Lab 2

Redes de Computadoras

Análisis de Fourier

Estudio de la distorsión introducida por el medio físico en la transmisión de una señal digital.

Objetivos

El análisis de Fourier es un concepto básico para entender algunos de los efectos que se producen en la transmisión de datos a través de un medio físico, como por ejemplo distintos tipos de cables. En esta práctica se pretende que el alumno entienda el cálculo de la descomposición de una señal digital mediante series de Fourier, y sea capaz de interpretar los efectos que ejerce el medio sobre esa misma señal. Para ello realizaremos el estudio de esta señal digital en el dominio de la frecuencia mediante el análisis de Fourier.

Descripción Teórica

Durante la transmisión de información a través de un medio físico se producen una serie de fenómenos de atenuación, ruido, etc. que distorsionan la misma. Algunos de estos fenómenos pueden ser estudiados perfectamente desde la observación de la evolución de la señal en el tiempo, sin embargo, otros efectos se estudian mejor desde el análisis de la señal en el domino de la frecuencia. Para poder realizar esta interpretación frecuencial de la señal transmitida vamos a utilizar una herramienta matemática que se denomina análisis de Fourier.

Introducción Teórica

A continuación se explican brevemente los fundamentos teóricos de la descomposición en series de Fourier. Para una mayor información, el alumno debe remitirse al punto 2.1 del capítulo 2 del libro Redes de Computadoras 4ta Ed. de Andrew S. Tanenbaum.

El análisis de Fourier se basa en el hecho, demostrado por Fourier, de que toda señal periódica *s(t)* con periodo *T* o sea *s(t)=s(t+T)* y frecuencia *f=1/T*, puede descomponerse en una serie trigonométrica de la forma siguiente:

$$s(t) = a_0 + \sum_{n=1}^{\infty} a_n \cos(2\pi nft) + b_n sen(2\pi nft) \quad \textbf{(1)}$$

Donde:

$$a_0 = \frac{1}{T}\int_0^T s(t)dt \quad \textbf{(2)}$$

$$a_n = \frac{2}{T}\int_0^T s(t)\cos(2\pi nft)dt \quad \textbf{(3)}$$

$$b_n = \frac{2}{T}\int_0^T s(t)sen(2\pi nft)dt \quad \textbf{(4)}$$

En la fórmula **(1)**, *f* es la inversa del periodo de la señal y la denominamos frecuencia fundamental de la señal *s(t)*. Por otra parte, a_0 es una constante a la que llamaremos componente de continua de *s(t)*, y que se calcula usando **(2)**.

El resto de términos de **(1)** lo forman los componentes de la sumatoria, que se denominan armónicos o componentes armónicas de *s(t)*. Estos armónicos son los que nos permiten el estudio en frecuencia de dicha señal periódica. De esta manera, el armónico n-ésimo de la señal *s(t)* viene determinado por

$$s_n(t) = a_n \cos(2\pi nft) + b_n sen(2\pi nft) \quad \textbf{(5)}$$

Con relación a **(5)**, si *nf=1*, entonces cos(2πt) +sen(2 πt) se corresponde con la suma de un seno y un coseno de periodo 1 (la multiplicación de la variable t por 2π se usa para cambiar el periodo de las sinusoidales de 2π a 1). De esta forma, dentro de **(5)**, los coeficientes *nf* definen la frecuencia de las sinusoidales. Por tanto, decimos que la frecuencia del armónico n-ésimo es f_n= *nf*.

Por último, la amplitud de cada sinusoidal de un armónico *n* viene marcada por los términos a_n y b_n. Estos coeficientes ponderan la contribución de la frecuencia de este armónico en la señal original *s(t)*, y se calculan por medio de las expresiones **(3)** y **(4)**. La potencia o amplitud de un armónico *n* se calcula como:

$$|P_n| = \sqrt{a_n^{\,2} + b_n^{\,2}} \quad \textbf{(6)}$$

La representación gráfica en la que se representa la amplitud de cada armónico de la señal periódica situado en su frecuencia correspondiente se conoce como espectro de frecuencias de *s(t)*. Más tarde veremos ejemplos de este tipo de representación ya discutida en clase de teoría.

Estudio de la influencia del medio físico en la transmisión de una señal

Al utilizar un cable para transmitir una señal, la propia señal que se transmite resulta inevitablemente atenuada. Sin embargo, un cable no atenúa a todos los armónicos de manera uniforme, sino que actúa como filtro paso bajo, de forma que los armónicos de menor frecuencia pasan por él sin problema, pero los de frecuencia más alta resultan atenuados o totalmente filtrados.

De manera teórica, vamos a considerar que un cable deja pasar todos los armónicos hasta una determinada frecuencia de corte f_c (frecuencia de corte), y que a partir de esa frecuencia el resto de armónicos se atenúan completamente. Esta suposición no es del todo realista, ya que la transición de la banda de paso del filtro a la banda de corte se realiza siempre de manera progresiva, por lo cual se define f_c como la frecuencia en donde la señal se atenúa en 3 dB. Un medio no necesariamente tiene una única f_c. Por otra parte, la banda de paso del medio físico (en nuestro caso, de 0 a f_c) va a definir lo que conocemos como el ancho de banda (*bandwidth*, *BW*) de ese medio.

Se puede demostrar que una señal digital está compuesta por infinitos armónicos, y que no existe una frecuencia a partir de la que todos los armónicos sean cero. Esto implica que para transmitir una señal digital con total precisión necesitaríamos un medio físico con un ancho de banda infinito. Desafortunadamente, no existe ningún medio físico con ese comportamiento, lo que de manera práctica implica que al transmitir una señal digital, no todos sus armónicos llegarán al destino.

Al llegar un número limitado de armónicos (*k*) al destino, el receptor recibirá una versión distorsionada de la señal original. Resulta lógico pensar que cuantos más armónicos puedan atravesar el canal, mayor será la calidad de la señal recibida en el destino. Sin

embargo, debemos destacar que los primeros armónicos transmiten la mayor parte de la potencia de una señal, y que es fundamental que éstos atraviesen el canal de comunicación. El resto de armónicos ayudan a definir mejor la señal, pero hay que tener en cuenta que lo importante es que lleguen el número suficiente de armónicos para que el dispositivo receptor sea capaz de reconocer y reconstruir la señal transmitida.

Más adelante, en esta práctica, veremos que el ancho de banda del medio no es el único factor que determina cuantos armónicos pasan el canal y llegan al destino, sino que también influye la velocidad de transmisión de los datos. También veremos gráficamente cómo es la señal que se recibe después de atravesar el cable.

Cálculo de la serie de Fourier de un carácter ASCII (byte) que se repite periódicamente

Para esta práctica vamos a suponer que deseamos transmitir continuamente un mismo byte, y que utilizamos una representación NRZ del mismo, en la que un nivel bajo de la señal indica un valor lógico de 0, y un nivel alto de la señal indica un valor lógico de 1.

Si llamamos *T* al periodo del byte, podemos definir formalmente la señal a transmitir *s(t)* como:

$$s(t) = \left\{ \begin{array}{ll} bit(0) & \forall t \in \left[0 + Tn, \frac{T}{8} + Tn \right] \\ bit(1) & \forall t \in \left[\frac{T}{8} + Tn, \frac{2T}{8} + Tn \right] \\ \vdots & \\ bit(7) & \forall t \in \left[\frac{7T}{8} + Tn, T + Tn \right] \end{array} \right\} \forall n \in K \quad \textbf{(7)}$$

donde *bit(i)* son unas constantes que definen el valor del bit i-ésimo del byte a transmitir.

Si la señal a transmitir no fuera periódica, no podríamos realizar este cálculo mediante series de Fourier, si no que deberíamos de aplicar la transformada de Fourier. El estudio de esta transformada es algo que queda fuera del alcance actual de este curso.

En este punto vamos a realizar el cálculo de los términos a_0, a_n y b_n de **(1)** para la señal definida en **(7)**.

Utilizando la ecuación **(2)** e integrando por partes puede demostrarse que la componente de continua

$$a_0 = \frac{1}{8} \sum_{i=0}^{7} bit(i) \quad \textbf{(8)}$$

Observa que la componente de continua a_0 de la señal *s(t)* es la media matemática de sus bits. Esto es una propiedad importante, derivado de la serie de Fourier, que determina claramente el significado de la componente de continua a_0 de una señal relacionándola con el teorema fundamental del valor medio del cálculo integral, por eso también recibe el nombre de valor medio de la señal.

El cálculo de a_n y b_n se puede realizar de la misma forma. Utilizando las fórmulas **(3)** y **(4)** se tiene como resultado las fórmulas **(9)** y **(10)**. El desarrollo de los resultados obtenidos se deja como ejercicio para el alumno.

Para su cálculo recuerda que $\int \cos(kt)dt = \frac{sen(kt)}{k}$ y que $\int sen(kt)dt = -\frac{\cos(kt)}{k}$.

$$a_n = \frac{1}{n\pi}\sum_{i=0}^{7}\left[bit(i)\left(sen\left(\frac{n\pi}{4}(i+1)\right) - sen\left(\frac{n\pi}{4}i\right)\right)\right] \textbf{ (9)}$$

$$b_n = \frac{1}{n\pi}\sum_{i=0}^{7}\left[bit(i)\left(\cos\left(\frac{n\pi}{4}i\right) - \cos\left(\frac{n\pi}{4}(i+1)\right)\right)\right] \textbf{ (10)}$$

Desarrollo de la práctica

En la primera parte de esta práctica, vamos a observar cómo se implementa en lenguaje de programación C el cálculo de los términos a_0, a_n y b_n de un carácter ASCII, utilizando para ello las ecuaciones de **(8)**, **(9)** y **(10)**. Posteriormente, observaremos la representación gráfica de este carácter, tal y como llega al receptor, y utilizaremos el simulador implementado para realizar una serie de ejercicios.

Material necesario

Para la realización del programa para el cálculo de los armónicos de un carácter ASCII se parte de un fichero llamado Fourier.c. Este programa lo vamos a desarrollar en C estándar, por lo que necesitaremos un compilador de C. En concreto proponemos utilizar el compilador gcc, que incluye cualquier distribución de Unix/Linux.

La ejecución e interacción con el programa a desarrollar se hará en modo consola, por lo que también podría utilizarse cualquier otro compilador de C bajo entorno MS Windows, como MS Visual C++ o Borland C++ Builder o Mingw. En este caso debería de crearse un proyecto del tipo *Win32 Console Application* e incluir el fichero Fourier.c dentro del proyecto.

Para poder visualizar las gráficas que genera nuestro programa, necesitamos la aplicación para el dibujo de gráficas GNUPlot. Es indispensable disponer de una versión igual o superior a la 3.6. Este programa es multiplataforma y se encuentra disponible en casi todas las distribuciones Linux. También se dispone de releases para otros sistemas operativos, como MS Windows.

En el foro tenemos el material necesario para esta práctica. En total se incluye una versión ejecutable completa del programa, tanto para Linux[1] como para Windows, el fichero Fourier.c para desarrollar el simulador y los enlaces a la página de GNUPlot y Bloodshed Dev C++[2].

Ejemplo de ejecución

Antes de implementar el programa, vamos a ver un ejemplo de ejecución de la versión compilada completa que tenemos disponible. Para esto, en primer lugar realiza una copia local del archivo ejecutable en un directorio temporal. Posteriormente abre una consola y ejecuta desde esta consola el programa sin especificar ningún tipo de parámetro. Cuando no se indica ningún tipo de parámetro, el programa utiliza el carácter, la velocidad de transmisión y el ancho de banda que se encuentran definidos por defecto.

En este punto, el programa ha simulado el proceso de transmisión de un carácter, a la velocidad de transmisión indicada, y usando un medio físico de transmisión con un ancho de banda (ideal) caracterizado por su *fc*. Como resultado de la ejecución, obtenemos por un lado un fichero de texto de nombre “armonico.txt”, que contiene la

[1] La cátedra solo ha verficado la ejecución y compilación sobre Windows usando el entorno sugerido.

[2] Entorno completo open software de Compilador C/C++ (Mingw) e IDE para Windows.

frecuencia (en Hz) y potencia de los armónicos que atraviesan el canal, y por otro lado se vuelca por la salida estándar el conjunto de comandos GNUPlot que nos servirá para dibujar la señal transmitida y recibida, y su espectro en frecuencias. El resultado de la anterior ejecución debe de ser el siguiente:

```
# REDES DE COMPUTADORES - FACULTAD DE INFORMATICA DE VALENCIA
# PRACTICA SOBRE ANALISIS DE SEÑALES MEDIANTE SERIES DE FOURIER
reset
set size 1.0, 1.0
set label "- Caracter: 97 'a' (61h)\n\n- Vel. txon: 1200 bps\n --> Duracion de un
bit:\n 0.8333 ms \n --> Periodo de un byte:\n 6.6667 ms \n -> Frec. fundamental:\n
150.00 Hz \n\n- Ancho de banda del medio \n(ideal): 1000.00 Hz\n --> Numero de
armonicos\n pasantes: 6\n\n"at screen 0.7,0.8
left
set multiplotset
nokey
set size 0.7, 0.5
set origin 0.0, 0.5
set yrange [-0.5:1.5]
set ylabel "Amplitud"
set xlabel "Tiempo (seg)"
f=150.000000
# Funcion que define el byte original que se transmite
byte(x)=(x<0.000833)?0:(x<0.001667)?1:(x<0.002500)?1:(x<0.003333)?0:(x<0.004167)?0:(x<
0.005000)?0:(x<0.005833)?0:(x<0.006667)?1:0
# Funcion que define la serie de Fourier como la suma de sus terminoss1(x) =
0.225079*cos(2*pi*1*f*x)+ (0.356927*sin(2*pi*1*f*x))
s2(x) = -0.159155*cos(2*pi*2*f*x)+ (-0.159155*sin(2*pi*2*f*x))
s3(x) = 0.075026*cos(2*pi*3*f*x)+ (-0.331182*sin(2*pi*3*f*x))
s4(x) = -0.000000*cos(2*pi*4*f*x)+ (-0.159155*sin(2*pi*4*f*x))
s5(x) = -0.045016*cos(2*pi*5*f*x)+ (-0.198709*sin(2*pi*5*f*x))
s6(x) = 0.053052*cos(2*pi*6*f*x)+ (-0.053052*sin(2*pi*6*f*x))
s(x)=0.375000+s1(x)+s2(x)+s3(x)+s4(x)+s5(x)+s6(x)
set title "Representacion de Fourier"
plot [0:0.006667] byte(x) with lines 3, s(x) with lines 1
set origin 0.0, 0.0
set arrow 23 from 1000,0 to 1000,0.6 lt 3
set yrange [0:1]
set ylabel "Amplitud"
set xlabel "Frecuencia (Hz)"
set title "Analisis espectral"
plot [0:11251] "armonico.txt" with impulse
set nomultiplot
pause -1 "Pulsa una tecla para finalizar"
```

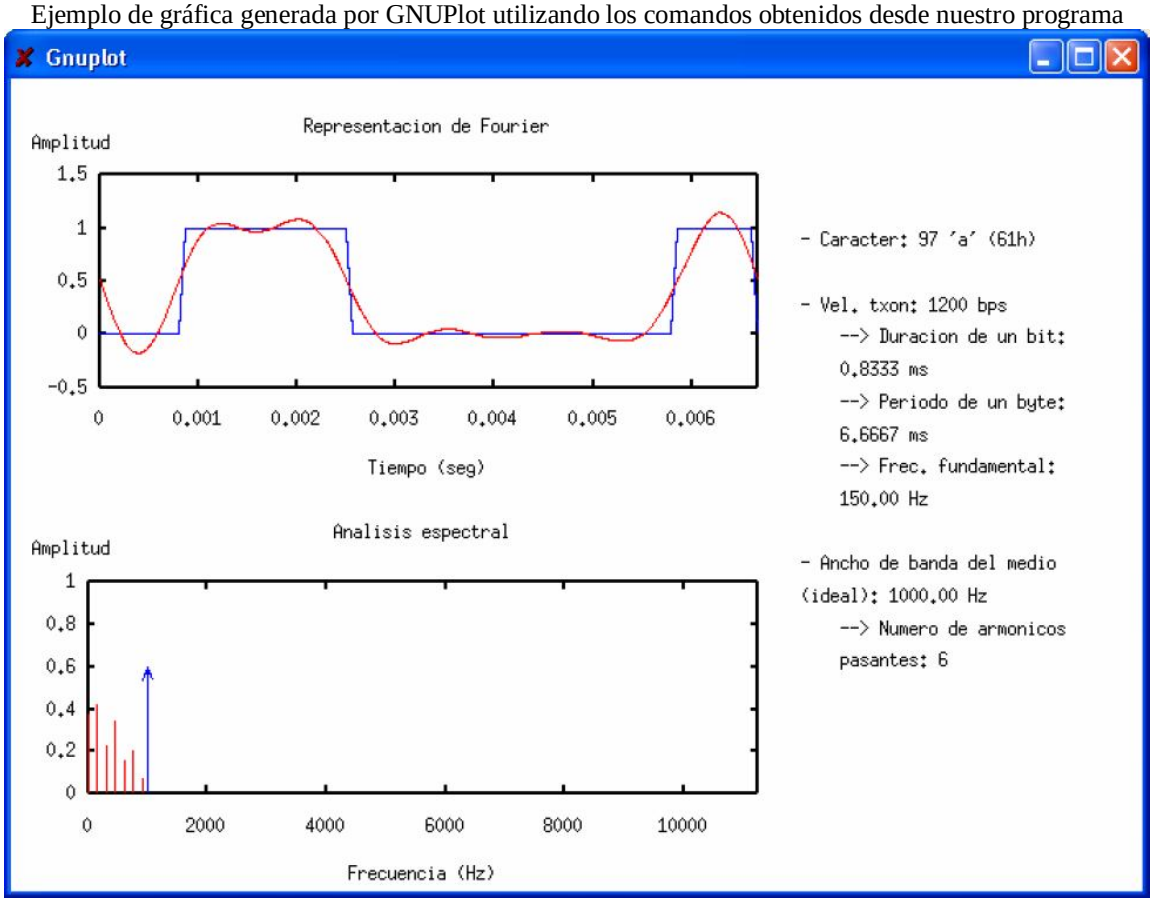
La mayor parte de comandos son del tipo *set*, y sirven para configurar el aspecto de la gráfica (tamaño, posición etc.) y sus etiquetas (rótulos de las gráficas y resto de información). Por otra parte, se define la función *byte(x)*, que describe el carácter original, tal y como lo transmite el emisor (es decir, sin distorsiones). Posteriormente, se define *s(x)* como la suma de los seis armónicos $s_1(x)$… $s_6(x)$ que atraviesan el canal. Para calcular los coeficientes a_0, a_n y b_n de estas funciones, el programa utiliza la llamada a los procedimientos que vamos a implementar en el siguiente punto. Por último, podemos destacar que el primer comando *plot* sirve para dibujar de manera solapada las funciones *byte(x)* y *s(x)* que acabamos de definir, mientras que el segundo plot dibuja el espectro de *s(x)*, utilizando para ello el fichero "armonico.txt". No vamos a estudiar en más profundidad los comandos de GNUPlot, ya que éste no es el objetivo de esta práctica. Para más información se puede consultar el manual que incluyen todas las distribuciones de GNUPlot (comando *help*).

A continuación vamos a generar las gráficas, para ello:

1. Activa el atributo de ejecución del fichero **Fourier** si no está activado. Para ello teclea **chmod +x Fourier**[3].
2. Teclea **./Fourier > comandos.txt** para redirigir la salida estándar sobre un fichero de texto.
3. A continuación pasaremos el fichero de texto con los comandos, al programa GNUPlot, y así generaremos las gráficas. Para ello teclea **gnuplot comandos.txt**[4].

Obtendremos una ventana similar a esta:

Ejemplo de gráfica generada por GNUPlot utilizando los comandos obtenidos desde nuestro programa



En la parte derecha de la ventana, tenemos información calculada a partir de los parámetros introducidos. Por un lado se muestra el carácter transmitido, la velocidad de transmisión y el ancho de banda del medio (*BW*), que son parámetros introducidos por el usuario (en este caso se han usado los parámetros por defecto). Además, a partir de la velocidad de transmisión se obtiene cuál es la duración de un bit, cuál es el periodo del byte que se transmite, y la inversa de este periodo determina la frecuencia fundamental f de la señal.

El último de los valores que se muestra en esta parte de la ventana es el número de armónicos que pasan por el medio. Su cálculo resulta sencillo, ya que sabemos que los armónicos se sitúan siempre a una distancia *f* entre ellos, a partir de 0 hz (es decir, a partir de la componente de continua). Para saber cuántos armónicos pasan por la banda

[3] Solo válido para entorno Linux

[4] Desde Windows basta con arrastrar comandos.txt a la consola de gnuplot abierta.

de 0 a *BW* basta con dividir el *BW* entre *f*, redondeando siempre hacia abajo (nunca puede pasar una fracción de armónico, un armónico pasa entero o no pasa)[5].

En la parte superior de la ventana, se dibuja un gráfico con la señal transmitida (en azul) y la señal tal y como se recibe en el destino (en rojo), después de verse atenuada por el medio. La señal azul se corresponde con la representación gráfica de la codificación NRZ del carácter transmitido, y está definida por la función *byte(x)*, mientras que la señal roja es el resultado de reconstruir la serie de Fourier del carácter, utilizando únicamente los armónicos que pasan por el medio, y viene definida por la función *s(x)*.

Por último en la parte inferior de la ventana tenemos la representación en frecuencias (o espectral) de la serie de Fourier. Cada uno de los armónicos pasantes se representa con una línea vertical, cuya situación en el eje horizontal indican la frecuencia de ese armónico, y cuya amplitud muestra la potencia (por tanto la importancia) de ese armónico. Observa como el ancho de banda del medio viene indicado por una flecha azul vertical, situada sobre su frecuencia de corte.

Implementación del cálculo de los coeficientes a_0, a_n y b_n.

A continuación vamos a implementar el cálculo de los coeficientes a_0, a_n y b_n en el programa Fourier.c. Para ello, edita este fichero y da un vistazo al código que contiene. Observa que las primeras funciones sólo tienen definida la cabecera y falta rellenar el cuerpo de la función. Tendrán el siguiente aspecto:

```
double a(int n, int caracter)
{
// IMPLEMENTAR
}
double b(int n, int caracter)
{
// IMPLEMENTAR
}
double a0(int caracter)
{
// IMPLEMENTAR
}
```

Las dos primeras funciones se corresponden con el cálculo de a_n y b_n. Para implementarlas puedes utilizar las ecuaciones **(9)** y **(10)**, donde la variable *n* de estas ecuaciones es un parámetro que obtenemos de la cabecera de la función, y el acceso a *bit(i)* lo puedes conseguir mediante la orden *bit(caracter,i)*. Para implementar el cálculo de a_0 deberás usar la ecuación **(8)**. En los tres casos recuerda que para devolver el resultado debes utilizar la función return. Una vez implementadas las tres funciones, deberían quedar aproximadamente como el código sugerido en el apéndice A.

Luego compila el programa y ejecútalo tal y como hicimos en el ejemplo de ejecución. Para compilar el programa en Unix utilizando el compilador gcc teclea:

gcc Fourier.c –o MiFourier –lm

Con la opción –o se indica el nombre que debe tener el fichero ejecutable generado, mientras que la opción –lm sirve para enlazar con la librería matemática, y es necesaria para poder utilizar las funciones trigonométricas de C *sin* y *cos*.

[5] Esto no es correcto, pero debido a limitaciones de tiempo lo dejaremos pasar, para entender mejor observe que pasa con los armónicos y la fundamental si utiliza el carácter ASCII 170.

Si estás utilizando otro compilador de C, como MS Visual C++, Borland C++ Builder o Bloodshed Dev C++, el proceso de compilación será diferente. Deberás seleccionar la orden *Build* del menú adecuado.

Después de ejecutar tu simulador, compara el resultado con el que obtuviste anteriormente para asegurarte de que está implementado correctamente.

Simulación de la influencia del medio físico en la señal a transmitir

En este punto vamos a utilizar nuestro simulador para ver gráficamente cómo afecta la atenuación introducida por el cable a la señal transmitida. Para esto, vamos a generar cuatro gráficas distintas que corresponden a la transmisión de la misma señal, con la misma velocidad, pero con medios físicos que tienen distinto ancho de banda. En concreto usaremos medios con ancho de banda igual a 400, 1000, 2000 y 9000 hz.

Ejecuta tu programa con los siguientes parámetros:

./MiFourier –c a –v 1200 –b 400 –a 10000 > bw400.txt

En esta línea, la opción –c sirve para indicar el carácter a transmitir (también podrías haber introducido su código ASCII usando –c 97 en lugar de –c a), la opción –v indica la velocidad a la que se desea transmitir ese byte (en bits por segundo), la opción –b indica cual es el ancho de banda del medio (introducimos la frecuencia de corte en hz), y por último la opción –a es meramente cosmética, y sirve para indicar a GNUPlot cual es el ancho con el que debe dibujar la ventana del espectro, que en este caso será de 0 a 10 Khz.

Como el carácter y la velocidad introducida son las que el programa utiliza por defecto, podemos omitirlas. Para construir las gráficas con otros anchos de banda teclea:

./MiFourier –b 1000 –a 10000 > bw1000.txt

./MiFourier –b 2000 –a 10000 > bw2000.txt

./MiFourier –b 9000 –a 10000 > bw9000.txt

Compara las cuatro gráficas utilizando para ello el programa GNUPlot. Es fácil ver que conforme se aumenta el ancho de banda, más armónicos son capaces de pasar por el medio, y por tanto mejor definición tiene la señal que se recibe en el otro extremo. ¿Crees que sería suficiente un ancho de banda de 400 hz para transmitir este carácter a una velocidad de 1.200 bps? ¿Sería necesario un ancho de banda de 9 Khz o piensas que podría bastar con un ancho de banda de 2 Khz?

Simulación de la influencia de la velocidad de transmisión en la señal

En general, todos tenemos la idea de que al transmitir por un medio físico una señal digital, la velocidad máxima de transmisión está acotada. Hay muchos factores que introducen esta limitación, pero uno de los principales es que el ancho de banda del medio es limitado. Como ya hemos visto, esto provoca que a partir de una determinada frecuencia, el resto de armónicos se atenúen.

Sin embargo, observa que la velocidad de transmisión influye directamente en el periodo del byte, de manera que cuánto más alta sea la velocidad, más corta es la duración de este byte. Además, la frecuencia fundamental (que determina la distancia entre armónicos) viene dada por la inversa del periodo del byte. Por tanto podemos concluir que cuanto mayor sea la velocidad de transmisión, mayor será la distancia entre armónicos, y por tanto el número de armónicos que atraviese el medio será menor.

Dicho de otra forma, cuánto más alta sea la velocidad de transmisión, más deformada llegará la señal al destino, ya que pasan menos armónicos.

A continuación vamos a utilizar nuestro programa para simular la influencia de la velocidad de transmisión en la señal. Para ello usaremos velocidades de 1200, 2400, 4800 y 9600 bps, y un ancho de banda de 6 Khz.

Teclea:

./MiFourier –b 6000 –a 10000 –v 1200 > v1200.txt

./MiFourier –b 6000 –a 10000 –v 2400 > v2400.txt

./MiFourier –b 6000 –a 10000 –v 4800 > v4800.txt

./MiFourier –b 6000 –a 10000 –v 9600 > v9600.txt

Observa el resultado de las cuatro gráficas. En las cuatro gráficas se ha transmitido la misma señal a distinta velocidad, por tanto los armónicos que se generan son los mismos, pero cada vez más distantes. De esta manera, al aumentar la velocidad de transmisión, el número de armónicos que atraviesan el medio es cada vez menor, y la señal se recibe cada vez más distorsionada ¿Te parece aceptable una velocidad de transmisión de 9600 bps para este medio físico?

En general es complicado determinar cuál es la velocidad máxima aceptable en un medio físico, ya que depende de muchos factores y su estudio supera el objetivo de esta práctica. A nivel teórico, podríamos determinar que la velocidad de transmisión adecuada es aquella que permita pasar un número de armónicos suficiente para poder reconstruir la señal original en el destino. Se puede considerar que esto empieza a cumplirse a partir del paso de los primeros seis armónicos.

A continuación utiliza el simulador para transmitir un carácter, usando un medio de ancho de banda de 6 Khz, y a una velocidad de 7600 bps. Posteriormente simula la transmisión del mismo carácter utilizando un medio con un ancho de banda de 45 Khz, y una velocidad de transmisión de 56000 bps. ¿Existe alguna diferencia en la forma de la señal recibida en ambos casos? ¿Por qué crees que es así? ¿Qué crees que resulta más importante a la hora de reconstruir la señal, el ancho de banda, la velocidad de transmisión o la relación entre ambos?

Cuestiones

Sea un canal de transmisión que funciona como un filtro paso bajo ideal, con un ancho de banda de 10 Khz (banda pasante de 0 a 10 Khz), se transmite continuamente una señal digital NRZ correspondiente a un carácter ASCII (cualquiera) de 8 bits, a una velocidad de 1000 bps. ¿Cuántos armónicos atraviesan el canal? Ídem si la velocidad de transmisión es de 2000 bps. Resuelve este problema analíticamente, y después simúlalo.

Utiliza distintos caracteres, descartando el carácter NUL y observa para una determinada velocidad de transmisión y ancho de banda cuál sería el peor y mejor caso y justifica tu respuesta usando los conceptos aprendidos.

Finalmente todo este laboratorio está basado en la premisa que la información enviada es transmitida en banda base usando codificación NRZ no polar. Si se cambiara dicha codificación por Manchester por ejemplo ¿crees que variarían los resultados? Justifica tu respuesta teóricamente.

Ampliaciones opcionales sobre la práctica

Una modificación interesante que se puede realizar al programa es hacer que el simulador contemple un medio que tuviera un comportamiento de filtro pasa-banda (como por ejemplo, las líneas de telefonía de la red telefónica básica, que tienen una banda pasante de 300 hz a 3,3 Khz). El objeto de esta modificación sería estudiar el efecto que tiene el desplazamiento del espectro en las transmisiones en banda base y la importancia de la componente de continua.

Otra ampliación que se propone es adaptar la atenuación de la señal a un caso más realista, en el que los armónicos reciban una atenuación determinada en función de la frecuencia y la longitud del medio. Para realizar esta ampliación, habría que buscar las tablas de atenuación en función de la frecuencia de cables de distinta categoría, e implementar esta atenuación sobre un número de armónicos especificado (por ejemplo, sobre los cien primeros). Si no encuentras información sobre estas tablas, puedes construirlas a partir de las gráficas sobre la atenuación que hay en el capítulo 2 del libro de la materia. También debería tenerse en consideración el factor de atenuación en relación a la longitud que debe recorrer la señal, aunque como el efecto es aproximadamente lineal, solo tiene sentido para conceptualizar la sensibilidad del receptor.

Indudablemente también sería muy interesante implementar un modelo como *Additive White Gaussian Noise* (AWGN) para instrumentar los efectos del ruido sobre el canal, tarea que está fuera del alcance actual de la cátedra, pero que ayudaría al estudiante a visualizar las limitaciones introducidas por el SNR.

Apéndice A

Posible implementación de las funciones solicitadas

```
double a(int n, int caracter)
{
   double suma = 0;
   int i;
   for(i=0;i<8;i++) suma+=bit(character,i)*(sin(n*PI*(i+1)/4.0)-sin(n*PI*i/4.0));
   return suma/(n*PI);
}
double b(int n, int caracter)
{
   double suma = 0;
   int i;
   for(i=0;i<8;i++) suma+=bit(character,i)*(cos(n*PI*i/4.0)-cos(n*PI*(i+1)/4.0));
   return suma/(n*PI);
}
double a0(int caracter)
{
   double suma = 0;
   int i;
   for(i=0;i<8;i++) suma+=bit(character,i);
   return suma/8.0;
}
```