



**INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO**



Teoría de Comunicaciones y Señales

“Convolución”

Resumen

Implementación de un circuito RC en configuración filtro pasa bajas, haciendo uso de la convolución discreta y la respuesta al impulso del circuito RC serie para señales provenientes de archivos WAV en lenguaje C.

Por:

Romero Gamarra Joel Mauricio

Profesor:

GUTIÉRREZ ALDANA EDUARDO

Septiembre 2017

Índice

Contenido

Introducción:.....	1
Análisis Teórico:	2
Software (librerías, paquetes, herramientas):.....	4
Procedimiento:	4
Resultados	7
Discusión:	11
Conclusiones:.....	14
Referencias:	14
Código	15

Introducción:

En el siguiente programa, se realizó el diseño de un filtro pasa – bajas en lenguaje C¹, haciendo uso del teorema de convolución, sin embargo al realizarse en una computadora con tiempo memoria finita, se hizo un pequeño cambio al teorema para migrarlo al tiempo discreto, ya que por definición una integral es una suma, por lo tanto el único cambio fue una sumatoria de 0 al 20 (memoria usada para el filtro FIR), que es un sistema de fácil implementación y diseño pero que consume recursos de memoria y procesamiento mayor, sin embargo son sistemas **estables**,¹ que en este caso se utilizarán 20 flotantes.

Al decir que un sistema es estable, nos referimos a que la respuesta del sistema a un impulso será 0 cuando el tiempo tiende a ∞ , si, por ejemplo, la respuesta del sistema al impulso tiende a un valor finito distinto de 0, se dice que es **críticamente estable**, y si tiende a un valor infinito, quiere decir que el sistema es **inestable**. En la Figura 1, podemos apreciar en el diagrama de polos y ceros la respuesta homogénea de un sistema a partir de sus polos y cómo se comporta el sistema en cada polo (estable o inestable).

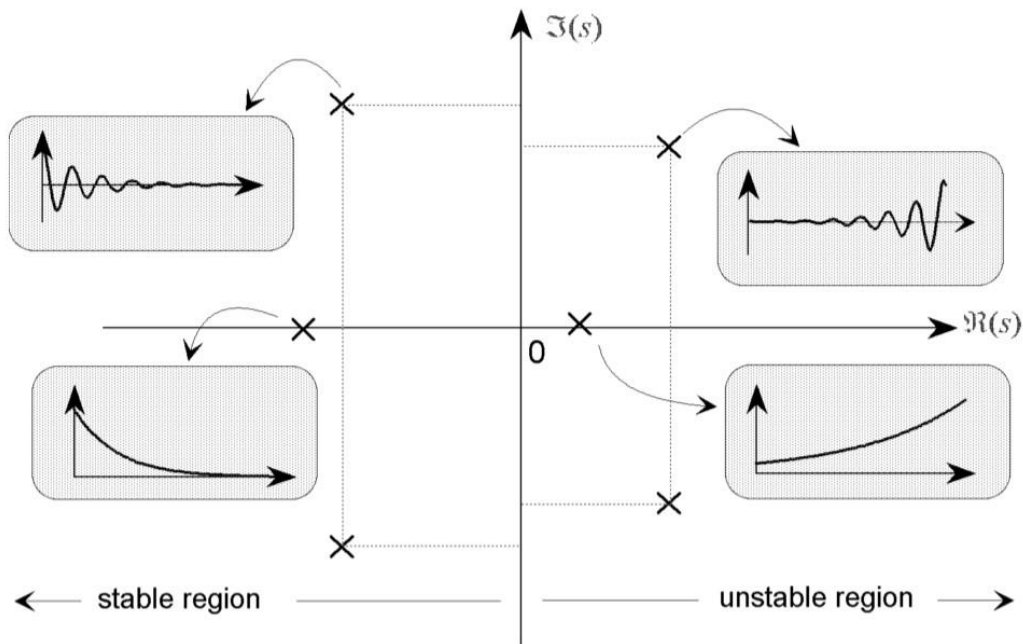


Figura 1. Respuesta homogénea al sistema en la gráfica de polos y ceros.

Después de observar la Figura 1, sabemos que, para ser un sistema estable, los polos de la función de transferencia deben estar ubicados en el semiplano izquierdo del diagrama de polos y ceros², este tipo de sistemas es el filtro pasa – bajas que diseñamos en esta práctica.

Análisis Teórico:

Esta sección necesita proveer un entendimiento del teórico, matemático y conceptual del contexto, antecedentes y justificación del trabajo.

Se pueden incluir diagramas, fórmulas, algoritmos, etc.

Comenzando con el análisis a priori del algoritmo a utilizar, necesitamos tener un mejor entendimiento de lo que es un filtro pasa – bajas y cómo funciona. Tomemos como ejemplo un circuito RC de primer orden, que es un circuito electrónico formado por una resistencia y un capacitor. Una de las características más importantes de este tipo de circuitos, es que este tipo de sistema es **lineal** (Cumple con propiedades de superposición) e **invariante en el tiempo** (la salida es la misma sin importar el instante de tiempo en el que se le aplique la entrada). En la Figura 2, podemos observar la configuración física de un circuito RC en una configuración pasa – bajas.³

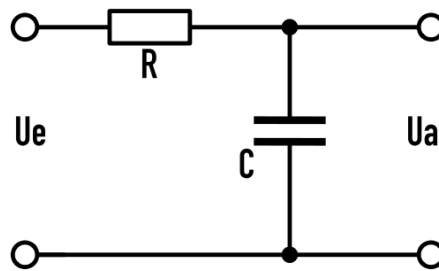


Figura 2. Configuración Pasa Bajas de un filtro RC de primer orden.

Lo que caracteriza a un filtro pasa bajas es que permite el paso de las frecuencias menores a la frecuencia de corte. Para altas frecuencias, la reactancia es baja logrando con esto que las señales sean atenuadas. En cambio, a bajas frecuencias (por debajo de la frecuencia de corte) la reactancia capacitiva es grande, lo que causa que estas frecuencias no se vean afectadas o son afectadas muy poco por el filtro.⁴

La frecuencia de corte es aquella donde la amplitud de la señal entrante cae hasta un 70.7 % de su valor máximo, esto ocurre cuando $XC = R$, a las frecuencias debajo de la frecuencia de corte se les conoce como **Banda de paso**, a las frecuencias por encima de la frecuencia de corte se les llama **Banda de atenuación**.⁴

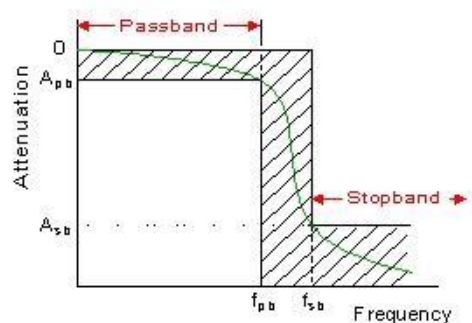


Figura 3. Comportamiento de un filtro pasa bajas.

En la Figura 3, se puede ver el comportamiento que tiene un filtro pasa bajas graficando la frecuencia, la zona donde comienza “*stopband*”, es donde las frecuencias superaron a la frecuencia de corte, por lo tanto, en ese momento baja a 0 (filtro ideal), sin embargo, un filtro real va atenuando poco a poco las frecuencias hasta llegar a 0.

Posteriormente, ya que tenemos un entendimiento general del funcionamiento de un circuito RC en configuración de un filtro pasa bajas, procedemos a dar una explicación acerca de la convolución.

Como se mencionó en la introducción, la convolución en tiempo continuo se define como sigue:

$$x(t) * h(t) = \int_{-\infty}^{\infty} x(\tau)h(t - \tau)d\tau$$

Donde:

- $x(t)$: Señal de entrada.
- $h(t)$: Respuesta del sistema al impulso.
- τ : Desplazamiento en el tiempo.

Sin embargo, al diseñar un filtro pasa bajas en tiempo discreto para una señal digital (en este caso un archivo WAV), debemos tener en cuenta que poseemos una memoria y un tiempo de cómputo finito, sabemos que, una integral es una suma de todos los números desde $-\infty$ hasta ∞ , eso represente un número infinito de números, inclusive, si hacemos la integral de 0 a 1 (suponiendo que solo tuviéramos valores en las señales en ese intervalo), entre ambos valores hay una infinidad de números, por lo tanto haremos un cambio para hacer una suma tomando valores discretos ($-\infty, \dots, -2, -1, 0, 1, 2, \dots, \infty$) como sigue¹:

$$x(n) * h(n) = \sum_{\tau = -\infty}^{\infty} x(\tau)h(n - \tau)$$

Ya que tomamos en cuenta únicamente valores discretos para realizar nuestra convolución, debemos notar que la suma está definida desde $-\infty$ hasta ∞ , por lo que es prácticamente imposible realizarla en una computadora. De modo que, en esta parte es donde entra en juego la cantidad de memoria utilizada, en esta práctica se realizó con un impulso con 20 valores, así que, finalmente nuestra ecuación a usar en lenguaje C, será la siguiente.

$$x(n) * h(n) = \sum_{\tau = 0}^{20} x(\tau)h(n - \tau)$$

Ya que tenemos la fórmula que usaremos para calcular la convolución, necesitamos 2 cosas: Nuestra señal de entrada, y la respuesta al impulso que usaremos, sabemos que la respuesta al impulso de un sistema se define como la respuesta de un sistema a $\delta(t)$, que se define así:

$$\int_{-\infty}^{\infty} \delta(t)dt = 1$$

Para obtener la respuesta al impulso del sistema, primero debemos obtener la función de transferencia de nuestro circuito RC en configuración pasa – bajas, ver Figura 2. Para resolver el circuito, definimos a la función de transferencia como sigue:

$$H(s) = \frac{V_{sal}(s)}{V_{ent}(s)}$$

Al observar la ecuación anterior, podemos ver que debemos obtener el voltaje de salida y el voltaje de entrada en el circuito RC. Al ver la Figura 2, vemos que el voltaje de salida se mide en el capacitor, por lo tanto, la impedancia compleja en el capacitor se define como $\frac{1}{sC}$, que colocaremos en el numerador de la ecuación anterior en la función de transferencia. Además, para el voltaje de entrada se observa que se encuentran el capacitor y la resistencia en serie, sabiendo que para calcular el voltaje en ambos elementos se deben sumar, lo realizamos, conociendo la impedancia en el capacitor y sumamos la resistencia, que se define como $\frac{1}{sC} + R$, que colocaremos como el denominador de la función de transferencia y la nueva ecuación nos queda de la siguiente forma:

$$H(s) = \frac{\frac{1}{sC}}{\frac{1}{sC} + R} = \frac{1}{1 + sRC}$$

Ya que tenemos calculada la función de transferencia para el circuito RC en configuración Pasa – Bajas, calculamos la transformada inversa de Laplace para pasar la función de transferencia al dominio del tiempo y obtener $h(t)$ de la siguiente forma:

$$\mathcal{L}\{H(s)\} = h(t) = e^{-\frac{2\tau\pi f_c}{44,100}}$$

Al tener la respuesta al impulso deseada, vemos que existe τ , y recordemos que, en nuestra convolución discreta, τ va a tener valores de 0 a 20, para poder calcular nuestro impulso que usaremos como $h(n)$ para realizar la convolución con la señal de entrada.

Software (librerías, paquetes, herramientas):

- GoldWave versión 4.26
- Frhed 1.6.0
- Dev C
- Sublime Text 3
- Bizagi Modeler

Procedimiento:

Para explicar el procedimiento utilizado, retomaremos la idea que este filtro pasa – bajas estará programado en lenguaje C, por lo tanto, debemos obtener el algoritmo para realizar la convolución, sabiendo que, por definición, la señal de entrada debe voltearse y multiplicar sus valores por la respuesta al impulso, sumando estos valores y escribiéndolos en una señal nueva. Por lo tanto, utilizaremos 2 arreglos de tipo short (ya conocemos la estructura de un archivo WAV⁵). Sin

embargo, como la señal generada con GoldWave es una señal que entra a un sistema (en este caso la computadora), ya entra “invertida”, por lo que únicamente se guardarán los elementos en el arreglo y se irá multiplicando por el impulso obtenido. A continuación, en la Figura 4, podemos observar el diagrama de flujo que nos muestra los pasos a seguir para realizar el filtrado a la señal de entrada.

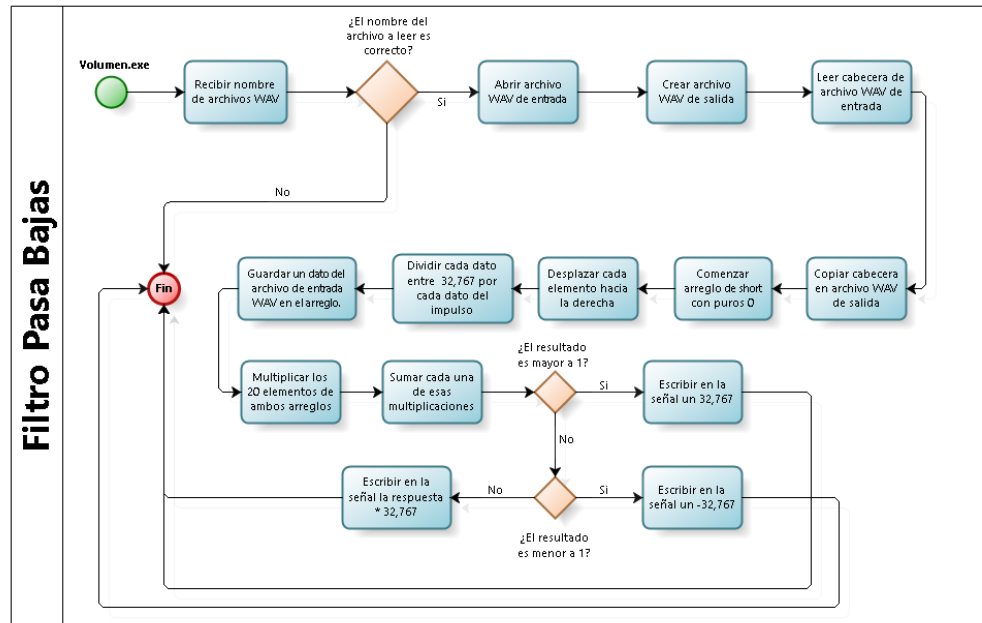


Figura 4. Diagrama de flujo del proceso general del filtro pasa bajas en lenguaje C.

Ahora que conocemos el proceso general (Ver Figura 4), podemos comenzar a codificar. Ya sabemos toda la estructura de la cabecera y los datos en un archivo WAV (revisar [5]), así que procederemos a explicar únicamente el algoritmo para realizar la convolución en lenguaje C¹.

El algoritmo es muy sencillo, como lo indica el diagrama primero debemos crear un arreglo de tipo short de tamaño 20 (que es la memoria utilizada), al que llamaremos entrada en donde guardaremos los datos de la señal generada con GoldWave. Debemos tener otro arreglo de tipo float y tamaño 20, sin embargo, lo llamaremos impulso y en él, guardaremos la respuesta del sistema al impulso unitario, que está descrito en el análisis teórico de este reporte, estos valores serán fijos durante toda la ejecución del programa.

```
float * generaImpulso ()
{
    float * impulso = (float *) malloc (sizeof (float) * TAM_ARREGLO);
    for (i = 0; i < TAM_ARREGLO; i ++)
    {
        impulso [i] = (exp ((-2 * PI * FREQ_CORTE * i) / 44100));
    }
    return impulso;
}
```

Figura 5. Función que genera el impulso a partir de las fórmulas obtenidas en el análisis teórico de la solución.

Posteriormente, utilizaremos otro flotante auxiliar al que llamaremos max, en el que guardaremos la suma de multiplicar cada elemento del arreglo impulso por 32,767 (valor máximo de un short, y valor máximo que puede tener la señal generada en GoldWave), para obtener nuestro peor caso. Ahora, teniendo ambos arreglos, procedemos a hacer un ciclo de 0 a SubChunkSize2⁵ para leer los datos de la señal de entrada, cada dato lo vamos a dividir entre max para redimensionar nuestra entrada y tener valores entre -1 y 1 únicamente. Procedemos a recorrer nuestros elementos del arreglo 'entrada' hacia la derecha y posteriormente guardar el dato dividido en la primera posición del arreglo.

```
for (i = 0; i < cab.SubChunk2Size; i ++, j++)
{
    lectura = fread(&muestra, sizeof (short), 1, archivoEntrada);
    for (k = (TAM_ARREGLO - 1); k >= 0; k --)
        entrada [k] = entrada [k - 1]; ← Desplazamiento de elementos
    entrada [0] = (muestra / max);
    muestra = convolucion (entrada, impulso);
    escritura = fwrite(&muestra, sizeof (short), lectura, archivoSalida);
}
```

Figura 6. Ciclo donde se realiza el desplazamiento, almacenamiento y escritura de la señal filtrada.

Ya que lo almacenamos, debemos hacer un ciclo de 0 a 20, en el que vamos a multiplicar los 2 arreglos (impulso y entrada) y vamos a ir sumando cada resultado.

```
float convolucion (float * entrada, float * impulso)
{
    float respuesta = 0;

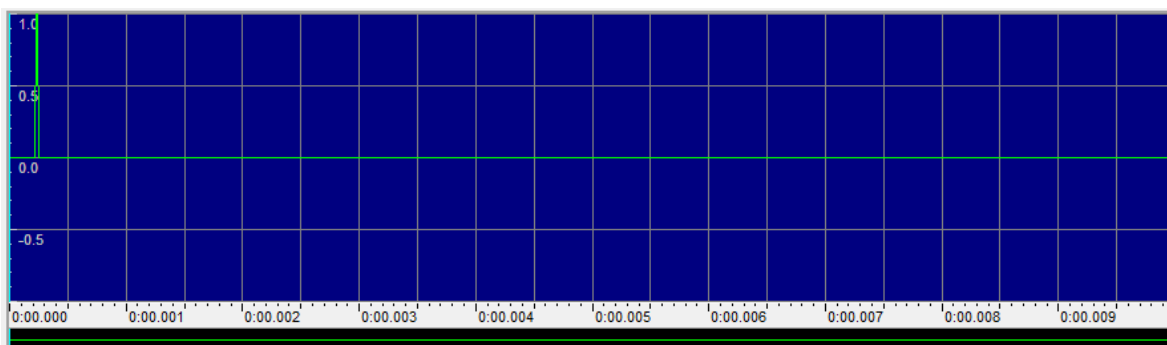
    //Realizamos la convolución
    for (i = 0; i < TAM_ARREGLO; i ++ )
        if (entrada [i] != 0)
            respuesta += (entrada [i] * impulso [i]);

    if (respuesta > 1)
        return (1 * 32767);
    if (respuesta < -1)
        return (-1 * 32767);
    else
        return (respuesta * 32767);
}
```

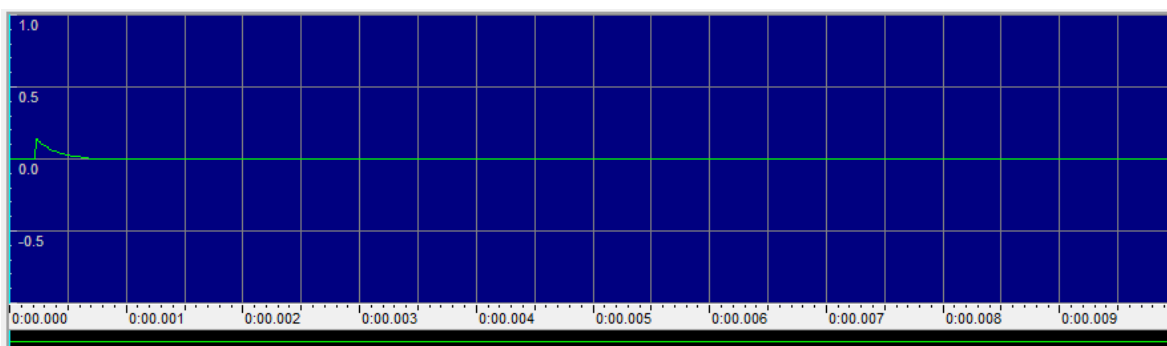
Figura 7. Algoritmo que realiza la convolución tomando en cuenta los valores posibles.

Después de esto, **existen 3 posibles valores** que obtendremos de esa suma, **valores menores a -1**, por lo que escribiremos un -32,767 (valor mínimo que puede tener un short), **valores mayores a 1**, por lo que escribiremos un 32,767 (valor máximo que puede tener un short), y **valores entre -1 y 1**, por lo que escribiremos el valor obtenido multiplicado por 32,767. Todo este algoritmo, se realizará en un ciclo hasta que leamos todos los datos, posteriormente, tendremos la señal filtrada. En la sección de resultados se muestran los ejemplos de entrada y las señales obtenidas al realizar el filtrado con el programa.

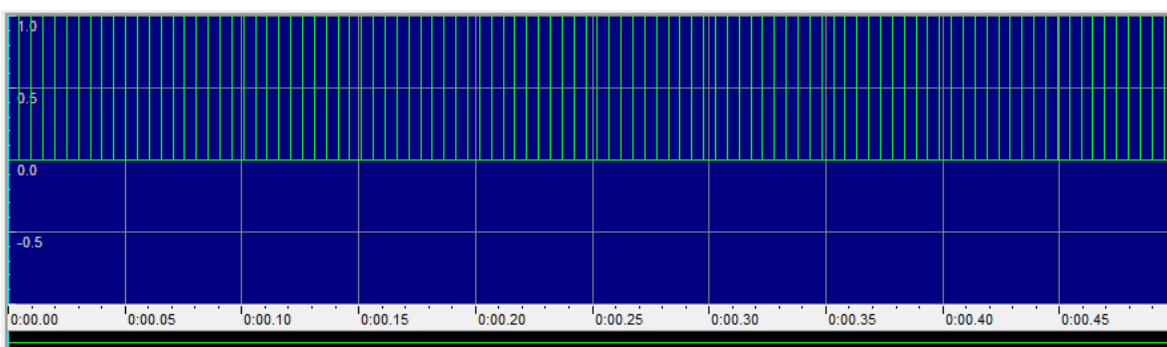
Resultados



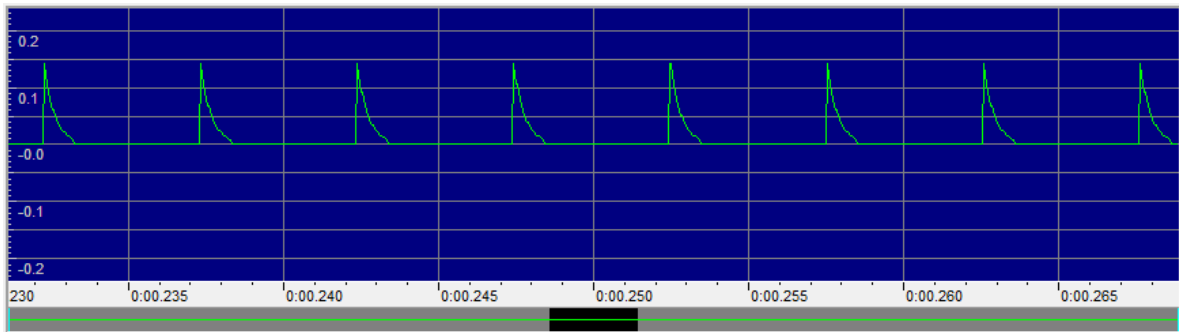
Señal de Entrada 1. Delta de Dirac



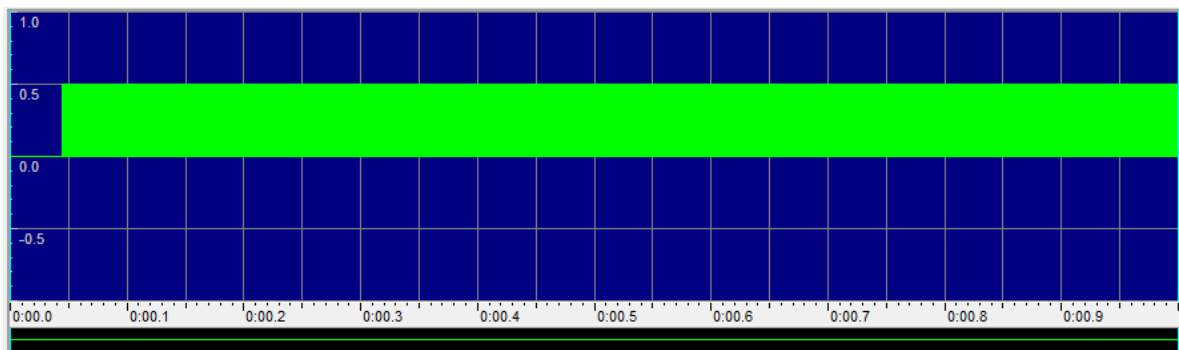
Señal Filtrada 1. Delta de Dirac



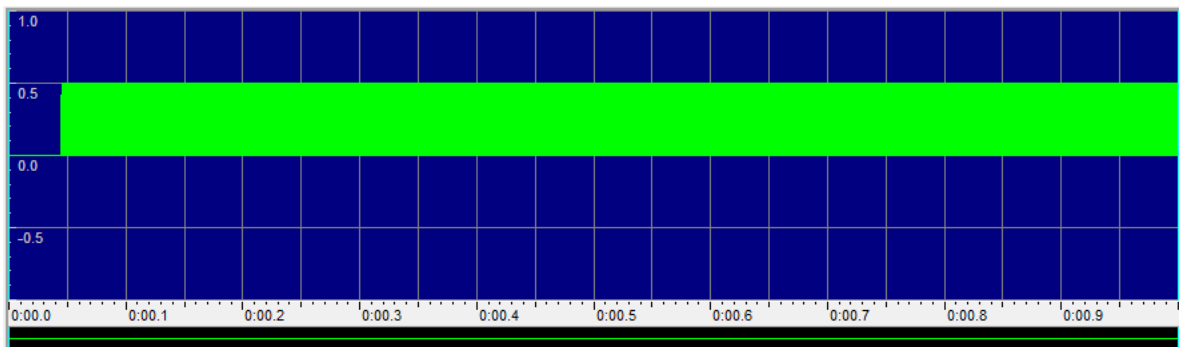
Señal de Entrada 2. Tren de Impulsos



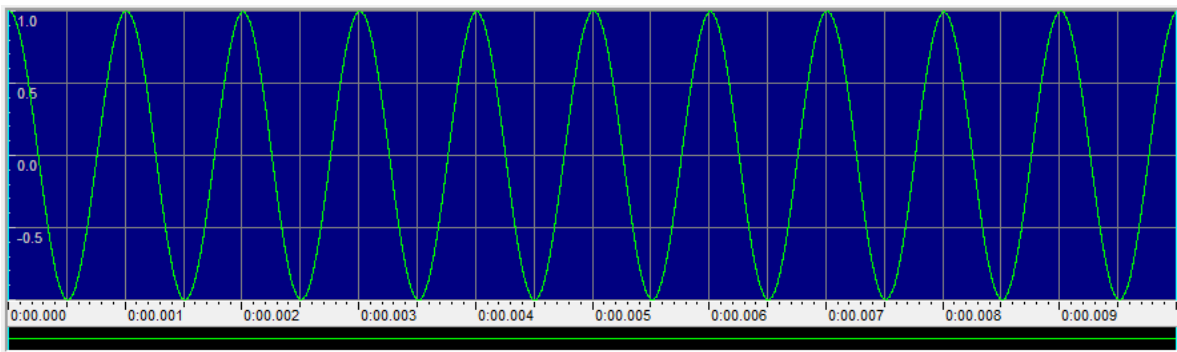
Señal Filtrada 2. Tren de Impulsos (Ampliado)



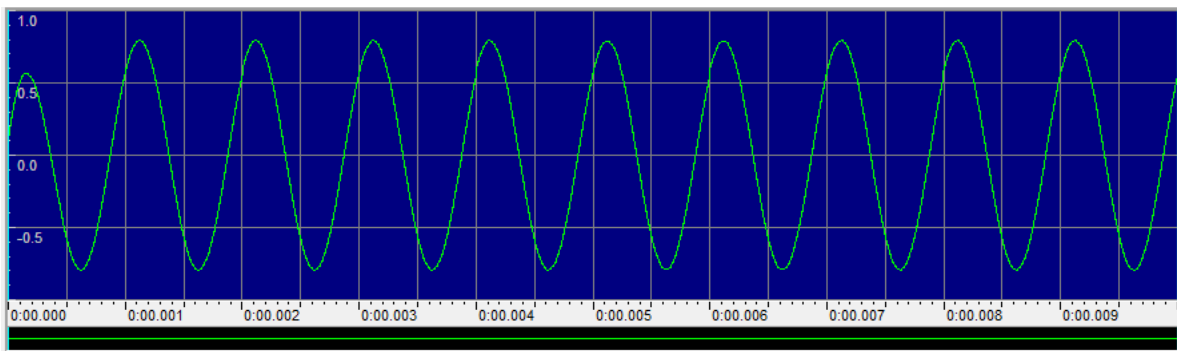
Señal de Entrada 3. Función Heaviside



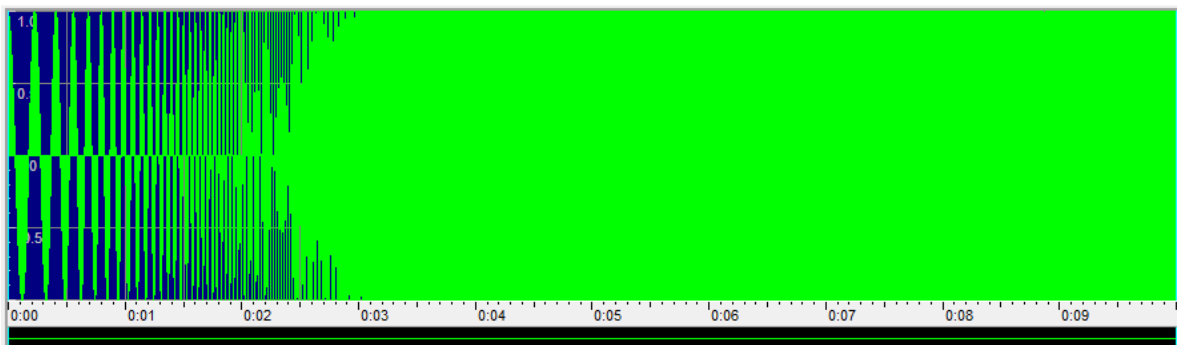
Señal Filtrada 3. Función Heaviside



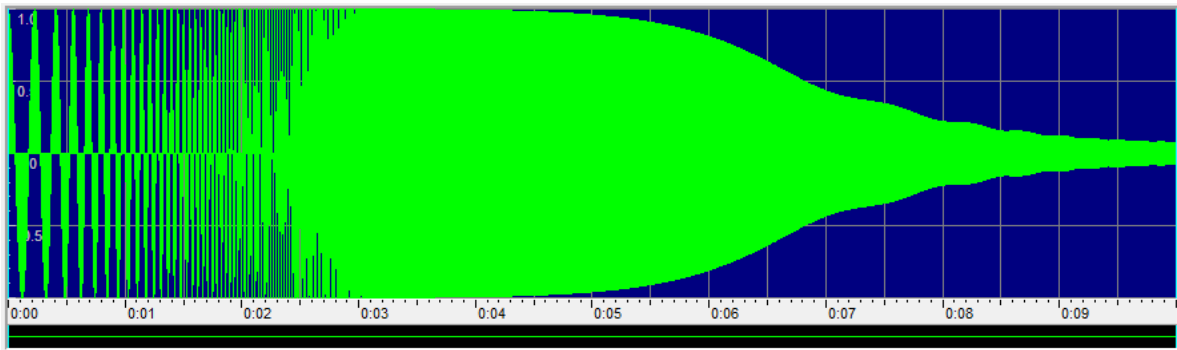
Señal de Entrada 4. Coseno



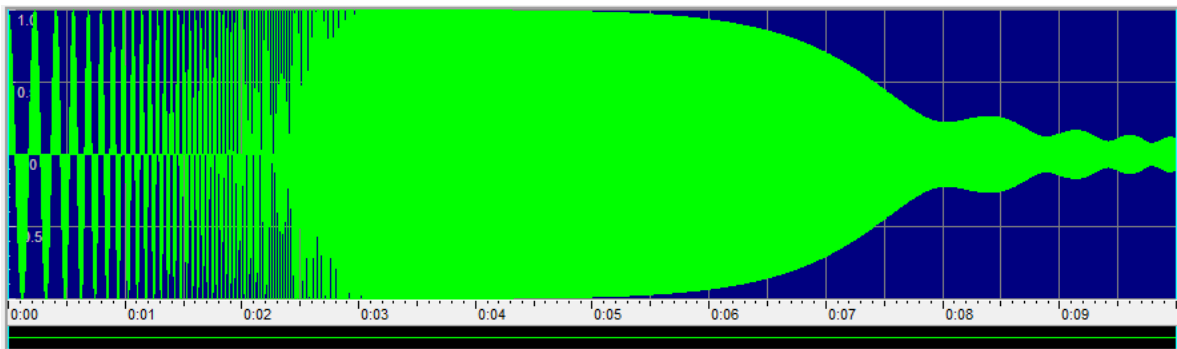
Señal Filtrada 4. Coseno



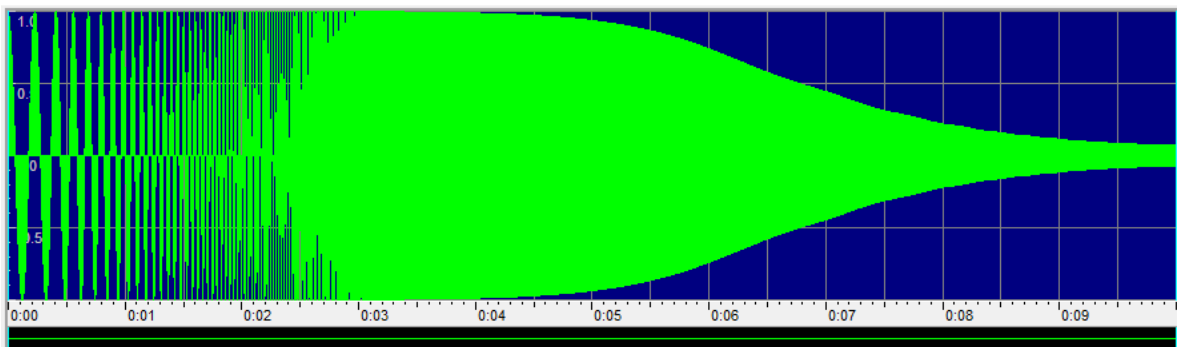
Señal de Entrada 5. Coseno Inestable



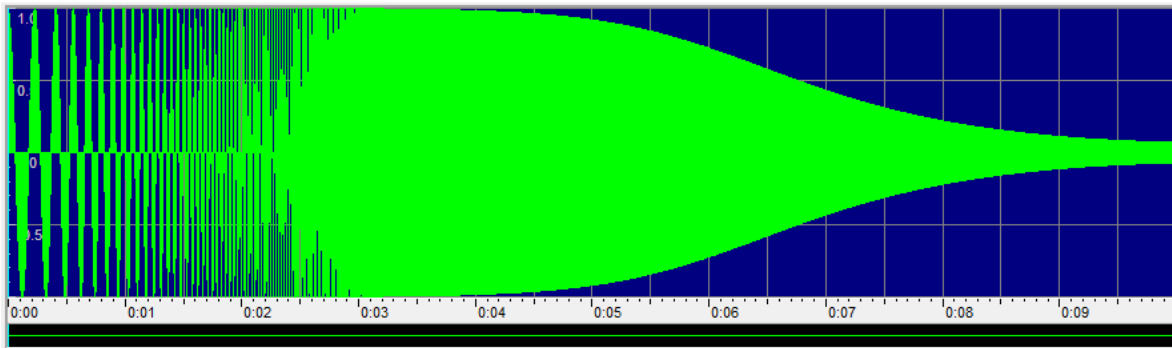
Señal Filtrada 5. Coseno Inestable (Memoria utilizada = 20)



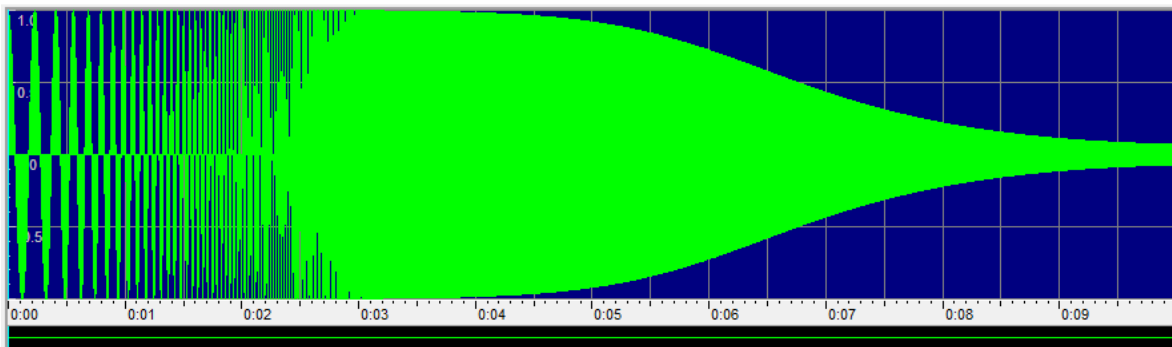
Señal Filtrada 6. Coseno Inestable (Memoria utilizada = 10)



Señal Filtrada 7. Coseno Inestable (Memoria utilizada = 30)



Señal Filtrada 8. Coseno Inestable (Memoria utilizada = 100)



Señal Filtrada 9. Coseno Inestable (Memoria utilizada = 1000)

Discusión:

La sección de discusión tiene 2 objetivos principales:

- Interpretar y explicar los resultados del estudio.
- Explorar la importancia del estudio, encontrando, calificando y explorando la importancia teórica de los resultados.

La discusión es también un espacio en el reporte donde cualquier calificación o reservación que se tiene sobre la investigación debe ser mencionada.

En los resultados obtenidos, se puede observar una notable diferencia entre la señal de salida que se introduce al sistema y la señal de salida al pasar por el filtro pasa bajas. En algunas señales, como la función Heaviside o el Coseno, no es tan notorio el cambio, sin embargo, en las demás, es bastante distinta la señal, dejando pasar únicamente las frecuencias menores a la frecuencia de corte (para esta práctica, la frecuencia de corte es de 1 KHz), y atenuando las frecuencias mayores a ésta.

Es importante resaltar que podemos comprobar que el comportamiento mostrado en la Figura 3 es correcto, además, aseveramos que la gráfica real difiere de la gráfica ideal de un filtro pasa bajas, debido a que existen distintos factores que hacen imposible el eliminar inmediatamente las frecuencias (es decir, mandarlas a 0), pero, lo que sí podemos hacer, es ir las atenuando paulatinamente.

En las señales filtradas del coseno inestable (recordemos que una señal inestable es una señal que crece indefinidamente, es decir que los polos que la describen se encuentran gráficamente en el semiplano derecho del diagrama de polos y ceros), es interesante notar que mientras menor sea la memoria utilizada para realizar la convolución, la señal filtrada tiene un poco de ruido, ya que no tiene un filtrado tan correcto como lo esperaríamos.

Además, debido a esto, podríamos esperar que mientras se utilice más memoria, la señal filtrada obtenida sería más limpia, y es algo cierto, sin embargo, existe un límite, como podemos ver si utilizamos 30 números flotantes, comparada con 10 o 20, la diferencia es muy notoria, pero si la aumentamos a 100, ya no hay diferencia, o al menos no gráficamente, y cuando aumentamos a 1,000, se puede ver que efectivamente no existe un cambio significativo en la señal de salida. Por lo cual, estaríamos consumiendo demasiado tiempo de cómputo y recursos de la computadora, realizando operaciones innecesarias para una respuesta idéntica o muy parecida.

Para poder corroborar esto, no podemos seguir viendo únicamente con el software GoldWave, así que nos auxiliaremos con Frhed que es un software que nos permite visualizar estas señales en formato binario o hexadecimal.

00000	52 49 46 46	be 75 0d 00	57 41 56 45	66 6d 74 20	10 00 00 00	01 00 01 00	44 ac 00 00	RIFF%u..wAvefmt	D~..
0001c	88 58 01 00	02 00 10 00	64 61 74 61	50 75 0d 00	61 16 ca 29	9f 3a 37 49	e0 55 da 60	.X.....dataPu..a.é).:7IaU0'	
00038	5f 6a a0 72	c9 79 ff 7f	ff 7f fe 7f	fe 7f fe 7f	fe 7f fe 7f	fe 7f fe 7f	fe 7f fd 7f	_j rÉyy.y.p.p.p.p.p.p.p.p.y.	
00054	fd 7f fd 7f	fd 7f fd 7f	fd 7f fc 7f	fc 7f fc 7f	fc 7f fc 7f	fb 7f fb 7f	fb 7f fa 7f	y.y.y.y.ü.ü.ü.ü.ü.ü.ü.ü.	
00070	fa 7f fa 7f	fa 7f f9 7f	f9 7f f9 7f	f8 7f f8 7f	f8 7f f7 7f	f7 7f f7 7f	f6 7f f6 7f	ü.ü.ü.ü.ü.ü.ü.ü.ü.ü.ü.ü.	
0008c	f5 7f f5 7f	f5 7f f4 7f	f4 7f f3 7f	f3 7f f2 7f	f2 7f f1 7f	f1 7f f0 7f	f0 7f ef 7f	ö.ö.ö.ö.ö.ö.ö.ö.ö.ö.ö.ö.	
000a8	ef 7f ee 7f	ee 7f ed 7f	ed 7f ec 7f	ec 7f eb 7f	eb 7f ea 7f	e9 7f e9 7f	e8 7f e7 7f	ï.ï.ï.ï.ï.ï.ï.ï.ï.ï.ï.ï.	
000c4	e7 7f e6 7f	e5 7f e5 7f	e4 7f e3 7f	e3 7f e2 7f	e1 7f e1 7f	e0 7f df 7f	de 7f dd 7f	ç.æ.ä.ä.ä.ä.ä.ä.ä.ä.ä.ä.	
000e0	dd 7f dc 7f	db 7f da 7f	da 7f d9 7f	d8 7f d7 7f	d6 7f d5 7f	d5 7f d4 7f	d3 7f d2 7f	Y.Ü.Ü.Ü.Ü.Ü.Ü.Ü.Ü.Ü.Ü.Ü.	
000fc	d1 7f d0 7f	cf 7f ce 7f	cd 7f cc 7f	cb 7f ca 7f	ca 7f c9 7f	c8 7f c7 7f	c5 7f c4 7f	N.ö.ï.ï.ï.ï.ï.ï.ï.ï.ï.ï.	
00118	c3 7f c2 7f	c1 7f c0 7f	bf 7f be 7f	bd 7f bc 7f	bb 7f ba 7f	b9 7f b7 7f	b6 7f b5 7f	A.Ä.Ä.Ä.Ä.Ä.Ä.Ä.Ä.Ä.Ä.	
00134	b4 7f b3 7f	b2 7f b0 7f	af 7f ae 7f	ad 7f ac 7f	aa 7f a9 7f	a8 7f a7 7f	a5 7f a4 7f	'.'.'.'.'.'.'.'.'.'.'.'.'.'.	
00150	a3 7f a2 7f	a0 7f 9f 7f	9e 7f 9c 7f	9b 7f 9a 7f	98 7f 97 7f	96 7f 94 7f	93 7f 91 7f	é.é.	
0016c	90 7f 8f 7f	8d 7f 8c 7f	8a 7f 89 7f	87 7f 86 7f	84 7f 83 7f	81 7f 80 7f	7e 7f 7d 7f	
00188	7b 7f 7a 7f	78 7f 77 7f	75 7f 74 7f	72 7f 70 7f	6f 7f 6d 7f	6b 7f 6a 7f	68 7f 66 7f	{.z.x.w.u.t.r.p.o.m.k.j.h.f.	

Hexadecimal 1. Coseno Inestable Filtrado (Memoria utilizada = 10)

00000	52 49 46 46	be 75 0d 00	57 41 56 45	66 6d 74 20	10 00 00 00	01 00 01 00	44 ac 00 00	RIFF%u..wAvefmt	D~..
0001c	88 58 01 00	02 00 10 00	64 61 74 61	50 75 0d 00	3c 11 2f 00	25 2d 63 38	23 42 97 4a	.X.....dataPu..<./}%-c8#B.J	
00038	ec 51 48 58	cb 5d 93 62	b9 66 51 6a	6f 6d 24 70	7c 72 84 74	48 76 cf 77	23 79 49 7a	iQHxE].b'fQjomSp r.thvIw#yIZ	
00054	48 7b 26 7c	e6 7c 8c 7d	1c 7e 99 7e	06 7f 64 7f	b5 7f fc 7f	fc 7f fc 7f	fb 7f fb 7f	H[&[æ].).~.~.~.~.~.~.~.~.	
00070	fb 7f fb 7f	fa 7f fa 7f	fa 7f f9 7f	f9 7f f9 7f	f8 7f f8 7f	f8 7f f8 7f	f7 7f f7 7f	ü.ü.ü.ü.ü.ü.ü.ü.ü.ü.ü.ü.	
0008c	f6 7f f6 7f	f6 7f f5 7f	f5 7f f4 7f	f4 7f f4 7f	f3 7f f3 7f	f2 7f f2 7f	f1 7f f1 7f	ö.ö.ö.ö.ö.ö.ö.ö.ö.ö.ö.ö.	
000a8	f0 7f f0 7f	ef 7f ef 7f	ee 7f ee 7f	ed 7f ed 7f	ec 7f ec 7f	eb 7f ea 7f	ea 7f e9 7f	ö.ö.ï.ï.ï.ï.ï.ï.ï.ï.ï.ï.	
000c4	e8 7f e8 7f	e7 7f e6 7f	e6 7f e5 7f	e4 7f e4 7f	e3 7f e2 7f	e2 7f e1 7f	e0 7f df 7f	è.è.ç.æ.æ.ä.ä.ä.ä.ä.ä.ä.ä.	
000e0	df 7f de 7f	dd 7f dc 7f	dc 7f db 7f	da 7f d9 7f	d8 7f d8 7f	d7 7f d6 7f	d5 7f d4 7f	ß.p.Y.Ü.Ü.Ü.Ü.Ü.Ü.Ü.Ü.Ü.Ü.	
000fc	d3 7f d2 7f	d1 7f d0 7f	d0 7f cf 7f	ce 7f cd 7f	cc 7f cb 7f	ca 7f c9 7f	c8 7f c7 7f	ö.ö.ö.ö.ö.ö.ö.ö.ö.ö.ö.ö.	
00118	c6 7f c5 7f	c4 7f c3 7f	c2 7f c1 7f	c0 7f bf 7f	be 7f bd 7f	bb 7f ba 7f	b9 7f b8 7f	Ä.Ä.Ä.Ä.Ä.Ä.Ä.Ä.Ä.Ä.Ä.Ä.	
00134	b7 7f b6 7f	b5 7f b4 7f	b2 7f b1 7f	b0 7f af 7f	ae 7f ac 7f	ab 7f aa 7f	a9 7f a7 7f	..µ.µ.'.'.'.'.'.'.'.'.'.'.	
00150	a6 7f a5 7f	a4 7f a2 7f	a1 7f a0 7f	9e 7f 9d 7f	9c 7f 9b 7f	99 7f 98 7f	96 7f 95 7f].µ.µ.é.j.	
0016c	94 7f 92 7f	91 7f 8f 7f	8e 7f 8d 7f	8b 7f 8a 7f	88 7f 87 7f	85 7f 84 7f	82 7f 81 7f	
00188	7f 7f 7e 7f	7c 7f 7b 7f	79 7f 78 7f	76 7f 74 7f	73 7f 71 7f	70 7f 6e 7f	6c 7f 6b 7f	..~.].{.y.x.v.t.s.q.p.n.l.k.	

Hexadecimal 2. Coseno Inestable Filtrado (Memoria utilizada = 30)

00000	52 49 46 46	be 75 0d 00	57 41 56 45	66 6d 74 20	10 00 00 00	01 00 01 00	44 ac 00 00	RIFF%u..wAvefmt	D~..
0001c	88 58 01 00	02 00 10 00	64 61 74 61	50 75 0d 00	0a 12 af 21	40 2f 05 3b	39 45 12 4e	.X.....dataPu..!@/;9E.N	
00038	be 55 66 5c	2b 62 2c 67	83 6b 47 6f	8a 72 5f 75	d3 77 f4 79	cc 7b 66 7d	ca 7e fe 7f	%Uf\+b.g.kGo.r_uöwöyI{f}É~p.	
00054	fd 7f fd 7f	fd 7f fd 7f	fd 7f fd 7f	fc 7f fc 7f	fc 7f fc 7f	fc 7f fb 7f	fb 7f fb 7f	y.y.y.y.y.ü.ü.ü.ü.ü.ü.ü.ü.	
00070	fb 7f fa 7f	fa 7f fa 7f	fa 7f f9 7f	f9 7f f9 7f	f8 7f f8 7f	f8 7f f7 7f	f7 7f f6 7f	ü.ü.ü.ü.ü.ü.ü.ü.ü.ü.ü.ü.	
0008c	f6 7f f6 7f	f5 7f f5 7f	f5 7f f4 7f	f4 7f f3 7f	f3 7f f2 7f	f2 7f f1 7f	f1 7f f0 7f	ö.ö.ö.ö.ö.ö.ö.ö.ö.ö.ö.ö.	
000a8	f0 7f ef 7f	ef 7f ee 7f	ee 7f ed 7f	ed 7f ec 7f	ec 7f eb 7f	ea 7f ea 7f	e9 7f e9 7f	ö.ï.ï.ï.ï.ï.ï.ï.ï.ï.ï.ï.ï.	
000c4	e8 7f e7 7f	e7 7f e6 7f	e5 7f e5 7f	e4 7f e3 7f	e2 7f e2 7f	e1 7f e0 7f	e0 7f df 7f	è.ç.ç.æ.ä.ä.ä.ä.ä.ä.ä.ä.ä.	
000e0	de 7f dd 7f	dd 7f dc 7f	db 7f da 7f	d9 7f d9 7f	d8 7f d7 7f	d6 7f d5 7f	d4 7f d3 7f	p.Y.Y.Ü.Ü.Ü.Ü.Ü.Ü.Ü.Ü.Ü.Ü.	
000fc	d3 7f d2 7f	d1 7f d0 7f	cf 7f ce 7f	cd 7f cc 7f	cb 7f ca 7f	c9 7f c8 7f	c7 7f c6 7f	ö.ö.N.ö.ï.ï.ï.ï.ï.ï.ï.ï.ï.	
00118	c5 7f c4 7f	c3 7f c2 7f	c1 7f c0 7f	bf 7f be 7f	bd 7f bc 7f	bb 7f ba 7f	b8 7f b7 7f	A.Ä.Ä.Ä.Ä.Ä.Ä.Ä.Ä.Ä.Ä.Ä.	
00134	b6 7f b5 7f	b4 7f b3 7f	b2 7f b0 7f	af 7f ae 7f	ad 7f ab 7f	aa 7f a9 7f	a8 7f a6 7f	µ.µ.'.'.'.'.'.'.'.'.'.'.	
00150	a5 7f a4 7f	a3 7f a1 7f	a0 7f 9f 7f	9d 7f 9c 7f	9b 7f 9a 7f	98 7f 97 7f	95 7f 94 7f	µ.µ.é.j.	
0016c	93 7f 91 7f	90 7f 8e 7f	8d 7f 8b 7f	8a 7f 89 7f	87 7f 86 7f	84 7f 83 7f	81 7f 80 7f	
00188	7e 7f 7d 7f	7b 7f 7a 7f	78 7f 76 7f	75 7f 73 7f	72 7f 70 7f	6e 7f 6d 7f	6b 7f 6a 7f	~.}.{.z.x.v.u.s.r.p.n.m.k.j.	

Hexadecimal 3. Coseno Inestable Filtrado (Memoria utilizada = 20)

00000	52 49 46 46	be 75 0d 00	57 41 56 45	66 6d 74 20	10 00 00 00	01 00 01 00	44 ac 00 00	RIFF%u..wavefmt	D~..
0001c	88 58 01 00	02 00 10 00	64 61 74 61	50 75 0d 00	ff 10 bc 1f	84 2c 9a 37	37 41 8d 49	.X.....dataPu..y.%..	..77A.I
00038	c8 50 0d 57	7d 5c 34 61	4b 65 d6 68	e9 6b 94 6e	e4 70 e5 72	a2 74 24 76	73 77 95 78	Ep.w)\4ake0hék.napárett\$vsu.x	
00054	91 79 6b 7a	29 7b cd 7b	5b 7c d6 7c	41 7d 9e 7d	ee 7d 34 7e	70 7e a5 7e	d2 7e f9 7e	'ykz){i{[0 A.}i}4~p~W~0~ú~	
00070	1b 7f 39 7f	52 7f 68 7f	7b 7f 8c 7f	9a 7f a6 7f	b1 7f ba 7f	c2 7f c9 7f	cf 7f d4 7f	..9.R.h.{..... ±.º.Á.É.Í.Ó.	
0008c	d8 7f dc 7f	df 7f e2 7f	e4 7f e6 7f	e7 7f e8 7f	e9 7f ea 7f	eb 7f eb 7f	ec 7f ec 7f	0.Ü.ß.ä.ä.æ.ç.ë.é.ë.ë.ë.í.í.	
000a8	ec 7f ec 7f	ec 7f ec 7f	ec 7f ec 7f	ec 7f eb 7f	eb 7f ea 7f	ea 7f ea 7f	e9 7f e9 7f	í.í.í.í.í.í.í.í.é.é.é.é.é.é.	
000c4	e8 7f e7 7f	e7 7f e6 7f	e6 7f e5 7f	e4 7f e4 7f	e3 7f e2 7f	e2 7f e1 7f	e0 7f e0 7f	è.ç.ç.æ.æ.ä.ä.ä.ä.ä.ä.ä.ä.	
000e0	df 7f de 7f	dd 7f dd 7f	dc 7f db 7f	da 7f da 7f	d9 7f d8 7f	d7 7f d6 7f	d5 7f d4 7f	ß.p.ý.ý.Ü.Ü.Ü.Ü.Ü.Ü.Ü.Ü.Ö.Ö.Ö.	
000fc	d3 7f d3 7f	d2 7f d1 7f	d0 7f cf 7f	ce 7f cd 7f	cc 7f cb 7f	cb 7f c9 7f	c8 7f c7 7f	Ó.Ó.Ö.Ö.Ü.Ü.í.í.í.í.É.É.É.É.Ç.	
00118	c6 7f c5 7f	c4 7f c3 7f	c2 7f c1 7f	c0 7f bf 7f	be 7f bd 7f	bc 7f bb 7f	ba 7f b9 7f	Æ.Ä.Ä.Ä.Ä.Ä.ä.ä.ä.ä.ä.ä.º.º.º.	
00134	b7 7f b6 7f	b5 7f b4 7f	b3 7f b2 7f	b0 7f af 7f	ae 7f ad 7f	ac 7f aa 7f	a9 7f a8 7f	..µ.µ.º.º.º.º.º.º.º.º.º.º.º.º.	
00150	a7 7f a5 7f	a4 7f a3 7f	a2 7f a0 7f	9f 7f 9e 7f	9c 7f 9b 7f	9a 7f 98 7f	97 7f 96 7f	\$.%..µ.é.é.é.	
0016c	94 7f 93 7f	91 7f 90 7f	8f 7f 8d 7f	8c 7f 8a 7f	89 7f 87 7f	86 7f 84 7f	83 7f 81 7f	
00188	80 7f 7e 7f	7d 7f 7b 7f	7a 7f 78 7f	77 7f 75 7f	74 7f 72 7f	70 7f 6f 7f	6d 7f 6b 7f	...~.}.f.z.x.w.u.t.r.p.o.m.k.	

Hexadecimal 4. Coseno Inestable Filtrado (Memoria utilizada = 100)

00000	52 49 46 46	be 75 0d 00	57 41 56 45	66 6d 74 20	10 00 00 00	01 00 01 00	44 ac 00 00	RIFF%u..wavefmt	D~..
0001c	88 58 01 00	02 00 10 00	64 61 74 61	50 75 0d 00	ff 10 bc 1f	84 2c 9a 37	37 41 8d 49	.X.....dataPu..y.%..	..77A.I
00038	c8 50 0d 57	7d 5c 34 61	4b 65 d6 68	e9 6b 94 6e	e4 70 e5 72	a2 74 24 76	73 77 95 78	Ep.w)\4ake0hék.napárett\$vsu.x	
00054	91 79 6b 7a	29 7b cd 7b	5b 7c d6 7c	41 7d 9e 7d	ee 7d 34 7e	70 7e a5 7e	d2 7e f9 7e	'ykz){i{[0 A.}i}4~p~W~0~ú~	
00070	1b 7f 39 7f	52 7f 68 7f	7b 7f 8c 7f	9a 7f a6 7f	b1 7f ba 7f	c2 7f c9 7f	cf 7f d4 7f	..9.R.h.{..... ±.º.Á.É.Í.Ó.	
0008c	d8 7f dc 7f	df 7f e1 7f	e4 7f e6 7f	e7 7f e8 7f	e9 7f ea 7f	eb 7f eb 7f	ec 7f ec 7f	0.Ü.ß.ä.ä.æ.ç.ë.é.ë.ë.ë.í.í.	
000a8	ec 7f ec 7f	ec 7f ec 7f	ec 7f ec 7f	eb 7f eb 7f	eb 7f ea 7f	ea 7f ea 7f	e9 7f e9 7f	í.í.í.í.í.í.í.í.é.é.é.é.é.é.	
000c4	e8 7f e7 7f	e7 7f e6 7f	e6 7f e5 7f	e4 7f e4 7f	e3 7f e2 7f	e2 7f e1 7f	e0 7f e0 7f	è.ç.ç.æ.æ.ä.ä.ä.ä.ä.ä.ä.ä.	
000e0	df 7f de 7f	dd 7f dd 7f	dc 7f db 7f	da 7f da 7f	d9 7f d8 7f	d7 7f d6 7f	d5 7f d4 7f	ß.p.ý.ý.Ü.Ü.Ü.Ü.Ü.Ü.Ü.Ü.Ö.Ö.Ö.	
000fc	d3 7f d3 7f	d2 7f d1 7f	d0 7f cf 7f	ce 7f cd 7f	cc 7f cb 7f	cb 7f c9 7f	c8 7f c7 7f	Ó.Ó.Ö.Ö.Ü.Ü.í.í.í.í.É.É.É.É.Ç.	
00118	c6 7f c5 7f	c4 7f c3 7f	c2 7f c1 7f	c0 7f bf 7f	be 7f bd 7f	bc 7f bb 7f	ba 7f b9 7f	Æ.Ä.Ä.Ä.Ä.Ä.ä.ä.ä.ä.ä.ä.º.º.º.	
00134	b7 7f b6 7f	b5 7f b4 7f	b3 7f b2 7f	b0 7f af 7f	ae 7f ad 7f	ac 7f aa 7f	a9 7f a8 7f	..µ.µ.º.º.º.º.º.º.º.º.º.º.º.º.	
00150	a7 7f a5 7f	a4 7f a3 7f	a2 7f a0 7f	9f 7f 9e 7f	9c 7f 9b 7f	9a 7f 98 7f	97 7f 96 7f	\$.%..µ.é.é.é.	
0016c	94 7f 93 7f	91 7f 90 7f	8f 7f 8d 7f	8c 7f 8a 7f	89 7f 87 7f	86 7f 84 7f	83 7f 81 7f	
00188	80 7f 7e 7f	7d 7f 7b 7f	7a 7f 78 7f	77 7f 75 7f	74 7f 72 7f	70 7f 6f 7f	6d 7f 6b 7f	...~.}.f.z.x.w.u.t.r.p.o.m.k.	

Hexadecimal 5. Coseno Inestable Filtrado (Memoria utilizada = 1000)

Al ver los archivos abiertos en Frhed, se puede notar claramente en la parte resaltada en amarillo que los archivos no son los mismos, a excepción de los Hexadecimales 4 y 5, que son señales idénticas.

Así que nuestra aseveración era correcta, las señales no son idénticas, sin embargo, es algo complicado saber en que difieren, ya que hay que observarlo en un editor hexadecimal, por lo tanto, aumentar la memoria es aumentar exponencialmente las operaciones que realiza el CPU, por un beneficio que a simple vista no es muy notorio, así que, personalmente, con una memoria de 30 es suficiente y no se realizan tantas operaciones, ya que son 3 ciclos anidados:

- El ciclo para ir recorriendo todos los datos (aproximadamente 88,200 iteraciones)
- El ciclo para desplazar los elementos hacia la derecha (se realiza 20 veces por cada iteración del ciclo que recorre los datos, así que se realiza 1,764,000 veces en total).
- El ciclo que ejecuta el algoritmo de la convolución discreta que al igual que el anterior, se ejecuta 1,764,000 veces.

Podemos notar que son demasiadas operaciones, y eso es para 20 unidades de memoria, si lo incrementamos a 30, las operaciones del segundo y tercer ciclo se realizarían 2,646,000 veces. Con una memoria de 100, se realizan 8,820,000 veces, y si la memoria es de 1,000, las operaciones se realizan 88, 200, 000 veces.

Si consideramos que una computadora puede procesar 1,000,000,000 de operaciones por segundo, para una memoria de 20, la ejecución del puro algoritmo (sin considerar copiar la cabecera, generar el impulso, etc.), se tardaría aproximadamente 0.00361 segundos. Para una memoria de 30, la ejecución tardaría 0.00538 segundos, para memoria de 100, la ejecución tardaría aproximadamente 0.01772 segundos y para de 1000, la ejecución tardaría aproximadamente 0.1764 segundos.

Como se puede apreciar, los tiempos de ejecución no parecen demasiado, pero poniéndolo en perspectiva, el tamaño más grande que se probó con estas señales fue de 1 segundo, si al filtro le

metemos un archivo WAV que dure 3 minutos (aproximadamente lo que dura una canción), el tiempo de ejecución aumentaría a 31.7678 segundos con una memoria utilizada de 1,000, que es un tiempo de cómputo demasiado grande. Sin embargo, para 20, el tiempo de ejecución aumentaría a 0.6509 segundos, que es un tiempo bastante aceptable, inclusive para 30, el tiempo de ejecución estimado es de 0.9684 segundos, que sigue siendo un tiempo bastante bueno.

Por lo tanto, la cantidad de memoria ideal se encuentra entre 20 y 30, para obtener una mejor relación de costo – beneficio (tomando en cuenta tiempo computacional y calidad de la señal de salida).

Conclusiones:

El diseño de filtros digitales es algo interesante, sin mencionar que la era de las computadoras está en pleno auge, debido a que están en todos lados. Además, a diario escuchamos música, ya sea en nuestro automóvil, en el radio, etc. Por lo que la manipulación de este tipo de archivos resulta interesante para poder experimentar y pensar distintas aplicaciones con ellos.

En el caso del filtro pasa bajas, como vimos en la señal del coseno inestable, al pasarla por el filtro, se volvió una señal estable, eliminando la peligrosidad de una señal que crece sin parar.

Una aplicación de este tipo de filtros, puede ser un ecualizador musical, que modifica el volumen del contenido en frecuencias de la señal de entrada, por lo que, podemos eliminar o añadir decibeles en determinadas frecuencias (en este caso, eliminar decibeles por encima de la frecuencia de corte), para compensar posibles errores en la grabación de un audio profesional.

Realizar la convolución en lenguaje C, es un algoritmo demasiado costoso computacionalmente hablando, además de que no puede realizarse en el tiempo continuo debido a las limitaciones que tiene cualquier computadora ya que contamos con una memoria y tiempo de procesamiento finitos, por lo tanto los resultados obtenidos son buenos, pero no excelentes, las aplicaciones que tiene, no solo en el procesamiento de señales, son bastante amplias e interesantes, algunas de ellas es el procesamiento de imágenes, específicamente el filtrado en las mismas, en acústica, ya que un eco es la convolución del sonido original con una función que represente los objetos variados que lo reflejan.

Referencias:

- [1] Clavijo Mendoza Juan Ricardo, ‘Diseño y Simulación de Sistemas Microcontrolados en lenguaje C’. Colombia: 2011, pp. 190 – 193.
- [2] Department of Mechanical Engineering, ‘Understanding Poles and Zeros’. [Online]. Disponible en: <http://web.mit.edu/2.14/www/Handouts/PoleZero.pdf>. [Accedido: 11 – septiembre – 2017].
- [3] Wikipedia, ‘Circuito RC’. [Online]. Disponible en: https://es.wikipedia.org/wiki/Circuito_RC. [Accedido: 11 – septiembre – 2017].
- [4] Electrónica Unicrom, ‘Filtro RC Paso Bajo’. [Online]. Disponible en: <https://unicrom.com/filtro-rc-paso-bajo/>. [Accedido: 11 – septiembre – 2017].

[5] Romero Gamarra Joel Mauricio, 'Bajar Volumen a un Archivo WAV'. [Online]. Disponible en: <https://github.com/JoelRomero97/Teoria-de-Comunicaciones-y-Senales/blob/master/Prácticas/Bajar%20Volumen/Reporte.docx>.

Código

Filtro.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include "Cabecera.c"
#define TAM_ARREGLO 20

int main(int argc, char const *argv[])
{
    FILE * archivoEntrada, * archivoSalida;
    cabecera cab;
    int i, lectura, escritura, j = 0, k;
    short muestra;
    float * impulso = (float *) malloc (sizeof (float) * 20);
    float * entrada = (float *) malloc (sizeof (float) * 20);
    char * nombreModificado = (char *) malloc (sizeof (char));
    char * nombreArch = (char *) malloc (sizeof (char));
    system ("cls");
    if (argc < 3)
        printf("Error, faltan argumentos.\n");
    else
    {
        nombreArch = (char *) argv [1];
        nombreModificado = (char *) argv [2];

        //Abrimos los archivos en modo binario
        archivoEntrada = abreArchivo (nombreArch, nombreModificado, 1);
        archivoSalida = abreArchivo (nombreArch, nombreModificado, 2);

        //Leemos e imprimimos la cabecera del archivo wav
        leerCabecera (archivoEntrada, archivoSalida, &cab);

        //Generamos la respuesta al impulso
        impulso = generaImpulso ();
        float max = 0;
        for (i = 0; i < TAM_ARREGLO; i++)
            max += (impulso [i] * 32767);

        //Llenamos el arreglo de entrada con puros ceros
        entrada = llenaArreglo ();

        //Escribimos el resto de los datos realizando la convolución
        for (i = 0; i < cab.SubChunk2Size; i++, j++)
        {
            lectura = fread(&muestra, sizeof (short), 1, archivoEntrada);
            for (k = (TAM_ARREGLO - 1); k >= 0; k--)
                entrada [k] = entrada [k - 1];
            entrada [0] = (muestra / max);

            //Insertamos los datos en el arreglo
            muestra = convolucion (entrada, impulso);
            escritura = fwrite(&muestra, sizeof (short), lectura, archivoSalida);
            //Escribimos los datos nuevos en el archivo
        }
        printf ("\n\n");
    }
}
```

```

        fclose (archivoEntrada);
        fclose (archivoSalida);
        printf ("Archivo '%s' modificado correctamente.\n", nombreArch);
        return 0;
    }

```

Cabecera.c

```

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "Cabecera.h"
#define PI 3.14159265
#define FREQ_CORTE 1000
#define TAM_ARREGLO 20

int i; //Variable global para manejar ciclos

FILE * abreArchivo (char * nombreArch, char * nombreModificado, int tipo)
{
    FILE * pt1, * pt2;
    pt1 = fopen (nombreArch, "rb");
    if (pt1 == NULL)
    {
        printf("Error al abrir archivo '%s'.\n", nombreArch);
        exit(0);
    }

    //Abrimos el archivo a escribir en modo binario
    pt2 = fopen (nombreModificado, "wb");
    if (pt2 == NULL)
    {
        printf("Error al crear el archivo '%s'.\n", nombreModificado);
        exit(1);
    }
    if (tipo == 1)
    {
        printf("Archivo '%s' abierto correctamente.\n", nombreArch);
        return pt1;
    }
    else
    {
        printf("Archivo '%s' creado correctamente.\n", nombreModificado);
        return pt2;
    }
}

void leerCabecera (FILE * archivoEntrada, FILE * archivoSalida, cabecera * cab)
{
    int lectura, escritura;
    unsigned char buffer4[4], buffer2 [2], * formatoArchivo;
    formatoArchivo = (char *) malloc (sizeof (char));

    //ChunkID
    lectura = fread(cab -> ChunkID, sizeof(cab -> ChunkID), 1, archivoEntrada);
    printf("\n\n(1-4) Chunk ID: %s\n\n", cab -> ChunkID);
    escritura = fwrite (cab -> ChunkID, sizeof (cab -> ChunkID), 1, archivoSalida);

    //ChunkSize
    //Se convierten los datos a entero
    lectura = fread(buffer4, sizeof(buffer4), 1, archivoEntrada);
    escritura = fwrite(buffer4, sizeof(buffer4), 1, archivoSalida);
    cab -> ChunkSize = (buffer4[0] | (buffer4[1]<<8) | (buffer4[2]<<16) |
(buffer4[3]<<24));
    printf("(5-8) ChunkSize: %u\n\n", cab -> ChunkSize);

    //Formato "Fmt"

```

```

lectura = fread(cab -> Format, sizeof(cab -> Format),1,archivoEntrada);
printf("(9-12) Format: %s\n\n",cab -> Format);
escritura = fwrite (cab -> Format, sizeof (cab -> Format), 1, archivoSalida);

//SubChunk1ID Formato de datos "fmt"
lectura = fread(cab -> SubChunk1ID,sizeof(cab -> SubChunk1ID),1,archivoEntrada);
printf("(13-16) SubChunk 1 ID: %s\n\n",cab -> SubChunk1ID);
escritura = fwrite (cab -> SubChunk1ID, sizeof (cab -> SubChunk1ID), 1,
archivoSalida);

//SubChunk1Size
lectura = fread(buffer4, sizeof(buffer4),1,archivoEntrada);
escritura = fwrite(buffer4, sizeof(buffer4),1,archivoSalida);
cab -> SubChunk1Size = buffer4[0] | (buffer4[1]<<8) | (buffer4[1]<<16) |
(buffer4[1]<<24);
printf("(17-20) SubChunk 1 Size: %u\n\n",cab -> SubChunk1Size);

// Formato de audio
lectura = fread(buffer2, sizeof(buffer2),1,archivoEntrada);
escritura = fwrite(buffer2, sizeof(buffer2),1,archivoSalida);
cab -> AudioFormat = buffer2[0] | (buffer2[1]<<8);
if (cab -> AudioFormat == 1)
    strcpy(formatoArchivo,"PCM");
printf("(21-22) Audio Format: %u,%s\n\n",cab -> AudioFormat,formatoArchivo);

//Canales
lectura = fread(buffer2, sizeof(buffer2),1,archivoEntrada);
escritura = fwrite(buffer2, sizeof(buffer2),1,archivoSalida);
cab -> NumChannels = buffer2[0] | (buffer2[1]<<8);
if (cab -> NumChannels == 1)
    strcpy(formatoArchivo,"Mono");
else
    strcpy(formatoArchivo,"Stereo");
printf("(23-24) Number of Channels: %u, Tipo: %s\n\n",cab ->
NumChannels,formatoArchivo);

//SampleRate
lectura = fread(buffer4, sizeof(buffer4),1,archivoEntrada);
escritura = fwrite(buffer4, sizeof(buffer4),1,archivoSalida);
cab -> SampleRate = buffer4[0] | (buffer4[1]<<8) | (buffer4[2]<<16) |
(buffer4[3]<<24);
printf("(25-28) Sample Rate: %u\n\n",cab -> SampleRate);

//ByteRate
lectura = fread(buffer4, sizeof(buffer4),1,archivoEntrada);
escritura = fwrite(buffer4, sizeof(buffer4),1,archivoSalida);
cab -> ByteRate = buffer4[0] | (buffer4[1]<<8) | (buffer4[2]<<16) |
(buffer4[3]<<24);
printf("(29-32) Byte Rate: %u BitRate: %u\n\n",cab -> ByteRate,cab -> ByteRate*8);

//Block Align
lectura = fread(buffer2, sizeof(buffer2),1,archivoEntrada);
escritura = fwrite(buffer2, sizeof(buffer2),1,archivoSalida);
cab -> BlockAlign = buffer2[0] | (buffer2[1]<<8);
printf("(33-34) Block Align: %u\n\n",cab -> BlockAlign);

//Bits per Sample
lectura = fread(buffer2, sizeof(buffer2),1,archivoEntrada);
escritura = fwrite(buffer2, sizeof(buffer2),1,archivoSalida);
cab -> BitsPerSample = buffer2[0] | (buffer2[1]<<8);
printf("(35-36) Bits Per Sample: %u\n\n",cab -> BitsPerSample);

//SubChunk2ID
lectura = fread(cab -> SubChunk2ID,sizeof(cab -> SubChunk2ID),1,archivoEntrada);
printf("(37-40) SubChunk 2 ID: %s\n\n",cab -> SubChunk2ID);
escritura = fwrite(cab -> SubChunk2ID,sizeof(cab -> SubChunk2ID),1,archivoSalida);

```

```

        //SubChunk2Size
        lectura = fread(buffer4, sizeof(buffer4), 1, archivoEntrada);
        escritura = fwrite(buffer4, sizeof(buffer4), 1, archivoSalida);
        cab -> SubChunk2Size = buffer4[0] | (buffer4[1]<<8) | (buffer4[2]<<16) |
(buffer4[3]<<24);
        printf("(41-44) SubChunk 2 Size: %u\n\n", cab -> SubChunk2Size);
    }

float * llenaArreglo ()
{
    float * entrada = (float *) malloc (sizeof (float) * TAM_ARREGLO);
    for (i = 0; i < TAM_ARREGLO; i++)
        entrada [i] = 0;
    return entrada;
}

float * generaImpulso ()
{
    float * impulso = (float *) malloc (sizeof (float) * TAM_ARREGLO);
    for (i = 0; i < TAM_ARREGLO; i++)
    {
        impulso [i] = (exp ((-2 * PI * FREC_CORTE * i) / 44100));
        //Fórmula de filtro RC con fc = 1,000 Hz
    }
    return impulso;
}

float convolucion (float * entrada, float * impulso)
{
    float respuesta = 0;

    //Realizamos la convolución
    for (i = 0; i < TAM_ARREGLO; i++)
        if (entrada [i] != 0)
            respuesta += (entrada [i] * impulso [i]);

    //Como entran valores entre -1 y 1, comprobamos la respuesta, para saber que
    enviamos (aun así todos los multiplicamos por 32767 que es
    //el valor máximo que puede tomar un short)
    if (respuesta > 1)
        return (1 * 32767);
    if (respuesta < -1)
        return (-1 * 32767);
    else
        return (respuesta * 32767);
}

```

Cabecera.h

```

typedef struct CABECERA
{
    char ChunkID[4]; //Contiene las 'RIFF'
    unsigned int ChunkSize; //Contiene el tamaño
total sin contar este y el segmento anterior (8 bytes)
    unsigned char Format[4]; //Contiene 'WAVE'

    //Aquí comienza el primer subchunk 'fmt'
    char SubChunk1ID[4]; //Contiene 'fmt'
    unsigned int SubChunk1Size; //Contiene el tamaño del resto de
el primer subchunk
    unsigned int AudioFormat; //Formato de audio, es es distinto
de 1, es forma de compresión
    unsigned int NumChannels; //Numero de canales, mono = 1,
estereo = 2, etc.
    unsigned int SampleRate; //8000, 44100, etc.
}

```

```

        unsigned int ByteRate;                                //(SampleRate * Numero
canales * Bits per Sample) / 8
        unsigned int BlockAlign;                             //(Numero canales * Bits per
Sample) / 8
        unsigned int BitsPerSample;                          //8 bits, 16 bits, etc.

        //Aqui comienza el segundo subchunk 'data'
        char SubChunk2ID[4];                                  //Contiene 'data'
        unsigned int SubChunk2Size;                           //Numero de bytes en los datos, es
decir, bytes despues de este segmento
        unsigned int data;                                    //Datos de sonidos reales

}cabecera;

FILE * abreArchivo (char * nombreArch, char * nombreModificado, int tipo);
//Para abrir los archivos de entrada y salida
void leerCabecera (FILE * archivoEntrada, FILE * archivoSalida, cabecera * cab);
//Para copiar e imprimir la cabecera
float * llenaArreglo ();
float * generaImpulso ();
float convolucion (float * entrada, float * impulso);

```