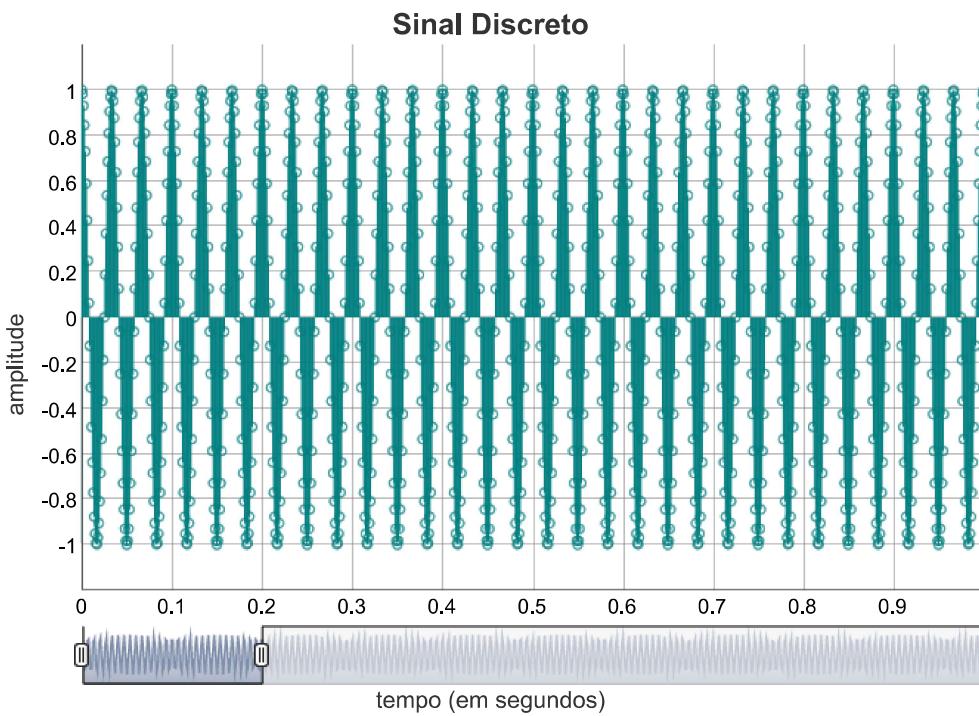
data.frame(t,y) %>%
  dygraph(xlab = "tempo (em segundos)",
         ylab = "amplitude",
         main = "Sinal Analógico",
         group = "S") %>%
  dyRangeSelector(dateWindow = c((517-1)*dt, (600-1)*dt))
```



### Exemplo 2:

O segundo exemplo se vale do sinal criado anteriormente, porém com uma abordagem diferente na plotagem que, por meio de um stem plot acentua as amostras discretas do vetor criado, demonstrando que na verdade aquele vetor se tratar de um sinal discreto.

```
data.frame(t,y) %>%
  dygraph(xlab = "tempo (em segundos)",
         ylab = "amplitude",
         main = "Sinal Discreto",
         group = "S") %>%
  dyRangeSelector(dateWindow = c((517-1)*dt, (600-1)*dt)) %>%
  dySeries(stemPlot=TRUE)
```



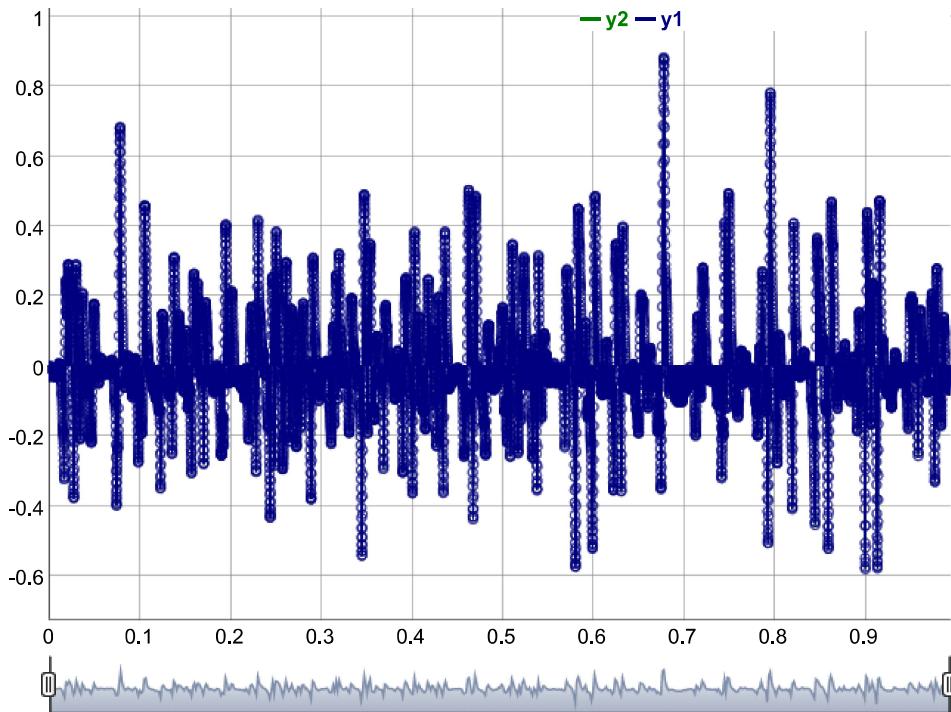
### Exemplo 3:

Este exemplo trata da leitura e extração de características de um arquivos .xlsx. São lidos os sinais e a característica de resolução temporal é extraída para que os plots sejam realizados.

```
library(openxlsx)
library(dygraphs)

df1 <- read.xlsx("Dados/Dados1.xlsx", sheet = 1, skipEmptyRows = FALSE, rows = c(1:10000)) # read data from .xlsx
dt <- df1$Tempo[2] - df1$Tempo[1] # extract sampling frequency

data.frame(time=df1$Time, y1 = df1$EMG1, y2=df1$EMG1) %>%
  dygraph(group="EMG") %>%
  dyRangeSelector() %>%
  dySeries("y1",stemPlot=TRUE)
```



### Exemplo 4:

Este exemplo ilustra a ocorrência de eventos discretos por meio do valor de pico em sinais de EMG.

```
library(pracma) #package para usar a função findpeaks
library(htmltools)
```

```

th <- max(df1$EMG1, na.rm = TRUE)*0.4 # Limiar

# help(findpeaks) --> para saber mais sobre os argumentos da função
pp <- findpeaks(df1$EMG1, minpeakheight=th) #encontrando picos (esta função retorna uma matriz)

indxpeaks <- pp[ , 2] #retorna a segunda coluna da matriz (que contém os índices em que os picos aconteceram)

## Saiba mais em http://dygraphs.com/tests/independent-series.html
df1$peaks <- NA #inicializando um vetor com dados faltantes (NA = not a number)

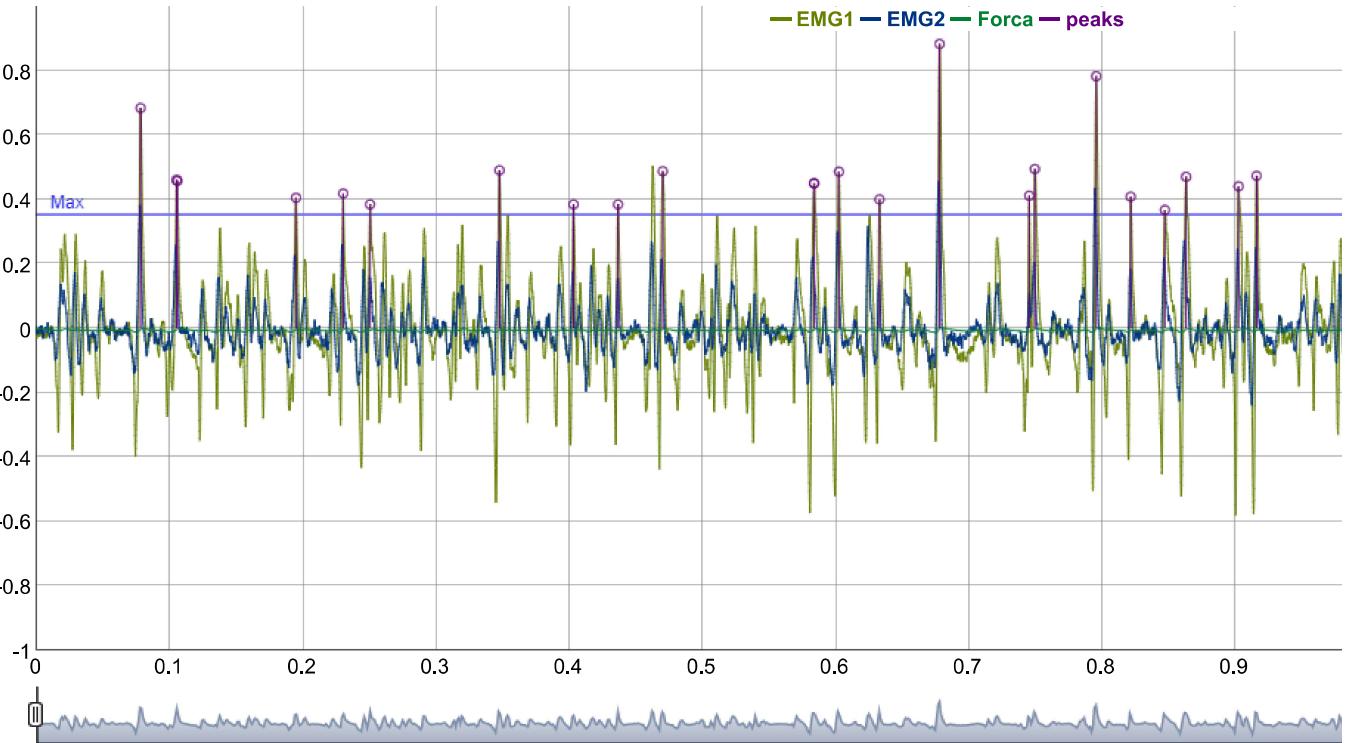
df1$peaks[indxpeaks] <- df1$EMG1[indxpeaks]

d1 <- dygraph(df1, group="E1")%>%
  #dyOptions(connectSeparatedPoints=TRUE)%>%
  dySeries("peaks", stemPlot=TRUE)%>%
  dyAxis("y", valueRange = c(-1, 1)) %>%
  dyLegend(width = 400)%>%
  dyLimit(th, "Max",
          strokePattern = "solid", color = "blue")%>%
  dyRangeSelector()

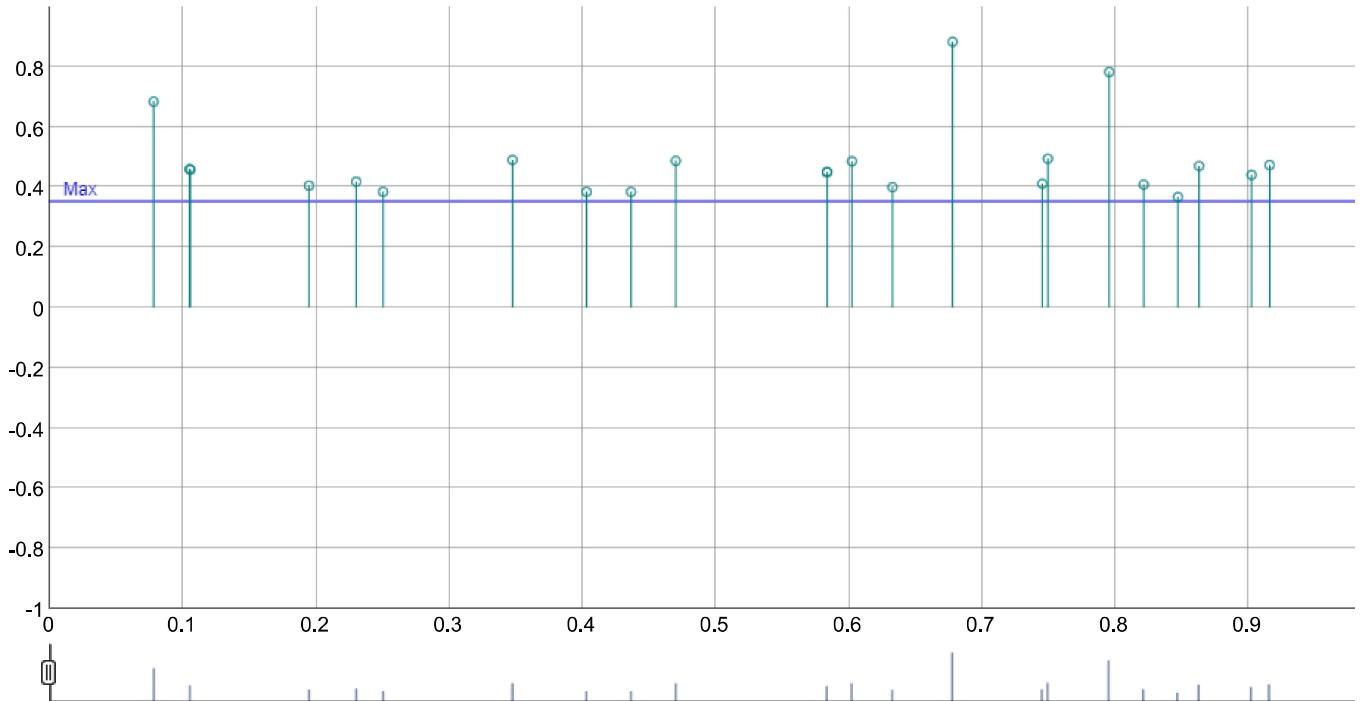
d2 <- dygraph(df1[c("Time", "peaks")], main="Eventos discretos", group="E1")%>%
  #dyOptions(connectSeparatedPoints=TRUE)%>%
  dySeries("peaks", stemPlot=TRUE)%>%
  dyAxis("y", valueRange = c(-1, 1)) %>%
  dyLegend(width = 400)%>%
  dyLimit(th, "Max",
          strokePattern = "solid", color = "blue")%>%
  dyRangeSelector()

dy_graph <- list(d1,d2)
# render the dygraphs objects using htmltools
htmltools::browsable(htmltools::tagList(dy_graph))

```



## Eventos discretos



### Exemplo 5:

Este exemplo ilustra a amostragem de sinais, mas iniciando com a definição de um tevot tempo e a frequência de amostragem.

```
fs <- 500 #frequência de amostragem em Hz
dt <- 1/fs #resolução temporal em segundos
ti <- 0.0 # tempo inicial em segundos
tf <- 3.5 # tempo final em segundos

t <- seq(from=ti, to=tf, by=dt) # vetor de tempo discreto (em segundos)

length(t) # número de amostras, N, do vetor de tempo discreto
```

```
## [1] 1751
```

```
min(t) # tempo mínimo (s)
```

```
## [1] 0
```

```
max(t) # tempo máximo (s)
```

```
## [1] 3.5
```

### Exemplo 6:

Com a definição do vetor tempo do exemplo anterior, sinais de experimentos podem ser coletados.

```
# install.packages("tuneR")
# https://www.rdocumentation.org/packages/tuneR/versions/1.3.3/topics/Waveforms
library(tuneR)

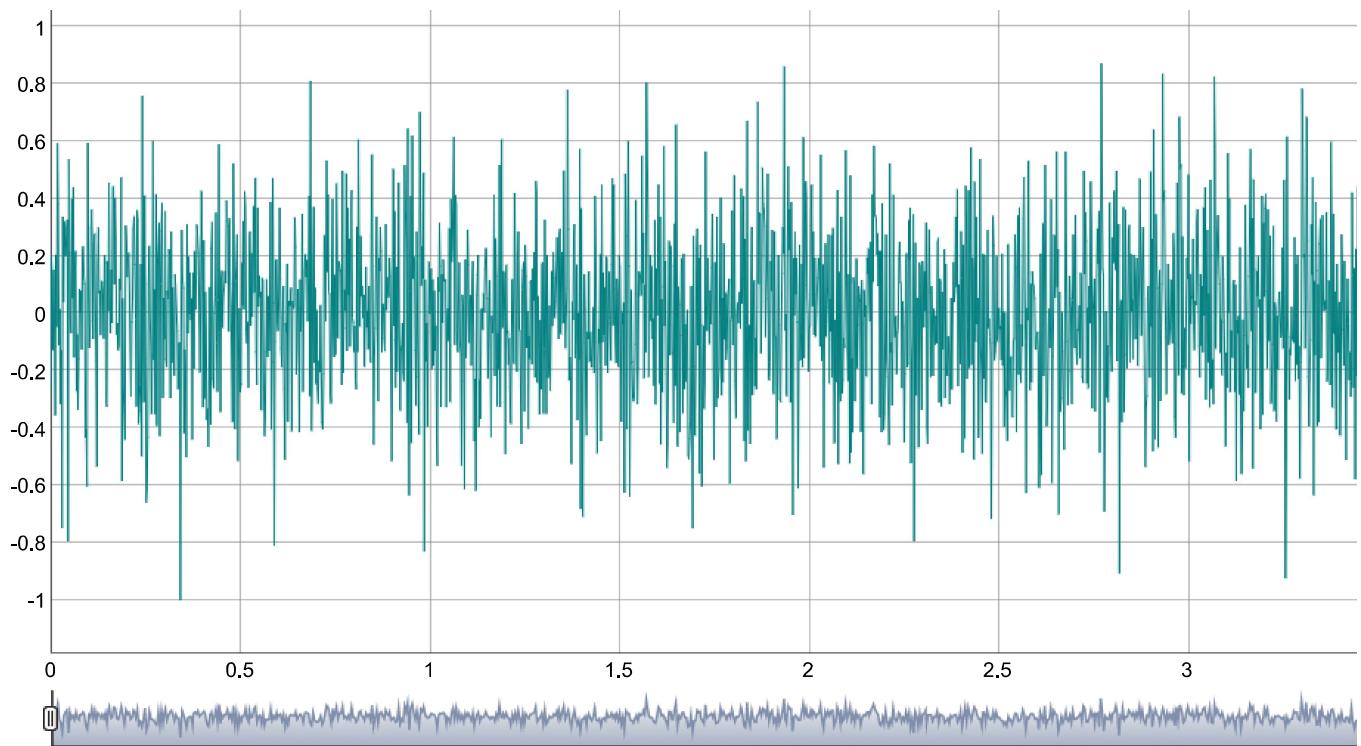
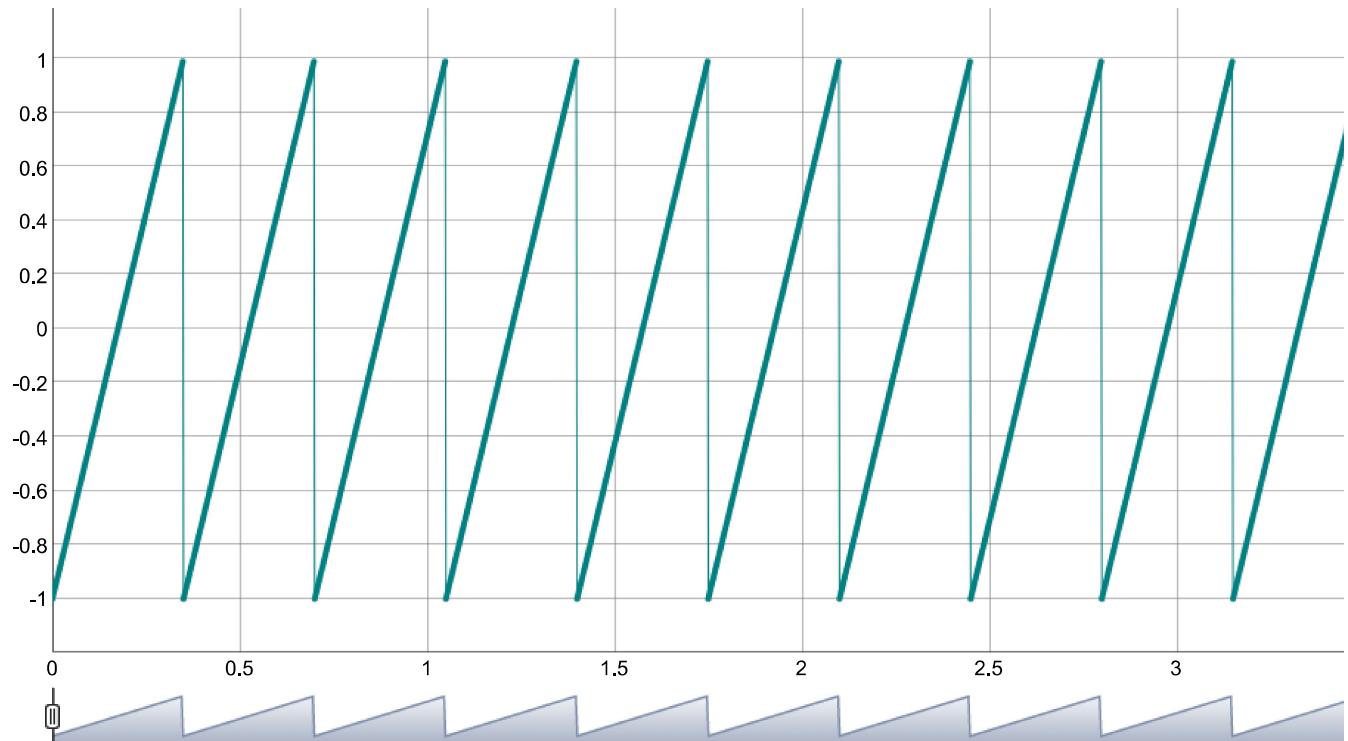
freq1 <- 10
y1 <- sawtooth(freq1, samp.rate = fs, duration = tf+dt,xunit = "time")@left
d1 <- dygraph(data.frame(time=t, y1)) %>%dyRangeSelector() %>% dyOptions(drawPoints = TRUE, pointSize = 2)

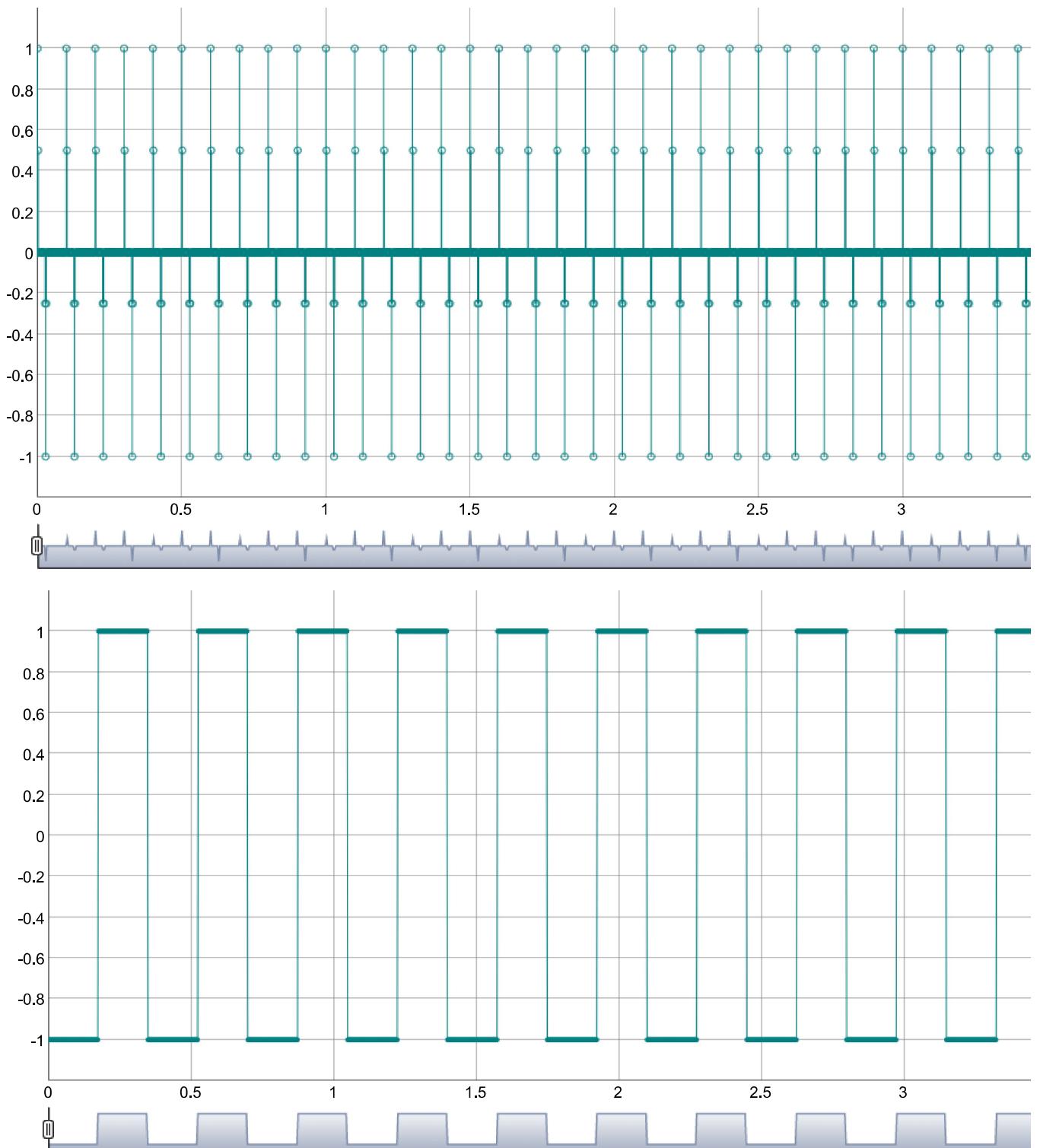
y2 <- noise(kind="white", samp.rate = fs, duration = tf+dt,xunit = "time")@left
d2 <- dygraph(data.frame(time=t, y2)) %>%dyRangeSelector()

y3 <- pulse(freq1, samp.rate = fs, duration = tf+dt,xunit = "time")@left
d3 <- dygraph(data.frame(time=t, y3)) %>%dyRangeSelector() %>% dySeries("y3", stemPlot = TRUE)

y4 <- square(freq1, samp.rate = fs, duration = tf+dt,xunit = "time")@left
d4 <- dygraph(data.frame(time=t, y4)) %>%dyRangeSelector() %>% dyOptions(drawPoints = TRUE, pointSize = 2)
```

```
dy_graph <- list(d1,d2,d3, d4)
# render the dygraphs objects using htmltools
htmltools::browsable(htmltools::tagList(dy_graph))
```





### Exemplo 7:

Esse exemplo trata da quantização, mostrando níveis discretos que o sinal de interesse quantizado pode assumir e, diferente da sua contraparte analógica, este assume níveis discretos e limitados.

```
#unLoadNamespace("tuneR")
library(dygraphs)

#Library(Lambda.tools)
fs <- 1000 # frequência de amostragem em Hz
dt <- 1/fs # resolução temporal em segundos
tf <- 5 # tempo final (em segundos) do evento
t <- seq(from = 0, to = tf, by = dt) # vetor de tempo

f <- 30 # frequência de oscilação

y <- cos(2*pi*f*t)
```

```

y <- ifelse(y > 0.98, 1, y) # esse passo é necessário para evitar erros de arredondamento (a intenção é aproximar o valor 0.98 para 1)

Vmax <- max(y) # valor máximo
Vmin <- min(y) # valor mínimo

k <- 2 # número de bits

Q <- (Vmax - Vmin)/2^k # resolução A/D

qts <- seq(from = Vmin, to = Vmax, by = Q) # níveis disponíveis

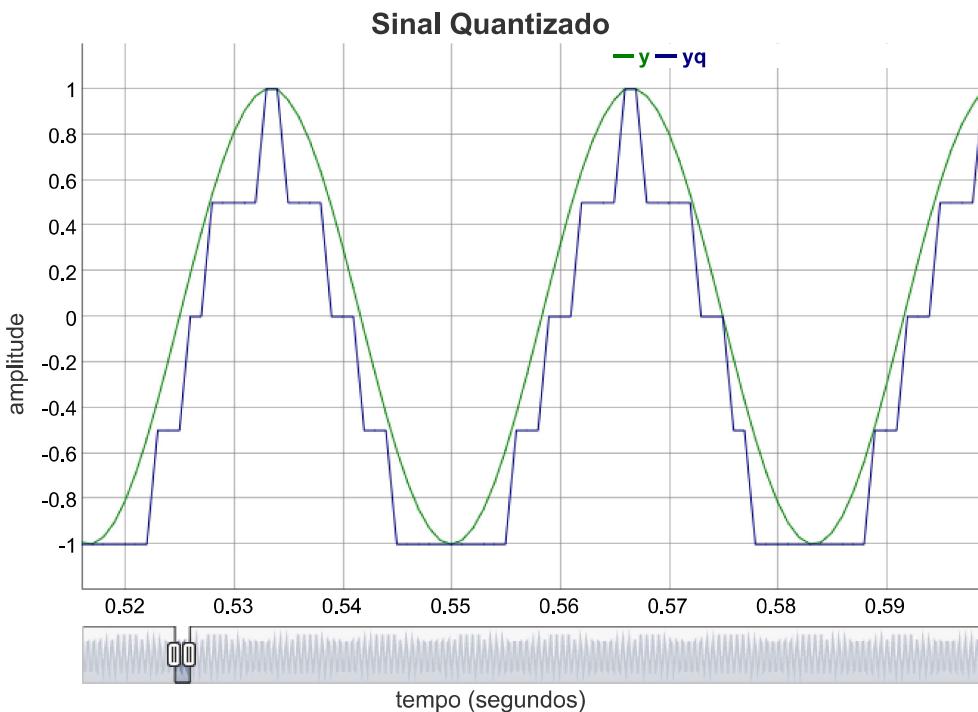
# Quando quantizamos um sinal analógico, pretendemos gerar uma aproximação para este de acordo com determinados níveis.

yq <- qts[findInterval(y, qts)]

#Lambda.tools::quantize(y,bins = qts) # aproximando o sinal aos níveis disponíveis

dygraph(data.frame(t, y, yq), xlab = "tempo (segundos)", ylab = "amplitude", main = "Sinal Quantizado")%>%dyRangeSelector(c((517-1)*dt, (600-1)*dt))

```



## Exemplo 8:

Este exemplo ilustra o efeito do Aliasing.

- Inicialmente o sinal é criado;
- Em seguida ele é replicado com uma taxa de amostragem maior (evitando completamente o Aliasing);
- Porém logo em seguida a taxa de amostragem é reduzida e este sofre do efeito de Aliasing, pois as amostras existentes não são representativas para reconstruir o sinal original;
- Os dois outros plots seguintes acentuam ainda mais o efeito do Aliasing;

```

library(ggplot2)
library(plotly)

## 
## Attaching package: 'plotly'

## The following object is masked from 'package:ggplot2':
## 
##     last_plot

## The following object is masked from 'package:stats':
## 
##     filter

## The following object is masked from 'package:graphics':
## 
## 
```

```

##      layout

fs <- 40 # frequencia de amostragem (em Hz)

dt <- 1/fs # resolucao temporal (em segundos)

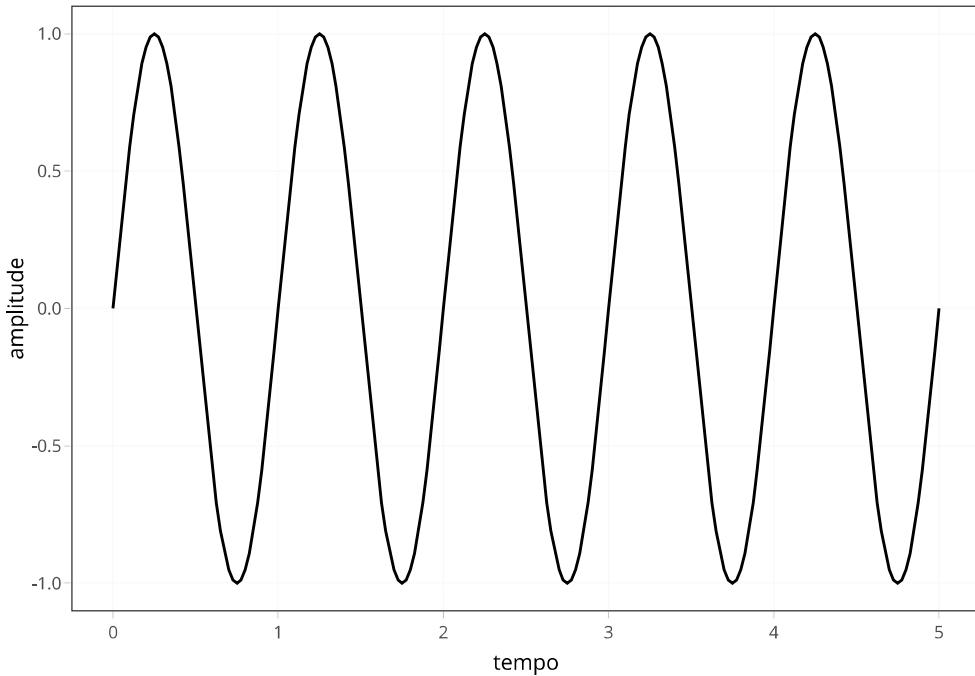
t <- seq(from = 0, to = 5, by = dt) # vetor de tempo

f <- 1 # frequencia de oscilacao (em Hz)

ys <- sin(2*pi*f*t) # sinal de interesse

p<-ggplot(data.frame(t,ys),aes(t,ys))+geom_line()+theme_bw() +xlab("tempo") +ylab("amplitude")
ggplotly(p)

```



```

#####
fs2 <- 100 # frequencia de amostragem (> 2 * f_sinal)

dt2 <- 1/fs2 # resolucao temporal p/ o sinal amostrado a 100 Hz

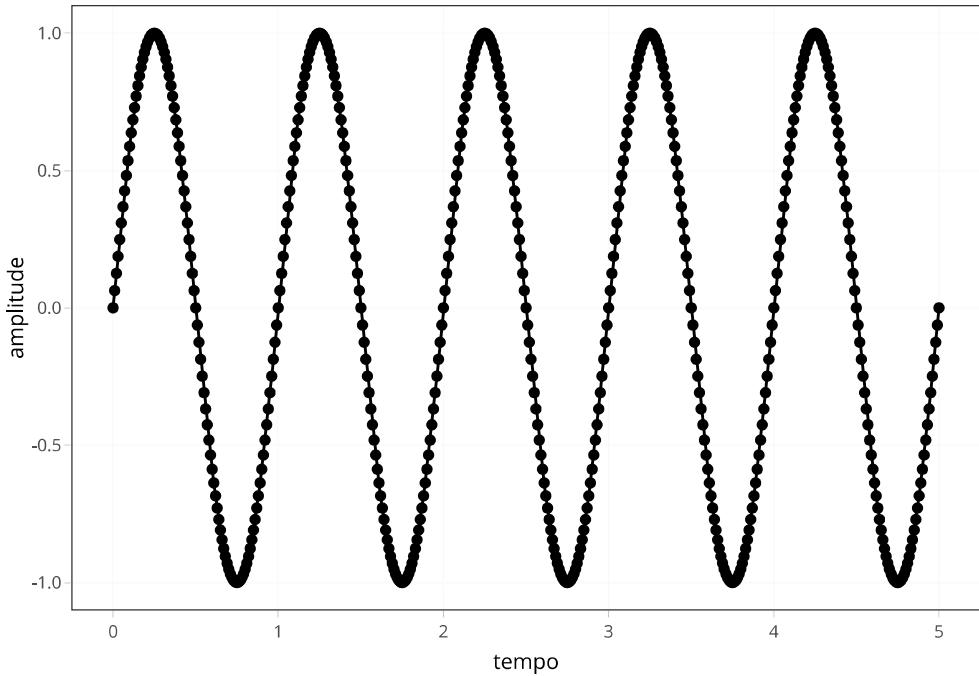
nmax2 <- floor(5/dt2) # numero de amostras p/ o sinal amostrado a 100 Hz

n2 <- 0:nmax2 # vetor numero de amostras

ys2 <- sin(2*pi*f*n2*dt2) # sinal amostrado a 100 Hz

p<-ggplot(data.frame(n2*dt2,ys2),aes(n2*dt2,ys2))+geom_line()+geom_point()+theme_bw() +xlab("tempo") +ylab("amplitude")
ggplotly(p)

```



```
#####
fs3 <- 15 # frequencia de amostragem (> 2 * f_sinal)

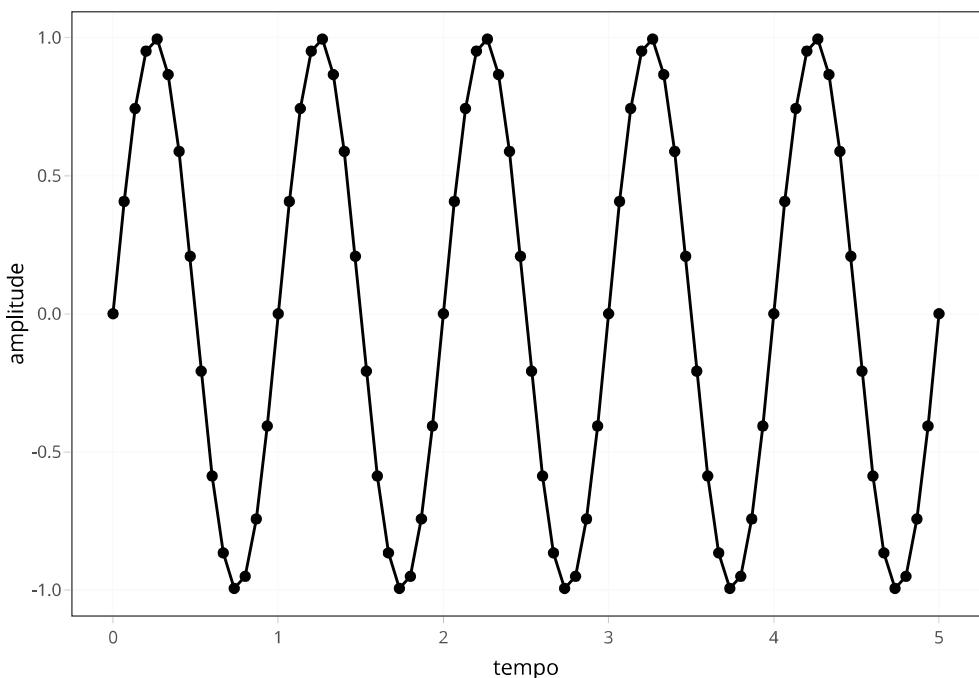
dt3 <- 1/fs3 # resolucao temporal p/ o sinal amostrado a 15 Hz

nmax3 <- floor(5/dt3) # numero de amostras p/ o sinal amostrado a 15 Hz

n3 <- 0:nmax3 # vetor numero de amostras p/ o sinal amostrado a 15 Hz

ys3 <- sin(2*pi*f*n3*dt3) # sinal amostrado a 15 Hz

p<-ggplot(data.frame(n3*dt3,ys3),aes(n3*dt3,ys3))+geom_line()+geom_point()+theme_bw()+xlab("tempo")+ylab("amplitude")
ggplotly(p)
```



```
#####
fs4 <- 0.75 # frequencia de amostragem (< 2 * f_sinal)

dt4 <- 1/fs4 # resolucao temporal p/ o sinal amostrado a 0.75 Hz
```

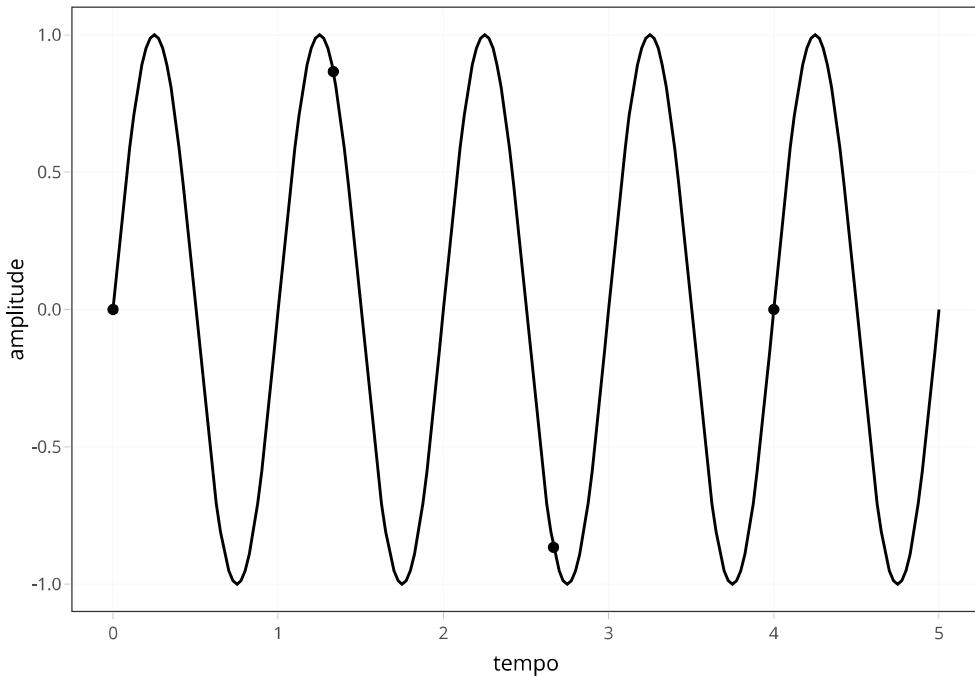
```

nmax4 <- floor(5/dt4) # numero de amostras p/ o sinal amostrado a 0.75 Hz
n4 <- 0:nmax4 # vetor numero de amostras p/ o sinal amostrado a 0.75 Hz
ys4 <- sin(2*pi*f*n4*dt4) # sinal amostrado a 0.75 Hz

df11<-data.frame(time=t,y=ys)
df2<-data.frame(time=n4*dt4,y=ys4)

p<-ggplot(df11,aes(time,y))+geom_line()+geom_point(data=df2,aes(time,y))+theme_bw() +xlab("tempo") +ylab("amplitude")
ggplotly(p)

```



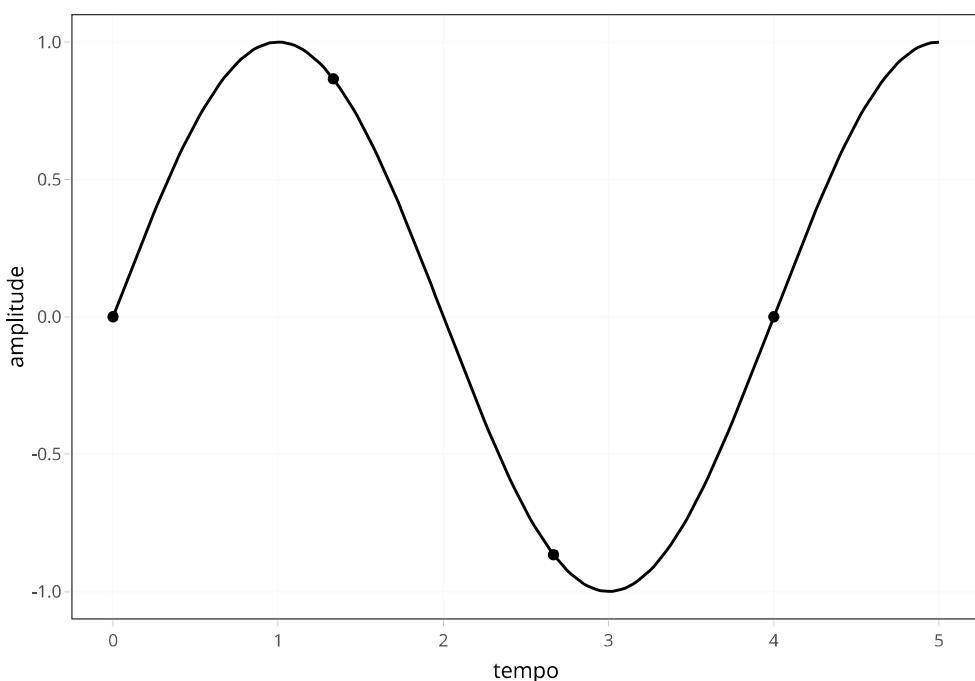
```

#####
yc = sin(2*pi*0.25*t) # sinal senoidal de 0.25 Hz

df11<-data.frame(time=t,y=yc)
df2<-data.frame(time=n4*dt4,y=ys4)

p<-ggplot(df11,aes(time,y))+geom_line()+geom_point(data=df2,aes(time,y))+theme_bw() +xlab("tempo") +ylab("amplitude")
ggplotly(p)

```

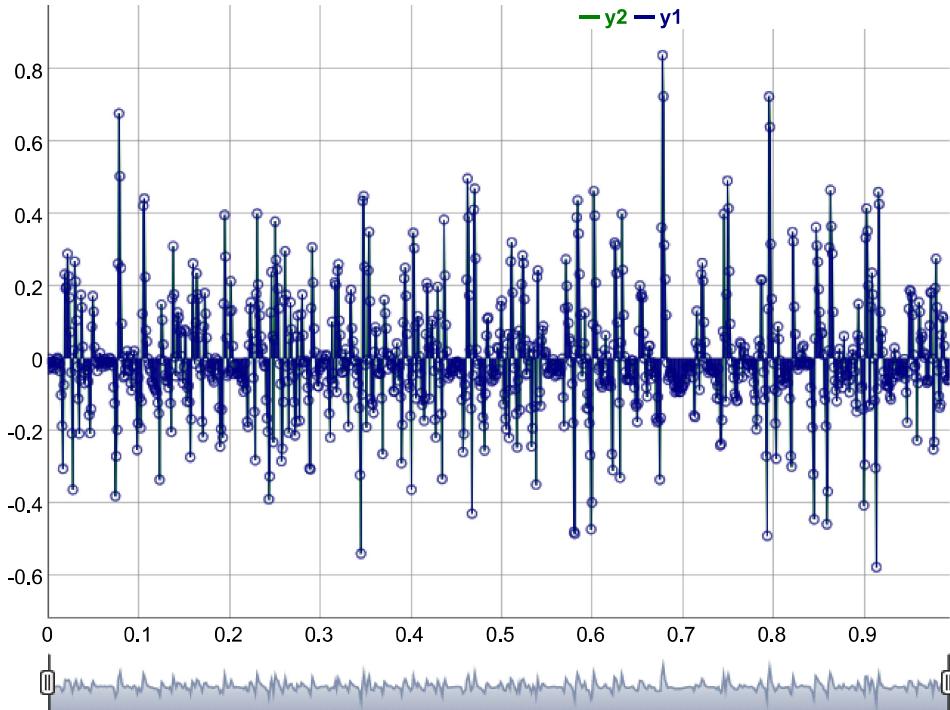


## Exemplo 9:

Este exemplo ilustra a subamostragem, quando amostras do sinal são descartadas.

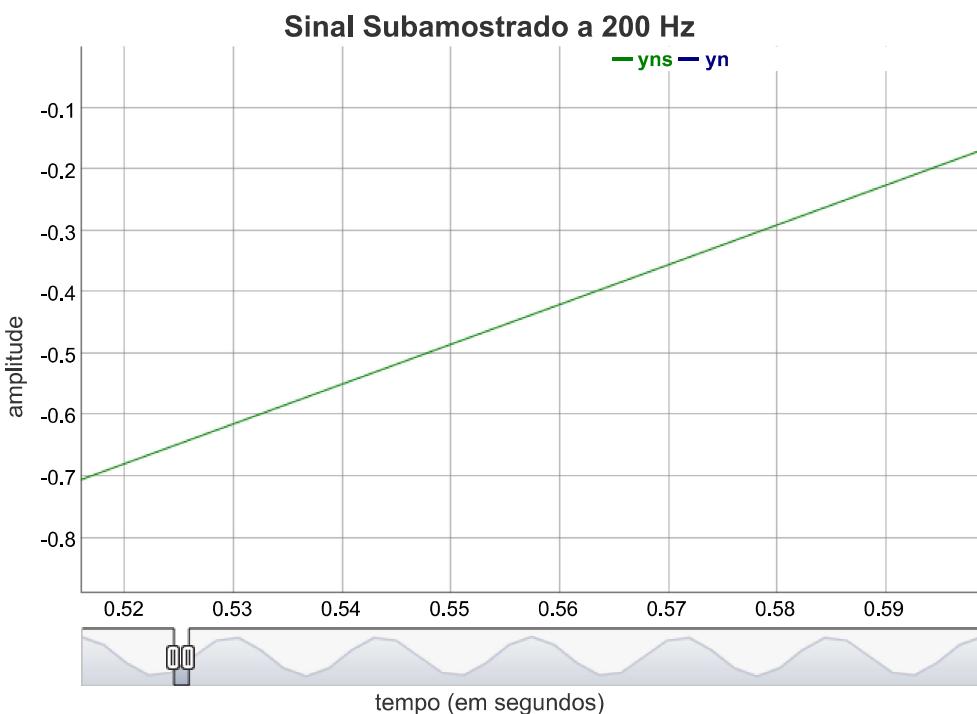
```
#subamostrando o sinal = reduzindo o número de amostras (observe o parâmetro by=10)
indx <- seq(from=1,to=length(df1$Time), by=10)
df2 <- data.frame(time=df1$Time[indx], y1 = df1$EMG1[indx])

dygraph(data.frame(time=df2$time, y1 = df2$y1, y2=df2$y1), group="EMG") %>%
  dyRangeSelector()%>
  dySeries("y1",stemPlot=TRUE)
```



```
i <- seq(from = 1, to = tf*fs+1, by = 5) # índices, descartando 4 amostras a cada 5

dt <- 1/1000
dygraph(data.frame(tn = t[i], yn = y[i], yns = y[i]), xlab = "tempo (em segundos)", ylab = "amplitude", main = "Sinal Subamostrado a 200 Hz")%>%dySeries(stemPlot=TRUE)%>%dyRangeSelector(c((517-1)*dt,(600-1)*dt))
```



## Exemplo 10:

Este exemplo ilustra a reamostragem de um sinal, permitindo que por meio de um sinal subamostrado, aplique-se a interpolação e sejam geradas amostras novas do sinal, aumentando a taxa de amostragem.

```
library(ggplot2)

# Definição da frequência de amostragem em Hz
fs1 <- 50

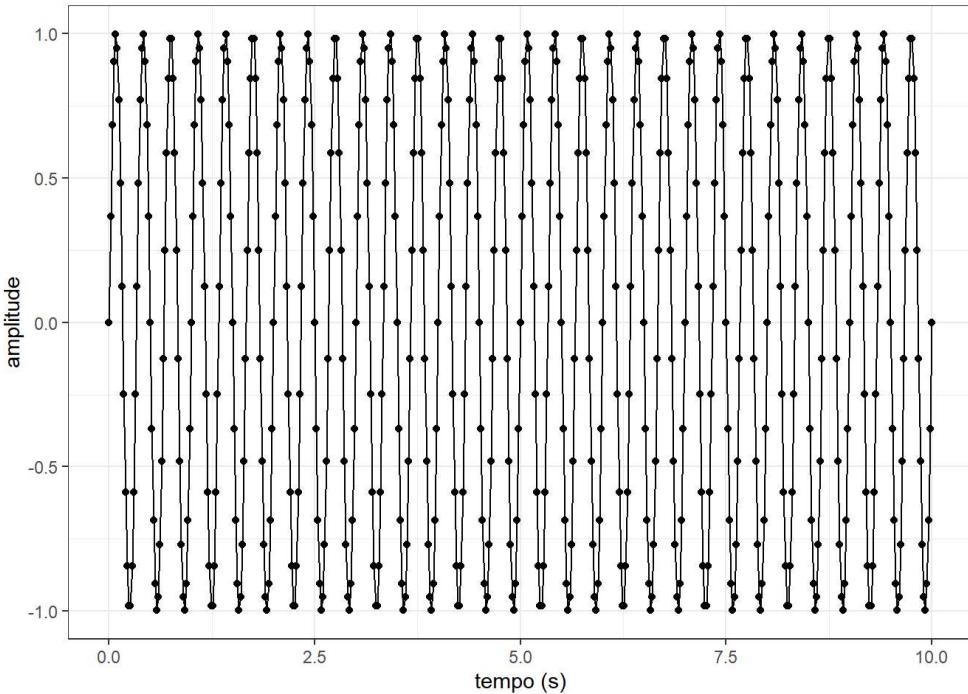
# Definição intervalo entre as amostras em segundos
dt1 <- 1/fs1

# Definição do vetor de tempo (em segundos)
# Para help digite: help("seq")
t1 <- seq(0, 10, by=dt1) # Aprenda mais em https://www.datamentor.io/r-programming/vector/

# Definição de um sinal senoidal
f1 <- 3 # frequência de oscilação da senoide
y1 <- sin(2*pi*f1*t1)

# Criando um data frame (é muito importante trabalhar com data frames no R!)
df1 <- data.frame(t1, y1) # Ler http://adv-r.had.co.nz/Data-structures.html

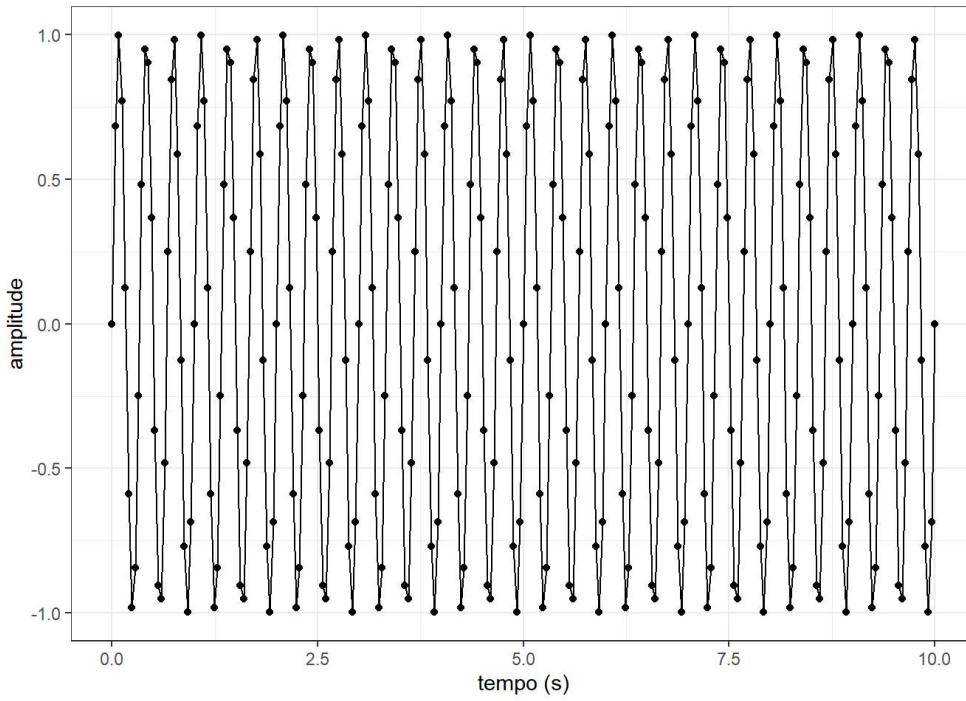
g <- ggplot(data = df1, aes(x = t1, y=y1)) +
  ggplot2::geom_line() +
  ggplot2::geom_point() +
  ggplot2::theme_bw() +
  ggplot2::xlab("tempo (s)") +
  ggplot2::ylab("amplitude")
print(g)
```



```
#####
```

```
# Alterando a frequência de amostragem para 25 Hz
fs2 <- 25 # frequência de amostragem em Hz
dt2 <- 1/fs2 # resolução temporal em segundos
t2 <- seq(0, 10, by=dt2) # tempo
f2 <- 3 # frequência de oscilação da senoide
y2 <- sin(2*pi*f2*t2)
df2 <- data.frame(t2, y2) # Criando um data frame

g <- ggplot(data = df2, aes(x = t2, y=y2)) +
  ggplot2::geom_line() +
  ggplot2:: geom_point() +
  ggplot2::theme_bw() +
  ggplot2::xlab("tempo (s)") +
  ggplot2::ylab("amplitude")
print(g)
```

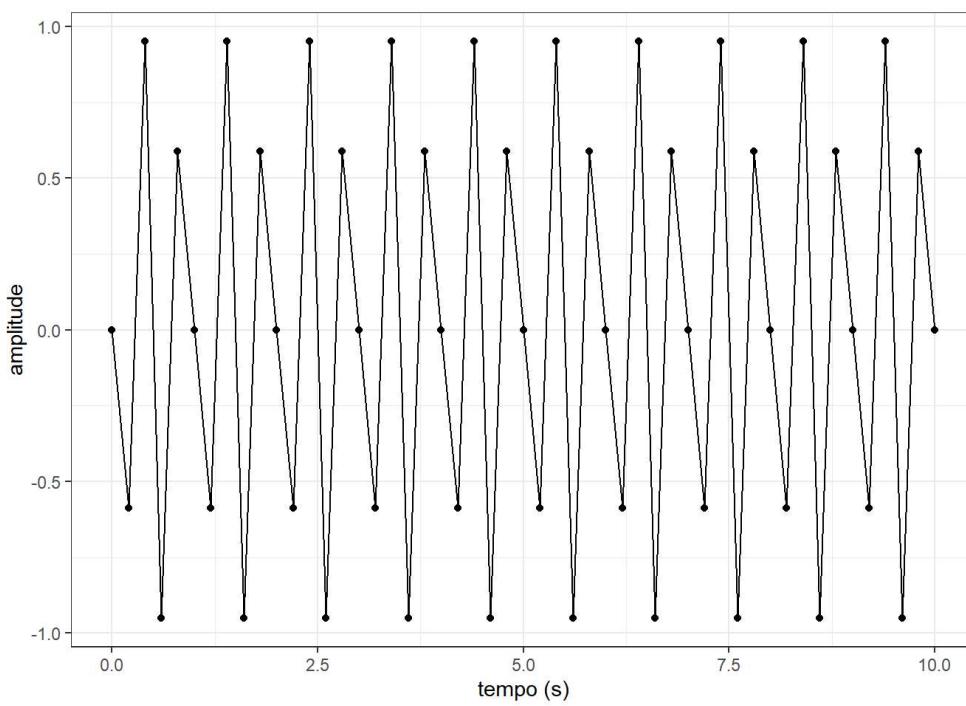


```
#####

```

```
# Alterando a frequência de amostragem para 5 Hz
fs3 <- 5 # frequência de amostragem em Hz
dt3 <- 1/fs3 # resolução temporal em segundos
t3 <- seq(0, 10, by=dt3) # tempo
f3 <- 3 # frequência de oscilação da senoide
y3 <- sin(2*pi*f3*t3)
df3 <- data.frame(t3, y3) # Criando um data frame

g <- ggplot(data = df3, aes(x = t3, y=y3)) +
  ggplot2::geom_line() +
  ggplot2:: geom_point() +
  ggplot2::theme_bw() +
  ggplot2::xlab("tempo (s)") +
  ggplot2::ylab("amplitude")
print(g)
```



```
#####

```

```
library(dygraphs)

# criando um vetor de tempo único, que contenha todos os instantes dos sinais gerados
tt <- dplyr::union(df1$t1, df2$t2)
tt <- dplyr::union(tt, df3$t3)
tt <- sort(tt) # ordenando o tempo em ordem crescente

# Criando um data frame para facilitar a plotagem de gráficos com eixos temporais distintos
# criando o data frame (inicializando cada um dos vetores no data frame)
df5 <- data.frame(time=tt)
df5$y1 <- NA
df5$y2 <- NA
df5$y3 <- NA
head(df5) # visualizando as primeiras linhas do data frame que foi criado
```

```

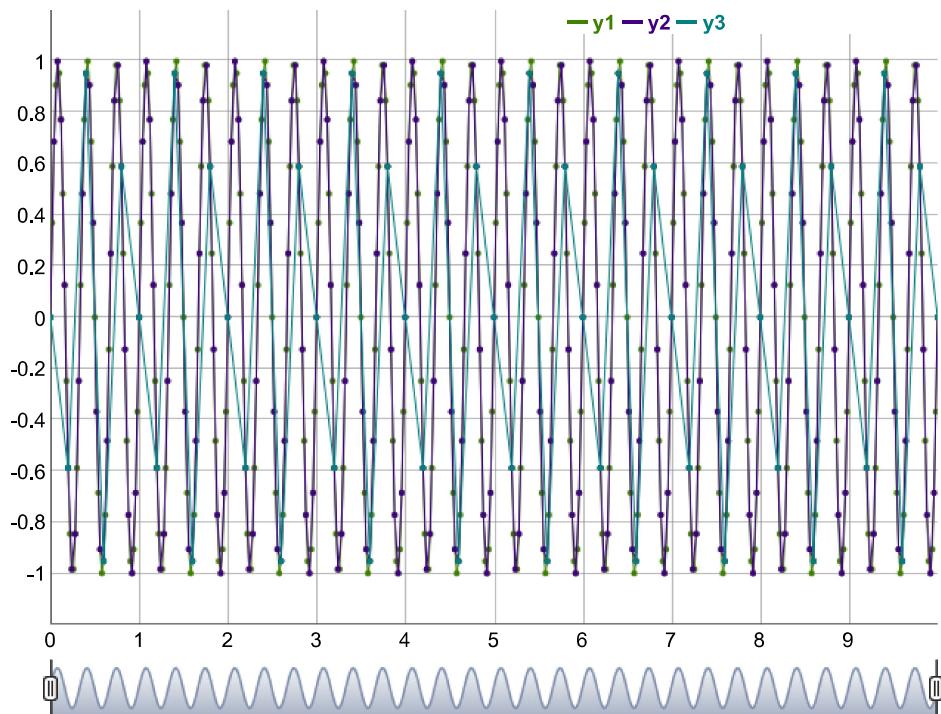
##    time y1 y2 y3
## 1 0.00 NA NA NA
## 2 0.02 NA NA NA
## 3 0.04 NA NA NA
## 4 0.06 NA NA NA
## 5 0.08 NA NA NA
## 6 0.10 NA NA NA

```

*#preenchendo cada vetor no data frame*

```
df5$y1[ tt %in% df1$t1 ] <- df1$y1  
df5$y2[ tt %in% df2$t2 ] <- df2$y2  
df5$y3[ tt %in% df3$t3 ] <- df3$y3
```

```
dygraph(df5) %>%
  dyRangeSelector() %>%
  dyOptions(drawPoints = TRUE, connectSeparatedPoints=TRUE, pointSize = 2)%>%
  dyHighlight(highlightCircleSize = 5,
              highlightSeriesBackgroundAlpha = 0.2,
              hideOnMouseOut = FALSE)
```



####

# Interpolação -----

```
tempo_do_sinal_original <- df3$t3  
amplitude_do_sinal_original <- df3$y3  
tempo discreto desejado <- df1$t1
```

```

        xout = tempo_discreto_desejado))

amplitude_do_sinal_interpolado <- df4$y
tempo_do_sinal_interpolado <- df4$x

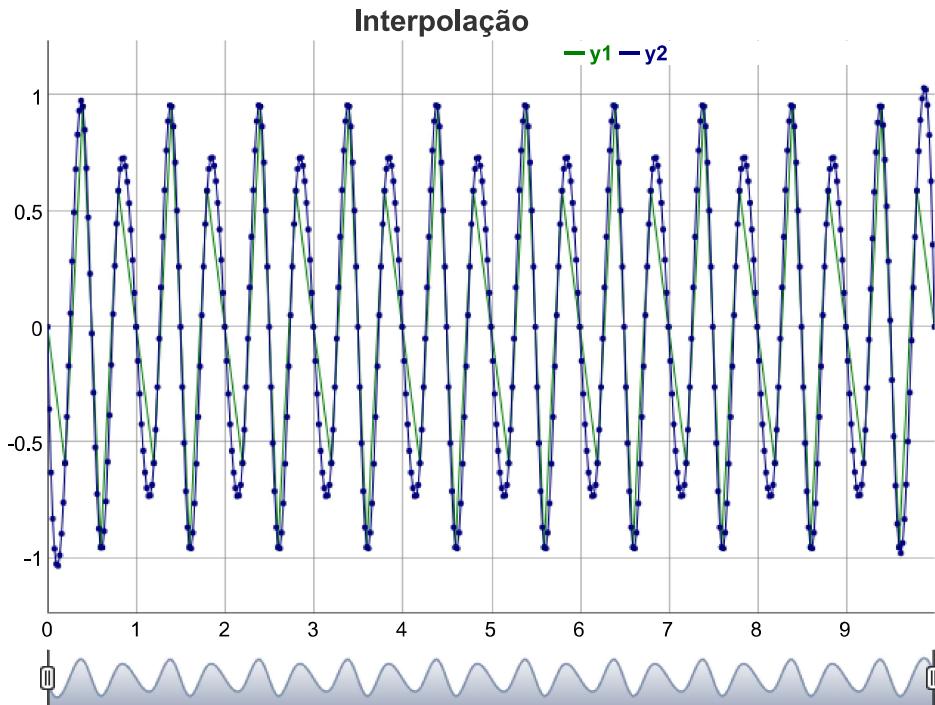
tt <- union(tempo_do_sinal_original, tempo_do_sinal_interpolado)
tt <- sort(tt)

df6 <- data.frame(time=tt)
df6$y1 <- NA
df6$y2 <- NA

df6$y1[tt %in% tempo_do_sinal_original] <- amplitude_do_sinal_original
df6$y2[tt %in% tempo_do_sinal_interpolado] <- amplitude_do_sinal_interpolado

dygraph(df6, main = "Interpolação") %>%
  dyRangeSelector() %>%
  dyOptions(drawPoints = TRUE, connectSeparatedPoints=TRUE, pointSize = 2) %>%
  dyHighlight(highlightCircleSize = 5,
              highlightSeriesBackgroundAlpha = 0.2,
              hideOnMouseOut = FALSE)

```



## Exemplo 11:

Este é um exemplo clássico da interpolação usando a função *splines*

```
# Exemplo de interpolação utilizando splines
```

```

library(ggplot2)
library(plotly)

fs <- 500 # Hz
dt <- 1/fs # sec

tf <- 5 # sec

t <- seq(from=0, to=tf, by = dt)
y <- sin(2*pi*t * 10)

fs1 <- 100 #Hz
dt1 <- 1/fs1
t1 <- seq(from=0, to=tf, by=dt1)

y1 <- spline(x=t, y =y,xout =t1)

df <- data.frame(t, y)
df1 <- data.frame(time = t1, y1$y)

```

```

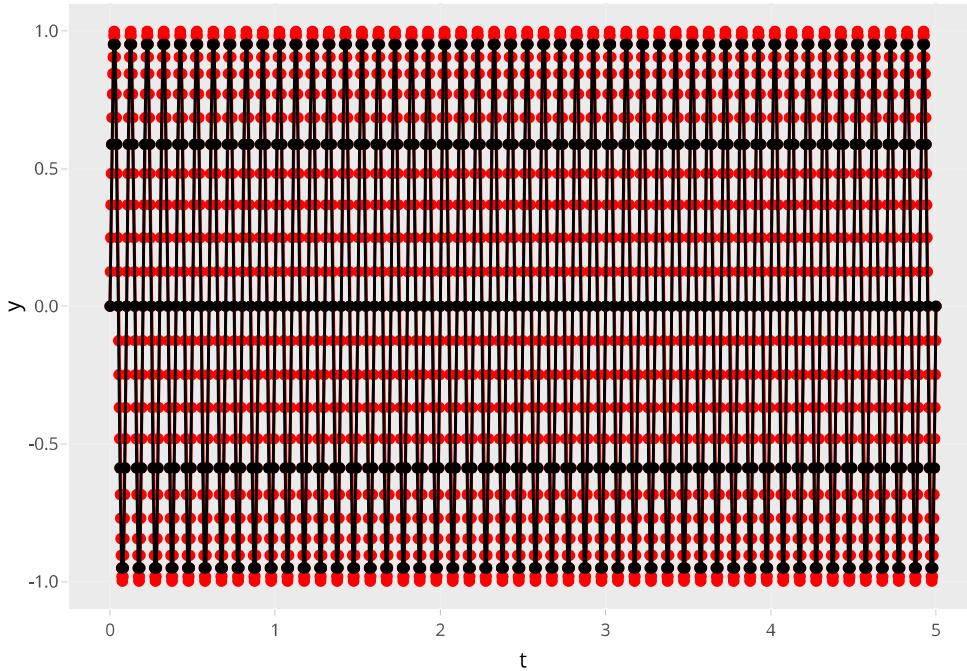
p <- ggplot()

p <- p + geom_line(data = df, aes(x = t, y = y), color = "red" ) +
      geom_point(data = df, aes(x = t, y = y), color = "red" )

p <- p + geom_point(data = df1, aes(x = time, y = y1.y)) +
      geom_line(data = df1, aes(x = time, y = y1.y))

ggplotly(p) # produz um gráfico interativo (disponível na biblioteca plotly)

```



## Exercício 2

Utilize o R para ilustrar, em um mesmo gráfico, os conceitos de sinais analógico e discreto.

Uma forma de ilustrar a comparação entre sianis analógicos e discretos é, por meio de um mesmo vetor, realizar sua plotagem com diferentes traços, sendo uma com uma linha contínua, enquanto que a outra apresentada por meio de marcadores discretos

```

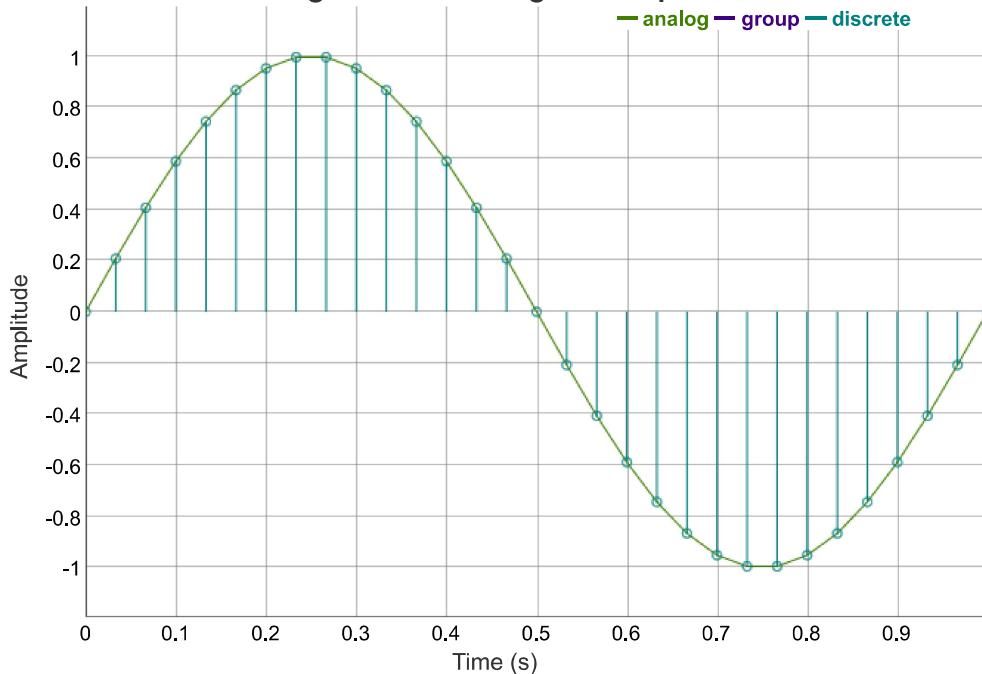
library(dygraphs)

fs <- 30                      # sampling frequency (Hz)
dt <- 1/fs                      # sampling period (s)
f <- 1                          # frequency of cosine
tf <- 1/f                        # duration (s) (two periods of the waveform)
t <- seq(from=0, to=tf, by=dt) # time vector
data <- sin(2*pi*f*t)          # cosine

data.frame(time=t, analog=data, discrete=data, group="Comparison") %>%
  dygraph(xlab = "Time (s)",
         ylab = "Amplitude",
         main = "Analog vs. discrete signal comparison",
         group = "S") %>%
  dySeries("discrete",stemPlot=TRUE)

```

## Analog vs. discrete signal comparison



## Exercício 3

Explique os passos necessários para fazer a detecção de picos em uma série temporal.

Os picos em uma série temporal são detectados quando a amplitude desta supera um limiar previamente definido. Porém não é certeza que existirá um único valor acima desse limiar, assim sendo necessário identificar o maior valor desse conjunto de valores que superam o limiar. Portanto, considerando os valores acima do limiar, o pico pode ser identificado ao monitorar a variação no sinal da derivada que, quando o sinal aumenta, esta é positiva, mas quando ele decresce, ela será negativa e o pico será exatamente no ponto de inflexão dessa derivada.

Para termos de comparação será utilizada a função `findpeaks` para rastrear os picos acima de um limiar positivo, enquanto que será utilizada uma função própria para identificar os picos negativos, mas que pode ser utilizado para identificar os picos positivos e foi utilizado dessa forma para que não coincida com os picos identificados pela função `findpeaks`.

```

library(openxlsx)
library(dygraphs)
library(pracma)

originalFindPeaks <- function(data, threshold, direction='max') {
  if(direction=='max') {
    filt <- data * (data>threshold) # filter data values LOWER than threshold
    dfilt <- c(0,diff(filt))>0      # detect if data amplitude is RISING (1) or not (0)

  } else if(direction=='min') {
    filt <- data * (data<threshold) # filter data values LOWER than threshold
    dfilt <- c(0,diff(filt))<0      # detect if data amplitude is DECREASING (1) or not (0)
  }
  ddfilt <- c(diff(dfilt),0)<0      # detect if there is a transition between rising and decreasing
  peakIdx <- which(ddfilt>0)        # extract peak indexes
  return(peakIdx)
}

df <- read.xlsx("Dados/Dados1.xlsx", sheet = 1, skipEmptyRows = FALSE, rows = c(1:600), cols = c(1,2))

pos_threshold <- 0.1
neg_threshold <- -0.1

pos_peaks <- findpeaks(df$EMG1, minpeakheight=pos_threshold)
pos_idxpeaks <- pos_peaks[, 2]
df$positive_peaks <- NA
df$positive_peaks[pos_idxpeaks] <- df$EMG1[pos_idxpeaks]

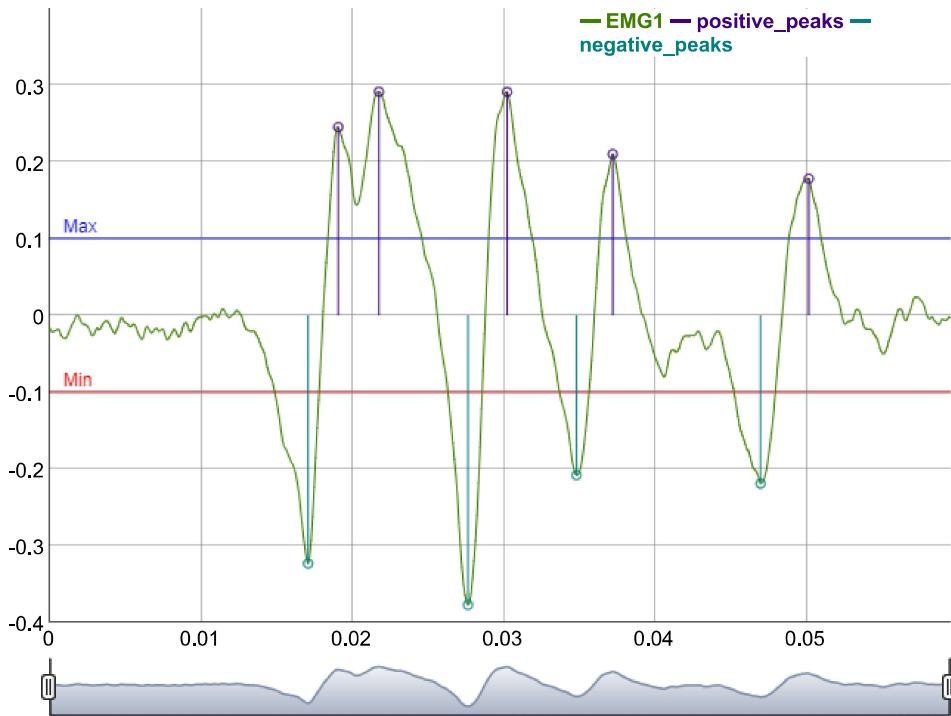
neg_idxpeaks <- originalFindPeaks(df$EMG1, threshold=neg_threshold, direction='min')
df$negative_peaks <- NA
df$negative_peaks[neg_idxpeaks] <- df$EMG1[neg_idxpeaks]

dygraph(df, group="EMG11")%>%
  dySeries("positive_peaks", stemPlot=TRUE)%>%
  
```

```

dySeries("negative_peaks",stemPlot=TRUE)%>%
dyAxis("y", valueRange = c(-0.4, 0.4)) %>%
dyLimit(pos_threshold, "Max", strokePattern = "solid", color = "blue")%>%
dyLimit(neg_threshold, "Min", strokePattern = "solid", color = "red")%>%
dyRangeSelector()

```



## Exercício 4

Gere um gráfico qualquer que represente um evento discreto ao longo do tempo.

Os eventos discretos ocorrerão em tempos discretos. O material de aula ilustra-os como picos em um sinal acima de um limiar de detecção, mas estes também podem ser ilustrados por descontinuidades na frequência de um sinal.

Inicialmente constrói um sinal para representar o ruído:

```

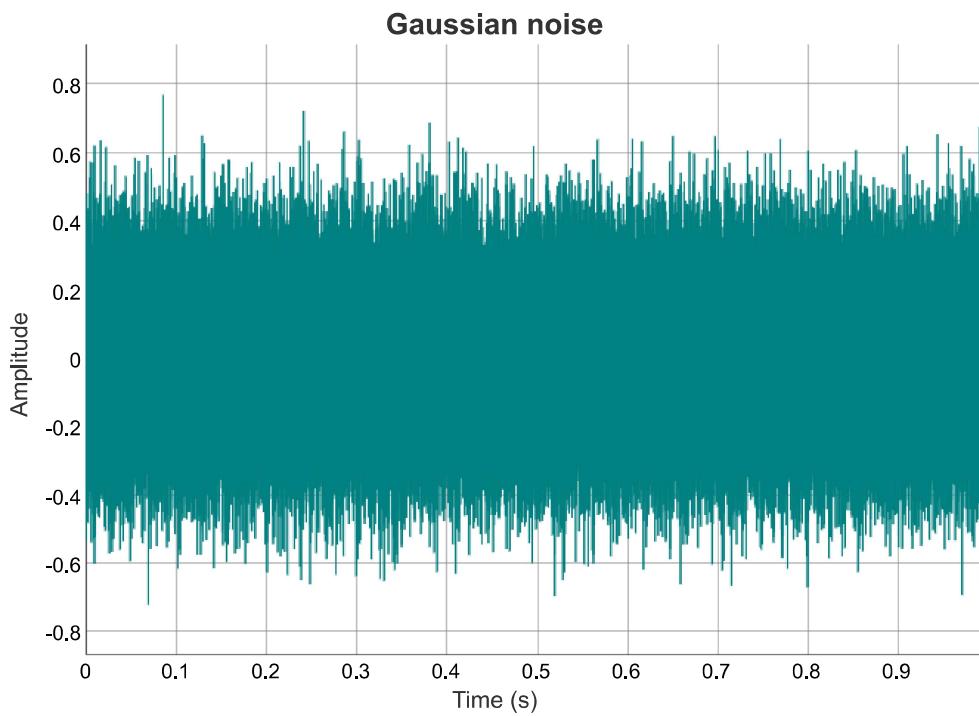
library(dygraphs)

t_end = 1
dt = 1e-5
t <- seq(from = 0, to = t_end, by = dt)

set.seed(42)
samples <- length(t)
snr <- 15
noise <- rnorm(n=samples, mean=0, sd=sqrt(10^(-snr/10)))

data.frame(time=t,noise=noise) %>%
  dygraph(xlab = "Time (s)",
         ylab = "Amplitude",
         main = "Gaussian noise",
         group = "noise")

```

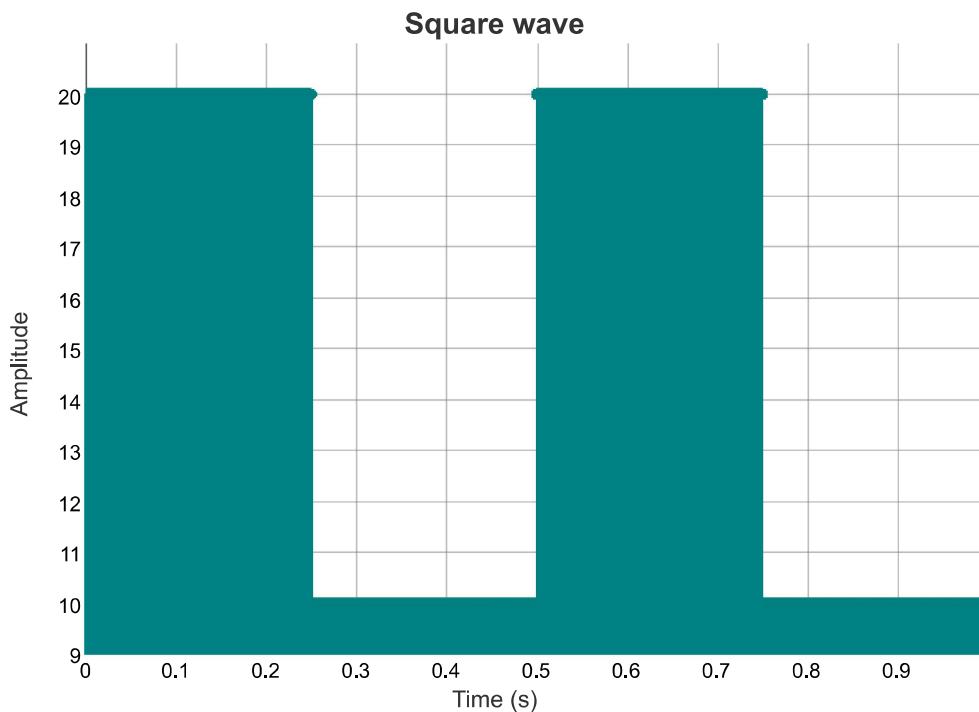


Em seguida será feito um trem de pulsos que será responsável por modular a frequência do sinal.

```

freq <- 2
squareWave <- 10+10*((t%%(1/freq) < 1/(2*freq)))

data.frame(time=t,wave=squareWave) %>%
  dygraph(xlab = "Time (s)",
         ylab = "Amplitude",
         main = "Square wave",
         group = "wave") %>%
  dySeries(stemPlot=TRUE)
  
```



Portanto, para testar os efeitos, será construído um sinal senoidal, mas que não possuirá frequência constante, mas sim uma frequência modulada pela forma de onda anterior.

```

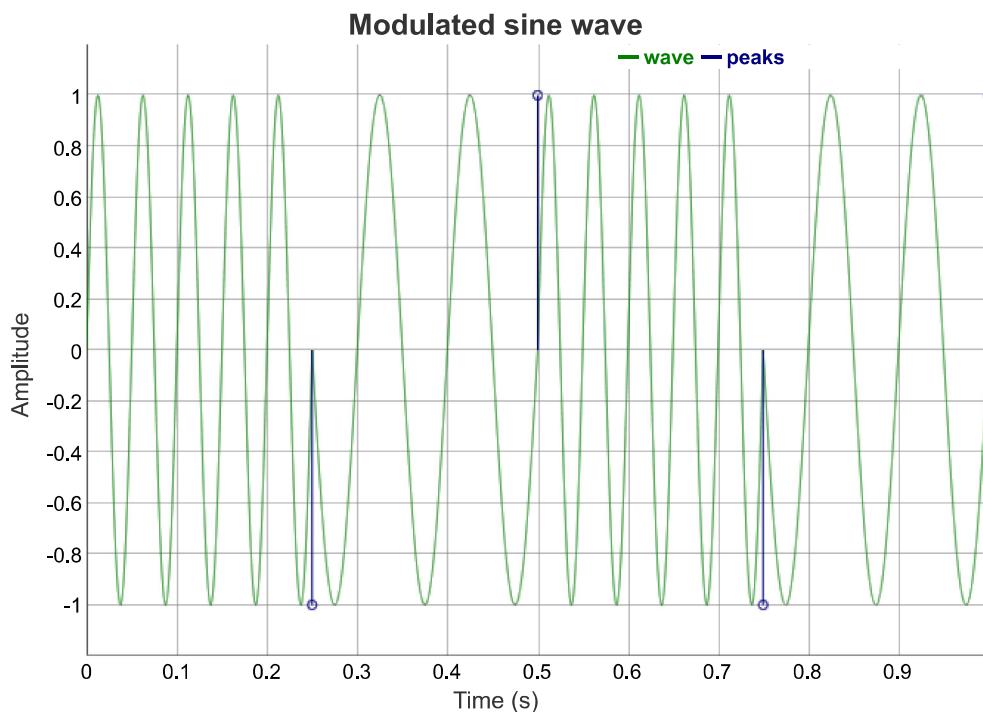
sineWave <- sin(2*pi*squareWave*t)
peaks <- c(0,diff(squareWave)/10)
peaks[peaks==0] = NA

data.frame(time=t,wave=sineWave, peaks=peaks) %>%
  dygraph(xlab = "Time (s)",
         ylab = "Amplitude",
         main = "Sine wave modulated by square wave")
  
```

```

main = "Modulated sine wave",
group = "final") %>%
dySeries("peaks",stemPlot=TRUE)

```



Da forma de onda senoidal observa-se os eventos discretos da mudança de frequência. Estes podem ser monitorados por um rastreador de fase. Mas para termos de ilustração, foi utilizada a própria onda quadrada para indicar o aumento ou redução na frequência, sendo estes os indicadores dos eventos discretos.

## Exercício 5

O erro de quantização pode ser simulado utilizando funções de arredondamento. Para saber mais sobre essas funções, digite na linha de comando do R o comando: `help("ceiling")`. Faça um programa no R que execute os seguintes passos:

- Carregue o arquivo de dados coletados (em sala de aula) com sensores iniciais;
- Selecione uma variável e aplique sobre ela a função `ceiling`;
- Gere um gráfico que mostre a forma de onda da variável selecionada (`x1`), do resultado da função `ceiling` (`x2`), e da diferença entre as duas variáveis (`y = x1-x2`).

O sinal foi normalizado e escalonado para que o efeito do `ceiling` fique mais destacado, mostrando assim um envelope no entorno da onda.

```

# step 1

library(openxlsx)
library(dygraphs)

df <- read.xlsx("Dados/Dados1.xlsx", sheet = 1, skipEmptyRows = FALSE, rows = c(1:100)) # read data from .xlsx

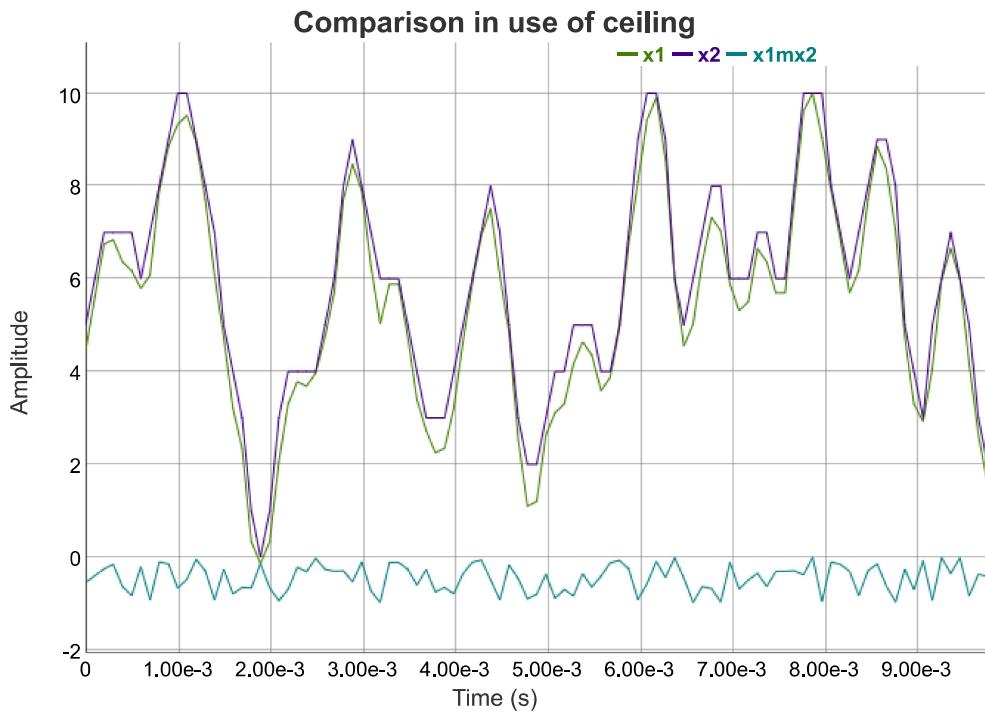
# step 2

used <- 10*df$EMG1/min(df$EMG1)
cld <- ceiling(used)
diffrcn <- used-cld

# step 3

data.frame(time=df$Time,x1=used,x2=cld,x1mx2=diffrcn) %>%
  dygraph(xlab = "Time (s)",
         ylab = "Amplitude",
         main = "Comparison in use of ceiling",
         group = "ceiling")

```

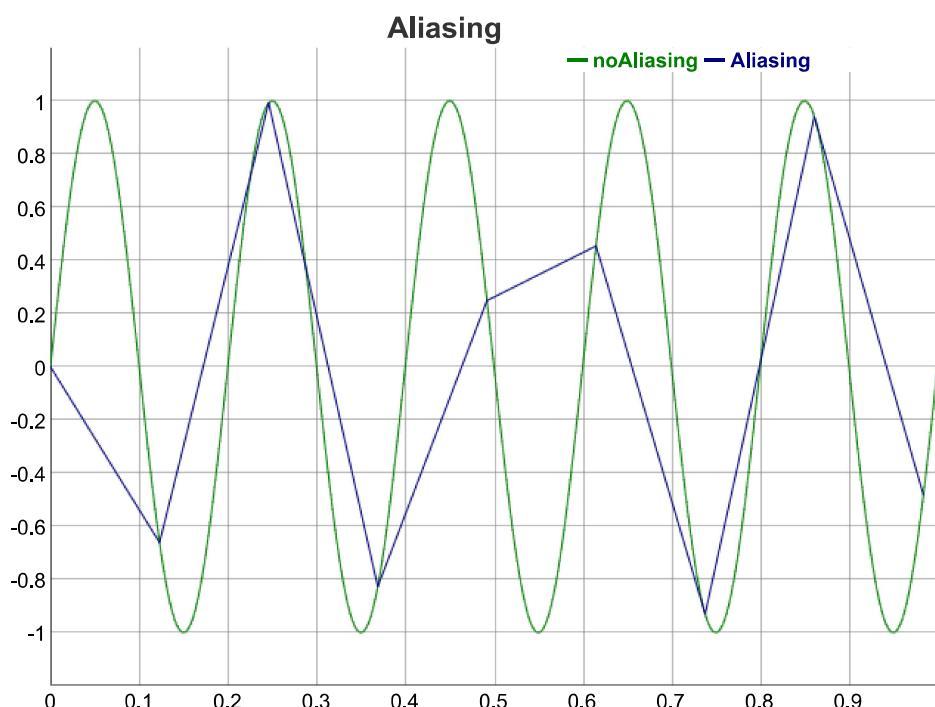


## Exercício 6

Implemente no R o exemplo que ilustra o conceito de “aliasing” abordado no vídeo “Aliasing: an introduction”.

Será gerada uma senoide a uma baixa taxa e em seguida esta será amostrada a uma taxa menor, resultando em uma senoide de diferente frequência da senoide original e no caso abaixo nem se assemelhando a uma senoide.

```
library(dygraphs)
t_end <- 1
dt <- 1/1000
t <- seq(0, 1, by=dt)
t_al <- seq(0, 1, by=dt*123)
x <- sin(2*pi*5*t)
y <- NA
y[t %in% t_al] <- x[t %in% t_al]
df <- data.frame(t=t, noAliasing=x, Aliasing=y)
dygraph(df, main = "Aliasing") %>% dyOptions(connectSeparatedPoints=TRUE, pointSize = 1)
```



## Exercício 7

Exemplifique por meio de um programa no R como podemos realizar a reamostragem de sinais por meio da interpolação.

A reamostragem por interpolação visa inserir amostras no sinal de interesse e interpolar os pontos das amostras existentes passando pelos novos pontos, para que estes possuam um valor que mantenha a continuidade da função. Vide exemplo a abaixo:

```
library(dygraphs)
t_end <- 1
dt_high <- 0.1
dt_low <- 0.01
t <- seq(0, t_end, by=dt_high)
t_aim <- seq(0, t_end, by=dt_low)
data <- sin(2*pi*t)
df <- data.frame(t, data)

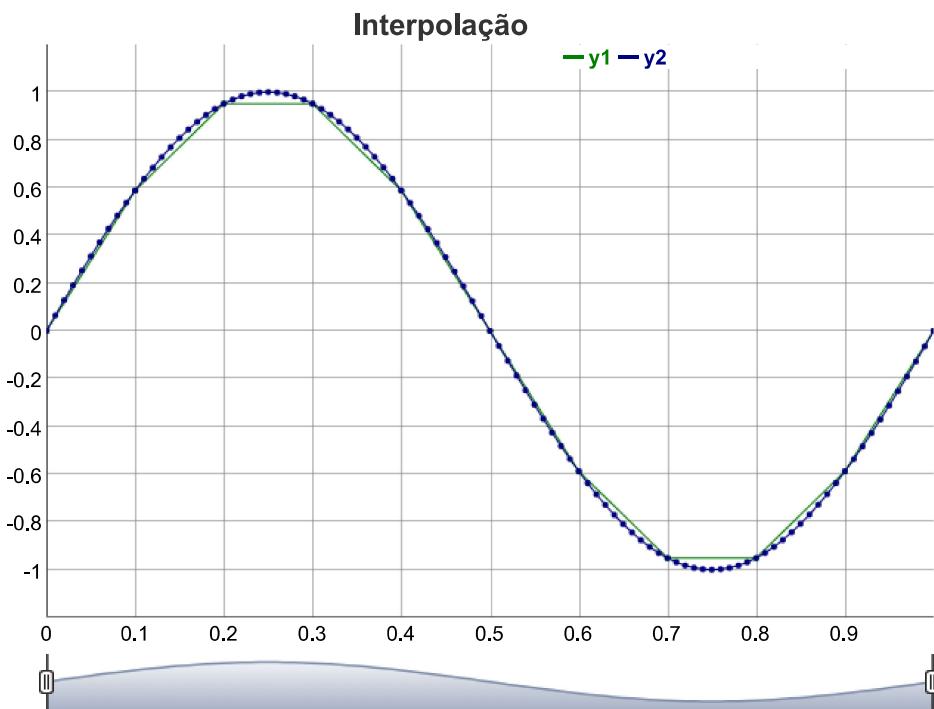
df_inter <- data.frame(spline(x=t, y=data, xout=t_aim))

new_data <- df_inter$y
new_t <- df_inter$x
tt <- union(t, new_t)
tt <- sort(tt)

df <- data.frame(time=tt)
df$y1 <- NA
df$y2 <- NA

df$y1[tt %in% t] <- data
df$y2[tt %in% new_t] <- new_data

dygraph(df, main = "Interpolação") %>%
  dyRangeSelector() %>%
  dyOptions(drawPoints = TRUE, connectSeparatedPoints=TRUE, pointSize = 2)
```



A partir do gráfico observa-se uma fina interpolação que permite que a senoide possua mais detalhes.

## Exercício 8

Qual a finalidade da função “union” utilizada em alguns exemplos deste tutorial? Exemplifique o uso da mesma.

A função **union** realiza a concatenação/união de arrays, considerando que cada um foi gerado individualmente e que o processamento final exige que eles estejam juntos. Vide um exemplo simples abaixo:

```
a <- c(1,2,3)
b <- c(4,5,6)
```

```
dplyr::union(a,b)
```

```
## [1] 1 2 3 4 5 6
```

Realizando a união de *a* e *b*.

## Exercício 9

Dado o trecho de código abaixo:

```
library("reshape2")
```

```
## Warning: package 'reshape2' was built under R version 4.1.1
```

```
library(ggplot2)
library(gganimate)
```

```
## Warning: package 'gganimate' was built under R version 4.1.1
```

```
library(dplyr)
```

```
## 
## Attaching package: 'dplyr'
```

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

```
##
```

```
##     filter, lag
```

```
## The following objects are masked from 'package:base':
```

```
##
```

```
##     intersect, setdiff, setequal, union
```

```
# Geracao dos dados -----
```

```
# sinal original
```

```
fs <- 100 # frequencia de amostragem em Hz
dt <- 1/fs # resolucao temporal em segundos
tf <- 5 # duracao em segundos
t <- seq(from = 0, to = tf, by = dt)# vetor temporal em segundos
f <- 1 # frequencia de oscilacao da senoide
y <- sin(2*pi*f*t) # senoide gerada
```

```
# parametros para subamostragem
```

```
fs1 <- 30 # reduzindo a taxa de amostragem
dt1 <- 1/fs1 # resolução temporal em segundos
t1 <- seq(from = 0, to = tf, by = dt1)# vetor temporal em segundos
y2 <- sin(2*pi*f*t1) # senoide gerada
```

```
# unindo vetores de tempo (repetições são excluídas, de forma que apenas um valor é mantido)
# e ordenando o vetor de tempo resultante
```

```
tt <- union(t, t1) # Left join dos vetores de tempo
tt <- sort(tt)# ordenacao do vetor tempo
```

```
v1 <- rep(NA, length(tt)) # criar vetor de tamanho apropriado com valores NAN
v2 <- rep(NA, length(tt)) # criar vetor de tamanho apropriado com valores NAN
```

```
v1[ (tt %in% t) == TRUE ] <- y # preenchendo sinal original em instantes adequados ao vetor tempo original
v2[ (tt %in% t1) == TRUE ] <- y2 # preenchendo sinal subamostrado em instantes adequados ao vetor tempo subamostrado
```

```
dfplot <- data.frame(t = tt, v1, v2) # criacao de data frame com os dados gerados
```

```
dfplotm <- melt(dfplot, id="t") #(para plotar séries multiplas de forma mais simples)
```

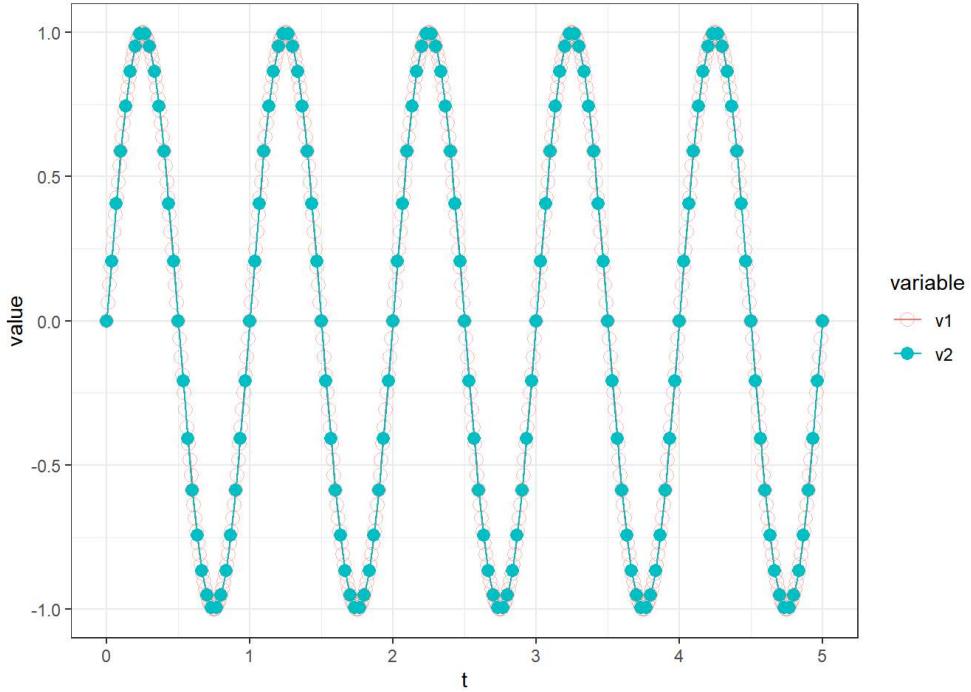
```
# Grafico estatico -----
```

```
# criacao de gráfico estatico com o ggplot2 para apresentacao dos resultados
theme_set(theme_bw()) # tema do ggplot2 para theme_bw()
```

```

p <- ggplot(dfplotm, # utiliza o data frame gerado anteriormente
             aes(x=t, y=value, colour = variable)) + # configura series temporais a serem usadas
geom_line(data=dfplotm[!is.na(dfplotm$value),]) + # plot com Linha geometricas quando valores diferentes de NAN
geom_point(data=dfplotm[!is.na(dfplotm$value),], # plot com pontos quando valores diferentes de NAN
           aes(shape=variable, size = variable, alpha = variable)) + # configuracoes de estilo para o plot gerado
scale_size_manual(values=c(3.2,2.8)) + # definir tamanho da janela de plot
scale_alpha_manual(values = c(0.5,1)) + # definir transparencia do plot
scale_shape_manual(values = c(1, 16)) # definir formato do plot
print(p) # plottar

```



Considere:

- Estude e explique cada uma das linhas de código.
- Qual conceito está ilustrado na animação?
- Qual cor de linha representa um sinal contínuo?
- Qual símbolo representa um sinal discreto?
- Qual a resolução temporal dos sinais v1 e v2?
- Qual sinal, v1 ou v2, possui a maior frequência de oscilação?
- Qual a implicação prática do sinal v1 ter sido amostrado a uma taxa de frequência maior do que a do sinal v2?
- Os sinais foram amostrados de acordo com a frequência de Nyquist?
- As amostras dos sinais v1 e v2 acontecem em um mesmo tempo discreto? Justifique.
- Qual o valor de pico a pico dos sinais v1 e v2?
- Qual o valor do pico e do vale dos sinais v1 e v2?
- Como poderíamos estimar um amostra em um tempo discreto inicialmente não existente no sinal v2?

Resposta:

- O conceito mostrado na animação é o de subamostragem;
- A cor de linha verde representa o sinal contínuo (v1) e a cor azul representa o sinal amostrado, portanto discreto e com subamostragem (v2);
- A resolução temporal do sinal contínuo é de 1/100 s e o do sinal amostrado é de 1/30 s;
- Ambos os sinais possuem a mesma frequência de oscilação;
- Um sinal com frequência maior de amostragem possui melhor qualidade de representação do sinal real, porém ocupa mais espaço para armazenamento;
- Os sinais foram amostrados de acordo com o teorema da amostragem de Nyquist, pois estes não indicam sinal de Aliasing;
- Uma vez que os sinais v1 e v2 estão armazenados em hardware, suas amostras ocorrem em tempo discreto, porém como v1 possui uma taxa de amostragem maior que v2, o intervalo que suas amostras acontecem é menor;
- O valor pico-a-pico dos sinais v1 e v2 é 2;
- O pico/vale dos sinais v1 e v2 é 1/-1;
- A interpolação pode ser usada para estimar uma amostra inexistente do sinal v2.