Práctica I: Introducción a R y Conversión A/D y D/A

Señales y Sistemas, Grado en Ciencia de Datos - UV

Sandra Paniagua Sánchez y Gema Bravo Aguilera

2022-10-17

Contents

1. Introducción a R

1.1. Pre-laboratorio

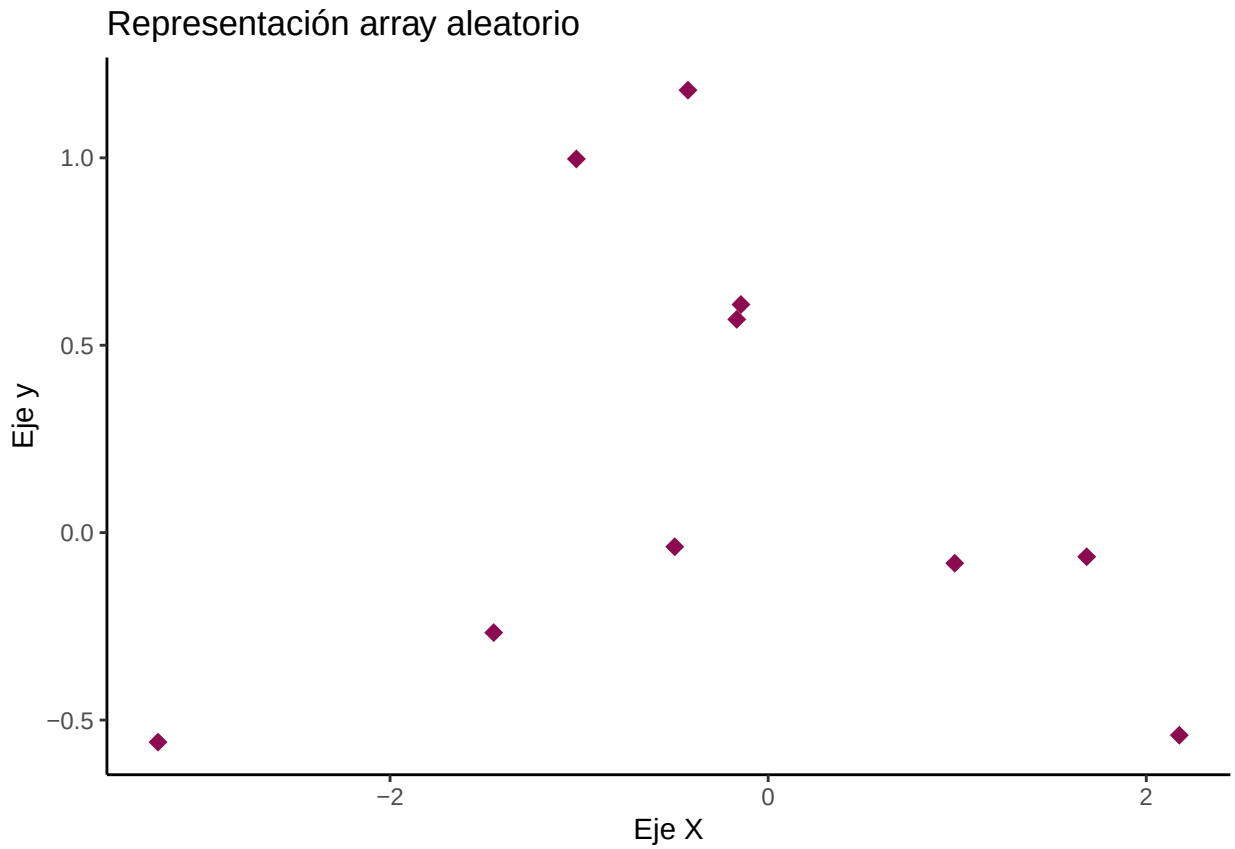
Las practicas se realizarán con R-Studio. Se recomienda al alumno leer los siguientes documentos, así como tenerlos disponibles durante la realización de las prácticas:

1.2. Ejercicios Prelaboratorio

1. Crear una array aleatorio (según una distribución normal) de 10 puntos en x, otro de 10 puntos en y, y represéntalos un plot. Utilizar la función rnorm.

```r
x <- array(rnorm(10))
y <- array(rnorm(10))

ggplot() + geom_point(aes(x, y), colour = "deeppink4", size = 3, shape = 18) +
  xlab("Eje X")+ # eje x
  ylab("Eje y")+ # eje y
  ggtitle("Representación array aleatorio")+ #título del gráfico
  theme_classic()
```



2. Calcular la media de la distribución anterior en x y en y.

```r
media_x <- mean(x)
media_y <- mean(y)

dimen <- c(media_x, media_y)
dimensiones <- matrix(dimen, ncol = 2)
colnames(dimensiones) <- c ('media x', 'media y')
dimensiones %>% kable() %>% kable_styling(full_width = F, position = "left")
```

media x	media y
-0.2075045	0.1805935

3. Realiza un script que realice las dos tareas anteriores. Llama a este script HelloRWorld.R. El scrip debe iniciarse imprimiendo el texto “Hello R World”.

```
palabra <- readline(prompt="Ingrese la palabra: ")
```

```
Ingrese la palabra:
```

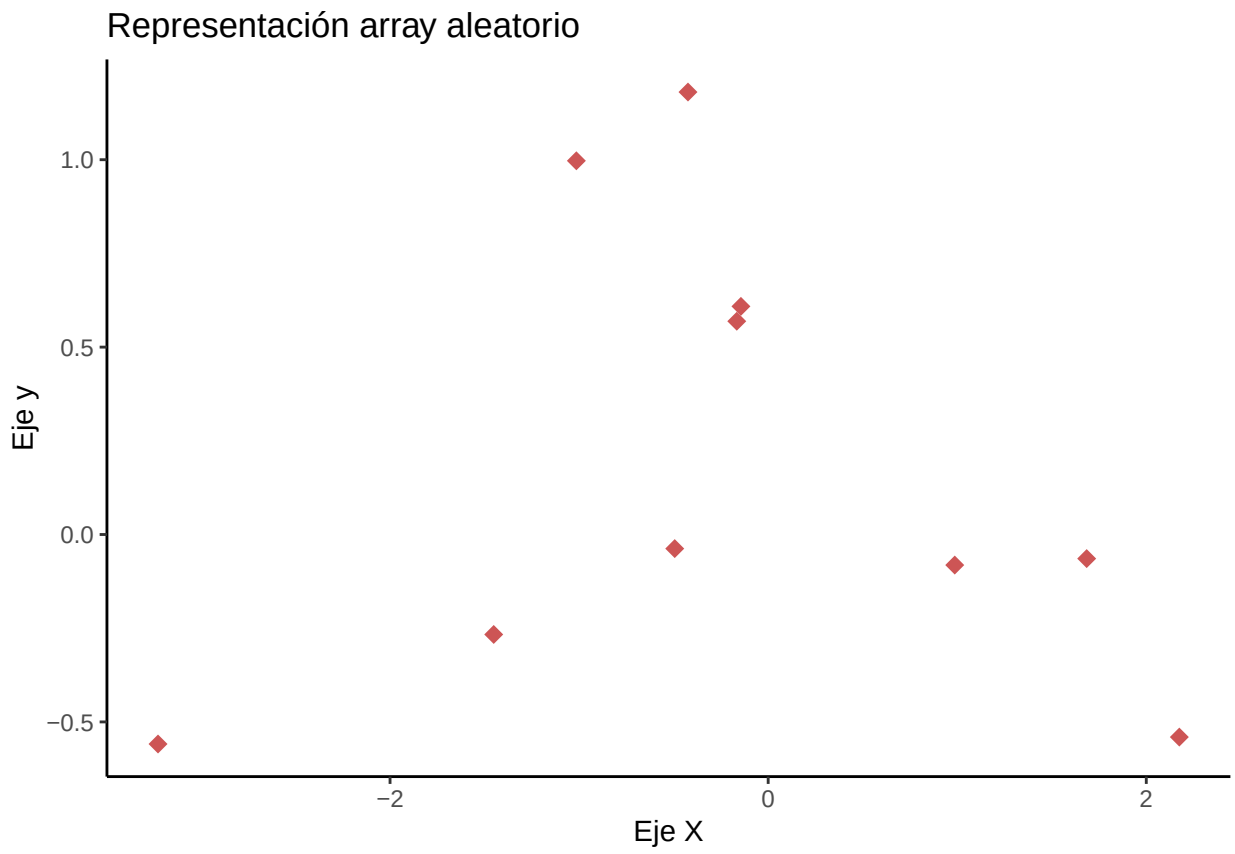
```
if (palabra == "Hello R World") {
  source("HelloRWorld.R")
}
```

4. Combina los dos arrays (x e y) en una sola matriz, m. Puedes utilizar la función rbind.

```
m <- rbind(x,y)
```

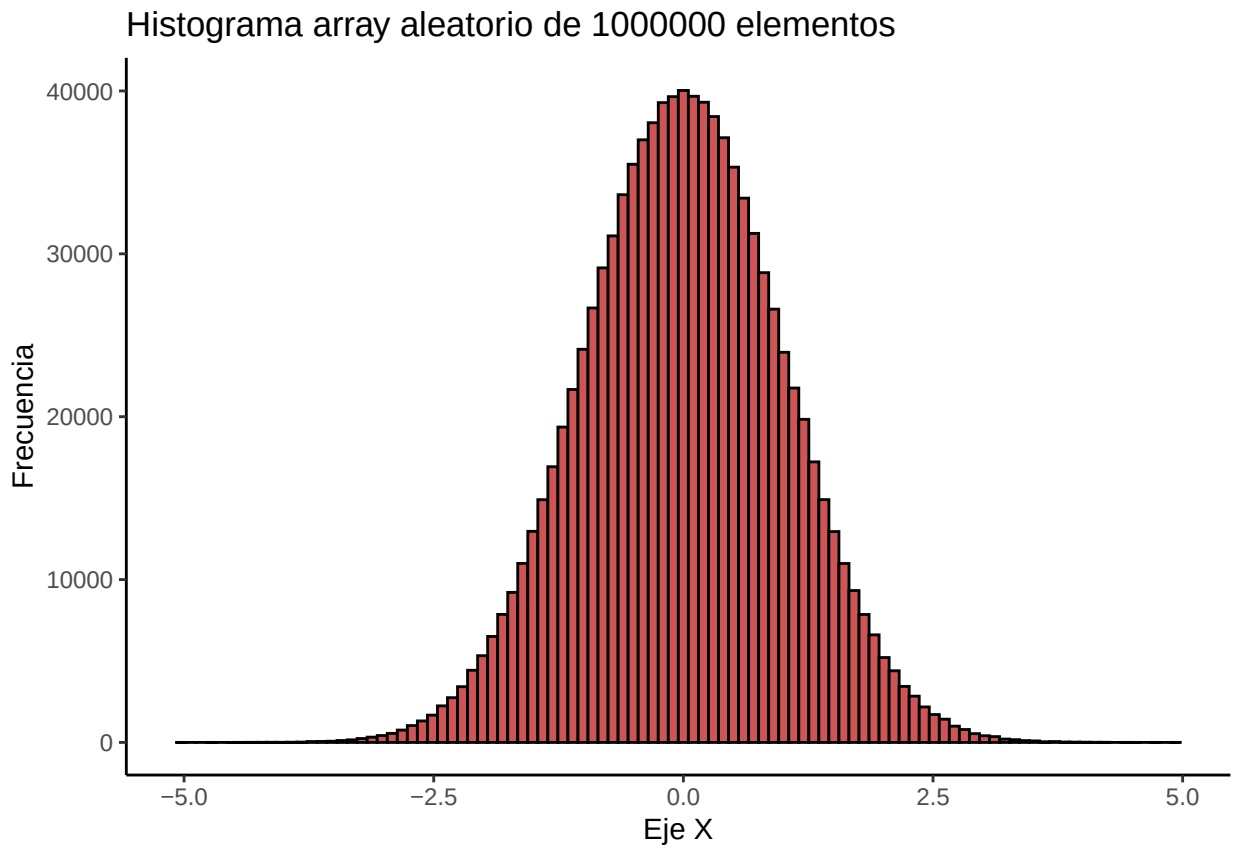
5. Obtén una figura idéntica a la del ejercicio uno desde la matriz m.

```
ggplot() + geom_point(aes(m[1,], m[2,]), colour = "indianred3", size = 3, shape = 18) +
  xlab("Eje X")+ # eje x
  ylab("Eje y")+ # eje y
  ggtitle("Representación array aleatorio")+ #título del gráfico
  theme_classic()
```



6. Obtén un array aleatorio de 1000000 de elementos y representa su histograma. Amplia el número de bines a 100.

```
ggplot() + geom_histogram(aes(array(rnorm(1000000))), bins = 100, fill = "indianred3", colour = "black")
  xlab("Eje X")+ # eje x
  ylab("Frecuencia")+ # eje y
  ggtitle("Histograma array aleatorio de 1000000 elementos")+ #título del gráfico
  theme_classic()
```



7. Genera un array de 10 elementos, que comience en uno y se incremente en cinco. Utilizar la función seq. Para obtener ayuda de la función puedes utilizar help(seq)

```
array(seq(1,by = 5, length.out = 10))
```

```
 [1]  1  6 11 16 21 26 31 36 41 46
```

2. Conversión A/D: Muestreo

2.1. Pre-laboratorio

2.1.1 Generación y muestreo de Señales

Genera dos sinusoides de 250 y 500 Hz respectivamente. La frecuencia de muestreo es de 1 kHz (periodo de muestreo 1/Fs = 1 ms). La duración de las señales debe ser de 10 ms (debes calcular el número de muestras al que equivalen esos 10 ms para la frecuencia de muestreo dada). 1. Representa las dos señales generadas en

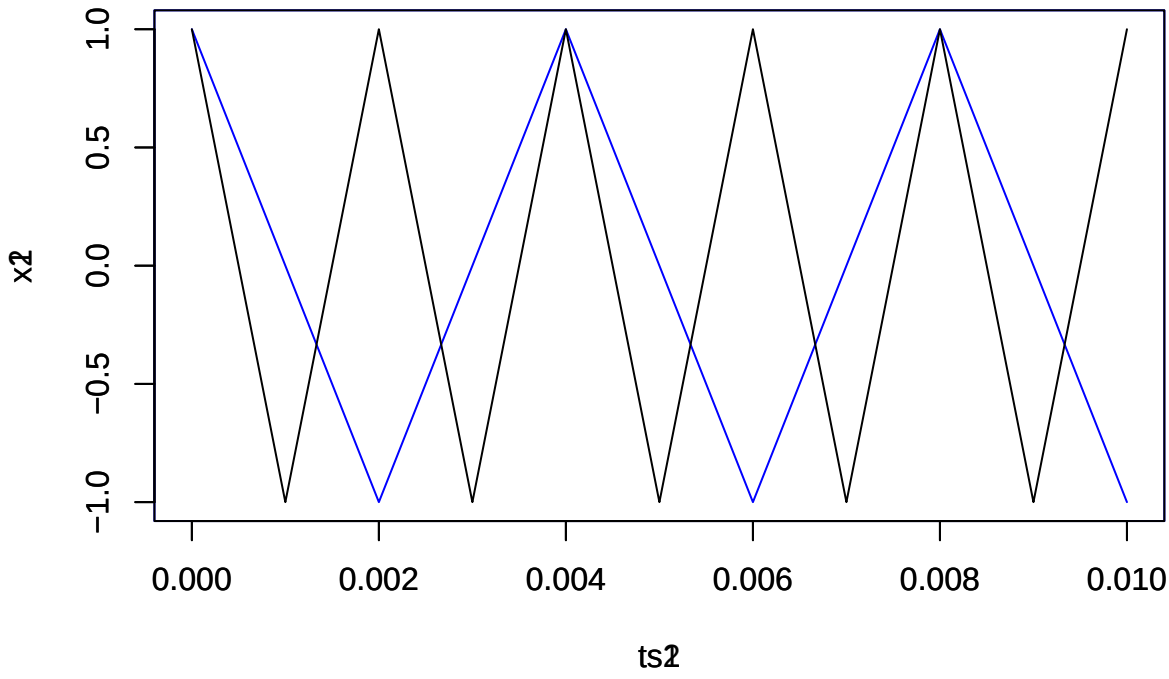
una misma figura utilizando plot y par(new=T). 2. Vuelve a representar las señales pero esta vez poniendo el eje X en segundos.

Ayuda: puedes ver cómo se hace algo muy parecido en el programa que aparece en la sección 2.1.3.

Calculamos el Periodo de muestreo:

$Periodo\ de\ muestreo = \frac{1}{Fs} = \frac{1}{1000} = 0.001$

```
rm(list=ls())
f1 = 250; # Frecuencia de la primera sinusoide (Hz)
f2 = 500; # Frecuencia de la segunda sinusoide (Hz)
T1 = 0.001; # Periodo de muestreo 1 (s)
T2 = 0.001; # Periodo de muestreo 2 (s)
t = 0.01; # Tiempo total de muestreo (s)
ts1= seq(0,t,T1);
ts2= seq(0,t,T2);
x1 = cos(2*pi*f1*ts1); # nos piden sinusoides
x2 = cos(2*pi*f2*ts2);
par(col="blue")
plot(ts1,x1,type = "l")
par(new=T)
par(col="black")
plot(ts2,x2,type = "l")
```



```
####### HACER CON GGPLOT

#Vuelve a representar las señales pero esta vez poniendo el eje X en segundos
```

2.1.2. Conversión D/A : Reconstructores

Para la preparación del tercer apartado de esta práctica vamos a programar un script, llamado genera sinusoides.R, que para una frecuencia f, una frecuencia de muestreo Fm y una tiempo de muestreo t nos genere dos sinusoides, una (xm) con dicha frecuencia de muestreo, que será nuestra señal muestreada a reconstruir, y otra (xc), muestreada a $10 \cdot \text{Fm}$, que hará las veces de señal continua aproximada.

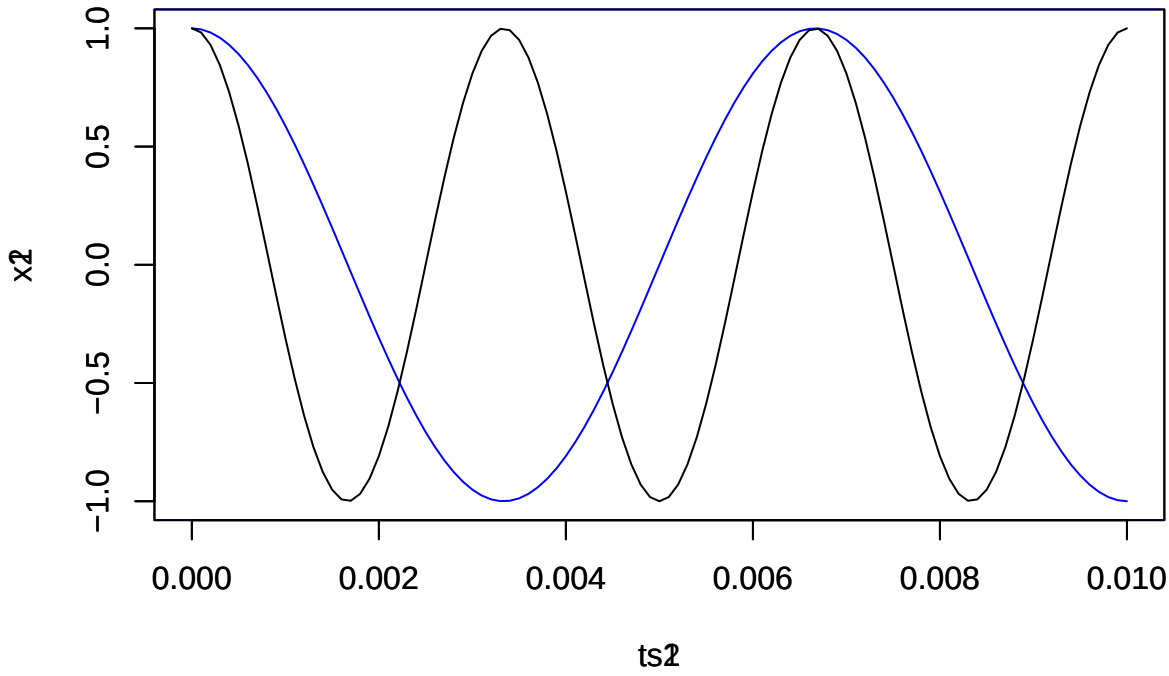
Estas son las lineas generales del programa:

```
# 1. Introducir la frecuencia de muestreo deseada (Modificar el script con el valor deseado)
# 2. Introducir la frecuencia de la senyal a generar
# 3. Introducir Fc como 10 * Fm
# 4. Generar xm mediante la funcion cos y los datos f, Fm y t
# 5. Generar xc mediante cos y los datos f, 10 * Fm y t
```

2.1.3. Conversión A/D: Muestreo

En primer lugar hay que tener en cuenta que R trabaja con datos discretos; puede parecer un poco ilógico usar esta herramienta para explicar una práctica de muestreo (¡los datos ya están muestreados!). La solución adoptada es considerar intervalos de tiempo muy pequeños frente al periodo de muestreo de tal forma que tendremos una señal "cuasi-continua". El siguiente programa implementa el muestreo de dos señales analógicas, sinusoides, de frecuencias f1 y f2 muestreadas con periodos de muestreo T1 y T2 hasta un tiempo T.

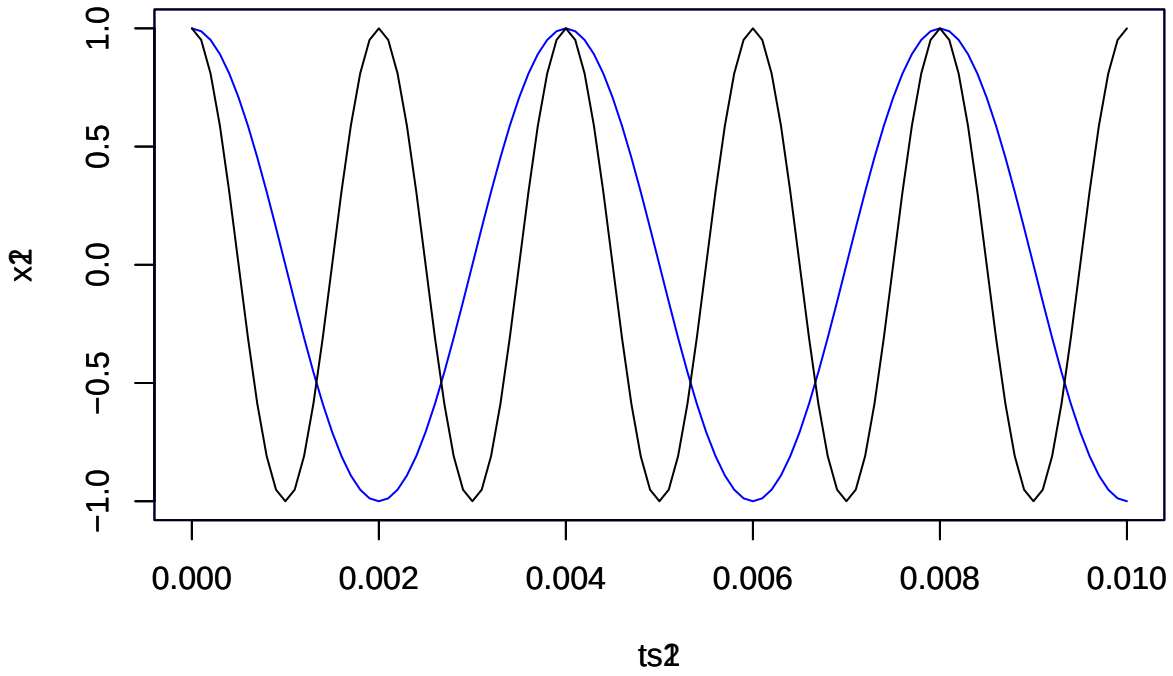
```
rm(list=ls())
f1 = 150; # Frecuencia de la primera sinusoide (Hz)
f2 = 300; # Frecuencia de la segunda sinusoide (Hz)
T1 = 0.0001; # Periodo de muestreo 1 (s)
T2 = 0.0001; # Periodo de muestreo 2 (s)
t = 0.01; # Tiempo total de muestreo (s)
ts1= seq(0,t,T1);
ts2= seq(0,t,T2);
x1 = cos(2*pi*f1*ts1);
x2 = cos(2*pi*f2*ts2);
par(col="blue")
plot(ts1,x1,type = "l")
par(new=T)
par(col="black")
plot(ts2,x2,type = "l")
```



Comprueba las siguientes combinaciones:

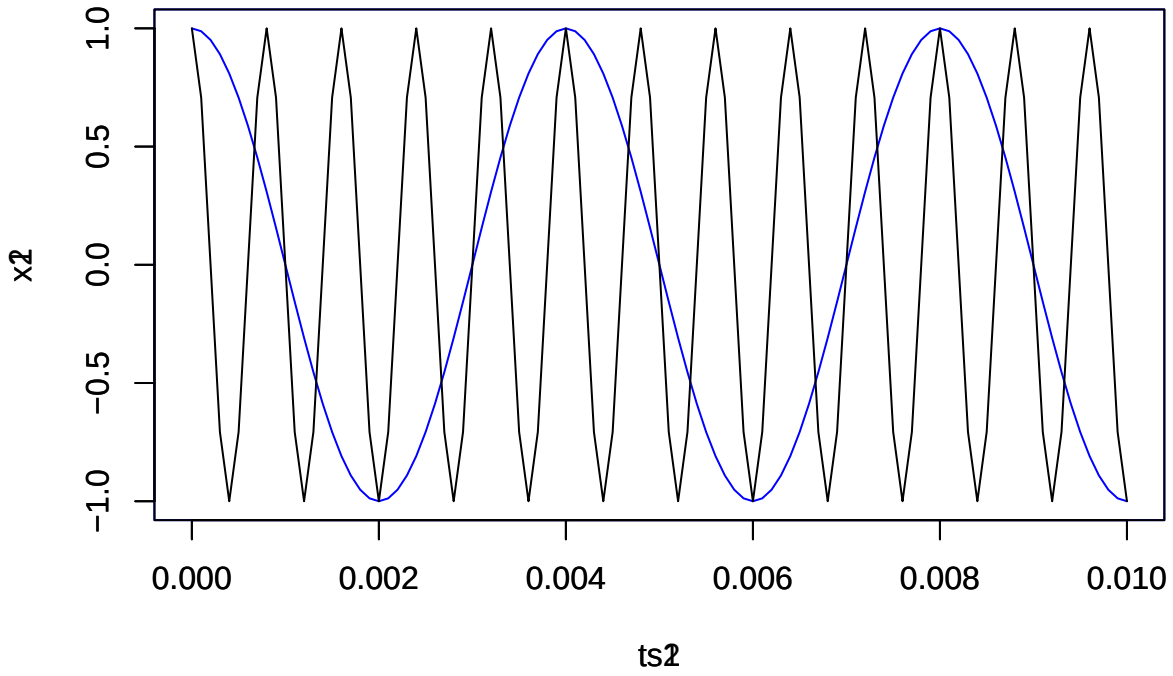
a) f1 = 250 Hz, f2 = 500 Hz

```
rm(list=ls())
f1 = 250; # Frecuencia de la primera sinusoide (Hz)
f2 = 500; # Frecuencia de la segunda sinusoide (Hz)
T1 = 0.0001; # Periodo de muestreo 1 (s)
T2 = 0.0001; # Periodo de muestreo 2 (s)
t = 0.01; # Tiempo total de muestreo (s)
ts1= seq(0,t,T1);
ts2= seq(0,t,T2);
x1 = cos(2*pi*f1*ts1);
x2 = cos(2*pi*f2*ts2);
par(col="blue")
plot(ts1,x1,type = "l")
par(new=T)
par(col="black")
plot(ts2,x2,type = "l")
```



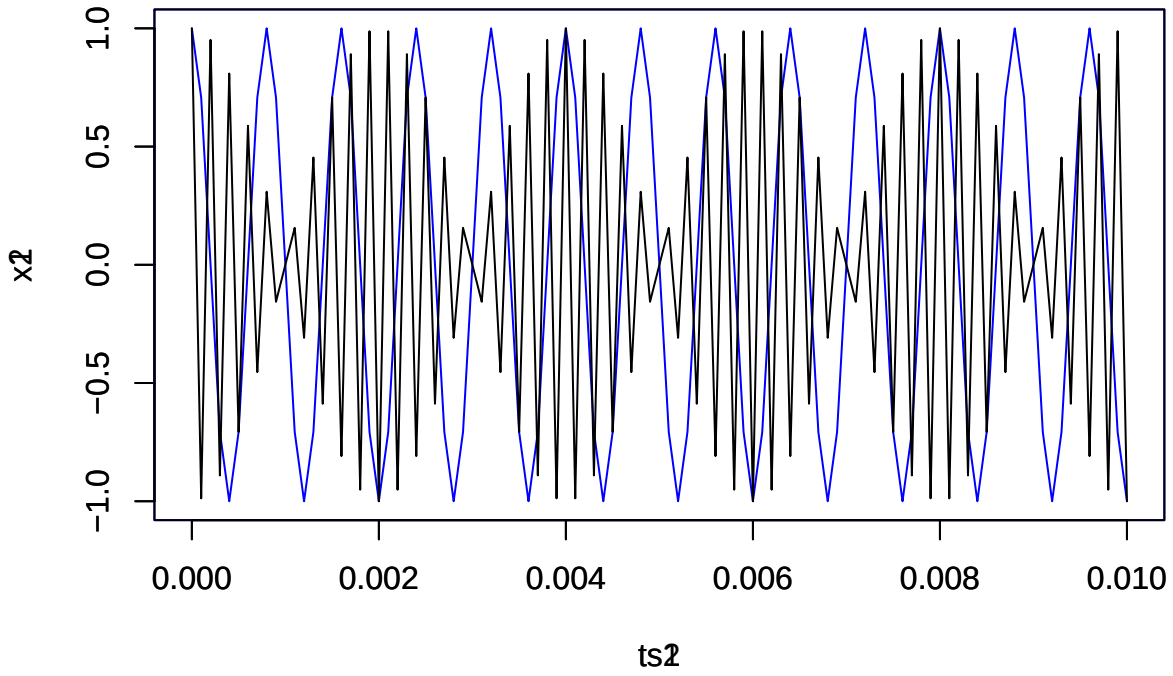
b) f1 = 250 Hz, f2 = 1250 Hz

```
rm(list=ls())
f1 = 250; # Frecuencia de la primera sinusoide (Hz)
f2 = 1250; # Frecuencia de la segunda sinusoide (Hz)
T1 = 0.0001; # Periodo de muestreo 1 (s)
T2 = 0.0001; # Periodo de muestreo 2 (s)
t = 0.01; # Tiempo total de muestreo (s)
ts1= seq(0,t,T1);
ts2= seq(0,t,T2);
x1 = cos(2*pi*f1*ts1);
x2 = cos(2*pi*f2*ts2);
par(col="blue")
plot(ts1,x1,type = "l")
par(new=T)
par(col="black")
plot(ts2,x2,type = "l")
```



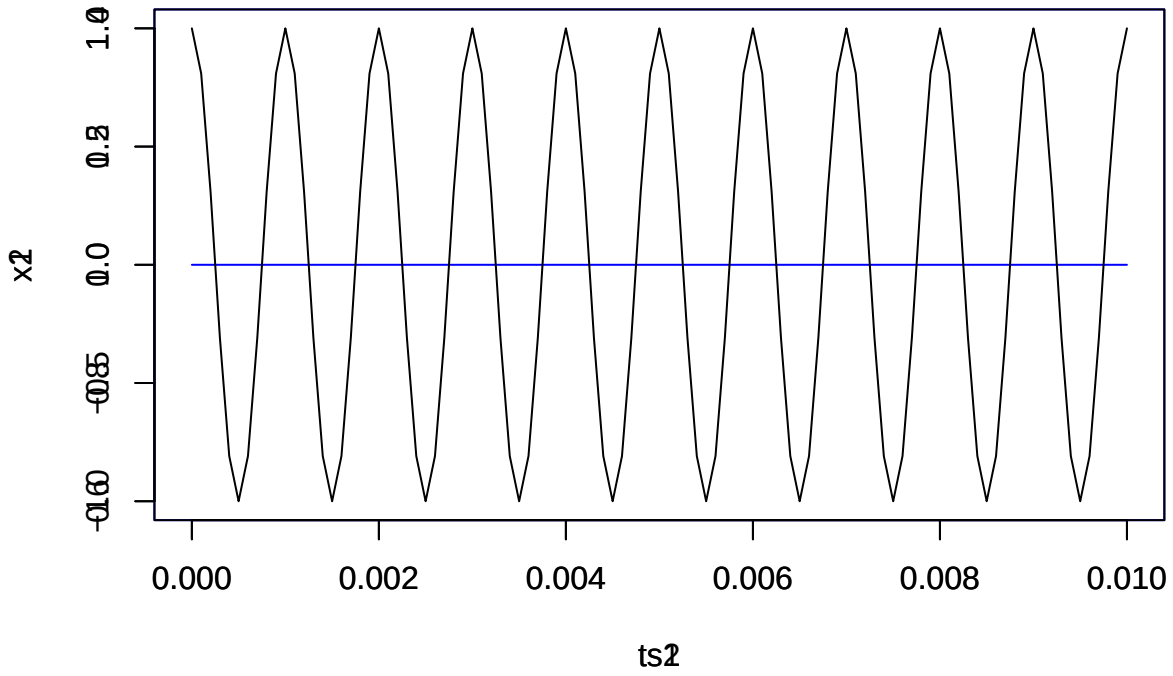
c) f1 = 1250 Hz, f2 = 5250 Hz

```
rm(list=ls())
f1 = 1250; # Frecuencia de la primera sinusoide (Hz)
f2 = 5250; # Frecuencia de la segunda sinusoide (Hz)
T1 = 0.0001; # Periodo de muestreo 1 (s)
T2 = 0.0001; # Periodo de muestreo 2 (s)
t = 0.01; # Tiempo total de muestreo (s)
ts1= seq(0,t,T1);
ts2= seq(0,t,T2);
x1 = cos(2*pi*f1*ts1);
x2 = cos(2*pi*f2*ts2);
par(col="blue")
plot(ts1,x1,type = "l")
par(new=T)
par(col="black")
plot(ts2,x2,type = "l")
```



d) f1 = 0 Hz, f2 = 1000 Hz

```
rm(list=ls())
f1 = 0; # Frecuencia de la primera sinusoide (Hz)
f2 = 1000; # Frecuencia de la segunda sinusoide (Hz)
T1 = 0.0001; # Periodo de muestreo 1 (s)
T2 = 0.0001; # Periodo de muestreo 2 (s)
t = 0.01; # Tiempo total de muestreo (s)
ts1= seq(0,t,T1);
ts2= seq(0,t,T2);
x1 = cos(2*pi*f1*ts1);
x2 = cos(2*pi*f2*ts2);
par(col="blue")
plot(ts1,x1,type = "l")
par(new=T)
par(col="black")
plot(ts2,x2,type = "l")
```



¿Qué conclusiones obtienes?

La formulación más conocida del teorema de muestreo de Nyquist - Shannon es que para poder reconstruir una señal muestreada, la frecuencia de muestreo debe ser superior al doble del ancho de banda.

2.2. Laboratorio Conversión A/D: Muestreo

2.2.1. Aplicación práctica 1

Se dispone de dos ficheros de audio, ce44100.wav y ce8000.wav muestreados a frecuencias 44.1 kHz y 8 kHz respectivamente.

1. Primero instala el paquete tuneR install.packages("tuneR")
2. A continuación el paquete sound install.packages("sound")
3. Carga los paquetes en R con la funcion library(tuneR)

```
#install.packages("tuneR")
#install.packages("sound")
#install.packages("seewave")

library(tuneR)
library(sound)
library(seewave)
```

4. Lee el fichero ce44100.wav con R utilizando la función readWave, cuyo formato es: y = readWave("file.wav").

```
y = readWave("ce44100.wav") #original
```

5. Escúchalo con la función play("file.wav"). Para ello antes deberás de configurar que software de reproducción de audio se va a utilizar. Utiliza los comandos findWavPlayer() y setWavPlayer('aplay'). Aplay es un ejemplo de software disponible en linux.

```
#findWavPlayer()
#setWavPlayer("aplay")
#setWavPlayer("playwave")
#setWavPlayer("mplay32/play")
#setWavPlayer("mplay32")
#play("ce44100.wav")
```

6. Vamos a simular un muestreo de la señal original a 22050 Hz y a 11025 Hz de la siguiente forma: y22050 = downsample(y,22050) y y11025 = downsample(y,11025). Escucha estas dos nuevas señales y comenta los resultados. Para oirlas deberás grabarlas en formato wav y reproducirlas con la función play("file.wav")

```
# y22050 = downsample(y,22050)
# y11025 = downsample(y,11025)
#
# savewav(wave=y22050, f=22050, filename = "y22050.wav", rescale = NULL) #De esta manera el audio se es
#
# savewav(wave=y11025, f=11025, filename = "y11025.wav", rescale = NULL)#Se escucha con menos claridad
#
# play("y22050.wav")
# play("y11025.wav")
```

SOLUCION: En el nuevo archivo de y22050.wav el audio se escucha con mas claridad. Y en el archivo y11025.wav el audio se escucha con menos definició respecto al anterior.

7. Utiliza ahora el fichero ce8000.wav y reproduce su contenido a la frecuencia a la que se ha muestreado, y cambiando ésta por ejemplo a 20 kHz y a 4 kHz (como en el ejemplo anterior). ¿A qué crees que se deben los efectos producidos?

```
v = readWave("ce8000.wav")
v8000 = downsample(v,8000)

#cambiando esta por 20 y 4:
savewav(wave=v8000, f=20000, filename = "v20.wav", rescale = NULL)
#play("v20.wav")

savewav(wave=v8000, f=4000, filename = "v4.wav", rescale = NULL)
#play("v4.wav")
```

Al cambiar la frecuencia a mas baja lo que ocurre es que hay menos definicion.

Esto se debe a que el minimo de la frecuencia de adquisicion tiene que ser mayor del doble de la frecuencia maxima que quieres adquirir. Si elegimos mas frecuencia de adquisicion se representará con mas definición y

por tanto se escuchara con mas claridad el sonido. El periodo de adquisicion, por otra parte, esta relacionado con la frecuencia de adquisision porque es su inversa.

Cuano tengo la señal en el mundo digital y la quiero representar en el mundo real, utilizo los reconstructores. Que tambien tienen una frecuencia. (DIBUJO)

Es decir, la frecuencia de adquision y reconstruccion estan relacionadas con la calidad en la que se escuchará la señal y representarla.

HABRA QUE QUITAR ESTO EXPLICACION CLASE !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! con downsample perdemos los tonos agudos filtro paso bajo

la cuantificación es la altura

2.2.2. Conversión A/D: Cuantificación

En esta parte se analizarán los efectos de la cuantificación en la conversión analógica digital para ello vamos a utilizar la función quanti cuyos argumentos son los siguientes:

```
xq = quanti(data,N,m)
```

siendo data la señal original (un solo canal), N el número de bits (2N niveles), m el rango de entrada (entre +m y -m), y xq la salida cuantificada. La cuantificación se realiza por truncamiento.

1. *Efecto sobre las formas de onda.* Se estudiará el efecto de la cuantificación sobre el aspecto una sinusoide. Para ello genera tres períodos de una sinusoide de amplitud 4, frecuencia 50 Hz y frecuencia de muestreo 1000 Hz y cuantifica ésta suponiendo que el parámetro m = 5. Prueba para valores de 4 a 16 bits. Representa la señal de error de cuantificación en cada caso (señal formada por el error en cada una de las muestras). Prueba también a cuantificar la señal sinusoidal con una frecuencia de muestreo de 100 Hz y represéntala junto con la original.
2. Efecto de la *cuantificación sobre una señal de audio.* Para ello vamos a utilizar el fichero p44100.wav. Lee dicho fichero con la instrucción y = readWave("p44100.wav") y escucha su contenido con play("p44100.wav.

Repite el proceso cuantificando la señal a 8, 4, 3 y 2 bits (utiliza como valor de m el valor máximo en valor absoluto de y y cuantifica cada uno de los canales del audio). ¿A qué se debe el ruido que se escucha? ¿Se podría resolver este problema aumentando la frecuencia de muestreo?

2.2.3. Opcional: Conversión D/A : Reconstructores

En esta parte de la práctica nos concentraremos en los reconstructores.

Con el programa de la sección Prelaboratorio listo, ahora vamos a crear una función, rec cero que realice una reconstrucción de orden cero. En este tipo de reconstrucción, el valor de la señal reconstruida durante un intervalo de tiempo entre t = t(n) y t(n+1) es, simplemente, el valor que tenga la muestra al inicio del intervalo en t(n). Partiendo de nuestra señal muestreada obtenida con el programa anterior, xm, nuestra función nos devolverá la señal reconstruida, xr. Los parámetros de la función serán rec cero(xm,L), donde xm es la señal muestreada que queremos reconstruir, y L será la longitud de la señal reconstruida, que en nuestro caso haremos igual a 10 veces de la longitud de la señal xm muestreada. Tener en cuenta que:

1. Ambas señales, la muestreada y la reconstruida, tienen la misma duración temporal. Sin embargo esto no significa que tengan el mismo número de muestras. De hecho la señal reconstruida es una aproximación a la señal de continua y en nuestro caso tendrá 10 veces más muestras que la señal xm.

2. Por lo tanto, en cada intervalo de 10 muestras, la señal reconstruida tomará un valor constante e igual al valor de la señal xm al principio del intervalo.

La función debe quedar de esta forma:

rec cero = function(xm,L)

```
# Tenemos que hacer que salida de la funcion sea la senyal reconstruida
# Ayuda: se trata de crear un vector de longitud 'L' cuyo contenido es 'xm'
# repetido a intervalos L / length(xm)
```

Una vez obtenida la señal reconstruida xr, debéis compararla con la señal xc de referencia representando ambas con distintos colores en una misma gráfica.