PRÁCTICA 2

SIMULACIÓN DE UN CIRCUITO RC

Teoría de comunicaciones y señales

Profesor: Gutiérrez Aldana Eduardo

Alumno:

Amador Nava Miguel Ángel

Grupo: 3CM8

Introducción

En esta práctica implementaremos un filtro pasa bajas, simulando un circuito RC del cual obtendremos su función de transferencia y de esta obtendremos la respuesta al impulso la cual a mayor número de muestras nos dará mayor fiabilidad.

Al hacer la convolución de la respuesta al impulso con una señal la cual obtendremos de un archivo wav obtendremos la señal que saldría del circuito RC.

Objetivos

Generales: Simular un circuito RC

Específicos:

- Implementar la convolución.
- Simular un filtro pasa-bajas a 500 Hz.

Software y equipo utilizado

- Ubuntu 17.10
- Wine 1.7
- GoldWave 4.26
- Sublime Text 3

Marco teórico

Sistema lineal invariante en el tiempo

Linealidad

Un sistema es lineal si satisface el principio de superposición, que engloba las propiedades de proporcionalidad o escalado y aditividad. Que sea proporcional significa que cuando la entrada de un sistema es multiplicada por un factor, la salida del sistema también será multiplicada por el mismo factor. Por otro lado, que un sistema sea aditivo significa que si la entrada es el resultado de la suma de dos entradas, la salida será la resultante de la suma de las salidas que producirían cada una de esas entradas individualmente.

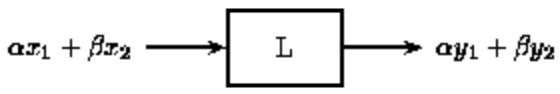


Ilustración 1 Principio de linealidad

Invariabilidad

Un sistema es invariante con el tiempo si su comportamiento y sus características son fijas. Esto significa que los parámetros del sistema no van cambiando a través del tiempo y que por lo tanto, una misma entrada nos dará el mismo resultado en cualquier momento (ya sea ahora o después.

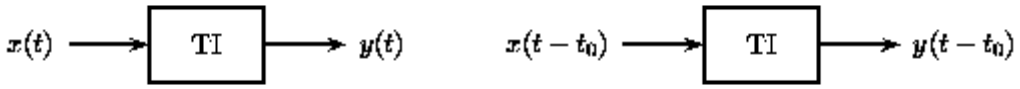


Ilustración 2 Sistema invariante en el tiempo

Si se tiene un sistema lineal invariante en el tiempo se puede conocer su salida a cualquier entrada mediante la convolución de la entrada y la respuesta al impulso del sistema.

Si se observa esto desde la transformada de Laplace, la convolución se convierte en una multiplicación en el dominio s.

$$Y(s) = F(s)H(s)$$

Donde $H(s)$ $se\ conoce\ como\ la\ función\ de\ transferencia\ del\ sistema$

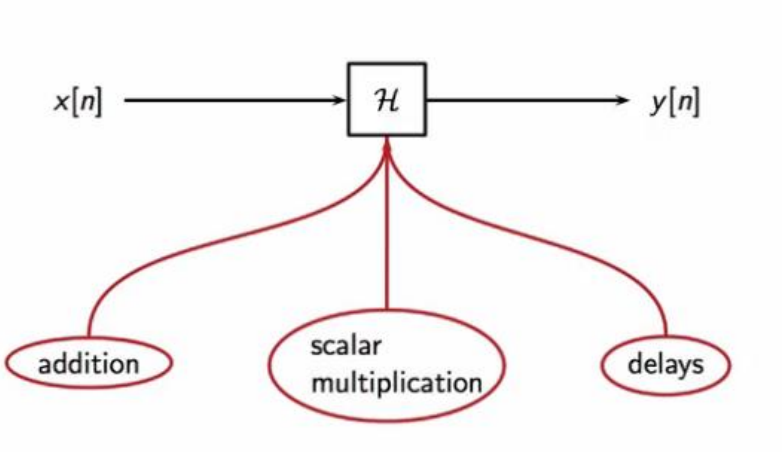


Ilustración 3 Componentes de un sistema lineal invariante en el tiempo

En este caso la salida es la función lineal de los valores pasados de entrada y los valores pasados de salida.

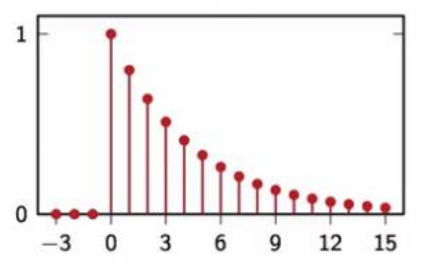
$$y[n] = H(x[n], x[n-1], x[n-2], \dots, y[n-1], y[n-2], \dots)$$

Respuesta al impulso

La respuesta al impulso es la salida de un filtro cuando la entrada es la función delta. Un resultado fundamental establece que la respuesta al impulso caracteriza por completo el comportamiento de un sistema lineal invariante en el tiempo (LTI).

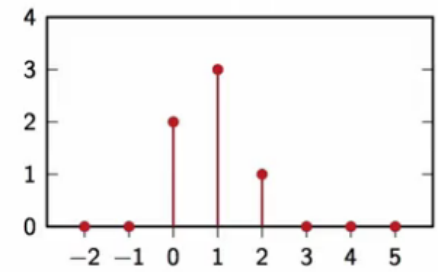
Supongamos que tenemos un filtro y podemos medir su respuesta al impulso ingresando una función delta y queremos usar el mismo filtro para filtrar una secuencia arbitraria $x[n]$ que es simplemente una secuencia de tres puntos donde es igual a 2 para n = 0, es igual a 3 para n = 1, y es igual a 1 para n = 2, y es 0 en cualquier otro lugar.

$$h[n]$$



$$h[n] = \alpha^n u[n]$$

$$x[n]$$



$x[n]$= 2 n=0
3 n=1
1 n=2
0 en otro caso

Siempre podemos escribir nuestra secuencia como una combinación lineal de función delta retardada.

$$x[n] = 2\delta[n] + 3\delta[n-1] + \delta[n-2]$$

Ya sabemos que la respuesta al impulso es $h[n] = H\{\delta[n]\}$

Y al explotar la linealidad y la invariancia en el tiempo, podemos calcular la respuesta del sistema a la secuencia de entrada $x[n] = H\{x[n]\}$

Al aplicar el filtro a la combinación lineal de deltas podemos dividir la operación del filtro sobre las tres componentes de la señal.

$$\begin{aligned} y[n] &= H\{2\delta[n] + 3\delta[n-1] + 4\delta[n-2]\} \\ &= 2H\{\delta[n]\} + 3H\{\delta[n-1]\} + H\{\delta[n-2]\} \\ &= 2h[n] + 3h[n-1] + h[n-2] \end{aligned}$$

De forma gráfica lo podemos ver de la siguiente manera:

Dos veces $\delta[n]$ y obtenemos 2 veces $h[n]$

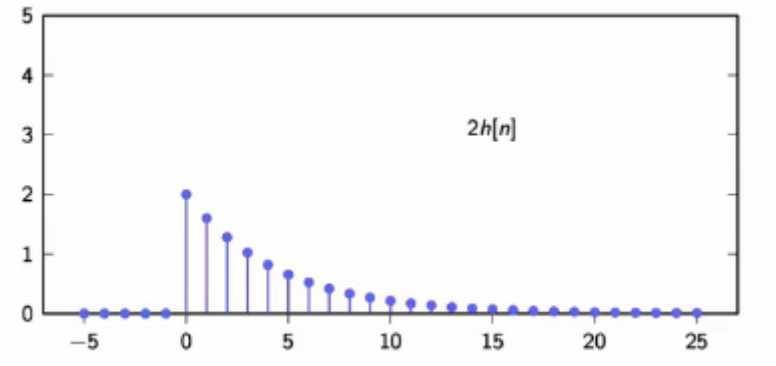


Ilustración 4 Filtrado del primer componente

Tres veces $\delta[n-1]$ y obtenemos 3 veces $h[n-1]$

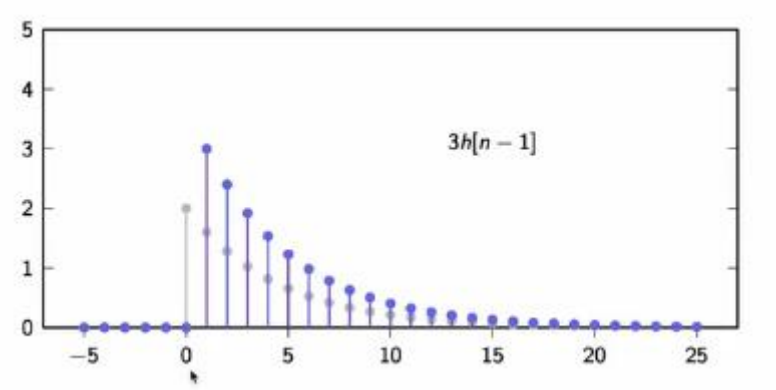


Ilustración 5 Filtrado del segundo componente

$\delta[n-2]$ y obtenemos $h[n-2]$

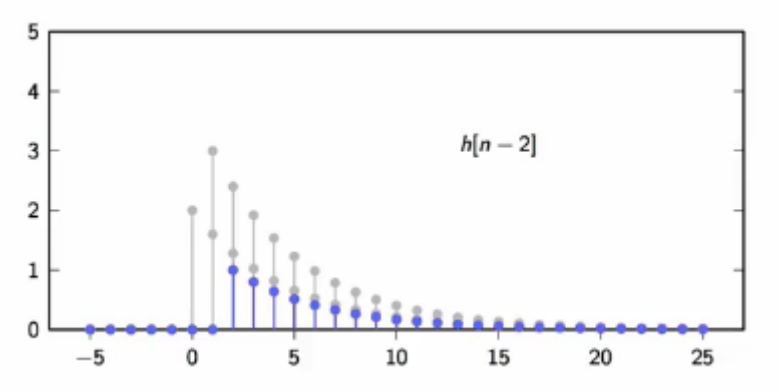


Ilustración 6 Filtrado del tercer componente

En la ilustración 7 podemos ver que salida del sistema es la suma de los 3 componentes

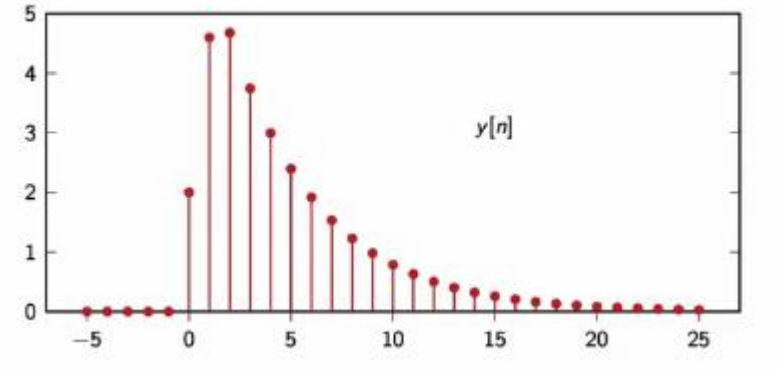


Ilustración 7 Respuesta del sistema a nuestra entrada arbitraria

Convolución Discreta

Representa la salida de un filtro dada su respuesta al impulso y una secuencia de entrada arbitraria $x[n]$. (Es la sumatoria realizada en el subtema anterior: respuesta al impulso).

$$y[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k]$$

$$= x[n] * h[n]$$

Circuito RC

Un circuito RC es un circuito eléctrico compuesto de resistencias y condensadores. La forma más simple de circuito RC es el circuito RC de primer orden, compuesto por una resistencia y un condensador. Los circuitos RC pueden usarse para filtrar una señal alterna, al bloquear ciertas frecuencias y dejar pasar otras. Los filtros RC más comunes son el filtro paso alto, filtro paso bajo, filtro paso banda, y el filtro de rechazo de banda. Entre las características de los circuitos RC está la de ser sistemas lineales e invariantes en el tiempo.

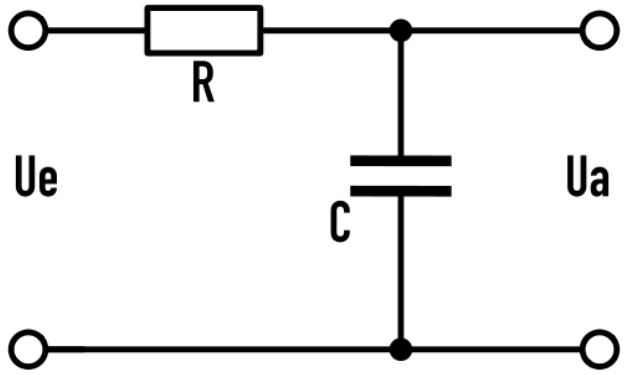


Ilustración 8 Circuito RC de primer orden

Frecuencia de corte

La frecuencia, bien por arriba o bien por debajo de la cual el nivel de salida de un circuito, tal como una línea, amplificador o filtro se reduce por un factor de "raíz de dos partido de dos" al valor de - 3 dB = 50% de la potencia respecto al nivel de referencia de 0 dB = 100%

En un circuito RC está dada por:

$$F_c = \frac{1}{2\pi RC}$$

Función de transferencia

Analizando el circuito:

$$Ri(t) + \frac{1}{C}\int i(t)dt$$

$$V_a = \frac{V_e \frac{1}{C}\int i(t)dt}{Ri(t) + \frac{1}{C}\int i(t)dt}$$

Aplicando la transformada de Laplace

$$V_a = \frac{V_e \frac{1}{sC}}{R + \frac{1}{sC}} = \frac{V_e \frac{1}{sC}}{\frac{RsC + 1}{sC}} = \frac{V_e}{RsC + 1}$$

$$\frac{V_a}{V_1} = \frac{1}{RsC + 1}$$

Obteniendo la respuesta al impulso

$$L^{-1}\left\{\frac{1}{RsC + 1}\right\} = \frac{1}{RC} e^{-tRC}$$

Despejamos de la frecuencia de corte del circuito RC a RC para dejar la expresión en términos de fc:

$$RC = \frac{1}{2\pi F_c}$$

Sustituyendo RC

$$2\pi F_c e^{\frac{-t}{2\pi F_c}}$$

Filtros FIR

FIR (Respuestá Finita al Impulso)

El diseño de los filtros FIR, es relativamente simple y utiliza estructuras ya definidas para cada tipo de filtro. Los filtros pueden ser de cinco naturalezas: Pasa bajas, pasa altas, pasa bandas, rechaza banda, y multi banda. Para el tratamiento de los filtros se debe tener presente las siguientes consideraciones:

- La frecuencia de muestreo en hercios Fs, es equivalente a una frecuencia en radianes/segundo de 2π.
- La máxima frecuencia tratada por los filtros es Fs/2, o π Radianes.
- La relación que existe entre radianes y hercios es: W = 2πF, y F = W/2π.
- Las frecuencias de corte de los filtros se denotan como Wc, y Fc.
- La frecuencia de corte se calcula como Wc = 2 π Fc / Fs.
- La banda de transición del filtro se denota como dW en radianes/segundo y dF en hercios.
- El orden de los sistemas FIR se denota como M.
- Se recomienda usar un número impar para M.

Dada la información anterior sabemos que

$$W_c = \frac{2\pi F_c}{F_s}$$

Por lo cual

$$\frac{F_c}{F_s} = \frac{1}{2\pi RC}$$

Despejando RC

$$RC = \frac{F_s}{2\pi F_c}$$

Sustituyendo en la respuesta al impulso previamente obtenida:

$$\frac{1}{RC} e^{-tRC} = \frac{\boldsymbol{2\pi F_c}}{\boldsymbol{F_s}} \boldsymbol{e}^{-t\frac{\boldsymbol{2\pi F_c}}{\boldsymbol{F_s}}}$$

Preparación del entorno para el desarrollo de la práctica

GoldWave 4.26 fue creado para Windows por lo cual para poder ejecutarlos en Ubuntu 17.10 necesitaremos instalar wine.

Introduciremos los siguientes comandos:

sudo add-apt-repository ppa:ubuntu-wine/ppa

sudo apt-get update

sudo apt-get install wine1.7 winetricks

Una vez instalado wine para ejecutar GoldWave lo haremos de la siguiente manera:

WINEPREFIX~=/.wine32 wine "nombre del ejecutable"

Planteamiento del problema

Se desea simular un circuito RC pasa bajas con una frecuencia de corte de 500Hz

Para probar el programa se crearan 3 archivos:

1. Archivo 1
   - Archivo monoaural 44100 muestras por segundo de 0.016s de duración.
   - Modificar $f(x) = step(n - 10) - step(n - 11)$
2. Archivo 2
   - Archivo monoaural 44100 muestras por segundo de 0.016s de duración.
   - Modificar $f(x) = step(n - 10) - step(n - 40)$
3. Archivo 3
   - Archivo monoaural 44100 muestras por segundo de 10s de duración.
   - Modificar $Cos\left(2\pi t\left(exp\left(log(20) + \frac{n}{N} * 6.6\right)\right)\right)$

La respuesta al impuso de hará con 10, 20 30 muestras.

Análisis Teórico

Cabecera de un archivo wav

Bytes	Campo	Descripción
4	ChunkID	Contiene las letras “RIFF” en ASCII
4	ChunkSize	36 bytes más SubChunk2Size. Este es el tamaño entero del archivo en bytes menos 8
4	Format	Contiene las letras “WAVE”
4	Subchunk1ID	Contiene las letras “fmt”
4	Subchunk1Size	Este es el tamaño del resto del Subchunk
2	AudioFormat	Los valores distintos de 1 indican alguna forma de compresión.
2	NumChannels	Numero d canales: 1= mono, 2=Stereo
4	SampleRate	Número de muestras por segundo
4	ByteRate	SampleRate * NumChannels * BitsPerSample/8
2	BlockAlign	NumChannels * BitsPerSample/8. El número de bytes para una muestra incluyen todos los canales
2	BitsPerSample	Bits por muesta. 8,16,etc.
4	Subchunk2ID	Contiene las letras “data”
4	Subchunk2Size	NumSamples * NumChannels * BitsPerSample/8. Este es el número de bytes en la data
*	Data	Datos del sonido

Tabla 1 Información de la cabecera de un archivo wav

Respuesta al impulso

De la ecuación que obtuvimos de los filtros FIR sustituiremos los valores:

$$\frac{2\pi F_c}{F_s} e^{-t\frac{2\pi F_c}{F_s}}$$

Es importante mencionar que al hacer la convolucion los valores aumentaran y sobrepasaran el tamaño de un short, al multiplicar por una constante la señal se atenuara o se amplificara por lo cual solo dejaremos la respuesta al impulso de la siguiente manera:

$$e^{-t\frac{2\pi F_c}{F_s}} = e^{-t\frac{2\pi 500}{44100}}$$

Y haremos una regla de tres para la salida de la señal, donde el valor máximo de las muestras después de la convolución será igual a 32767 y cualquie muestra obtenida después de la convolucion será igual a x, entonces:

$$MuestraDeSalida = \frac{muestra\ despues\ de\ la\ convolución * 32767}{muestra\ maxima\ después\ de\ la\ convolución}$$

Aunque las muestras las generaremos en tiempo de ejecución en la tabla 2 se muestran los valores y en la ilustración 9 la gráfica de estos.

Fs	Fc	t	Resultado
44100	500	0	1
44100	500	1	0.931240142
44100	500	2	0.867208202
44100	500	3	0.807579089
44100	500	4	0.752050065
44100	500	5	0.70033921
44100	500	6	0.652183985
44100	500	7	0.607339907
44100	500	8	0.565579301
44100	500	9	0.526690148
44100	500	10	0.490475008
44100	500	11	0.456750016
44100	500	12	0.42534395
44100	500	13	0.396097361
44100	500	14	0.368861762
44100	500	15	0.34349888
44100	500	16	0.319879946
44100	500	17	0.297885046
44100	500	18	0.277402512
44100	500	19	0.258328355
44100	500	20	0.240565734
44100	500	21	0.224024468
44100	500	22	0.208620578
44100	500	23	0.194275856
44100	500	24	0.180917476
44100	500	25	0.168477616
44100	500	26	0.156893119
44100	500	27	0.14610517
44100	500	28	0.136059
44100	500	29	0.126703602

Ilustración 9 Valores de la respuesta al impulso con 30 muestras

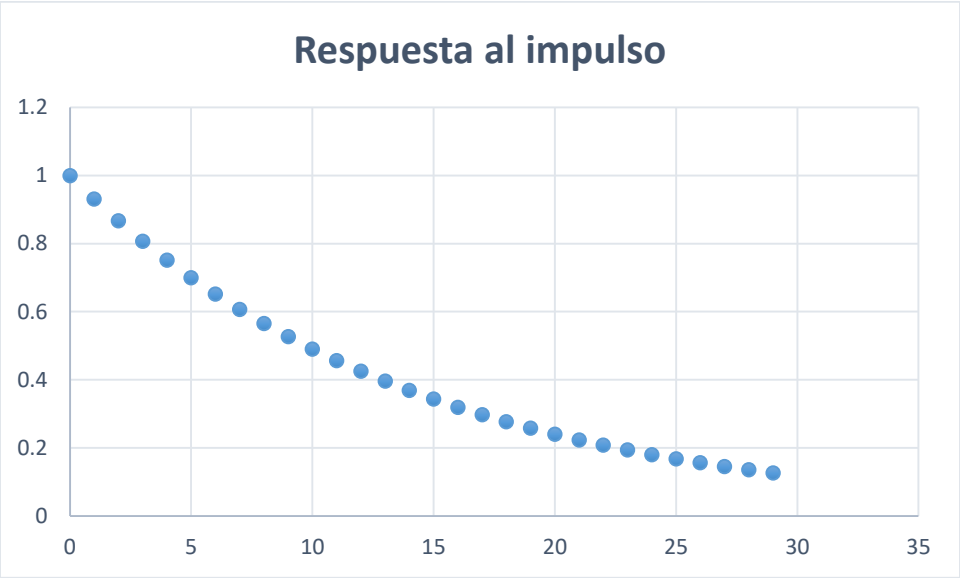


Ilustración 10 Grafica de la respuesta al impulso con 30 muestras

Análisis Práctico

Definiremos la frecuencia de corte, la frecuencia de muestro y a constante PI en nuestro archivo convolucion.c

```
#define PI 3.141592653589793
short Fc=500;//500Hz
unsigned short Fs=44100;//44100 muestras por segundo
```

Generaremos las muestras el cual el numero a generar la pasaremos como parametro:

```
double Constante = (2*PI*Fc)/Fs;
char numMuestras=atoi(argv[3]);
double *h= malloc(sizeof(double)*numMuestras);

            /*
            *     Calculamos las muestras para el h(n)
            */
for(char i=0;i<numMuestras;i++)h[i]=exp(-i*Constante);
```

La convolución queda de la siguiente manera:

```
//Convolucion
double *muestrasAudio_2=malloc(sizeof(double)*numMuestrasAudio);
for(int i=0;i<numMuestrasAudio;i++){
      for(int j=0;j<numMuestras;j++)
            muestrasAuxiliares[j]=h[j]*muestrasAudio[i];

      for(int p=0;p<numMuestras && i<numMuestrasAudio;p++)
            muestrasAudio_2[i+p]+=muestrasAuxiliares[p];
}
```

Para volver a dar la salida de nuestro archivo wav de -1 a 1 donde 1 = 32767 y -1=32767 que son los valores máximos del tipo de dato short lo haremos de la siguiente manera:

```
int valorMaximo=muestrasAudio_2[0];//buscamos la muestra mas grande
for(int i=1;i<numMuestrasAudio;i++){
      if(valorMaximo<muestrasAudio_2[i])
            valorMaximo=muestrasAudio_2[i];
}

for(int i=0;i<numMuestrasAudio;i++)//aplicamos la regla de 3
      muestrasAudio2[i]=muestrasAudio_2[i]*32767/valorMaximo;
```

Para compilar usaremos el siguiente comando:

ggc convolucion.c –o convolucion.exe -lm

Genrarmos las señales en GoldWave

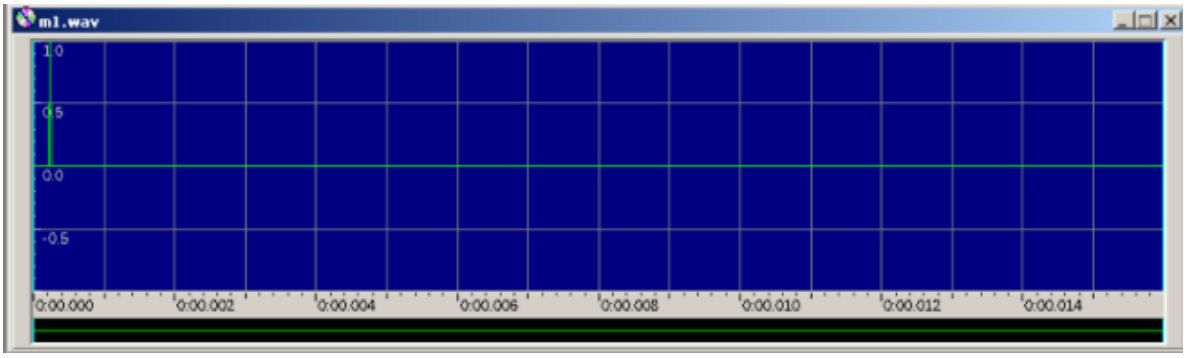


Ilustración 11 Imagen de la señal 1

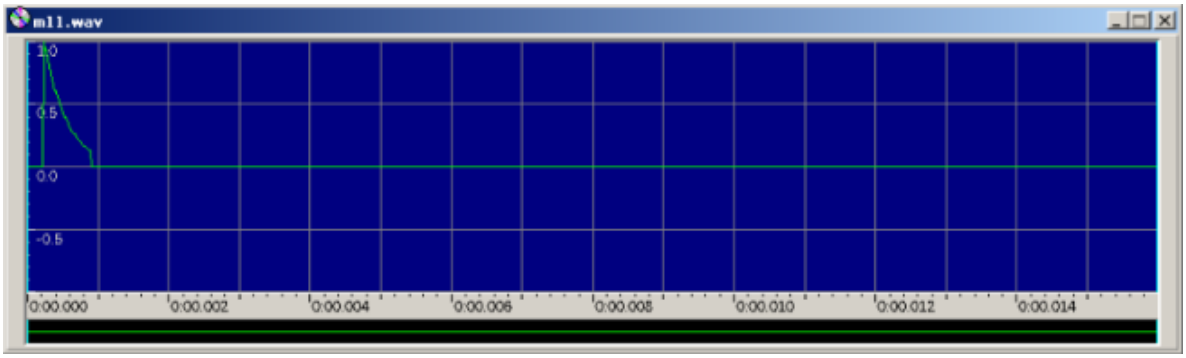


Ilustración 12 Imagen de la señal 1 filtrada

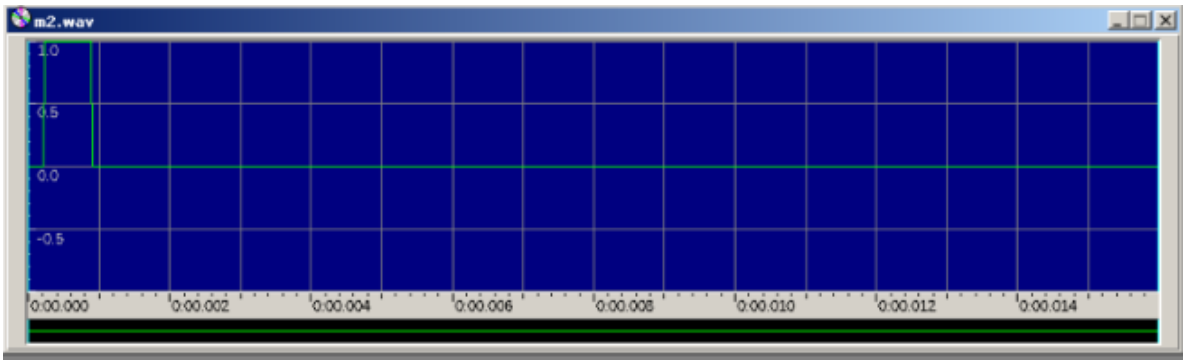


Ilustración 13 Imagen de la señal 2

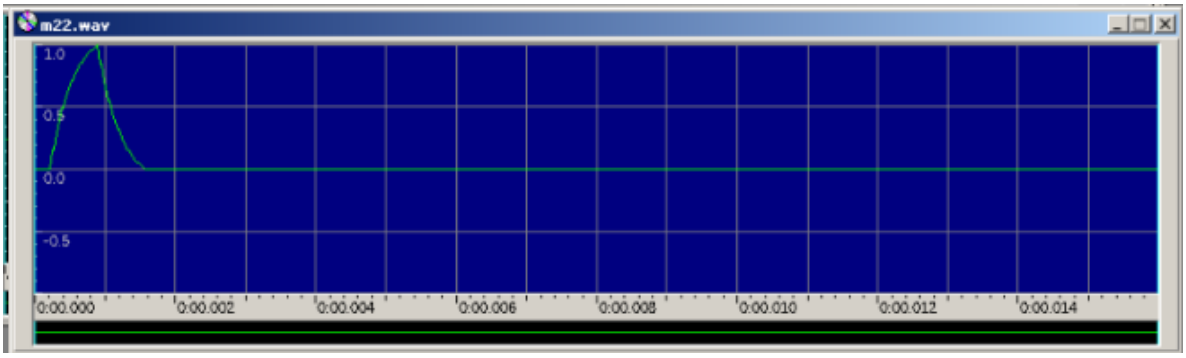


Ilustración 14 Imagen de la señal 2 filtrada

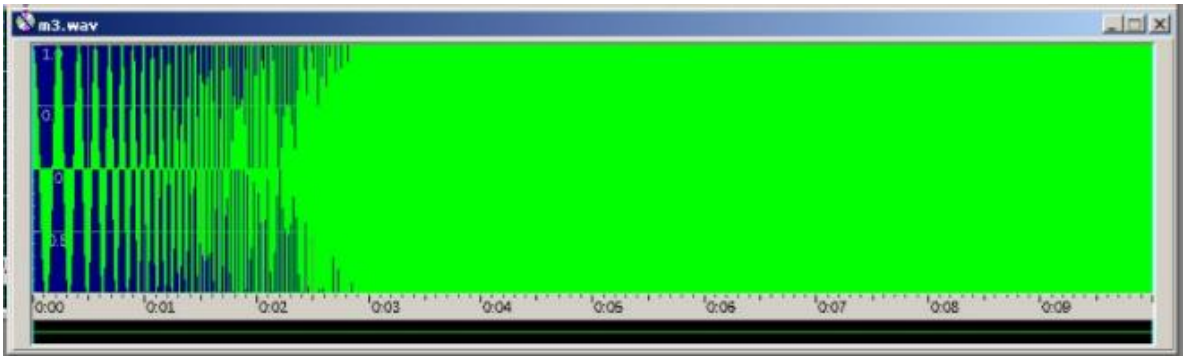


Ilustración 15 Imagen de la señal 3

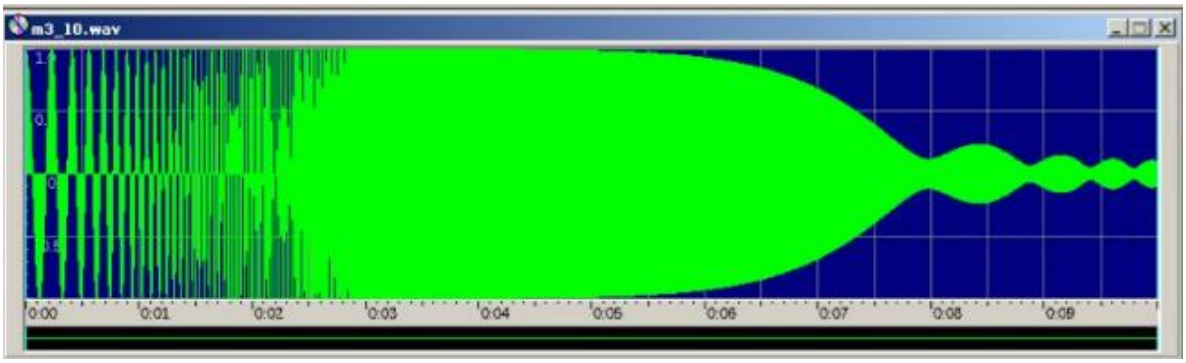


Ilustración 16 Imagen de la señal 3 filtrada a 10 muestra

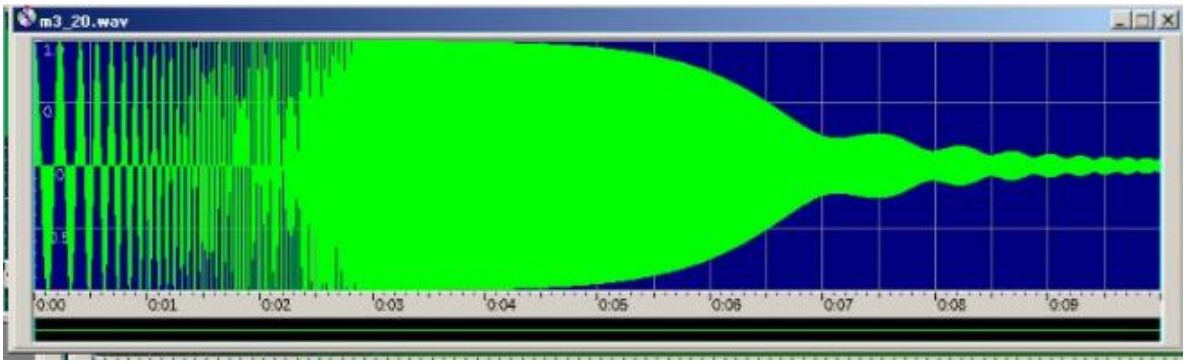


Ilustración 17 Imagen de la señal 3 filtrada a 20 muestras

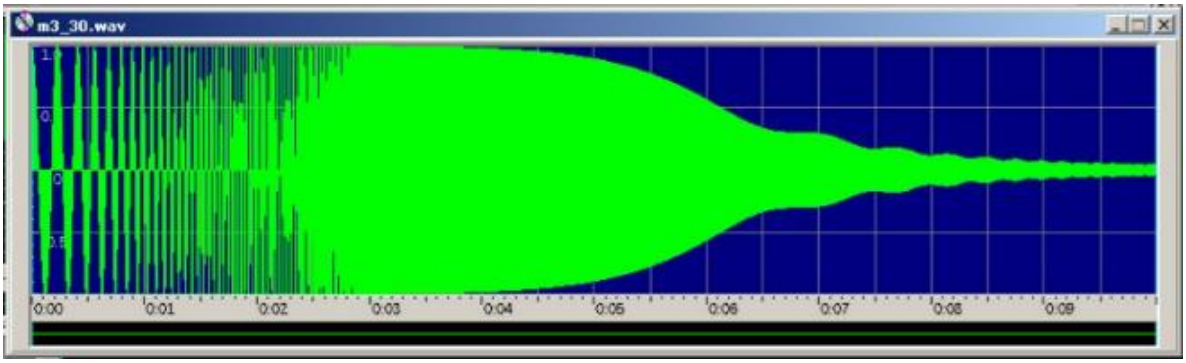


Ilustración 18 Imagen de la señal 3 filtrada a 30 muestras

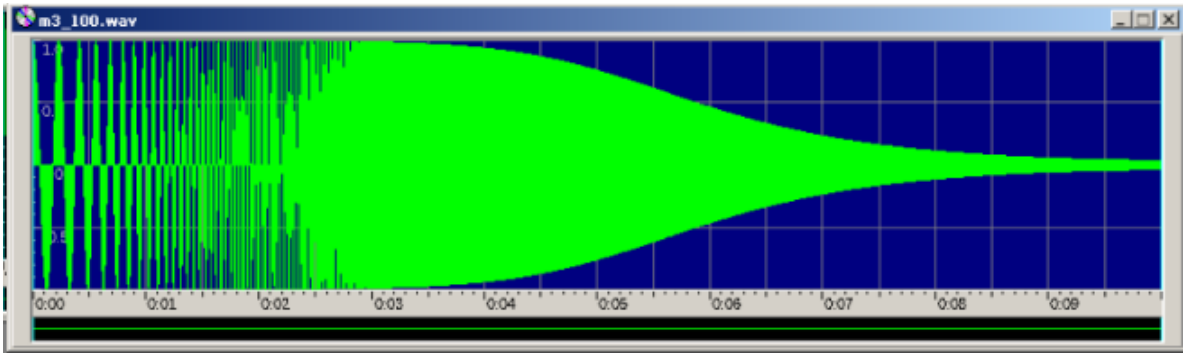


Ilustración 19 Imagen de la señal 3 filtrada a 100 muestras

Buscamos cuando la magnitud de la señal este en .7071

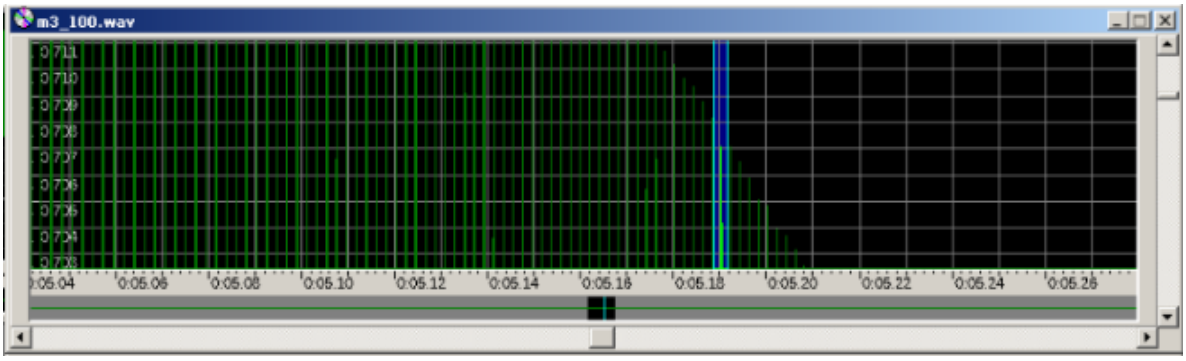


Ilustración 20 Señal a .7071 de magnitud

Una vez localizada a .7071 seleccionamos noise reducción como se muestra en la ilustración 21 y miramos la frecuencia a la que se encuentra. Sabemos que cuando se está a -3dB es la frecuencia de corte o lo que es igual cuando la señal se encuentra a una magnitud de .7071

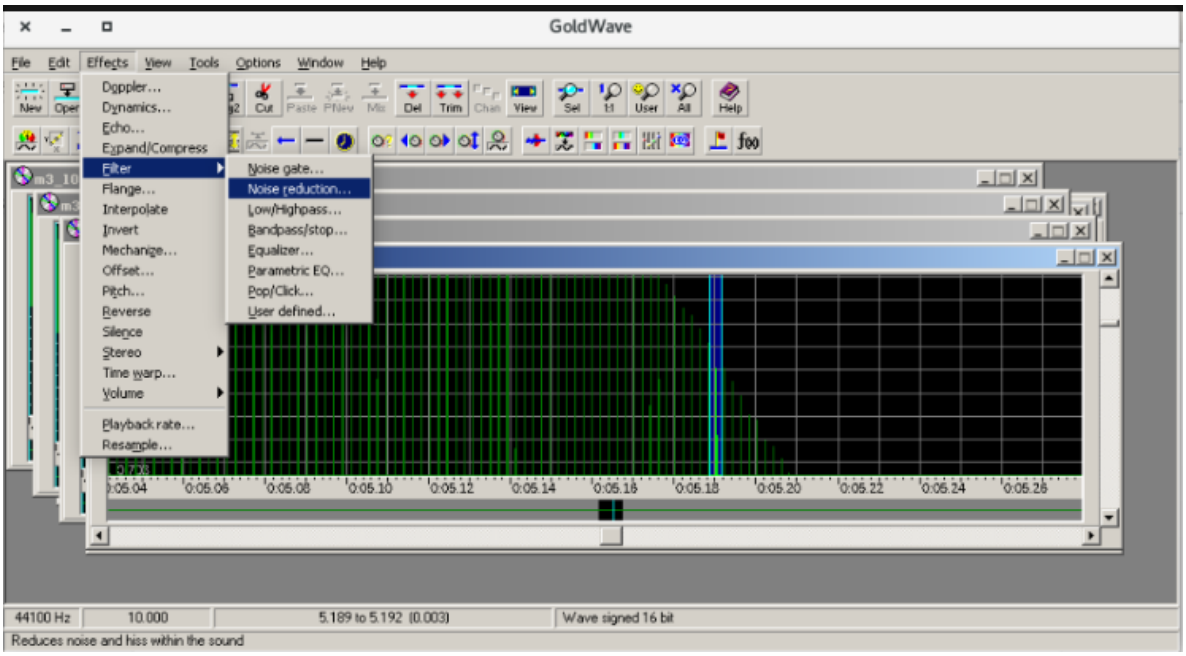


Ilustración 21 Pasos para mirar la frecuencia

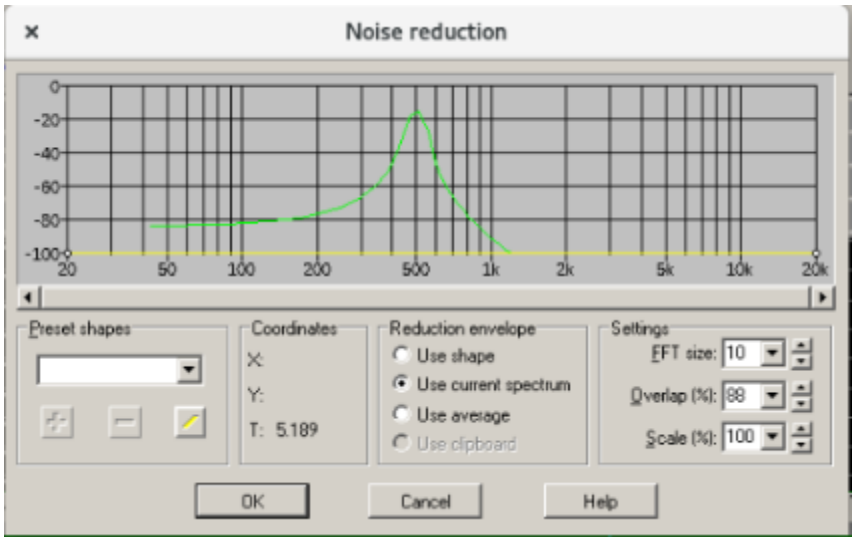


Ilustración 22 Vista de la frecuencia de corte a 500Hz con 100 muestras

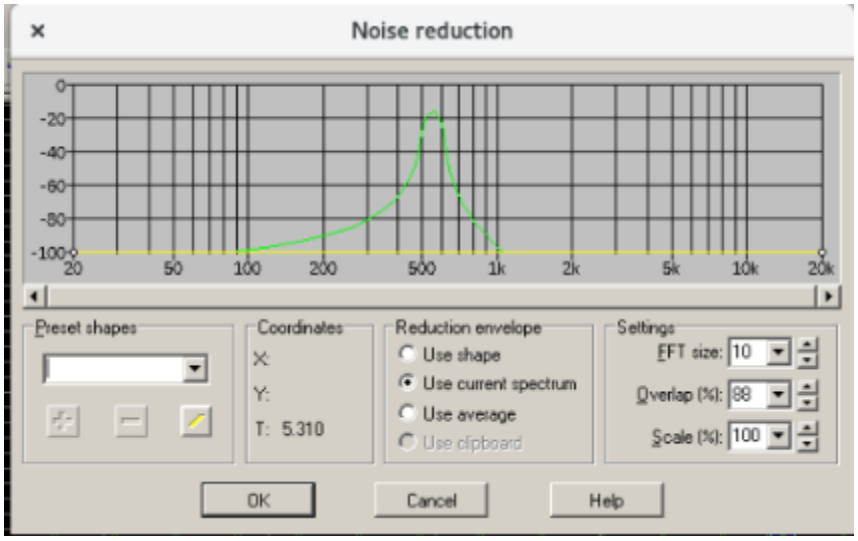


Ilustración 23Frecuencia de corte con 50 muestras

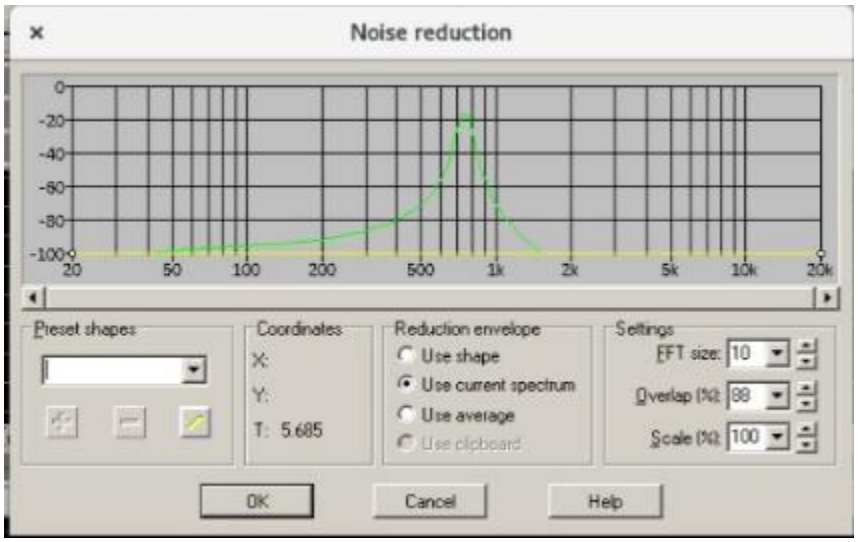


Ilustración 24 Frecuencia de corte con 30 muestras

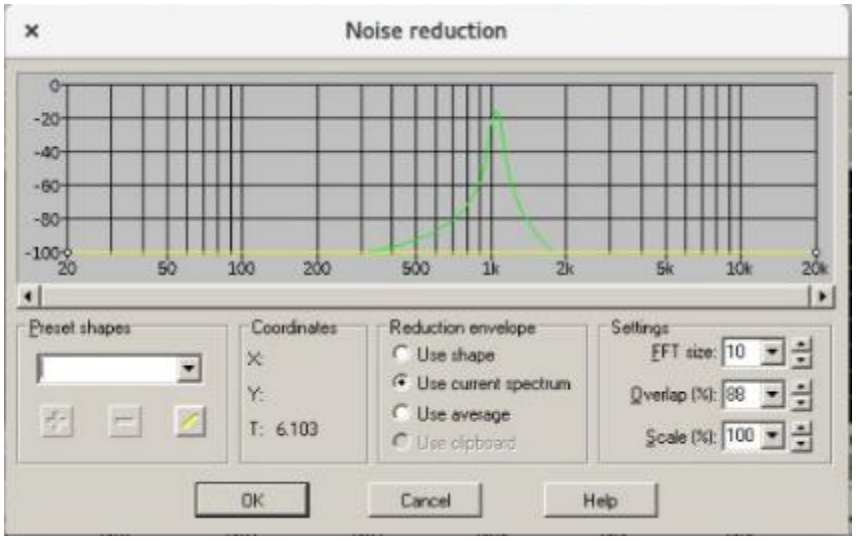


Ilustración 25 Frecuencia de corte con 20 muestras

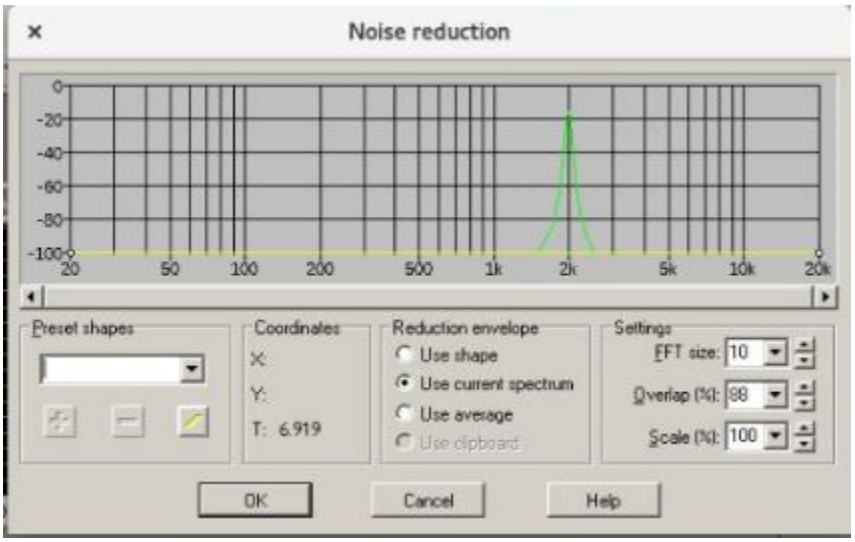


Ilustración 26 Frecuencia de corte con 10 muestras

Conclusiones:

Como pudimos observar a mayor número de muestras mayor precisión, es importante notar que con la convolución podemos obtener la señal que se tendría de pasar por un filtro pasa bajas, pero lo podemos hacer simulando el circuito RC. Por lo cual si estamos trabajando con archivos de audio podemos eliminar frecuencias y si estamos trabajando con circuitos podemos simular con un PIC el circuito RC o algún otro circuito con sus debidas adaptaciones.

Es importante mencionar que a mayor número de nuestras se requerirá mayor poder de cómputo pero la fiabilidad será mayor.

Cualquier señel la podemos trabajar de manera digital y mandarla a algún medio analógico pues como vimos en la práctica 1, el enviar la información se hace en la forma opuesta a como se adquiere la señal.

Código

wav.h

```
#include <stdio.h>
#include <stdlib.h>

typedef struct CabeceraWAV{
	unsigned char ChunkID[4];
	unsigned int ChunkSize;
	unsigned char Format[4];
	unsigned char Subchunk1_ID[4];
	unsigned char Subchunk1_Size[4];
	unsigned char AudioFormat[2];
	unsigned char NumChannels[2];
	unsigned int SampleRate;
	unsigned int ByteRate;
	unsigned short BlockAlign;
	unsigned short BitsPerSample;
	unsigned char Subchunk2_ID[4];
	unsigned int Subchunk2_Size;//Catidad de bytes de informacion de la
señal
}Cabecera;

void informacionAudio(Cabecera cabeceraAudio){
	printf("%c%c%c%c\n",cabeceraAudio.ChunkID[0],cabeceraAudio.ChunkID[
1],cabeceraAudio.ChunkID[2],cabeceraAudio.ChunkID[3]);
	printf("ChunkSize: %d\n",cabeceraAudio.ChunkSize);
	printf("%c%c%c%c\n",cabeceraAudio.Format[0],cabeceraAudio.Format[1]
,cabeceraAudio.Format[2],cabeceraAudio.Format[3]);
	printf("%c%c%c%c\n",cabeceraAudio.Subchunk1_ID[0],cabeceraAudio.Sub
chunk1_ID[1],cabeceraAudio.Subchunk1_ID[2],cabeceraAudio.Subchunk1_ID[3])
;
	printf("Subchunk1_Size:%d
%d%d%d\n",cabeceraAudio.Subchunk1_Size[0],cabeceraAudio.Subchunk1_Size[1]
,cabeceraAudio.Subchunk1_Size[2],cabeceraAudio.Subchunk1_Size[3]);
	printf("Formato del audio: %d\n",cabeceraAudio.AudioFormat[0] );
	printf("Numero de canales: %d\n",cabeceraAudio.NumChannels[0] );
	printf("Muestras por segundo %d\n",cabeceraAudio.SampleRate);
	printf("Bytes por segundo: %d\n",cabeceraAudio.ByteRate);
	printf("Numero de bytes por muestra %d\n",cabeceraAudio.BlockAlign
);
	printf("Numero de bits por muestra: %d\n",
cabeceraAudio.BitsPerSample);
	printf("%c%c%c%c\n",cabeceraAudio.Subchunk2_ID[0],cabeceraAudio.Sub
chunk2_ID[1],cabeceraAudio.Subchunk2_ID[2],cabeceraAudio.Subchunk2_ID[3])
;
	printf("Numero de bytes de la informacion
%d\n",cabeceraAudio.Subchunk2_Size);
	printf("Numero de muestras
%d\n",cabeceraAudio.Subchunk2_Size/(cabeceraAudio.NumChannels[0]*(cabecer
aAudio.BitsPerSample/8)));
	printf("Bytes de sobra (informaciondesconocida):
%d\n",cabeceraAudio.ChunkSize-36-cabeceraAudio.Subchunk2_Size);
	printf("Tam total del archivo: %d\n",cabeceraAudio.ChunkSize+8);
}
```

```
//Resegra el tamaño completo de el archivo en bytes
unsigned int getFileSize(Cabecera CW){
	return CW.ChunkSize+8;
}

//Regresa el tamaño de las muestras en bytes
char getSampleSize(Cabecera CW){
	return (CW.BitsPerSample/8);
}

//Regresa el numero de canales;
char getCannalNumber(Cabecera CW){
	return CW.NumChannels[0];
}

//Regresa el numero de muestras de audio
unsigned int getNumberAudioSamples(Cabecera CW){
	return CW.Subchunk2_Size/(CW.NumChannels[0]*(CW.BitsPerSample/8));
}

unsigned int getNumberBytesAudioInformation(Cabecera CW){
	return CW.Subchunk2_Size;
}

//regresa el numero de bytes del pie del archivo wav
unsigned int getNumberBytesFoot(Cabecera CW){
	return CW.ChunkSize-36-CW.Subchunk2_Size;
}

//Regresa la cabecera del archvo wav
Cabecera getHeader(char *archivoWavEntrada){
	Cabecera CW;
	FILE *archivo=fopen(archivoWavEntrada,"r");
	fread(&CW,44,1,archivo);
	fclose(archivo);
	return CW;
}

char* getAudioSamples8bits(char *archivoWavEntrada){
	Cabecera CW;
	FILE *archivo=fopen(archivoWavEntrada,"r");
	fread(&CW,44,1,archivo);
	unsigned int
numeroMuestras=CW.Subchunk2_Size/(CW.NumChannels[0]*(CW.BitsPerSample/8))
;
	char *Samples=malloc(sizeof(char)*numeroMuestras);
	fread(Samples,numeroMuestras,1,archivo);
	fclose(archivo);
	return Samples;
}

short* getAudioSamples16bits(char *archivoWavEntrada){
	Cabecera CW;
	FILE *archivo=fopen(archivoWavEntrada,"r");
	fread(&CW,44,1,archivo);
```

```
	unsigned int
numeroMuestras=CW.Subchunk2_Size/(CW.NumChannels[0]*(CW.BitsPerSample/8))
;
	short *Samples=malloc(sizeof(short)*numeroMuestras);
	fread(Samples,numeroMuestras*2,1,archivo);
	fclose(archivo);
	return Samples;
}

char* getFileFoot(char *archivoWavEntrada){
	Cabecera CW;
	FILE *archivo=fopen(archivoWavEntrada,"r");
	fread(&CW,44,1,archivo);
	unsigned int bytesSobrantes=CW.ChunkSize-CW.Subchunk2_Size-36;
	char *bytes=malloc(sizeof(char)*CW.Subchunk2_Size);
	fread(bytes,CW.Subchunk2_Size,1,archivo);
	bytes=malloc(sizeof(char)*bytesSobrantes);
	fread(bytes,bytesSobrantes,1,archivo);
	return bytes;
}
```

Convolucion.c

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "wav.h"

#define PI 3.141592653589793
short Fc=500;//500Hz
unsigned short Fs=44100;//44100 muestras por segundo

void main(int argc, char *argv[]){

	Cabecera CW;
	FILE *archivoWavEntrada,*archivoWavSalida;
	archivoWavEntrada=fopen(argv[1],"r");

	if(archivoWavEntrada==NULL){
		printf("Error en la lectura del archivo de audio\n");
	}else{

		double Constante = (2*PI*Fc)/Fs;
		char numMuestras=atoi(argv[3]);
		double *h= malloc(sizeof(double)*numMuestras);

		/*
		*	Calculamos las muestras para el h(n)
		*/
		for(char i=0;i<numMuestras;i++)h[i]=exp(-i*Constante);


		archivoWavSalida=fopen(argv[2],"w");

		if(archivoWavSalida!=NULL){
			Cabecera CW=getHeader(argv[1]);
			fwrite(&CW,44,1,archivoWavSalida);
			short *muestrasAudio=getAudioSamples16bits(argv[1]);
			short *muestrasAudio2=getAudioSamples16bits(argv[1]);
			double *x=malloc(sizeof(double)*numMuestras);
			double
*muestrasAuxiliares=malloc(sizeof(double)*numMuestras);
			unsigned int
numMuestrasAudio=getNumberAudioSamples(CW);

			//Convolucion
			double
*muestrasAudio_2=malloc(sizeof(double)*numMuestrasAudio);
			for(int i=0;i<numMuestrasAudio;i++){
				for(int j=0;j<numMuestras;j++)
					muestrasAuxiliares[j]=h[j]*muestrasAudio[i];

				for(int p=0;p<numMuestras &&
i<numMuestrasAudio;p++)
					muestrasAudio_2[i+p]+=muestrasAuxiliares[p];
			}
			int valorMaximo=muestrasAudio_2[0];
```

```
                    for(int i=1;i<numMuestrasAudio;i++){
                                if(valorMaximo<muestrasAudio_2[i])
                                      valorMaximo=muestrasAudio_2[i];
                    }

                    printf("%d",valorMaximo);
                    for(int i=0;i<numMuestrasAudio;i++){

      muestrasAudio2[i]=muestrasAudio_2[i]*32767/valorMaximo;

                    }

                    unsigned int
bytesAudio=getNumberBytesAudioInformation(CW);
                    fwrite(muestrasAudio2,bytesAudio,1,archivoWavSalida);

                    char *pie=getFileFoot(argv[1]);
                    fwrite(pie,getNumberBytesFoot(CW),1,archivoWavSalida);
                    fclose(archivoWavSalida);
              }
      }
}
```