Reporte 04A.- Transformada Discreta de Fourier

Alumno: Monroy Martos Elioth
Boleta: 2016630258
Profesor: Gutierrez Aldana Eduardo
Materia: Teoría de Comunicaciones y Señales
Grupo: 3CM6

10 de diciembre de 2017

Índice

1. Introducción

El análisis de Fourier es una familia de técnicas matemáticas, basadas en la descomposición de señales en señales sinusoidales. La Transformada Discreta de Fourier (TDF) es un miembro de esta familia la cual es usada para trabajar con señales digitales.
El objetivo de la descomposición de la señal es obtener algo más sencillo con lo cual trabajar (en este caso, con senos y cosenos) que lo que se tenia originalmente.
Para el tratamiento de señales discretas se usa la TDF, la cual esta definida en la Figura 1:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} kn} \qquad k = 0, \ldots, N-1$$

Figura 1: Transformada Discreta de Fourier

Al usar la identidad de Euler (Figura 2) se puede obtener otra versión de la TDF donde la parte real este contenida en una sumatoria de cosenos y la parte imaginaria contenida en una sumatoria de senos.

$$e^{ix} = \cos x + i \operatorname{sen} x$$

Figura 2: Identidad de Euler

Lo cual se puede apreciar en la siguiente figura:

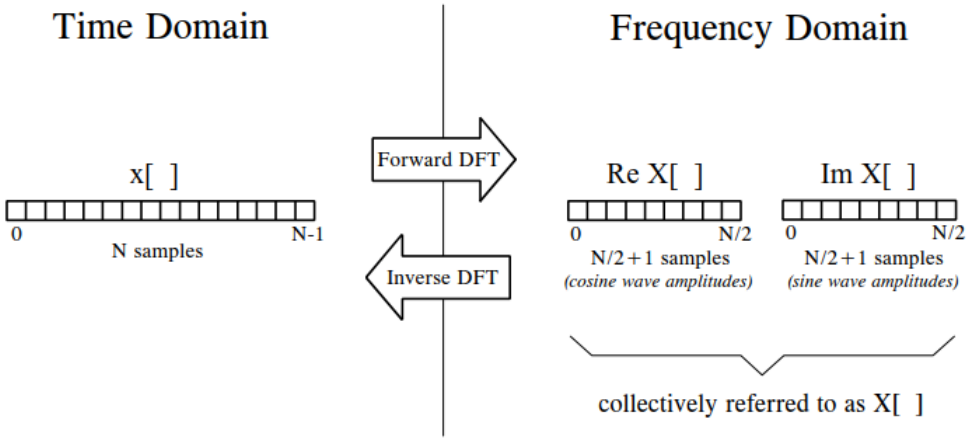

Figura 3: Parte Real e Imaginaria de la TDF

Para este trabajo, cada una de las partes de la TDF (real e imaginaria) serán contenidas en un canal de un archivo wav, canal izquierdo para la parte real y canal derecho para la parte imaginaria. Por lo cual, como entrada para el programa de la TDF se recibirá un archivo de un canal (muestras de la señal discreta) y se obtendrá como salida un archivo wav de dos canales, donde el archivo de salida tendrá el mismo número de muestras que el archivo de entrada, pero el tamaño de las mismas ahora será de 4 bytes (2 bytes por el canal izquierdo y 2 bytes por el canal derecho).
Así como fue realizada la práctica del reporte 01A donde se simulaba un circuito RC (filtro pasabajas) mediante el uso de la convolución, en esta práctica se busca simular lo mismo, pero en lugar de usar la convolución, se usará el producto en frecuencia, lo cual es equivalente a realizar la convolución en el tiempo. Para realizar ese procedimiento, es necesario primero aplica la TDF a la señal de entrada, y posteriormente la salida obtenida multiplicarla por el filtro ideal en frecuencia que se desarrolló.
Finalmente al resultado de la multiplicación es necesario ingresarlo al programa que calcula la TDF Inversa, la cual regresa al dominio del tiempo una señal que se encuentra en el dominio de la frecuencia (en este caso, el archivo que fue resultado de la multiplicación).
Cabe señalar, que el programa que calcula la TDF tiene 4 modos de ejecución, donde además de obtener la TDF se puede obtener otra información importante como la magnitud y fase de la transformada.

2. Código

La implementación del filtro mediante multiplicación de complejos en frecuencia se realizó con los siguientes programas:
funciones.h:

```
#ifndef __FUNCIONES_H__
#define __FUNCIONES_H__
  //Librerías de C
  #include <stdio.h>
  #include <stdlib.h>
  #include <math.h>
  //Librería que contiene los máximos y mínimos de los
    diferentes tipos de datos en c
  #include <limits.h>
  //Libreria para conocer tiempo de ejecución
  #include <time.h>
  //Metodos
  void leerCabeceras(char**);
  void escribirArchivo(short*,short*);
  void leerMuestras(short*);
  void leerMuestras2Canales(short*,short*);
  //Cabeceras
  int chunkid;
  int chunksize;
  int format;
  int subchunk1id;
  int subchunk1size;
  short audioformat;
  short numchannels;
  int samplerate;
  int byterate;
  short blockalign;
  short bitspersample;
  int subchunk2id;
  int subchunk2size;
  //Archivo
  FILE* entrada;
  FILE* salida;
  int modo;
  //Variables para muestras
  short muestra;
  int total_muestras;
  short headers[37];
  //Métodos TDF
```

```
#define PI acos(-1.0)//Defino la constante PI
void calcularTDF(short*);
void calcularTDF1(short*);
void calcularTDF2(short*);
void calcularTDF3(short*);
void calcularTDFI(short*,short*);
#endif
```

tdf.c:

```
#include"funciones.h"
int main(int argc, char *argv[]){
  //Leo las cabeceras
  leerCabeceras(argv);
  //Defino variables
  total_muestras=subchunk2size/blockalign;
  short *muestras=(short *)malloc(total_muestras * sizeof(short)
    );
  printf("Total de muestras:%d\n", total_muestras);
  //Leo las muestras
  leerMuestras(muestras);
  //Calculo la TDF
  if (modo==0){
    //Transformada normal
    calcularTDF(muestras);
  }else if (modo==1){
    //Canal izquierdo señal original, derecho magnitud
    calcularTDF1(muestras);
  }else if(modo==2){
    //Canal izquierdo parte real TDF, derecho magnitud
    calcularTDF2(muestras);
  }else if(modo==3){
    //Canal izquierdo magnitud TDF, derecho fase TDF
    calcularTDF3(muestras);
  }
}
void leerCabeceras(char ** argv){
  //Primero voy a recordar a leer archivos
  entrada = fopen(argv[1], "rb");
  salida=fopen(argv[2],"wb");
  modo=atoi(argv[3]);
  if(!entrada){
    perror("\nFile opening failed");
    exit(0);
  }
  fread(&chunkid,sizeof(int),1,entrada);
```

```
36    fread(&chunksize,sizeof(int),1,entrada);
37    fread(&format,sizeof(int),1,entrada);
38    fread(&subchunk1id,sizeof(int),1,entrada);
39    fread(&subchunk1size,sizeof(int),1,entrada);
40    fread(&audioformat,sizeof(short),1,entrada);
41    fread(&numchannels,sizeof(short),1,entrada);
42    fread(&samplerate,sizeof(int),1,entrada);
43    fread(&byterate,sizeof(int),1,entrada);
44    fread(&blockalign,sizeof(short),1,entrada);
45    fread(&bitspersample,sizeof(short),1,entrada);
46    fread(&subchunk2id,sizeof(int),1,entrada);
47    fread(&subchunk2size,sizeof(int),1,entrada);
48  }
49  void leerMuestras(short *muestras){
50    int i=0;
51    while (feof(entrada) == 0){
52      if(i<total_muestras){
53        fread(&muestra,sizeof(short),1,entrada);
54        muestras[i]=muestra;
55        i++;
56      }else{
57        fread(&headers,sizeof(short),37,entrada);
58        break;
59      }
60    }
61  }
62  void escribirArchivo(short* muestrasRe,short* muestrasIm){
63    //Escribo el archivo
64    fwrite(&chunkid,sizeof(int),1,salida);
65    fwrite(&chunksize,sizeof(int),1,salida);
66    fwrite(&format,sizeof(int),1,salida);
67    fwrite(&subchunk1id,sizeof(int),1,salida);
68    fwrite(&subchunk1size,sizeof(int),1,salida);
69    fwrite(&audioformat,sizeof(short),1,salida);
70    fwrite(&numchannels,sizeof(short),1,salida);
71    fwrite(&samplerate,sizeof(int),1,salida);
72    fwrite(&byterate,sizeof(int),1,salida);
73    fwrite(&blockalign,sizeof(short),1,salida);
74    fwrite(&bitspersample,sizeof(short),1,salida);
75    fwrite(&subchunk2id,sizeof(int),1,salida);
76    fwrite(&subchunk2size,sizeof(int),1,salida);
77    //Ahora escribo las muestras
78    int i=0;
79    for(i=0;i<total_muestras;i++){
80      fwrite(&muestrasRe[i],sizeof(short),1,salida);
```

```
81      fwrite(&muestrasIm[i],sizeof(short),1,salida);
82    }
83    //Y por último los headers de goldwave
84    for(i=0;i<37;i++){
85      fwrite(&headers[i],sizeof(short),1,salida);
86    }
87  }
88  void calcularTDF(short* muestras){
89    //Aquí va el algoritmo para la TDF
90    short *Xre=(short *)malloc(total_muestras * sizeof(short));
91    short *Xim=(short *)malloc(total_muestras * sizeof(short));
92    //Variables para obtener tiempo de ejecución
93    clock_t inicio, final;
94    double total;
95    inicio = clock();
96    //Algoritmo TDF
97    int n,k;
98    for (k = 0; k < total_muestras; k++){
99        Xre[k]=0;
100       Xim[k]=0;
101     for (n = 0; n < total_muestras; n++){
102       Xre[k]+=(muestras[n]/total_muestras)*cos(2*k*n*PI/
      total_muestras);
103       Xim[k]-=(muestras[n]/total_muestras)*sin(2*k*n*PI/
      total_muestras);
104     }
105   }
106   //Obtener tiempo e imprimir
107   final = clock();
108   total = (double)(final - inicio) / CLOCKS_PER_SEC;
109   printf("Tiempo de ejecucion: %f\n", total);
110   //La salida ahora sera un archivo tipo estereo (2 canales)
111   //Por lo cual hay que cambiar el el numero de canales del
       archivo y todas las demas cabeceras que dependan de esta
112   chunksize-=subchunk2size;
113   numchannels*=2;
114   byterate*=numchannels;
115   blockalign*=numchannels;
116   subchunk2size*=numchannels;
117   chunksize+=subchunk2size;
118   escribirArchivo(Xre,Xim);
119 }
120 void calcularTDF1(short *muestras){
121   //Canal izquierdo señal original, derecho magnitud
122   short *Xre=(short *)malloc(total_muestras * sizeof(short));
```

```
123   short *Xim=(short *)malloc(total_muestras * sizeof(short));
124   short *magnitud=(short *)malloc(total_muestras * sizeof(short)
      );
125   //Variables para obtener tiempo de ejecución
126   clock_t inicio, final;
127   double total;
128   inicio = clock();
129   //Algoritmo TDF
130   int n,k;
131   for (k = 0; k < total_muestras; k++){
132     Xre[k]=0;
133     Xim[k]=0;
134     for (n = 0; n < total_muestras; n++){
135       Xre[k]+=(muestras[n]/total_muestras)*cos(2*k*n*PI/
      total_muestras);
136       Xim[k]-=(muestras[n]/total_muestras)*sin(2*k*n*PI/
      total_muestras);
137     }
138   }
139   //Obtener tiempo e imprimir
140   final = clock();
141   total = (double)(final - inicio) / CLOCKS_PER_SEC;
142   printf("Tiempo de ejecucion: %f\n", total);
143   //Ahora calcularé la magnitud de la transformada
144   for (k = 0; k < total_muestras; k++){
145     magnitud[k]=sqrt(pow(Xre[k],2)+pow(Xim[k],2));
146   }
147   //La salida ahora sera un archivo tipo estereo (2 canales)
148   //Por lo cual hay que cambiar el numero de canales del archivo
149   //y todas las demas cabeceras que dependan de esta
150   chunksize-=subchunk2size;
151   numchannels*=2;
152   byterate*=numchannels;
153   blockalign*=numchannels;
154   subchunk2size*=numchannels;
155   chunksize+=subchunk2size;
156   escribirArchivo(muestras,magnitud);
157 }
158 void calcularTDF2(short *muestras){
159   //Canal izquierdo parte real transformada, derecho magnitud
160   short *Xre=(short *)malloc(total_muestras * sizeof(short));
161   short *Xim=(short *)malloc(total_muestras * sizeof(short));
162   short *magnitud=(short *)malloc(total_muestras * sizeof(short)
      );
163   //Variables para obtener tiempo de ejecución
```

```
164   clock_t inicio, final;
165   double total;
166   inicio = clock();
167   //Algoritmo TDF
168   int n,k;
169   for (k = 0; k < total_muestras; k++){
170     Xre[k]=0;
171     Xim[k]=0;
172     for (n = 0; n < total_muestras; n++){
173       Xre[k]+=(muestras[n]/total_muestras)*cos(2*k*n*PI/
      total_muestras);
174       Xim[k]-=(muestras[n]/total_muestras)*sin(2*k*n*PI/
      total_muestras);
175     }
176   }
177   //Obtener tiempo e imprimir
178   final = clock();
179   total = (double)(final - inicio) / CLOCKS_PER_SEC;
180   printf("Tiempo de ejecucion: %f\n", total);
181   //Ahora calcularé la magnitud de la transformada
182   for (k = 0; k < total_muestras; k++){
183     magnitud[k]=sqrt(pow(Xre[k],2)+pow(Xim[k],2));
184   }
185   //La salida ahora sera un archivo tipo estereo (2 canales)
186   //Por lo cual hay que cambiar el numero de canales del archivo
187   //y todas las demas cabeceras que dependan de esta
188   chunksize-=subchunk2size;
189   numchannels*=2;
190   byterate*=numchannels;
191   blockalign*=numchannels;
192   subchunk2size*=numchannels;
193   chunksize+=subchunk2size;
194   escribirArchivo(Xre,magnitud);
195 }
196 void calcularTDF3(short *muestras){
197   //Canal izquierdo magnitud, derecho fase
198   short *Xre=(short *)malloc(total_muestras * sizeof(short));
199   short *Xim=(short *)malloc(total_muestras * sizeof(short));
200   short *magnitud=(short *)malloc(total_muestras * sizeof(short)
      );
201   short *fase=(short *)malloc(total_muestras * sizeof(short));
202   //Variables para obtener tiempo de ejecución
203   clock_t inicio, final;
204   double total;
205   inicio = clock();
```

```
206   //Algoritmo TDF
207   int n,k;
208   for (k = 0; k < total_muestras; k++){
209     Xre[k]=0;
210     Xim[k]=0;
211     for (n = 0; n < total_muestras; n++){
212       Xre[k]+=(muestras[n]/total_muestras)*cos(2*k*n*PI/
      total_muestras);
213       Xim[k]-=(muestras[n]/total_muestras)*sin(2*k*n*PI/
      total_muestras);
214     }
215   }
216   //Obtener tiempo e imprimir
217   final = clock();
218   total = (double)(final - inicio) / CLOCKS_PER_SEC;
219   printf("Tiempo de ejecucion: %f\n", total);
220   //Ahora calcularé la magnitud de la transformada
221   for (k = 0; k < total_muestras; k++){
222     magnitud[k]=sqrt(pow(Xre[k],2)+pow(Xim[k],2));
223   }
224   //La fase
225   float valor=180.0/PI;
226   //printf("Valor %f",valor);
227   for (k = 0; k < total_muestras; k++){
228     if (magnitud[k]>1000){
229       //atan nos devuelve un valor en radianes, hay que pasarlo
      a grados
230       if(Xre[k]==0){
231         if (Xim[k]==0){
232           fase[k]=0;
233         }else if (Xim[k]<0){//Xim es negativa
234           fase[k]=-90;//o 270, tengo que checarlo
235         }else{
236           fase[k]=90;
237         }
238       }else{
239         //printf("arctan:%f\n", atan(Xim[k]/Xre[k]));
240         fase[k]=atan(Xim[k]/Xre[k])*valor;
241       }
242     }else{
243       fase[k]=0;
244     }
245     fase[k]=(fase[k]*SHRT_MAX)/180;//Para que se pueda ver en
      goldwave
246   }
```

```
247   //La salida ahora sera un archivo tipo estereo (2 canales)
248   //Por lo cual hay que cambiar el numero de canales del archivo
249   //y todas las demas cabeceras que dependan de esta
250   chunksize-=subchunk2size;
251   numchannels*=2;
252   byterate*=numchannels;
253   blockalign*=numchannels;
254   subchunk2size*=numchannels;
255   chunksize+=subchunk2size;
256   escribirArchivo(magnitud,fase);
257 }
```

tdfi.c:

```
1  #include"funciones.h"
2  int main(int argc, char *argv[]){
3    //Leo las cabeceras
4    leerCabeceras(argv);
5    //Defino variables
6    total_muestras=subchunk2size/blockalign;
7    //printf("Total Muestras: %d\n", total_muestras);
8    short *muestrasRe=(short *)malloc(total_muestras * sizeof(
       short));
9    short *muestrasIm=(short *)malloc(total_muestras * sizeof(
       short));
10   //short muestrasRe[total_muestras],muestrasIm[total_muestras];
11   //Leo las muestras
12   leerMuestras2Canales(muestrasRe,muestrasIm);
13   //Calculo la TDF
14   calcularTDFI(muestrasRe,muestrasIm);
15 }
16 void leerCabeceras(char ** argv){
17   //Primero voy a recordar a leer archivos
18   entrada = fopen(argv[1], "rb");
19   salida=fopen(argv[2],"wb");
20   if(!entrada) {
21     perror("\nFile opening failed");
22     exit(0);
23   }
24   fread(&chunkid,sizeof(int),1,entrada);
25   fread(&chunksize,sizeof(int),1,entrada);
26   fread(&format,sizeof(int),1,entrada);
27   fread(&subchunk1id,sizeof(int),1,entrada);
28   fread(&subchunk1size,sizeof(int),1,entrada);
29   fread(&audioformat,sizeof(short),1,entrada);
30   fread(&numchannels,sizeof(short),1,entrada);
```

```
31    fread(&samplerate,sizeof(int),1,entrada);
32    fread(&byterate,sizeof(int),1,entrada);
33    fread(&blockalign,sizeof(short),1,entrada);
34    fread(&bitspersample,sizeof(short),1,entrada);
35    fread(&subchunk2id,sizeof(int),1,entrada);
36    fread(&subchunk2size,sizeof(int),1,entrada);
37  }
38  void leerMuestras2Canales(short *muestrasRe,short* muestrasIm){
39    int i=0;
40    while (feof(entrada) == 0){
41      if(i<total_muestras){
42        fread(&muestrasRe[i],sizeof(short),1,entrada);
43        fread(&muestrasIm[i],sizeof(short),1,entrada);
44        i++;
45        //printf("Muestra %s: %d\n",i,muestras[i-1]);
46      }else{
47        fread(&headers,sizeof(short),37,entrada);
48        break;
49      }
50    }
51  }
52  void escribirArchivo(short* muestrasRe,short* muestrasIm){
53    //Escribo el archivo
54    fwrite(&chunkid,sizeof(int),1,salida);
55    fwrite(&chunksize,sizeof(int),1,salida);
56    fwrite(&format,sizeof(int),1,salida);
57    fwrite(&subchunk1id,sizeof(int),1,salida);
58    fwrite(&subchunk1size,sizeof(int),1,salida);
59    fwrite(&audioformat,sizeof(short),1,salida);
60    fwrite(&numchannels,sizeof(short),1,salida);
61    fwrite(&samplerate,sizeof(int),1,salida);
62    fwrite(&byterate,sizeof(int),1,salida);
63    fwrite(&blockalign,sizeof(short),1,salida);
64    fwrite(&bitspersample,sizeof(short),1,salida);
65    fwrite(&subchunk2id,sizeof(int),1,salida);
66    fwrite(&subchunk2size,sizeof(int),1,salida);
67    //Ahora escribo las muestras
68    int i=0;
69    for(i=0;i<total_muestras;i++){
70      fwrite(&muestrasRe[i],sizeof(short),1,salida);
71      fwrite(&muestrasIm[i],sizeof(short),1,salida);
72    }
73    //Y por último los headers de goldwave
74    for(i=0;i<37;i++){
75      fwrite(&headers[i],sizeof(short),1,salida);
```

```
  }
}
void calcularTDFI(short* muestrasRe,short* muestrasIm){
  //Aquí va el algoritmo para la TDF
  short *Xre=(short *)malloc(total_muestras * sizeof(short));
  short *Xim=(short *)malloc(total_muestras * sizeof(short));
  //Variables para obtener tiempo de ejecución
  clock_t inicio, final;
  double total;
  inicio = clock();
  //Algoritmo TDFI
  int n,k;
  double algo;
  for (k = 0; k < total_muestras; k++){
    Xre[k]=0;
    Xim[k]=0;
    for (n = 0; n < total_muestras; n++){
      //Lo que va dentro del coseno y seno.
      algo=(2*k*n*PI)/(total_muestras);
      //Calculo las partes real e imaginarias
      Xre[k]+=muestrasRe[n]*cos(algo)-muestrasIm[n]*sin(algo);
      Xim[k]+=muestrasRe[n]*sin(algo)+muestrasIm[n]*cos(algo);
    }
  }
  //Obtener tiempo e imprimir
  final = clock();
  total = (double)(final - inicio) / CLOCKS_PER_SEC;
  printf("Tiempo de ejecucion: %f\n", total);
  //La salida seguirá siendo un archivo tipo estereo (2 canales)
    por lo cual no hay que alinear nada
  escribirArchivo(Xre,Xim);
}
```

funciones2.h:

```
#ifndef __FUNCIONES2_H__
#define __FUNCIONES2_H__
  //Librerías de C
  #include <stdio.h>
  #include <stdlib.h>
  //Librería que contiene los máximos y mínimos de los
    diferentes tipos de datos en c
  #include <limits.h>
  //Metodos
  void leerCabeceras(char**);
  void leerMuestras(short*,short*);
```

```
11    void leerMuestras2Canales(short*,short*,short*,short*);
12    void escribirArchivo(short*,int);
13    void escribirArchivo2Canales(short*,short*,int);
14    void calcularProducto(short*,short*);
15    void calcularProductoComplejos(short*,short*,short*,short*);
16    //Cabeceras primer archivo
17    int chunkid1;
18    int chunksize1;
19    int format1;
20    int subchunk1id1;
21    int subchunk1size1;
22    short audioformat1;
23    short numchannels1;
24    int samplerate1;
25    int byterate1;
26    short blockalign1;
27    short bitspersample1;
28    int subchunk2id1;
29    int subchunk2size1;
30    //Cabeceras segundo archivo
31    int chunkid2;
32    int chunksize2;
33    int format2;
34    int subchunk1id2;
35    int subchunk1size2;
36    short audioformat2;
37    short numchannels2;
38    int samplerate2;
39    int byterate2;
40    short blockalign2;
41    short bitspersample2;
42    int subchunk2id2;
43    int subchunk2size2;
44    //Archivo
45    FILE* entrada1;
46    FILE* entrada2;
47    FILE* salida;
48    //Variables para muestras primer archivo
49    int total_muestras1;
50    //Variables segundo archivo
51    int total_muestras2;
52    //Variables que se pueden reutilizar
53    short muestra;
54    short headers[37];
55 #endif
```

producto.c:

```
#include"funciones2.h"
int main(int argc, char *argv[]){
  //Leo las cabeceras
  leerCabeceras(argv);
  if(numchannels1==1){
    //Defino variables para el primer archivo
    total_muestras1=subchunk2size1/blockalign1;
    short *muestras1=(short *)malloc(total_muestras1 * sizeof(
    short));
    //Defino variables para el segundo archivo
    total_muestras2=subchunk2size2/blockalign2;
    short *muestras2=(short *)malloc(total_muestras2 * sizeof(
    short));
    //Leo las muestras del primer archivos
    leerMuestras(muestras1,muestras2);
    //Calculo el producto
    calcularProducto(muestras1,muestras2);
  }else if(numchannels1==2){
    //Defino variables para el primer archivo
    total_muestras1=subchunk2size1/blockalign1;
    short *muestrasRe1=(short *)malloc(total_muestras1 * sizeof(
    short));
    short *muestrasIm1=(short *)malloc(total_muestras1 * sizeof(
    short));
    //Defino variables para el segundo archivo
    total_muestras2=subchunk2size2/blockalign2;
    short *muestrasRe2=(short *)malloc(total_muestras2 * sizeof(
    short));
    short *muestrasIm2=(short *)malloc(total_muestras2 * sizeof(
    short));
    //Leo las muestras del primer archivos
    leerMuestras2Canales(muestrasRe1,muestrasIm1,muestrasRe2,
    muestrasIm2);
    //Calculo el producto
    calcularProductoComplejos(muestrasRe1,muestrasIm1,
    muestrasRe2,muestrasIm2);
  }
}
void leerCabeceras(char ** argv){
  //Primero voy a recordar a leer archivos
  entrada1 = fopen(argv[1], "rb");
  entrada2 = fopen(argv[2], "rb");
  salida=fopen(argv[3],"wb");
  if(!entrada1 & !entrada2){
```

```
37      perror("\nFile opening failed");
38      exit(0);
39    }
40    //Lectura de cabeceras primer archivo
41    fread(&chunkid1,sizeof(int),1,entrada1);
42    fread(&chunksize1,sizeof(int),1,entrada1);
43    fread(&format1,sizeof(int),1,entrada1);
44    fread(&subchunk1id1,sizeof(int),1,entrada1);
45    fread(&subchunk1size1,sizeof(int),1,entrada1);
46    fread(&audioformat1,sizeof(short),1,entrada1);
47    fread(&numchannels1,sizeof(short),1,entrada1);
48    fread(&samplerate1,sizeof(int),1,entrada1);
49    fread(&byterate1,sizeof(int),1,entrada1);
50    fread(&blockalign1,sizeof(short),1,entrada1);
51    fread(&bitspersample1,sizeof(short),1,entrada1);
52    fread(&subchunk2id1,sizeof(int),1,entrada1);
53    fread(&subchunk2size1,sizeof(int),1,entrada1);
54    //Lectura de cabeceras segundo archivo
55    fread(&chunkid2,sizeof(int),1,entrada2);
56    fread(&chunksize2,sizeof(int),1,entrada2);
57    fread(&format2,sizeof(int),1,entrada2);
58    fread(&subchunk1id2,sizeof(int),1,entrada2);
59    fread(&subchunk1size2,sizeof(int),1,entrada2);
60    fread(&audioformat2,sizeof(short),1,entrada2);
61    fread(&numchannels2,sizeof(short),1,entrada2);
62    fread(&samplerate2,sizeof(int),1,entrada2);
63    fread(&byterate2,sizeof(int),1,entrada2);
64    fread(&blockalign2,sizeof(short),1,entrada2);
65    fread(&bitspersample2,sizeof(short),1,entrada2);
66    fread(&subchunk2id2,sizeof(int),1,entrada2);
67    fread(&subchunk2size2,sizeof(int),1,entrada2);
68  }
69  void leerMuestras(short *muestras1, short *muestras2){
70    int i=0;
71    //Lectura muestras primer archivo
72    while (feof(entrada1) == 0){
73      if(i<total_muestras1){
74        fread(&muestra,sizeof(short),1,entrada1);
75        printf("Muestra %d del primer archivo:%hi\n",i+1,muestra);
76        muestras1[i]=muestra;
77        i++;
78      }else{
79        fread(&headers,sizeof(short),37,entrada1);
80        break;
81      }
```

```
82    }
83    //Lectura muestras segundo archivo
84    i=0;
85    while (feof(entrada2) == 0){
86      if(i<total_muestras2){
87        fread(&muestra,sizeof(short),1,entrada2);
88        printf("Muestra %d segundo archivo:%hi\n", muestra);
89        muestras2[i]=muestra;
90        i++;
91      }else{
92        fread(&headers,sizeof(short),37,entrada2);
93        break;
94      }
95    }
96  }
97  void leerMuestras2Canales(short *muestrasRe1,short* muestrasIm1,
      short* muestrasRe2, short* muestrasIm2){
98    int i=0;
99    //Lectura de muestras del primer archivo
100   while (feof(entrada1) == 0){
101     if(i<total_muestras1){
102       fread(&muestrasRe1[i],sizeof(short),1,entrada1);
103       fread(&muestrasIm1[i],sizeof(short),1,entrada1);
104       //printf("Muestra Real %d: %hi\n",i+1,muestrasRe1[i]);
105       //printf("Muestra Imaginaria %d: %hi\n",i+1,muestrasIm1[i
      ]);
106       i++;
107     }else{
108       fread(&headers,sizeof(short),37,entrada1);
109       break;
110     }
111   }
112   i=0;
113   //Lectura de muestras del segundo archivo
114   while (feof(entrada2) == 0){
115     if(i<total_muestras2){
116       fread(&muestrasRe2[i],sizeof(short),1,entrada2);
117       fread(&muestrasIm2[i],sizeof(short),1,entrada2);
118       i++;
119     }else{
120       fread(&headers,sizeof(short),37,entrada2);
121       break;
122     }
123   }
124 }
```

```
125 void escribirArchivo(short* muestras,int decision){
126   //Escribo las cabeceras del archivo de salida
127   if (decision==0){
128     //Se escriben las cabeceras del primer archivo
129     fwrite(&chunkid1,sizeof(int),1,salida);
130     fwrite(&chunksize1,sizeof(int),1,salida);
131     fwrite(&format1,sizeof(int),1,salida);
132     fwrite(&subchunk1id1,sizeof(int),1,salida);
133     fwrite(&subchunk1size1,sizeof(int),1,salida);
134     fwrite(&audioformat1,sizeof(short),1,salida);
135     fwrite(&numchannels1,sizeof(short),1,salida);
136     fwrite(&samplerate1,sizeof(int),1,salida);
137     fwrite(&byterate1,sizeof(int),1,salida);
138     fwrite(&blockalign1,sizeof(short),1,salida);
139     fwrite(&bitspersample1,sizeof(short),1,salida);
140     fwrite(&subchunk2id1,sizeof(int),1,salida);
141     fwrite(&subchunk2size1,sizeof(int),1,salida);
142     //Ahora escribo las muestras
143     int i=0;
144     for(i=0;i<total_muestras1;i++){
145       fwrite(&muestras[i],sizeof(short),1,salida);
146     }
147     //Y por último los headers de goldwave
148     for(i=0;i<37;i++){
149       fwrite(&headers[i],sizeof(short),1,salida);
150     }
151   }else{
152     //Se escriben las cabeceras del segundo archivo
153     fwrite(&chunkid2,sizeof(int),1,salida);
154     fwrite(&chunksize2,sizeof(int),1,salida);
155     fwrite(&format2,sizeof(int),1,salida);
156     fwrite(&subchunk1id2,sizeof(int),1,salida);
157     fwrite(&subchunk1size2,sizeof(int),1,salida);
158     fwrite(&audioformat2,sizeof(short),1,salida);
159     fwrite(&numchannels2,sizeof(short),1,salida);
160     fwrite(&samplerate2,sizeof(int),1,salida);
161     fwrite(&byterate2,sizeof(int),1,salida);
162     fwrite(&blockalign2,sizeof(short),1,salida);
163     fwrite(&bitspersample2,sizeof(short),1,salida);
164     fwrite(&subchunk2id2,sizeof(int),1,salida);
165     fwrite(&subchunk2size2,sizeof(int),1,salida);
166     //Ahora escribo las muestras
167     int i=0;
168     for(i=0;i<total_muestras2;i++){
169       fwrite(&muestras[i],sizeof(short),1,salida);
```

```
170     }
171     //Y por último los headers de goldwave
172     for(i=0;i<37;i++){
173       fwrite(&headers[i],sizeof(short),1,salida);
174     }
175   }
176 }
177 void escribirArchivo2Canales(short* muestrasRe, short*
      muestrasIm,int decision){
178   //Escribo las cabeceras del archivo de salida
179   if (decision==0){
180     //Se escriben las cabeceras del primer archivo
181     fwrite(&chunkid1,sizeof(int),1,salida);
182     fwrite(&chunksize1,sizeof(int),1,salida);
183     fwrite(&format1,sizeof(int),1,salida);
184     fwrite(&subchunk1id1,sizeof(int),1,salida);
185     fwrite(&subchunk1size1,sizeof(int),1,salida);
186     fwrite(&audioformat1,sizeof(short),1,salida);
187     fwrite(&numchannels1,sizeof(short),1,salida);
188     fwrite(&samplerate1,sizeof(int),1,salida);
189     fwrite(&byterate1,sizeof(int),1,salida);
190     fwrite(&blockalign1,sizeof(short),1,salida);
191     fwrite(&bitspersample1,sizeof(short),1,salida);
192     fwrite(&subchunk2id1,sizeof(int),1,salida);
193     fwrite(&subchunk2size1,sizeof(int),1,salida);
194     //Ahora escribo las muestras
195     int i=0;
196     for(i=0;i<total_muestras1;i++){
197       fwrite(&muestrasRe[i],sizeof(short),1,salida);
198       fwrite(&muestrasIm[i],sizeof(short),1,salida);
199     }
200     //Y por último los headers de goldwave
201     for(i=0;i<37;i++){
202       fwrite(&headers[i],sizeof(short),1,salida);
203     }
204   }else{
205     //Se escriben las cabeceras del segundo archivo
206     fwrite(&chunkid2,sizeof(int),1,salida);
207     fwrite(&chunksize2,sizeof(int),1,salida);
208     fwrite(&format2,sizeof(int),1,salida);
209     fwrite(&subchunk1id2,sizeof(int),1,salida);
210     fwrite(&subchunk1size2,sizeof(int),1,salida);
211     fwrite(&audioformat2,sizeof(short),1,salida);
212     fwrite(&numchannels2,sizeof(short),1,salida);
213     fwrite(&samplerate2,sizeof(int),1,salida);
```

```
214     fwrite(&byterate2,sizeof(int),1,salida);
215     fwrite(&blockalign2,sizeof(short),1,salida);
216     fwrite(&bitspersample2,sizeof(short),1,salida);
217     fwrite(&subchunk2id2,sizeof(int),1,salida);
218     fwrite(&subchunk2size2,sizeof(int),1,salida);
219     //Ahora escribo las muestras
220     int i=0;
221     for(i=0;i<total_muestras2;i++){
222       fwrite(&muestrasRe[i],sizeof(short),1,salida);
223       fwrite(&muestrasIm[i],sizeof(short),1,salida);
224     }
225     //Y por último los headers de goldwave
226     for(i=0;i<37;i++){
227       fwrite(&headers[i],sizeof(short),1,salida);
228     }
229   }
230 }
231 void calcularProducto(short* muestras1, short* muestras2){
232   int i;
233   //Reviso que archivo tiene más muestras
234   if(total_muestras1<=total_muestras2){
235     //El segundo archivo tiene más
236     puts("El segundo archivo tiene mas muestras");
237     short *resultado=(short *)malloc(total_muestras2 * sizeof(
      short));
238     for (i = 0; i < total_muestras2; i++){
239       if(i<total_muestras1){
240         resultado[i]=(muestras1[i]*muestras2[i])/(SHRT_MAX+2);
241       }else{
242         resultado[i]=0;
243       }
244     }
245     //Escribo el resultado y cabeceras segundo archivo
246     escribirArchivo(resultado,1);
247   }else{
248   //El primer archivo tiene más
249   puts("El primer archivo tiene mas muestras");
250   short *resultado=(short *)malloc(total_muestras1 * sizeof(
      short));
251   for (i = 0; i < total_muestras1; i++){
252     if(i<total_muestras2){
253       resultado[i]=(muestras1[i]*muestras2[i])/(SHRT_MAX+2);
254     }else{
255       resultado[i]=0;
256     }
```

```
257   }
258   //Escribo el resultado y cabeceras primer archivo
259   escribirArchivo(resultado,0);
260   }
261 }
262 void calcularProductoComplejos(short* muestrasRe1, short*
      muestrasIm1, short* muestrasRe2, short* muestrasIm2){
263   int i;
264   //Reviso que archivo tiene más muestras
265   if(total_muestras1<=total_muestras2){
266   //El segundo archivo tiene más
267   puts("El segundo archivo tiene mas muestras o igual numero de
      muestras");
268   short *resultadoRe=(short *)malloc(total_muestras2 * sizeof(
      short));
269   short *resultadoIm=(short *)malloc(total_muestras2 * sizeof(
      short));
270   for (i = 0; i < total_muestras2; i++){
271     if(i<total_muestras1){
272       //Tenemos que hacer producto de complejos
273       resultadoRe[i]=((muestrasRe1[i]*muestrasRe2[i])-(
      muestrasIm1[i]*muestrasIm2[i]))/(SHRT_MAX+2);
274       resultadoIm[i]=((muestrasRe1[i]*muestrasIm2[i])+(
      muestrasRe2[i]*muestrasIm1[i]))/(SHRT_MAX+2);
275     }else{
276       resultadoRe[i]=0;
277       resultadoIm[i]=0;
278     }
279   }
280   //Escribo el resultado y cabeceras segundo archivo
281   escribirArchivo2Canales(resultadoRe,resultadoIm,1);
282   }else{
283     //El primer archivo tiene más
284     puts("El primer archivo tiene mas muestras");
285     short *resultadoRe=(short *)malloc(total_muestras1 * sizeof(
      short));
286     short *resultadoIm=(short *)malloc(total_muestras1 * sizeof(
      short));
287   for (i = 0; i < total_muestras1; i++){
288     if(i<total_muestras2){
289       //Tenemos que hacer producto de complejos
290       resultadoRe[i]=((muestrasRe1[i]*muestrasRe2[i])-(
      muestrasIm1[i]*muestrasIm2[i]))/(SHRT_MAX+2);
291       resultadoIm[i]=((muestrasRe1[i]*muestrasIm2[i])+(
      muestrasRe2[i]*muestrasIm1[i]))/(SHRT_MAX+2);//FFT: 5650
```

```
292       }else{
293         resultadoRe[i]=0;
294         resultadoIm[i]=0;
295       }
296     }
297     //Escribo el resultado y cabeceras primer archivo
298     escribirArchivo2Canales(resultadoRe,resultadoIm,0);
299     }
300 }
```

3. Pruebas

Para comprobar el funcionamiento de los programas se usaron los siguientes archivos wav.
Para realizar el filtrado mediante el producto en frecuencia, se uso como entrada para el programa de la transformada discreta de Fourier el siguiente archivo:

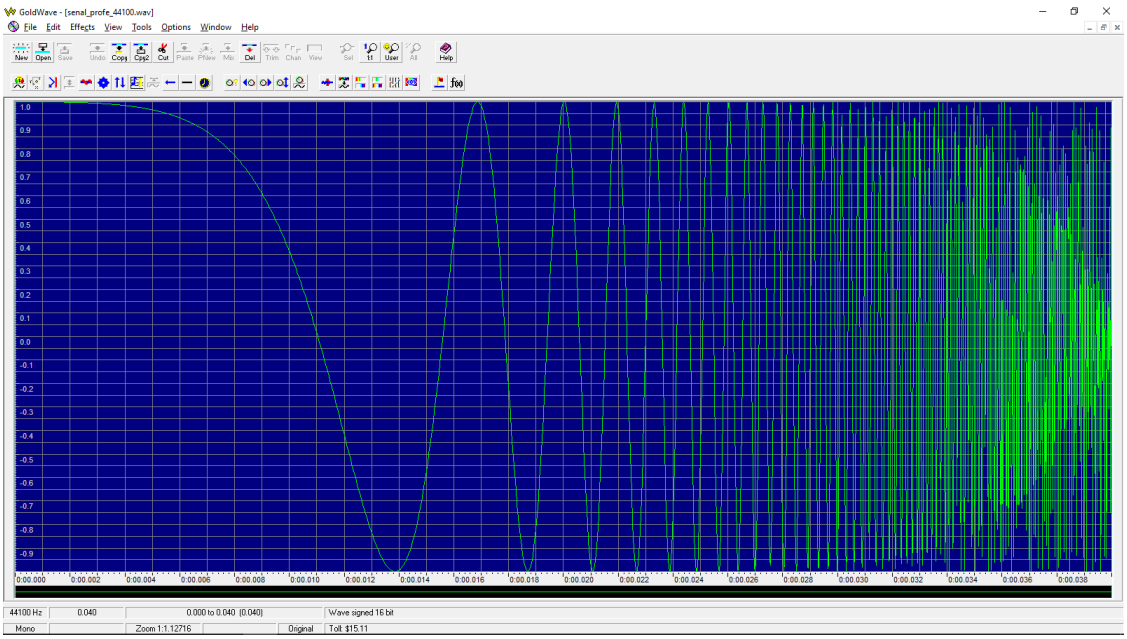

Figura 4: Entrada para TDF

El cual tiene una frecuencia de muestreo de 44100 muestras/s y una duración de .04 segundos.
A diferencia del archivo usado en el reporte 02A-Simulación circuito RC, donde la duración del archivo era de 1 segundo, para esta prueba se disminuyo la duración para que el número de muestras que tuviera que procesar el programa de la TDF no fuera tan grande (con esa duración y frecuencia de muestreo el total de muestras fue de 1764) y no demorara mucho tiempo ejecutándose.
La función usada en el archivo fue: cos(2*pi*t*(exp(log(20)+n/N*6.6))).

La salida obtenida del programa TDF fue la siguiente:

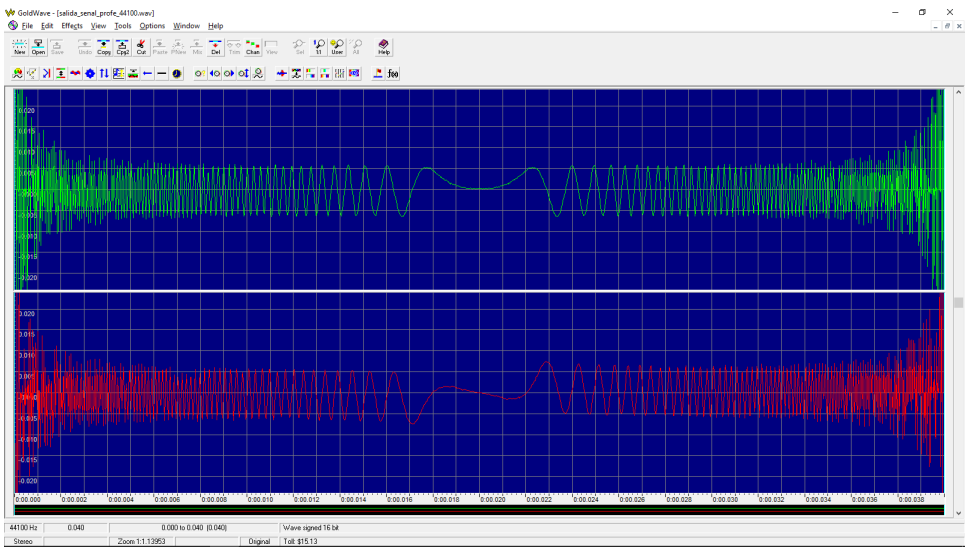

Figura 5: Salida TDF

El filtro para realizar el producto se muestra en la siguiente Figura, cabe señalar que se modelo un filtro ideal con una frecuencia de corte de 1000Hz. Se creo, seleccionando solamente el canal izquierdo del archivo e ingresando la siguiente función:(step(n)-step(n-40))+step(n-(1764-40)).

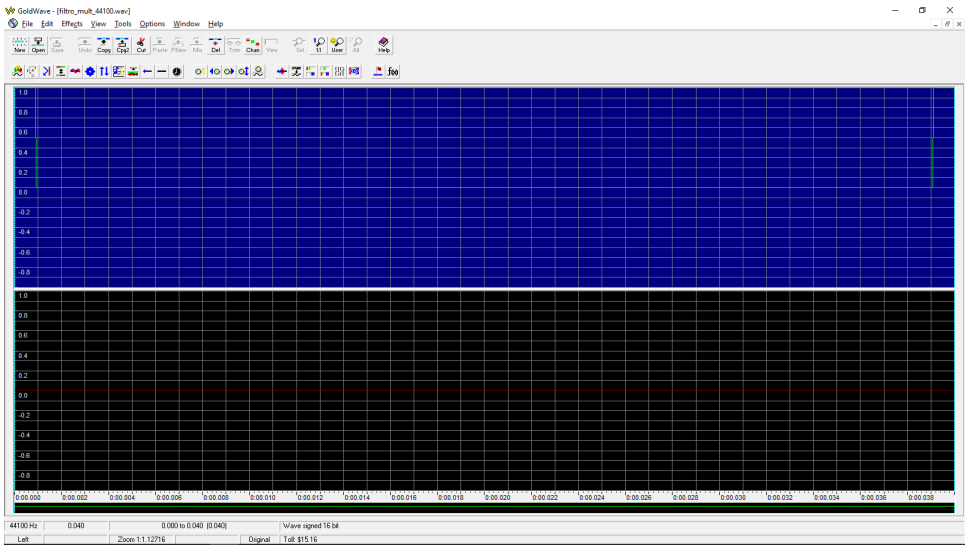

Figura 6: Filtro ideal con una frecuencia de corte de 1000Hz

Posteriormente, la salida obtenida de la TDF y el filtro fueron multiplicados usando el programa producto.exe, y se obtuvo la siguiente salida:

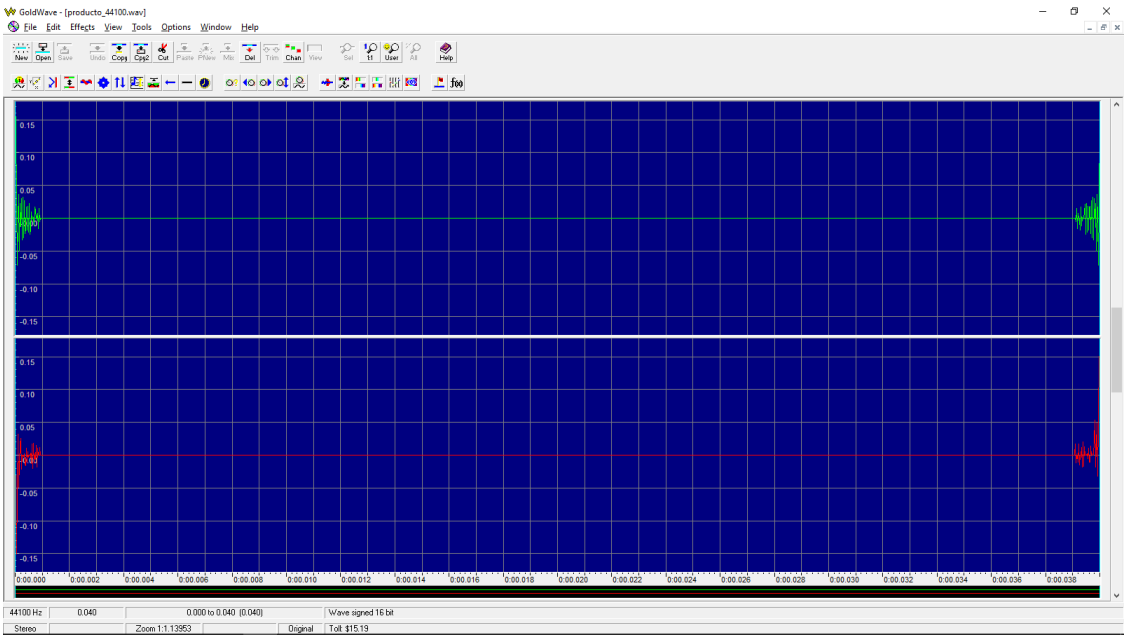

Figura 7: Producto obtenido de multiplicar la salida de la TDF y el filtro ideal

A la salida obtenida por el programa del producto, finalmente fue ingresado al programa que calcula TDF Inversa, y se obtuvo la siguiente salida.

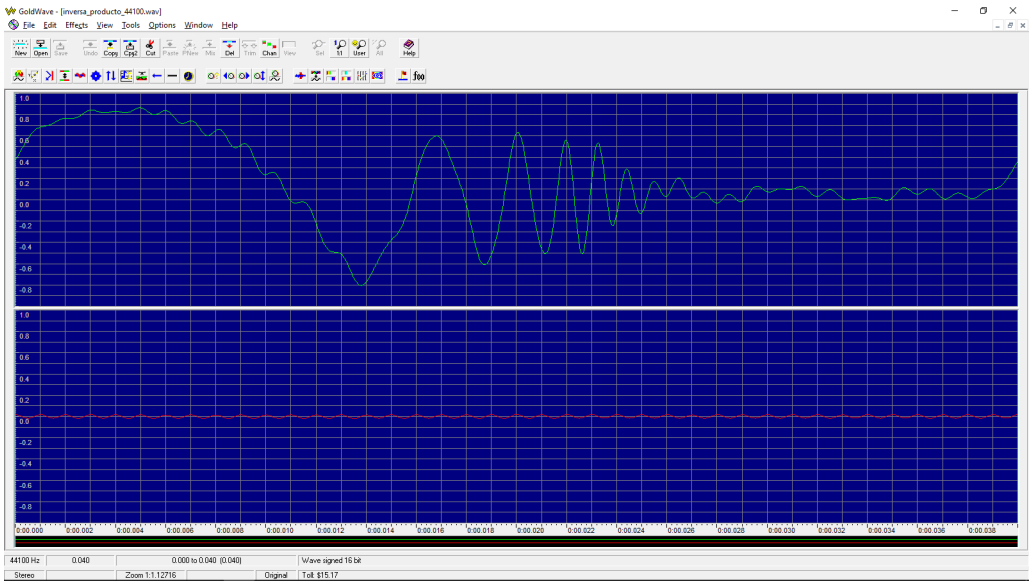

Figura 8: Salida obtenida de la TDFI

Para comprobar que el resultado fuera el correcto, a la señal original de entrada se le filtro usando Goldwave, de lo cual se obtuvo la siguiente salida:

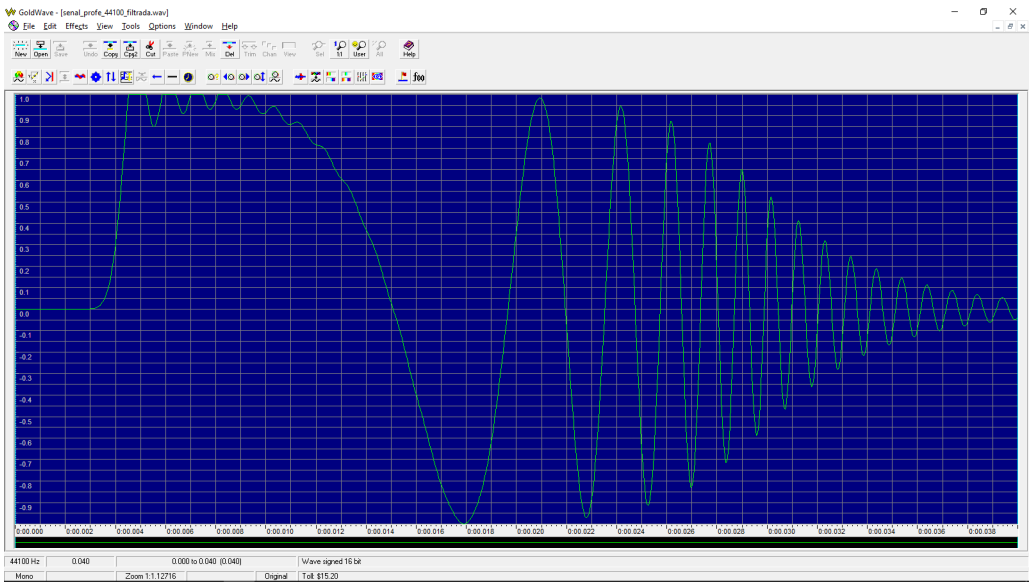

Figura 9: Señal original filtrada usando Goldwave

Como se puede observar, el filtrado realizado mediante frecuencia no es tan preciso como el hecho por Goldwave pero tiene una gran similitud con este mismo. En trabajo posterior, se repite este proceso pero usando la Transformada Rápida de Fourier (FFT).

4. Conclusiones

El análisis de Fourier es una herramienta útil no solo en las matemáticas, si no en muchos otros campos de la ciencia e ingeniería, como lo son el análisis y detección de voz, de instrumentos musicales, y de otros sonidos, teniendo como principal condición, que estos tengan una frecuencia mediante la cual se puedan identificar. Para poderlo aplicar en la computación, es necesario el uso de la Transformada Discreta de Fourier (TDF) la cual nos permite trabajar con señales discretas (se caracterizan por tener información cada cierto tiempo, según sea su frecuencia de muestreo), esto quiere decir que en lugar de tener una señal continua que tiene un valor en cada instante de tiempo (es decir, que contengan información infinita), tenemos una señal que solo contiene información cada cierto tiempo, gracias a lo cual, la señal tiene una cantidad de información finita, lo que permite que esta pueda ser procesada por una computadora (recordando que las computadoras cuentan con una memoria finita, lo cual les impide trabajar con una cantidad ilimitada de información).
El efecto que tiene la TDF sobre la señal de entrada a la que se le aplique, es que primeramente, el espectro en frecuencia de la señal de entrada se volverá periódico. Y segundo, la TDF al igual que la TF transforman una señal en el domino del tiempo al dominio de la frecuencia. Siendo esto muy útil debido a que es posible realizar un analisis en frecuencia a la señal de entrada.
Además, se puedo visualizar la equivalencia entre la convolución en el tiempo y el producto en frecuencia, Ya que cuando se realiza la convolución en el tiempo entre dos señales, uno de los efectos que esto causa, es que ambos espectros de frecuencia se multiplican como producto de complejos. y viceversa, uno de los efectos de realizar el producto en frecuencia, es que es lo mismo que convolucionar en el tiempo.
Como principal defecto, la TDF es un algoritmo con una gran complejidad ($O(N^2)$), por lo cual es un algoritmo muy lento en su ejecución, este tiempo aumenta conforme aumenta el número de muestras para las que se tienen que calcular la TDF.

Referencias

[1] S. W. Smith, *The Scientist and Engineer's Guide to Digital Signal Processing*. California Technical Publishing, 2011.

[2] Sengpielaudio, "Rc filter and cutoff frecuency [online]. disponible en: http://www.sengpielaudio.com/calculator-rcpad.htm."