# ELO 314 - Procesamiento Digital de Señales
# Lab. 1 - Guía complementaria:
# Señales Discretas en MatLab

**Preparado por**

Dr. Gonzalo Carrasco, e-mail: Gonzalo.Carrasco@usm.cl

## INTRODUCCIÓN

En esta guía se presentan actividades para experimentar y con ello profundizar más en los fenómenos y conceptos estudiados en el Laboratorio 1 del ramo usando Matlab. Se incluyen: fenómeno de doblaje, ruido de cuantización, y representación numérica en punto fijo.

## IV. EXPERIMENTANDO DOBLAJE

Usando Matlab, reproduzca el vector de datos `aliasing_test` del archivo `aliasing_test_16_16.mat` que se encuantra en la carpeta `audio_files` del repositorio usando `soundsc` a 16000ksps. Luego:

- considerando que el número de muestras es N, genere un nuevo vector de la mitad del tamaño tomando solo las muestras $2\,n$ para $n$ entre 1 y $N/2$.
- Similarmente, obtenga un vector con 1 de cada 3 muestras con un tercio del largo total. Esto es hacer un *Downsampling*.
- Reproduzca los vectores nuevos con soundsc a (16000/2)sps y (16000/3)sps respectivamente y analice lo que ocurre con los sonidos originales.

Para preparar su informe:

1) (**5ptos**) Comente que escucha y observa en base a la teoría.
2) (**5ptos**) ¿Cuál es la mínima tasa a la que podría re-muestrear este vector de datos? Observe el espectrograma de las señales usando `spectrogram(aliasing_test, 256, [], [], Fs, 'yaxis')`.

## V. EFECTOS DE LA CUANTIZACIÓN EN PROCESADORES DIGITALES

### A. Ruido de cuantización en conversores AD

Uno de los índices más usados para relacionar una señal de interés con el ruido asociado a la cuantización es el SQNR : *signal-to-quantization-noise ratio*:

$$SQNR \;=\; \frac{rms(\text{señal})}{rms(\text{ruido de cuantización})} \tag{1}$$

que suele medirse en $dB$. En un conversor Análogo a Digital (AD) el ruido de cuantización y por lo tanto esta relación, dependen de la señal análoga, tanto en amplitud, forma y frecuencia relativa a la frecuencia de muestreo. Sin embargo si se asume que el número de niveles de conversión es alto, y que la señal de prueba es de una amplitud que cubre gran parte del rango de conversión y (de ser periódica) tiene una frecuencia no divisor entero de la frecuencia de muestreo (no correlacionada con ella), entonces se puede asumir que el ruido de cuantización es no correlacionado con la señal y que está uniformemente distribuido entre la frecuencia 0 y $f_s/2$.

Si se asume una señal triangular de prueba aplicada a todo el rango de conversión de un convertidor *AD ideal* se cumplirá que para un convertidor de $N$ bits:

$$SQNR \;\approx\; 6.02\,N\;dB \tag{2}$$

Si la señal de prueba es sinusoidal que cubre todo le rango de conversión entonces el $SQNR \approx 1.76 + 6.02\,N\;dB$ (mejora un poco). **Notar** que en esta expresión el valor rms del ruido (que se asume uniformemente distribuido) se mide en todo el rango 0 a $f_s/2$, pues si se mide el ruido usando un ancho de banda menor, el SQNR subirá. Al contrario, si la amplitud de la señal no cubre todo el rango de conversión, el SQNR será menor.

Este modelo del ruido se debe considerar una solo como una base de analisis, pues de no cumplirse los supuestos es fácil interpretar de forma erronea su información. Además, no confundir con el SNR que es un índice que es la realción entre el valor rms de la señal y el valor rms de todo lo que se considere ruido (no solo el de cuantización).

### B. Actividad

Cargue y escuche los archivos `sonidos_voz_16_16.mat` y `música_16_16.wav`.

1) Cuando el número de niveles es suficientemente alto, una señal cuantizada se diferenciará de la original por un error de cuantización $e$ que es aproximable por una variable aleatoria de distribución uniforme.

   Implemente una función de cuantización `cuantiza(x, N)` que permita reducir los niveles de cuantización de una señal $x$ a $N$ niveles. Para este punto la cuantización se hará dividiendo el rango de la señal entre su máximo y mínimo en $N$ niveles considerando que:

$$\Delta = \frac{max(x) - min(x)}{N - 1} \tag{3}$$

   donde $\Delta$ es la separación entre niveles en las unidades de la señal original. En la figura **??** se observan los niveles posible a re-cuantizar si $N = 7$.

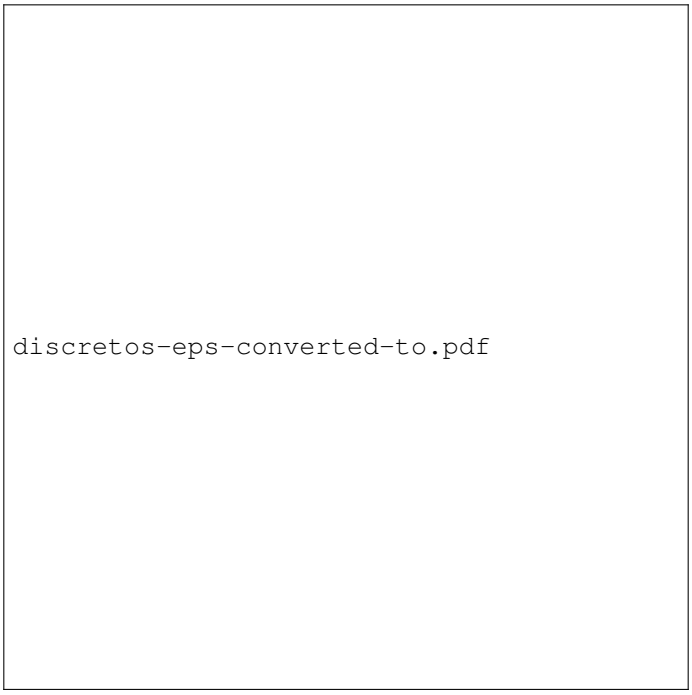


Figure 1. Aproximación de señal de alta resolución por una cantidad reducida de niveles

   Su función debe proceder a:

   a) Restar $min(x)$ a la señal original (queda una señal solo positiva) y luego dividirla por $\Delta$
   b) Redondear el resultado al entero más cercano

   (**4ptos**) Muestre el segmento de código de su función.
   Procese las señales para reducirle la cuantización a una equivalente a $b$ = 12, 8, 4, 2 y 1 bits, esto es $N = 2^b$ niveles.
   (**4ptos**) Reproduzca las nuevas señales usando `soundsc()` para 12, 8, 4, 2 y 1 bit y describa y comente el deterioro de las señales.
   (**4ptos**) ¿Cuál audio se deteriora más con la reducción de bits?.

2) Una señal de distribución uniforme se dice de ruido blanco pues no está correlacionada consigo misma. Se observará la relación entre la percepción de escuchar de la señal cuantizada y el tipo de ruido de cuantización y su correlación con la señal original.

   a) (**3ptos**) Modifique la función `cuantiza(x,N)` y cree la función `[y, e] = cuantiza2(x,N)` que entregue $y$ la señal cuantizada pero llevada al mismo rango de niveles de la señal original $x$, y también el error $e$ de aproximación:
      - A la señal redondeada, multiplique por $\Delta$ y sume el $min(x)$ para obtener $y$
      - El error es $e = y - x$

      (**4ptos**) Grafique de forma superpuesta las señales original y su versión cuantizada para $b = 2$.

b) Utilizando la función `cuantiza2()` junto con la función `hist(e, 20)` obtenga un histograma de 20 bins del error para cada señal de voz y de música. Grafique usando subplot para ver ambos histogramas. Observe el efecto para las 5 cantidades de bits, pero solo registre en su informe para b=2.
(**3ptos**) ¿Cómo afecta la reducción de bits a la distribución del histograma? ¿es uniformemente distribuida?.
(**3ptos**) Explique por qué el histograma toma una nueva forma.

c) Utilizando la función de Matlab `[r,l] = xcorr(e, 200, 'unbiased')` obtenga la autocorrelación del error para las diferentes cuantizaciones y observe la gráfica `plot(l,r)`.
Utilizando la función de Matlab `[r,l] = xcorr(e, y, 200, 'unbiased')` obtenga la correlación del error con la señal original para las diferentes cuantizaciones y observe la gráfica `plot(l,r)`.

(**3ptos**) Observe las correlaciones a medida que reduce el número de bits, pero grafique en un mismo gráfico las autocorrelaciones y correlación para $b = 12$ (voz y música), y en otro gráfico para $b = 2$.

(**3ptos**) ¿Cómo cambia la autocorrelación con la reducción de bits?

(**3ptos**) ¿Afecta el numero de niveles de cuantización a la correlación?.

3) Ahora, realice el proceso anterior modificando la función ahora llámela `cuantiza_dither(x, N)`, su función ahora:
- calcular el $\Delta$ como antes,
- luego, agregue ruido gaussiano con una desviación estándar cercana al 0.25 del valor del $\Delta$.
- A la nueva señal recalcule el $\Delta$ para la señal con ruido y proceda como a) y b) de la pregunta 1.

Esto es hacer *Dithering* de la señal cuantizada para descorelacionar el ruido de cuantización. Notar que esto se hace a costo de "cubrir" el ruido de cuantización con ruido blanco.

(**3ptos**) Muestre el segmento de código en su informe.
Reproduzca las señales y comente su percepción de la información experimentando para 4, 2 y 1 bit.
(**3ptos**) Para su oído ¿se hace más inteligible la señal que cuando no se hace dithering?. Comente al respecto.

## VI. Operación básica con punto fijo

En la práctica no todos los procesadores cuentan con unidades de aritmética y lógica (ALU) para operaciones con punto flotante. La tendencia actual está llevando incluso a que los microcontroladores de gamas medias cuenten con ALUs de punto flotante. Sin embargo, ya sea para reducir los tiempos de procesamiento, para reducir el área en silicio de diseño ASIC o en FPGA, o para reducir el consumo energético[1] que en los dos casos anteriores se incurre, el procesamiento aritmético en punto fijo sigue siendo una herramienta necesaria para reducir los costos necesarios para cubrir los tres recursos antes mencionados.

### A. Representación en punto fijo

En representación complemento 2 de un número entero de $n$ bits, la coma binaria está implicitamente a la derecha del bit menos significativo ($b_0$).

$$a = 0B\, b_{n-1} b_{n-2} \dots b_2 b_1 b_0 \tag{4}$$

Para $a$ el valor decimal que representa en complemento 2 es:

$$a_{entero} = -\mathbf{b_{n-1}}\, 2^{n-1} + \mathbf{b_{n-2}}\, 2^{n-2} + \cdots + \mathbf{b_2}\, 2^2 + \mathbf{b_1}\, 2^1 + \mathbf{b_0}\, 2^0 \tag{5}$$

Una forma típica de representar un número fraccionario en binario consiste en normalizar las variables para que su rango se encuntre entre -1 y 1. Luego, la representación interna que mejor aproxima dicho rango numérico[2] usando $n$ bits es dejar $n-1$ bits para la la parte fraccionaria y un bit de signo. La misma variable $a$ llevada a decimal ahora sería:

$$a_{fraccionario} = -\mathbf{b_{n-1}}\, 2^0 + \mathbf{b_{n-2}}\, 2^{-1} + \cdots + \mathbf{b_2}\, 2^{-(n-3)} + \mathbf{b_1}\, 2^{-(n-2)} + \mathbf{b_0}\, 2^{-(n-1)} \tag{6}$$

lo que implica que el punto binario se encuentra implícitamente entre los bits $b_{n-1}$ y $b_{n-2}$. Notar que se puede escribir como:

$$\begin{aligned} a_{fraccionario} &= \left(-\mathbf{b_{n-1}}\, 2^{n-1} + \mathbf{b_{n-2}}\, 2^{n-2} + \cdots + \mathbf{b_2}\, 2^2 + \mathbf{b_1}\, 2^1 + \mathbf{b_0}\, 2^0\right) 2^{(n-1)} && (7) \\ a_{fraccionario} &= a_{ri}\, 2^{(n-1)} && (8) \end{aligned}$$

donde la representación interna ($ri$) resulta como un entero semejante a $a_{entero}$. El número de bits (q) dejados para la parte fraccionaria junto a la letra **Q** se usa para referirse a la representación en punto fijo, por ejemplo, para 16 bits en representación $Q15$ la palabra 0B0110000000000000 representa 0.75 y 0B1110000000000 representa -0.25.

Notar también que las operaciones aritméticas que el procesador realice son las mismas para un número entero o un número en Q15, pero la **interpretación de la palabra binaria y las operaciones aritméticas es la que cambia**. Al sumar (o restar) dos variables, hay que cuidar que no sumen un número fuera del rango -1 y 1, y **especial cuidado hay que poner en las multiplicaciones y divisiones** dado que la representación interna ($ri$) modificará la cantidad de bits fraccionarios. Considerando que la representación externa de interés para el programador es:

$$a = a_{ri}\, \mathbf{2^{-q}} \tag{9}$$

entonces una operación de multipliación $c = a \cdot b$ internamente en el procesador se ejecutará como:

$$\hat{c} = a_{ri} \cdot b_{ri} \tag{10}$$

pero se debe interpretar externamente como

$$\begin{aligned} c &= a \cdot b && (11) \\ &= a_{ri}\, 2^{-q} \cdot b_{ri}\, 2^{-q} && (12) \\ &= \left(a_{ri}\, b_{ri}\, 2^{-q}\right) \mathbf{2^{-q}} && (13) \end{aligned}$$

lo que implica que:

$$c_{ri} = a_{ri}\, b_{ri}\, 2^{-q} \tag{14}$$

y por lo tanto luego de la multiplicación de las representaciones internas de $a$ y $b$ hay que dividir $\hat{c}$ por $2^q$. Para la arquitectura binaria de un procesador digital una división por una potencia de 2 no es más costosa que un desplazamiento aritmético a la derecha.

[1] procesar más tiempo implica mayor consumo de energía eléctrica en la electronica interna, y un circuito más grande en silicio también puede consumir más energía eléctrica, crítico en dispositivos portátiles

[2] el rango representable resulta entre $-1$ y $\left(1 - 2^{-(n-1)}\right)$

### *B. Matlab y tipos de dato enteros*

Matlab cuenta con la posibilidad de representar números enteros con y sin signo mediante los tipos:
`Uint8`, `Int8`, `Uint16`, `Int16`, `Uint32`, `Int32`, `Uint64` e `Int64`
Es posible *castear* una variable a uno de estos tipo usando su nombre como función:

```
a = 15;             % Por defecto es tipo 'double'
b = uint8(a);
c = int16(32767);
```

Además permite ingresar literales numéricos en formato hexadecimales y binario usando los prefijos `0x` o `0X` para un número hexadecimal y `0b` o `0B` para un número binario. Estos números se almacenan como enteros (no son tipos ni hexadecimal ni binario, solo son formas literales de entregar a Matlab un número entero).

```
d = 0xFF
e = 0B10001010
```

Por defecto Matlab usará el tipo entero sin signo más pequeño de los antes mencionados en que se pueda representar el literal numérico. De esta manera `e` resulta del tipo `uint8`.
Es posible forzar el tipo de dato entero al literal hexadecimal o binario entregado agregando el sufijo `u8`, `u16`, `u32` y `u64` para los tipos sin signo, y `s8`, `s16`, `s32` y `s64` para los tipos con signo, interpretando en este último caso el valor entregado en complemento 2, si el bit de signo es 1 (posición más significativa en cada caso). La variable `f` representa -1 en 8 bits y `g` el 138 en 16 bits.

```
f = 0xFFs8
g = 0B10001010u16
```

También existen algunas funciones orientadas *al bit*, tales como `bitand(a,b)`, `bitor(a,b)`, `bitset(a, k)` y `bitshift(a, k)` que permite hacer desplazamientos aritméticos a la derecha e izquierda.

### *C. **Actividad***

Considere el caso de aplicar un enventanamiento con una ventana de tipo Blackman a un vector de datos de audio. Suponga que debe operar solo usando aritmética de punto fijo, y que su conversor AD o códec de audio le entrega también las muestras cuantizadas como un número entero con signo. Asuma que el vector de datos a procesar tiene largo fijo $N = 161$ y que consecuentemente los taps o puntos de la ventana blackman tendrán esa longitud fija que debe almacenar en memoria no volátil como constantes.

a) Lea el archivo de audio `aliasing_test_16_16.wav` con su tipo de dato nativo usando `audioread('<>.wav', 'native')` y rescate en una variable $x$ las primeras $N$ muestras. Aplique una ponderación a cada muestra correspondiente a un enventanamiento de las muestras con los datos obtenidos con `blackman()`. Para todas sus operaciones y variables intermedias use solo los tipos de datos y funciones de la familia mencionadas en esta sección, excepto para generar $w$ la primera vez que debe resultar en tipo int16 a partir del tipo double que entrega `blackman()`.

   (**3ptos**) ¿Cuál es el vector $w$ de constantes de la ventana que dejaría en memoria?.
   **Muestre como genera** $w$. Usando el comando `whos` de Matlab, **observe y anote** cuanta memoria en bytes requiere el vector entregado por `blackman()` y el espacio mínimo que requiere su representación interna para $w$.

   (**4ptos**) **Muestre el código** que calcula el vector $y$ (en el tipo de dato nativo) correspondiente a $x$ ponderado por $w$. ¿**Cuanta memoria requiere** $y$? ¿**Cuanta memoria requiere** $y$ cuando se importa con `audioread('<>.wav')`?.

   (**3ptos**) **Grafique** en un mismo eje coordenado $x$, $w$ e $y$ usando plot solo con marcadores distintos (sin interpolar entre puntos).