|  |  |  |
| --- | --- | --- |
| **ipn** | **INSTITUTO POLITÉCNICO NACIONAL**  **ESCUELA SUPERIOR DE CÓMPUTO** |  |

**Teoría de Comunicaciones y Señales**

**“Transformada Rápida de Fourier”**

Resumen

Comparación del algoritmo de la Transformada Discreta de Fourier contra la Transformada Rápida de Fourier realizando la simulación de un circuito RC para filtrar señales en el dominio de la frecuencia.

**Por:**

**Joel Mauricio Romero Gamarra**

Profesor:

EDUARDO GUTIÉRREZ ALDANA

Diciembre 2017

**Índice**

Contenido

[Introducción: 1](#_Toc500703854)

[Análisis Teórico: 3](#_Toc500703855)

[Software (librarías, paquetes, herramientas): 6](#_Toc500703856)

[Procedimiento: 6](#_Toc500703857)

[Resultados 8](#_Toc500703858)

[Discusión: 10](#_Toc500703859)

[Conclusiones: 10](#_Toc500703860)

[Referencias 11](#_Toc500703861)

[Código 12](#_Toc500703862)

# Introducción:

Como sabemos, la TDF se define de la siguiente manera:

Y la TDF Inversa como sigue:

Además, en la fórmula tenemos una exponencial, que representa un número complejo, sin embargo, la computadora no conoce de números complejos y hay que separarlo en la parte real y la parte imaginaria, es decir, convertir el número complejo de la forma de Euler a una suma de senos y cosenos.

y

Así que:

Por lo tanto:

y

Sin embargo, en estas fórmulas existe redundancia ya que se calculan coeficientes previamente calculados, además, como vimos en la práctica anterior, el número de operaciones realizadas aumenta exponencialmente cuando el número de muestras aumenta.

El algoritmo para calcular la FFT es un algoritmo de diezmado en el tiempo, este tipo de algoritmos son muy eficientes (sin embargo, la FFT tiene la restricción de que el número de muestras N debe ser potencia de 2), estos algoritmos se basan en el principio de divide y vencerás, debido a que calculan las TDF de cada subsecuencia y al final combinan las soluciones para formar una solución general, como punto de partida, se toma la ecuación de la TDF: [1]

Ahora, se separa la ecuación, sin embargo, a diferencia de la TDF, en vez de separar la ecuación en sus partes real e imaginaria, se separa para cuando la n es par o impar y resulta:

La descomposición inicial de separar la señal en pares e impares, se muestra a continuación en la Figura 1.

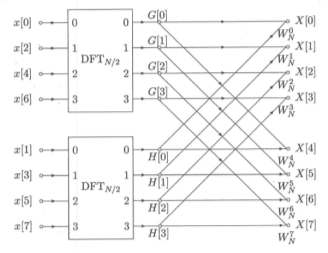


Figura 1. Algoritmo diezmado en el tiempo para 8 muestras

Como se mencionó anteriormente, se deriva de la estrategia divide y vencerás, por lo que como se puede observar en la Figura 1, la TDF de 8 muestras, se dividió en 2 TDF de 4 muestras cada una, y eso se vuelve a dividir para quedarnos de la siguiente forma:

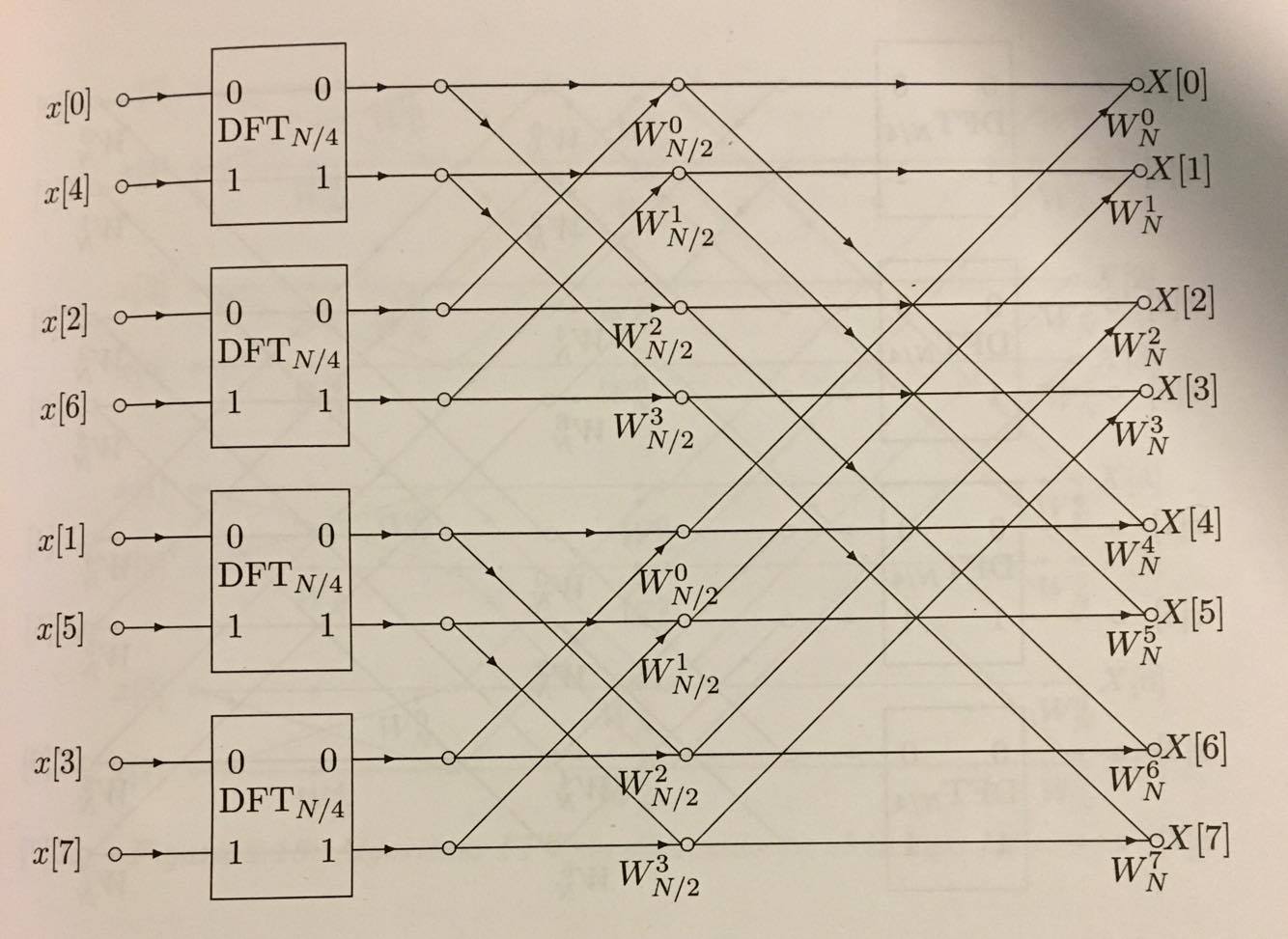


Figura 2. Algoritmo FFT para 8 muestras

Como vimos, este ejemplo se conformaba por 8 muestras, y el algoritmo de FFT lo dividió 2 veces (primero en 2 TDF de 4 muestras cada una y luego en 4 TDF de 2 muestras cada una), sin embargo, se vuelve a dividir una última vez (para este caso de 8 muestras), es decir, que el número de veces que se va a hacer la división para resolver la transformada será , siendo N el número de muestras, y como es un logaritmo base 2, necesitamos que el número de divisiones sea un número entero, por lo tanto, se demuestra lo que ya se había mencionado como restricción, *el número de muestras de la señal debe ser potencia de 2*.

# Análisis Teórico:

El algoritmo de la FFT está definido por el siguiente código:

ciclos = (log (muestras) / (log (2.0)));

**for** (i = 0; i < (muestras); i ++)

{

real [i] = signal [i];

imaginario [i] = 0.0;

}

//Bit reversal

**for** (i = 0; i < muestras; i ++)

{

j = 0;

**for** (k = 0; k < ciclos; k ++)

j = ((j << 1) | (1 & (i >> k)));

**if** (j < i)

{

swap (&real [i], &real [j]);

swap (&imaginario [i], &imaginario [j]);

}

}

//FFT Algorithm

**for** (i = 0; i < ciclos; i ++)

{

n = w = pow (2, (**float**) i);

w = (PI / n);

**if** (inverse)

w = -w;

k = 0;

**while** (k < (muestras - 1))

{

**for** (j = 0; j < n; j ++)

{

angulo = (-j \* w);

j1 = (j + k);

parte\_real = ((real [j1 + n] \* cos (angulo)) - (imaginario [j1 + n] \* sin (angulo)));

parte\_imaginaria = ((imaginario [j1 + n] \* cos (angulo)) + (real [j1 + n] \* sin (angulo)));

real [j1 + n] = (real [j1] - parte\_real);

imaginario [j1 + n] = (imaginario [j1] - parte\_imaginaria);

real [j1] = (real [j1] + parte\_real);

imaginario [j1] = (imaginario [j1] + parte\_imaginaria);

}

k += (2 \* n);

}

}

El orden de complejidad del algoritmo de la FFT es O (n · log2 (n)), que es bastante aceptable, y concuerda con los órdenes de complejidad de un algoritmo divide y vencerás, debido a que, la parte de dividir a la mitad está dada por el logaritmo base 2 y la parte de combinar las soluciones al final está dada por el n que multiplica al logaritmo, a continuación se muestra una tabla con diferente número de muestras y un aproximado de instrucciones comparando el algoritmo FFT contra el algoritmo de la TDF.

|  |  |  |
| --- | --- | --- |
| Número de muestras | Número de instrucciones (TDF) | Número de instrucciones (FFT) |
| 32 | 1, 024 | 160 |
| 128 | 16, 384 | 896 |
| 1, 024 | 1, 048, 576 | 10, 240 |
| 2, 048 | 4, 194, 304 | 22, 528 |
| 8, 192 | 67, 108, 864 | 106, 496 |
| 65, 536 | 4, 294, 967, 296 | 1, 048, 576 |

Figura 3. Tabla de complejidad del algoritmo TDF

Como podemos ver, el algoritmo de la FFT reduce muchísimo el número de instrucciones, por ejemplo, para 128 muestras es una diferencia de más de 15,000, es decir, más del 90% de instrucciones ahorradas, y para un archivo con 65, 536 muestras la diferencia es de poco más de 4, 293, 000, 000, es decir, más del 90% de operaciones.

Para poder observar mejor la comparación de los 2 algoritmos, se tomó el tiempo dentro del código de lo que tarda (al igual que en la TDF) para archivos con distinto número de muestras y se graficó para ver el comportamiento de un orden de complejidad cuadrado respecto a un orden de complejidad nlogn, la gráfica de la comparación se muestra en la Figura 4.

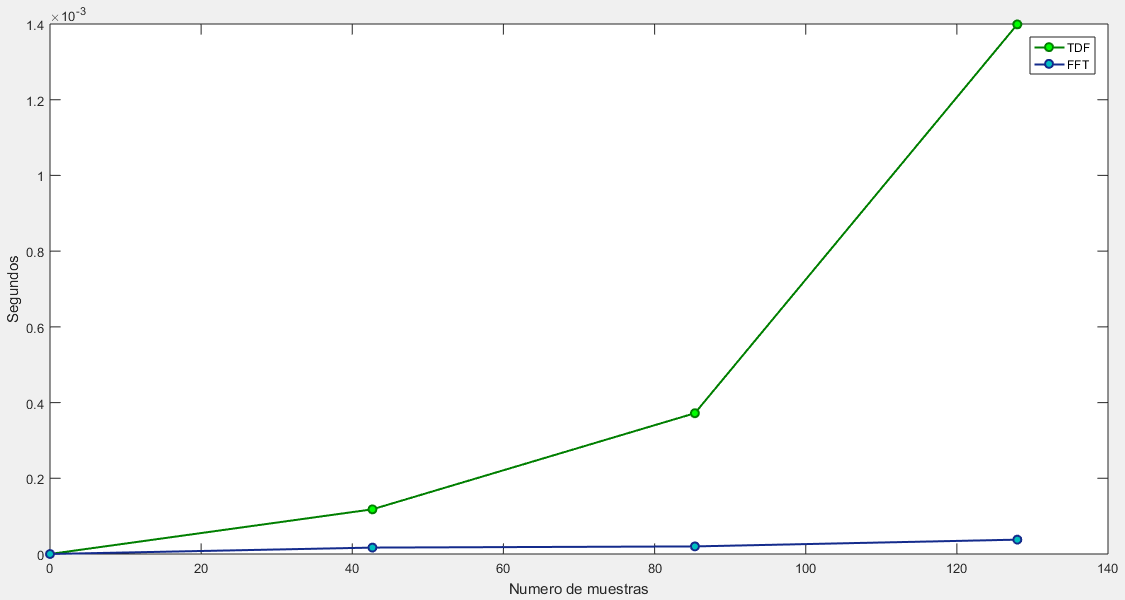


Figura 4. Comparación en tiempo de ejecución hasta 128 muestras

Ahora, procederemos a aumentar las muestras desde 0 hasta 1024 para observar el comportamiento de la gráfica en una perspectiva de muestras más grande.

Estas gráficas se muestran a continuación en las Figuras 5 y 6.

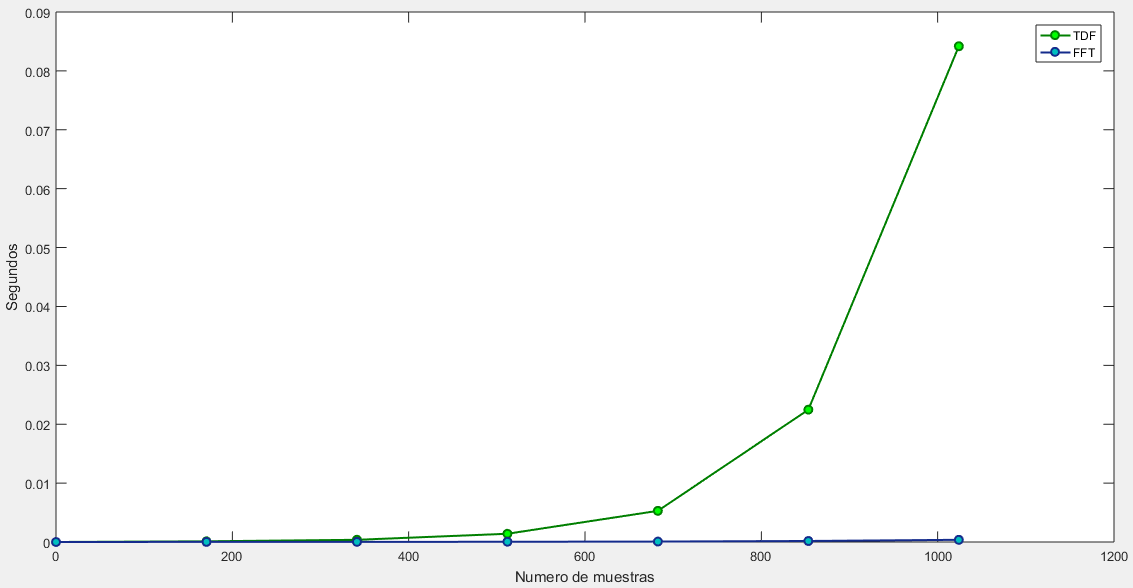


Figura 5. Tiempo de ejecución hasta 1, 024 muestras

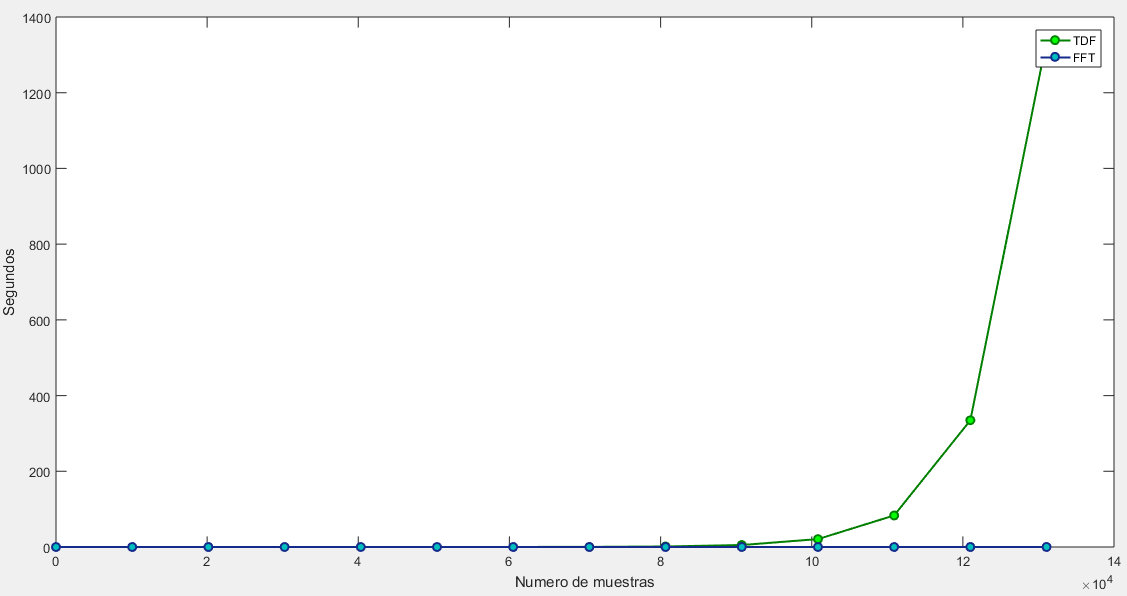


Figura 6. Tiempo de ejecución hasta 131, 072 muestras

Como podemos ver en las gráficas mostradas, la diferencia entre ambos algoritmos es abismal, ya que para muestras desde 0, 32, 64 y 128 la diferencia es notoria, y para cuando llegamos hasta 131, 072 muestras ya la diferencia es enorme, porque hace menos del 95% de operaciones que la TDF y por eso los tiempos son tan reducidos.

Para este programa, se tendrán las mismas 3 opciones que en la práctica de la TDF:

* Mostrar la señal original en el canal izquierdo y la magnitud en el canal derecho.
* Mostrar la parte real en el canal izquierdo y la parte imaginaria en el canal derecho.
* Mostrar la magnitud en el canal izquierdo y la fase en el canal derecho.

Como ya se había explicado en la práctica anterior, las fórmulas de la fase y magnitud son las siguientes:

# Software (librarías, paquetes, herramientas):

* GoldWave versión 4.26 [2]
* MATLAB R2016a [3]
* Sublime Text 3 [4]

# Procedimiento:

Lo primero que debemos hacer es generar un archivo wav con 2 canales para convertirlo en un archivo estéreo y poder meter en cada uno de los canales la opción correspondiente que seleccione el usuario.

En cuanto a la ejecución del programa, algunas pruebas puntuales son presentadas en la sección de resultados, como probar las 3 distintas opciones, y posteriormente usar la opción 2 y regresar a la señal original aplicando la FFT inversa, sin embargo, para hacer la graficación de los órdenes de complejidad comparados (presentado en la sección de análisis teórico) donde se analizó la optimización del algoritmo FFT contra el de la TDF, se utilizaron 2 Scripts en Linux.

Se realizaron las pruebas para una señal coseno con distinto número de pruebas a una frecuencia de muestreo de 2, 000 Hz variando únicamente la duración (es decir, duplicar la duración, comenzando por 0.016 hasta 65.535 segundos) y utilizando únicamente la opción 2 (ya que calcular la magnitud y fase consume un poco más de operaciones, y solo estamos analizando el algoritmo de la transformada en sí).

**SCRIPTS UTILIZADOS:**

#!/bin/bash

**echo** 'TIEMPOS CON LA TRANSFORMADA DISCRETA DE FOURIER' >> Tiempos**.**txt

**cd** /home/joel/Escritorio/Se�ales/Transformada

gcc TDF**.**c -o TDF -lm

**.**/TDF -2 Coseno32**.**wav TDF\_Coseno32**.**wav >> Tiempos**.**txt

**.**/TDF -2 Coseno64**.**wav TDF\_Coseno64**.**wav >> Tiempos**.**txt

**.**/TDF -2 Coseno128**.**wav TDF\_Coseno128**.**wav >> Tiempos**.**txt

**.**/TDF -2 Coseno256**.**wav TDF\_Coseno256**.**wav >> Tiempos**.**txt

**.**/TDF -2 Coseno512**.**wav TDF\_Coseno512**.**wav >> Tiempos**.**txt

**.**/TDF -2 Coseno1024**.**wav TDF\_Coseno1024**.**wav >> Tiempos**.**txt

**.**/TDF -2 Coseno2048**.**wav TDF\_Coseno2048**.**wav >> Tiempos**.**txt

**.**/TDF -2 Coseno4096**.**wav TDF\_Coseno4096**.**wav >> Tiempos**.**txt

**.**/TDF -2 Coseno8192**.**wav TDF\_Coseno8192**.**wav >> Tiempos**.**txt

**.**/TDF -2 Coseno16384**.**wav TDF\_Coseno16384**.**wav >> Tiempos**.**txt

**.**/TDF -2 Coseno32768**.**wav TDF\_Coseno32768**.**wav >> Tiempos**.**txt

**.**/TDF -2 Coseno65536**.**wav TDF\_Coseno65536**.**wav >> Tiempos**.**txt

**.**/TDF -2 Coseno131072**.**wav TDF\_Coseno131072**.**wav >> Tiempos**.**txt

**cd** /home/joel/Escritorio/Se�ales/Transformada

chmod 777 FFT**.**sh

**.**/FFT.sh

#!/bin/bash

**echo** 'TIEMPOS CON LA TRANSFORMADA RAPIDA DE FOURIER' >> Tiempos**.**txt

**cd** /home/joel/Escritorio/Se�ales/Transformada

gcc FFT**.**c -o FFT -lm

**.**/FFT -2 Coseno32**.**wav FFT\_Coseno32**.**wav >> Tiempos**.**txt

**.**/FFT -2 Coseno64**.**wav FFT\_Coseno64**.**wav >> Tiempos**.**txt

**.**/FFT -2 Coseno128**.**wav FFT\_Coseno128**.**wav >> Tiempos**.**txt

**.**/FFT -2 Coseno256**.**wav FFT\_Coseno256**.**wav >> Tiempos**.**txt

**.**/FFT -2 Coseno512**.**wav FFT\_Coseno512**.**wav >> Tiempos**.**txt

**.**/FFT -2 Coseno1024**.**wav FFT\_Coseno1024**.**wav >> Tiempos**.**txt

**.**/FFT -2 Coseno2048**.**wav FFT\_Coseno2048**.**wav >> Tiempos**.**txt

**.**/FFT -2 Coseno4096**.**wav FFT\_Coseno4096**.**wav >> Tiempos**.**txt

**.**/FFT -2 Coseno8192**.**wav FFT\_Coseno8192**.**wav >> Tiempos**.**txt

**.**/FFT -2 Coseno16384**.**wav FFT\_Coseno16384**.**wav >> Tiempos**.**txt

**.**/FFT -2 Coseno32768**.**wav FFT\_Coseno32768**.**wav >> Tiempos**.**txt

**.**/FFT -2 Coseno65536**.**wav FFT\_Coseno65536**.**wav >> Tiempos**.**txt

**.**/FFT -2 Coseno131072**.**wav FFT\_Coseno131072**.**wav >> Tiempos**.**txt

Como se puede ver, los scripts utilizados son bastante simples, al final del script que corre la TDF normal manda a llamar (después de ejecutar sus respectivas pruebas) a la FFT para ejecutar exactamente las pruebas con los mismos archivos, los tiempos fueron escritos en un archivo y posteriormente se toman esos tiempos de ejecución en segundos del archivo para hacer la gráfica de comparación de los 2 algoritmos.

**FUNCIÓN PARA MEDIR EL TIEMPO DE EJECUCIÓN:** [5]

**void** uswtime(**double** \*usertime, **double** \*systime, **double** \*walltime)

{

**double** mega = 1.0e-6;

**struct** rusage buffer;

**struct** timeval tp;

**struct** timezone tzp;

getrusage(RUSAGE\_SELF, &buffer);

gettimeofday(&tp, &tzp);

\*usertime = (**double**) buffer.ru\_utime.tv\_sec +1.0e-6 \* buffer.ru\_utime.tv\_usec;

\*systime = (**double**) buffer.ru\_stime.tv\_sec +1.0e-6 \* buffer.ru\_stime.tv\_usec;

\*walltime = (**double**) tp.tv\_sec + 1.0e-6 \* tp.tv\_usec;

}

# Resultados

Al ser el mismo programa que la práctica anterior, con la única diferencia de que está optimizado, en esta ocasión en la sección de resultados se presentarán únicamente las señales originales y posteriormente la FFT indicando la opción con la que fue ejecutado el programa.

La primera señal de prueba será un tren de impulsos a una frecuencia de muestreo de 8,192 con una duración de 1 segundo (8, 192 muestras).

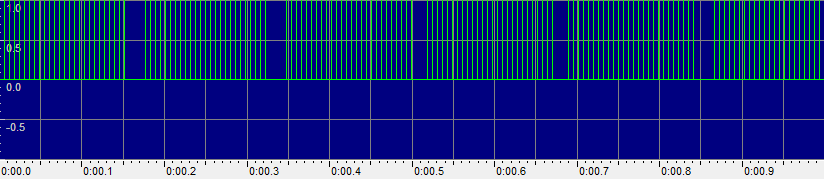


Figura 7. Señal original

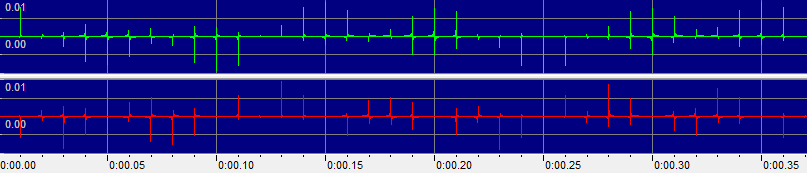


Figura 8. FFT de la señal original (opción 2)

A continuación, se muestra otro ejemplo, pero ahora con una señal coseno a una frecuencia de 8, 192 Hz pero con una duración de 2 segundos, dando un total de 16, 384 muestras.

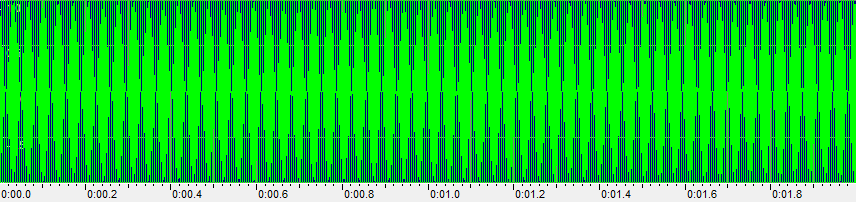


Figura 9. Señal original

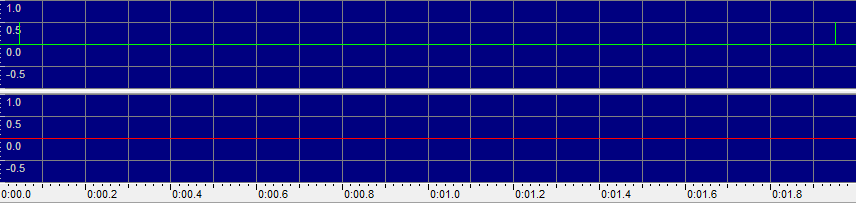


Figura 10. FFT de la señal original (opción 2)

A continuación, se muestra una tabla para comparar los tiempos de ejecución entre la TDF y la FFT:

|  |  |  |
| --- | --- | --- |
| Número de muestras | Tiempo de Ejecución (TDF) | Tiempo de Ejecución (FFT) |
| 32 | 0.0001180172 | 0.0000169277 |
| 64 | 0.0003719330 | 0.0000200272 |
| 128 | 0.0013990402 | 0.0000379086 |
| 256 | 0.0052890778 | 0.0000758171 |
| 512 | 0.0224690437 | 0.0001709461 |
| 1,024 | 0.0841879845 | 0.0003709793 |
| 2,048 | 0.3301670551 | 0.0010309219 |
| 4,096 | 1.3135201931 | 0.0024490356 |
| 8,192 | 5.2289988995 | 0.0042259693 |
| 16,384 | 20.8389170170 | 0.0084071159 |
| 32,768 | 83.3325860500 | 0.0176849365 |
| 65,536 | 334.7199959755 | 0.0381770134 |
| 131,072 | 1336.4872810841 | 0.0823328495 |

# Discusión:

La Transformada Discreta de Fourier tiene distintas aplicaciones, sin embargo, al igual que la convolución, es un algoritmo muy costoso, y por ejemplo si quisiéramos filtrar una señal directo desde la frecuencia con una TDF y luego una multiplicación, es más fácil que hacer una convolución. En cuanto a la velocidad, habría que analizar bien ambos, ya que si la convolución se implementa como se hizo (Como un filtro FIR, es decir, Finito), entonces conviene más hacer una TDF y luego una multiplicación, debido a que para la convolución primero hay que llenar todo con 0, luego recorrer todos los elementos, posteriormente multiplicar todos los elementos y sumarlos para obtener un elemento, y todo esto en un ciclo que va de 1 a N donde N son las muestras que forman a la señal.

El tiempo de ejecución y el uso de memoria excesivo es algo que tiene este algoritmo, ya que hay que tener un arreglo con la parte imaginaria, uno con la parte real, y en cuanto a operaciones, sabemos que realizar operaciones a nivel de bit es muy rápido, sin embargo involucrar funciones como coseno, seno o tangente seguramente tardan muchísimo en ejecutarse y obviamente, hacen más lento el programa.

# Conclusiones:

Algo interesante es que para cambiar el programa de hacer la TDF a la TDF inversa es únicamente cambiar un signo para la parte imaginaria, y al final dividir entre el número de muestras ambas respuestas, por lo tanto es demasiado simple, prácticamente no hay que hacer muchos cambios.

El algoritmo de la FFT es mucho más rápido realizando menos operaciones que la TDF, sin embargo se analizará con detenimiento en la siguiente práctica, es bastante útil debido a que como ya vimos en la tabla en la sección de análisis teórico y en las gráficas con los tiempos que se tardan para distintos tipos de muestras que para un número de muestras considerable es muy tardado, por ejemplo, en la vida real las frecuencias a las que escuchamos la música está muestreada a 44, 100 Hz, es decir, 44, 100 muestras por segundo, ahora, imaginando que tuviéramos toda una canción a la que queremos aplicarle un filtro (como un ecualizador o algo así por el estilo), tomemos una duración de 3 minutos, es decir, 180 segundos. Tendríamos un total de 7, 938, 000 muestras que es casi 7 veces el número mayor de muestras que utilizamos para graficar.

Otra aplicación, sería reconocer una canción por medio del análisis de frecuencia, como por ejemplo la famosa aplicación Shazam que escucha una canción por un periodo de aproximadamente 10 segundos, detectar las frecuencias de cada muestra para reconocer una canción debe ser un trabajo muy extenso que involucra una transformada de Fourier para analizar las frecuencias y posteriormente “matchear” con las frecuencias de alguna canción ya conocida debe ser un trabajo que se realice en menos de 3 segundos.

Es importante saber cómo optimizar los algoritmos ya conocidos que tienen aplicaciones en la vida real, como por ejemplo, la convolución para filtrar una señal es un algoritmo muy tardado que podemos optimizar haciendo uso de la transformada Z en cuanto a tiempos de ejecución y memoria utilizada ya que necesitamos de únicamente 4 valores, y en este caso para la FFT (Fast Fourier Transform) se utiliza un concepto de diezmado en el tiempo y un arreglo de mariposa para evitar repetir operaciones previamente realizadas. [1]

# Referencias

|  |  |
| --- | --- |
| [1] | A. A. Colomer, V. Naranjo Ornedo y J. Prades Nebot, Tratamiento Digital de la Señal Teoría y Aplicaciones, México: Limusa, 2009. |
| [2] | G. W. Inc, «GoldWave Version 4.26 Download,» [En línea]. Available: https://www.goldwave.com/release426.php. |
| [3] | MathWorks, «R2016a,» [En línea]. Available: https://www.mathworks.com/products/new\_products/release2016a.html. |
| [4] | S. Text, «Sublime Text 3 Download,» [En línea]. Available: https://www.sublimetext.com/3. |
| [5] | E. A. F. Martínez, «Pruebas a Posteriori,» Octubre 2017. [En línea]. Available: http://www.eafranco.com/docencia/analisisdealgoritmos/files/practicas/01/Practica01.pdf. |
| [6] | G. Carman, «Efecto Aliasing,» Julio 2014. [En línea]. Available: http://grupocarman.com/blog/efecto-aliasing/. [Último acceso: Diciembre 2017]. |
| [7] | E. A. F. Martínez, «Análisis de Algoritmos No Recursivos,» Octubre 2017. [En línea]. Available: http://www.eafranco.com/docencia/analisisdealgoritmos/files/05/Tema05.pdf. [Último acceso: Diciembre 2017]. |
| [8] | Microsoft, «Office 365,» [En línea]. Available: https://products.office.com/es-mx/products?tab=O-Home. |

# Código

**Cabecera.h**

**typedef** **struct** CABECERA

{

**char** ChunkID[4]; //Contiene 'RIFF'

**int** ChunkSize; //Contiene el tama�o total del archivo - 8 bytes

**char** Format[4]; //Contiene 'WAVE'

//Aqui comienza el primer subchunk 'fmt'

**char** SubChunk1ID[4]; //Contiene 'fmt'

**int** SubChunk1Size; //Contiene el tama�o del primer subchunk

**short** AudioFormat; //Formato de audio

**short** NumChannels; //Numero de canales

**int** SampleRate; //Frecuencia de muestreo

**int** ByteRate; //(SampleRate \* Numero canales \* Bits per Sample) / 8

**short** BlockAlign; //Bytes por muestra

**short** BitsPerSample; //8 bits, 16 bits, etc.

//Aqui comienza el segundo subchunk 'data'

**char** SubChunk2ID[4]; //Contiene 'data'

**int** SubChunk2Size; //Numero de bytes en los datos

}cabecera;

**void** imprimir\_cabecera (cabecera \* cab);

**void** opcion\_uno (FILE \* salida, **float** \* re, **float** \* im, **int** numero\_muestras);

**void** opcion\_dos (FILE \* salida, **float** \* re, **float** \* im, **int** numero\_muestras);

**void** opcion\_tres (FILE \* salida, **float** \* re, **float** \* im, **int** numero\_muestras);

**void** swap (**float** \* a, **float** \* b);

**void** uswtime (**double** \* usertime, **double** \* walltime);

**void** calculaTiempo (**double** utime0, **double** wtime0, **double** utime1, **double** wtime1, **int** n, **int** opcion);

**Cabecera.c**

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <string.h>

#include <sys/resource.h>

#include <sys/time.h>

#include "Cabecera.h"

#define PI 3.14159265

**double** utime0, wtime0, utime1, wtime1;

**void** imprimir\_cabecera (cabecera \* cab)

{

**char** \* formatoArchivo = (**char** \*) malloc (**sizeof** (**char**));

printf ("\n\n");

printf ("(1-4) Chunk ID: %s\n", cab -> ChunkID);

printf ("(5-8) ChunkSize: %u\n", cab -> ChunkSize);

printf ("(9-12) Format: %s\n", cab -> Format);

printf ("(13-16) SubChunk 1 ID: %s\n", cab -> SubChunk1ID);

printf ("(17-20) SubChunk 1 Size: %u\n", cab -> SubChunk1Size);

**if** (cab -> AudioFormat == 1)

strcpy (formatoArchivo, "PCM");

printf ("(21-22) Audio Format: %u, %s\n", cab -> AudioFormat,formatoArchivo);

**if** (cab -> NumChannels == 1)

strcpy (formatoArchivo, "Mono");

**else**

strcpy (formatoArchivo, "Stereo");

printf ("(23-24) Number of Channels: %u, Tipo: %s\n", cab -> NumChannels,formatoArchivo);

printf ("(25-28) Sample Rate: %u\n", cab -> SampleRate);

printf ("(29-32) Byte Rate: %u BitRate: %u\n", cab -> ByteRate,cab -> ByteRate \* 8);

printf ("(33-34) Block Align: %u\n", cab -> BlockAlign);

printf ("(35-36) Bits Per Sample: %u\n", cab -> BitsPerSample);

printf ("(37-40) SubChunk 2 ID: %s\n", cab -> SubChunk2ID);

printf ("(41-44) SubChunk 2 Size: %u\n\n\n", cab -> SubChunk2Size);

}

**void** opcion\_uno (FILE \* salida, **float** \* xr, **float** \* xi, **int** numero\_muestras)

{

**int** i, j, k, j1, m, n;

**float** arg, s, c, w, tempr, tempi;

**short** parte\_real, parte\_imaginaria, magnitud;

**short** \* original = (**short** \*) malloc (**sizeof** (**short**) \* numero\_muestras);

m = (log (numero\_muestras) / log (2));

fseek (salida, 44, SEEK\_SET);

**for** (i = 0; i < numero\_muestras; i ++)

original [i] = (**short**) xr [i];

uswtime (&utime0, &wtime0);

//Bit Reversal

**for** (i = 0; i < numero\_muestras; ++ i)

{

j = 0;

**for** (k = 0; k < m; ++ k)

j = (j << 1) | (1 & (i >> k));

**if** (j < i)

{

swap (&xr [i], &xr [j]);

swap (&xi [i], &xi [j]);

}

}

**for** (i = 0; i < m; i ++)

{

n = w = pow (2, i);

w = (PI / n);

k = 0;

**while** (k < (numero\_muestras - 1))

{

**for** (j = 0; j < n; j ++)

{

arg = (-j \* w);

c = cos (arg);

s = sin (arg);

j1 = (k + j);

tempr = ((xr [j1 + n] \* c) - (xi [j1 + n] \* s));

tempi = ((xi [j1 + n] \* c) + (xr [j1 + n] \* s));

xr [j1 + n] = (xr [j1] - tempr);

xi [j1 + n] = (xi [j1] - tempi);

xr [j1] = (xr [j1] + tempr);

xi [j1] = (xi [j1] + tempi);

}

k += (2 \* n);

}

}

**for** (i = 0; i < numero\_muestras; i ++)

{

parte\_real = xr [i];

parte\_imaginaria = xi [i];

parte\_real /= 2;

parte\_imaginaria /= 2;

magnitud = sqrt (pow (parte\_real, 2) + pow (parte\_imaginaria, 2));

fwrite (&original [i], **sizeof** (**short**), 1, salida);

fwrite (&magnitud, **sizeof** (**short**), 1, salida);

}

uswtime (&utime1, &wtime1);

calculaTiempo (utime0, wtime0, utime1, wtime1, numero\_muestras, 1);

}

**void** opcion\_dos (FILE \* salida, **float** \* xr, **float** \* xi, **int** numero\_muestras)

{

**int** i, j, k, j1, m, n;

**float** arg, s, c, w, tempr, tempi;

**short** parte\_real, parte\_imaginaria;

m = (log (numero\_muestras) / log (2));

fseek (salida, 44, SEEK\_SET);

uswtime (&utime0, &wtime0);

//Bit Reversal

**for** (i = 0; i < numero\_muestras; ++ i)

{

j = 0;

**for** (k = 0; k < m; ++ k)

j = (j << 1) | (1 & (i >> k));

**if** (j < i)

{

swap (&xr [i], &xr [j]);

swap (&xi [i], &xi [j]);

}

}

**for** (i = 0; i < m; i ++)

{

n = w = pow (2, i);

w = (PI / n);

k = 0;

**while** (k < (numero\_muestras - 1))

{

**for** (j = 0; j < n; j ++)

{

arg = (-j \* w);

c = cos (arg);

s = sin (arg);

j1 = (k + j);

tempr = ((xr [j1 + n] \* c) - (xi [j1 + n] \* s));

tempi = ((xi [j1 + n] \* c) + (xr [j1 + n] \* s));

xr [j1 + n] = (xr [j1] - tempr);

xi [j1 + n] = (xi [j1] - tempi);

xr [j1] = (xr [j1] + tempr);

xi [j1] = (xi [j1] + tempi);

}

k += (2 \* n);

}

}

**for** (i = 0; i < numero\_muestras; i ++)

{

parte\_real = xr [i];

parte\_imaginaria = xi [i];

parte\_real /= 2;

parte\_imaginaria /= 2;

fwrite (&parte\_real, **sizeof** (**short**), 1, salida);

fwrite (&parte\_imaginaria, **sizeof** (**short**), 1, salida);

}

uswtime (&utime1, &wtime1);

calculaTiempo (utime0, wtime0, utime1, wtime1, numero\_muestras, 2);

}

**void** opcion\_tres (FILE \* salida, **float** \* xr, **float** \* xi, **int** numero\_muestras)

{

**int** i, j, k, j1, m, n;

**float** arg, s, c, w, tempr, tempi;

**short** parte\_real, parte\_imaginaria, magnitud, fase;

m = (log (numero\_muestras) / log (2));

fseek (salida, 44, SEEK\_SET);

uswtime (&utime0, &wtime0);

//Bit Reversal

**for** (i = 0; i < numero\_muestras; ++ i)

{

j = 0;

**for** (k = 0; k < m; ++ k)

j = (j << 1) | (1 & (i >> k));

**if** (j < i)

{

swap (&xr [i], &xr [j]);

swap (&xi [i], &xi [j]);

}

}

**for** (i = 0; i < m; i ++)

{

n = w = pow (2, i);

w = (PI / n);

k = 0;

**while** (k < (numero\_muestras - 1))

{

**for** (j = 0; j < n; j ++)

{

arg = (-j \* w);

c = cos (arg);

s = sin (arg);

j1 = (k + j);

tempr = ((xr [j1 + n] \* c) - (xi [j1 + n] \* s));

tempi = ((xi [j1 + n] \* c) + (xr [j1 + n] \* s));

xr [j1 + n] = (xr [j1] - tempr);

xi [j1 + n] = (xi [j1] - tempi);

xr [j1] = (xr [j1] + tempr);

xi [j1] = (xi [j1] + tempi);

}

k += (2 \* n);

}

}

**for** (i = 0; i < numero\_muestras; i ++)

{

parte\_real = xr [i];

parte\_imaginaria = xi [i];

parte\_real /= 2;

parte\_imaginaria /= 2;

magnitud = sqrt ((parte\_real \* parte\_real) + (parte\_imaginaria \* parte\_imaginaria));

**if** (magnitud == 0)

fase = 0;

**else**

{

**if** ((parte\_real == 0) && (parte\_imaginaria < 0))

fase = (**short**) (- PI / 2);

**else** **if** ((parte\_real == 0) && (parte\_imaginaria > 0))

fase = (**short**) (PI / 2);

**else** **if** ((parte\_real < 0) && (parte\_imaginaria >= 0))

fase = (**short**) (atan (parte\_imaginaria / parte\_real) + PI);

**else** **if** ((parte\_real < 0) && (parte\_imaginaria < 0))

fase = (**short**) (atan (parte\_imaginaria / parte\_real) - PI);

}

fwrite (&magnitud, **sizeof** (**short**), 1, salida);

fwrite (&fase, **sizeof** (**short**), 1, salida);

}

uswtime (&utime1, &wtime1);

calculaTiempo (utime0, wtime0, utime1, wtime1, numero\_muestras, 3);

}

**void** swap (**float** \* a, **float** \* b)

{

**float** temp;

temp = \*a;

\*a = \*b;

\*b = temp;

**return**;

}

**void** uswtime (**double** \* usertime, **double** \* walltime)

{

**double** mega = 1.0e-6;

**struct** rusage buffer;

**struct** timeval tp;

**struct** timezone tzp;

getrusage (RUSAGE\_SELF, &buffer);

gettimeofday (&tp, &tzp);

\*usertime = (**double**) buffer.ru\_utime.tv\_sec +1.0e-6 \* buffer.ru\_utime.tv\_usec;

\*walltime = (**double**) tp.tv\_sec + 1.0e-6 \* tp.tv\_usec;

}

**void** calculaTiempo (**double** utime0, **double** wtime0, **double** utime1, **double** wtime1, **int** n, **int** opcion)

{

**char** \* option = (**char** \*) malloc (**sizeof** (**char**));

**switch** (opcion)

{

**case** 1:

strcpy (option, "Opcion 1");

**break**;

**case** 2:

strcpy (option, "Opcion 2");

**break**;

**case** 3:

strcpy (option, "Opcion 3");

**break**;

}

printf("Tiempo con la %s para un archivo de %d muestras:\n\n", option, n);

printf("Tiempo total: %.10f s\n", wtime1 - wtime0);

printf("Tiempo de procesamiento en CPU: %.10f s\n", utime1 - utime0);

printf ("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

}

**FFT.c**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include "Cabecera.h"

**int** main(**int** argc, **char** **const** \*argv[])

{

FILE \* entrada, \* salida;

cabecera cab;

**short** muestra;

**int** opcion\_seleccionada, i, numero\_muestras;

**char** \* archivo\_salida = (**char** \*) malloc (**sizeof** (**char**));

**char** \* archivo\_entrada = (**char** \*) malloc (**sizeof** (**char**));

**if** (argc < 4)

{

printf("Error, faltan argumentos.\n");

printf ("Ejemplo: '%s -1 Entrada.wav Salida.wav'\n\n", argv [0]);

exit (0);

}**else**

{

opcion\_seleccionada = atoi (argv [1]);

archivo\_entrada = (**char** \*) argv [2];

archivo\_salida = (**char** \*) argv [3];

}

//Abrimos los archivos en modo binario

entrada = fopen (archivo\_entrada, "rb");

**if** (entrada == NULL)

{

printf ("Error al abrir el archivo: '%s'", archivo\_entrada);

exit (0);

}

salida = fopen (archivo\_salida, "wb");

**if** (salida == NULL)

{

printf ("Error al crear el archivo: '%s'", archivo\_salida);

exit (0);

}

//Leer la cabecera del archivo de entrada

fread (&cab, 44, 1, entrada);

numero\_muestras = (cab.SubChunk2Size / cab.BlockAlign);

//Modificar algunos par�metros de la cabecera

cab.ChunkSize = (cab.ChunkSize \* 2);

cab.NumChannels = (**short**) 2;

cab.ByteRate = (cab.ByteRate \* 2);

cab.BlockAlign = (**short**) 4;

cab.BitsPerSample = (**short**) 16;

cab.SubChunk2Size = (cab.SubChunk2Size \* 2);

//Copiar la cabecera modificada al archivo de salida

fwrite (&cab, 44, 1, salida);

//Imprimir los valores de la cabecera

imprimir\_cabecera (&cab);

**float** \* re = (**float** \*) malloc (**sizeof** (**float**) \* numero\_muestras);

**float** \* im = (**float** \*) malloc (**sizeof** (**float**) \* numero\_muestras);

**for** (i = 0; i < numero\_muestras; i ++)

{

fread (&muestra, **sizeof** (**short**), 1, entrada);

re [i] = (**float**) muestra;

im [i] = 0.0;

}

**if** (opcion\_seleccionada == -1)

opcion\_uno (salida, re, im, numero\_muestras);

**else** **if** (opcion\_seleccionada == -2)

opcion\_dos (salida, re, im, numero\_muestras);

**else** **if** (opcion\_seleccionada == -3)

opcion\_tres (salida, re, im, numero\_muestras);

**else**

{

printf ("\nOpcion invalida");

exit (0);

}

fclose (entrada);

fclose (salida);

**return** 0;

}

**Cabecera.h (Inversa)**

**typedef** **struct** CABECERA

{

**char** ChunkID[4]; //Contiene 'RIFF'

**int** ChunkSize; //Contiene el tama�o total del archivo - 8 bytes

**char** Format[4]; //Contiene 'WAVE'

//Aqui comienza el primer subchunk 'fmt'

**char** SubChunk1ID[4]; //Contiene 'fmt'

**int** SubChunk1Size; //Contiene el tama�o del primer subchunk

**short** AudioFormat; //Formato de audio

**short** NumChannels; //Numero de canales

**int** SampleRate; //Frecuencia de muestreo

**int** ByteRate; //(SampleRate \* Numero canales \* Bits per Sample) / 8

**short** BlockAlign; //Bytes por muestra

**short** BitsPerSample; //8 bits, 16 bits, etc.

//Aqui comienza el segundo subchunk 'data'

**char** SubChunk2ID[4]; //Contiene 'data'

**int** SubChunk2Size; //Numero de bytes en los datos

}cabecera;

**void** imprimir\_cabecera (cabecera \* cab);

**void** transformada\_inversa (FILE \* salida, **float** \* re, **float** \* im, **int** numero\_muestras);

**void** swap (**float** \* a, **float** \* b);

**void** uswtime (**double** \* usertime, **double** \* walltime);

**void** calculaTiempo (**double** utime0, **double** wtime0, **double** utime1, **double** wtime1, **int** n, **int** opcion);

**Cabecera.c (Inversa)**

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <string.h>

#include <sys/resource.h>

#include <sys/time.h>

#include "Cabecera.h"

#define PI 3.14159265

**double** utime0, wtime0, utime1, wtime1;

**void** imprimir\_cabecera (cabecera \* cab)

{

**char** \* formatoArchivo = (**char** \*) malloc (**sizeof** (**char**));

printf ("\n\n");

printf ("(1-4) Chunk ID: %s\n", cab -> ChunkID);

printf ("(5-8) ChunkSize: %u\n", cab -> ChunkSize);

printf ("(9-12) Format: %s\n", cab -> Format);

printf ("(13-16) SubChunk 1 ID: %s\n", cab -> SubChunk1ID);

printf ("(17-20) SubChunk 1 Size: %u\n", cab -> SubChunk1Size);

**if** (cab -> AudioFormat == 1)

strcpy (formatoArchivo, "PCM");

printf ("(21-22) Audio Format: %u, %s\n", cab -> AudioFormat,formatoArchivo);

**if** (cab -> NumChannels == 1)

strcpy (formatoArchivo, "Mono");

**else**

strcpy (formatoArchivo, "Stereo");

printf ("(23-24) Number of Channels: %u, Tipo: %s\n", cab -> NumChannels,formatoArchivo);

printf ("(25-28) Sample Rate: %u\n", cab -> SampleRate);

printf ("(29-32) Byte Rate: %u BitRate: %u\n", cab -> ByteRate,cab -> ByteRate \* 8);

printf ("(33-34) Block Align: %u\n", cab -> BlockAlign);

printf ("(35-36) Bits Per Sample: %u\n", cab -> BitsPerSample);

printf ("(37-40) SubChunk 2 ID: %s\n", cab -> SubChunk2ID);

printf ("(41-44) SubChunk 2 Size: %u\n\n\n", cab -> SubChunk2Size);

}

**void** transformada\_inversa (FILE \* salida, **float** \* re, **float** \* im, **int** numero\_muestras)

{

**int** i, j, k, j1, m, n;

**float** arg, s, c, w, tempr, tempi;

**short** parte\_real, parte\_imaginaria;

m = (log (numero\_muestras) / log (2));

fseek (salida, 44, SEEK\_SET);

uswtime (&utime0, &wtime0);

//Bit Reversal

**for** (i = 0; i < numero\_muestras; ++ i)

{

j = 0;

**for** (k = 0; k < m; ++ k)

j = (j << 1) | (1 & (i >> k));

**if** (j < i)

{

swap (&xr [i], &xr [j]);

swap (&xi [i], &xi [j]);

}

}

**for** (i = 0; i < m; i ++)

{

n = w = pow (2, i);

w = ((PI / n) \* -1);

k = 0;

**while** (k < (numero\_muestras - 1))

{

**for** (j = 0; j < n; j ++)

{

arg = (-j \* w);

c = cos (arg);

s = sin (arg);

j1 = (k + j);

tempr = ((xr [j1 + n] \* c) - (xi [j1 + n] \* s));

tempi = ((xi [j1 + n] \* c) + (xr [j1 + n] \* s));

xr [j1 + n] = (xr [j1] - tempr);

xi [j1 + n] = (xi [j1] - tempi);

xr [j1] = (xr [j1] + tempr);

xi [j1] = (xi [j1] + tempi);

}

k += (2 \* n);

}

}

**for** (i = 0; i < numero\_muestras; i ++)

{

parte\_real = xr [i];

parte\_imaginaria = xi [i];

parte\_real /= 2;

parte\_imaginaria /= 2;

fwrite (&parte\_real, **sizeof** (**short**), 1, salida);

fwrite (&parte\_imaginaria, **sizeof** (**short**), 1, salida);

}

uswtime (&utime1, &wtime1);

calculaTiempo (utime0, wtime0, utime1, wtime1, numero\_muestras, 2);

}

**void** swap (**float** \* a, **float** \* b)

{

**float** temp;

temp = \*a;

\*a = \*b;

\*b = temp;

**return**;

}

**void** uswtime (**double** \* usertime, **double** \* walltime)

{

**double** mega = 1.0e-6;

**struct** rusage buffer;

**struct** timeval tp;

**struct** timezone tzp;

getrusage (RUSAGE\_SELF, &buffer);

gettimeofday (&tp, &tzp);

\*usertime = (**double**) buffer.ru\_utime.tv\_sec +1.0e-6 \* buffer.ru\_utime.tv\_usec;

\*walltime = (**double**) tp.tv\_sec + 1.0e-6 \* tp.tv\_usec;

}

**void** calculaTiempo (**double** utime0, **double** wtime0, **double** utime1, **double** wtime1, **int** n, **int** opcion)

{

**char** \* option = (**char** \*) malloc (**sizeof** (**char**));

**switch** (opcion)

{

**case** 1:

strcpy (option, "Opcion 1");

**break**;

**case** 2:

strcpy (option, "Opcion 2");

**break**;

**case** 3:

strcpy (option, "Opcion 3");

**break**;

}

printf("Tiempo para un archivo de %d muestras:\n\n", option, n);

printf("Tiempo total: %.10f s\n", wtime1 - wtime0);

printf("Tiempo de procesamiento en CPU: %.10f s\n", utime1 - utime0);

printf ("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

}

**IFFT.c**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include "Cabecera.h"

**int** main(**int** argc, **char** **const** \*argv[])

{

FILE \* entrada, \* salida;

cabecera cab;

**short** muestra;

**int** i, numero\_muestras;

**char** \* archivo\_salida = (**char** \*) malloc (**sizeof** (**char**));

**char** \* archivo\_entrada = (**char** \*) malloc (**sizeof** (**char**));

**if** (argc < 3)

{

printf("Error, faltan argumentos.\n");

printf ("Ejemplo: '%s Entrada.wav Salida.wav'\n\n", argv [0]);

exit (0);

}**else**

{

archivo\_entrada = (**char** \*) argv [2];

archivo\_salida = (**char** \*) argv [3];

}

//Abrimos los archivos en modo binario

entrada = fopen (archivo\_entrada, "rb");

**if** (entrada == NULL)

{

printf ("Error al abrir el archivo: '%s'", archivo\_entrada);

exit (0);

}

salida = fopen (archivo\_salida, "wb");

**if** (salida == NULL)

{

printf ("Error al crear el archivo: '%s'", archivo\_salida);

exit (0);

}

//Leer la cabecera del archivo de entrada

fread (&cab, 44, 1, entrada);

fwrite (&cab, 44, 1, salida);

numero\_muestras = (cab.SubChunk2Size / cab.BlockAlign);

//Imprimir los valores de la cabecera

imprimir\_cabecera (&cab);

**float** \* re = (**float** \*) malloc (**sizeof** (**float**) \* numero\_muestras);

**float** \* im = (**float** \*) malloc (**sizeof** (**float**) \* numero\_muestras);

**for** (i = 0; i < numero\_muestras; i ++)

{

fread (&muestra, **sizeof** (**short**), 1, entrada);

re [i] = (**float**) muestra;

fread (&muestra, **sizeof** (**short**), 1, entrada);

im [i] = (**float**) muestra;

}

transformada\_inversa (salida, re, im, numero\_muestras);

fclose (entrada);

fclose (salida);

**return** 0;

}