UNIVERSIDAD AUTONOMA METROPOLITANA

Unidad Iztapalapa

Ciencias Básicas e Ingeniería

REPORTE DE PROYECTO TERMINAL

Que presentan para obtener el titulo de
Ingeniería en Electrónica:

Ivan Hernández García 70270945
&
Laura Angélica Molina Salas 93321209

Asesor:

Ing. Jesús ~~Barrios Romano~~

"FILTROS DIGITALES"

México, D.F. Marzo, 2001

" Todo hombre debe decidir, aunque sea una vez en la vida, si se lanza a triunfar arriesgándolo todo o si se sienta a contemplar el paso de los triunfadores "

Dedicatoria

Iván Hernández García.

" Quiero dedicar este trabajo a dos personas que son muy importantes para mí: A mis padres, porque de ellos he tenido sabios consejos, paciencia y comprensión en toda mi vida y que sin su ayuda y apoyo no hubiera concluido mis estudios."

A mis hermanos:

"A ellos les agradezco porque siempre que los he necesitado me han ayudado para salir adelante."

A mis sobrinos:

"A ellos agradezco el cariño que me brindan."

A mi cuñado:

"A el agradezco su amistad y la paciencia que siempre me brindo, yo se, que ya es muy tarde y el ya no me escucha, pero quiero decirle que para mi fue como un hermano, descansa en paz hermano."

"Quiero hacer patente mi más profundo agradecimiento a todos los profesores y amigos que con su conocimiento y consejos me ayudaron a formarme profesionalmente."

DEDICATORIA

El presente trabajo lo dedico a la persona que me ha dado todo su apoyo y amor durante toda la trayectoria de mi vida

A ti mamá ,
con todo mi cariño.

A mis familiares,
por el apoyo y comprensión que siempre me han brindado y por sus palabras de aliento siempre oportunas.

A mis amigos y compañeros con los que comparti momentos de estudio y cuya amistad conservo.

Y un agradecimiento especial al profesor J. Barrios por el apoyo en la realización de este proyecto.

Laura **A**ngélica **M**olina **S**alas

INDICE

CAPITULO IV **Filtros Dígitales con respuesta al impulso infinito RII**

PREFACIO

El presente trabajo tiene como objetivo principal el presentar los conceptos básicos sobre filtros digitales, así como también el analizar y estudiar diversos algoritmos para el procesamiento de señales. Para ello primeramente se presenta en el capitulo I los conceptos sobre filtros analógicos y la construcción de respuestas de frecuencia en base a diferentes autores como lo son Butterworh, Chebyshev y Bessel, con el fin de conocer las ventajas y desventajas de cada uno de ellos, y así posteriormente comparar estas mismas, con las que se tienen al estudiar los filtros digitales.

Como ya es sabido, para realizar el procesamiento digital de señales se requiere hacer uso de la teoría de las transformadas y de las matemáticas discretas, para esto contamos con la Transformada discreta de Fourier, la cual se expone en el capitulo II de este trabajo. En este cápitulo se presentan los algoritmos que permiten calcular la Transformada discreta de Fourier, así como también , la transformada rápida .de Fourier.

Posteriormente en el capitulo III se presentan los conceptos sobre filtros digitales, espeficamente con respuesta al impulso finito, RIF. De igual forma se mencionan algunos métodos para el diseño de estos filtros.

Finalmente en el capitulo IV se contempla el diseño de filtros digitales con respuesta al impulso infinito, RII.

Cabe mencionar que al final de cada capitulo se presentan los códigos en lenguaje C que ejemplifican el diseño de filtros analógicos, la aplicación de las transformadas y el diseño de filtros digitales, según el caso. Además, en los anexos se listan, tanto el programa principal, como las funciones auxiliares necesarias para la ejecución de todos los algoritmos presentados en este trabajo.

INTRODUCCION

I. Fundamentos de Procesamiento Digital de Señales

El procesamiento digital de señales (DSP, Digital Signal Processing) esta basado en el hecho de que una señal analógica puede ser digitalizada, e introducida a una computadora (digital) de propósito general o a una de propósito específico. De esta manera es posible realizar toda clase de operaciones matemáticas sobre la secuencia de muestras digitales, como son: suma, resta y multiplicación.

II. Digitalización

La digitalización es el proceso de convertir una señal analógica, como puede ser voltaje o corriente variantes en tiempo, en una secuencia de valores digitales. Por supuesto, habrá que cuidar que la señal se muestree y cuantice; para ello existen tres tipos básicos de muestreo, a saber: ideal, instantáneo, y natural, como se muestra en la figura I.

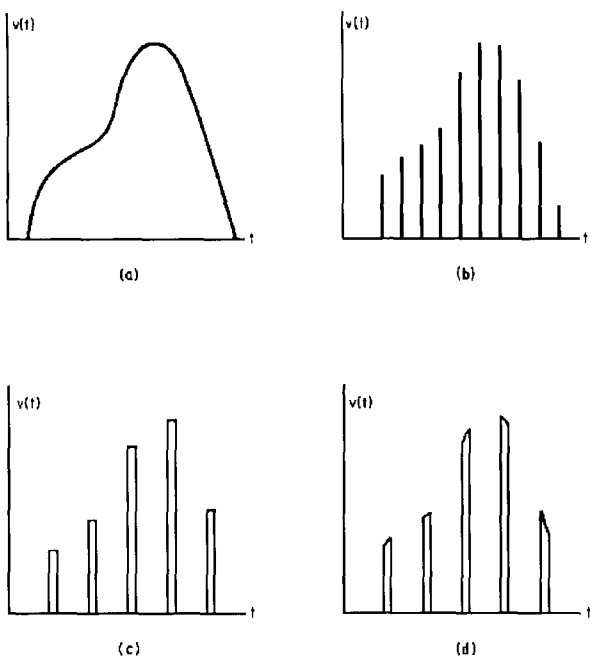

Figura I. Señal analógica (a) tres tipos de muestreo: (b) ideal (c) instantáneo, y (d) natural

En esta figura se observa el proceso de muestreo de una señal que esta definida en un intervalo de tiempo continúo, la cuál toma valores sólo en instantes de tiempo discreto (muestreo ideal). Para el caso de muestreo instantáneo, y natural, el muestreo presenta subintervalos de tiempo $d(t)$ para cada muestra. Como se observa en las figuras c y d, respectivamente. Cada uno de los tres tipos básicos de muestreo se pueden presentar en diferentes lugares dentro de un sistema DSP.

Como ejemplo de un muestreo instantáneo podemos mencionar un convertidor digital-analógico (DAC), cuya salida supone una señal muestreada instantáneamente; por otra parte, los datos muestreados pueden ser interpretados como las amplitudes para una secuencia de muestras ideales. Por último, un ejemplo de muestreo natural se encuentra cuando se realiza el análisis de multicanalización analógica. En estas tres formas de muestreo los valores muestreados pueden asumir un valor apropiado de los posibles valores de una señal analógica.

La cuantización es la parte de la digitalización que está relacionada con la conversión de las amplitudes de una señal analógica a valores que pueden ser representados por números binarios. Una cuantización se muestra en la figura II.

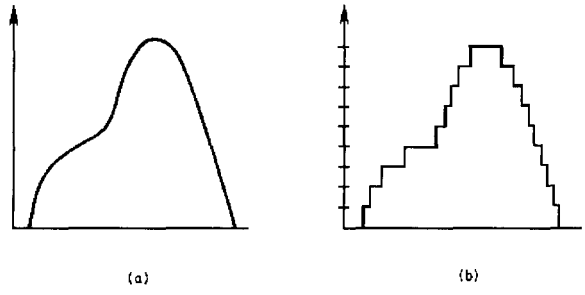

Figura II. (a) Señal analógica y (b) la señal cuantizada correspondiente

En el proceso de muestreo y cuantización se introducen cambios significativos en el espectro de una señal digitalizada, cambios que dependerán de la precisión en la operación de cuantización y del modelo de muestreo.

III. Muestreo ideal

El muestreo ideal de una señal comprende una secuencia de impulsos separados uniformemente. La medida de cada impulso equivale a la amplitud de la señal analógica que corresponde al instante de tiempo en que se toma la muestra, como se aprecia en la figura III.

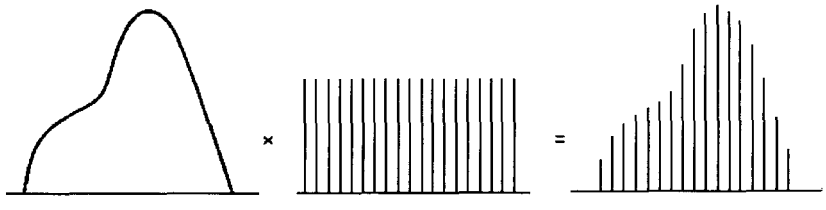

Figura III. Muestreo ideal

Podemos asumir que una señal muestreada es el resultado de la multiplicación de una señal analógica $x(t)$ por un tren de impulsos unitario.

$$x_s(\cdot) = x(t) \sum_{n=-\infty}^{\infty} \delta(t - nT) \qquad \text{(III.1)}$$

El espectro de la señal muestreada se puede obtener convolucionando el espectro de la señal analógica con el espectro del tren de impulsos, como se muestra en la figura IV:

$$\mathrm{F}\left[x(t) \sum_{n=-\infty}^{\infty} \delta(t - nT) \right] = X(f) * \left[f_s \sum_{n=-\infty}^{\infty} \delta(f - mf_s) \right] \qquad \text{(III.2)}$$

La convolución produce copias del espectro original, repitiendo a éste periódicamente. El espacio entre las imágenes se denota como f_s, la cual representa la frecuencia de muestreo, y el espacio entre los bordes de replicas contiguas se denota como $f_S - 2f_H$. Esto garantiza que se cumpla el teorema de muestreo, pudiéndose con esto recuperar la señal original mediante un proceso de filtrado (filtro pasa bajas) que removerá las imágenes introducidas por el muestreo.

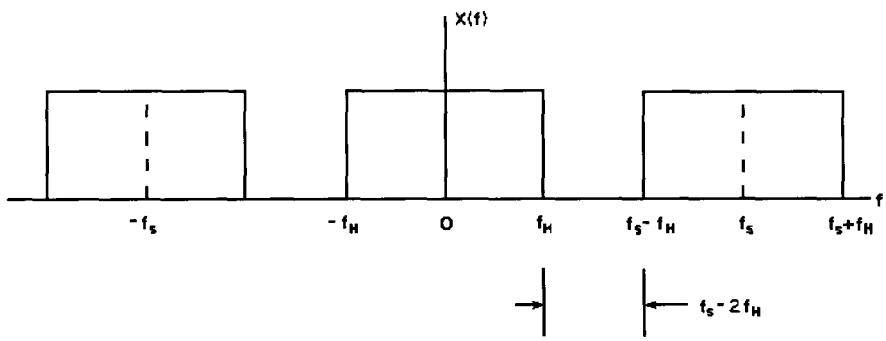

Figura IV. Espectro de una señal muestreada idealmente

IV. Selección de velocidad de muestreo

Si f_s es menor que $2\ f_H$ las imágenes se traslaparan y ocurrirá el fenómeno llamado aliasing; lo cual impide que se pueda recuperar la señal original. Sin embargo, si f_s es igual a $2\ f_H$ (velocidad de Nyquist), es posible la recuperación de la señal. Cuando la velocidad es mayor a $2\ f_H$, la señal presenta un espectro de energía igual al de la señal muestreada.

Al diseñar un sistema de procesamiento de señales, raramente se tiene información relevante de la ocupación exacta del espectro del ruido. Consecuentemente, cuando se diseña un sistema, en la práctica se selecciona el valor de f_H, de la señal deseada, y ésta será filtrada por un pasabajas previamente. A este tipo de filtros se les llama filtros antialias. La selección de velocidad de muestreo y el filtro se diseñan de tal manera que la atenuación que proveen sea de $40dB$ o más para todas las frecuencias que están por encima de $f_s/2$. El espectro de una señal con muestreo ideal se muestra en la figura V, dónde se observa que ocurre algo de aliasing; sin embargo, los componentes de aliasing son

suprimidos a menos de $40dB$ por debajo de los requerimientos deseados. No obstante, existe otra manera para eliminar el aliasing, siendo ésta la más común y consiste en seleccionar el tipo de filtro (Butterworth, Chebyshev, Bessel y Caver) y orden para poder obtener una banda de rechazo para la atenuación que preserve las características fundamentales de la señal muestreada.

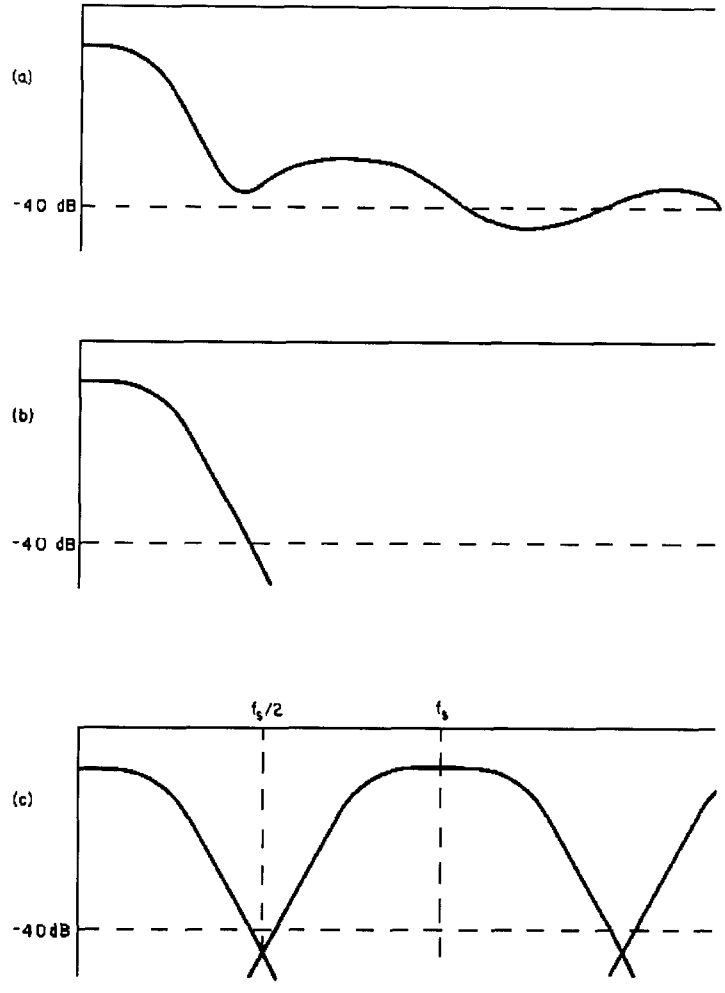

Figura V. Espectro de un a señal idealmente muestreada: (a) espectro de una señal analógica, (b) espectro de la señal filtrada, y (c) espectro de la señal muestreada.

V. Muestreo Instantáneo

En el muestreo instantáneo cada muestra tiene un ancho distinto de cero y un tope plano, como se muestra en la figura VI.

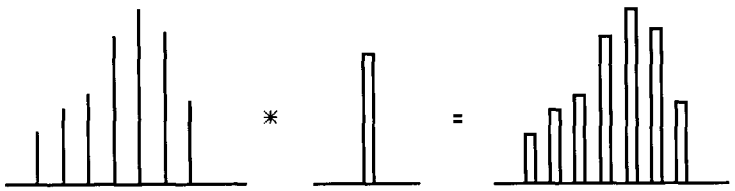

Figura VI. Muestreo instantáneo.

La señal así obtenida se puede considerar como el resultado de convolucionar un pulso de muestreo $p(t)$ con las muestras ideales de la señal analógica, de este modo:

$$x_s(\cdot) = p(t) * \left[x(t) \sum_{n=-\infty}^{\infty} \delta(t - nT) \right] \qquad \text{(V.1)}$$

Donde $p(t)$ es un pulso rectangular de muestreo y $x(t)$ es la señal analógica.

El espectro de las muestras instantáneas se obtiene multiplicando el espectro del pulso de muestreo con el espectro de la señal muestreada idealmente.

$$\mathrm{F}\left\{ p(t) * \left[x(t) \sum_{n=-\infty}^{\infty} \delta(t - nT) \right] \right\} = P(f) \cdot \left\{ X(f) * \left[f_s \sum_{m=-\infty}^{\infty} \delta(f - mf_s) \right] \right\} \qquad \text{(V.2)}$$

La diferencia entre el muestreo ideal y el instantáneo está en la distorsión de amplitud, introducida por el espectro muestreado, como se observa en la figura VII. A esta distorsión se le conoce como el efecto de abertura.

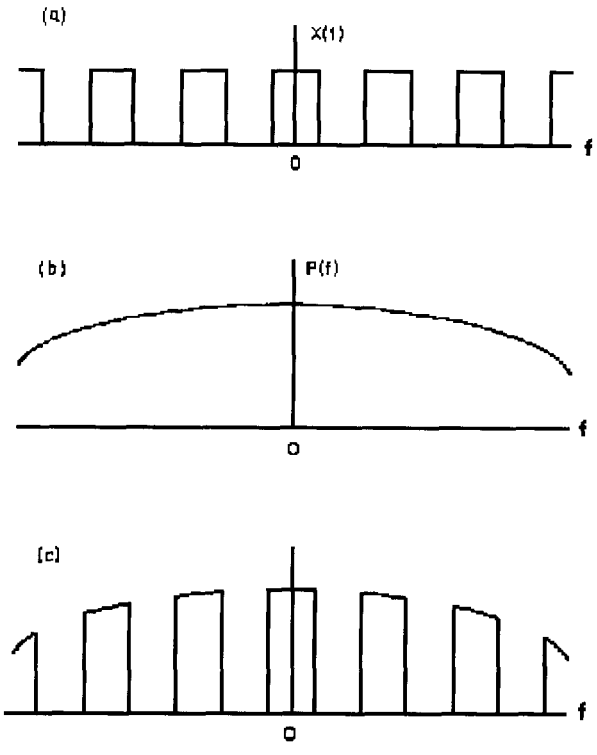

Figura VII. Espectro de una señal muestreada instantáneamente es igual a el espectro (a) de una señal muestreada idealmente multiplicada por el espectro (b) de una muestra de pulsos.

VI Muestreo Natural

En el muestreo natural, la amplitud de cada muestra sigue a la forma de la señal analógica durante toda la duración del muestreo, como se muestra en la figura VIII.

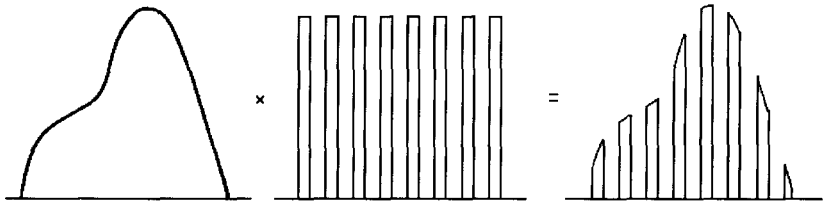

Figura VIII. Muestreo natural.

La señal muestreada naturalmente se obtiene multiplicando la señal analógica por un tren rectangular de pulsos periódicos.

$$x_s(\cdot) = x(t) \cdot \left\{ p(t) * \left[\sum_{n=-\infty}^{\infty} \delta(t - nT) \right] \right\} \qquad \text{(VI.1)}$$

Para obtener el espectro de la señal muestreada basta convolucionar el espectro de la señal analógica con el espectro del tren de pulsos.

$$\mathrm{F}\left[x_s(\cdot)\right] = X(f) * \left[P(f) f_s \sum_{m=-\infty}^{\infty} \delta(f - m f_s) \right] \qquad \text{(VI.2)}$$

El resultado de esta operación se muestra en la figura IX. Este espectro es similar al de una señal con muestreo instantáneo, y se puede observar que todas las frecuencias del espectro de la señal, $X(f)$, son atenuadas por el espectro del tren de pulsos.

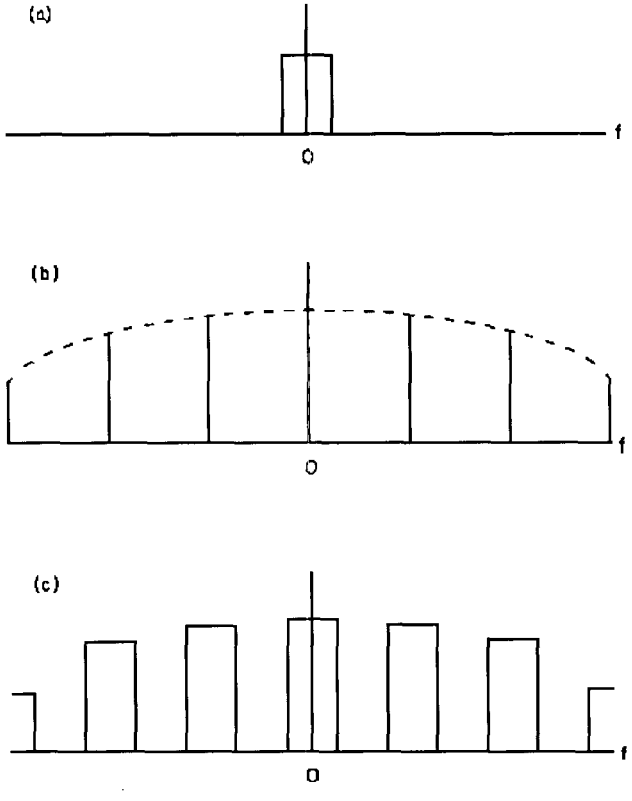

Figura IX. Espectro (c) de una señal muestreada naturalmente es igual al espectro (a) de la señal analógica multiplicada por el espectro (b) de el tren de pulsos muestreados.

CAPITULO I

FILTROS ANALOGICOS

FILTROS ANALOGICOS

En el mundo del procesamiento analógico de las señales, existen múltiples formas de construcción de respuestas de frecuencia interesantes, que pueden representar alguna ventaja; tal es el caso de las conocidas funciones Butterworth, Chebyshev, Bessel o Elípticas.

1.1 FILTROS BUTTERWORTH

Los filtros Butterworth pasabajas se diseñan con el propósito de ofrecer una respuesta lo más plana posible en las frecuencias bajas, respecto a una frecuencia de referencia (usualmente la frecuencia de corte), y al mismo tiempo con un comportamiento monótonamente decreciente conforme aumenta la frecuencia.

1.1.1 FUNCION DE TRANSFERENCIA

Los filtros Butterworth se rigen por una función básica expresada por:

$$H(s) = \frac{1}{\prod_{i=1}^{n}(s - s_i)} = \frac{1}{(s - s_1)(s - s_2)\cdots(s - s_n)} \qquad (1.1)$$

donde $s_i = e^{j\pi[(2i+n-1)/2n]} = \cos\left(\pi\frac{2i+n-1}{2n}\right) + j\,\mathrm{sen}\left(\pi\frac{2i+n-1}{2n}\right)$

Para la cual n representa el orden del filtro, mismo que producirá una mejor o peor respuesta dependiendo de su valor. Es el caso, por ejemplo, que para la función

$$H(s) = \frac{1}{(s - s_1)(s - s_2)(s - s_3)} \qquad (1.2)$$

donde los valores de s_1, s_2 y s_3 representan los polos de la función, ubicados en:

$$\begin{aligned} s_1 &= \cos\left(\frac{2\pi}{3}\right) + j\,\mathrm{sen}\left(\frac{2\pi}{3}\right) = -0.5 + 0.866j \\ s_2 &= e^{jn} = \cos(\pi) + j\,\mathrm{sen}(\pi) = -1 \\ s_3 &= \cos\left(\frac{4\pi}{3}\right) + j\,\mathrm{sen}\left(\frac{4\pi}{3}\right) = -0.5 - 0.866j \end{aligned} \qquad (1.3)$$

De manera que la ecuación 1.4 representa la función de transferencia Butterworth de tercer orden.

$$\begin{aligned} H(s) &= \frac{1}{(s + 0.5 - 0.866j)(s + 1)(s + 0.5 + 0.866j)} \\ &= \frac{1}{s^3 + 2s^2 + 2s + 1} \end{aligned} \qquad (1.4)$$

De igual modo, es posible construir funciones de orden superior; sin embargo, dado lo conocido del procedimiento, lo que se hace es referirse a las tablas de polinomios que existen para la solución del filtro Butterworth, como la tabla 1.1.

n	Polos
2	-0.707107 ± 0.707107j
3	-1.0
	-1.0 ± 0.866025j
4	-0.382683 ± 0.923880j
	-0.923880 ± 0.382683j
5	-1.0
	-0.809017 ± 0.587785j
	-0.309017 ± 0.951057j
6	-0.258819 ± 0.965926j
	-0.707107 ± 0.707107j
	-0.965926 ± 0.258819j
7	-1.0
	-0.900969 ± 0.433884j
	-0.623490 ± 0.781831j
	-0.222521 ± 0.974918j
8	-0.195090 ± 0.980785j
	-0.555570 ± 0.831470j
	-0.831470 ± 0.555570j
	-0.980785 ± 0.195090j

Tabla 1.1 Polos de filtros Butterworth pasabajas

1.1.2 RESPUESTA EN FRECUENCIA

La respuesta en frecuencia de un filtro Butterworth es como se muestra en las figuras 1.1, 1.2 y 1.3.

Las cuales muestran la respuesta en magnitud en la banda de paso, la magnitud en la banda de rechazo y fase respectivamente; estas gráficas están normalizadas a una frecuencia de corte de $1Hz$, y para desnormalizarlas sólo se requerirá multiplicar por la frecuencia de corte deseada.

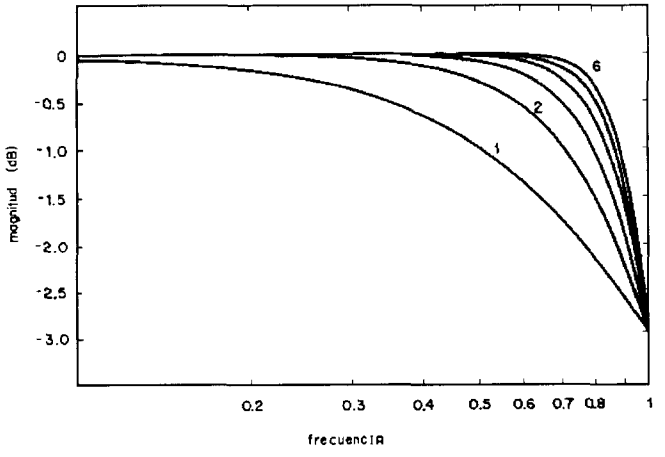

Figura 1.1 Respuesta en Amplitud de un filtro Butterworth pasabajas de orden 1 hasta orden 6.

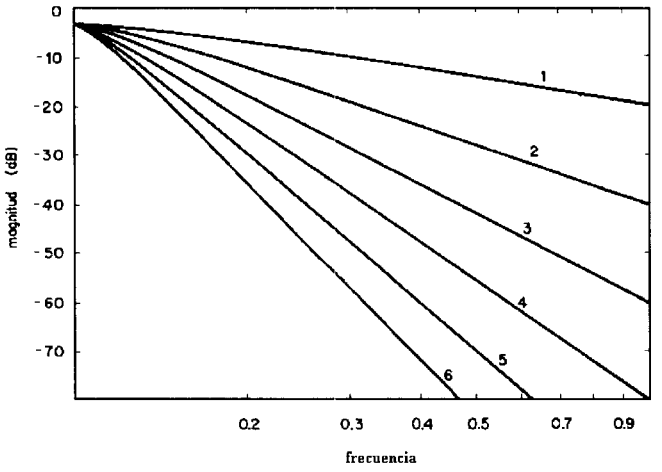

Figura 1.2 Respuesta en amplitud en la banda de rechazo para un filtro Butterworth

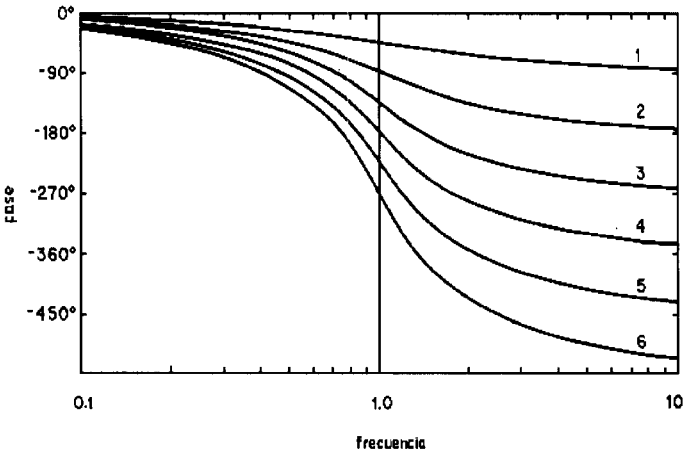

Figura 1.3 Respuesta en fase de un filtro pasabajas Butterworth

Si se desea obtener la respuesta en magnitud y fase a $800Hz$, de un filtro Butterworth pasa bajas de sexto orden $(n=6)$, con una frecuencia de corte de $400Hz$ $(f_c=400)$, lo que se requiere es desnormalizar (multiplicando por 400) las frecuencias normalizadas 1 y $2Hz$, de modo que las figuras 1.2 y 1.3 nos lleven a las figuras 1.4. y 1.5, donde se puede observar que a la frecuencia de $800Hz$ la magnitud es aproximadamente $-36dB$ y la fase es -425^0, respectivamente.

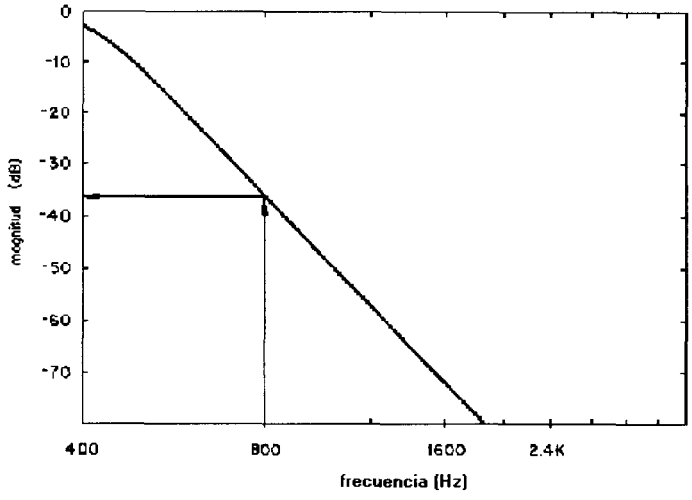

Figura 1.4 Respuesta de amplitud desnormalizada

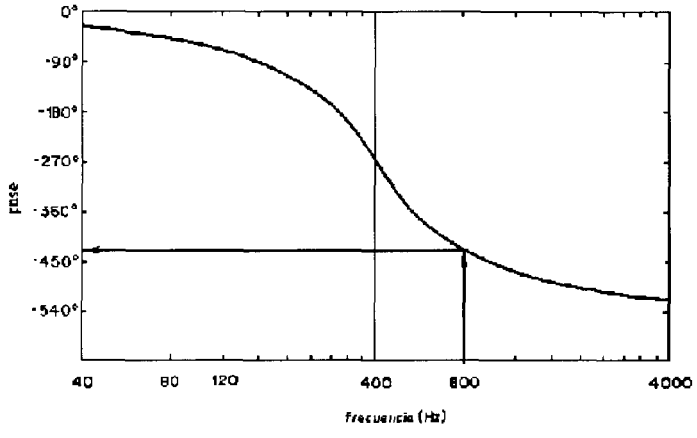

Figura 1.5 Respuesta en fase desnormalizada

A continuación se presenta un corrida del programa de filtros analógicos Butterworth de respuesta en frecuencia, de cuarto orden , n = 4 , y su frecuencia normalizada es de 0.8 Hz. Cuyos valores se pueden comprobar con la tabla 1.1 y las gráficas 1.1 y 1.3.

```
                    FILTROS BUTTERWORTH

Resultados:

  Magnitud = -0.673581(dB)   Fase = -135.859194°


        polo 1= -0.382683 + 0.923880 j
        polo 2= -0.923880 + 0.382683 j
        polo 3= -0.923880 + -0.382683 j
        polo 4= -0.382683 + -0.923880 j




        ¿Desea hacer otro calculo <S/N> ?: _
```

1.1.3 RESPUESTA IMPULSIVA

Para obtener la respuesta impulsiva en un filtro Butterworth de orden n, es necesario calcular la transformada inversa de Laplace de la función de transferencia. Aplicando la expansión de Heaviside a la ecuación 1.1. se obtiene:

$$h(t) = \mathrm{L}^{-1}[H(s)] = \sum_{r=1}^{n} K_r e^{s_r t} \qquad (1.5)$$

donde $K_r = \left.\dfrac{(s-s_r)}{(s-s_1)(s-s_2)\cdots(s-s_r)}\right|_{s=s_r}$

donde K_r y s_r son números complejos. Cuando n es un número par, la ecuación 1.5 se puede expresar como:

$$h(t) = \sum_{r=1}^{n/2}\left[2\,\mathrm{Re}(K_r)e^{\sigma_r t}\cos(\omega_r t) - 2\,\mathrm{Im}(K_r)e^{\sigma_r t}\,\mathrm{sen}(\omega_r t)\right] \qquad (1.6)$$

con $s_r = \sigma_r + j\omega_r$ para $r = 1,2,....,{}^{n}/_{2}$

Cuando el orden n es impar, la ecuación 1.5 adopta la forma siguiente:

$$h(t) = Ke^{-t} + \sum_{r=1}^{(n-1)/2}\left[2\,\mathrm{Re}(K_r)e^{\sigma_r t}\cos(\omega_r t) - 2\,\mathrm{Im}(K_r)e^{\sigma_r t}\,\mathrm{sen}(\omega_r t)\right] \qquad (1.7)$$

Con las raíces s_r para $r = 1,2,....,{}^{(n-1)}/_{2}$.

La respuesta impulsiva de un filtro Butterworth pasa bajas se muestra en las figuras 1.6 y 1.7, y son respuestas normalizadas a una frecuencia de corte de $1\,rad/s$. Para desnormalizar esta respuesta, tan solo habrá que dividir el tiempo por la frecuencia de corte deseada $(\omega_c = 2\pi f_c)$. Por ejemplo, para determinar la magnitud en el tiempo $1.6ms$, después de haberse aplicado un pulso unitario a la entrada de un filtro Butterworth de quinto orden ($n = 5$), con una frecuencia de corte de $250Hz$ $(f_c = 250Hz)$, se requerirá desnormalizar la respuesta para $n = 5$ en la figura 1.7, lo cual se hace dividiendo el tiempo por $2\pi f_c = 500\pi$ obteniéndose una escala en milisegundos donde $1.6ms$ reales corresponden a $2.51s$

normalizados, en tanto que la magnitud (normalizada) de 0.24 corresponde a una magnitud efectiva de $0.24(500\pi) = 378$, aproximadamente, como se muestra en la figura 1.8

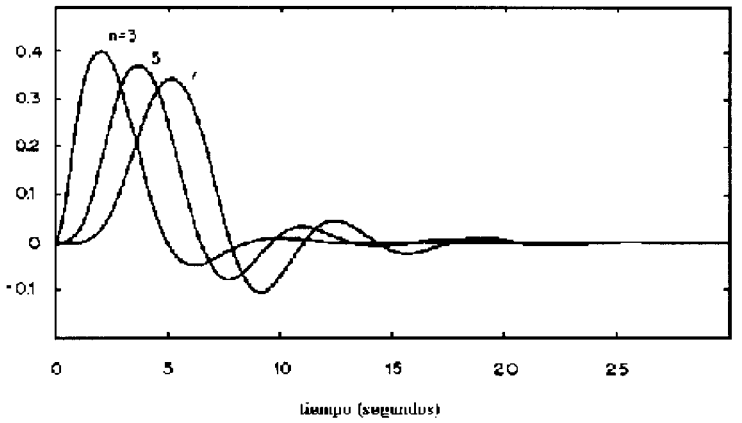

Figura 1.6 Respuesta Impulsiva para filtros Butterworth de orden impar

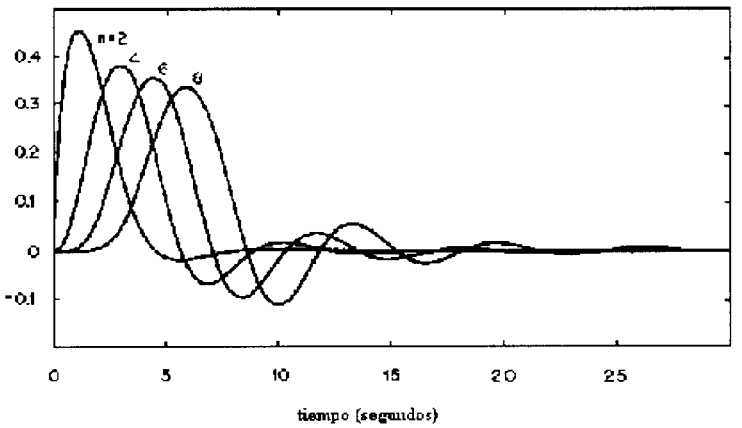

Figura 1.7 Respuesta Impulsiva para filtros Butterworth de orden par

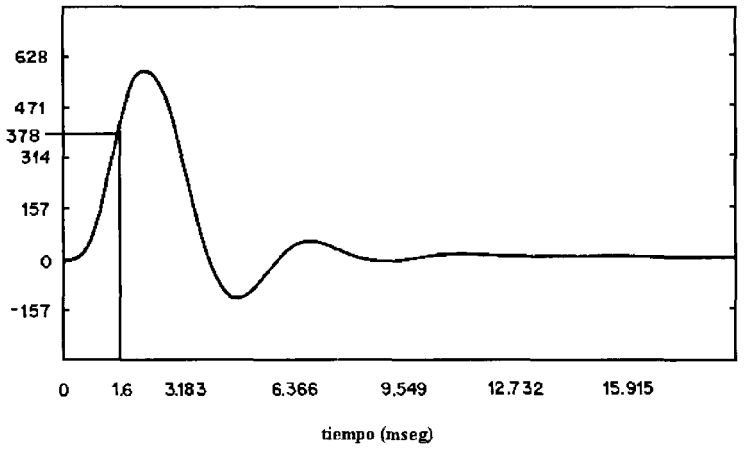

Figura 1.8 Respuesta Impulsiva Desnormalizada

A continuación se presenta una corrida del programa para el diseño de filtros analógicos Butterworth con respuesta impulsiva, para un filtro de quinto orden, n=5, con 10 puntos (de 1 a 10 seg.) cuya separación entre ellos es de 1 seg. Con estos resultados obtenemos los valores de la gráfica 1.6. De igual forma este programa puede obtener los resultados de la gráfica 1.7 que representa la respuesta impulsiva para un filtro de orden par.

```
RESPUESTA  IMPULSIVA  BUTTERWORH

      t          h[t]

      0       0.000000
      1       0.021010
      2       0.155300
      3       0.323772
      4       0.357025
      5       0.227106
      6       0.046235
      7      -0.066826
      8      -0.078098
      9      -0.030681
     10       0.015205


¿Desea hacer otro calculo <S/N> ?: _
```

1.1.4 RESPUESTA AL ESCALON

La respuesta al escalón se obtiene integrando la respuesta impulsiva. Las figuras 1.9 y 1.10 muestran la respuesta al escalón para un filtro Butterworth pasa bajas para orden par y para orden impar, respectivamente. Estas respuestas son normalizadas a una frecuencia de corte de $1\,rad/s$, para desnormalizar la respuesta, se divide el tiempo por la frecuencia de corte deseada $(\omega_c = 2\pi f_c)$. Por ejemplo, para determinar el tiempo de la respuesta al escalón de un filtro Butterworth de tercer orden $(n = 3)$ con una frecuencia de corte de $4kHz$, al primer punto equiparable a su valor final (cuando se ha alcanzado la estabilidad), lo que habrá que hacer es dividir por $2\pi f_c = 8000\pi$, lo cual deja una escala real en microsegundos como se puede apreciar en la figura 1.11; es de observar que la respuesta alcanza el valor de 1 (equiparable a su valor final) en aproximadamente $150\mu s$.

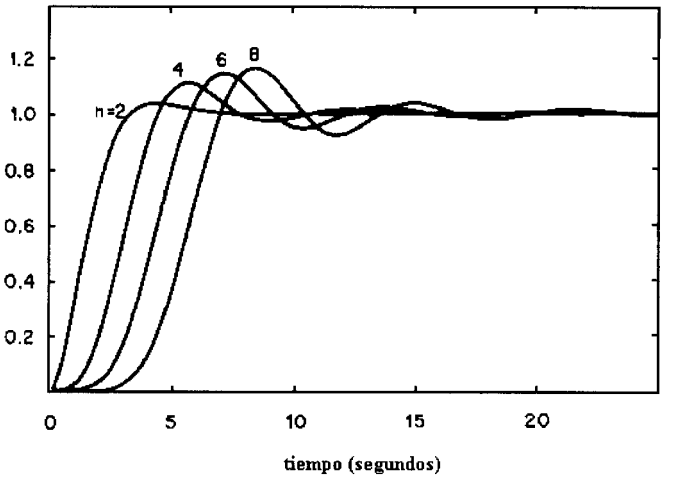

Figura 1.9 Respuesta al escalón de un filtro Butterworth pasa bajas, orden par.

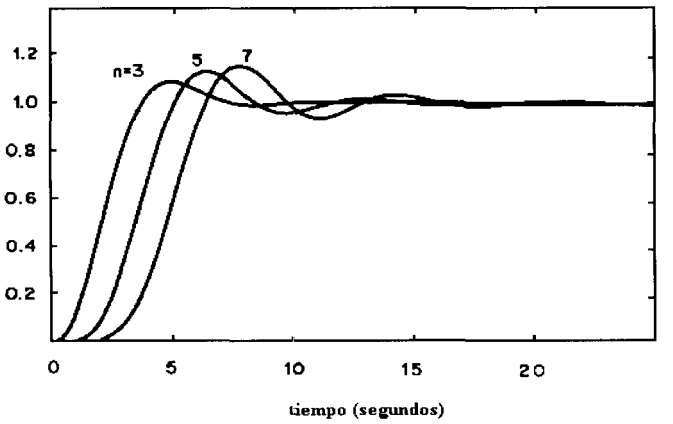

Figura 1.10 Respuesta al escalón de un filtro Butterworth pasa bajas, orden impar.

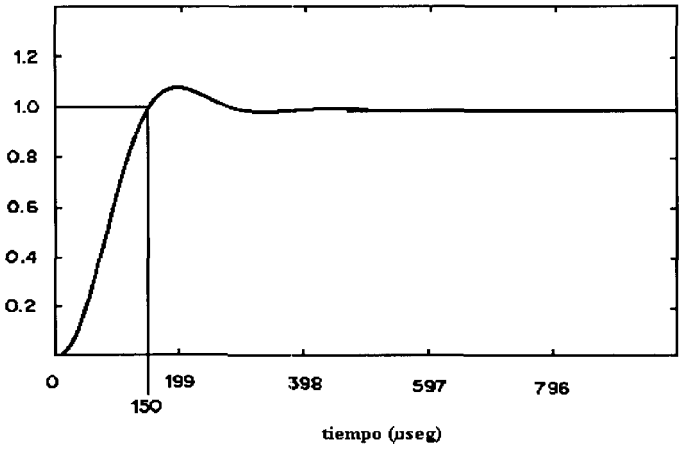

Figura 1.11 Respuesta al escalón desnormalizada.

Los ejemplos que se plantearon para encontrar las diferentes respuestas de los filtros pasa bajas Butterworth se muestran al final del capitulo.

1. 2 FILTROS CHEBYSHEV

Los filtros Chebyshev se diseñan para lograr una respuesta de amplitud con una transición relativamente aguda de la banda de paso a la banda de rechazo, agudeza que se logra a expensas de incorporar un rizo en la respuesta. Este tipo de respuesta está caracterizado por la función:

$$|H(j\omega)|^2 = \frac{1}{1+\in^2 T_n^2(\omega)} \qquad (1.8)$$

donde $\in^2 = 10^{r/10} - 1$

$T_n(\omega)$ = Polinomio de Chebyshev de orden n.

r = Nivel de rizo en la banda de paso en dB.

Una lista de polinomios Chebyshev, hasta $n = 10$, se muestra en la tabla 1.2.

n	$T_n(\omega)$
0	1
1	ω
2	$2\omega^2-1$
3	$4\omega^3-3\omega$
4	$8\omega^4-8\omega^2+1$
5	$16\omega^5-20\omega^3+5\omega$
6	$32\omega^6-48\omega^4+18\omega^2-1$
7	$64\omega^7-112\omega^5+56\omega^3-7\omega$
8	$128\omega^8-256\omega^6+160\omega^4-32\omega^2+1$
9	$256\omega^9-576\omega^7+432\omega^5-120\omega^3+9\omega$
10	$512\omega^{10}-1280\omega^8+1120\omega^6-400\omega^4+50\omega^2+1$

Tabla 1.2

1.2.1 FUNCION DE TRANSFERENCIA

La forma general de la respuesta de magnitud Chebyshev se muestra en la siguiente figura 1.12

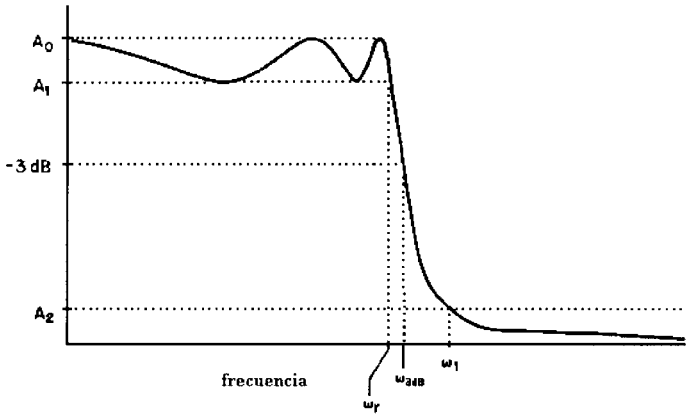

Figura 1.12 Respuesta en magnitud de un filtro pasa bajas Chebyshev

Esta respuesta se puede normalizar como lo muestra la figura 1.13.

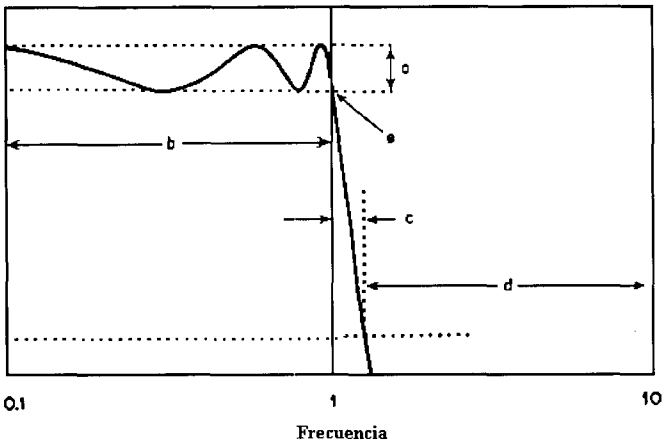

Figura 1.13 Respuesta normalizada a ω= 1 rad/s. (a) limites de rizo, (b)pasa banda, (c)banda de transición, (d) banda de rechazo

Donde el ancho de banda, en el qué queda contenido el rizo está acotado por la frecuencia $\omega_T = {}^{1rad}/_{s}$. La misma respuesta se puede normalizar, en forma ligeramente distinta, ajustando que el punto de respuesta de $-3dB$ corresponda a la frecuencia $\omega_0 = {}^{1rad}/_{s}$, como se ilustra en figura. 1.14

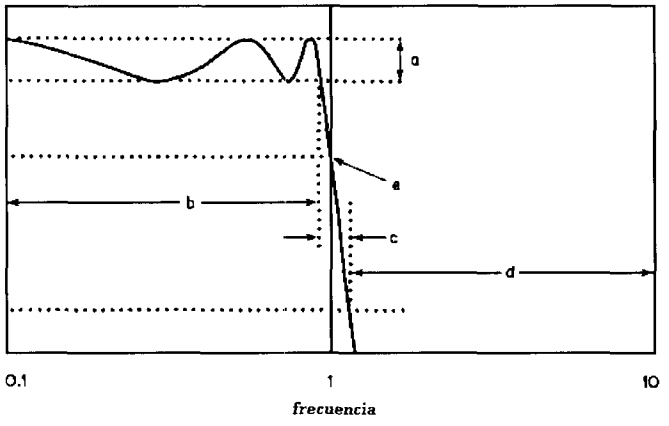

Figura 1.14 Respuesta Chebyshev normalizada (a) limites de rizo, (b) pasa banda, (c) banda de transición, (d) banda de rechazo (e) respuesta a -3dB que caen en ω=1 rad/s.

El primer tipo de normalización (el del ancho de banda del rizo) involucra cálculos más simples, pero el segundo tipo (el del punto de $-3dB$) permite comparar más fácilmente las respuestas Chebyshev con las producidas por otro tipo de filtros. (Butterworth, Bessel, etc.).

La expresión general para la función de transferencia de un filtro pasa bajas Chebyshev de orden n es:

$$H(s) = \frac{H_0}{\prod_{i=1}^{n}(s - s_i)} = \frac{H_0}{(s - s_1)(s - s_2)\cdots(s - s_n)} \qquad (1.9)$$

donde

$$H_0 = \begin{cases} \prod_{i=1}^{n}(-s_i) & n \ par \\ 10^{r/20}\prod_{i=1}^{n}(-s_i) & n \ impar \end{cases} \quad (1.10)$$

$$s_i = \sigma_i + j\omega_i \quad (1.11)$$

$$\sigma_i = \left[\frac{(1/\gamma)-\gamma}{2}\right] sen\frac{(2_i-1)\pi}{2n} \quad (1.12)$$

$$\omega_i = \left[\frac{(1/\gamma)+\gamma}{2}\right] \cos\frac{(2_i-1)\pi}{2n} \quad (1.13)$$

$$\gamma = \left(\frac{1+\sqrt{1+\epsilon^2}}{\epsilon}\right)^{1/n} \quad (1.14)$$

$$\epsilon = \sqrt{10^{r/10}-1} \quad (1.15)$$

Las fórmulas para la ubicación de los polos de la función de transferencia resultan más complicadas que las correspondientes a filtros Butterworth, además se requiere determinar los parámetros ϵ, γ y r antes que los valores de los polos sean calculados. De igual manera, todos los polos están involucrados con el cálculo del numerador H_0.

Algoritmo 1.1 Determina los polos de un filtro Chebyshev pasabajas de orden n con un ancho de banda de $1Hz$.

Paso 1 Determinar la cantidad máxima (en dB), de rizo que se puede permitir en la banda de paso de la respuesta en magnitud. Hacer r igual o menor que este valor.

Paso 2 Calcular ϵ.

$$\epsilon = \sqrt{10^{r/10}-1} \quad (1.16)$$

Paso 3 Seleccionar un orden n para el filtro, que asegure el desempeño adecuado.

Paso 4 Calcular γ

$$\gamma = \left(\frac{1+\sqrt{1+\epsilon^2}}{\epsilon}\right)^{1/n} \quad (1.17)$$

Paso 5 Para $i = 1,2,....n$, calcular:

$$\sigma_i = \left[\frac{(1/\gamma)-\gamma}{2}\right] sen\frac{(2_i-1)\pi}{2n} \quad (1.18)$$

$$\omega_i = \left[\frac{(1/\gamma)+\gamma}{2}\right] \cos\frac{(2_i-1)\pi}{2n} \quad (1.19)$$

que son la parte real e imaginaria, respectivamente, de cada polo s_i.

Paso 6 Calcular

$$H_0 = \begin{cases} \prod_{i=1}^{n}(-s_i) & n \ par \\ 10^{r/20}\prod_{i=1}^{n}(-s_i) & n \ impar \end{cases} \qquad (1.20)$$

Paso 7 Sustituir los valores H_0 y S_i a S_n en

$$H(s) = \frac{H_0}{\prod_{i=1}^{n}(s-s_i)} = \frac{H_0}{(s-s_1)(s-s_2)\cdots(s-s_n)} \qquad (1.21)$$

Consideremos la aplicación de este algoritmo para determinar la función de trasferencia de un filtro Chebyshev de tercer orden, con rizo de $0.5dB$ en la banda de paso.

solución:

Paso 1 $r = 0.5$

Paso 2 $\epsilon = \sqrt{10^{0.5/10} - 1} = \sqrt{1.122018 - 1} = 0.349311$

Paso 3 $n = 3$

Paso 4 $$\gamma = \left(\frac{1+\sqrt{1+\epsilon^2}}{\epsilon}\right)^{1/n} = \left(\frac{1+\sqrt{1+0.122018}}{0.349311}\right)^{1/3}$$

$$= \left(\frac{1+1.0592}{0.349311}\right)^{1/3} = (5.895187)^{1/3} = 1.806477$$

Paso 5 $i = 1$

$$\sigma_1 = \left(\frac{0.5535 - 1.806477}{2}\right)\text{sen}\left(\frac{\pi}{6}\right) = -0.313228$$

$$\omega_1 = \left(\frac{0.5535 + 1.806477}{2}\right)\cos\left(\frac{\pi}{6}\right) = 1.021927$$

$i = 2$

$$\sigma_2 = (-0.626488)\,\text{sen}\left(\frac{3\pi}{6}\right) = -0.626457$$

$\omega_2 = 0$

$i = 3$

$$\sigma_3 = (-0.626457)\,\text{sen}\left(\frac{3\pi}{6}\right) = -0.313228$$

$$\omega_3 = (1.180020)\cos\left(\frac{3\pi}{6}\right) = -1.021927$$

Así,

$$S_1 = -0.313228 + 1.021928j$$
$$S_2 = -0.626457$$
$$S_3 = -0.313228 - 1.021928j$$

Paso 6 con $n = 3$

$$H_0 = \prod_{i=1}^{3}(-S_i) = (-S_1)(-S_2)(-S_3) = -S_1S_2S_3$$
$$= -(-0.313228 + 1.021928j)(-0.626457)(-0.313228 - 1.021928j)$$
$$= 0.715659.$$

Paso 7

$$H(s) = \frac{0.715695}{(s + 0.313228 - 1.021928j)(s + 0.626457)(s + 0.313228 + 1.02938j)}$$

$$= \frac{0.715695}{s^3 + 1.252913s^2 + 1.534896s + 0.715695}$$

La relación correspondiente a la función de transferencia Chebyshev muestra que un filtro de orden n siempre tendrá n polos y no tendrá ceros finitos, en tanto que los polos se ubicarán en el semiplano complejo s, exactamente sobre el perímetro de una elipse cuyo eje mayor estará sobre el eje $j\omega$, y su eje menor sobre el eje σ.

Las dimensiones de la elipse y las posiciones de los polos (de la figura 1.15) dependerán de la cantidad de rizo permitido en la banda de paso; valores típicos de este rizo van desde 0.1 a $1dB$.Entre menor sea el rizo en la banda de paso, mayor será la banda de transición; de hecho, para $0dB$ de rizo, los filtros Chebyshev y Butterworth tienen exactamente la misma función de transferencia y características de respuesta.

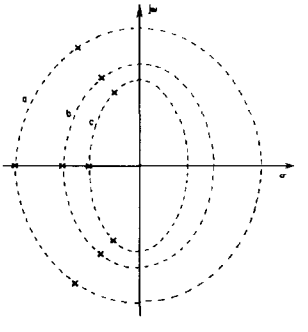

Figura 1.15 Ubicación de polos

Las tablas 1.3, 1.4 y 1.5 siguientes muestran los valores de los polos para valores de rizo de 0.1, 0.5 y 1.0 *dB*, respectivamente:

n	Polos
2	-1.186178 ± 1.380948j
3	-0.969406
	-0.484703 ± 1.206155j
4	-0.637730 ± 0.465000j
	-0.264156 ± 1.122610j
5	-0.538914
	-0.435991 ± 0.667707j
	-0.166534 ± 1.080372j
6	-0.428041 ± 0.283093j
	-0.313348 ± 0.773426j
	-0.114693 ± 1.056519j
7	-0.376778
	-0.339465 ± 0.463659j
	-0.234917 ± 0.835485j
	-0.083841 ± 1.041833j
8	-0.321650 ± 0.205314j
	-0.272682 ± 0.584684j
	-0.182200 ± 0.875041j
	-0.063980 ± 1.032181j

Tabla 1.3 Valores de los polos para filtro Chebyshev pasabajas con un rizo de 0.1 dB

n	Polos
2	-0.712812 ± 1.00402j
3	-0.626457
	-0.313228 ± 1.021928j
4	-0.423340 ± 0.420946j
	-0.175353 ± 1.016253j
5	-0.362320
	-0.293123 ± 0.625177j
	-0.111963 ± 1.011557j
6	-0.289794 ± 0.270216j
	-0.212144 ± 0.738245j
	-0.077650 ± 1.008461j
7	-0.256170
	-0.230801 ± 0.447894j
	-0.159719 ± 0.807077j
	-0.057003 ± 1.006409j
8	-0.219293 ± 0.199907j
	-0.185908 ± 0.569288j
	-0.124219 ± 0.852000j
	-0.043620 ± 1.005002j

Tabla 1.4 Valores de los polos para filtro Chebyshev pasabajas con un rizo de 0.5 dB

n	Valores
2	-0.548867 ± 0.895129j
3	-0.494171
	-0.247085 ± 0.965999j
4	-0.336870 ± 0.407329j
	-0.139536 ± 0.983379j
5	-0.289493
	-0.234205 ± 0.611920j
	-0.089458 ± 0.990107j
6	-0.232063 ± 0.266184j
	-0.169882 ± 0.727227j
	-0.062181 ± 0.993411j
7	-0.205414
	-0.185072 ± 0.442943j
	-0.128074 ± 0.798156j
	-0.045709 ± 0.995284j
8	-0.175998 ± 0.198206j
	-0.149204 ± 0.564444j
	-0.099695 ± 0.844751j
	-0.035008 ± 0.996451j

Tabla 1.5 Valores de polos para filtro Chebyshev pasabajas con un rizo de 1.0 dB

Todas las funciones de transferencia y valores de los polos corresponden a filtros normalizados, según el criterio de ancho de banda de $1Hz$. Sin embargo, es posible renormalizar la función de transferencia para lograr que el punto de $-3dB$ quede en la frecuencia de $1Hz$, usando el siguiente algoritmo. Este algoritmo asume que los valores de $\in$, H_0 y los polos s_i, ya se han obtenido para la función de transferencia que tiene ancho de banda de rizo de $1Hz$.

Algoritmo 1.2 Renormalización de una función de transferencia de un filtro pasabajas Chebyshev:

Paso 1 Calcular A usando la siguiente formula:

$$A = \frac{\cosh^{-1}[(1/\in)]}{n} = \frac{1}{n}\log\left(\frac{1+\sqrt{1-\in^2}}{\in}\right) \qquad (1.22)$$

Paso 2 Usando el resultado del paso anterior calcular R dada por:

$$R = \cosh A = \frac{e^A + e^{-A}}{2} \qquad (1.23)$$

(Nota: La tabla 1.6 lista los factores R para varios órdenes y niveles de rizo)

Paso 3 Usar R para calcular $H_{2dB}(s)$, como:

$$H_{3dB}(s) = \frac{H_0 / R^n}{\prod_{i=1}^{n}[s-(s_i/R)]} \qquad (1.24)$$

La tabla 1.6 muestra los valores de renormalización de la función de transferencia de un filtro Chebyshev .

Rizo	Orden						
	2	3	4	5	6	7	8
0.1	1.94322	1.38899	1.21310	1.13472	1.09293	1.06800	1.05193
0.2	1.67427	1.28346	1.15635	1.09915	1.06852	1.05019	1.03835
0.3	1.53936	1.22906	1.12680	1.08055	1.05571	1.04083	1.03121
0.4	1.45249	1.19348	1.10736	1.06828	1.04725	1.03464	1.02649
0.5	1.38974	1.16749	1.09310	1.05926	1.04103	1.03009	1.02301
0.6	1.34127	1.14724	1.08196	1.05220	1.03616	1.02652	1.02028
0.7	1.30214	1.13078	1.07288	1.04644	1.03218	1.02361	1.01806
0.8	1.26955	1.11699	1.06526	1.04160	1.02883	1.02116	1.01618
0.9	1.24176	1.10517	1.05872	1.03745	1.02596	1.01905	1.01457
1.0	1.21763	1.09487	1.05300	1.03381	1.02344	1.01721	1.01316
1.1	1.19637	1.08576	1.04794	1.03060	1.02121	1.01557	1.01191
1.2	1.17741	1.07761	1.04341	1.02771	1.01922	1.01411	1.01079
1.3	1.16035	1.07025	1.03931	1.02510	1.01741	1.01278	1.00978
1.4	1.14486	1.06355	1.03558	1.02272	1.01576	1.01157	1.00886
1.5	1.13069	1.05740	1.03216	1.02054	1.01425	1.01046	1.00801

Tabla 1.6

1.2.2 RESPUESTA EN FRECUENCIA

Las siguientes figuras, desde 1.16 hasta 1.21 de respuesta en frecuencia muestran, para efectos comparativos, las respuestas de magnitud y fase de varios filtros Chebyshev con niveles de rizo que van de 0.1 a $1.0dB$; todas estas gráficas se han normalizado para exhibir una frecuencia de corte en $1Hz$.

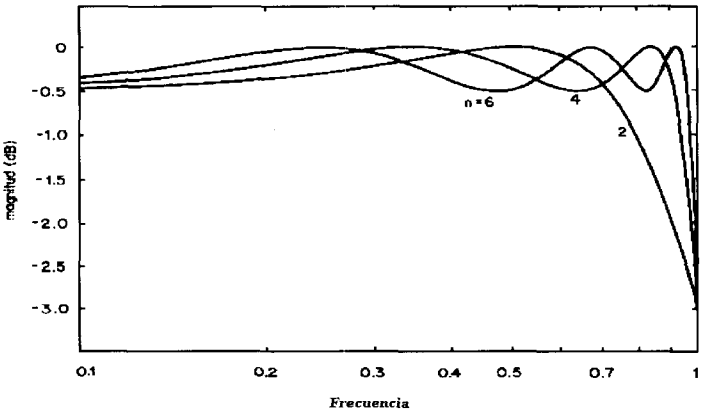

Figura 1.16 Respuesta en magnitud en la banda de paso de un filtro Chebyshev pasa bajas de orden par con rizo 0.5dB.

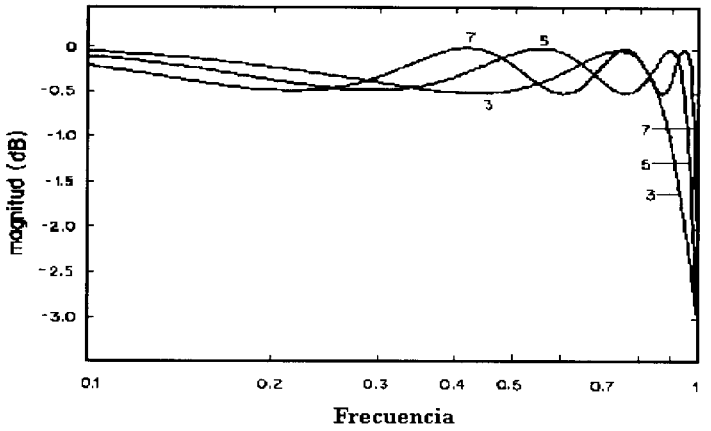

Figura 1.17 Respuesta en magnitud en la banda de paso de un filtro Chebyshev pasa bajas de orden impar con rizo 0.5dB.

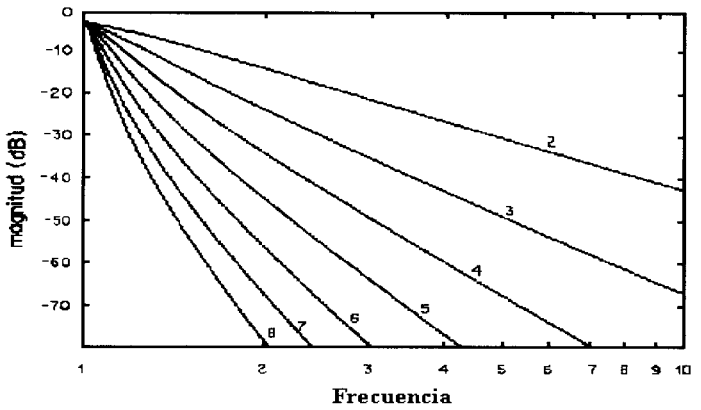

Figura 1.18 Respuesta en magnitud en la banda de rechazo de un filtro Chebyshev pasa bajas de orden impar con rizo 0.5dB.

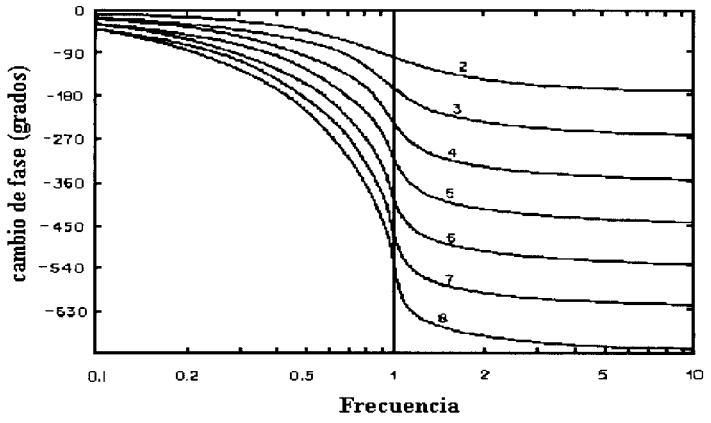

Figura 1.19 Respuesta en fase de un filtro Chebyshev pasa bajas con rizo de 0.5 dB en la banda de paso

Un ejemplo de la respuesta en magnitud Chebyshev utilizando el programa de filtros analógicos se muestra a continuación. Cuyo resultado se muestra en las figuras 1.16 y 1.19, al cambiar el valor del rizo por 0.1 ó 1 se obtienen las gráficas 1.20 y 1.21 respectivamente.

```
                    FILTROS CHEBYSHEV

                  Respuesta en frecuencia

          Orden del filtro: 4
        Valor del rizo (dB): .5
   Normalización (para 3dB teclea 3):  3
Valor de la frecuencia (Normalizada):  .7

 El resultado es:

 Magnitud = -0.446056(dB)   Fase = 138.781457°


          ¿Desea hacer otro calculo <S/N> ?: _
```

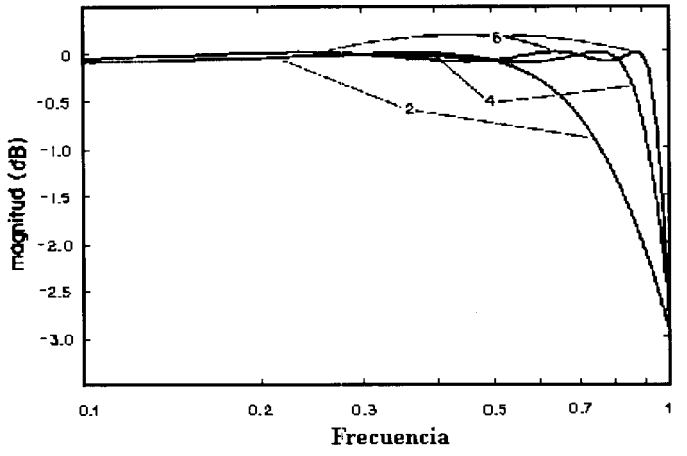

Figura 1.20 Respuesta en magnitud en la banda de paso de un filtro Chebyshev pasa bajas de orden par con rizo 0.1dB.

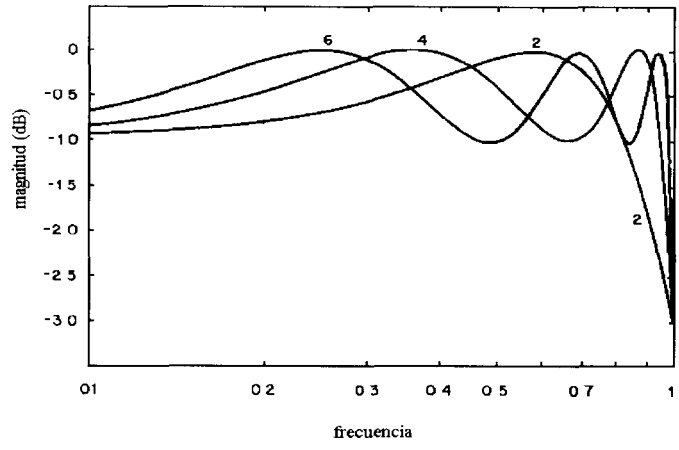

Figura 1.21 Respuesta en magnitud en la banda de paso de un filtro Chebyshev pasa bajas de orden par con rizo 1.0dB.

1.2.3 RESPUESTA IMPULSIVA Y ESCALON

En la figura 1.22 se muestran algunas respuestas impulsivas para filtros Chebyshev pasabajas con rizo de $0.5dB$.

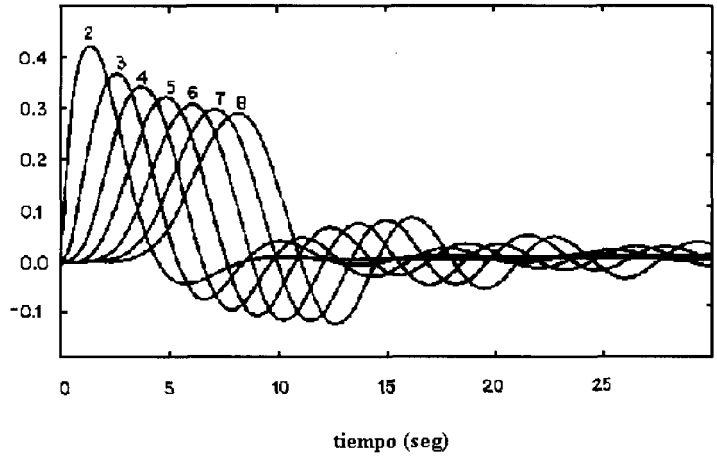

Figura 1.22 Respuesta impulsiva de un filtro Chebyshev pasa bajas con rizo de 0.5dB.

La respuesta escalón se puede obtener integrando la respuesta impulsiva a partir de la cual se pueden generar gráficas como las que se muestran en la figura 1.23.

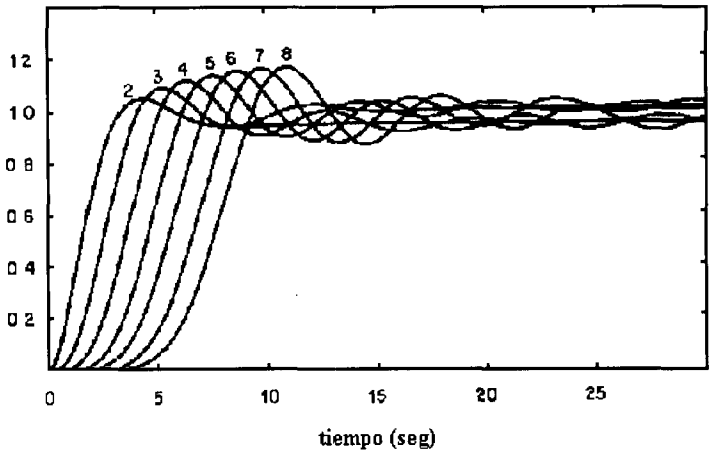

Figura. 1.23 Respuesta escalón de un filtro Chebyshev pasa bajas con rizo de 0.5dB.

A continuación se presenta un ejemplo de la respuesta impulsiva de un filtro Chebyshev, utilizando el programa de filtros analógicos, cuyos valores se verifican en la figura 1.22.

```
                    FILTROS CHEBYSHEV

                   Respuesta  impulsiva

                      Orden del filtro: 4
                   Valor del rizo (dB): .5
            Normalización (para 3dB teclea 3): 3
                     Dame el valor de delta_T
              (Separación entre puntos, segs): 1
                     Dame el número de puntos:  5
hSubZero = 0.250642
0   0.000000
1   0.030189
2   0.156230
3   0.298753
4   0.336408

              ¿Desea hacer otro calculo <S/N> ?: _
```

El programa que permite encontrar la respuesta en frecuencia de un filtro Chebyshev, así como su respuesta impulsiva se presentan a final del capitulo.

1.3 FILTROS ELIPTICOS

En los filtros Chebyshev los rizos en la banda de paso permiten obtener una mejor selectividad que los filtros Butterworth. Por otro lado, los filtros Elípticos mejoran la respuesta a partir de la realización de los filtros Chebyshev, para permitir rizos en la banda de paso y en la banda de rechazo.

La respuesta de un filtro Elíptico satisface:

$$\left|H(j\omega)^2\right| = \frac{1}{1+\epsilon^2 R_n^2(\omega, L)} \qquad (1.25)$$

donde $R_n(\omega, L)$ es un filtro Chebyshev con parámetros de rizo L de orden n

1.3.1 ESPECIFICACION DE PARAMETROS

Así como se determinó la función de transferencia de un filtro pasabajas Butterworth, con amplitud normalizada, que requiere de la especificación de dos parámetros, (frecuencia de corte ω_c y orden del filtro n), también para la determinación de la función de transferencia de filtros Chebyshev se requiere de estas especificaciones, además de un tercer parámetro (el rizo de la banda de paso).

Por otro lado, la función de transferencia para un filtro Elíptico requiere del orden, y de los siguientes parámetros, como se muestra en la figura 1.24.

A_p = máxima pérdida en la banda de paso, en dB.

A_s = mínima pérdida en la banda de rechazo, en dB

ω_p = frecuencia de corte en la banda de paso.

ω_s = frecuencia de corte en la banda de rechazo.

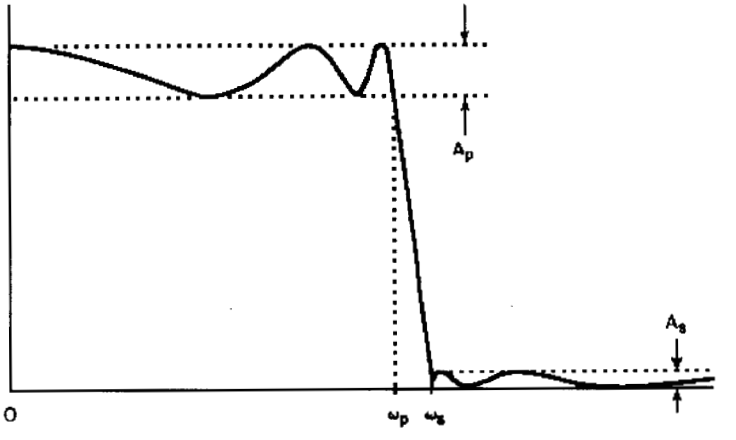

Figura 1.24 Respuesta en frecuencia de un filtro elíptico

En el diseño de filtros elípticos se considera que la amplitud máxima en la banda de paso es la unidad, donde A_p es el tamaño del rizo de la banda de paso, y A_s es el tamaño de rizo en la banda de rechazo. Cuatro de los cinco parámetros de los filtros pueden ser especificados independientemente, y con el quinto (orden n) se fija la respuesta natural de los filtros elípticos. La especificación de A_p, A_s, ω_p y ω_s tendrá que considerar los requerimientos de la aplicación en particular.

Algoritmo 1.3 Determina el orden de filtros elípticos.

Paso 1 Determinar la máxima pérdida A_p, en la banda de paso, y la mínima pérdida A_s en dB, para la banda de rechazo.

Paso 2 Determinar la frecuencia de corte ω_p en la banda de paso y la frecuencia de corte ω_s en la banda de rechazo.

Paso 3 Usando ω_p y ω_s, calcular el factor de selectividad k como:

$$k = \omega_p / \omega_s \qquad (1.26)$$

Paso 4 Calcular la constante modular q usando:

$$q = u + 2u^5 + 15u^9 + 150u^{13} \qquad (1.27)$$

$$\text{donde } u = \frac{1-\sqrt[4]{1-k^2}}{2\left(1+\sqrt[4]{1-k^2}\right)} \qquad (1.28)$$

Paso 5 Calcular el factor de discriminación D como:

$$D = \frac{10^{A_s/10}-1}{10^{A_p/10}-1} \qquad (1.29)$$

Paso 6 Calcular el mínimo orden de n requerido, como:

$$n = \left\lceil \frac{\log 16D}{\log\left(1/q\right)} \right\rceil \qquad (1.30)$$

donde $\lceil x \rceil$ denota el número entero menor ó igual a x.

La banda de rechazo mínima prevé la pérdida para cualquier combinación de A_p, A_s, ω_p y n esta dada por:

$$A_s = 10\log\left(1+\frac{10^{A_p/10}-1}{16q^n}\right) \qquad (1.31)$$

dónde q es la constante modular dada por la ecuación del paso 4.

Haciendo uso del algoritmo se puede determinar el orden mínimo de un filtro elíptico para el cual A_p es 1, $A_s \geq 50.0$, $\omega_p = 3000$ y $\omega_s = 3200$. De este modo:

$$k = \frac{3000}{3200} = 0.9375$$
$$u = 0.12897$$
$$q = 0.12904$$
$$D = \frac{10^5 - 1}{10^{0.01} - 1} = 4{,}293{,}093.82$$
$$n = \lceil 8.81267 \rceil = 9$$

Haciendo uso del programa de filtros analógicos Elípticos para este mismo ejemplo se obtiene lo siguiente:

```
Estimar orden para un Filtro Elíptico

   Frecuencia límite en la banda de paso:  3000
Frecuencia límite en la banda de rechazo:  3200
 Perdida máxima en la banda de paso (dB):  1
Nivel mínimo en la banda de rechazo (dB):  50


El orden es: 9
El nivel mínimo recalculado es: 53.232406(dB)



¿Desea hacer otro calculo <S/N> ?:
```

1.3.2 FUNCION DE TRANSFERENCIA NORMALIZADA

El diseño de los filtros elípticos se puede realizar a partir de la respuesta característica a frecuencias normalizadas, y a partir de la cual se puede verificar para cualquier frecuencia de interés. En este caso, en vez de normalizar el ancho de banda a $3dB$ ó el ancho de banda del rizo igual a la unidad, un filtro elíptico es normalizado de modo que $\sqrt{\omega_{pN}\omega_{sN}} = 1$, donde ω_{pN} y ω_{sN} son la frecuencia de corte de la banda de paso normalizada y la frecuencia de corte de la banda de rechazo normalizada, respectivamente. El factor de escala de frecuencia esta dado por:

$$\omega_{pN} = \frac{\omega_p}{\alpha} \quad \omega_{sN} = \frac{\omega_S}{\alpha} \tag{1.32}$$

sustituyendo ω_{pN} y ω_{sN} en la ecuación anterior se pueden obtener los valores de α

$$\sqrt{\frac{\omega_p \omega_s}{\alpha^2}} = 1$$

$$\alpha = \sqrt{\omega_p \omega_s} \tag{1.33}$$

Para obtener el factor de selectividad k, hay que usar las frecuencia ω_{pN} y ω_{sN}, como se muestra en la siguiente ecuación:

$$k = \frac{\omega_{pN}}{\omega_{sN}} = \frac{\omega_p / \alpha}{\omega_s / \alpha} = \frac{\omega_p}{\omega_s} \tag{1.34}$$

A continuación se presenta el algoritmo para obtener la función de transferencia de un filtro Elíptico normalizado.

Algoritmo 1.4. Determinar la función de transferencia de un filtro Elíptico.

Paso 1 Usar el algoritmo anterior para determinar los valores de A_p, A_s, ω_p, ω_s y n

Paso 2 Usando ω_p y ω_s calcular el valor del factor de selectividad k como:

$$k = \omega_p / \omega_s$$

Paso 3 Usando el factor de selectividad, calcular la constante modular q como:

$$q = u + 2u^5 + 15u^9 + 150u^{13}$$

donde $u = \dfrac{1 - \sqrt[4]{1-k^2}}{2\left(1 + \sqrt[4]{1-k^2}\right)}$

Paso 4 Usando los valores A_p y n calcular V como:

$$V = \frac{1}{2n} \ln\left(\frac{10^{A_p/20} + 1}{10^{A_p/20} - 1}\right) \tag{1.35}$$

Paso 5 Usando los valores de q y de V calcular P_0 como:

$$P_0 = \left| \frac{q^{1/4} \sum_{m=0}^{\infty} (-1)^m q^{m(m+1)} senh[(2m+1)V]}{0.5 + \sum_{m=1}^{\infty} (-1)^m q^{m^2} \cosh 2mV} \right| \tag{1.36}$$

Paso 6 Usando los valores de k y P_0 calcular W como:

$$W = \left[\left(1 + \frac{p_0^2}{k}\right)\left(1 + k p_0^2\right)\right]^{1/2} \tag{1.37}$$

Paso 7 Determinar r (número de sección cuadrática en el filtro), como:

$r = n/2$ para n par, y $r = (n-1)/2$ para n impar

Paso 8 Para $i = 1,2,...r$ calcular X_i como:

$$X_i = \frac{2q^{1/4}\sum_{m=0}^{\infty}(-1)^m q^{m(m+1)} \operatorname{sen}[(2m+1)\mu\pi/n]}{1+2\sum_{m=1}^{\infty}(-1)^m q^{m^2}\cos(2m\mu\pi/n)} \tag{1.38}$$

donde $\mu = \begin{cases} i & n \ impar \\ i-1/2 & n \ par \end{cases}$

Paso 9 Para $i = 1,2,...r$,calcular Y_i como:

$$Y_i = \left[\left(1-\frac{X_i^2}{k}\right)\left(1-kX_i^2\right)\right]^{1/2} \tag{1.39}$$

Paso 10 Para $i = 1,2,...r$, usar W , X_i y Y_i para calcular los coeficientes a_i , b_i y c_i como:

$$a_i = \frac{1}{X_i^2} \tag{1.40}$$

$$b_i = \frac{2p_0 Y_i}{1+p_0^2 X_i^2} \tag{1.41}$$

$$c_i = \frac{(p_0 Y_i)^2 + (X_i W)^2}{\left(1+p_0^2 X_i^2\right)^2} \tag{1.42}$$

Paso 11 Usando a_i y c_i , calcular H_0 como:

$$H_0 = \begin{cases} p_0 \prod_{i=1}^{r} \frac{c_i}{a_i} & n \ impar \\ 10^{-A_p/20} \prod_{i=1}^{r} \frac{c_i}{a_i} & n \ par \end{cases} \tag{1.43}$$

Paso 12 Finalmente, calcular la función de transferencia normalizada $H_N(s)$ como:

$$H_N(s) = \frac{H_0}{d}\prod_{i=1}^{r}\frac{s^2+a_i}{s^2+b_i s+c_i} \tag{1.44}$$

donde $d = \begin{cases} s+p_0 & n \ impar \\ 1 & n \ par \end{cases}$

Usando el programa para calcular los coeficientes de un filtro Elíptico de noveno orden, n=9, con A_p = 0.1 dB, ω_p = 3000 rad/s, ω_s = 3200 rad/s se obtiene lo siguiente:

```
        Calcular coeficientes de un Filtro Elíptico

  Frecuencia límite en la banda de paso: 3000
Frecuencia límite en la banda de rechazo: 3200
 Perdida máxima en la banda de paso (dB): 0.1
 Orden del filtro:  9

     El número de secciones es:  4
 El valor de Ho  normalizado es:  0.015317
 El valor de Po  normalizado es:  0.470218

 Los coeficientes normalizados de las secciones son:

 Sección      ai          bi          ci
   1      4.174973    0.678623    0.437460
   2      1.606396    0.309200    0.741549
   3      1.182293    0.112740    0.898826
   4      1.076828    0.027262    0.953895

    ¿Desea reescalar los coeficientes (s/n)? _
```

1.4 FILTROS BESSEL

1.4.1 FUNCION DE TRANSFERENCIA

Los filtros Bessel se diseñan para tener un retardo de grupo máximamente plano, y para estos la expresión general de la función de transferencia pasabajas de orden n esta dado por:

$$H(s) = \frac{b_0}{q_n(s)} \qquad (1.45)$$

donde

$$q_n(s) = \sum_{k=1}^{n} b_k s^k$$

$$b_k = \frac{(2n-k)!}{2^{n-k}\, k!(n-k)!}$$

La siguiente ecuación recursiva permite determinar $q_n(s)$ a partir de $q_{n-1}(s)$ y $q_{n-2}(s)$:

$$q_n = (2n-1)q_{n-1} + s^2 q_{n-2} \qquad (1.46)$$

En la tabla 1.7 se listan los valores de $q_n(s)$ desde $n = 2$ hasta $n = 8$

n	$Q_n(s)$
2	$S^2 + 3s + 3$
3	$S^3 + 6s^2 + 15s + 15$
4	$S^4 + 10s^3 + 45s^2 + 105s + 105$
5	$S^5 + 15s^4 + 105s^3 + 420s^2 + 945s + 945$
6	$S^6 + 21s^5 + 210s^4 + 1260s^3 + 4725s^2 + 10{,}395s + 10{,}395$
7	$S^7 + 28s^6 + 378s^5 + 3150s^4 + 17{,}325s^3 + 62{,}370s^2 + 135{,}135s + 135{,}135$
8	$S^8 + 36s^7 + 630s^6 + 6930s^5 + 51975s^4 + 270{,}270s^3 + 945{,}945s^2 + 2{,}027{,}025s + 2{,}027{,}025$

Tabla 1.7

Los coeficientes que representan la función de Transferencia de un filtro Bessel de orden 8 , los obtenemos mediante el programa de filtros analógicos, los cuales se ilustran en la tabla 1.7.

```
                El valor de los coeficientes es:

El coeficiente de S^0 = 135135.000000
El coeficiente de S^1 = 135135.000000
El coeficiente de S^2 = 62370.000000
El coeficiente de S^3 = 17325.000000
El coeficiente de S^4 = 3150.000000
El coeficiente de S^5 = 378.000000
El coeficiente de S^6 = 28.000000
El coeficiente de S^7 = 1.000000


¿Desea hacer otro calculo <S/N> ?: _
```

La ecuación que expresa la función general de la función de transferencia de un filtro Bessel, no presenta una expansión explícita para los polos, por lo que el denominador deberá tratarse de modo de encontrar sus raíces para determinar la posición de sus polos. La tabla 1.8 lista la ubicación aproximada de polos, desde $n=2$ hasta $n=8$.

n	Valores de los polos
2	-1.5 ± 0.8660j
3	-2.3222
	-1.8390 ± 1.7543j
4	-2.1039 ± 2.6575j
	-2.8961 ± 0.8672j
5	-3.6467
	-2.3247 ± 3.5710j
	-3.3520 ± 1.7427j
6	-2.5158 ± 4.4927j
	-3.7357 ± 2.6263j
	-4.2484 ± 0.8675j
7	-4.9716
	-2.6857 ± 5.4206j
	-4.0701 ± 3.5173j
	-4.7584 ± 1.7393j
8	-5.2049 ± 2.6162j
	-4.3683 ± 4.4146j
	-2.8388 ± 6.3540j
	-5.5878 ± 0.8676j

Tabla 1.8

La función de transferencia obtenida esta normalizada para lograr un retardo unitario en $\omega=0$. Los polos p_k y los coeficientes del denominador, b_k, pueden ser renormalizados para una frecuencia $\omega=1$ en el punto $-3dB$, usando el siguiente cambio de variable:

$$p_k' = Ap_k \qquad b_k' = A^{n-k} b_k$$

dónde el valor aproximado de A para un orden n es seleccionado de la tabla 1.9 (los valores han sido incorporados al programa realizados en C). Las figuras 1.25 y 1.26 muestran la respuesta en magnitud de los filtros Bessel para diferentes órdenes, en tanto que la respuesta, en términos del retardo de grupo, se aprecia en la figura 1.27.

n	A
2	1.35994
3	1.744993
4	2.13011
5	2.42003
6	2.69996
7	2.95000
8	3.17002

Tabla 1.9

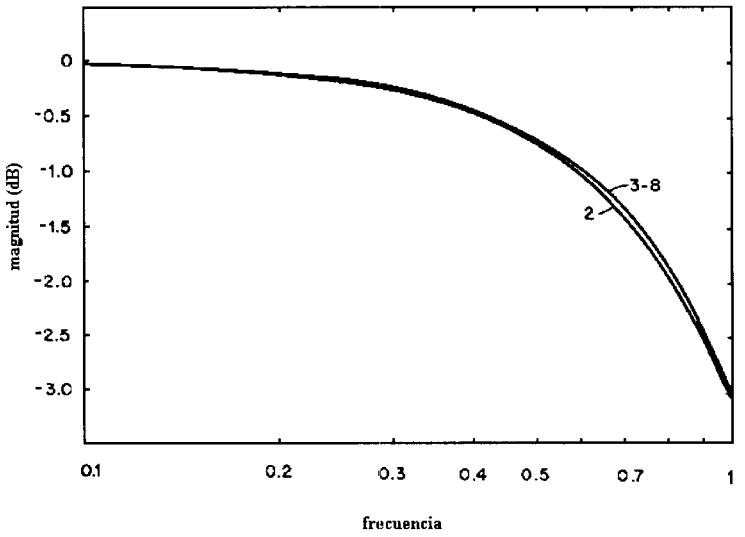

Figura 1.25 Respuesta en magnitud de un filtro pasa bajas Bessel en la banda de paso

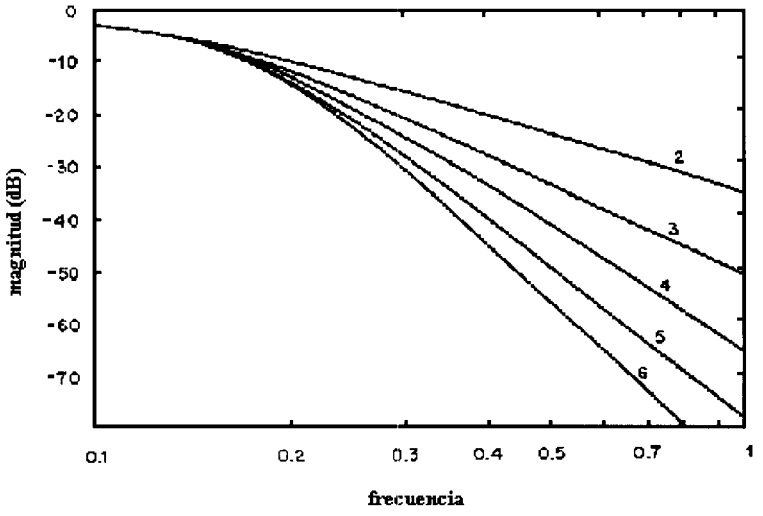

Figura 1.26 Respuesta en magnitud de un filtro Bessel en la banda de rechazo

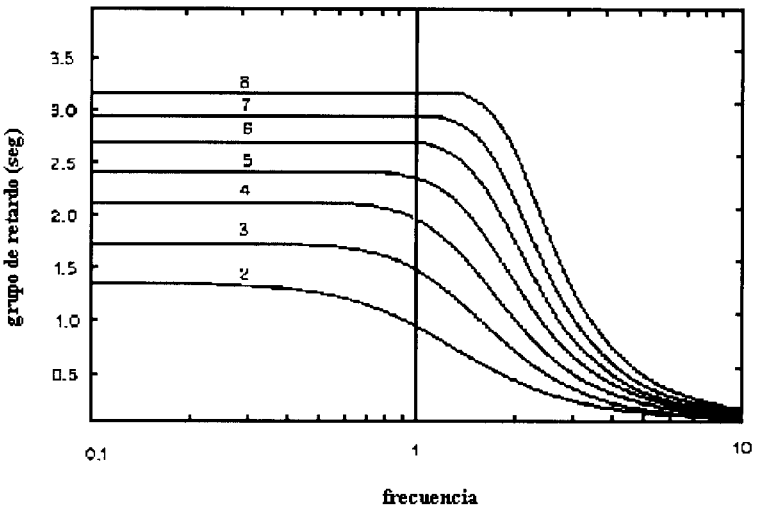

Figura 1.27 Respuesta de retardo de grupo de un filtro Bessel pasa bajas

Para calcular la respuesta en frecuencia de un filtro Bessel de orden 5 normalizado con una frecuencia de 0.4 Hz, mediante el programa de filtros analógicos Bessel obtenemos lo siguiente:

```
                Calcular la respuesta en frecuencia

                              Orden del filtro: 5

     Tipo de aproximación para el retardo
  de grupo (d)esnormalizada/(n)ormalizada: n
           ¿Cúal es la frecuencia?: .4
        ¿Cúal es el valor de delta?: 1

             La magnitud es:  -0.455330(dB)
                 La fase es:  -48.818329°
           El retardo de grupo es:  2.071354 segs.


  ¿Desea para otra frecuencia (s/n)?
```

LISTADO DE PROGRAMAS DE FILTROS ANALOGICOS

```
/*-------------------------------------------------------------------------*/
/*                                  | Esta funcion calcula la magnitud y   */
/*   Programa 1.1                   | la fase de un filtro Butterworth en  */
/*                                  | una frecuencia dada.  Ademas calcula */
/*   RespuestaFrecButterworth       | los polinomios correspondientes al   */
/*                                  | orden del filtro                     */
/*-------------------------------------------------------------------------*/

#include <math.h>
#include <stdio.h>
#include "globDefs.h"
#include "protos.h"

void respuestaFrecbutterworth(int ordenf,real frecuenciaf,
                              real *magnitudf, real *fasef)
{
   struct complex polo, s, numer, denom, funciontransfer;
   real x;
   int k;
   numer = cmplx(1.0,0.0);          // inicializacion del numerador y denominador
   denom = cmplx(1.0,0.0);

   s = cmplx(0.0, frecuenciaf);

   for(k=1; k<=ordenf; k++){        // se calcula cada uno de los polos que
                                    //  corresponden al orden del filtro.

       x = PI * ((double)(ordenf + (2*k)-1)) / (double)(2*ordenf);
         //Ecuacion 1.1
       polo = cmplx( cos(x), sin(x));
       denom = cMult(denom, cSub(s,polo));       //Se imprime cada polo obtenido
       gotoxy(18,8+k);printf("polo %d= %lf + %lf j",k,polo.x,polo.y );
   }
                                //Se obtiene la Función de Transferencia
   funciontransfer = cDiv(numer, denom);
   *magnitudf = 20.0 * log10(cAbs(funciontransfer));    // se obtiene la magnitud
   *fasef = 180.0 * arg(funciontransfer) / PI;          // se calcula la fase
   return;
}
```

```
/*                                                                          */
/*                                  :  Esta funcion calcula la respuesta    */
/*  Programa 1.2                    :  impulsiva de un filtro Butterworth   */
/*                                                                          */
/*  RespuestaImpulsivaButterworth()                                         */
/*                                                                          */
/*                                                                          */
```

```
void respuestaImpulsivabutterw(      int ordenf,real delta_t,int npts,real yval[])
{
   real L, M, x, R, I, LT, MT, cosPart, sinPart, h_de_t;
   real K, sigma, omega, t;
   int ix, r, p, ii, iii;
   real ymax, ymin;

   gotoxy(20,1);printf ("  RESPUESTA  IMPULSIVA  BUTTERWORH \n");
   p=5;
   gotoxy(29,3);printf ("t         h[t]  ");
   for( ix=0; ix <= npts; ix++){

        gotoxy(29,p);printf("%d",ix);
        h_de_t = 0.0;
        t = delta_t * ix;      //Se aplica la definición de la Función
                               //de Transferencia para F. Butterworth
        for( r=1; r <= (ordenf>>1); r++){
             x = PI * (double)(ordenf + (2*r)-1) / (double)(2*ordenf);
             sigma = cos(x);
             omega = sin(x);
             L = 1.0; M = 0.0;
             for( ii=1; ii<=ordenf; ii++){
                  if( ii == r ) continue;
                  x = PI * (double)(ordenf + (2*ii)-1) / (double)(2*ordenf);
                  R = sigma - cos(x);
                                    //Se obtiene la parte real e imaginaria
                  I = omega - sin(x);
                  LT = L*R - M*I;
                  MT = L*I + R*M;
                  L = LT; M = MT;
             }
             L = LT / (LT*LT + MT*MT);        //Ecuación 1.6
             M = -MT /(LT*LT + MT*MT);
             cosPart = 2.0 * L * exp(sigma*t) * cos(omega*t);
             sinPart = 2.0 * M * exp(sigma*t) * sin(omega*t);
                                  // Construcción d la respuesta h(t)
             h_de_t = h_de_t + cosPart - sinPart;
        }
        if( (ordenf%2) == 0){          // Se obtiene la respuesta impulsiva
            yval[ix] = h_de_t;
            gotoxy(35,p);printf("   %lf\n",yval[ix]);
               // para un filtro de n par
            if( (real) h_de_t > ymax) ymax = h_de_t;
            if( (real) h_de_t < ymin) ymin = h_de_t;
            p=p+1;
            continue;
        }
```

```
/* Se calcula el componente  exponencial real para una respuesta de orden impar */

        K = 1.0;L = 1.0;M = 0.0;
        r = (ordenf+1)/2;       // Aplicación de la ecuación 1.1
        x = PI * (double)(ordenf + (2*r)-1) / (double)(2*ordenf);
                                // Se calcula la parte real e imaginaria de h(t)
        sigma = cos(x);
        omega = sin(x);
        for( iii=1; iii<=ordenf; iii++){
             if( iii == r) continue;
             x= PI * (double)(ordenf + (2*iii)-1) / (double)(2*ordenf);
             R = sigma - cos(x);
             I = omega - sin(x);

             LT = L*R - M*I;
             MT = L*I + R*M;
             L = LT; M = MT;
        }
        K = LT / (LT*LT + MT*MT);
        h_de_t = h_de_t + K * exp(-t);              // construcción de h(t)
        yval[ix] = h_de_t;


        gotoxy(35,p);printf(" %lf\n",yval[ix]);     // se imprime el resultado
                                                    // para un filtro de n impar
        if( (real) h_de_t > ymax) ymax = h_de_t;
        if( (real) h_de_t < ymin) ymin = h_de_t;
        p=p+1;
    }
    return;
}
```

```
/*                                                                    */
/*                                  |  Programa que calcula la magnitud y fase     */
/*  Programa 1.4                    |  de un filtro Chebyshev                      */
/*                                                                    */
/*  RespuestaFrecChebyshev()                                          */
/*                                                                    */
/*                                                                    */

#include <math.h>
#define PI 3.141592653589


void respuestaFrecChebyshev(int ordenf, real rizof, char tiponormalizacion,
                            real frecuenciaf, real *magnitudf, real *fasef)
{
   double A, gamma, epsilon, work;
   double rp, ip, x, i, r, rpt, ipt;
   double Frecuencianormaliz, hSubcero;
   int k, ix;
                                        // Ecuación 1.8.  Sección 1.2
   epsilon = sqrt( -1.0 + pow( (double)10.0, (double)(rizof/10.0) ));
   gamma = pow( (( 1.0 + sqrt( 1.0 + epsilon*epsilon))/epsilon),
                               (double)(1.0/(float) ordenf) );
   if( tiponormalizacion == '3' ){    // Normalización a 3dBs
       work = 1.0/epsilon;
       A = ( log( work + sqrt( work*work - 1.0) ) ) / ordenf;
       Frecuencianormaliz = frecuenciaf * ( exp(A) + exp(-A))/2.0;
   }
   else{
       Frecuencianormaliz = frecuenciaf;
   }
   rp = 1.0;
   ip = 0.0;
                       // Expresión para obtener la función de Transferencia
                       // de un filtro Chebyshev. Ecuación 1.9, 1.11 y 1.12

   for( k=1; k<=ordenf; k++){
       x = (2*k-1) * PI / (2.0*ordenf);
       i = 0.5 * (gamma + 1.0/gamma) * cos(x);
       r = -0.5 * (gamma - 1.0/gamma) * sin(x);
       rpt = ip * i - rp * r;
       ipt = -rp * i - r * ip;
       ip = ipt; rp = rpt;
   }
   hSubcero = sqrt( ip*ip + rp*rp);
   if( ordenf%2 == 0 ){                 //Se obtiene ho de un filtro de orden par
       hSubcero = hSubcero / sqrt(1.0 + epsilon*epsilon);
   }
   rp = 1.0;
   ip = 0.0;            //Aplicación de la ecuación 1.11 y 1.12
   for( k=1; k<=ordenf; k++){
       x = (2*k-1)*PI/(2.0*ordenf);
       i = 0.5 * (gamma + 1.0/gamma) * cos(x);
       r = -0.5 * (gamma - 1.0/gamma) * sin(x);
       rpt = ip*(i-Frecuencianormaliz) - rp*r;
       ipt = rp*(Frecuencianormaliz-i) - r*ip;
       ip=ipt;
       rp=rpt;
   }           // Se calcula la magnitud y fase de la función de Transferencia.

   *magnitudf = 20.0 * log10(hSubcero/sqrt(ip*ip+rp*rp));
   *fasef = 180.0 * atan2( ip, rp) /PI;
   return;
}
```

```
/*                                                                    */
/*                                  | Programa que calcula la respuesta */
/*   Programa 1.4                   | impulsiva de un filtro Chebyshev, */
/*                                  | de orden n                        */
/*   RespuestaImpulsivaChebyshev    |                                   */
/*                                  |                                   */
/*                                                                    */
```

```
void respuestaImpulsivaCheby (int ordenf, real rizof, char tiponormalizacion,
                              real delta_t, int npts, real yval[])
{
   double m, p;
   double A, gamma, epsilon, work, normFactor;      // variables locales
   double rp, ip, x, i, r, rpt, ipt, ss;
   double hSubcero, h_de_t, t, sigma, omega;
   double K, L, M, LT, MT, I, R, cosPart, sinPart;
   int k, ix, ii, iii, rrr;

   epsilon = sqrt( -1.0 + pow( (double)10.0, (double)(rizof/10.0) ));

   if( tiponormalizacion == '3'){  // normalizacion a 3dB
       work = 1.0/epsilon;
       A = ( log( work + sqrt( work*work - 1.0) ) ) / ordenf;
       normFactor = ( exp(A) + exp(-A))/2.0;
   }
   else {
       normFactor = 1.0;
   }
   gamma = pow( (( 1.0 + sqrt( 1.0 + epsilon*epsilon))/epsilon),
                              (double)(1.0/(float) ordenf) );

   //  Se calcula H_cero
   rp = 1.0;
   ip = 0.0;

   for( k=1; k<=ordenf; k++){
       x = (2*k-1) * PI / (float)(2*ordenf);
       i = 0.5 * (gamma + 1.0/gamma) * cos(x)/normFactor;
       r = -0.5 * (gamma - 1.0/gamma) * sin(x)/normFactor;
       rpt = ip * i - rp * r;
       ipt = -rp * i - r * ip;
       ip = ipt;
       rp = rpt;
   }
   hSubcero = sqrt( ip*ip + rp*rp);

   if( ordenf%2 == 0 ){
       hSubcero = hSubcero / sqrt(1.0 + epsilon*epsilon);
   }
   printf("hSubcero = %f\n",hSubcero);       /* se imprime el resultado */

   for( ix=0; ix<npts; ix++){
       printf("%d",ix);
       h_de_t = 0.0;
       t = delta_t * ix;

       for( rrr=1; rrr <= (ordenf >> 1); rrr++){

           x = (2*rrr-1)*PI/(2.0*ordenf);

           sigma = -0.5 * (gamma - 1.0/gamma) * sin(x)/normFactor;

           omega = 0.5 * (gamma + 1.0/gamma) * cos(x)/normFactor;
```

```
            // Se calcula Lx y Mx

            L = 1;
            M = 0;
            for(ii=1; ii<=ordenf; ii++){
              if( ii == rrr) continue;
              x = (2*ii-1) * PI /(float)(2*ordenf);
              R = sigma -(-0.5*(gamma -1.0/gamma))*sin(x) / normFactor;
              I = omega -(0.5*(gamma +1.0/gamma))*cos(x) / normFactor;


              LT = L * R - M * I;
              MT = L * I + R * M;
              L = LT; M = MT;
            }
            L = LT / (LT * LT + MT * MT);
            M = -MT / (LT * LT + MT * MT);
            cosPart = 2.0 * L * exp(sigma*t) * cos(omega*t);
            sinPart = 2.0 * M * exp(sigma*t) * sin(omega*t);


            h_de_t = h_de_t + cosPart - sinPart;
        }
        if( (ordenf%2) == 0 ){
            yval[ix] = h_de_t * hSubcero;
            printf("   %lf\n",yval[ix]);
        }
        else {
              /*  Se calcula la componente  exponencial real
              .*  presente en la respuesta de orden impar

            K = 1;  L = 1;  M = 0;
            rrr = (ordenf+1) >> 1;
            x = (2*rrr-1) * PI / (float)(2*ordenf);
            sigma = -0.5 * (gamma - 1.0/gamma) * sin(x) / normFactor;
            omega = 0.5 * (gamma + 1.0/gamma) * cos(x) / normFactor;

            for( iii=1; iii<= ordenf; iii++){
                if(iii == rrr) continue;
                x = (2*iii-1) * PI / (float)(2*ordenf);
                R = sigma -(-0.5*(gamma -1.0/gamma))*sin(x) / normFactor;
                I = omega -(0.5*(gamma +1.0/gamma))*cos(x) / normFactor;
                LT = L * R - M * I;
                MT = L * I + R * M;
                L = LT;M = MT;
            }

            K = LT / (LT*LT + MT*MT);
            h_de_t = h_de_t + K * exp(sigma*t);
            yval[ix] = h_de_t * hSubcero;
            printf("\n%lf\n",yval[ix]);
        }                                          //imprime resultado
    }
    return;
}
```

```
/*-------------------------------------------------------------------------*/
/*                                                                         */
/*   Programa 1.5          |  Este programa determina el orden de un filtro */
/*                         |  Elíptico, a partir de especificar los parametros */
/*   estimarOrden()        |  caracteristicos de estos filtros.(Ap, As, Wp, Ws) */
/*                         |  Algoritmo 1.3                                */
/*-------------------------------------------------------------------------*/

#include "support.c"
FILE *dumpFile;

void estimarOrden( real omegaPaso,real omegaStop,
                   real maxperdidapaso,real minperdidaparo,
                   int *ordenf,real *actualMinperdidaparo)
{
   real k, u, q, dd, kk, lambda, w, mu, om;
   real sum, term, denom, numer, sigma, v;
   int i, m, r;
   k=omegaPaso/omegaStop;            // Se calcula factor de selectividad

   kk=sqrt(sqrt(1.0 - k*k));         // Ecuación (1.28)
   u=0.5*(1.0-kk)/(1.0+kk);

   q = 150.0 * ipow(u,13);           // Constante modular. Ecuación (1.27)
   q = q + 15.0 * ipow(u,9);
   q = q + 2.0 * ipow(u,5);
   q = q + u;

   dd = pow(10.0, minperdidaparo/10.0) - 1.0;     // Factor de descriminación.
     // Ec (1.29)
   dd = dd/ (pow(10.0,maxperdidapaso/10.0) - 1.0);

   *ordenf = ceil( log10(16.0*dd) / log10(1.0/q));   // Calcular orden mínimo.
                                                     // Ecuación (1.30)

                                                    //  Ecuación (1.31)
   numer = pow(10.0, (maxperdidapaso/10.0))-1.0;    //  Mínima perdida en banda
                                                    //  de paso
   *actualMinperdidaparo = 10.0 * log10(numer/(16*ipow(q,*ordenf))+1.0);
   return;
}
```

```
/*-------------------------------------------------------------------*/
/*                                                                   */
/*   Programa 1.5          Este programa calcula los coeficientes   */
/*                         de la función de Transferencia de un     */
/*   calcularCoeficient()  filtro eliptico.                         */
/*                                                                   */
/*-------------------------------------------------------------------*/

void calcularCoefic(real omegaPaso, real omegaStop,real maxperdidapaso,
                                     int ordenf, real aa[], real bb[],
real cc[],
                                     int *numSecs, real *hZero, real *pZero)
{
   real k, kk, u, q, vv, ww, mu, xx, yy;
   real sum, term, denom, numer;
   int i, m, r;

   k=omegaPaso/omegaStop;            // Calcula el factor de selectividad


   kk=sqrt(sqrt(1.0 - k*k));         // Ecuación 1.28
   u=0.5*(1.0-kk)/(1.0+kk);

   q = 150.0 * ipow(u,13);           // Constante modular
   q = q + 15.0 * ipow(u,9);
   q = q + 2.0 * ipow(u,5);
   q = q + u;
                                     // Ecuacion (1.25)
   numer = pow(10.0,maxperdidapaso/20.0)+1.0;
   vv = log( numer / (pow(10.0, maxperdidapaso/20.0)-1))/(2.0*ordenf);

   sum = 0.0;                        // Paso 4 algoritmo 1.4
   for( m=0; m<5; m++) {
         term = ipow(-1.0,m);
         term = term * ipow(q, m*(m+1));
         term = term * sinh((2*m+1) * vv);
         fprintf(dumpFile,"for m=%d, term = %e\n",m,term);
         sum = sum + term;
   }
   numer = 2.0 * sum * sqrt(sqrt(q));
   sum = 0.0;
   for( m=1; m<5; m++) {
         term = ipow(-1.0,m);
         term = term * ipow(q,m*m);
         term = term * cosh(2.0 * m * vv);
         sum = sum + term;
   }
   denom = 1.0 + 2.0*sum;
   *pZero = fabs(numer/denom);               // Se obtiene Po

   ww = 1.0 + k * *pZero * *pZero;           // Determinar num. de selección
   ww = sqrt(ww * (1.0 + *pZero * *pZero/k));
   r = (ordenf-(ordenf%2))/2;                 // Calcular Xi
   *numSecs = r;

   for(i=1; i<=r; i++) {              // Loop para Algoritmo 1.4, pasos 8, 9, 10
        if(ordenf%2){
           mu = i;}
        else   {
          mu = i - 0.5;}
        sum = 0.0;                    // Ecuacion (1.38) numerador

        for(m=0; m<5; m++) {
               term = ipow(-1.0,m);
               term = term * ipow(q, m*(m+1));
               term = term * sin( (2*m+1) * PI * mu / ordenf);
               sum = sum + term;
        }
```

```
        numer = 2.0 * sum * sqrt(sqrt(q));
        sum = 0.0;                      // Ecuacion (1.38) denominador
        for(m=1; m<5; m++) {
              term = ipow(-1.0,m);
              term = term * ipow(q,m*m);
              term = term * cos(2.0 * PI * m * mu / ordenf);
              fprintf(dumpFile,"for m=%d, term = %e\n",m,term);


              sum = sum + term;
        }

        denom = 1.0 + 2.0 * sum;
        xx = numer/denom;
        yy = 1.0 - k * xx*xx;                   // Calcula Yi
        yy = sqrt(yy * (1.0-(xx*xx/k)));
        aa[i] = 1.0/(xx*xx);                    // Calcula coeficiente ai
        denom = 1.0 + ipow(*pZero*xx, 2);       // Calcula coeficiente bi
        bb[i] = 2.0 * *pZero * yy/denom;
        denom = ipow(denom,2);                  // Calcula coeficiente ci
        numer = ipow(*pZero*yy,2) + ipow(xx*ww,2);
        cc[i] = numer/denom;
   }

   term = 1.0;
   for(i=1; i<=r; i++) {                        // Se obtiene la Función
        term = term * cc[i]/aa[i];              // de transferencia normalizada
   }
   if(ordenf%2){
        term = term * *pZero;}
   else {
        term = term * pow(10.0, maxperdidapaso/(-20.0));}
   *hcero = term;              // Valor de Ho incluyendo la perdida máxima de paso
   return;
}
```

```
/*                                                                    */
/*                              Este programa calcula la magnitud y fase      */
/*   Programa 1.7               de un filtro Eliptico. Y determinar la        */
/*                              función  de transferencia.                    */
/*   calcuRespuestaFrec()                                             */
/*                                Algoritmo 1.4                       */
/*                                                                    */

void calcuRespuestaFrec(int ordenf, real aa[], real bb[],real cc[],
                        real hZero,    real pZero,    real frecuenciaf,
                        real *magnitudf, real *fasef)
{
   double normalizedFrecuency;
   int r, k, ix, i;
   struct complex s, cProd, cTermNumer, cTermDenom;

   r = (ordenf-(ordenf%2))/2;
   s = cmplx(0.0, frecuenciaf);

   if(ordenf%2) {
cTermDenom = cAdd(s, cmplx(pZero, 0.0));     // Se obtiene Función
                                             // de Transferencia.
         cProd = cDiv(cmplx(1.0,0.0), cTermDenom);   // para un filtro de n par
         cProd = sMult(hZero, cProd);
   }
   else {
         cProd = cmplx(hZero,0.0);
   }
   for( i=1; i<=r; i++) {                     // con los coeficientes se
                                              // recalcula
         cTermNumer=cMult(s,s);               // la función de transferencia
         cTermDenom=cAdd(cTermNumer,sMult(bb[i],s));
         cTermNumer.x = cTermNumer.x + aa[i];
         cTermDenom.x = cTermDenom.x + cc[i];
         cProd = cMult(cProd, cTermNumer);
         cProd = cDiv(cProd, cTermDenom);
  }
  *magnitudf = 20.0* log10(cAbs( cProd));     //se calcula la magnitud de la
                                              //función
  *fasef = 180.0 * arg(cProd)/PI;             //se calcula la fase
  return;
}
```

```
/*                                                                    */
/*                          Este programa calcula los coeficientes    */
/*   Programa 1.8           a otra frecuencia dada, a partir de los   */
/*                          ya obtenidos.                             */
/*   reescalarCoef()                                                  */
/*                                                                    */
/*                                                                    */

void reescalarCoef(int ordenf, real aa[], real bb[], real cc[],
                   real *hZero, real *pZero,  real alpha)
{
   real alphaSqrd;
   int r, i;
   alphaSqrd = alpha*alpha;
   if( ordenf%2) {                    //filtro de orden par
        r = (ordenf-1)/2;
        *hZero = *hZero * alpha;
        *pZero = *pZero * alpha;
   }
   else {
        r = ordenf/2;                  //filtro de orden impar
   }
   for(i=1; i<=r; i++) {
        aa[i] = aa[i] * alphaSqrd;     //calculo de los nuevos coeficientes
        cc[i] = cc[i] * alphaSqrd;
        bb[i] = bb[i] * alpha;
   }
}
```

```
/*                                                                    */
/*                              | Este programa permite obtener los coeficien- */
/*   Programa 1.9               | tes de un filtro Bessel, los cuales se       */
/*                              | listan en la tabla 1.7 del capitulo 1.       */
/*   CoeficientesBessel()       |                                              */
/*                                                                    */
/*                                                                    */
```

```
#include <math.h>
#include "globDefs.h"
#include <conio.h>

void coeficientesBessel (int ordenf, char tiponormalizacion, real coef[])
{
   int i, N,x, index, indexM1, indexM2;
   real B[3][MAXORDEN];
   real A, renorm[MAXORDEN];
   renorm[2] = 0.72675;
   renorm[3] = 0.57145;
   renorm[4] = 0.46946;
   renorm[5] = 0.41322;    //inicialización del arreglo
   renorm[6] = 0.37038;
   renorm[7] = 0.33898;
   renorm[8] = 0.31546;
   A = renorm[ordenf];
   index = 1;
   indexM1 = 0;            // Inicialización de variables
   indexM2 = 2;
   for( i=0; i<(3*MAXORDEN); i++) B[0][i] = 0;
      B[0][0] = 1.0;
      B[1][0] = 1.0;       // Inicializacion del arreglo
      B[1][1] = 1.0;

   for( N=2; N<=ordenf; N++){
         index = (index+1)%3;
         indexM1 = (indexM1 + 1)%3;
         indexM2 = (indexM2 + 1)%3;
         for( i=0; i<N; i++){                  //calculo de los coeficientes
              B[index][i] = (2*N-1) * B[indexM1][i];
         }
         for( i=2; i<=N; i++){
               B[index][i] = B[index][i] + B[indexM2][i-2];
        }
   }
   if(tiponormalizacion == 'd'){          // si es desnormalizada los
                                          // coeficientes cambian
        for( i=0; i<=ordenf; i++) coef[i] = B[index][i];
   }
   else {
        for( i=0; i<=ordenf; i++){
              coef[i] = B[index][i] * pow(A, (ordenf - i) );
        }
   }
   gotoxy(26,4);printf("El valor de los coeficientes es: ");
                     //imprime en pantalla los coeficientes
   x=6;
   for (i=0; i<=ordenf;i++){
        gotoxy(15,x);printf("El coeficiente de S^%d = %f", i,B[index][i]);
        x++;
   }
   return;
}
```

```
/*                                                                    */
/*                                  Este programa calcula la magnitud y fase */
/*   Programa  1.10                 de un filtro Bessel de orden n, asi como */
/*                                  su funcion de transferencia.      */
/*   RespuestaEnFrecuenciaBessel()                                    */
/*                                                                    */
/*                                                                    */
```

```
void respuestaFrecBessel(int ordenf, real coef[], real frecuenciaf,
                         real *magnitudf,      real *fasef)
{
   struct complex numer, omega, denom, funcionTransfer;
   int i;
   numer = cmplx( coef[0], 0.0);
   omega = cmplx( 0.0, frecuenciaf);          // inicializacion de las
                                              // estructuras
   denom = cmplx( coef[ordenf], 0.0);         // complex, implementadas en esta
                // rutina

   for( i=ordenf-1; i>=0; i--) {
         denom = cMult(omega,denom);           //Se obtienen los polos de la
                //función
         denom.x = denom.x + coef[i];
   }
   funcionTransfer = cDiv( numer, denom);            // calcula la función de
                                                     //Transferencia
   *magnitudf = 20.0 * log10(cAbs(funcionTransfer));
   *fasef = 180.0 * arg(funcionTransfer) / PI;       // calcula magnitud y fase
   return;
}
```

```
/*                                                                    */
/*                                                                    */
/*  Programa 1.11              |  Este programa permite calcular el     */
/*                             |  retardo de grupo de un filtro Bessel. */
/*  retardoDeGrupoBessel()     |                                        */
/*                             |                                        */
/*                                                                    */
```

```
void retardoGrupoBessel(int ordenf, real coef[], real frecuenciaf,
                        real delta, real *retardoGrupo)
{
   struct complex numer, omega, omegaPlus, denom, funcionTransfer;
   int i;
   real fasef, fasef2;

   numer = cmplx( coef[0], 0.0);
   denom = cmplx( coef[ordenf], 0.0);      //inicialización de variables
   omega = cmplx( 0.0, frecuenciaf);

   for( i=ordenf-1; i>=0; i--) {
         denom = cMult(omega,denom);        //se obtienen los polos
         denom.x = denom.x + coef[i];
   }
   funcionTransfer = cDiv( numer, denom);
   fasef = arg(funcionTransfer);                  //calcula la primera fase

   denom = cmplx( coef[ordenf], 0.0);             //se reinicializa omega
   omegaPlus = cmplx(0.0, frecuenciaf + delta);

   for( i=ordenf-1; i>=0; i--) {
         denom = cMult(omegaPlus,denom);       //la multiplicación de num.
                //complejos
         denom.x = denom.x + coef[i];         // los guarda en un apuntador
   }
   funcionTransfer = cDiv( numer, denom);       // calcula la nueva funcion de
                                                // Transferencia
   fasef2 = arg(funcionTransfer);               // calcula la fase de la nueva
                                                // función  de transferencia
   *retardoGrupo = (fasef - fasef2)/delta;      // calcula el grupo de retardo
   return;
}
```

CAPITULO II

SEÑALES EN TIEMPO DISCRETO

SEÑALES Y SISTEMAS EN TIEMPO DISCRETO

En este capítulo se hará referencia a las señales discretas en tiempo, usando la función de muestreo unitario (función impulso Delta de Kronecker), que sustituye a la función impulso Delta de Dirac. Esta se muestra en la figura 2.1

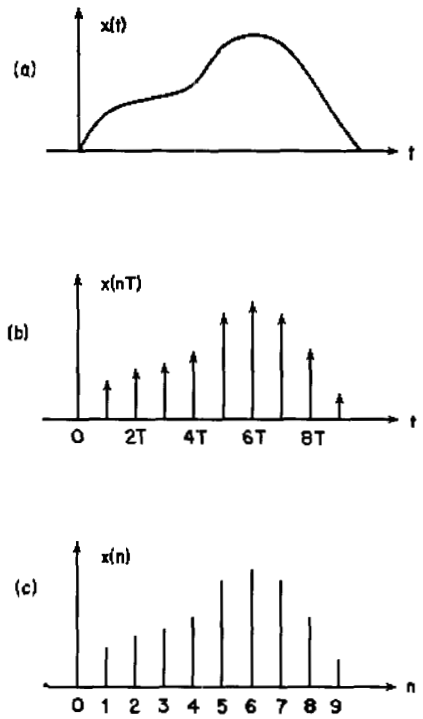

Figura 2.1 Muestreo con función impulso Delta.
(a) señal continua, (b) muestreo con impulso Delta de Dirac
(c) muestreo con impulso Kronecker.

Donde la variable independiente es un valor entero denotado por n, el cual representa a los instantes discretos, donde las muestras pueden ocurrir, y están espaciadas un tiempo discreto $T = 1$.

Nota: La notación que se usará para representar funciones discretas en tiempo, en éste proyecto, es la siguiente:

$$x_k \cong x(kT)$$
$$H_n \cong H(e^{jn\theta})$$
$$\phi_m \cong \phi(mF)$$

2.1 SISTEMAS DE TIEMPO DISCRETO

2.1.1 ECUACION DE DIFERENCIA

Con las ecuaciones de diferencias es más fácil trabajar los sistemas discretos en tiempo. Un sistema discreto, lineal e invariante en el tiempo que acepta una secuencia de entrada $x[n]$, produce una secuencia de salida $y[n]$ que puede ser descrita por una ecuación de diferencias lineal de la forma siguiente:

$$\begin{aligned} &y[n]+a_1y[n-1]+a_2y[n-2]+\cdots+a_ky[n-k] \\ &\quad =b_0x[n]+b_1x[n-1]+b_2x[n-2]+\cdots+b_kx[n-k] \end{aligned} \qquad (2.1)$$

Sin embargo, en ocasiones los coeficientes $a_1, a_2, ..., a_k$ son iguales a cero, por lo que la ecuación se reduce a:

$$y[n]=b_0x[n]+b_1x[n-1]+b_2x[n-2]+\cdots+b_kx[n-k] \qquad (2.2)$$

Podemos ejemplificar el uso de esta ecuación no - recursiva para el caso de un sistema de promediación simple , en el que la salida $n=i$ es igual al promedio aritmético de cinco entradas, desde $n=i-4$ hasta $n=i$. La ecuación de diferencias es como sigue:

$$\begin{aligned} y[n]&=\frac{x[n]+x[n-1]+x[n-2]+x[n-3]+x[n-4]}{5} \\ &=0.2x[n]+0.2x[n-1]+0.2x[n-2]+0.2x[n-3]+0.2x[n-4] \end{aligned} \qquad (2.3)$$

Relacionando esta ecuación con la forma de la ecuación no-recursiva tenemos $k=4$, $a_i=0$ para toda i y $b_0=b_1=b_2=b_3=b_4=0.2$.

2.1.2 CONVOLUCION DISCRETA

En los sistemas discretos en tiempo, la respuesta impulsiva es la respuesta producida cuando una función de muestra unitaria es aplicada a la entrada de un sistema en reposo. De igual forma que con sistemas continuos, podemos obtener $y[n]$ mediante una convolución discreta de la señal $x[n]$ y la respuesta al impulso $h[n]$. Esta convolución discreta esta dada por:

$$y[n]=\sum_{m=0}^{\infty}h[m]x[n-m] \qquad (2.4)$$

Para el caso del sistema de promediación móvil presentado antes, la respuesta impulsiva, $h[n]$, se obtiene directamente al evaluar la ecuación $y[n]$, ante la presencia de una $x[n]$ igual a una función de muestreo unitaria.

$$h[n]=y[n] \ para \ x[n]=\begin{cases}1 & n=0\\ 0 & n\neq 0\end{cases} \qquad (2.5)$$

así $h[n]=\begin{cases}0.2 & 0\le n\le 4\\ 0 & otro\ caso\end{cases}$

Las ecuaciones de diferencias se pueden representar mediante diagramas que gráficamente describan las operaciones involucradas (suma, retardo y multiplicación). Por ejemplo, para la ecuación

$$y[k]=\frac{1}{3}x[k]+\frac{1}{3}x[k-1]+\frac{1}{3}x[k-2] \qquad (2.6)$$

el diagrama de bloques resulta como se muestra en la figura 2.2

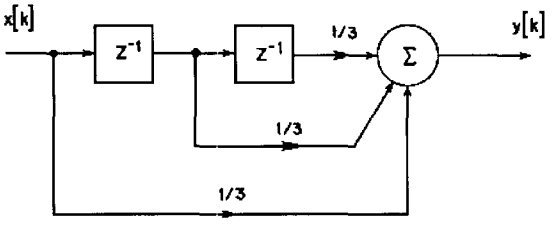

Figura 2.2 Diagrama a bloques

2.2 TRANSFORMADA DE FOURIER DISCRETA EN TIEMPO

La serie de Fourier de una función es expresada de la siguiente manera:

$$x(t)=\sum_{n=-\infty}^{\infty}X[n]e^{j2\pi nFt} \qquad (2.7)$$

donde $F=\frac{1}{t_0}$ es el espacio entre las muestras en el dominio de la frecuencia

con $X[n]=\frac{1}{t_0}\int_{t_0}x(t)e^{-jn2\pi Ft}dt$, y $t_0=$ periodo de $x(t)$. (2.8)

En realidad la señal $x(t)$ y la secuencia $X[n]$ forman un par transformado, con un intervalo de muestreo en el dominio de la frecuencia de F. Así :

$$x(t)\xleftrightarrow{FS;F}X[n] \qquad (2.9)$$

Una vez definida una secuencia en tiempo discreto $X[n]$, la trasformada discreta de Fourier en tiempo (DTFT, Discrete-time Fourier Transform) es usada para obtener directamente el espectro correspondiente de una secuencia, sin tener que recurrir a impulsos y análisis de Fourier en tiempo continúo. La transformación de Fourier discreta en tiempo esta definida como:

$$X(e^{j\omega T})=\sum_{n=-\infty}^{\infty}x[n]e^{-j\omega nT} \qquad (2.10)$$

y la transformada inversa por:

$$x[n]=\frac{1}{2\pi}\int_{-\pi}^{\pi}X(e^{j\omega})e^{j\omega nT}d\omega \qquad (2.11)$$

RELACION CON LAS SERIES DE FOURIER

Existe una dualidad entre la serie de Fourier y la transformada de Fourier discreta en tiempo, específicamente:

$$f[k] \xleftrightarrow{TFDT} F(e^{j\omega T}) \qquad (2.12)$$

Si hacemos que

$$\omega_0 = T$$

$$x(t) = F(e^{j\omega T})\Big|_{\omega=T}$$

$$X[n] = f[k]_{k=-n}$$

entonces $x(t) \xleftrightarrow{FS;\omega_0} X[n]$ (2.13)

Como primer ejemplo de la Transformada de Discreta de Fourier tenemos que para un número de puntos N igual a 10 se obtienen los siguientes resultados.

```
                Tranformada Discreta de Fourier I

¿Cúal es el número de puntos (N)? 10

Dame el valor de x[0]=  1
Dame el valor de x[1]=  2
Dame el valor de x[2]=  3
Dame el valor de x[3]=  4
Dame el valor de x[4]=  5
Dame el valor de x[5]=  6
Dame el valor de x[6]=  7
Dame el valor de x[7]=  8
Dame el valor de x[8]=  9
Dame el valor de x[9]=  10


  oprima una tecla para continuar ... _


    Los datos de la transformada son:

xx[0]= 55.000000 +j 0.000000
xx[1]= -5.000000 +j 15.388418
xx[2]= -5.000000 +j 6.881910
xx[3]= -5.000000 +j 3.632713
xx[4]= -5.000000 +j 1.624598
xx[5]= -5.000000 +j -0.000000
xx[6]= -5.000000 +j -1.624598
xx[7]= -5.000000 +j -3.632713
xx[8]= -5.000000 +j -6.881910
xx[9]= -5.000000 +j -15.388418


oprima una tecla para continuar ...
```

2.3 TRANSFORMADA DISCRETA DE FOURIER

La serie de Fourier (SF) vincula el dominio de tiempo continúo con el dominio de la frecuencia discreta, y la transformada de Fourier (TF) liga el dominio de tiempo continúo con el dominio de la frecuencia continúa. La transformada de Fourier discreta en tiempo (TFDT) liga el dominio en tiempo discreto con el dominio de la frecuencia continúa, por otro lado la transformada discreta de Fourier (TDF) liga el dominio en tiempo discreto con el dominio de la frecuencia discreta, la cual resulta útil para el diseño de filtros digitales.

La transformada discreta de Fourier y su inverso están dados por:

$$X[m]=\sum_{n=0}^{N-1}x[n]e^{-j2\pi mnFT} \quad m=0,1,...,N-1 \quad (2.14) \qquad (2.14)$$

$$=\sum_{n=0}^{N-1}x[n]\cos(2\pi mnFT)+j\sum_{n=0}^{N-1}x[n]sen(2\pi mnFT) \qquad (2.14b)$$

$$x[n]=\sum_{m=0}^{N-1}X[m]e^{j2\pi mnFT} \quad n=0,1,...,N-1 \qquad (2.15)$$

$$=\sum_{m=0}^{N-1}X[m]\cos(2\pi mnFT)+j\sum_{m=0}^{N-1}X[m]sen(2\pi mnFT) \qquad (2.15b)$$

haciendo un cambio de variable para estas dos últimas ecuaciones tenemos:

$$X[m]=\sum_{n=0}^{N-1}x[n]W_N^{-mn} \qquad (2.16)$$

$$x[n]=\sum_{m=0}^{N-1}X[m]W_N^{mn} \qquad (2.17)$$

Las cuales, usualmente, se utilizan para el análisis y síntesis en el Procesamiento digital de Señales.

Para implementar la Transformada Discreta de Fourier (TDF), deben ser elegidos los valores de los parámetros N, T y F, dónde N es el número de intervalos de tiempo de $x[n]$ sobre el cual la sumatoria de la TDF es realizada para calcular la secuencia en frecuencia. En cualquier caso, las sumatorias consideran el calculo de N puntos para $x[n]$ ó $X[m]$, y a estas se les puede considerar, dentro de un programa, como arreglo de entrada o salida según se trate de la transformada directa o inversa. El parámetro T es el intervalo de tiempo entre dos muestras consecutivas en la secuencia de tiempo, en tanto que F es el intervalo de frecuencia entre dos muestras consecutivas en la secuencia de frecuencia. La selección de los parámetros N, F y T está sujeta a las siguientes reglas, que son una consecuencia del teorema de muestreo y las propiedades inherentes de la TDF:

1. La TDF requiere que $NFT=1$.

2. El intervalo de tiempo debe ser tal que $T \le 1/(2f_{max})$, dónde f_{max} es la frecuencia máxima de la señal en tiempo.

3 La longitud del registro en tiempo es igual a NT ó $1/F$.

4. Los algoritmos rápidos de la TDF requieren que N sea una potencia de 2.

Considere el caso de una señal con F_{max} de $300Hz$, y para la cual se requiere una resolución de $5Hz$. en el dominio transformado. Los valores de los parámetros N, T y F, se pueden calcular conociendo que $T \leq 1.66ms$ (aplicando el teorema de muestreo); y como $NTF = 1$, entonces $N \geq 125$, valor que se ajusta a 128 (dado que N debe ser una potencia de 2). Con esto se hace necesario ajustar el valor de F, que resulta en $F = 4.883Hz$, y por lo que, finalmente, se obtiene un ancho en el registro de tiempo igual a $NT = 204.8ms$.

2.3.1 PERIODICIDAD

Como ya sabemos, una función periódica en el tiempo tendrá un espectro discreto en frecuencia, y una función discreta en tiempo tendrá un espectro periódico continuo. La TDF relaciona una función discreta en tiempo con la función discreta en frecuencia correspondiente, esto significa que ambas funciones, la función en el tiempo y la función en frecuencia, son periódicas y discretas. Por esta razón se debe tener cuidado al seleccionar los parámetros de la transformación y al interpretar los resultados de la misma. En base a la periodicidad inherente de las TDF, en la práctica es común observar que los puntos que van de $n = 1$ hasta $n = N/2$ se consideran sobre el eje positivo y los puntos que van de $n = N/2$ hasta $n = N-1$ son reflejo de los puntos sobre el eje negativo en el intervalo que va de $n = -N/2$ a $n = -1$. Esta convención permite redefinir el concepto de sucesiones pares e impares de la siguiente manera: Si $x[N-n] = x[n]$, entonces $x[n]$ es simétrico par, y si $x[N-n] = -x[n]$, entonces $x[n]$ es simétrico impar o asimétrico.

A continuación presentamos un ejemplo de la Transformada Discreta de Fourier que en el programa presentamos como DFT2 donde N el numero de puntos es igual a 10 y arroja los siguientes resultados.

```
                Tranformada Discreta de Fourier II

¿Cúal es el número de puntos (N)?    10

Dame el valor de x[0]=  1
Dame el valor de x[1]=  2
Dame el valor de x[2]=  3
Dame el valor de x[3]=  4
Dame el valor de x[4]=  5
Dame el valor de x[5]=  6
Dame el valor de x[6]=  7
Dame el valor de x[7]=  8
Dame el valor de x[8]=  9
Dame el valor de x[9]=  10


  oprima una tecla para continuar ...
```

```
    Los datos de la transformada son:

xx[0]= 55.000000 +j 0.000000
xx[1]= -5.000000 +j 15.388418
xx[2]= -5.000000 +j 6.881910
xx[3]= -5.000000 +j 3.632713
xx[4]= -5.000000 +j 1.624598
xx[5]= -5.000000 +j -0.000000
xx[6]= -5.000000 +j -1.624598
xx[7]= -5.000000 +j -3.632713
xx[8]= -5.000000 +j -6.881910
xx[9]= -5.000000 +j -15.388418


oprima una tecla para continuar ...
```

2.3.2 PROPIEDADES DE LA TRANSFORMADA DISCRETA DE FOURIER

La Transformada discreta de Fourier tiene propiedades útiles que son parecidas a las propiedades de la transforma de Fourier continúa, a saber:

- **Linealidad**

Si

$$x[n] \underset{IDFT}{\overset{TDF}{\longleftrightarrow}} X[m] \tag{2.18a}$$

Entonces

$$aX[n] \underset{TDFI}{\overset{TDF}{\longleftrightarrow}} aX[m] \tag{2.18b}$$

y si

$$x[n]+y[n] \underset{TDFI}{\overset{TDF}{\longleftrightarrow}} X[m]+Y[m] \tag{2.18c}$$

entonces

$$ax[n]+by[n] \underset{TDFI}{\overset{TDF}{\longleftrightarrow}} aX[m]+bY[m] \tag{2.18d}$$

- **Simetría**

Sí

$$x[n] \underset{TDFI}{\overset{TDF}{\longleftrightarrow}} X[m] \tag{2.19a}$$

entonces

$$\frac{1}{N} X[n] \underset{TDFI}{\overset{TDF}{\longleftrightarrow}} x[-m] \tag{2.19b}$$

- **Desplazamiento en el tiempo**

Si

$$x[n] \underset{TDFI}{\overset{DTF}{\longleftrightarrow}} X[m] \tag{2.20a}$$

entonces

$$x[n-k] \underset{TDFI}{\overset{TDF}{\longleftrightarrow}} X[m]e^{-j2\pi mk/N} \tag{2.20b}$$

- **Desplazamiento en la frecuencia**

Sí

$$x[n] \underset{TDFI}{\overset{TDF}{\longleftrightarrow}} X[m] \tag{2.21a}$$

entonces

$$x[n]e^{j2\pi mk/N} \underset{TDFI}{\overset{TDF}{\longleftrightarrow}} X[m-k] \tag{2.21b}$$

- **Simetría Par e Impar**

Si $x[n]$ es par, entonces $X[m]$ es un valor real y par.

$$x[-n]=x[n]\Leftrightarrow X[m]=X_R[m]=X_R[-m] \qquad (2.22)$$

Si $x[n]$ es impar, entonces $X[m]$ es imaginario e impar.

$$x[-n]=-x[n]\Leftrightarrow X[m]=jX_I[m]=-jX_I[-m] \qquad (2.23)$$

- **Propiedades Real e Imaginario**

Dada una secuencia en el tiempo $x[n]=x_R[n]+jx_I[n]$ y su correspondiente secuencia en frecuencia $X[m]=X_R[m]+jX_I[m]$, entonces:

$$x[n]=x_R[n]\Leftrightarrow X_R[m]=X_R[-m] \qquad X_I[m]=-X_I[-m] \qquad (2.24)$$

$$x[n]=jx_I[n]\Leftrightarrow X_R[m]=-X_R[-m] \qquad X_I[m]=X_I[-m] \qquad (2.25)$$

2.3.3 APLICACIÓN DE LA TRANSFORMADA RAPIDA DE FOURIER.

La rutina en C que utilizamos para ejemplificar la transformada discreta de Fourier suele ser ineficiente, debido a que las funciones seno y coseno son ejecutadas, cada una, N^2 veces para obtener un punto de la TDF. Sin embargo, considerando la siguiente expresión:

$$e^{\left(\frac{-2\pi jk}{N}\right)}=e^{\left[\frac{-2\pi j(k \bmod N)}{N}\right]} \qquad (2.26a)$$

podemos decir que sólo hay N valores diferentes de **phi.** Es posible intercambiar espacio por velocidad calculando previamente los valores de **sen(phi)** y **cos(phi)** para **phi** $=0,1,...,N-1$, lo cual hacemos en el programa **dft2( )**.

Para entender la operación del programa **dft2( ),** consideremos el caso dónde $N=8$. El cálculo de **sumRe** involucra el producto de **x[n].Re** y **cosVAL[k]** para $n=0,1,...,N-1$; por otra parte, para cualquier valor dado de n, el valor de k es determinado por el valor de m usando la siguiente ecuación:

$$k=mn \bmod uloN \qquad (2.26b)$$

Para nuestro caso particular existen 64 combinaciones posibles de (m,n) y sólo 8 valores posibles de k. En la siguiente tabla (tabla 2.1) se muestran los valores de k como función de (m,n) de una TDF con 8 puntos.

N	K(0,n)	K(1,n)	K(2,n)	K(3,n)	K(4,n)	K(5,n)	K(6,n)	K(7,n)
0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7
2	0	2	4	6	0	2	4	6
3	0	3	6	1	4	7	2	5
4	0	4	0	4	0	4	0	4
5	0	5	2	7	4	1	6	3
6	0	6	4	2	0	6	4	2
7	0	7	6	5	4	3	2	1

Tabla 2.1 Valores de k como función de (m, n) para 8 puntos de la TDF

Para $N=8$ la función **dft2( )** calcula el producto ***x*[0].Re*cosVal[0]** 8 veces para cada valor de m diferente , del mismo modo el producto ***x*[4].Re*cosVal[0]** es calculado cuatro veces, una vez por cada valor impar de m. Para evitar la repetición de operaciones para encontrar la TDF, se han desarrollado una gran variedad de algoritmos rápidos de TDF, con el propósito de reordenar y reagrupar su cálculo, y así lograr minimizar y/o eliminar la innecesaria tarea de realizar cálculos múltiples de un mismo producto., dando origen a la transformada rápida de Fourier (TRF).

En la tabla 2.2 se muestra la expansión de las ecuaciones calculadas desde $X(0)$ hasta $X(7)$ para una TDF de 8 puntos.

$X(0)=x(0)W^0+x(1)W^0+x(2)W^0+x(3)W^0+x(4)W^0+x(5)W^0+x(6)W^0+x(7)W^0$
$X(1)=x(0)W^0+x(1)W^1+x(2)W^2+x(3)W^3+x(4)W^4+x(5)W^5+x(6)W^6+x(7)W^7$
$X(2)=x(0)W^0+x(1)W^2+x(2)W^4+x(3)W^6+x(4)W^8+x(5)W^{10}+x(6)W^{12}+x(7)W^{14}$
$X(3)=x(0)W^0+x(1)W^3+x(2)W^6+x(3)W^9+x(4)W^{12}+x(5)W^{15}+x(6)W^{18}+x(7)W^{21}$
$X(4)=x(0)W^0+x(1)W^4+x(2)W^8+x(3)W^{12}+x(4)W^{16}+x(5)W^{20}+x(6)W^{24}+x(7)W^{28}$
$X(5)=x(0)W^0+x(1)W^5+x(2)W^{10}+x(3)W^{15}+x(4)W^{20}+x(5)W^{25}+x(6)W^{30}+x(7)W^{35}$
$X(6)=x(0)W^0+x(1)W^6+x(2)W^{12}+x(3)W^{18}+x(4)W^{24}+x(5)W^{30}+x(6)W^{36}+x(7)W^{42}$
$X(7)=x(0)W^0+x(1)W^7+x(2)W^{14}+x(3)W^{21}+x(4)W^{28}+x(5)W^{35}+x(6)W^{42}+x(7)W^{49}$

Tabla 2.2

Implementando la ecuación 2.5 y aplicando las propiedades conmutativa, asociativa y distributiva de la suma y la multiplicación, las ecuaciones mostradas en la tabla 2.2 pueden ser reescritas como se muestran en la tabla 2.3.

$X(0)=\{[x(0)+x(4)W^0]+W^0[x(2)+x(6)W^0]\}+W^0\{[x(1)+x(5)W^0]+W^0[x(3)+x(7)W^0]\}$
$X(1)=\{[x(0)+x(4)W^4]+W^2[x(2)+x(6)W^4]\}+W^1\{[x(1)+x(5)W^4]+W^2[x(3)+x(7)W^4]\}$
$X(2)=\{[x(0)+x(4)W^0]+W^4[x(2)+x(6)W^0]\}+W^2\{[x(1)+x(5)W^0]+W^4[x(3)+x(7)W^0]\}$
$X(3)=\{[x(0)+x(4)W^4]+W^6[x(2)+x(6)W^4]\}+W^3\{[x(1)+x(5)W^4]+W^6[x(3)+x(7)W^4]\}$
$X(4)=\{[x(0)+x(4)W^0]+W^0[x(2)+x(6)W^0]\}+W^4\{[x(1)+x(5)W^0]+W^0[x(3)+x(7)W^0]\}$
$X(5)=\{[x(0)+x(4)W^4]+W^2[x(2)+x(6)W^4]\}+W^5\{[x(1)+x(5)W^4]+W^2[x(3)+x(7)W^4]\}$
$X(6)=\{[x(0)+x(4)W^0]+W^4[x(2)+x(6)W^0]\}+W^6\{[x(1)+x(5)W^0]+W^4[x(3)+x(7)W^0]\}$
$X(7)=\{[x(0)+x(4)W^4]+W^6[x(2)+x(6)W^4]\}+W^7\{[x(1)+x(5)W^4]+W^6[x(3)+x(7)W^4]\}$

Tabla 2.3

De estas ecuaciones, se observa que tienen varios términos en común, los cuales pueden ser calculados una vez y utilizarlos cuantas veces se requiera, sin necesidad de volverlos a calcular. Para entender mejor el uso de los términos en común, las ecuaciones son presentadas en un diagrama de flujo de la señal., el cual se muestra en la figura 2.3.

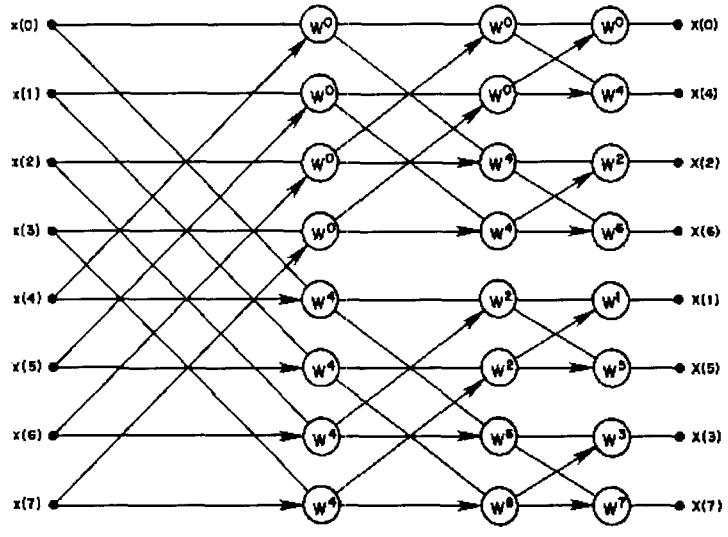

Figura 2.3 Diagrama de flujo que representa las ecuaciones de la tabla 2.3

El diagrama anterior puede ser extendido para cualquier valor de N que sea un valor entero y potencia de 2. El listado de la rutina en C que implementa esta técnica se llama **fft( )**, y se muestra al igual que los anteriores al final de este capítulo.

Como un ejemplo de la Transformada Rápida de Fourier tenemos para N igual a 10 los siguientes resultados.

```
                TRANSFORMADA DISCRETA DE FOURIER

¿Cúal es el número de puntos (N)? 10

Dame el valor de x[0]=  1
Dame el valor de x[1]=  2
Dame el valor de x[2]=  3
Dame el valor de x[3]=  4
Dame el valor de x[4]=  5
Dame el valor de x[5]=  6
Dame el valor de x[6]=  7
Dame el valor de x[7]=  8
Dame el valor de x[8]=  9
Dame el valor de x[9]=  10


  oprima una tecla para continuar ... _
```

```
Los datos de la transformada son:

xx[0]= -3.281153 +j -5.290067
xx[1]= 1.364745 +j 6.379881
xx[2]= -9.635255 +j -7.383938
xx[3]= 18.173762 +j 0.085757
xx[4]= -3.090170 +j -5.877853
xx[5]= 19.354102 +j 3.889983
xx[6]= 3.819660 +j -1.175571
xx[7]= 21.527864 +j 3.355198
xx[8]= -5.000000 +j -6.881910
xx[9]= -5.000000 +j -15.388418


oprima una tecla para continuar ... _
```

2.3.4 APLICACION DE LA TRANSFORMADA DISCRETA DE FOURIER

- **Señales limitadas en tiempo**

Considere una señal continúa limitada en tiempo y su espectro continúo como se muestra en las figuras. 2.4 a y 2.4b, respectivamente, y recuérdese que una señal no puede ser limitada en tiempo y en banda a la vez. Al muestrear esta señal se produce la secuencia en tiempo mostrada en la figura 2.4c necesaria para utilizar la TDF; si la longitud N del registro de entrada se elige para ser más grande que la longitud de la secuencia de tiempo de entrada, la secuencia completa se podrá ajustar dentro del registro de entrada, como se muestra. La TDF tratará la secuencia de entrada como si fuera la secuencia periódica mostrada en la figura 2.4d , de lo cual resultara un espectro de frecuencia discreto periódico, como se ve en la figura 2.4e. La salida producida por el algoritmo será la secuencia de valores desde $m=0$ hasta $m=N-1$, teniendo que soportar algo de aliasing debido a la naturaleza limitada en tiempo (y consecuentemente al ancho de banda ilimitado) de la señal de entrada.

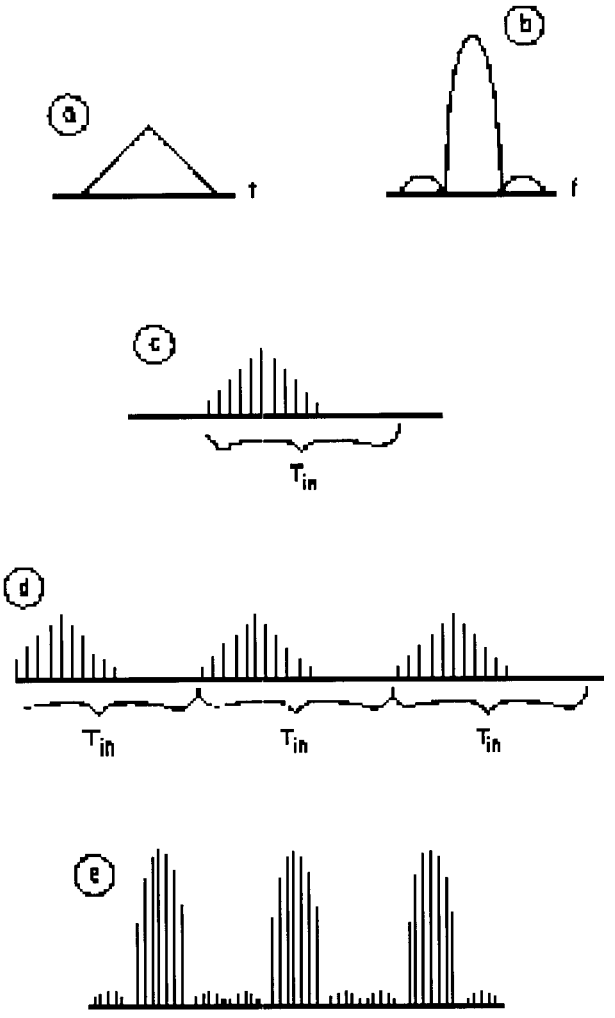

Figura 2.4 Señales y secuencias para la TDF de una señal limitada en tiempo

Señales Periódicas

Si consideramos una señal periódica en tiempo continúo, limitada en banda, y su espectro, como se muestra en la figura 2.5:

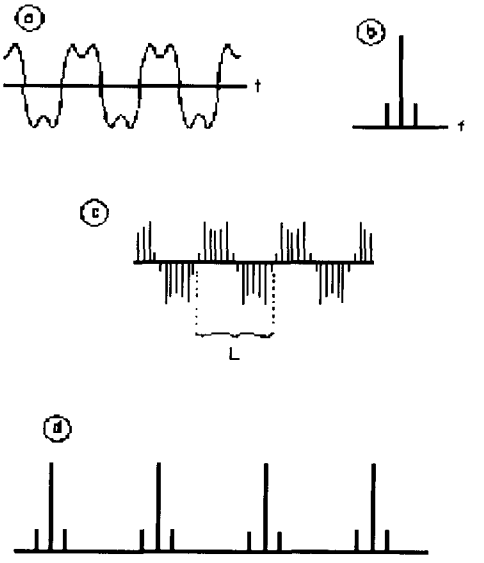

Figura 2.5 Señal y secuencias para la TDF de una señal periódica. La longitud L del registro de entrada de la TDF equivale al periodo de la señal.

podemos muestrear esta señal para producir una secuencia como se muestra en la figura 2.5c, para la entrada a la TDF. Si la longitud N del registro de entrada se elige para ser exactamente igual a la longitud de un período de esta secuencia, la suposición de periodicidad implícita en la TDF causará que ésta trate al registro de entrada como si fuera una secuencia completa. El correspondiente espectro periódico de frecuencia discreta es mostrado en la figura 2.5d. La secuencia de salida de la TDF consiste en un período, que es exactamente el espectro de la figura 2.5b.

Señales no periódicas largas

Existen aplicaciones de procesamiento de señales, dónde frecuentemente la señal analizada no es periódica ni limitada en tiempo; y donde la secuencia correspondiente a los valores de la señal digitalizada es más larga que el registro de entrada de la TDF, y por lo que la secuencia tendrá que ser truncada a las N muestras antes de que la TDF pueda ser aplicada. La naturaleza periódica de la TDF causará que la secuencia truncada, como se muestra en la figura 2.6b, se interprete como se observa en la figura 2.6c, la cual exhibe una discontinuidad grande en los puntos extremos del registro. Esto producirá componentes de alta frecuencia dentro del espectro producido por la TDF; a este fenómeno se le conoce como efecto de fuga (leakage). Para reducir los efectos de leakage, en la práctica, es común multiplicar la secuencia de entrada truncada por una ventana, antes de aplicar la TDF.

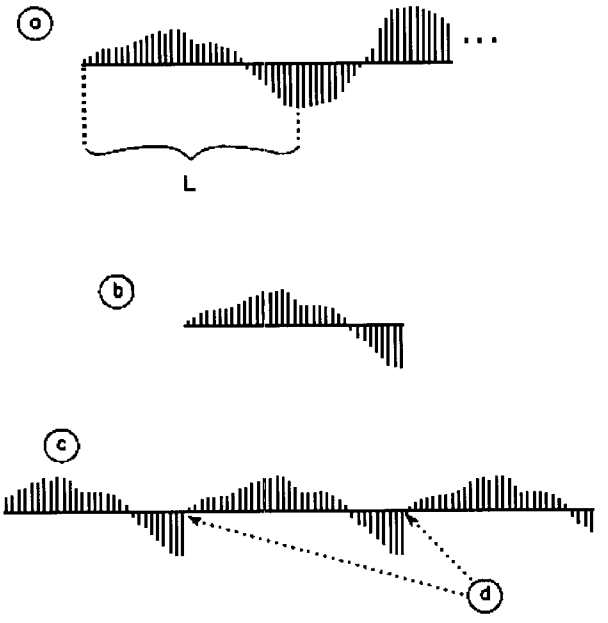

Figura 2.6 Discontinuidad causada por el truncamiento de la secuencia de entrada de la TDF. (a) longitud de la secuencia de entrada (b) secuencia de entrada truncada , (c) secuencia de entrada interpretada por la TDF , y (d) discontinuidades resultantes.

2.4 TRANSFORMADA Z

Relaciones entre las transformadas Z y Laplace

La transformada z es la expresión discreta de la transformada de Laplace y, usualmente, se aplica a secuencias que pueden ser obtenidas por el muestreo de una función de tiempo continúo. Específicamente para una secuencia causal, como la siguiente:

$$x[n]=\sum_{n=0}^{\infty} x_a(nT)\delta(t-nT) \qquad (2.27)$$

La transformada de Laplace esta dado por:

$$X_a(s)=\sum_{n=0}^{\infty} x_a(nT)e^{-nTs} \qquad (2.28)$$

Si sustituimos $z=e^{sT}$ y $x[n]=x_a(nT)$ en la ecuación anterior, se obtiene la transformada z, definida por:

$$X(z)=\sum_{n=0}^{\infty} x[n]z^{-n} \qquad (2.29)$$

La relación entre los planos s y z es regulada por la expresión:

$$z=e^{sT}=e^{\sigma T}e^{j\omega T}=e^{\sigma T}(\cos\omega T+j\,\mathrm{sen}\,\omega T) \qquad (2.30)$$

Dado que $\left|e^{j\omega T}\right|=\left(\cos^2\omega T+\mathrm{sen}^2\,\omega T\right)^{1/2}=1$ y $T>0$, podemos concluir que $|z|<1$ para $\sigma<0$. En otras palabras, la mitad izquierda del plano s se mapea en el interior del círculo unitario en el plazo z. Del mismo modo, para $|z|=1$ y $\sigma=0$, el eje $j\omega$ del plano s se mapea sobre el círculo unitario en el plano z.

2.4.1 REGION DE CONVERGENCIA

Para que una secuencia quede adecuada y totalmente definida se debe establecer su región de convergencia, es decir, el conjunto de valores que la hacen analítica, de modo de evitar incertidumbres. El conocimiento de $X(z)$ y su región de convergencia determinan de manera única la secuencia correspondiente $x[n]$; sin embargo, es de notar que en el caso de secuencias unilaterales se puede prescindir de la región de convergencia para especificar a $x[n]$ de modo único.

Considérese el caso de una secuencia de longitud finita que existe dentro del intervalo (a, b) y para la cual la representación en el dominio de z es:

$$X(z)=x_a z^{-a}+x_{a+1}z^{-(a+1)}+\ldots+x_b z^{-b} \qquad (2.31)$$

que es un polinomio en z que converge absolutamente para todo valor de z, excepto para $z=\infty$ si $a<0$.

Para el caso de secuencias semiinfinitas, donde el intervalo de la secuencia es (a,∞) la $x(z)$ converge para toda z tal que $R_- < |z|$, excepto para $z=\infty$ si $a<0$. Por otro lado, para el caso de secuencias infinitas donde $x(z)$ posee potencias negativas y positivas de z, la región de convergencia es: $R_- < |z| < R_+$, con R_- (radio de convergencia interior) determinado por el comportamiento de $x[n]$ para $n>0$ y R_+ por el comportamiento de $x[n]$ para $n<0$.

2.4.2 FUNCION DE TRANSFERENCIA

La transformada z de la respuesta de un sistema discreto en tiempo, a una secuencia de muestreo unitaria, juega un papel similar en el análisis de sistemas al de la transformada de Laplace respecto de la respuesta al impulso del sistema continúo en tiempo; de ahí que se siga denominando como función de transferencia del sistema, y es denotada por $H(z)$. La función de transferencia puede ser derivada de la ecuación de diferencias que describe al sistema, a saber:

$$\begin{aligned} &y[n]+a_1 y[n-1]+a_2 y[n-2]+\cdots+a_k y[n-k] \\ &\quad = b_0 x[n]+b_1 x[n-1]+b_2 x[n-2]+\cdots+b_k x[n-k] \end{aligned} \tag{2.32}$$

para obtener

$$\begin{aligned} &Y(z)+a_1 z^{-1}Y(z)+a_2 z^{-2}Y(z)+\cdots+a_k z^{-k}Y(z) \\ &\quad = b_0 X(z)+b_1 z^{-1}X(z)+b_2 z^{-2}X(z)+b_k z^{-k}X(z) \end{aligned} \tag{2.33}$$

Factorizando la salida $Y(z)$ y $X(z)$ y resolviendo para $H(z)=Y(z)/X(z)$ tenemos

$$H(z)=\frac{Y(z)}{X(z)}=\frac{b_0+b_1 z^{-1}+b_2 z^{-2}+b_k z^{-k}}{1+a_1 z^{-1}+a_2 z^{-2}+\cdots+a_k z^{-k}} \tag{2.34}$$

Ecuación que se puede factorizar de modo de exhibir explícitamente a sus polos y ceros, $p_1, p_2, \ldots, p_k$, y $q_1, q_2, \ldots, q_m$, respectivamente.

La transformada z inversa esta dada por la integral de contorno siguiente:

$$x[n]=\frac{1}{2\pi j}\oint_c X(z)z^{n-1}dz \tag{2.35}$$

dónde la trayectoria de contorno es en sentido contrario de las manecillas del reloj, conteniendo la región de convergencia de $X(z)$. Si $X(z)$ es racional, se usa el teorema de residuos para evaluar la ecuación; sin embargo, éste no es usada con frecuencia en la práctica, y en su lugar se opta por usar pares transformados ya establecidos y propiedades de las transformadas.

LISTADO DE PROGRAMAS DE LA TRANSFORMADA DE FOURIER

```
/*                                                                    */
/*                                  | Este programa calcula la Transformada */
/*   Programa 2.1                   | Discreta de Fourier aplicando la   */
/*                                  | definición de la TDF               */
/*                                                                    */
/*   TransformadaDiscretaFourier()  |                                    */
/*                                                                    */
/*                                                                    */
```

```
void dft( struct complex x[], struct complex xx[], int N)
{
   int n, m;
   real sumRe, sumIm, phi;

   for( m=0; m<N; m++) {
           sumRe = 0.0;                        // inicializacion de variables
           sumIm = 0.0;
           for( n=0; n<N; n++) {

                  /* se calcula el angulo phi para aplicar la ecuacion 2.14 */
             phi = 2.0 * PI * m * n /N;

                   sumRe += x[n].x * cos(phi) + x[n].y * sin(phi);
                   sumIm += x[n].y * cos(phi) - x[n].x * sin(phi);
           }
           xx[m] = cmplx(sumRe, sumIm);        // los valores obtenidos se
                                               // almacenan en un arreglo
   }
   return;
}
```

```
/*-------------------------------------------------------------------------*/
/*                          Programa que calcula la Transformada Discreta  */
/*   Programa 2.2           de Fourier  de una forma mas simplificada.     */
/*                                                                         */
/*   dft2()                                                                */
/*                                                                         */
/*-------------------------------------------------------------------------*/

void dft2( struct complex x[],  struct complex xx[],  int N)
{
   int n, m, k;
   real sumRe, sumIm, phi;
   static real cosVal[DFTSIZE], sinVal[DFTSIZE];

   for( k=0; k<N; k++) {              //implementacion de la ecuacion 2.26a
          cosVal[k] = cos(2.0 * PI * k /N);
          sinVal[k] = sin(2.0 * PI * k /N);
   }
   for( m=0; m<N; m++) {
          sumRe = 0.0;                //inicializacion de variables
          sumIm = 0.0;
          for( n=0; n<N; n++) {       // Aplicacion de  la definicion de la TDF
                 k = (m*n)%N;
                 sumRe += x[n].x * cosVal[k] + x[n].y * sinVal[k];
                 sumIm += x[n].y * cosVal[k] - x[n].x * sinVal[k];
          }
          xx[m] = cmplx(sumRe, sumIm);     // los valores obtenidos se guardan en
   }                                              // en un arreglo.
   return;
}
```

```
/*                                                                    */
/*   Programa 2.3                    |  Este programa calcula la Transformada  */
/*                                   |  rapida de Fourier.                     */
/*   TransformadaRapidaFourier()     |  El orden del filtro debe ser entero y  */
/*                                   |  de potencia 2.                         */
/*--------------------------------------------------------------------*/
```

```
void fft(         struct complex xIn[], struct complex xOut[],int N)
{
   static struct complex x[2][DFTSIZE];
   static real cc[DFTSIZE], ss[DFTSIZE];
   int ping, pong, n, L, nSkip, nivel, nG, nB, kt, kb, nBot;

   for( n=0; n<N; n++) {
          x[0][n] = xIn[n];
          cc[n] = cos(2.0 * PI * n /N);              //Ecuacion 2.26a
          ss[n] = sin(2.0 * PI * n /N);
   }
   pong = 0;
   ping = 1;
   L = log2(N);
   nSkip = N;
   for( nivel=1; nivel<=L; nivel++) {
          nSkip /= 2;
          n = 0;
          for(nG=0; nG<ipow(2,(nivel-1)); nG++) {
                 kt = bitRev(L,2*nG);
                 kb = bitRev(L,2*nG+1);
                 for(nB=0; nB<nSkip; nB++) {
                        nBot = n + nSkip;
                        x[ping][n].x = x[pong][n].x + cc[kt] * x[pong][nBot].x
                                     + ss[kt] * x[pong][nBot].y;
                        x[ping][n].y = x[pong][n].y + cc[kt] * x[pong][nBot].y
                                     - ss[kt] * x[pong][nBot].x;
                        x[ping][nBot].x = x[pong][n].x + cc[kb] * x[pong][nBot].x
                                     + ss[kb] * x[pong][nBot].y;
                        x[ping][nBot].y = x[pong][n].y + cc[kb] * x[pong][nBot].y
                                     - ss[kb] * x[pong][nBot].x;
                        n++;
                 }
                 n += nSkip;
          }
        ping = !ping;
        pong = !pong;
   }
   for(n=0; n<N; n++) xOut[bitRev(L,n)] = x[pong][n];
   return;
}
```

CAPITULO III

FILTROS DIGITALES CON RESPUESTA AL IMPULSO FINITO

3.1 FUNDAMENTOS DE FILTROS DIGITALES

Los filtros digitales, en general, son clasificados de acuerdo a la duración de sus respuestas impulsivas, las cuales pueden ser de naturaleza finita o infinita. Los primeros (con respuesta al impulso finito, RIF) son filtros que responden a un impulso unitario, generando una secuencia de duración finita; en tanto que los segundos (con respuesta al impulso infinito, RII) responden al impulso unitario con una secuencia cuya duración es infinita. Cuando se les compara a ambos tipos de filtro (RIF y RII), se pueden observar ventajas y desventajas, de uno con respecto al otro; sin embargo, ninguno de ellos es mejor en todas las situaciones.

3.1.1 FILTROS RIF

La forma general, de la salida, $y[k]$ de un sistema lineal RIF invariante en el tiempo, en el tiempo k, esta dada por:

$$y[k] = \sum_{n=0}^{N-1} h[n]x[k-n] \qquad (3.1)$$

donde $h[n]$ es la respuesta al impulso del sistema. La ecuación permite observar que la salida es una combinación lineal de la entrada presente y de las N entradas previas.

Algunas de las características que hacen interesante el trabajar con filtros RIF son:

- A los filtros RIF se les diseña fácilmente para tener retardos de fase y/o grupo constantes.
- Los filtros RIF, implantados con técnicas no recursivas, siempre son estables, y libres de las oscilaciones características de los filtros RII.
- El ruido por redondeo, característico de la aritmética de precisión finita, puede ser mínimo implementando las técnicas no recursivas.
- Si se quiere, los filtros RIF también pueden implementarse con técnicas recursivas.

Por otro lado, los filtros RIF exhiben algunos inconvenientes, desde el punto de vista del diseño, por ejemplo:

Aunque la duración de la respuesta impulsiva es finita, esta debe ser muy larga para lograr características de corte agudo.

- En situaciones similares, el diseño de filtros RIF es, generalmente, más difícil que el diseño equivalente de filtros RII.

3.1.2 EVALUACION DE LA RESPUESTA DE FRECUENCIA DE FILTROS RIF

La respuesta al impulso de un filtro digital, $h[n]$, esta relacionada con la respuesta en frecuencia a través de la transformada de Fourier discreta en tiempo:

$$H(e^{j\lambda}) = \sum_{n=-\infty}^{\infty} h[n]e^{-jn\lambda} \qquad (3.2)$$

Para un filtro RIF, $h[n]$ es diferente de cero sólo para $0 \le n < N$. Por lo tanto, los límites de la sumatoria pueden cambiarse para obtener:

$$H(e^{j\lambda}) = \sum_{n=0}^{N-1} h[n]e^{-jn\lambda} \qquad (3.3)$$

Esta ecuación se puede evaluar directamente para cualquier valor de λ , con $\lambda = \omega T$. De este modo, el valor de frecuencia ω_m correspondiente al índice m de frecuencia discreta esta dado por:

$$\omega_m = 2\pi mF$$

que al sustituirse en la ecuación deja:

$$H(e^{j\lambda}) = \sum_{n=-\infty}^{\infty} h[n]e^{-jn\lambda} \qquad (3.4)$$

que no es otra cosa que la transformada discreta de Fourier:

$$H[m] = \sum_{n=0}^{N-1} h[n]e^{(-2\pi jnmFT)} \qquad (3.5)$$

Relación que puede ser fácilmente codificada en un programa, para ejecutarse sobre un procesador, haciendo uso de una diversidad de algoritmos, entre los que destaca el denominado transformada rápida de Fourier.

Como ya se ha mencionado, el retardo de grupo constante es una propiedad deseable en los filtros, debido a que el retardo de grupo no constante provoca distorsión de envolvente, sobre las señales moduladas, y distorsión en la forma de los pulsos en la señal digital de banda base.

La respuesta en frecuencia, $H(e^{j\omega})$, de un filtro se puede expresar en términos de la respuesta en amplitud $A(\omega)$ y la respuesta en fase $\theta(\omega)$, como:

$$H(e^{j\omega}) = A(\omega)e^{j\theta(\omega)} \qquad (3.6)$$

Si un filtro tiene una respuesta de fase lineal de la forma:

$$\theta(\omega) = -\alpha\omega \quad \text{donde} \quad -\pi \le \omega \le \pi \qquad (3.7)$$

entonces, tendrá un retardo de fase, τ_p , y un retardo de grupo, τ_g , constantes. En este caso $\tau_p = \tau_g = \alpha$,

pudiendo mostrarse que la respuesta al impulso es:

$$h[n]=\begin{cases} c & n=0 \\ 0 & n\neq 0 \end{cases}$$

Si α es diferente de cero, la ecuación $\theta(\omega)=-\alpha\omega$ se satisface si y sólo si

$$\alpha=\frac{N-1}{2} \qquad (3.8)$$

y

$$h[n]=h[N-1-n] \qquad 0\le n\le N-1 \qquad (3.9)$$

Esta restricción da lugar a dos tipos de filtros, a saber: Tipo I, cuando N es impar y, Tipo II cuando N es par. Para los filtros del primer tipo, el eje de simetría para $h[n]$ cae sobre $n=(N-1)/2$, como se muestra en la figura 3.1; en tanto que para los filtros del segundo tipo el eje de simetría cae entre los puntos $n=N/2$ y $n=(N-2)/2$, como se muestra en la figura 3.2

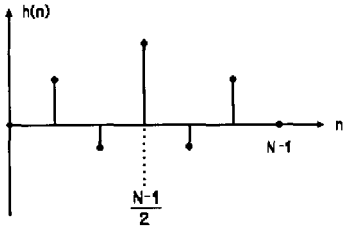

Figura 3.1 Respuesta al impulso para un filtro RIF de fase lineal de tipo 1 mostrando simetría par en n=(N-1)/2

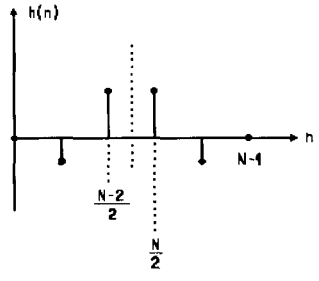

Figura 3.2 Respuesta al impulso para un filtro RIF de fase lineal de tipo 2, mostrando simetría par entre n= (N-2)/2 y N/2

Los filtros pueden tener un retardo de grupo constante sin tener retardo de fase constante, cuando la respuesta de fase es una línea recta que no pasa por el origen, es decir: $\theta(\omega)=\beta+\alpha\omega$, para $-\pi\le\omega\le\pi$. La respuesta en fase de un filtro de este tipo deberá cumplir con las siguientes restricciones:

$$\alpha=\frac{N-1}{2} \qquad (3.10a)$$

$$\beta=\pm\frac{\pi}{2} \qquad (3.10b)$$

$$h[n]=-h[N-1-n] \qquad 0\le n\le N-1 \qquad (3.11)$$

Particularmente, una respuesta al impulso como la que expresa la ecuación anterior se denomina como simétrica impar. Con estas restricciones, es posible identificar otros dos tipos de

filtros, referidos como de Tipo 3 y Tipo 4, que se asumen como filtros de fase lineal a pesar de que, en estricto sentido, su respuesta de fase no es verdaderamente lineal. Los filtros de tipo 3 satisfacen las tres ecuaciones anteriores con N impar, y los filtros de tipo 4 con N par. Para los filtros de tipo 3 el eje de antisimetria para $h(n)$ esta situado en $N=(N-1)/2$, como se muestra en la figura 3.3:

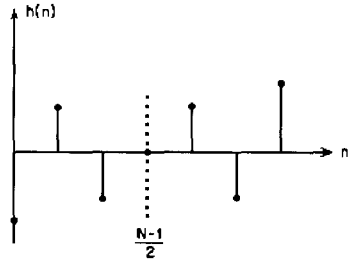

Figura 3.3 Respuesta al impulso para un filtro RIF de fase lineal de tipo 3 mostrando simetría impar en n= (N-1)/2

Cuando $n=(N-1)/2$, con N par, la ecuación para $h(n)$ queda como:

$$h\left[\frac{N-1}{2}\right]=-h\left[\frac{N-1}{2}\right] \qquad (3.12)$$

Por lo tanto, $h\left[{}^{(n-1)}/_{2}\right]$ siempre es cero en filtros de tipo 3. Por otro lado, para los filtros de tipo 4 el eje de antisimetria queda situado entre los puntos $n={}^{N}/_{2}$ y $n={}^{(N-2)}/_{2}$, como se muestra en la figura 3.4

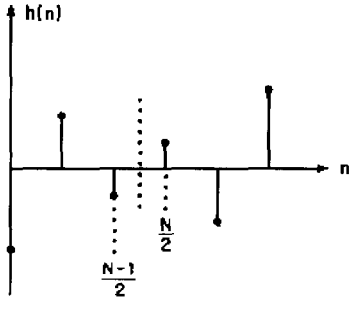

Figura 3.4 Respuesta al impulso para un filtro RIF de fase lineal de tipo 4 mostrando simetría impar entre n= (N-2)/2 y n= N/2.

La transformada de Fourier discreta en tiempo (TFDT) se puede usar para obtener la respuesta en frecuencia de cualquier filtro RIF, al mismo tiempo que se pueden aprovechar las propiedades de

fase lineal y el asumir señales puramente reales, para obtener pares transformados que permitan la rápida representación de las respuestas.

La respuesta en frecuencia $H(e^{j\omega T})$ y la respuesta en amplitud $A(e^{j\omega T})$, para los cuatro tipos de filtros RIF, son listadas en la tabla 3.1, en tanto que sus propiedades están anotadas en la tabla 3.2

$h(nT)$ simetria	N	$H(e^{j\omega T})$	$A(e^{j\omega T})$
Par	Impar	$e^{-j\omega(N-1)T/2}A(e^{j\omega T})$	$\sum_{k=0}^{(N-1)/2} a_k \cos\omega kT$
Par	Par	$e^{-j\omega(N-1)T/2}A(e^{j\omega T})$	$\sum_{k=1}^{N/2} b_k \cos[\omega(k-\frac{1}{2})T]$
Impar	Impar	$e^{-j[\omega(N-1)T/2-\pi/2]}A(e^{j\omega T})$	$\sum_{k=1}^{(N-1)/2} a_k sen\omega kT$
Impar	Par	$e^{-j[\omega(N-1)T/2-\pi/2]}A(e^{j\omega T})$	$\sum_{k=1}^{N/2} b_k sen[\omega(k-\frac{1}{2})T]$

$$a_0 = h\left[\frac{(N-1)T}{2}\right] \qquad a_k = 2h\left[\left(\frac{N-1}{2}-k\right)T\right] \qquad b_k = 2h\left[\left(\frac{N}{2}-k\right)T\right]$$

Tabla 3.1 Formulas de respuesta en frecuencia para filtros de fase lineal RIF

	Tipo			
	1	2	3	4
Longitud N	Impar	Par	Impar	Par
Simetría con respecto a ω=0	Par	Par	Impar	Impar
Simetría con respecto a ω=π	Par	Impar	Impar	Par
periodicidad	2π	4π	2π	4π

Tabla 3.2 Propiedades de los filtros RIF teniendo un retardo de grupo constante.

Como primer ejemplo de diseño de filtros veremos los de respuesta RIF con el siguiente X(n).
X(n) = {3,5,1,2,0,0,0,0}, par con n = 8, con un número de pasos igual al de puntos.

```
                    MENU PRINCIPAL

          1. FILTROS ANALOGICOS
          2. TRANSFORMADA DISCRETA DE FOURIER
          3. DISEÑO DE FILTROS FIR
          4. DISEÑO DE FILTROS IIR
          5. SALIR DEL PROGRAMA

              Elija su OPCION:
```

elige la opción 3

```
              DISEÑO DE FILTROS RIF

         1. RESPUESTA DE FILTROS RIF
         2. POR SERIES DE FOURIER
         3. POR MUESTREO EN FRECUENCIA
         4. SALIR

            Elija su OPCION:  _
```

elige la opción 1 e introduce los datos que se te piden.

```
                RESPUESTA AL IMPULSO FINITO

                  TIPO DE FILTRO

       1. Simetrico  PAR,  longitud IMPAR
       2. Simetrico  PAR,  longitud PAR
       3. Simetrico IMPAR, longitud IMPAR
       4. Simetrico IMPAR, longitud PAR
       5. Regresar al menu

        Elija su opcion:  1
        Numero de pasos: 8
      Numero de puntos: 8_
```

los resultados que arroja el programa son los siguientes:

```
                RESPUESTA AL IMPULSO FINITO

         Dame el valor del punto x[0]= 3
         Dame el valor del punto x[1]= 5
         Dame el valor del punto x[2]= 1
         Dame el valor del punto x[3]= 2
         Dame el valor del punto x[4]= 0
         Dame el valor del punto x[5]= 0
         Dame el valor del punto x[6]= 0
         Dame el valor del punto x[7]= 0

       ¿Desea su resultado en escala de dBs <S/N>:? _
```

```
RESPUESTA AL IMPULSO FINITO

Los resultados son:

Hd[0] =  -6136.466965  8
Hd[1] =  -6136.466965
Hd[2] =  -6136.466965
Hd[3] =  -6136.466965
Hd[4] =  -6136.466965
Hd[5] =  -6136.466965
Hd[6] =  -6136.466965
Hd[7] =  -6136.466965


Oprima una tecla para continuar_
```

3.2 DISEÑO DE FILTROS RIF CON EL METODO DE SERIES DE FOURIER.

El método de series de Fourier para diseñar filtros RIF se basa en el hecho de que la respuesta en frecuencia de un filtro digital es periódica y, por tanto, representable por una serie de Fourier. Por esta razón, resulta útil expandir la respuesta en frecuencia en una serie de Fourier, la cual, por razones operativas, se truncará a un número finito de términos, a partir de los cuales será posible construir la respuesta impulsiva finita que, a su vez, quedará relacionada con los coeficientes del filtro.

Esta forma de pensar el diseño de un filtro RIF puede quedar plasmada en el siguiente algoritmo.

Algoritmo 3.1 Diseño de filtros RIF por medio del método Series de Fourier.

Paso1 Especificar una respuesta en frecuencia $H_d(\lambda)$.

Paso 2 Especificar el número de N puntos (derivaciones) del filtro.

Paso 3 Calcular los coeficientes, $h[n]$, del filtro para $n = 0,1,2,...,N-1$.
con la siguiente formula:

$$h[n] = \frac{1}{2\pi}\int_{2\pi} H_d(\lambda)[\cos(m\lambda) + j\,\mathrm{sen}(m\lambda)]d\lambda \qquad (3.13)$$

donde $m = n - \frac{N-1}{2}$.

Paso 4 Usar las consideraciones de simetría y calcular la respuesta en frecuencia del filtro resultante.
Si la respuesta no es adecuada, cambiar N o $H_d(\lambda)$ y regresar al paso 3.

Podemos usar el algoritmo anterior para diseñar un filtro RIF de 21 derivaciones cuya respuesta en amplitud sea aproximadamente igual a la de un filtro pasabajas ideal con una frecuencia de corte de $2KHz$, y para el cual se utiliza un velocidad de muestreo de $5KHz$. En este caso, la frecuencia de corte normalizada es $\lambda = 2\pi/5$, y la respuesta en frecuencia deseada es como se presenta en la figura 3.5

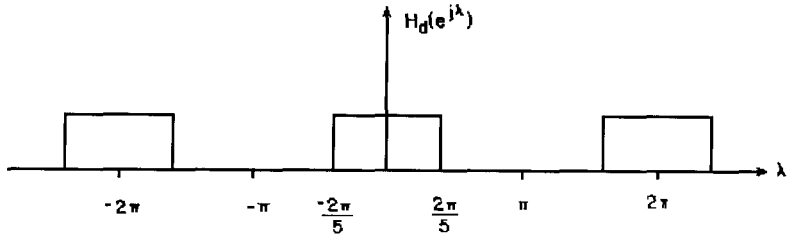

Figura 3.5 Respuesta en frecuencia para el diseño de un filtro RIF de 21 derivaciones.

Usando la ecuación del paso 3 del algoritmo, se obtiene lo siguiente:

$$h[n] = \frac{1}{2\pi}\int_{-2\pi/5}^{2\pi/5}\cos(m\lambda)d\lambda + j\frac{1}{2\pi}\int_{-2\pi/5}^{2\pi/5}\text{sen}(m\lambda)d\lambda$$

Donde, dado que el integrando en la segunda integral es impar y los límites de integración son simétricos, ésta es igual a cero. Por lo tanto:

$$h[n] = \frac{\text{sen}(m\lambda)}{2m\pi}\Big|_{\lambda=-2\pi/5}^{2\pi/5}$$
$$= \frac{\text{sen}(2m\pi/5)}{m\pi}$$

donde $m = n - 10$.

La regla de L'Hospital es usada para evaluar la ecuación anterior, para el caso cuando $m = 0$ (esto es, $n = 10$):

$$h[10] = \frac{(d/dm)\,\text{sen}(2m\pi/5)}{(d/dm)m\pi}\Big|_{m=0}$$
$$= \frac{(2\pi/5)\cos(2m\pi/5)}{\pi}\Big|_{m=0}$$
$$= \frac{2}{5} = 0.4$$

Los valores resultantes de $h[n]$ son listados en la tabla 3.3 y su correspondiente respuesta en magnitud se muestra en las figuras 3.6 y 3.7

h[0]	=	h[20]	=	0.000000
h[1]	=	h[19]	=	0.033637
h[2]	=	h[18]	=	0.023387
h[3]	=	h[17]	=	0.026728
h[4]	=	h[16]	=	0.050455
h[5]	=	h[15]	=	0.000000
h[6]	=	h[14]	=	-0.075683
h[7]	=	h[13]	=	-0.062366
h[8]	=	h[12]	=	0.093549
h[9]	=	h[11]	=	0.302731
		h[10]	=	0.400000

Tabla 3.3 coeficientes de respuesta al impulso Para un filtro pasa bajas de 21 derivaciones.

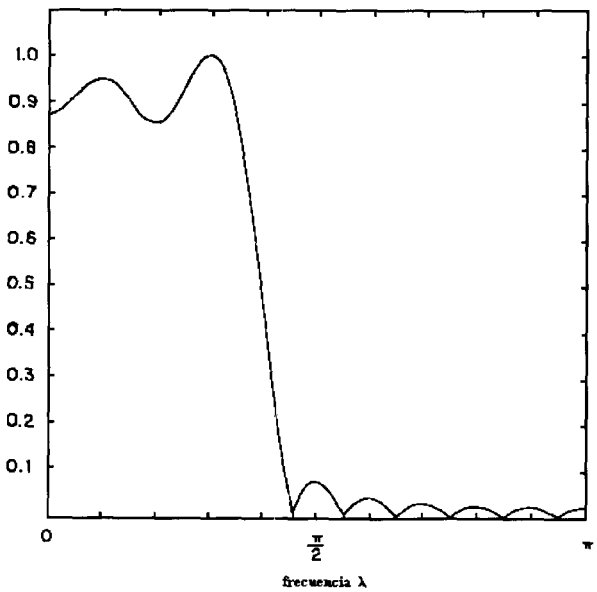

Figura 3.6 Respuesta en magnitud (como un porcentaje de pico) obtenidos de un filtro pasa bajas de 21 derivaciones.

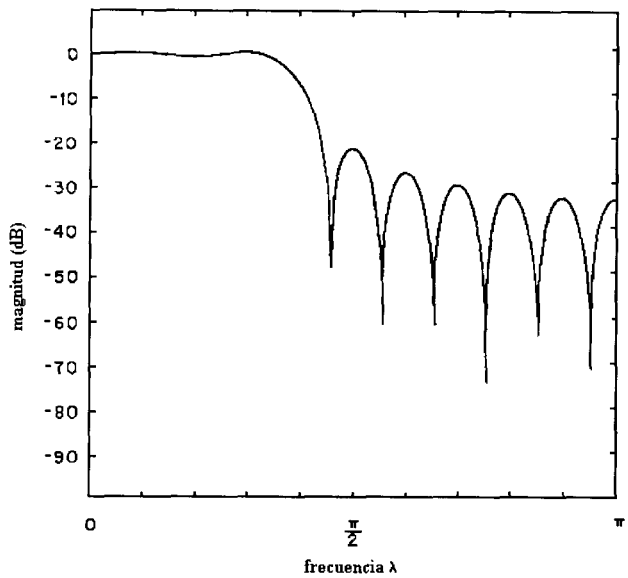

Figura 3.7 Respuesta en magnitud (en decibeles) obtenida para un filtro pasa bajas de 21 derivaciones.

En las figuras anteriores se puede apreciar la intensidad de los rizos en las bandas de paso y de rechazo, que son más pronunciados cuando el eje de referencia vertical se considera en una escala lineal, para el caso de la pasabanda, y cuando el eje esta en una escala logarítmica de *dB's* para el caso de la rechazabanda.

3.2.1 PROPIEDADES DEL METODO POR SERIES DE FOURIER

1. Los filtros diseñados usando el algoritmo 3.1 exhibirán la propiedad de fase lineal, asumido el hecho de que la respuesta en frecuencia sea simétrica o antisimétrica

2. Como consecuencia del fenómeno de Gibbs, la respuesta en frecuencia de los filtros diseñados con el algoritmo 3.1 contendrá saltos bruscos en los limites de la pasabanda, como se muestra en la figura 3.6 y 3.7. Por supuesto, la única manera de minimizar estas distorsiones es aumentar el número de derivaciones del filtro.

3.2.2 APROXIMACION RIF PARA FILTROS DIGITALES.

Filtro Pasabajas

Los coeficientes de la respuesta al impulso para una aproximación RIF, de un filtro pasabajas ideal como se muestra en la figura 3.8, se rigen por la siguiente ecuación:

$$h[n] = \frac{\text{sen}(m\lambda_u)}{m\pi} \qquad (3.14)$$

donde $n = 0,1,...,N-1$ y $m = n-(N-1)/2$

Para filtros con longitud impar, el coeficiente para $n = (N-1)/2$ se obtiene aplicando la regla de L'Hospital, para dar la siguiente ecuación:

$$n\left[\frac{N-1}{2}\right] = \frac{\lambda_u}{\pi} \qquad (3.15)$$

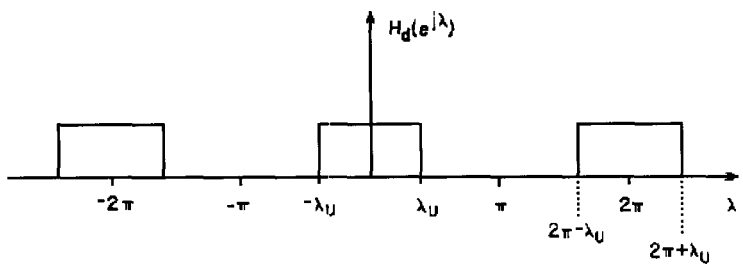

Figura 3.8 Respuesta en frecuencia de un filtro digital ideal pasa bajas

Los coeficientes para una aproximación RIF para filtros pasabajas son obtenidos usando la función en C llamada **PasaBajasIdeal( )** que se muestra al final del capítulo.

Filtros Pasaaltas

Los coeficientes de la respuesta al impulso para la aproximación RIF de un filtro pasaaltas ideal, como se muestra en la figura 3.9, se obtienen con la siguiente ecuación:

$$h[n] = \begin{cases} 1 - \dfrac{\lambda_L}{\pi} & m = 0 \\ \dfrac{-\text{sen}(m\lambda_L)}{m\pi} & m \neq 0 \end{cases} \qquad (3.16)$$

donde $m = n-(N-1)/2$

Esta relación, que nos da los coeficientes de un filtro pasaaltas, la hemos implementado en la

rutina **PasaAltasIdeal( )**, que se muestra al final de éste capítulo.

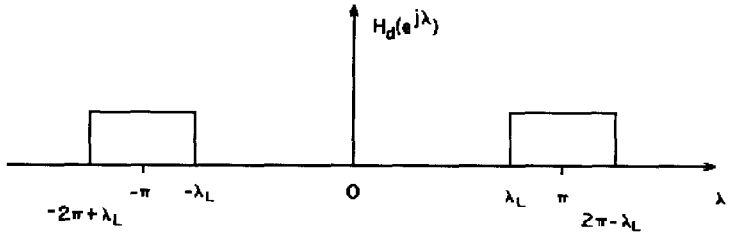

Figura 3.9 Respuesta en frecuencia de un filtro digital ideal pasa altas.

Filtros Pasabanda

Los coeficientes de la respuesta al impulso para la aproximación RIF de un filtro pasabanda ideal, como se muestra en la figura. 3.10, se rigen por la siguiente ecuación: Los coeficientes de la respuesta al impulso para una aproximación RIF de un pasabanda ideal,

$$h[n]=\begin{cases}\dfrac{\lambda_u-\lambda_L}{\pi} & m=0\\ \dfrac{1}{m\pi}\left[\text{sen}(m\lambda_u)-\text{sen}(m\lambda_L)\right] & m\neq 0\end{cases} \qquad (3.17)$$

donde $m=n-(N-1)/2$.

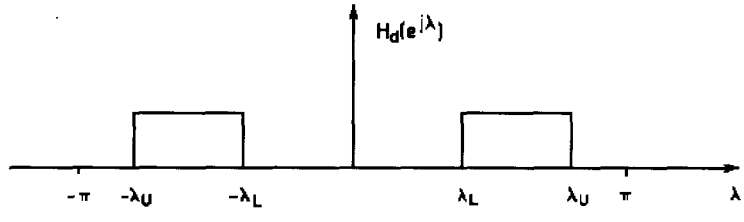

Figura 3.10 Respuesta en frecuencia de un filtro digital ideal pasa banda.

La rutina en lenguaje C que hemos implementado para desarrollar la expresión anterior tiene el nombre de **PasaBandaIdeal( )** , la cual presentamos al final del capítulo.

Rechaza Banda

Los coeficientes de la respuesta al impulso para un filtro RIF con aproximaciones a la respuesta de un filtro rechazabanda ideal, como se muestra en la figura 3.11, están dados por la ecuación siguiente:

$$h[n]=\begin{cases}1+\dfrac{\lambda_L-\lambda_u}{\pi} & m=0\\ \dfrac{1}{m\pi}\left[\text{sen}(m\lambda_L)-\text{sen}(m\lambda_u)\right] & m\neq 0\end{cases} \qquad (3.18)$$

donde $m=n-(N-1)/2$.

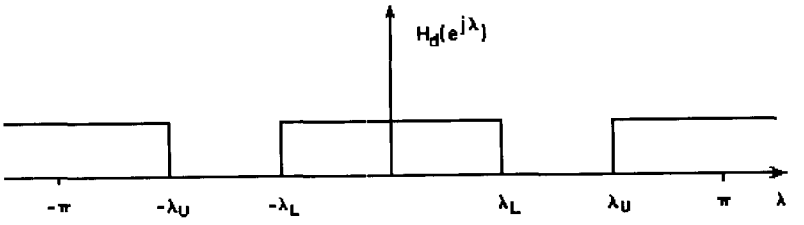

Figura 3.11 Respuesta en frecuencia de un filtro digital ideal rechaza banda

Los coeficientes para el rechazabanda son calculados usando la función en C llamada **RechazaBandaIdeal( )**, la cual se muestra al final del capítulo.

A continuación presentamos un ejemplo de diseño de filtros ideales RIF, haciendo uso de los programas que presentamos en este trabajo, tomemos como ejemplo la figura 3.5 con 21 pulsos, $\lambda_U=-2\pi/5$ y $\lambda_L=2\pi/5$.

```
                MENU PRINCIPAL

        1. FILTROS ANALOGICOS
        2. TRANSFORMADA DISCRETA DE FOURIER
        3. DISEÑO DE FILTROS FIR
        4. DISEÑO DE FILTROS IIR
        5. SALIR DEL PROGRAMA

            Elija su OPCION:
```

Elige la opción 3

```
            DISEÑO DE FILTROS RIF

        1. RESPUESTA DE FILTROS RIF
        2. POR SERIES DE FOURIER
        3. POR MUESTREO EN FRECUENCIA
        4. SALIR

           Elija su OPCION: _
```

elige la opción 2

```
          Respuestas para filtros ideales

          Numero de pasos (max. 28):  21
El valor de lamda u:  -1.256
El valor de lamda l:  1.256
```

Elige la opción 1 e introduce los valores del problema propuesto

A continuación se muestran los resultados que el programa arroja para los diferente filtros.

```
                PARA UN FILTRO PASA BAJAS IDEAL

hh[0] = 0.000203                    hh[14] = 0.075745
hh[1] = 0.033699                    hh[15] = 0.000203
hh[2] = 0.023223                    hh[16] = -0.050392
hh[3] = -0.026892                   hh[17] = -0.026892
hh[4] = -0.050392                   hh[18] = 0.023223
hh[5] = 0.000203                    hh[19] = 0.033699
hh[6] = 0.075745                    hh[20] = 0.000203
hh[7] = 0.062202
hh[8] = -0.093713
hh[9] = -0.302668
hh[10] = -0.399797
hh[11] = -0.302668
hh[12] = -0.093713
hh[13] = 0.062202

      Oprima una tecla para continuar con un F. PASA ALTAS

                PARA UN FILTRO PASA ALTAS IDEAL

hh[0] =  0.000203                   hh[14] =  0.075745
hh[1] =  0.033699                   hh[15] =  0.000203
hh[2] =  0.023223                   hh[16] =  -0.050392
hh[3] =  -0.026892                  hh[17] =  -0.026892
hh[4] =  -0.050392                  hh[18] =  0.023223
hh[5] =  0.000203                   hh[19] =  0.033699
hh[6] =  0.075745                   hh[20] =  0.000203
hh[7] =  0.062202
hh[8] =  -0.093713
hh[9] =  -0.302668
hh[10] =  0.600203
hh[11] =  -0.302668
hh[12] =  -0.093713
hh[13] =  0.062202
```

PARA UN FILTRO PASA BANDA IDEAL

```
hh[0] =  0.000406
hh[1] =  0.067398
hh[2] =  0.046446
hh[3] =  -0.053784
hh[4] =  -0.100784
hh[5] =  0.000406
hh[6] =  0.151490
hh[7] =  0.124404
hh[8] =  -0.187426
hh[9] =  -0.605336
hh[10] =  -0.799594
hh[11] =  -0.605336
hh[12] =  -0.187426
hh[13] =  0.124404
hh[14] =  0.151490
hh[15] =  0.000406
hh[16] =  -0.100784
hh[17] =  -0.053784
hh[18] =  0.046446
hh[19] =  0.067398
hh[20] =  0.000406
```

Oprima una tecla para continuar con un F RECHAZA BANDA_

PARA UN FILTRO EN LA BANDA DE RECHAZO

```
hh[0] =  -0.000203
hh[1] =  -0.067329
hh[2] =  -0.046651
hh[3] =  0.053550
hh[4] =  0.100889
hh[5] =  -0.000000
hh[6] =  -0.151333
hh[7] =  -0.124950
hh[8] =  0.186604
hh[9] =  0.605958
hh[10] =  1.799594
hh[11] =  0.604702
hh[12] =  0.188243
hh[13] =  -0.123855
hh[14] =  -0.151644
hh[15] =  -0.000811
hh[16] =  0.100677
hh[17] =  0.054017
hh[18] =  -0.046239
hh[19] =  -0.067465
hh[20] =  -0.000608
```

Oprima una tecla para continuar ...

PARA UN FILTRO EN LA BANDA DE RECHAZO

```
hh[0] =  -0.000203
hh[1] =  -0.067329
hh[2] =  -0.046651
hh[3] =  0.053550
hh[4] =  0.100889
hh[5] =  -0.000000
hh[6] =  -0.151333
hh[7] =  -0.124950
hh[8] =  0.186604
hh[9] =  0.605958
hh[10] =  1.799594
hh[11] =  0.604702
hh[12] =  0.188243
hh[13] =  -0.123855
hh[14] =  -0.151644
hh[15] =  -0.000811
hh[16] =  0.100677
hh[17] =  0.054017
hh[18] =  -0.046239
hh[19] =  -0.067465
hh[20] =  -0.000608
```

Oprima una tecla para continuar ...

3.2.3 VENTANA RECTANGULAR

El diseño de filtros usando el método por series de Fourier produce el fenómeno de Gibbs, el cual hace aparecer desbordamientos y rizos en la respuesta de frecuencia. Una manera de reducir estos efectos consiste en multiplicar la respuesta al impulso por una ventana, de modo que la respuesta al impulso sea modificada por alguna secuencia-función que no sea la rectangular, implícita en la misma respuesta impulsiva.

En general, el truncamiento de la respuesta impulsiva de un filtro se puede considerar como el producto de una respuesta impulsiva de longitud infinita por una ventana rectangular, como se muestra en la figura 3.12

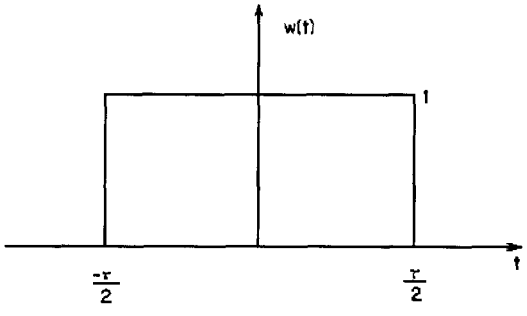

Figura 3.12 Ventana rectangular

La ventana rectangular tiene valor unitario para los valores de t donde se quiere preservar la respuesta al impulso, en tanto que vale cero para aquellos valores donde se quiere eliminar esta respuesta; tal ventana es definida por:

$$\omega(t) = \begin{cases} 1 & |t| < \dfrac{\tau}{2} \\ 0 & \text{otro caso} \end{cases}$$

La transformada de Fourier de la ventana rectangular está dada por

$$W(f) = \frac{\tau \operatorname{sen} \pi f \tau}{\pi f \tau} \qquad (3.19)$$

La gráfica de magnitud de la ventana rectangular es como se muestra en la figura 3.13

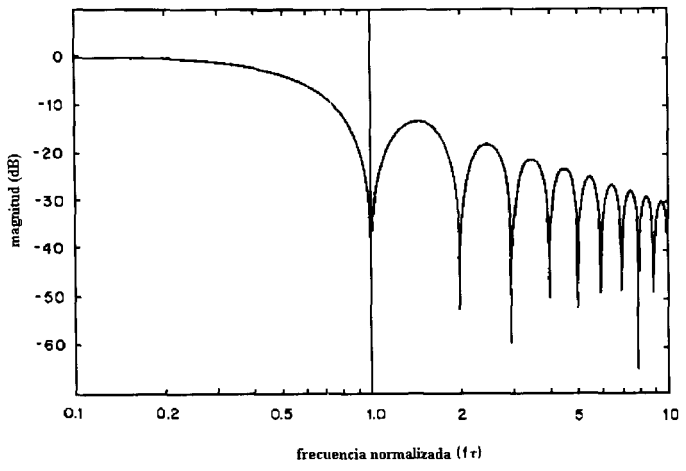

Figura 3.13 Magnitud del espectro para una ventana rectangular continua en tiempo.

Es de apreciar que los picos de los primeros nueve lóbulos laterales se ubican en puntos con atenuación de: 13.3, 17.8, 20.8, 23.0, 24.7, 26.2, 27.4, 28.5 y 29.5 dB, respectivamente. Estos valores de atenuación pueden verificarse con la función en lenguaje C **RespuestaRectangular()**, que hemos elaborado y presentamos al final del capítulo. Cabe señalar que la respuesta de la ventana rectangular sirve, usualmente, como referencia con respecto a la cual se puede hacer la comparación hacia respuestas de otras ventanas.

3.2.3.1 VENTANA DISCRETA EN TIEMPO.

Como se sabe, los coeficientes de los filtros RIF existen sólo para valores enteros de n, o valores discretos de $t = nT$; por ello, resulta conveniente trabajar con funciones ventana que estén definidas en términos de n, en lugar de t. Si la función definida para $\omega(t)$ se muestrea usando $N = 2M + 1$ muestras, con una para $t = 0$ y las demás en nT para $n = \pm 1, \pm 2, ..., \pm M$, la función ventana discreta queda especificada por

$$\omega[n] = \begin{cases} 1 & -M \le n \le M \\ 0 & \text{otro caso} \end{cases} \qquad (3.21)$$

Para un número par de muestras, la ventana rectangular queda definida como:

$$\omega[n] = 1 \qquad -(M-1) \le n \le M \qquad (3.22)$$

ó

$$\omega[n] = 1 \qquad -M \le n \le (M-1) \qquad (3.23)$$

La ventana especificada por la ecuación (3.22) quedará centrada entre los puntos en $n = 0$ y $n = 1$, en tanto que la ventana especificada por la ecuación (3.23) lo estará entre $n = -1$ y $n = 0$. En muchas aplicaciones (especialmente en lenguajes como C que usan el índice referido a cero), es conveniente tener a $\omega[n]$, para $0 \le n \le (N-1)$, definida por :

$$\omega[n]=1 \qquad 0 \le n \le (n-1) \qquad (3.24)$$

Es posible diferenciar entre las ventanas que se han definido, una para valores positivos y negativos de la variable de tiempo y la otra solo para valores positivos, aprovechando algunos de los términos propios de la teoría de series de tiempo. Es el caso que a las ventanas con valores positivos y negativos en el índice de tiempo se les denomina ventanas tipo lag, en tanto que a las otras se les conoce como ventanas de tipo data, las cuales, cuando N es par, son particularmente útiles para reducir el efecto de fuga en las aplicaciones TRF.

3.2.3.2 VENTANAS EN FRECUENCIA Y VENTANAS ESPECTRALES.

La transformada de Fourier discreta en tiempo (TFDT) de la ventana lag definida en la ecuación a esta especificada por:

$$W(f)=\frac{\operatorname{sen}[\pi f(2M+1)]}{\operatorname{sen}(\pi f)} \qquad (3.25)$$

La ecuación anterior esta estrechamente relacionada con la función **KerneldeDirichlet**, **Dn (.)**, la cual esta definida en las formas siguientes:

$$D_n(\theta) \equiv \frac{1}{2\pi}\sum_{k=-n}^{n}\cos k\theta = \frac{\operatorname{sen}\{[n+(1/2)]\theta\}}{\operatorname{sen}(\theta/2)}$$

$$D_n(x) \equiv \sum_{k=-n}^{n}\exp(2\pi jkx) = \frac{\operatorname{sen}\{(2n+1)\pi x\}}{\operatorname{sen}(\pi x)}$$

$$D_n(x) \equiv \frac{1}{2\pi}\sum_{k=-n}^{n}\cos kx = \frac{\operatorname{sen}\{[n+(1/2)]x\}}{2\operatorname{sen}(x/2)}$$

La magnitud de la ecuación 3.25 es gratificada en la figura 3.14, para $N=11$, y en la figura 3.15 para $N=21$. Como se observa en estas dos gráficas, cuando el número de puntos en la ventana se incrementa, el ancho de los lóbulos laterales decrece. Por otro lado, es posible observar que los lóbulos laterales, en la figura para $N=11$, se encuentran en niveles de atenuación de: 13.0, 17.1, 19.3, 20.5 y 20.8 dB; en tanto que para $N=21$, los niveles de atenuación son: 13.2, 17.6, 20.4, 22.3, 23.7, 24.8, 25.5, 26.1 y 26.3 dB. Estos valores se pueden verificar con la función en lenguaje C que hemos elaborado y presentamos al final del capitulo con el nombre de **RespuestaRetangularDisc( )**.

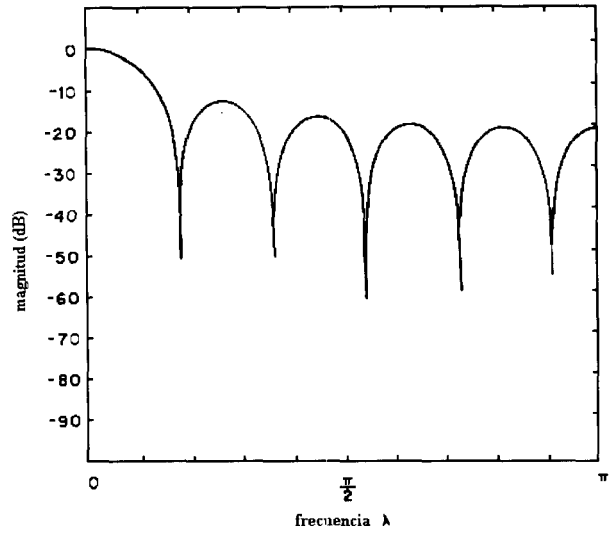

Figura 3.14 Magnitud de la Transformada de Fourier discreta en tiempo para una ventana rectangular de 11 puntos.

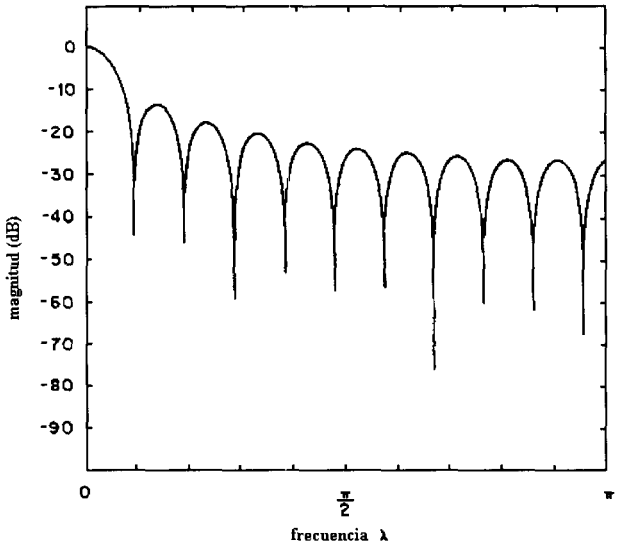

Figura 3.15 Magnitud de la Transformada de Fourier discreta en tiempo para una ventana rectangular con 21 puntos

La TFDT de la ventana data definida en la ecuación (3.24) está dada por la siguiente ecuación:

$$W(f) = \exp[-j\pi f(N-1)]\frac{\text{sen}(N\pi f)}{\text{sen}(\pi f)} \qquad (3.26)$$

Una función como la definida por la ecuación 3.25, que es la transformada de Fourier de una ventana lag, recibe el nombre de ventana espectral; en tanto que una función definida como en la

ecuación (3.26), que es la transformada de Fourier de una ventana data es llamada ventana de frecuencia. Cabe señalar, también, la existencia de alguna diferencia entre las transformadas de las ventanas lag y data respecto de la ventana continua en tiempo; por supuesto, esta diferencia esta asociada al aliasing introducido en el muestreo de la ventana continua limitada en tiempo (ilimitada en frecuencia).

A continuación presentamos un ejemplo de respuesta rectangular y triangular para un filtro con una frecuencia de corte de 2khz con $\tau = 0.01$, cuando $M = 0$, que al igual que los anteriores haremos uso del programa de este trabajo.

```
                MENU PRINCIPAL

        1. FILTROS ANALOGICOS
        2. TRANSFORMADA DISCRETA DE FOURIER
        3. DISEÑO DE FILTROS FIR
        4. DISEÑO DE FILTROS IIR
        5. SALIR DEL PROGRAMA

            Elija su OPCION:
```

Elige la opción 3

```
            DISEÑO DE FILTROS RIF

        1. RESPUESTA DE FILTROS RIF
        2. POR SERIES DE FOURIER
        3. POR MUESTREO EN FRECUENCIA
        4. SALIR

           Elija su OPCION:
```

Elige la opción 2

```
             RIF por series de Fourier

      1. Respuestas para filtros ideales
      2. Respuesta rectangular y triangular
      3. Ventana rectangular, Hann y Hamming
      4. Regresar al menu

         Elija su opcion: _
```

Elige la opción 2 y despues introduce los valores que se te piden en el ejemplo.

```
          Respuesta Rectangular y Triangular

Cual es la frecuencia: 2000
     El valor de tau: .01
Para ver el resultado en db oprima 1: 1
Cual es el valor de M:10

     La respueta  rectangular CONTINUA es: -90.000000

     La respuesta rectangular DISCRETA es: 1.000000

     La respuesta  triangular CONTINUA es: -90.000000
```

3.2.4 VENTANA TRIANGULAR

La ventana triangular esta definida por

$$\omega(t) = 1 - \frac{2|t|}{\tau} \qquad |\tau| \leq \frac{\tau}{2} \tag{3.27}$$

Las funciones ventana casi siempre se presentan con simetría par, por lo que es costumbre mostrar únicamente la porción positiva del índice de tiempo de la ventana, como en la figura 3.16

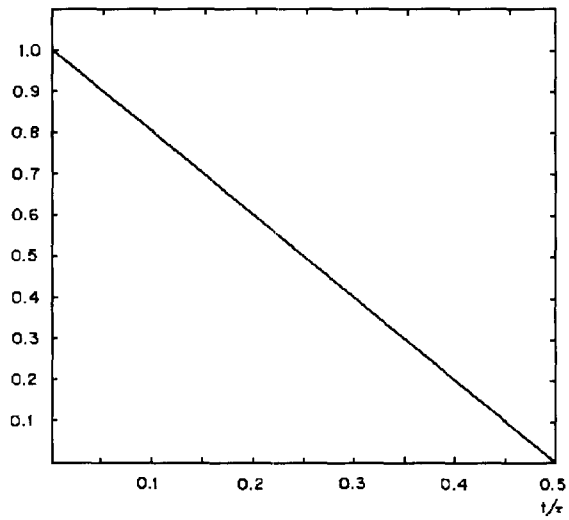

Figura 3.16 Grafica del lado derecho de una ventana triangular

La transformada de Fourier de la ecuación (3.27), que representa a la ventana triangular, es la siguiente:

$$W(f) = \frac{\tau}{2}\left[\frac{\text{sen}(\pi f\tau/2)}{(\pi f\tau/2)}\right]^2 \tag{3.28}$$

La magnitud de 3.28 se gráfica en la figura 3.17. donde los picos de los primeros cuatro lóbulos aparecen con un nivel de atenuación de: 26.5, 35.7, 41.6 y 46.0 dB, respectivamente. Estos resultados se pueden verificar con la función en lenguaje C **RespuestaTriangularCont( )** listada al final del capítulo.

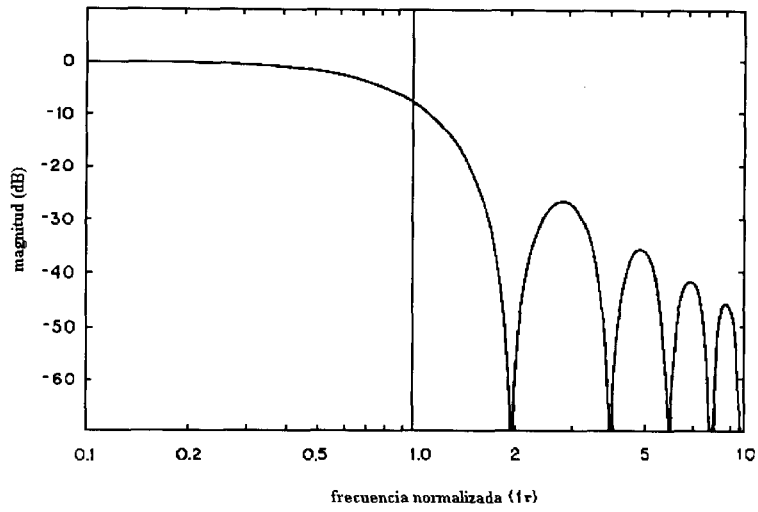

Figura 3.17 Respuesta en magnitud de una ventana triangular continua en tiempo.

3.2.4.1 VENTANA TRIANGULAR EN TIEMPO DISCRETO

Si la función definida por la ecuación (3.27) es muestreada usando $N = 2M + 1$ muestras con $\tau = 2MT$, con una muestra en $t = 0$, y las demás en nT para $n = \pm 1, \pm 2, ..., \pm M$, se tendrá que:

$$\omega[n] = 1 - \frac{2|n|}{2M} \qquad -M \le n \le M \qquad (3.29)$$

para la normalización con $T = 1$. Esta ecuación es expresada en términos del total de muestras N, sustituyendo $(N-1)/2$ por M se obtiene la siguiente ecuación:

$$\omega[n] = 1 - \frac{2|n|}{N-1} \qquad -\frac{(N-1)}{2} \le n \le \frac{N-1}{2} \qquad (3.30)$$

Ecuación que define la ventana triangular en tiempo discreto. De esta expresión es posible observar que los dos puntos extremos no contribuyen con el contenido de la ventana, por lo que la longitud de ésta se puede reducir a $N-2$ muestras.

(Al final del capítulo se presentan algunas funciones en lenguaje C que generan varias formas de una ventana triangular discreta en tiempo.)

3.2.4.2 VENTANAS ESPECTRALES Y DE FRECUENCIA

El espectro de la ventana obtenida pro la TFDT de la ventana lag de la ecuación (3.30), esta dada por:

$$W(f) = \frac{1}{M}\left[\frac{\text{sen}(M\pi f)}{\text{sen}(\pi f)}\right]^2 \qquad (3.31a)$$

ó

$$W(\theta) = \frac{2}{N}\left\{\frac{\text{sen}[(N/4)\theta]}{\text{sen}[(1/2)\theta]}\right\} \qquad (3.31b)$$

donde $M = \frac{N-1}{2}$ y $\theta = \frac{2\pi f}{f_s}$

Las respuestas de magnitud de la ecuación, (3.31a) para $N = 11$ y $N = 21$, están graficadas en la figura 3.18, con los valores que resultan de usar la función en lenguaje C **RespuestaTriangularDisc( )** descrita al final de esta capítulo.

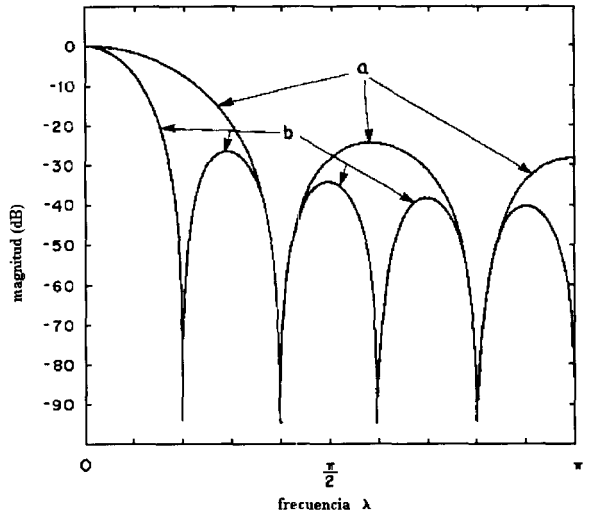

Figura 3.18 Magnitud de la Transformada de Fourier discreta en tiempo para (a) una ventana triangular de 11 puntos y (b) para una de 21 puntos.

3.2.5 APLICACION DE VENTANAS A FILTROS CON SERIES DE FOURIER.

Conceptualmente, una ventana es una función de peso que se aplica a la entrada de un filtro RIF ideal, de modo que lo que resulta es la respuesta impulsiva de este filtro modificada para exhibir coeficientes ponderados por los valores de la ventana punto a punto. Sin embargo, aprovechando que la multiplicación es una operación asociativa, la operación de transformación se puede reducir a multiplicar los coeficientes de la ventana y los coeficientes del filtro original para obtener el conjunto modificado de los coeficientes del filtro . En nuestro caso, los coeficientes producidos para la respuesta impulsiva han sido generados usando el formato de ventana data, es decir $h[n]$ esta definida para $0 \le n \le N-1$

Consideremos el caso del filtro pasabajas con 21 derivaciones, frecuencia de corte de $2KHz$, y frecuencia de muestreo de $5KHz$, y apliquémosle una ventana triangular. Los resultados de aplicar la ventana aparecen en la tabla 3.4, donde se listan los valores originales del filtro, los coeficientes de la ventana y, por supuesto, el valor final de los coeficientes después de que la ventana ha sido aplicada. La respuesta en frecuencia del filtro ventaneado se muestra en las figuras 3.19 y 3.20 donde la escala lineal deja ver una respuesta muy clara, particularmente en la banda de transición, como se observa en la figura 3.19, claridad que desmerece cuando lo que se observa es la gráfica de la figura 3.20 que usa una escala en $dB's$.

n	h[n]	ω[n]	ω[n]· h[n]
0,20	0.000000	0.000000	0.000000
1,19	-0.033637	0.090909	-0.006116
2,18	-0.023387	0.181818	-0.006378
3,17	0.026728	0.272727	0.009719
4,16	0.050455	0.363636	0.022934
5,15	0.000000	0.454545	0.000000
6,14	-0.075683	0.545455	-0.048162
7,13	-0.062366	0.636364	-0.045357
8,12	0.093549	0.727273	0.076540
9,11	0.302731	0.909091	0.275210
10	0.400000	1.00	0.400000

Tabla 3.4 coeficientes de un filtro pasabajas de 21 derivaciones.
{$h[n]$ es el coeficiente original; $\omega[n]$ son los coeficientes de la ventana triangular}

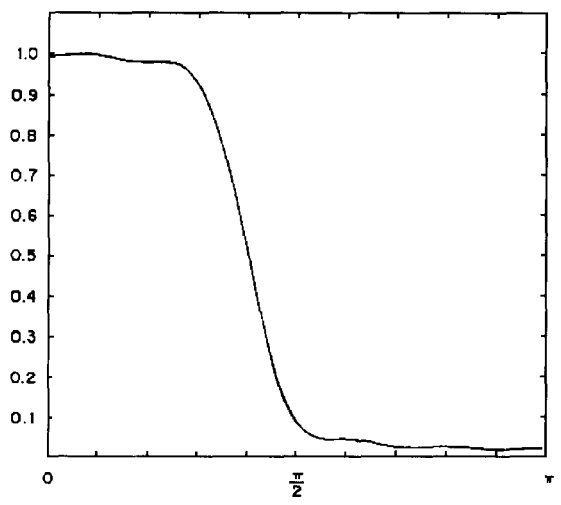

Figura 3.19 Respuesta en magnitud (como un porcentaje de pico) para un ventana triangular de un filtro pasa bajas de 21 derivaciones.

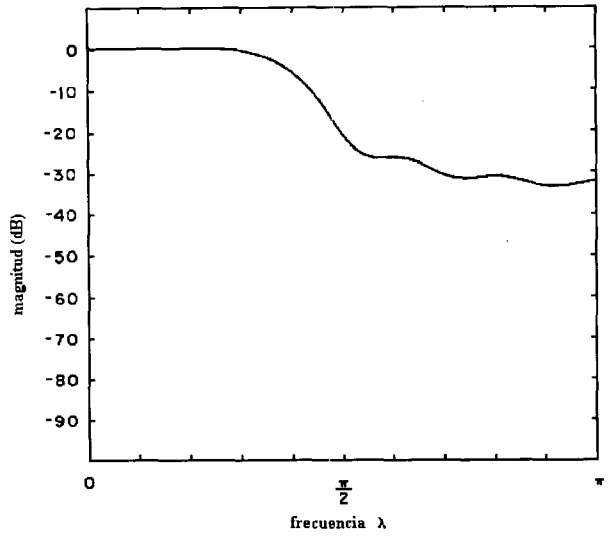

Figura 3.20 Respuesta en magnitud (en decibeles) para una ventana triangular de un filtro pasa bajas de 21 derivaciones.

3.2.6 VENTANA VON HANN

La función ventana en tiempo continúo von Hann mostrada en la figura 3.21 esta definida por:

$$\omega(t) = 0.5 + 0.5\cos\frac{2\pi t}{\tau} \qquad |t| \le \frac{\tau}{2} \tag{3.32}$$

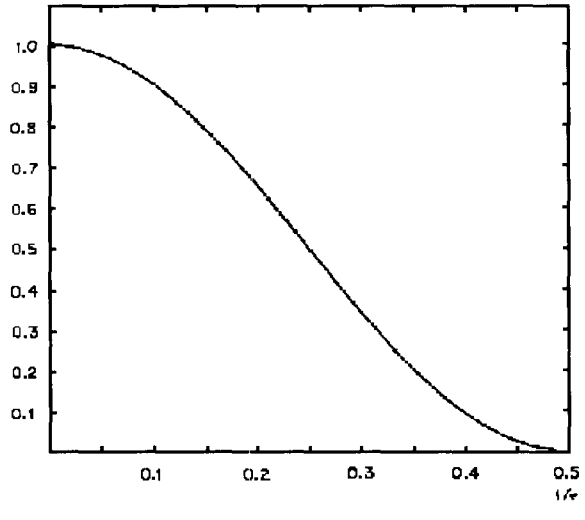

Figura 3.21 Ventana Von Hann.

La correspondiente respuesta en frecuencia, que se muestra en la figura 3.22 esta dada por la siguiente ecuación:

$$W(f) = 0.54\tau\,\mathrm{sen}\,c(\pi f\tau) + 0.23\tau\,\mathrm{sen}\,c[\pi\tau(f-\tau)] + 0.23\tau\,\mathrm{sen}\,c[\pi\tau(f+\tau)] \tag{3.33}$$

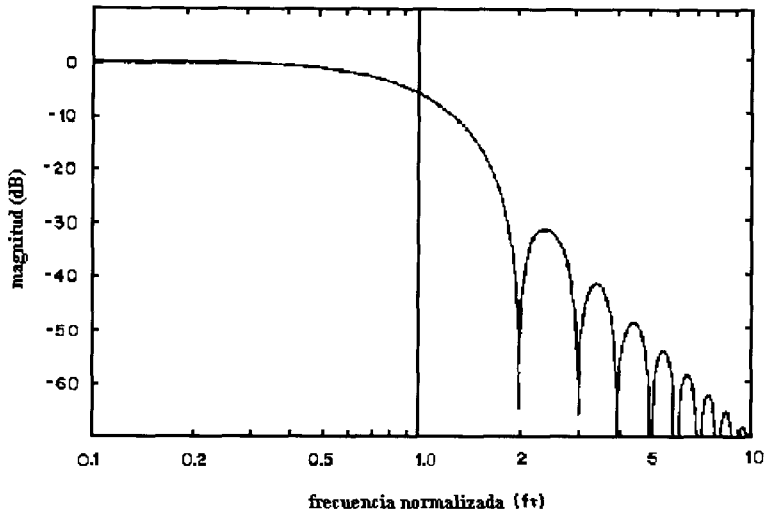

Figura 3.22 Respuesta en magnitud de una ventana Von Hann.

El primer rizo de esta secuencia es 31.4 *dB* abajo del lóbulo principal, y este es dos veces más grande como el lóbulo principal de la ventana rectangular.

3.2.6.1 VENTANA VON HANN EN TIEMPO DISCRETO

Si la función definida por la ecuación 3.32 es muestreada usando $N = 2M + 1$ muestras, con una muestra en $t = 0$ y muestras en nT para $n = \pm 1, \pm 2, ..., \pm M$, la función mostrada llega a hacer la siguiente:

$$\omega[n] = 0.5 + 0.5 \cos \frac{\pi n}{M} \qquad -M \le n \le M \qquad (3.34)$$

Para la normalización cuando $T = 1$. La ecuación 3.34 revela que $\omega(n) = 0$ para $n = \pm M$. Esto quiere decir, que dos últimos puntos no contribuyen para el contenido de la ventana, y que la longitud de la ventana es efectivamente reducido a $\omega = 21$ muestras. Para obtener un total de N muestras distintas de cero se debe sustituir $M + 1$ por M en la ecuación 3.34 para obtener la siguiente ecuación 3.35:

$$\omega[n] = 0.5 + 0.5 \cos \frac{2\pi n}{2(M + 1)} \qquad -M \le n \le M \qquad (3.35)$$

La ecuación 3.35 puede ser reescrita en términos de N sustituyendo $(N-1)/2$ por M para obtener la siguiente ecuación.

$$\omega[n] = 0.5 + 0.5 \cos \frac{2\pi n}{N - 1} \qquad \frac{-(N-1)}{2} \le n \le \frac{N-1}{2} \qquad (3.36)$$

n impar

Para un número par de muestras, los valores de las ventanas se obtienen por sustitución, ya sea por $(n+½)$ o $(n-½)$ sustituidas por n en la ecuación 3.36 de dónde se obtiene las siguientes ecuaciones:

$$\omega[n]=0.5+0.5\cos\frac{\pi(2n+1)}{N-1} \qquad \frac{-N}{2}\le n\le\frac{N}{2}-1 \qquad (3.37)$$

para N par, centrado alrededor de $n"="\frac{-1}{2}$

$$\omega[n]=0.5+0.5\cos\frac{\pi(2n+1)}{N-1} \qquad \frac{-N}{2}+1\le n\le\frac{N}{2} \qquad (3.38)$$

Para N par, centrado alrededor de $n"="\frac{1}{2}$

La función en C **VentanaHann( )** genera los coeficientes de von Hann Ventana y se muestra al final del capítulo.

Aplicando una ventana von Hann para un filtro pasabajas de 21 derivaciones del ejemplo presentado al inicio del capitulo se obtiene la tabla 3.5 en la cual se muestran los valores originales del filtro los correspondientes coeficientes de la ventana en tiempo discreto, y los valores finales de los coeficientes del filtro, después de ser aplicado el ventaneo.

La respuesta al filtro ventaneado se muestra en la figura 3.23

0.20	0.000000	0.000000	0.000000
1.19	0.033637	0.244720	-0.000823
2.18	0.023387	0.095492	-0.002233
3.17	0.026728	0.206107	0.005509
4.16	0.050455	0.345492	0.017432
5.15	0.000000	0.500000	0.000000
6.14	-0.075683	0.654508	-0.049535
7.13	-0.062366	0.793893	-0.495120
8.12	0.093549	0.904508	0.084616
9.11	0.302731	0.975528	0.295323
10	0.400000	1.000000	0.400000

Tabla 3.5 Coeficientes de un filtro pasabajas de 21 derivaciones $h[n]$ es el coeficiente original; $\omega[n]$ son los coeficientes de la ventana von Hann}

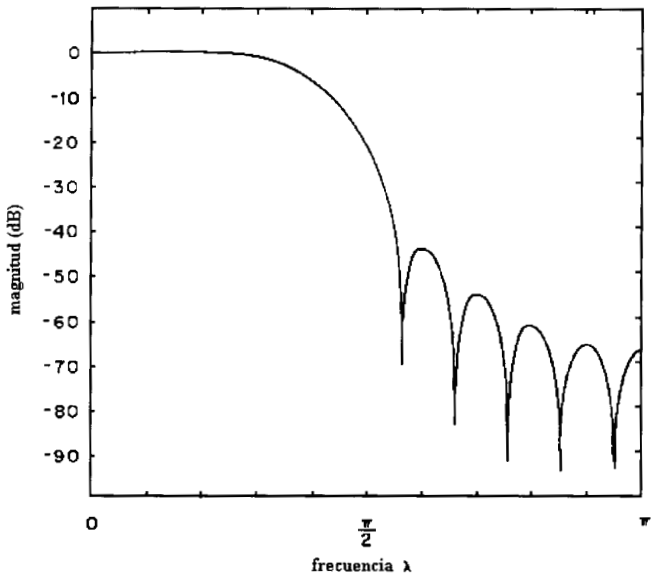

Figura 3.23 Respuesta en magnitud para una ventana Von Hann de un filtro pasabajas de 21 derivaciones.

3.2.7 VENTANA DE HAMMING

La función de la ventana de Hamming en tiempo continuo, que se muestra en la figura 3.24, esta definida por la siguiente ecuación.

$$\omega(t) = 0.54 + 0.46\cos\frac{2\pi t}{\tau} \qquad |t| \le \frac{\tau}{2} \tag{3.39}$$

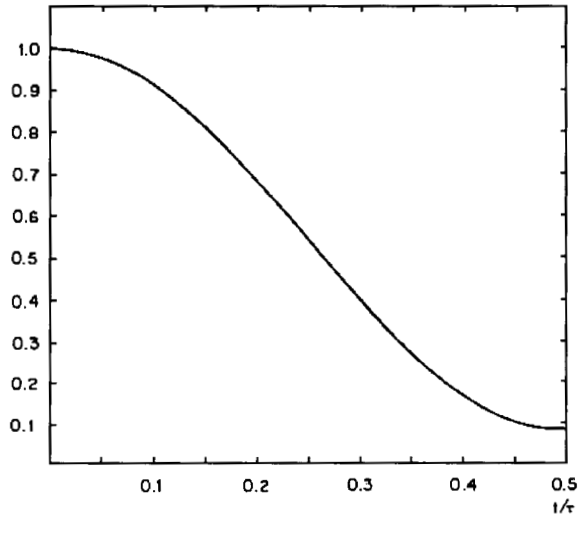

Figura 3.24 Ventana Hamming

La transformada de Fourier de ésta ecuación esta definida por la ecuación.

$$W(f) = 0.54\tau\,\mathrm{sen}\,c(\pi f\tau) + 0.23\tau\,\mathrm{sen}\,c[\pi\tau(f-\tau)] + 0.23\tau\,\mathrm{sen}\,c[\pi\tau(f+\tau)] \tag{3.40}$$

La magnitud de la ecuación 3.40 es gráficada en la figura 3.25. El lóbulo más grande es de 42.6 *dB* por debajo del principal lóbulo que es dos veces ancho, así como el principal lóbulo de la respuesta del ventaneo rectangular.

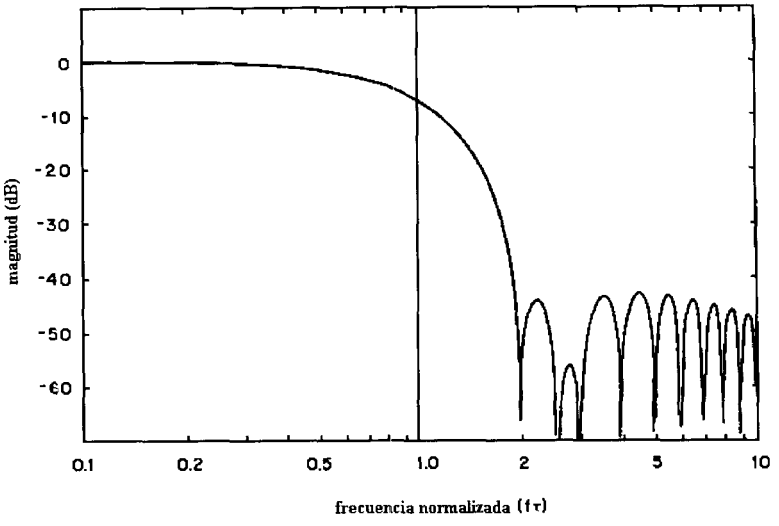

Figura 3.25 Respuesta en magnitud de una ventana Hamming

3.2.7.1 VENTANA DE HAMMING EN TIEMPO DISCRETO

Si la función definida por la ecuación 3.39 es muestreada usando $N = 2M + 1$ muestras, con una muestra en $T = 0$ y muestras en nT para $n = \pm 1, \pm 2, ..., \pm M$, el muestreo de la función de ventaneo llega a ser una ventana desfasada definida por la siguiente ecuación.

$$\omega[n] = 0.54 + 0.46 \cos \frac{2\pi n}{2M} \qquad -M \le n \le M \qquad (3.41)$$

Para el caso de la normalización de $T = 1$, la ecuación 3.41 puede ser expresada en términos del número total de muestras N sustituyendo $(N-1)/2$ por M para obtener la siguiente ecuación

$$\omega[n] = 0.54 + 0.46 \cos \frac{2\pi n}{N-1} \qquad \frac{-(N-1)}{2} \le n \le \frac{N-1}{2} \qquad (3.42)$$

donde n es impar

Para un número par de muestras los valores de las ventanas pueden ser obtenidos por sustitución de $n + \frac{1}{2}$ por n en la ecuación 3.42 para obtener la siguiente ecuación.

$$\omega[n] = 0.54 + 0.46 \cos \frac{\pi(2n+1)}{N-1} \qquad \frac{-N}{2} \le n \le \frac{N}{2} - 1 \qquad (3.43)$$

para N par, centrada alrededor de $n = \frac{-1}{2}$

La forma de la ventana de datos puede ser obtenida sustituyendo $[n-(N-1)/2]$ por n en la ecuación 3.42 ó sustituyendo $(n - N/2)$ por n en la ecuación 3.43, para obtener la siguiente ecuación:

$$\omega[n] = 0.54 + 0.46\cos\frac{2\pi n}{N-1} \qquad 0 \le n \le N-1 \qquad (3.44)$$

Aplicando una ventana de Hamming para un filtro pasabajas se obtienen los valores ventaneados de $h[k]$, los cuales son listados en la tabla 3.6, y su correspondiente, respuesta en frecuencia se muestra en la figura 3.26.

k	$h[k]$
0,20	0.000000
1,19	-0.003448
2,18	-0.003926
3,17	0.007206
4,16	0.020074
5,15	0.000000
6,14	-0.051627
7,13	-0.050540
8,12	0.085330
9,11	0.295915
10	0.400000

Tabla 3.6 Coeficientes de un filtro pasabajas con 21 derivaciones de un ventaneo Hamming

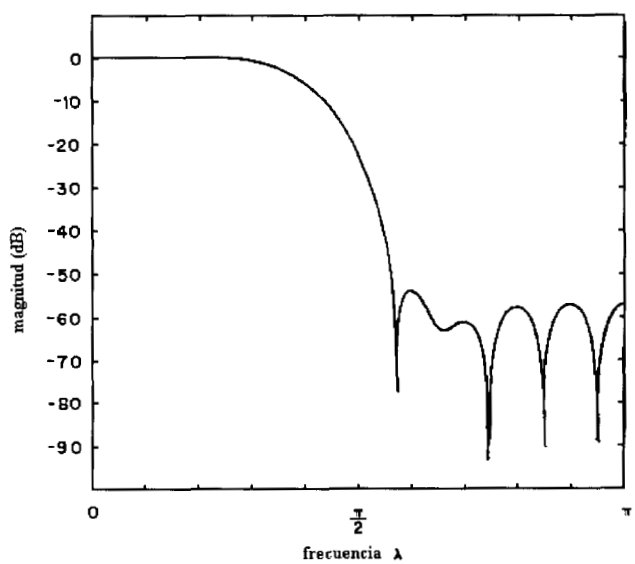

Figura 3.26 Respuesta en magnitud de una ventana Hamming --para un filtro pasa bajas de 21 derivaciones

La función en C **VentanaHamming( )**, genera ordenadamente la ventana de Hamming. Esta función se puede encontrar al final del capitulo

3.2.8 VENTANA DOLPH- CHEBYSHEV

Las ventanas Dolph-Chebysher, a diferencia de las otras ventanas, estas se definen con una expresión en forma cerrada, cuando la ventana en el dominio del tiempo no esta definida; en su lugar, esta ventana se define como la transformada inversa de Fourier de la respuesta muestreada en frecuencia, la cual se define de la siguiente forma:

$$W[k] = (-1)^k \frac{\cos\left\{N \cos^{-1}\left[\beta \cos(\pi k / N)\right]\right\}}{\cosh(N \cosh^{-1} \beta)} \qquad -(N-1) \le k \le N-1 \qquad (3.45)$$

Un nivel del rizo a - 80 dB es obtenido por esta respuesta, pero en realidad, la ecuación 3.45 define una familia de ventanas en las cuales la mínima atenuación de la banda de rechazo es un factor de β. Una atenuación de la banda de rechazo de 20 α dB se obtiene por una evaluación de β dada por la ecuación 3.46.

$$\beta = \cosh\left[\frac{1}{N} \cosh^{-1}(10^{\alpha})\right] \qquad (3.46)$$

A menudo $\beta > 1$ y consecuentemente, la evaluación de la ecuación 3.44 puede comportarse como el coseno inverso, con valores tan grandes como la unidad; en tal caso puede usarse la ecuación 3.47:

$$\cos^{-1} x \begin{cases} \dfrac{\pi}{2} - \tan^{-1}\left(\dfrac{x}{\sqrt{1-x^2}}\right) & |x| < 1 \\ \ln\left(x + \sqrt{x^2 - 1}\right) & |x| \ge 1 \end{cases} \qquad (3.47)$$

Para ilustrar las ventanas presentamos el siguiente ejemplo, el cual hace referencia a las ventanas Triangular, Hann y Hamming.

Obtener las respuestas en las diferentes ventanas, cuando N=11 y el filtro esta centrado en 5

```
        MENU PRINCIPAL

1. FILTROS ANALOGICOS
2. TRANSFORMADA DISCRETA DE FOURIER
3. DISEÑO DE FILTROS FIR
4. DISEÑO DE FILTROS IIR
5. SALIR DEL PROGRAMA

    Elija su OPCION:
```

Elige la opción 3

```
    DISEÑO DE FILTROS RIF

1. RESPUESTA DE FILTROS RIF
2. POR SERIES DE FOURIER
3. POR MUESTREO EN FRECUENCIA
4. SALIR

   Elija su OPCION: _
```

Elige la opción 2

```
RIF por series de Fourier

1. Respuestas para filtros ideales
2. Respuesta rectangular y triangular
3. Ventana rectangular, Hann y Hamming
4. Regresar al menu

Elija su opcion: _
```

Elige la opción 3

```
VENTANA TRIANGULAR, HANN Y HAMMING

Introducir el valor de N: 11
Cual es el centro: 5
```

Introduce los valores que te dan en el problema.

```
RESPUESTAS DE LAS DIFERENTES VENTANAS

Los valores de la VENTANA RECTANGULAR son:

ventana[0]= 1.000000
ventana[1]= 0.833333
ventana[2]= 0.666667
ventana[3]= 0.500000
ventana[4]= 0.333333
ventana[5]= 0.166667

Oprime una tecla para continuar...

Los valores de la VENTANA LAG son:

V. LAG[0]= 1.000000
V. LAG[1]= 0.833333
V. LAG[2]= 0.666667
V. LAG[3]= 0.500000
V. LAG[4]= 0.333333
V. LAG[5]= 0.166667

Oprime una tecla para continuar...

Los valores de la VENTANA DATA son:

V. DATA[0]= 0.166667
V. DATA[1]= 0.333333
V. DATA[2]= 0.500000
V. DATA[3]= 0.666667
V. DATA[4]= 0.833333
V. DATA[5]= 1.000000

Oprime una tecla para continuar...
```

```
Los valores de la VENTANA HAMMING son:

     V. Hamming[0]= 1.000000
     V. Hamming[1]= 0.912148
     V. Hamming[2]= 0.682148
     V. Hamming[3]= 0.397852
     V. Hamming[4]= 0.167852
     V. Hamming[5]= 0.080000

Oprima una tecla para continuar...
```

3.3 DISEÑO DE FILTROS RIF CON EL METODO DE MUESTREO EN FRECUENCIA.

Para el caso del diseño de filtros RIF por el método de series de Fourier, la respuesta deseada se especificó en el dominio de la frecuencia continúa, en tanto que los coeficientes de la respuesta impulsiva discreta en tiempo se obtuvieron por un calculo de series de Fourier. Este procedimiento se puede modificar de modo que la respuesta en frecuencia deseada esté especificada en el dominio de la frecuencia discreta, y entonces se pueda usar la transformada discreta de Fourier inversa (TDFI) para obtener la respuesta al impulso en el tiempo discreto correspondiente.

Considérese el caso de un filtro pasabajas con 21 derivaciones y con una frecuencia de corte normalizada $\lambda_u = 3\pi/7$ La respuesta en magnitud de la muestra para frecuencias positivas se ilustra en la figura 3.27 La frecuencia de corte normalizada λ_u esta centrada entre $n=4$ y $n=5$, en tanto que la frecuencia folding normalizada en $\lambda=\pi$ la cual se encuentra entre $n=10$ y $n=11$. Si asumimos que $H_d(-n)=H_d(n)$ y usamos la TDF inversa podemos obtener los coeficientes del filtro que se listan en la tabla 3.7.

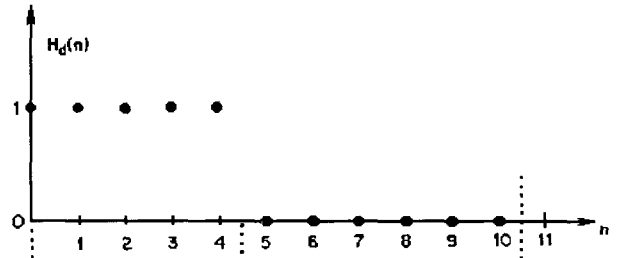

Figura3.27 Respuesta en magnitud en frecuencia discreta para un filtro pasabajas con $\lambda_U=3\pi/7$

h[0]	=	h[20]	=	0.037334
h[1]	=	h[19]	=	-0.021192
h[2]	=	h[18]	=	-0.049873
h[3]	=	h[17]	=	0.000000
h[4]	=	h[16]	=	0.059380
h[5]	=	h[15]	=	0.030376
h[6]	=	h[14]	=	-0.066090
h[7]	=	h[13]	=	-0.085807
h[8]	=	h[12]	=	0.070096
h[9]	=	h[11]	=	0.311490
h[10]	=	0.428571		

Tabla 3.7 Coeficientes para el filtro de 21 derivaciones.

La respuesta en el dominio de frecuencia continua de este filtro RIF, en una escala lineal, se ilustra en la figura 3.28, en tanto que la misma respuesta usando una escala logarítmica se deja ver en la figura 3.29.

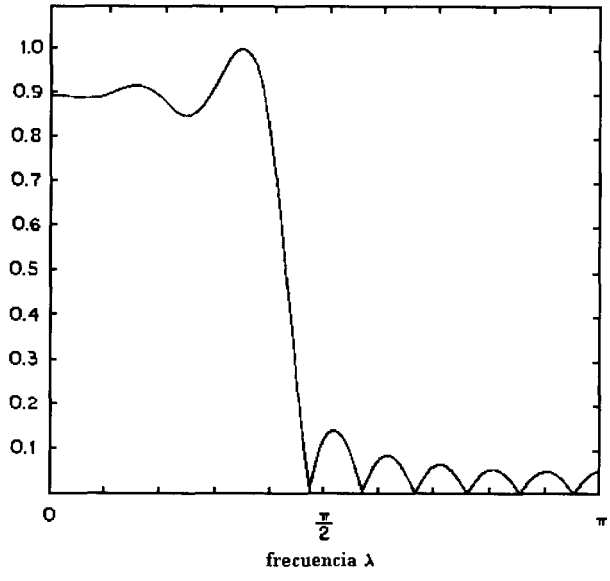

Figura 3.28 Respuesta en magnitud para el filtro de 21 derivaciones

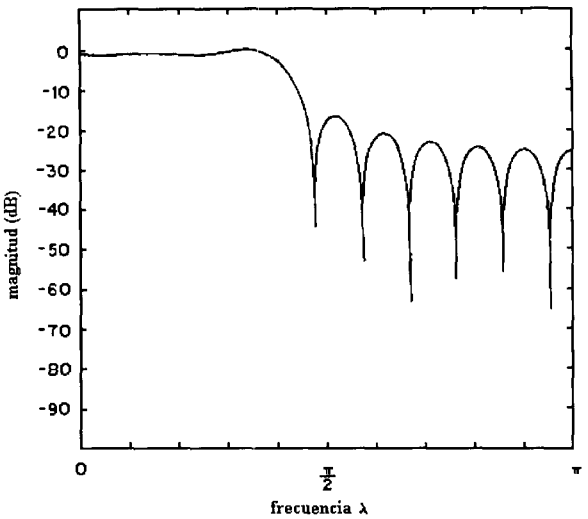

Figura 3.29 Respuesta del filtro de 21 derivaciones, graficado en decibeles.

Los rizos de las respuestas, en la banda de paso y la banda de rechazo, se pueden modificar incorporando una o más muestras en la banda de transición, con valores entre $H_d(m) = 1$ (banda de paso) y $H_d(m) = 0$ (banda de rechazo). Por ejemplo, la figura 3.30 muestra una variante de la respuesta observada en la figura 3.27 que introduce una muestra en la banda de transición, $H_d(5) = 0.5$. La respuesta en frecuencia continúa de este filtro modificado se ilustra en la figura. 3.31, en tanto que sus coeficientes se listan en la tabla 3.8.

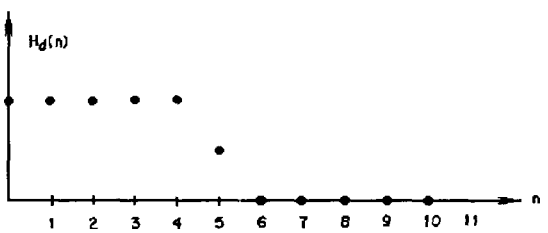

Figura 3.30 Respuesta en magnitud en frecuencia discreta con una muestra en la banda de transición entre los niveles de la banda de paso y la de rechazo ideales.

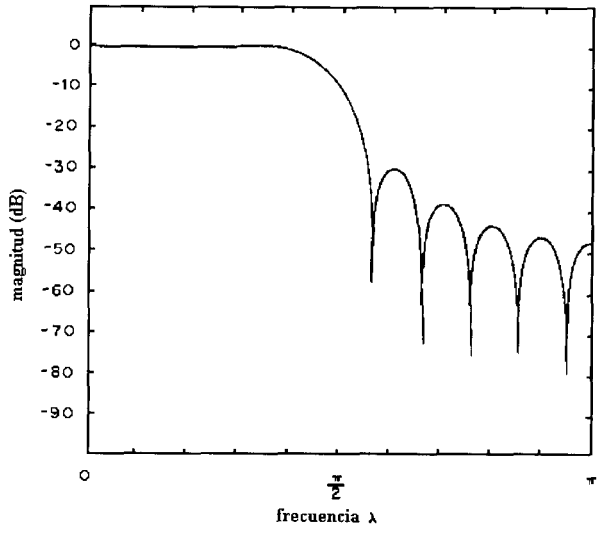

Figura 3.31 Respuesta en magnitud en frecuencia continua
--correspondiente a la respuesta en frecuencia discreta de la fig. 3.29

h[0]	=	h[20]	=	0.002427
h[1]	=	h[19]	=	0.008498
h[2]	=	h[18]	=	-0.010528
h[3]	=	h[17]	=	-0.023810
h[4]	=	h[16]	=	0.016477
h[5]	=	h[15]	=	0.047773
h[6]	=	h[14]	=	-0.020587
h[7]	=	h[13]	=	-0.096403
h[8]	=	h[12]	=	0.023009
h[9]	=	h[11]	=	0.315048
h[10]	=	0.476190		

Tabla 3.8 Coeficientes para el filtro de 21 derivaciones
con una banda de transición simple muestreada a un valor de 0.5

Como se puede apreciar, el nivel de rizo en la banda de rechazo se ha reducido por 13.3 *dB*. Por supuesto, se puede obtener una reducción mayor si el valor, o los valores, incorporados en la banda de transición se optimizaran, en vez de usar arbitrariamente el valor medio entre los niveles en la banda de paso y la banda de rechazo. Es de señalar que los métodos para optimizar los valores de los puntos en la banda de transición son iterativos e involucran un cálculo repetido de los coeficientes de la respuesta al impulso y la respuesta en frecuencia correspondiente; por consiguiente, consideramos conveniente examinar algunos de los detalles matemáticos que soportan este proceso.

3.3.1 N PAR CONTRA N IMPAR

Considérese la respuesta que se muestra en la figura 3.32,.

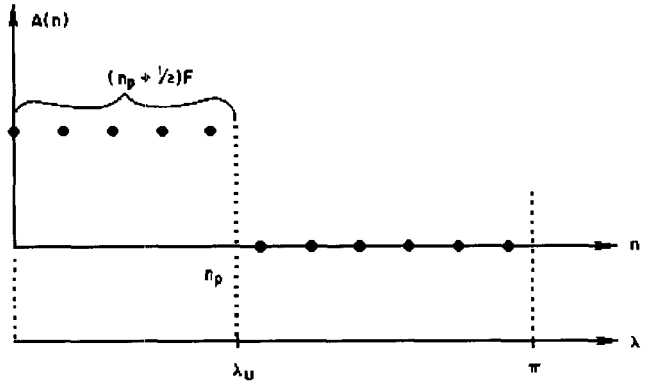

Figura 3.32 Respuesta muestreada en frecuencia para un filtro de -longitud impar sin muestras en la banda de transición

para el caso de un filtro de longitud impar sin banda de transición. Si asumimos que el corte esta centrado entre $n = n_p$ y $n = n_p + 1$, la frecuencia de corte queda en $2\pi F(n_p + \frac{1}{2})$, donde F es el intervalo entre las muestras en el dominio de la frecuencia. Para el caso normalizado, donde $T = 1$, encontramos que $F = 1/N$, por lo que la frecuencia de corte esta dada por:

$$\lambda_u = \frac{\pi(2n_p + 1)}{N} \qquad (3.48)$$

Esta ecuación nos permite calcular la frecuencia de corte, cuando n_p y N son conocidas. Sin embargo, en la mayoría de las situaciones necesitaremos empezar con valores de N y λ_u conocidos (deseados), y entonces proceder a determinar n_p. Se puede resolver la ecuación anterior para n_p dando un valor arbitrario a λ_u, y como el resultado puede ser no entero la solución se puede dejar anotada como:

$$n_p = \left\lfloor \frac{N\lambda_{UD}}{2\pi} - \frac{1}{2} \right\rfloor \qquad (3.49)$$

donde λ_{UD} denota a λ_u deseada y el $\lfloor \bullet \rfloor$ denota el "floor" de la función, que trunca la parte fraccionaria de su argumento. La ecuación produce un valor para n_p que garantiza que el corte este en algún lugar entre n_p y $n_p + 1$, con un valor no necesariamente igual a. 0.5. La diferencia de $\Delta\lambda = |\lambda_U - \lambda_{UD}|$ es una indicación de que tan buenas son las elecciones de n_p y N (entre mas pequeña es $\Delta\lambda$, mejores son la elecciones.

Es una practica común asumir que la respuesta de corte se sitúa en el punto medio entre $n = n_p$ y $n = n_p + 1$. Si la respuesta de amplitud en frecuencia continua es una línea recta entre $A(n) = 1$ en $n = n_p$ el valor de la respuesta entre estos puntos será 0.5, sin embargo, cuando $A(n)$ es la respuesta en amplitud, la atenuación en el punto de corte asumido es de $6\,dB$; para lograr una atenuación de $3\,dB$, el

punto de corte debe asignarse de modo que quede 0.293 a la derecha de n_p, y 0.707 a la izquierda de n_p+1.

Si asumimos que el corte esta en $n_p+0.293$, la frecuencia de corte es $2\pi F(n_p+0.293)$ y el punto de corte normalizado queda especificado por:

$$\lambda_u = \frac{2\pi(n_p+0.293)}{N} \qquad (3.50)$$

El número de muestras requerida en la banda de paso (bilateral) es $2n_p+1$ donde

$$n_p = \left\lfloor \frac{F\lambda_{UD}}{2\pi} - 0.293 \right\rfloor \qquad (3.51)$$

Por conveniencia denotamos a λ_u de la ecuación 3.48 como λ_6 y la λ_u de la ecuación 3.50 como λ_3.

- **N PAR**

Consideremos la respuesta que se muestra en la figura.3.33.

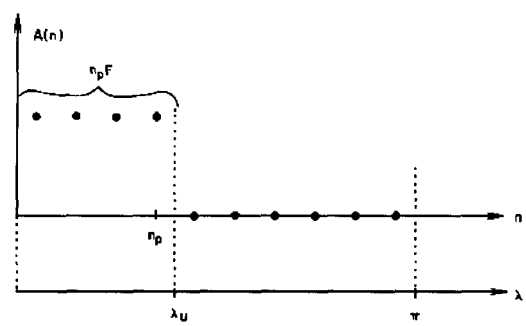

Figura 3.33 Respuesta de muestreo en frecuencia para un filtro de longitud par

para el caso de una longitud par, sin banda de transición. Si asumimos que el corte esta centrado entre $n=n_p$ y $n=n_p+1$, la frecuencia de corte es: $2\pi F n_p$, y la frecuencia normalizada correspondiente es:

$$\lambda_6 = \frac{2\pi n_p}{N} \qquad (3.52)$$

Resolviendo para n_p y usando la función "Floor", para garantizar valores enteros, obtenemos:

$$n_p = \left\lfloor \frac{n\lambda_{6D}}{2\pi} \right\rfloor \qquad (3.53)$$

Por otro lado, si asumimos que la frecuencia de corte esta en $n_p + 0.293$, entonces la frecuencia de corte se deberá considerar como $2\pi F(n_p - 0.207)$, en tanto que la frecuencia normalizada será:

$$\lambda_3 = \frac{2\pi(n_p - 0.207)}{N} \qquad (3.54)$$

De lo anterior resulta que el número requerido de muestras en la pasabanda (bilateral), $2n_p$, es:

$$n_p = \left\lfloor \frac{N\lambda_{3D}}{2\pi} + 0.207 \right\rfloor \text{ (multiplicar por 2 en ambos lados de la ecuación)}$$

Si las restricciones de procesamiento imponen un límite superior, $N_{\max}$, al numero total de derivaciones que pueden usarse en alguna situación particular, seria conveniente elegir entre $N = N_{\max}$ y $N = (N_{\max} - 1)$, supuesto que el valor de N proporciona una λ_U mas próxima a λ_{UD}. Por ejemplo, para $N_{máx} = 21$ y $\lambda_{6D} = 3\pi/7$, podemos determinar si $N = 21$ o $N = 20$ son la mejor opción dependiendo de los valores de $\Delta\lambda$:

Para $N = 20$

$$n_p = \left\lfloor \frac{20[(3\pi/7)]}{2\pi} \right\rfloor = \left\lfloor \frac{30}{7} \right\rfloor = 4$$

$$\lambda_6 = \frac{2\pi(4)}{20} = \frac{2\pi}{5}$$

$$\Delta\lambda = \left| \frac{3\pi}{7} - \frac{2\pi}{5} \right| = \frac{\pi}{35}$$

para $N = 21$,

$$n_p = \left\lfloor \frac{21[(3\pi/7)]}{2\pi} - \frac{1}{2} \right\rfloor = \lfloor 4 \rfloor = 4$$

$$\lambda_6 = \frac{9\pi}{21} = \frac{3\pi}{7}$$

$$\Delta\lambda = \left| \frac{3\pi}{7} - \frac{3\pi}{7} \right| = 0$$

Para este caso $N = 21$ es la mejor opción, proporcionando un valor de $\Delta\lambda = 0$.

Por otro lado, para $N_{máx} = 21$ y $\lambda_{3D} = 2\pi/5$, también podemos determinar si $N = 21$ o $N = 20$ son la mejor opción dependiendo de los valores de $\Delta\lambda$:

$$n_p = \left\lfloor \frac{20[(2\pi/5)]}{2\pi} + 0.207 \right\rfloor = \lfloor 4.209 \rfloor = 4$$

$$\lambda L3 = \frac{2\pi(4-0.207)}{20} = 1.1916$$

$$\Delta\lambda = \left| \frac{2\pi}{5} - 1.1916 \right| = 0.065$$

para $N = 21$,

$$n_p = \left\lfloor \frac{21[(2\pi/5)]}{2\pi} - 0.293 \right\rfloor = \lfloor 3.907 \rfloor = 3$$

$$\lambda_3 = \frac{2\pi(3.283}{21} = 0.9853$$

$$\Delta\lambda = \left| \frac{2\pi}{5} - 0.9853 \right| = 0.2714$$

En este caso, dado que $0.065 < 2.714$, la mejor opción parece ser $N = 20$.

Como ya lo hemos apuntado, la transformada inversa se puede usar para obtener los coeficientes $h(n)$ de la respuesta impulsiva, a partir de una respuesta de frecuencia deseada y especificada en frecuencias discretas uniformemente espaciadas. Sin embargo, Para el caso especial de filtros FIR con retardo de grupo constante, la transformación inversa se puede modificar de modo de tomar ventaja de las condiciones de simetría. Para el caso de frecuencia discreta, la TDF se puede adaptar de manera similar para obtener las fórmulas explícitas para $A(k)$ dadas en la tabla 3.9 (para el caso normalizado con $T = 1$), que muestra los coeficientes $h(n)$ para los cuatro retardos de grupo constantes para los filtros RIF. Las relaciones inversas correspondientes, formulas de diseño, se listan en la tabla 3.10, y han sido implementadas con la función en C **DisenoFiltro()** que se muestra al final del capitulo.

Tipo	
1 $h[n]$ simétrico N impar	$h[M] + \sum_{n=0}^{M-1} 2h[n]\cos\left[\frac{2\pi(M-n)k}{N}\right] = h[M] + \sum_{n=1}^{M} 2h[M-n]\cos\left(\frac{2\pi kn}{N}\right)$
2 $h[n]$ simétrico N par	$\sum_{h=0}^{(N/2)-1} 2h[n]\cos\left[\frac{2\pi(M-n)k}{N}\right] = \sum_{n=1}^{N/2} 2h\left[\frac{N}{2}-n\right]\cos\left\{\frac{2\pi k[n-(1/2)]}{N}\right\}$
3 $h[n]$ antisimétrico N impar	$\sum_{N=0}^{M-1} 2h[n]sen\left[\frac{2\pi(M-n)k}{N}\right] = \sum_{n=1}^{M} 2h[M-n]sen\left(\frac{2\pi kn}{N}\right)$
4 $h[n]$ antisimétrico N par	$\sum_{h=0}^{(N/2)-1} 2h[n]sen\left[\frac{2\pi(M-n)k}{N}\right] = \sum_{n=1}^{N/2} 2h\left[\frac{N}{2}-n\right]sen\left\{\frac{2\pi k[n-(1/2)]}{N}\right\}$

Tabla 3.9 Respuesta en amplitud de frecuencia de filtros RIF con retardo de grupo constante

Tipo	$h[n]$ $n = 0,1,2,..., N-1$
1 $h[n]$ simétrico N impar	$\frac{1}{N}\left\{A(0) + \sum_{k=1}^{M} 2A(k)\cos\left[\frac{2\pi(n-M)k}{N}\right]\right\}$
2 $h[n]$ simétrico N par	$\frac{1}{N}\left\{A(0) + \sum_{k=1}^{(N/2)-1} 2A(k)\cos\left[\frac{2\pi(n-M)k}{N}\right]\right\}$
3 $h[n]$ antisimétrico N impar	$\frac{1}{N}\left\{\sum_{k=1}^{M} 2A(k)sen\left[\frac{2\pi(M-n)k}{N}\right]\right\}$
4 $h[n]$ antisimétrico N par	$\frac{1}{N}\left\{A\left(\frac{N}{2}\right)sen[\pi(M-n)] + \sum_{k=1}^{(N/2)-1} 2A(k)sen\left[\frac{2\pi(M-n)k}{N}\right]\right\}$

Tabla 3.10 Formulas para el diseño de frecuencias muestreadas de filtros RIF con grupo de retardo constante

3.3.2 DISEÑO POR MUESTREO DE FRECUENCIA CON MUESTRAS EN LA BANDA DE TRANSICION

Como se menciono en la introducción de esta sección, la inclusión de una o más muestras en la banda de transición puede mejorar el desempeño de los filtros diseñados con el método de muestreo en frecuencia. En la sección 3.3 se obtuvo una mejoría colocando una muestra en la banda de transición, en el punto medio y con amplitud igual al promedio de las bandas contiguas, entre la amplitud unitaria de la banda de paso y el valor cero de la banda de rechazo. Sin embargo, se puede lograr una mejoría adicional si el valor de la muestra en la banda de transición se optimiza. Para lograr esta optimización podríamos ubicar la muestra que minimice el rizo de la banda de paso, minimizar el rizo de la banda de rechazo, o alguna función que dependa del rizo en ambas bandas; la aproximación más comúnmente usada es optimizar el valor de la banda de transición de modo de minimizar el pico del rizo de la banda de rechazo.

Para un conjunto de muestras correspondientes a una respuesta en amplitud requerida, la determinación del pico del rizo de la banda de rechazo supone los siguientes pasos:

1. A partir del juego de muestras de una respuesta en amplitud, H_d ,se debe calcular el correspondiente grupo de coeficientes h para la respuesta al impulso, usando la función en C **DisenoFiltro()**.

2. Para los coeficientes generados en el paso 1, calcular una aproximación en la frecuencia discreta a la respuesta en amplitud en frecuencia continúa usando la función en C. **RespuestaFir( )**.

3. Inspeccionar la respuesta en amplitud, generada en el paso 2, para encontrar el valor pico en la

banda de rechazo. Esta búsqueda puede realizarse usando la función en C **ValorPico( )**.

En general, se requerirá de cinco parámetros para especificar la ubicación de las banda(s) de rechazo, de modo que **ValorPico( )** "sepa" donde buscar. El primer parámetro especifica la configuración de la banda, pasabajas, pasaaltas, pasabanda o rechazabanda. Los otros parámetros son índices de las primeras y las últimas muestras en las bandas de paso y rechazo del filtro. Los filtros pasaaltas y pasabajas necesitan solo dos parámetros; n, y n_2, mientras que los filtros pasabanda y rechazabanda necesitan cuatro n_1, n_2, n_3 y n_4. El significado especifico de estos parámetros para las configuraciones de filtro básicas, se muestran en la figura 5.8. Para pasar los argumentos, hemos diseñado un mecanismo de entrada por arreglos, para la configuración especifica de un filtro, que será recibido por **conf_banda ( )** ,como sigue:

conf_banda[0] = 1 para pasabajas, 2 para pasaaltas, 3 para pasabanda y 4 para rechazabanda
conf_banda[0] = n_1
conf_banda[0] = n_2
conf_banda[0] = n_3
conf_banda[0] = n_4
conf_banda[0] = numero de derivaciones en el filtro.

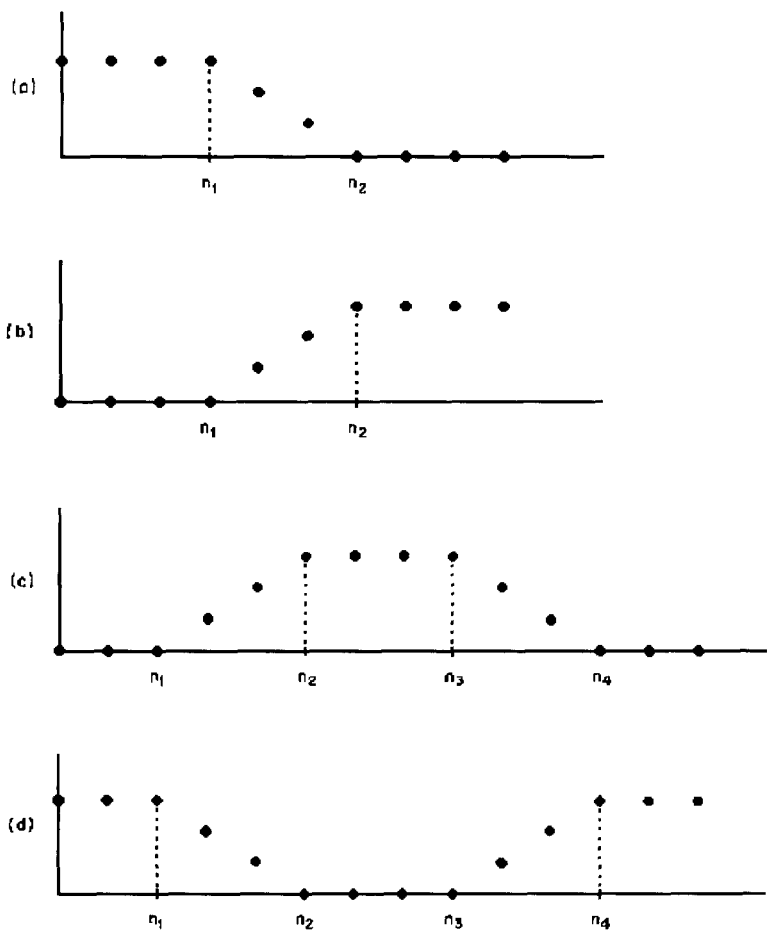

Figura 3.34. Parámetros de las configuraciones banda: (a) pasabajas, (b) pasaaltas, (c) pasabanda,y (d) rechazabanda

Como ejemplo, considérese el caso de un filtro pasabajas donde n_2 es el índice de la primera muestra en la banda de rechazo, de la respuesta deseada $H_d[N]$. El propósito es encontrar el valor piso de la banda de rechazo en la respuesta de magnitud continua en frecuencia del filtro. La computadora, por supuesto, calculará las muestras de una versión discreta en frecuencia, aproximación que no debe confundirse con la respuesta deseada $H_d[n]$, que también es una respuesta de magnitud discreta en frecuencia que contiene N muestras, con N el número de derivaciones en el filtro. La aproximación para la respuesta en frecuencia continúa debe contener un número mucho mayor de puntos. El número de muestras en la aproximación de la respuesta continúa (en un solo lado) es proporcionada a la función **ValorPico( )** como el argumento entero **numPts;** para los ejemplos que hemos trabajado, usamos valores para **numPts** en el rango de 120 a 480. En la localización del pico de una respuesta pasabajas, **ValorPico( )** dirige su atención a muestras n_s y más allá en la aproximación de la frecuencia discreta para la respuesta en amplitud en frecuencia continua, donde:

$$n_s = \frac{2Ln_s}{N} \tag{3.55}$$

con:

L = número de muestras en la aproximación (de un solo lado) a la respuesta continúa (**numPts**)

N = número de derivaciones en el filtro.

n_2 = índice de la primera muestra en la banda de rechazo deseada

Para filtros pasaaltas, pasabajas, y rechaza banda, la búsqueda queda limitada, de forma similar, a la de banda de paso.

- **Optimización**

Por otro lado, T_A podrá usarse para denotar el valor de la muestra en la banda de transición. Un método para optimizar el valor T_A consiste en comenzar con $T_A = 1$ y decrementar este valor por una cantidad fija, evaluando el pico del rizo de la banda de rechazo con cada decremento. Al principio, el rizo será decrementado cada vez que T_A disminuya ; sin embargo, una vez que el valor optimo el rizo aumentará si se continua decrementando el valor de T_A, obligándonos a proceder en sentido inverso y así sucesivamente hasta alcanzar el valor optimo que se considere suficiente, lo cual, por supuesto, supone una disminución del incremento cada vez que se cambia de dirección en la búsqueda del optimo. Una estrategia ligeramente más sofisticada para encontrar el valor óptimo de T_A se proporciona por el llamado algoritmo golden, basado en el supuesto de que el mínimo de una función $f(x)$ se conoce que esta delimitado por una tripleta de puntos $a < b < c$, tal que $f(b) < f(a)$ y f(b)<f(c). Una vez que el intervalo inicial es establecido, la anchura del intervalo puede ser continuamente decrementado hasta que los tres puntos a, b y c converjan a un valor mínimo sobre la abscisa. El nombre de este algoritmo proviene del hecho de que la búsqueda más eficiente resulta cuando el punto medio del intervalo esta a 0.67803 de la distancia a uno de los extremos del intervalo y a 0.38197 del otro extremo. El código para este algoritmo lo hemos dejado dispuesto en la función en C etiquetada con el nombre de **BusquedaMaxima()**, la cual invoca a las funciones **DisenoFiltro( )**, **RespuestaFir()**, **RespuestaNormalizada( ) ValorPico( )**, y **Transicion( )**, cuyos códigos son expuestos, también en este trabajo.

Para el caso de una muestra única esta función es sumamente simple; sin embargo, la mantendremos como una función separada para facilitar extensiones para el caso de muestras simples en la banda de transición que serán tratadas en la sección 3.3.2. La entradas aceptadas por **BusquedaMaxima( )** son las siguientes:

Firtype: 1 para N impar, $h[n]$ simétrico; 2 para N par, $h[n]$ simétrico; 3 para N impar $h[n]$ antisimétrico; 4 para N par y $h[n]$ antisimétrica.

numTaps: El número de derivaciones en el diseño de filtros RIF

Hd[ ]: Muestras en frecuencia positiva de la respuesta en magnitud deseada

tol: La tolerancia usada para terminar el algoritmo golden.

NumFregPts: el número de muestras en la aproximación de frecuencia discreta a la respuesta del filtro continuo en frecuencia.

bandconfig(): un arreglo que contiene la información de la configuración del filtro similar a findsbpeak().

La función proporciona dos salidas el valor del pico de la banda de rechazo de la respuesta en magnitud es proporcionada como el valor de retorno de la función, y la abscisa correspondiente (frecuencia) que es escrita dentro de ***fmin**.

Como ejemplo consideramos un filtro pasabajas de 21 derivaciones, encontrar el valor para el cual encontramos el valor para la muestra en la banda de transición $H_d[5]$, de modo que el valor pico en la banda de rechazo sea minimizado. En este caso, el valor óptimo para $H_d[5]$ es 0.400147, y la respuesta en amplitud correspondiente se muestra en la figura 3.35. Los coeficientes del filtro aparecen listados en la tabla 3.11; en tanto que la comparación con el caso donde $H_d[5] = 0.5$, nos permite observar que el pico del rizo en la banda de rechazo ha sido reducido por 11.2 dB.

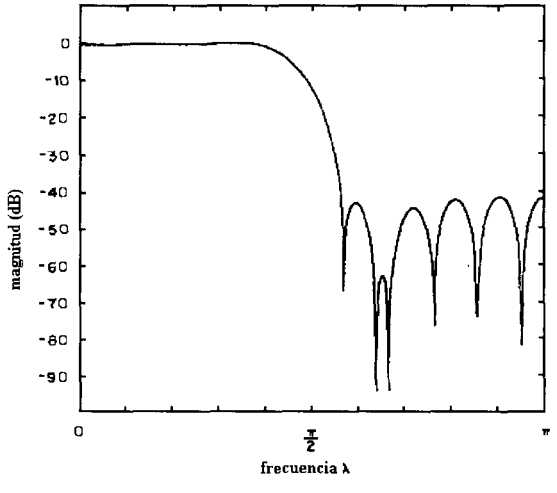

Figura 3.35 Respuesta en magnitud del filtro de 21 derivaciones.

h[0]	=	h[20]	=	0.009532
h[1]	=	h[19]	=	0.002454
h[2]	=	h[18]	=	-0.018536
h[3]	=	h[17]	=	-0.018963
h[4]	=	h[16]	=	0.025209
h[5]	=	h[15]	=	0.044232
h[6]	=	h[14]	=	-0.029849
h[7]	=	h[13]	=	-0.094246
h[8]	=	h[12]	=	0.032593
h[9]	=	h[11]	=	0.314324
h[10] = 0.466498				

Tabla 3.11 Coeficientes para el filtro con 21 derivaciones

3.3.3 OPTIMIZACION CON DOS MUESTRAS EN LA BANDA DE TRANSICION

El problema de optimización es un poco más difícil cuando se presentan dos o más muestras en la banda de transición. Consideremos el caso de un filtro pasabajas de tipo I con 21 derivaciones, para el cual se ha establecido una respuesta deseada especificada por:

$$H_d[n] = \begin{cases} 1.0 & 0 \le |n| \le 4 \\ H_B & |n| = 5 \\ H_A & |n| = 6 \\ 0.0 & 7 \le |n \le 10| \end{cases}$$

Donde los valores de H_A y H_B tendrán que ser optimizados para producir el filtro con el pico de rizo más pequeño en la banda de rechazo.

1. Haciendo $H_B = 1$ y usando una tolerancia tope de 0.01 en la función **BusquedaMaxima( )** para una sola muestra, encontramos que el pico de rizo en la banda de rechazo es minimizado para $H_A = 0.398227$. De este modo hemos definido un punto en el plano $H_A - H_B$, específicamente $(H_{A1} = 0.398227,\ H_{B1} = 1.0)$.

2. Definimos un segundo punto en el plano haciendo $H_B = 0.97$ y una vez más ubicamos el valor óptimo de H_A que minimice el valor pico del rizo de la banda de rechazo. Esto resulta en un segundo punto en (0.376941, 0.97).

3. Los dos puntos (0.398227, 1) y (0.376941, 0.97) pueden usarse para definir una línea en el plano $H_A - H_B$ como se muestra en la figura 3.35. Nuestra última meta es determinar el par ordenado (H_A, H_B) que minimiza el pico del rizo en la banda de rechazo del filtro. En la vecindad de $(H_{A1},1)$ la línea mostrada en la figura 3.35 representa la mejor trayectoria sobre la cual se desarrolla la búsqueda, por lo que sele conoce como línea de "mejor descenso" un recurso de utilidad intermedia consiste encontrar el punto, sobre la línea, en el que el rizo del filtro en la banda de rechazo es minimizado. A fin de usar el algoritmo de búsqueda para una sola muestra, sobre esta línea, podemos definir posiciones sobre la línea en término de sus proyecciones hacia el eje. Para

evaluar la respuesta del filtro para un valor dado de H_A, necesitamos tener a H_B expresado como una función de H_A. La pendiente de la línea se determina fácilmente a partir de los puntos 1 y 2 como:

$$m = \frac{1-0.97}{0.398227-0.376941} = 1.4093$$

pudiendo escribir

$$H_A = 1.4093H_A + b \qquad (3.56)$$

donde b es la intersección H_B.

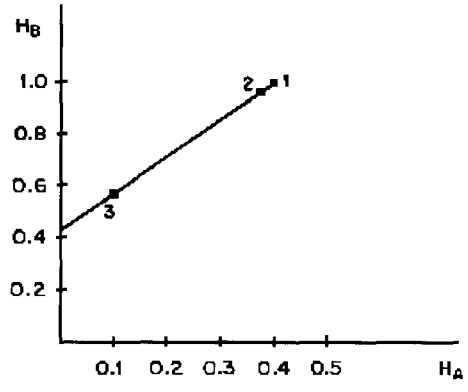

Figura 3.36. Línea de la mejor trayectoria graficada --en el plano H_A-H_B

Resolviendo para b por sustitución de los valores H_A, H_B en el punto 1, se obtendrá

$$b = H_B - 1.4093H_A$$
$$b = 1 - 1.40938(0.398227) = 0.438779$$

De esta forma la línea de mejor descenso es definida en el plano $H_A - H_B$ como:

$$H_B = 1.4093\ H_A + 0.438779 \qquad (3.57)$$

La naturaleza del problema en el diseño de filtros requiere que $0 \le H_A \le 1$ y $0 \le H_B \le 1$; además, la relación 3.57 supone que $H_B < H_A$ para todos los valores entre cero y la unidad. De este modo, el hecho de que H_B no debe exceder la unidad puede ser usado para lograr la restricción de los valores de H_A.; así, encontramos que $H_B = 1$ para $H_A = 0.39823$.Por consiguiente la búsqueda a lo largo de la línea queda limitada a los valores de H_A, con $0 \le H_A \le 0.39823$ el punto a lo largo de la línea (ecuación 3.57) para el cual el rizo en la banda de rechazo es minimizado se ha encontrado que es (0.099248, 0.57863), en tanto que el nivel de atenuación en este punto es -66.47 $dB's$.

4. El nivel de rizo de -66.47 *dB* es adecuado, pero no es el mejor que se puede lograr. La línea recta que se muestra en la figura 5.10 es justamente una extrapolación de los puntos 1 y 2. Generalmente, la trayectoria de descenso más pronunciada no será una línea recta y divergirá más lejos de la línea extrapolada, conforme la distancia del punto 1 se incremente. Así, cuando encontramos el punto óptimo (punto 3) sobre la línea recta, realmente no hemos encontrado el punto óptimo en general. Una manera de tratar esta situación es mantener H_B constante en el valor correspondiente al punto 3, y encontrar el valor óptimo de H_A. Sin que necesariamente H_A caiga sobre la línea, lo cual resulta en el punto 4 (0.98301, 057863), como se muestra en la figura 3.37.

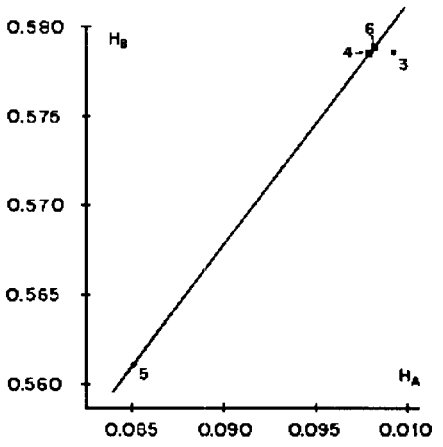

Figura 3.37. La mejor segunda línea de descenso

5. Si ahora modificamos H_B, para tomar el 97% del valor correspondiente del punto 4 (que es, H_B = (0.97) (0.57863) = (0.561271), y buscamos el valor de H_A que minimice el pico del rizo en la banda de rechazo, obtenemos el punto 5 (0.085145, 0.561271).

6. Los dos puntos, (0.099248, 0.57863) y (0.085145, 0.561271), pueden entonces ser usados para definir la nueva línea de descenso, mostrada en la figura 3.37. Usando el acercamiento referido en el paso 3, encontramos el punto, sobre la línea, en el cual el pico del rizo en la banda de rechazo es minimizado. Este punto es encontrado para ser (0.098592, 0.579014), y el pico de rizo correspondiente resulta en -69.680885 *dB*.

7. Finalmente, podemos continuar este proceso para definir líneas de descenso más pronunciadas y optimizar a lo largo de la línea, hasta rebasar y/o alcanzar un límite preescrito. Típicamente, la optimización se termina cuando el pico del rizo cambia por menos de 0.1 *dB* entre iteraciones; usando este criterio, el presente diseño converge después de la cuarta línea de descenso hasta encontrar el punto (H_A = 0.098403, H_B =0.579376) donde el pico del rizo de la banda de rechazo es - 71.08 *dB*.

3.3.4 CONSIDERACIONES DE PROGRAMACIÓN

Optimizar el valor de H_A, con H_B expresado como una función de H_A, requiere algunos cambios de manera que la función **ValorPico( )** interface con la función **BusquedaMaxima( )**. En el caso de una transición de una muestra, la búsqueda fue conducida con H_A como la variable independiente proporcionada. (en la localización apropiada de *Hd*[ ]) a **ValorPico( )**. Para el caso de una transición con dos muestras, el programa se diseño para conducir la búsqueda en términos del desplazamiento ρ medido sobre una línea arbitraria. Por otro lado, la función **ValorPico( )** "espera" tener el valor H_A y H_B dentro de las posiciones apropiadas en el arreglo *Hd*[ ]; en tanto que la función **BusquedaMaxima2()** (cuyo código se presenta al final de este capitulo) ha sido modificada para incluir la invocación a **Transicion( )** antes de cada llamada a **ValorPico( )**. La función **Transicion( )** mostrada en la figura 5.6 acepta ρ como una entrada para resolver esta dentro de los componentes H_A y H_B necesarias para **ValorPico( )** para calcular la respuesta al impulso y la subsecuente estimación de la respuesta en amplitud en frecuencia continúa. La línea sobre la cual ρ es medida es especificada a **Transicion( )** vía los arreglos **Origen**[ ] y **Puntos**[ ].

Los valores correspondientes de H_A y H_B para $\rho = 0$ son pasados a **origen(1)** y **origen(2)** respectivamente. Los cambios en H_A y H_B que correspondientes a $\Delta\rho = 1$ son pasados a **puntos**[1] y **puntos**[2] respectivamente. Hacer **puntos**[1] = 1 y **puntos**[1] = 0 es la manera correcta para especificar $H_P = \rho$. (Note que si hacemos **puntos**[1]=1, **puntos**[1]= 0, **puntos**[2]=0 y **puntos**[2]=0, el caso de una sola muestra puede ser manejado como un caso especial del caso de dos muestras, dado que estos valores son equivalentes a hacer $H_A = \rho$ y $H_B = 0$. Las iteraciones de la estrategia de optimización son realizadas por la función **optimizacion2( )**. Después de cada llamada a **BusquedaMaxima2( )**, la función **optimizacion2( )** usa la función **dumpRectComps( )** para imprimir las proyecciones de H_A y H_B retornadas por **BusquedaMaxima2( )**.

Como se mencionó previamente, cuando **BusquedaMaxima2( )** es usada con una tolerancia de 0.01 el diseño del filtro converge después de cuatro líneas de descenso; Cada línea involucra 3 puntos, 2 puntos para definir la línea y uno más donde el rizo esta minimizado. Las coordenadas y niveles del rizo de la banda de rechazo para los doce puntos del ejemplo diseñado son listados en la tabla 3.12. Cada uno de esos puntos requiere 8 iteraciones de **BusquedaMaxima2()**. Por otro lado, los coeficientes de la respuesta impulso para el filtro correspondiente a los valores de la banda de transición, $H_A = 0.098403$ y $H_B = 0.579376$ están listados en la tabla 5.7, en tanto que la respuesta de magnitud es dibujada en la figura 3.38.

Iteración	H_A	H_B	Pico de la banda de rechazo en dB
1	0.398227	1.0	-42.22
2	0.376941	0.97	-42.76
3	0.099248	0.578630	-66.47
4	0.098301	0.578630	-69.93
5	0.085145	0.561271	-65.87
6	0.098592	0.579014	-69.68
7	0.098301	0.579014	-71.05
8	0.085145	0.561643	-65.20
9	0.098473	0.579241	-70.89
10	0.098301	0.579241	-71.02
11	0.085145	0.561864	-64.61
12	0.098403	0.579376	-71.08

Tabla 3.12. Puntos generados en la optimización del ejemplo de 21 derivaciones.

h[0]	=	h[20]	=	0.002798
h[1]	=	h[19]	=	0.004783
h[2]	=	h[18]	=	-0.006541
h[3]	=	h[17]	=	-0.018285
h[4]	=	h[16]	=	0.007862
h[5]	=	h[15]	=	0.042175
h[6]	=	h[14]	=	-0.007896
h[7]	=	h[13]	=	-0.092308
h[8]	=	h[12]	=	0.007530
h[9]	=	h[11]	=	0.313553
h[10]	=	0.492659		

Tabla 3.13 Coeficientes de respuesta al impulso --para el filtro 21 derivaciones.

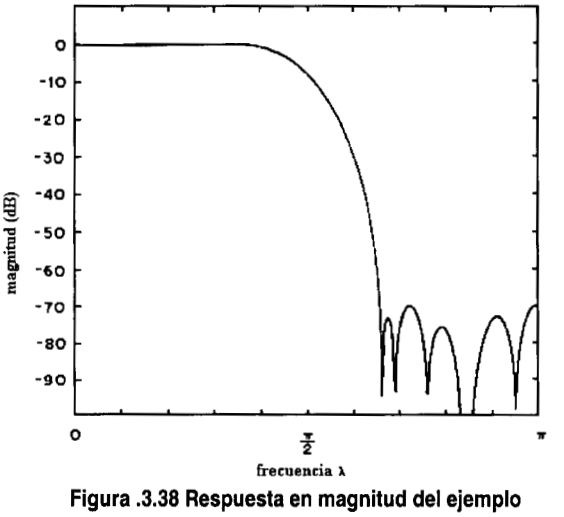

Figura .3.38 Respuesta en magnitud del ejemplo

Un examen cuidadoso de los valores en la tabla 3.12 revelan varias anomalías. Los puntos 1, 2, 4, 5, 7, 8, 10 y 11 definen líneas del descenso más pronunciadas; y los puntos 3, 6, 9 y 12 son los puntos óptimos correspondientes sobre estas líneas. El desempeño del rizo del punto óptimo número 6 es -69.68 , mientras que el desempeño del punto 4 es - 69.93. Estos dos puntos caen sobre la misma línea, y el desempeño del punto 4 es mejor que el del punto 6. Una situación similar ocurre en los puntos 7 y 9, tal conducta indica que el criterio de paro para **BusquedaMaxima2( )** no es bastante severo, permitiendo que la búsqueda se detenga antes de alcanzar punto sobre la línea.

Como último ejemplo de este capitulo tenemos un de un filtro de 21 pulsos el cual queremos diseñar por el metodo de muestreo en frecuencia. Los X(n) = {1,1,5,5,5,0,0,0,9,1,4,7,6,8,3,3,3,0,1,12}.

```
                    MENU PRINCIPAL

          1. FILTROS ANALOGICOS
          2. TRANSFORMADA DISCRETA DE FOURIER
          3. DISEÑO DE FILTROS FIR
          4. DISEÑO DE FILTROS IIR
          5. SALIR DEL PROGRAMA

              Elija su OPCION:
```

Elige la opción 3

```
                DISEÑO DE FILTROS RIF

          1. RESPUESTA DE FILTROS RIF
          2. POR SERIES DE FOURIER
          3. POR MUESTREO EN FRECUENCIA
          4. SALIR

             Elija su OPCION:  _
```

Elige la opción 3

```
DISENO DE FILTRO RIF POR METODO DE MUESTREO

                TIPO DE FILTRO

      1. Simetrico  PAR,  longitud IMPAR
      2. Simetrico  PAR,  longitud PAR
      3. Simetrico IMPAR, longitud IMPAR
      4. Simetrico IMPAR, longitud PAR
      5. Regresar al menu

         opcion: 1

      Numero de puntos: 21


  oprima una tecla para introducir los valores de los puntos... _
```

Elige la opción 1 y escribe el numero de puntos .

```
                    DISEÑO POR METODO DE MUESTREO

              Valor del punto x[0]= 1
              Valor del punto x[1]= 1
              Valor del punto x[2]= 5
              Valor del punto x[3]= 5
              Valor del punto x[4]= 5
              Valor del punto x[5]= 0
              Valor del punto x[6]= 0
              Valor del punto x[7]= 0
              Valor del punto x[8]= 9
              Valor del punto x[9]= 1
              Valor del punto x[10]= 4
              Valor del punto x[11]= 7
              Valor del punto x[12]= 6
              Valor del punto x[13]= 8
              Valor del punto x[14]= 3
              Valor del punto x[15]= 3
              Valor del punto x[16]= 3
              Valor del punto x[17]= 0
              Valor del punto x[18]= 1
              Valor del punto x[19]= 1
Valor del punto x[20]= 2_
```

escribe los puntos que se te piden en el problema.

```
        DISEÑO DE FILTRO RIF POR METODO DE MUESTREO

Los coeficientes obtenidos son:

h[0]= 0.693319                  h[14]= -1.214110
h[1]= -0.750861                 h[15]= 0.421263
h[2]= 0.711852                  h[16]= 0.152175
h[3]= -0.523810                 h[17]= -0.523810
h[4]= 0.152175                  h[18]= 0.711852
h[5]= 0.421263                  h[19]= -0.750861
h[6]= -1.214110                 h[20]= 0.693319
h[7]= -0.687028
h[8]= 0.332708
h[9]= -0.087889
h[10]= 2.904762
h[11]= -0.087889
h[12]= 0.332708
h[13]= -0.687028

   Oprima una tecla para continuar...
```

estos son los resultados que despliega el programa.

LISTADO DE PROGRAMAS DE FILTROS DIGITALES

```
/*-------------------------------------------------------------------------*/
/*                                    | Este programa permite calcular   */
/*   Programa 3.1                     | la respuesta al impulso finito   */
/*                                    | de un filtro de orden n, cuya    */
/*   Respuesta al impulso finito ()   | simetria  y/o longitud sea       */
/*                                    | para o impar                     */
/*              FIR                   |                                  */
/*-------------------------------------------------------------------------*/

#include <conio.h>

void respuestaImpulFinita(int tipofiltro,int numbTaps,real hh[],
                          logical escaladB, int numeroDPuntos, real Hd[])
{
   int index, L, n, y;
   real lambda, work;

   clrscr();
   gotoxy(28,2);printf("RESPUESTA AL IMPULSO FINITO");
   gotoxy(32,4);printf(" Los resultados son: ");

   for( L=0; L<=numeroDPuntos-1; L++){
        lambda = L * PI / (real) numeroDPuntos;
        switch (tipofiltro) {
             case 1:                              // simetria par longitud impar
                    work = hh[(numbTaps-1)/2];
                    for( n=1; n<=((numbTaps-1)/2); n++) {
                         index = (numbTaps-1)/2 - n;
                         work = work + 2.0 * hh[index] * cos(n*lambda);
                    }break;
              case 2:                            // simetria par longitud par
                    work = 0.0;
                    for( n=1; n<=(numbTaps/2); n++) {
                         index = numbTaps/2-n;
                         work = work + 2.0 * hh[index] * cos((n-0.5)*lambda);
                    }break;
              case 3:                           // simetria impar longitud impar
                    work = 0.0;
                    for( n=1; n<=((numbTaps-1)/2); n++) {
                         index = (numbTaps-1)/2 - n;
                         work = work + 2.0 * hh[index] * sin(n*lambda);
                    }break;
              case 4:                            // simetria impar longitud par
                    work = 0.0;
                    for( n=1; n<=(numbTaps/2); n++) {
                         index = numbTaps/2-n;
                         work = work + 2.0 * hh[index] * sin((n-0.5)*lambda);
                    }break;
          } //fin del case

        y=6+L;
        if(escaladB){                                    // escala en dBs.
             Hd[L] = 20.0 * log10(fabs(work));
             gotoxy(30,y);printf(" Hd[%d] =  %lf",L,Hd[L]);
             y++;
        }
        else{                                            // escala normal
             Hd[L] = fabs(work);
             gotoxy(30,y);printf(" Hd[%d] =  %lf",L,Hd[L]);
              // imprime resultado
             y++;
        }
        if(!(L%10)) printf("%3d\r",numeroDPuntos-L);
    }/* fin del for */
    gotoxy(21,y+4);printf("Oprima una tecla para continuar");
    getche();
    return;
}
```

```
/*-------------------------------------------------------------------------*/
/*                                |   Este programa permite normalizar     */
/*   Programa 3.2                 |   la  respuesta al impulso  obtenida    */
/*                                |   con el algoritmo anterior.            */
/*   Respuesta Normalizada()      |                                         */
/*                                |                                         */
/*-------------------------------------------------------------------------*/

void respuestaNormalizada( logical escaladB, int numPts, real H[])
{
   int n,i;
   real biggest;

   clrscr();i=5;
   gotoxy(30,3); printf("La respuesta normalizada es: ");

   if( escaladB ){                    //respuesta normalizada en dBs
        biggest = -100.0;
        for( n=0; n<=numPts-1; n++){
            if(H[n]>biggest)
               biggest = H[n];
            //gotoxy(15,i); printf("H[%d]=  %lf",n,H[n]);
            //i++;
        }
        for( n=0; n<=numPts-1; n++){
             H[n] = H[n]-biggest;
             gotoxy(30,i); printf(" H[%d] =  %lf",n,H[n]);
             i++;
        }
   }
   else{                              //respuesta normalizada en escala lineal
       biggest = 0.0;
       for( n=0; n<=numPts-1; n++){
            if(H[n]>biggest)
               biggest = H[n];
      }
       for( n=0; n<=numPts-1; n++){
            H[n] = H[n]/biggest;
            gotoxy(30,i); printf(" H[%d] =  %lf", n, H[n]);
            i++;
      }
   }
   gotoxy(21,i+3);printf("Oprima una tecla para continuar");
   getche();
   return;
}
```

```
/*-------------------------------------------------------------------------*/
/*                          |                                              */
/*   Programa 3.3           | Este algoritmo permite calcular la           */
/*                          | aproximación RIF  para un                    */
/*   pasaBajasIdeal()       | filtro digital  ideal pasabajas              */
/*                          |                                              */
/*-------------------------------------------------------------------------*/

void pasaBajasIdeal(   int numbTaps, real lambdaU, real hh[])
{
   int n,nMax,yy,xx,b;
   real mm;

    if(numbTaps<15) {
       xx=30;}
   else {
       xx=10;
   }
   yy=6;
   b=0;
   gotoxy(23,3);
   printf("PARA UN FILTRO PASA BAJAS IDEAL");
   for( n=0; n<numbTaps; n++){
        mm = n - (real)(numbTaps-1)/2.0;
        if( mm==0 )
               {hh[n] = lambdaU/PI;}
        else
               {hh[n] = sin(mm * lambdaU)/(mm * PI);}      //Se calculan los
                                                  //coeficientes de la respuesta al
                                                  //impulso aplicando la ec. 3.14
        gotoxy(xx,yy);
        printf (" hh[%d] = %lf  ",n,hh[n]);
        yy=yy+1;                        //imprime el resultado de cada coeficiente
        if(yy==20) {
          b=1;
          xx=45;
          yy=6;
        }
   }

   if(b==1)
      yy=22;
   else
      yy=yy+2;
   gotoxy(16,yy);
   printf("Oprima una tecla para continuar con un F. PASA ALTAS");
   getch();
   return;
}
```

```
/*-------------------------------------------------------------------------*/
/*                              | Este algoritmo permite calcular la       */
/*   Programa 3.4               | aproximación RIF  para un                */
/*                              | filtro digital  ideal pasa altas         */
/*   PasaAltasIdeal()           |                                          */
/*                              |                                          */
/*-------------------------------------------------------------------------*/


void pasaAltasIdeal( int numbTaps, real lambdaL, real hh[]){
   int n,nMax,yy,xx,b;
   real mm;

   if(numbTaps<15)
     {xx=30;}
   else{
      xx=10;
   }
   gotoxy(23,4);   printf("PARA UN FILTRO PASA ALTAS IDEAL");
   yy=6;

        // Se calculan los coeficientes de la respuesta al
        // impulso aplicando la ec. 3.16

   for( n=0; n<numbTaps; n++) {
         mm = n - (real)(numbTaps-1)/2.0;
         if(mm==0)
               {hh[n] = 1.0 - lambdaL/PI;}
         else
               {hh[n] = -sin(mm * lambdaL)/(mm * PI);}
         gotoxy(xx,yy);
         printf (" hh[%d] =  %lf  ",n,hh[n]);
         yy++;
   if(yy==20){
         b=1;
         xx=45;
         yy=6;
   }
}

if(b==1)
      yy=22;
else
      yy=yy+2;

   gotoxy(16,yy+2);  printf("Oprima una tecla para continuar con un F PASABANDA");
   getch();
   return;
}
```

```
/*-------------------------------------------------------------------------*/
/*                          | Calcula los coeficientes de la respuesta     */
/*   Programa 3.5           | al impulso para la aproximación RIF          */
/*                          | de un filtro digital ideal pasa banda        */
/*   BandaDepasoIdeal()     |                                              */
/*                          |                                              */
/*-------------------------------------------------------------------------*/


void pasaBandaIdeal(  int numbTaps, real lambdaL,  real lambdaU,  real hh[])
{
   int n,nMax,yy,xx,b;
   real mm;

  if(numbTaps<15)
     {xx=30;}
   else{
     xx=10; }
   gotoxy(23,4);
   yy=6;
   printf("PARA UN FILTRO PASA BANDA IDEAL");

        // Se calculan los coeficientes de la respuesta al
        // impulso para un filtro pasabanda ideal
        // aplicando la ec. 3.17

   for( n=0; n<numbTaps; n++){
         mm = n - (real)(numbTaps-1)/2.0;
         if(mm==0)
             {hh[n] = (lambdaU - lambdaL)/PI;}
         else{
             hh[n] = (sin(mm * lambdaU) - sin(mm * lambdaL))/(mm * PI);}

        // Se imprime cada coeficiente en pantalla
         gotoxy(30,yy);
         printf (" hh[%d] =  %lf  ",n,hh[n]);
         yy++;
         if(yy==20){
            b=1;
            xx=45;
            yy=6;
         }
   }
   if(b==1)
      yy=22;
   else
       yy=yy+2;
   gotoxy(16,yy);
   printf("Oprima una tecla para continuar con un F RECHAZA BANDA");
   getch();
   return;
}
```

```
/*------------------------------------------------------------------------ -*/
/*                            |                                             */
/*   Programa 3.6             | Programa que calcula los coeficientes de la */
/*                            | respuesta al impulso para la aproximación   */
/*   BandaDRechazoIdeal       | RIF para un filtro banda de rechazo ideal   */
/*                            |                                             */
/*-------------------------------------------------------------------------*/

void rechazaBandaIdeal( int numbTaps, real lambdaL, real lambdaU, real hh[]){
   int n,nMax,yy,xx,b;
   real mm;
   if(numbTaps<15)
     {xx=30;}
   else{
      xx=10;
   }

   gotoxy(23,4);
   printf(" PARA UN FILTRO EN LA BANDA DE RECHAZO ");
   yy=6;

        // Se calculan los coeficientes de la respuesta al
        // impulso para un filtro rechaza banda
        // aplicando la ec. 3.18


   for( n=0; n<numbTaps; n++) {
         mm = n - (real)(numbTaps-1)/2.0;
         if(mm==0)
               {hh[n] = 1.0 + (lambdaL - lambdaU)/PI;}
         else
               {hh[n] = (sin(n * lambdaL) - sin(mm * lambdaU))/(mm * PI);}
         gotoxy(xx,yy);
         printf (" hh[%d] =  %lf  ", n, hh[n]);
         yy++;
         if(yy==20){
            b=1;
            xx=45;
            yy=6;
       }
   }
   if(b==1)
      yy=22;
   else
      yy=yy+2;
    gotoxy(16,yy+2);  printf("    Oprima una tecla para continuar ...");
   getch();
   return;
}
```

```
/*-------------------------------------------------------------------------*/
/*                                |                                        */
/*   Programa 3.7                 | Este programa permite calcular la      */
/*                                | respuesta impulsiva multiplicada por   */
/*   respuestaRectangular()       | una ventana rectangular para reducir   */
/*                                | el fenomeno de Gibbs                   */
/*-------------------------------------------------------------------------*/

#define TINY 3.16e-5
real respuestaRectangularCont( real freq, real tau, int escaladB)
{
   real x;

   x =tau* sinc(PI * freq * tau);
   if( escaladB)      {
        if(fabs(x) < TINY)
              x = -90.0;
        else
              x = 20.0*log10(fabs(x));
   }
   return(x);
}
```

```
/*-------------------------------------------------------------------------*/
/*                                |                                        */
/*   Programa 3.8                 | Programa que calcula la magnitud  de   */
/*                                | la transformada de Fourier discreta    */
/*   respuestaRectangularDisc()   | en tiempo para una ventana rectangular */
/*                                |                                        */
/*-------------------------------------------------------------------------*/

real respuestaRectangularDisc( real freq, int M, int normalizedAmplitude)
{
   real result;

   if( freq == 0.0)

        //Se obtiene el resultado aplicando la ecuación 3.25

         result = (real) (2*M+1);
   else
        { result = fabs(sin(PI * freq * (2*M+1))/ sin( PI * freq));}
   if( normalizedAmplitude ) result = result / (real) (2*M+1);

   return(result);
}
```

```
/*-------------------------------------------------------------------------*/
/*                                |                                        */
/*   Programa  3.9                | Programa que calcula la respuesta      */
/*                                | en magnitud de una ventana triangular  */
/*   respuestaTriangularCont()    | continua en tiempo                     */
/*                                |                                        */
/*-------------------------------------------------------------------------*/


real respuestaTriangularCont( real freq, real tau, int escaladB)
{
       // Se calcula la Transformada de Fourier
       // aplicando la ecuación 3.28

   real amp0, x;
   amp0 = 0.5 * tau;
   x = PI * freq * tau / 2.0;
   x = 0.5 * tau * sincSqrd(x);
   if( escaladB) {
        if( fabs(x/amp0) < TINY)
            {x = -90.0;}
        else
            {x = 20.0*log10(fabs(x/amp0));}
   }
   return(x);
}
```

```
/*-------------------------------------------------------------------------*/
/*                                |                                        */
/*   Programa   3.10              | Programa  que permite calcular la      */
/*                                | Transformada de Fourier discreta en    */
/*   respuestaTriangularDisc()    | en tiempo de una ventana rectangular   */
/*                                |                                        */
/*-------------------------------------------------------------------------*/


real respuestaTriangularDisc(real freq, int M, int normalizedAmplitude)
{
   real result;

   if( freq == 0.0)
        { result = (real) M;}              //Se aplica la ecuacion 3.31a
   else {
         result = (sin(PI * freq * M) * sin(PI * freq * M)) /
                             (M * sin( PI * freq) * sin( PI * freq));
   }
   if( normalizedAmplitude )
       result = result / (real) M;
   return(result);
}
```

```
/*-------------------------------------------------------------------------*/
/*                                |                                        */
/*   Programa  3.11               | Imprime los resultados obtenidos       */
/*                                | de cada una de las ventanas trianagulares */
/*   ventanaTriangular()          | calculadas                             */
/*-------------------------------------------------------------------------*/

void ventanaTriangular( int N, real Ventana[]){
real offset,respuestaVentana[30];
int n,xx,yy;
offset = (real) (1-(N%2));
xx=25; yy=6;
gotoxy(17,4);
printf(" Los valores de la VENTANA RECTANGULAR son: ");

for(n=0; n<(N/2.0); n++)
        {
          Ventana[n] = 1.0 - (2.0*n + offset)/(N+1.0);
          gotoxy(xx,yy);
          printf(" ventana[%d]= %f",n,Ventana[n]);
          yy++;
        }
gotoxy(22,yy+2);
printf("Oprime una tecla para continuar...");
getch();
return;
}
```

```
/*-------------------------------------------------------------------------*/
/*   Programa  3.12        |                                               */
/*                           Calcula los coeficientes para una ventana LAG */
/*   makeLagVentana()      |                                               */
/*                         |                                               */
/*-------------------------------------------------------------------------*/

void makeLagVentana(  int N,ral Ventana[],int centro, real respuestaVentana[])
{
int n, M, xx, yy, negativo;
negativo=0;
xx=30; yy=6;
gotoxy(21,4);
printf(" Los valores de la VENTANA LAG son: ");
if(N%2) {
        M=(N-1)/2;
        for(n=0; n<=M; n++) {
                resultVentana[n] = Ventana[n];
                gotoxy(xx,yy);
                printf(" V. LAG[%d]= %f",n,resultVentana[n]);
                resultVentana[-n] = resultVentana[n];
                yy++;
        }
}
else {
        M=(N-2)/2;
        if(center == negative) {
                for( n=0; n<=M; n++) {
                        resultVentana[n] = Ventana[n];
                        resultVentana[-(1+n)] = Ventana[n];
                        gotoxy(xx,yy);
                        printf("V. LAG[%d]= %f",n,resultVentana[n]);
                yy++;
                }
        }
        else {
                for( n=0; n<=M; n++) {
                        resultVentana[n+1] = Ventana[n];
                        resultVentana[-n] = Ventana[n];
                        gotoxy(xx,yy);
                        printf("V. LAG[%d]= %f",n,resultVentana[n]);
                        yy++;
                }
        }
}
gotoxy(22,yy+2);
printf("Oprime una tecla para continuar...");
getch();
return;
}
```

```
/*-------------------------------------------------------------------------*/
/*                                                                         */
/*   Programa   3.13                                                       */
/*                                                                         */
/*   makeDataVentana()                                                     */
/*                                                                         */
/*-------------------------------------------------------------------------*/

void makeDataVentana( int N, real Ventana[], real respuestaVentana[]){

   int n,M, xx, yy;
   gotoxy(21,4);
   printf(" Los valores de la VENTANA DATA son: ");
   xx=28; yy=6;
   if(N%2) {
        M=(N-1)/2;
        for( n=0; n<=M; n++) {
              resultVentana[n] = Ventana[M-n];
              resultVentana[M+n] = Ventana[n];
              gotoxy(xx,yy);
              printf("V. DATA[%d]= %f",n,resultVentana[n]);
              yy++;
        }
   }
   else {
        M=(N-2)/2;
        for( n=0; n<=M; n++) {
              resultVentana[n] = window[M-n];
              resultVentana[M+n+1] = Ventana[n];
              gotoxy(xx,yy);
              printf("V. DATA[%d]= %f ",n,resultVentana[n]);
              yy++;
        }
   }
gotoxy(22,yy+2); printf("Oprime una tecla para continuar...");
getch();
return;
}
```

```
/*-------------------------------------------------------------------------*/
/*                          |                                              */
/*   Programa  3.14         | Calcula la respuesta en magnitud de una      */
/*                          | Von Hann en tiempo discreto                  */
/*   ventanaHann()          |                                              */
/*                          |                                              */
/*-------------------------------------------------------------------------*/

void ventanaHann( int N, real Ventana[])
{
   logical odd;
   int n, xx, yy ;
   odd = N%2;
   xx=26; yy=6;

    gotoxy(21,4);
   printf(" Los valores de la VENTANA HANN son: ");
   for( n=0; n<(N/2.0); n++){

        //Para realizar esta rutina se aplican las ecuaciones 3.37 y 3.38

         if( odd){
               Ventana[n] = 0.5 + 0.5 * cos(TWO_PI*n/(N-1));
               gotoxy(xx,yy);
               printf("V. Hann[%d]= %f",n,Ventana[n]);
               yy++;
          }
          else{
               Ventana[n] = 0.5 + 0.5 * cos(TWO_PI * (2*n+1)/(2.0*(N-1)));
               gotoxy(xx,yy);
               printf(" V. Hann[%d]= %f",n,Ventana[n]);
               yy++;

          }
   }
   gotoxy(22,yy+2);
   printf("Oprime una tecla para continuar...");
   return;
}
```

```
/*-------------------------------------------------------------------------*/
/*                                                                         */
/*   Programa  3.15        |  Este programa permite calcular la funcion de  */
/*                         |  la ventana Hamming en tiempo discreto         */
/*   ventanaHamming()      |                                                */
/*                                                                         */
/*-------------------------------------------------------------------------*/

void ventanaHamming( int N, real Ventana[])
{
logical odd;
int n, xx, yy;
odd = N%2;

gotoxy(10,4);
xx=28; yy=6;
gotoxy(23,4);
printf(" Los valores de la VENTANA HAMMING son: ");
   for(n=0; n<(N/2.0); n++)
        {
        if( odd)
                {Ventana[n] = 0.54 + 0.46 * cos(TWO_PI*n/(N-1));
                gotoxy(xx,yy);
                printf(" V. Hamming[%d]= %f",n,Ventana[n]);
                yy++;
        }
        else
                {Ventana[n] = 0.54 + 0.46 * cos(TWO_PI * (2*n+1)/(2.0*(N-1)));

                gotoxy(xx,yy);
                printf("V. Hamming[%d]= %f",n,Ventana[n]);
                yy++;
        }
   }
gotoxy(15,22);
printf("Oprima una tecla para continuar...");

return;
}
```

CAPITULO IV

FILTROS DIGITALES CON RESPUESTA AL IMPULSO INFINITO

La forma general para la respuesta de un filtro con respuesta al impulso infinito esta dado por:

$$y[n]=\sum_{n=1}^{N} a_n y[k-n]+\sum_{m=0}^{M} b_m x[k-m] \qquad (4.1)$$

Esta ecuación indica que la salida del filtro es una combinación lineal de la entrada presente, las M entradas previas y las N salidas previas. La función sistema correspondiente esta dada por (4.2)

$$H(z)=\frac{\sum_{m=0}^{M} b_m z^{-m}}{1-\sum_{n=1}^{N} a_n z^{-n}} \qquad (4.2)$$

donde al menos una de las a_n es diferente de cero y una de las raíces del denominador no es cancelada por una de las raíces del numerador.

Para un filtro estable, todos los polos de H(z) deben caer dentro de un circulo unitario, aunque los ceros pueden caer en cualquier parte del plano z. Es usual que M, el número de ceros, sea menor o igual que N el número de polos. Como quiera que sea, cuando el numero de ceros excede el numero de polos, el filtro puede ser separado en un filtro RIF con M-N derivaciones en cascada con un filtro RII con N polos y N ceros. De este modo, las técnicas de diseño quedan convencionalmente restringidas a los casos en que $M \leq N$.

Exceptuando el caso especial donde todos los polos caen sobre el circulo unitario, no es posible diseñar un filtro RII que tenga fase lineal; por tanto, a diferencia de los procedimientos de diseño de filtros RIF en donde lo que importa es la respuesta en magnitud, los procedimientos de diseño de los filtros RII tienen que ver tanto con la respuesta en magnitud como con la respuesta en fase.

4.1 RESPUESTA EN FRECUENCIA DE FILTROS RII

La respuesta en frecuencia de un filtro RII es calculada con los coeficientes a_n y b_m, haciendo uso de la siguiente expresión:

$$H[k]=\frac{\sum_{m=0}^{L-1} \beta_m \exp(j2\pi mk/L)}{\sum_{m=0}^{L-1} \alpha_n \exp(j2\pi mk/L)} \qquad (4.3)$$

donde

$$\alpha_n=\begin{cases}1 & n=0\\ -a_n & 0<n\leq N\\ 0 & N<n\end{cases}$$

$$\beta_m=\begin{cases}b_m & 0\leq m\leq M\\ 0 & M<m\end{cases}$$

Para visualizar las respuestas producidas por la relación anterior, hemos elaborado una rutina en C que aparece al final de este capitulo.

4.2 REALIZACIONES DE FILTROS RII

Una realización directa de la ecuación 4.1 se muestra en la figura 4.1. La estructura mostrada es conocida como la realización en forma directa I; el estudio de este diagrama revela que el sistema puede ser visto como dos sistemas en cascada - el primero usando $x[k-M]$ a través $x[k]$ para generar una señal intermedia, $w[k]$, y el segundo usando $w[k]$ y $y[k-N]$ a través de $y[k-1]$ para generar $y[k]$. Dado que estos dos sistemas son LIT, el orden de la cascada puede ser invertido para obtener el sistema equivalente mostrado en la figura 4.2. Un examen de esta figura, revela que los retardos unitarios que van del centro del diagrama hacia abajo pueden apareare de modo que una pareja de dos retardos pueda tomar la misma señal de entrada. Este hecho puede aprovechar para mezclar las dos cadenas de retardo en una sola, como se muestra en la figura 4.3, a la cual se le conoce como forma de realización directa II.

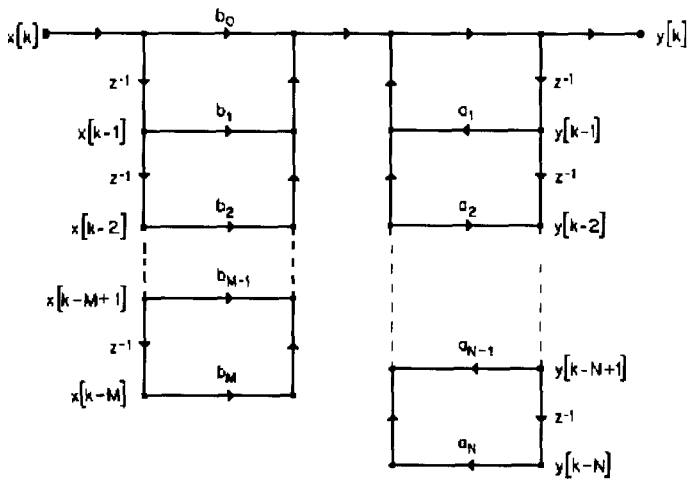

Figura 4.1 Diagrama de flujo de una estructura de forma directa I para un sistema RII

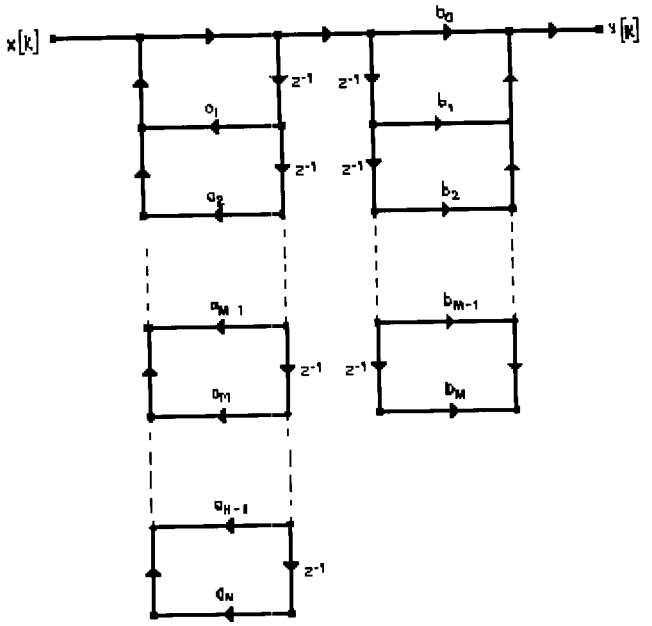

Figura 4.2 de flujo que muestra un orden inverso -en cascada de laDiagram figura 6.1

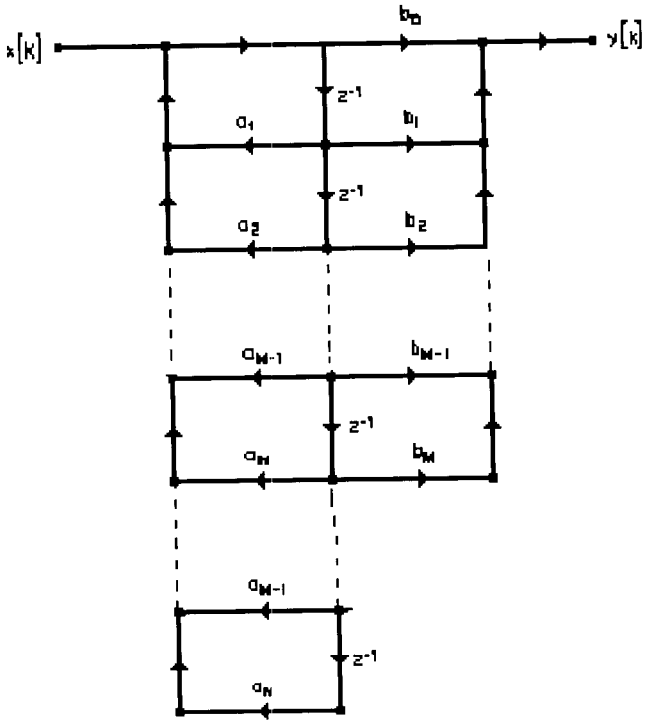

Figura 4.3 Diagrama de flujo de la señal de realización de la forma directa II para un sistema RII

4.3 INVARIANCIA AL IMPULSO

La idea básica detrás del método de invariancia al impulso es muy simple: la respuesta al impulso del filtro digital debe ser igual a la secuencia de muestras espaciadas uniformemente de la respuesta al impulso de un filtro analógico:

$$h[n] = h_a(nT) \tag{4.4}$$

Esta aproximación es conceptualmente sencilla, aunque su evaluación no es inmediata. Por definición. para un filtro con respuesta al impulsiva infinita, la secuencia $h[n]$ será diferente de cero sobre un rango infinito de n; además, dadas las características de mapeo del plano s al plano z, podemos concluir que la valuación de (4.4) no resultará en una relación sencilla entre la correspondiente respuesta en frecuencia para h[n] y la correspondiente respuesta en frecuencia para $h_a(t)$. De hecho, esta relación será:

$$H(e^{j\lambda}) = \frac{1}{T}\sum_{k=-\infty}^{\infty} H_a\left(j\frac{\lambda + 2\pi k}{T}\right) \tag{4.5}$$

donde

$$h[n] \xleftarrow{DTFT} H(e^{j\lambda})$$

$$h_a(t) \xleftarrow{FT} H_a(j\omega)$$

La ecuación 4.5 simplemente indica que $H(e^{j\lambda})$ será una versión alias $H_a(j\omega)$. la única forma de evitar el efecto de aliasing es haciendo que $H_a(j\omega)$ sea de banda limitada, tal que:

$$H_a(j\omega) = 0 \qquad \text{para } |\omega| \geq \frac{\pi}{T} \tag{4.6}$$

Si se satisface 4.6 entonces:

$$H(e^{j\lambda}) = \frac{1}{T}H_a\left(j\frac{\lambda}{T}\right) \qquad |\lambda| \leq \pi \tag{4.7}$$

Para un filtro analógico práctico, la ecuación (4.6) nunca será satisfecha exactamente pero el método de invariancia al impulso puede ser usado para aprovechar respuestas que son diferentes de cero pero despreciables más allá de alguna frecuencia.

La función de transferencia del prototipo analógico puede expresar en la forma de:

$$H_a(s) = \sum_{k=1}^{N}\frac{A_k}{s - s_k} \tag{4.8}$$

donde los s_k son los polos de $H_a(s)$ y el A_k esta dado por:

$$A_k = [(s - s_k)H_a(s)]_{s=s_k}$$

Basándose en el par transformado

$$e^{-at} \leftrightarrow \frac{1}{s + a}$$

La respuesta al impulso puede escribirse como:

$$h_a(t) = \sum_{k=1}^{N} A_k e^{s_k t} u(t) \qquad (4.9)$$

La respuesta de un muestreo al impulso unitario del filtro digital se forma muestreando la respuesta impulsiva del filtro prototipo para obtener:

$$h[n] = \sum_{k=1}^{N} A_k (e^{s_k T})^n u(t) \qquad (4.10)$$

La correspondiente función sistema para el filtro digital H(z) se obtiene como la transformada z de (4.10)

$$H(z) = \sum_{k=1}^{N} \frac{A_k}{1 - e^{s_k T} z^{-1}} \qquad (4.11)$$

Basados en el método descrito, hemos establecido el siguiente algoritmo para el diseño de un filtro RII:

Algoritmo 4.1 Diseño de una respuesta invariante al impulso de un filtro RII

Paso 1 Obtener la función de transferencia $H_a(s)$ para diseñar el prototipo analógico deseado

Paso 2 Para $k = 1,2,\ldots\ldots,N$, determinar los polos s_k de $H_a(s)$ y calcular los coeficientes A_k usando

$$A_k = \left[(s - s_k) H_a(s)\right] s = s_k \qquad (4.12)$$

Paso 3 Usando los coeficientes A_k obtenidos en el paso 2, generar la función sistema $H(z)$ del filtro digital como:

$$H(z) = \sum_{k=1}^{N} \frac{A_k}{1 - \exp(s_k T) z^{-1}} \qquad (4.13)$$

donde T es el intervalo de muestreo del filtro digital.

Paso 4 El resultado obtenido en el paso 3 será una suma de fracciones. Obtener un denominador común y expresar $H(z)$ como una relación de polinomios en z^{-1} de la forma:

$$H(z) = \frac{\sum_{k=0}^{M} b_k z^{-k}}{1 - \sum_{k=1}^{N} a_k z^{-k}} \qquad (4.14)$$

Paso 5 Usar las a_k y b_k obtenidos en el paso 4 para realizar el filtro en cualquiera de las estructuras dada en la tema 4.1

Como ejemplo, usamos la técnica de invariancia al impulso para derivar un filtro pasabajas digital RII a partir de un filtro analógico Butterworth de orden 2 con una frecuencia de corte de 3kHz.. La velocidad de muestreo para el filtro digital es 30,000 muestras por segundo.

La función de transferencia normalizada para un filtro Butterworth de segundo orden es:

$$H(s) = \frac{1}{(s-s_1)(s-s_2)}$$

donde

$$\begin{aligned} s_1 &= \cos\frac{3\pi}{4} + j\,\text{sen}\,\frac{3\pi}{4} \\ &= \frac{-\sqrt{2}}{2} + j\frac{\sqrt{2}}{2} \\ s_2 &= \cos\frac{5\pi}{4} + j\,\text{sen}\,\frac{5\pi}{4} \\ &= \cos\frac{3\pi}{4} - j\,\text{sen}\,\frac{3\pi}{4} \\ &= \frac{-\sqrt{2}}{2} - j\frac{\sqrt{2}}{2} \end{aligned}$$

La frecuencia de corte especificada de $f = 3000$ hace que $\omega_c = 6000\pi$ y la respuesta desnormalizada esta dada por:

$$\begin{aligned} H_a(s) &= \frac{{\omega_c}^2}{(s-\omega_c s_1)(s-\omega_c s_2)} \\ &= \frac{{\omega_c}^2}{\left[s+\omega_c\left(\sqrt{2/2}\right)-j\omega_c\left(\sqrt{2/2}\right)\right]\left[s+\omega_c\left(\sqrt{2/2}\right)+j\omega_c\left(\sqrt{2/2}\right)\right]} \end{aligned}$$

La expansión en fracciones parciales de $H(s)$ esta dada por:

$$H_a(s) = \frac{A_1}{\left[s+\omega_s\left(\sqrt{2/2}\right)-j\omega_c\left(\sqrt{2/2}\right)\right]} + \frac{A_2}{\left[s+\omega_s\left(\sqrt{2/2}\right)+j\omega_c\left(\sqrt{2/2}\right)\right]}$$

donde

$$A_1 = \frac{-j\sqrt{2}}{2\omega_c}$$

$$A_2 = \frac{j\sqrt{2}}{2\omega_c}$$

Usando estos valores para A_1 y A_2 más el hecho de que:

$$\omega_c T = \frac{6000\pi}{30000} = \frac{\pi}{5}$$

Obtenemos la función sistema $H(z)$ como:

$$H(z) = \frac{-j\sqrt{2}\,/(2\omega_c)}{1-\exp\left(\frac{\pi\sqrt{2}}{10}+j\frac{\pi\sqrt{2}}{10}\right)z^{-1}} + \frac{j\sqrt{2}\,/(2\omega_c)}{1-\exp\left(\frac{-\pi\sqrt{2}}{10}-j\frac{\pi\sqrt{2}}{10}\right)z^{-1}}$$

$$= \frac{2.06797\times10^{-6}\,z^{-1}}{1-1.158045z^{-1}+0.4112407z^{-2}}$$

4.4 METODO DE INVARIANCIA AL ESCALON

Una desventaja para diseñar filtros por medio del método de invariancia al impulso es su sensitividad a las características especificas de la señal de entrada. La respuesta al impulso unitario del filtro digital es una versión muestreada de la respuesta impulsiva del filtro prototipo; sin embargo, la respuesta del filtro prototipo a una entrada arbitraria, en general, no puede ser muestreada para obtener la respuesta escalón es de mas ayuda que la respuesta al impulso. En tales casos, la técnica de invariancia al impulso puede ser modificada para diseñar un filtro digital basado en el principio de invariancia al escalón, para lo cual es posible seguir el siguiente algoritmo:

Algoritmo 4.2 Diseño de Filtro RII por invariancia al escalón

Paso 1 Obtener la función de transferencia $H_a(s)$, para el filtro analógico prototipo deseado

Paso 2 Multiplicar $H_a(s)$ por $1/s$, para obtener $G_a(s)$, la respuesta del filtro a la función escalón unitario.

Paso 3 Para $k = 1,2,.....N$, determinar los polos s_k de $G_a(s)$ y calcular los coeficientes A_k usando:

$$A_k = \left[(s-s_k)G_a(s)\right]_{s=s_k} \qquad (4.15)$$

Paso 4 Usando los coeficientes A_k obtenidos en el paso 3, generar la función sistema G(z) como:

$$G(z) = \sum_{k=1}^{N} \frac{A_k}{1-\exp(s_k T)z^{-1}} \qquad (4.16)$$

Paso 5 Multiplicar $G(z)$ por $(1-z^{-1})$, para remover la transformada z del escalón unitario y obtener $H(z)$ como:

$$H(z) = (1-z^{-1})\sum_{k=1}^{N}\frac{A_s}{1-\exp(s_k T)z^{-1}} \qquad (4.17)$$

Paso 6 Obtener el denominador común, para los términos en la sumatoria del paso 5, y expresar $H(z)$ como una razón de polinomios en z^{-1} de la forma:

$$G(z) = \frac{\sum_{k=0}^{M} b_k z^{-k}}{1-\sum_{k=1}^{N} a_k z^{-k}} \qquad (4.18)$$

Paso 7 Usar a_k y b_k, obtenidos en el paso 6, para realizar el filtro en alguna de las estructuras dadas en la tema 4.1.

Los pasos de este algoritmo los hemos empleado para desarrollar la rutina en C, misma que se presentan al final de capitulo.

4.5 METODO POR TRANSFORMACION BILINEAL

La transfor mación bilineal convierte a la función de transferencia de un filtro analógico en la función sistema de un filtro digital, mediante la sustitución.

$$s \rightarrow \frac{2}{T}\frac{1-z^{-1}}{1+z^{-1}} \qquad (4.19)$$

Un mérito de este método es que si el filtro prototipo analógico es estable, se producirá un filtro digital estable. Esta técnica se puede desarrollar siguiendo el siguiente algoritmo:

Algoritmo 4.3 Transformación Bilineal

Paso 1 Obtener la función de Transferencia $H_a(s)$ para el filtro prototipo analógico deseado.

Paso 2 En la función de Transferencia obtenida en el paso 1, hacer la siguiente sustitución:

$$s = \frac{2}{T}\frac{1-z^{-1}}{1+z^{-1}}$$

donde T es el intervalo de muestreo del filtro digital. El resultado es llamado Función Sistema Digital $H(z)$

Paso 3 La función de Transferencia $H_a(s)$ de los filtros prototipos analógicos, en general, será una razón de polinomios en s. Por lo tanto, la función sistema $H(z)$ obtenida en el paso 2, en general, contendrá potencias de la razón $(1-z^{-1})/(1+z^{-1})$ en el numerador y denominador. Multiplicar el numerador y el denominador por la potencia mayor de $(1+z^{-1})$, y coleccionar los términos para obtener $H(z)$, como una razón de polinomios z^{-1} de la forma:

$$H(z)=\frac{\sum_{k=0}^{M} b_k z^{-k}}{1-\sum_{k=1}^{N} a_k z^{-k}} \qquad (4.20)$$

Paso 4 Usar a_k y b_k obtenidos en el paso 3, para realizar un filtro con las estructuras dadas en la sección 4.1.

Como ejemplo del uso de la Transformación Bilineal, podemos obtener un filtro RII a partir de un filtro analógico Butterworth de segundo orden con una frecuencia de corte de 3 kHz, considerando una velocidad de muestreo para el filtro digital de 30,000 muestras por segundo.

La función de transferencia del filtro analógico prototipo está dada por:

$$H_a(s)=\frac{{\omega_c}^2}{s^2+\sqrt{2}\omega_c s+{\omega_c}^2} \qquad (4.21)$$

donde $\omega_c = 6000\pi$.

Haciendo la sustitución $s=2(1-z^{-1})/\left(T(1+z^{-1})\right)$ obtenemos:

$$H(z)=\frac{{\omega_c}^2}{\left(\frac{2}{T}\left(\frac{1-z^{-1}}{1+z^{-1}}\right)\right)^2+\sqrt{2}\omega_c\left(\frac{2}{T}\left(\frac{1-z^{-1}}{1+z^{-1}}\right)\right)+{\omega_c}^2} \qquad (4.22)$$

donde $T = 1/30{,}000$.

Después de las simplificaciones algebraicas apropiadas, y haciendo uso del hecho que:

$$\omega_c T=\frac{6000\pi}{30{,}000}=\frac{\pi}{5}$$

obtenemos la forma deseada de H(z) como:

$$H(z)=\frac{0.063964+0.127929z^{-1}+0.06396z^{-2}}{1-1.168261z^{-1}+0.424118z^{-2}} \qquad (4.22b)$$

de lo cual se obtiene

$a_1 = -1.168261$ $\qquad$ $a_2 = 0.424118$

$b_0 = 0.063964$ $\qquad$ $b_1 = 0.127929$ $\qquad$ $b_2 = 0.063964$

4.6 FORMA FACTORIZADA DE LA TRANSFORMACION BILINEAL

Con frecuencia, un filtro analógico es especificado en términos de sus polos y ceros, es decir, el numerador y denominador de la función de transferencia, quedarán en forma factorizada. La transformación bilineal puede ser aplicada directamente a está forma factorizada, con el beneficio adicional de facilitar la ubicación de los polos y ceros del filtro digital. Cada factor en el numerador de la Función de Transferencia del filtro analógico será de la forma $(s-q_n)$, y para el denominador de la forma $(s-p_n)$, donde q_n y p_n son el n-ésimo cero y n-ésimo polo del filtro, respectivamente.

Cuando se aplica la transformación bilineal, los factores correspondientes serán:

$$\left(\frac{2}{T}\frac{1-z^{-1}}{1+z^{-1}}-q_n\right) \qquad (4.23a)$$

y

$$\left(\frac{2}{T}\frac{1-z^{-1}}{1+z^{-1}}-p_n\right) \qquad (4.23b)$$

Los ceros del filtro digital se obtienen ubicando los valores de z para los que:

$$\frac{2}{T}\frac{1-z^{-1}}{1+z^{-1}}-q_n=0$$

Los valores deseados de z son dados por:

$$z_p=\frac{2+q_nT}{2-q_nT} \qquad (4.24)$$

En forma similar, los polos del filtro digital se obtienen a partir de los polos del filtro analógico usando:

$$z_p=\frac{2+p_nT}{2-p_nT} \qquad (4.25)$$

El anterior procedimiento, que describe la aplicación de la transformación bilineal cuando una transformación de transferencia es exhibida en términos de sus polos y ceros, ha sido desarrollada en forma algorítmica al tiempo que hemos formulado una rutina en código C la cual mostramos al final de este capitulo. El algoritmo es el que mostramos a continuación:

Algoritmo 4.4 Transformación Bilineal para funciones de transferencia en forma factorizada.

Paso 1 Para el filtro prototipo analógico deseado, obtener la función de Transferencia $H_a(s)$ en la forma factorizada dada por:

$$H_a(s) = H_0 \frac{\prod_{m=1}^{M}(s - q_m)}{\prod_{n=1}^{N}(s - p_n)} \qquad (4.26)$$

Paso 2 Obtener los polos z_{pn} del filtro digital a partir de los polos p_n del filtro analógico usando:

$$z_{pn} = \frac{2 + p_n T}{2 - p_n T} \qquad n = 1,2,....,N \qquad (4.27)$$

Paso 3 Obtener los ceros z_{zm} del filtro digital a partir de los ceros q_m del filtro analógico usando:

$$z_{zm} = \frac{2 + q_m T}{2 - q_m T} \qquad m = 1,2,....,M \qquad (4.28)$$

Paso 4 Usando los valores de z_{pn} obtenidos en el paso 2 y los valores de z_{zm} obtenidos en el paso 3 , formar $H(z)$ como:

$$H(z) = H_0 \frac{T^N}{\prod_{n=1}^{N}(2 - p_n T)} \cdot \frac{(z+1)^{N-M}\prod_{m=1}^{M}(z - z_{zm})}{\prod_{n=1}^{N}(z - z_{pn})} \qquad (4.29)$$

El factor $(z+1)^{N-M}$ proporciona los ceros en $z = -1$, que corresponde a los ceros de $s = \infty$ para el filtro analógico $M < N$. El primer factor racional en la ecuación 4.29 es un factor de ganancia constante, necesario para obtener los mismos resultados que en el algoritmo los cuales marcan exactamente los resultados obtenidos por medio de este.

Como ejemplo de la aplicación factorizada de la transformación bilinial, consideramos el caso del filtro Butterworth que hemos venido tratando, para el cual se tiene una función de transferencia factorizada como:

$$H_a(s) = \frac{{\omega_c}^2}{\left[s + \omega_c\left(\sqrt{2}/2\right) - j\omega_c\left(\sqrt{2}/2\right)\right]\left[s + \omega_c\left(\sqrt{2}/2\right) + j\omega_c\left(\sqrt{2}/2\right)\right]}$$

El filtro prototipo analógico tiene polos en

$$s = \omega_c \frac{-\sqrt{2}}{2} \pm j\omega_c \frac{\sqrt{2}}{2}$$

Usando 4.25, obtenemos los polos de los filtros digitales como:

$$z_{p1} = \frac{2+\left(\frac{-\sqrt{2}}{2}+j\frac{\sqrt{2}}{2}\right)\omega_c T}{2-\left(\frac{-\sqrt{2}}{2}+j\frac{\sqrt{2}}{2}\right)\omega_c T}$$

$$= 0.584131 + 0.28794j$$

$$z_{p2} = \frac{2+\left(\frac{-\sqrt{2}}{2}-j\frac{\sqrt{2}}{2}\right)\omega_c T}{2-\left(\frac{-\sqrt{2}}{2}-j\frac{\sqrt{2}}{2}\right)\omega_c T}$$

$$= 0.584131 - 0.28794j$$

Los dos ceros en $s=\infty$ son mapeados como ceros de $z=-1$, por lo que la función sistema esta dada:

$$H(z) = H_c \frac{(z+1)^2}{(z-0.584131+0.287941j)(z-0.584131-0.287941j)}$$

donde:

$$H_c = \frac{H_0 T^2}{(2-p_1 T)(2-p_2 T)}$$

$$= \frac{(6000\,\pi)^2}{(30{,}000)\left(2+\frac{\pi\sqrt{2}}{10}-j\frac{\pi\sqrt{2}}{10}\right)\left(2+\frac{\pi\sqrt{2}}{10}+j\frac{\pi\sqrt{2}}{10}\right)}$$

lo cual se puede simplificar a:

$$H(z) = \frac{0.063964(1+2z^{-1}+z^{-2})}{1-1.168261z^{-1}+0.424118z^{-2}} \qquad (4.30)$$

El cual corresponde al resultado del ejemplo anterior, (ecuación 4.22b).

4.7 PROPIEDADES DE LA TRANSFORMADA BILINEAL

Asumido que el filtro prototipo analógico tienen un polo en $s_p = \sigma + j\omega$, el filtro RII correspondiente diseñado con la transformación bilineal tendrá un polo en:

$$z_p = \frac{2+sT}{2-sT}$$
$$= \frac{2+(\sigma+j\omega)T}{2+(\sigma-j\omega)T}$$
$$= \frac{2+\sigma T+j\omega T}{2-\sigma T-j\omega T}$$

La magnitud y ángulo de este polo quedan dadas por:

$$|z_p| = \sqrt{\frac{(2+\sigma T)^2+(\omega T)^2}{(2-\sigma T)^2-(\omega T)^2}} \qquad (4.31)$$

$$\arg(z_p) = \tan^{-1}\left(\frac{\omega T}{2+\sigma T}\right) - \tan^{-1}\left(\frac{-\omega T}{2-\sigma T}\right) \qquad (4.32)$$

Los polos de un filtro analógico estable deben caer en el plano izquierdo del plano s, es decir $\sigma < 0$. Cuando $\sigma < 0$, el numerador de 4.31 será menor que el denominador, y de esta manera $|z_p| < 1$. Esto quiere decir, que los polos analógicos en el lado izquierdo del plano s son mapeados a polos digitales dentro del circulo unitario del plano z.. Los polos que caen sobre el eje $j\omega$ del plano s tiene $\sigma = 0$ y consecuentemente sé mapean a polos en plano z, con magnitud unitaria. Los polos analógicos en $s = 0$ mapean polos digitales en $z = 1$ en tanto que los polos analógicos en $s = \pm j\infty$,. mapeados dentro de los polos digitales en $z = -1$.

- **Deformación de Frecuencia (warping)**

El mapeo del eje $j\omega$ del plano s hacia el circulo unitario en plano z es un mapeo altamente no lineal. La frecuencia analógica ω_0 puede tener el variar de $-\infty$ a $+\infty$, pero la frecuencia digital ω_d es limitada al rango de $\pm\pi$.

La relación entre ω_a y ω_d esta dada por:

$$\omega_d = 2tan^{-1}\frac{\omega_a T}{2} \qquad (4.33)$$

Si un filtro prototipo analógico con una frecuencia de corte ω_a es usado para diseñar un filtro por medio de la Transformación Bilineal, el filtro digital resultante tendrá una frecuencia de corte ω_d, donde ω_d se relaciona a ω_a por medio de la ecuación 4.33.

Para ejemplificar la aplicación de la relación anterior, usamos un filtro prototipo pasabajas con una frecuencia de corte de 3kHz., para un filtro RII, con una velocidad de muestreo de 30000 muestras por segundo. En este caso, la ecuación 6.33 produce:

$$\omega_d = 2tan^{-1}\frac{(6000\pi)(1/30{,}000)}{2}$$
$$= 0.6088$$

donde $\omega_d = \pi$ que corresponde a una frecuencia de $30{,}000/2 = 15{,}000Hz$, la frecuencia de corte del filtro esta dada por:

$$\omega_c = \frac{0.6088}{\pi}(15{,}000) = 2906.8Hz$$

Lo cual nos indica un desplazamiento en la frecuencia de corte a 3dB, que se ubica en 2908.6 Hz y no en los 3000 Hz; por supuesto, los efectos de deformación serán más severos cuando la frecuencia de interés se incremente con la velocidad de muestreo del filtro digital.

Considérese el caso de un filtro analógico con una frecuencia de corte de 3kHz. Usado como el prototipo de un filtro RII diseñado vía la Transformación Bilineal. Determine el impacto sobra la frecuencia de 3dB si el muestreo rápido es cambiado de 10,000 muestras por segundo a 30,000 muestras por segundo en pasos de 1000 muestras por segundo.

Las nuevas velocidades de muestreo y la correspondiente deformación de frecuencias a 3dB se listan en la tabla 4.1

Afortunadamente esta es una simple forma para contrarrestar los efectos de la frecuencia deformada. Por la predeformación de las frecuencias criticas de los filtros prototipo analógicos, la cual es una manera de deformación causada por la restauración de la transformación bilineal de las frecuencias criticas, para sus valores originales. La ecuación 4.33 puede ser invertida para dar la ecuación necesaria para esta deformación.

$$\omega_a = \frac{2}{T}tan\frac{\omega_d}{2} \qquad (4.34)$$

Velocidad de muestreo	Frecuencia de corte en Hz	% Error
10,000	2405.8	-19.81
11,000	2480.5	-17.32
12,000	2543.1	-15.23
13,000	2595.8	-13.47
14,000	2640.4	-11.99
15,000	2678.5	-10.72
16,000	2711.1	-9.63
17,000	2739.3	-8.69
18,000	2763.6	-7.88
19,000	2784.9	-7.17
20,000	2803.5	-6.55
21,000	2819.9	-6.00
22,000	2834.4	-5.52
23,000	2847.2	-5.09
24,000	2858.7	-4.71
25,000	2868.9	-4.37
26,000	2878.1	-4.06
27,000	2886.4	-3.79
28,000	2893.8	-3.54
29,000	2900.6	-3.31
30,000	2906.8	-3.11

Tabla 41 Deformación de frecuencias de corte

Queremos diseñar un filtro RII con una frecuencia de 3kHz., a 3dB y una velocidad de muestreo de 30,000 muestras por segundo que determina la frecuencia requerida para la predeformación a 3 dB para el filtro prototipo analógico.

donde $\omega_d = \pi$. Corresponde a una frecuencia de $30{,}000/2 = 15{,}000Hz$. A una frecuencia de 3kHz corresponde a ω_d de:

$$\omega_d = \frac{3000\pi}{15{,}000} = \frac{\pi}{5}$$

La frecuencia ω_a del filtro analógico prototipo se obtienen usando los valores de ω_d en la ecuación 4.34:

$$\omega_a = \frac{2}{(1/3000)} tan\frac{\pi}{10} = 19{,}495.18$$

El filtro prototipo analógico debe tener una frecuencia de $19{,}495.18/(2\pi) = 3102.75Hz$, a 3dB, a fin de que el filtro RII tenga una frecuencia de 3Khz a 3dB después de la deformación.

4.8 PROGRAMACION DE LA TRANSFORMACION BILINEAL

Si asumimos que la función de Transferencia del filtro prototipo analógico es de la forma:

$$H_a(s) = H_0 \frac{\prod_{m=1}^{M}(s - q_m)}{\prod_{n=1}^{N}(s - p_n)} \qquad (4.35)$$

donde p_n y q_n, denota respectivamente los polos y ceros del filtro, podemos generar un filtro digital vía la Transformación Bilineal, haciendo la siguiente sustitución.

$$s = \frac{2}{T}\left(\frac{1 - z^{-1}}{1 + z^{-1}}\right)$$

para obtener:

$$H(z) = H_0 \frac{\prod_{m=1}^{M}\left[\frac{2}{T}\left(\frac{1 - z^{-1}}{1 + z^{-1}}\right) - q_m\right]}{\prod_{m=1}^{M}\left[\frac{2}{T}\left(\frac{1 - z^{-1}}{1 + z^{-1}}\right) - p_n\right]} \qquad (4.36)$$

El cual, después de algunas manipulaciones algebraicas se puede escribir como:

$$H(z) = H_0 \frac{(1 + z^{-1})^{N-M} \prod_{m=1}^{N}\left[\left(\frac{2}{T} - q_m\right) - \left(\frac{2}{T} + q_m\right)z^{-1}\right]}{\prod_{m=1}^{N}\left[\left(\frac{2}{T} - p_n\right) - \left(\frac{2}{T} + p_n\right)z^{-1}\right]} \qquad (4.36b)$$

Así el denominador de $H(z)$ esta dado por:

$$D(z) = \prod_{n=1}^{N} (\gamma_n + \delta_n z^{-1}) \qquad (4..37)$$

donde

$$\gamma_n = \frac{2}{T} - p_n$$

$$\delta_n = \frac{-2}{T} - p_n$$

Para ver como 4.37 puede ser fácilmente evaluada por computadora, examinamos la secuencia de productos parciales $\{D_k(z)\}$ encontrados en la evaluación:

$$D_1(z) = (\gamma_1 + \delta_1 z^{-1})$$
$$D_2(z) = (\gamma_2 + \delta_2 z^{-1}) D_1(z) = \gamma_2 D_1(z) + \delta_2 z^{-1} D_1(z)$$
$$D_3(z) = (\gamma_3 + \delta_3 z^{-1}) D_2(z) = \gamma_3 D_2(z) + \delta_3 z^{-1} D_2(z)$$
$$D_4(z) = (\gamma_4 + \delta_4 z^{-1}) D_3(z) = \gamma_4 D_3(z) + \delta_4 z^{-1} D_3(z)$$
$$\vdots$$
$$D(z) = D_N(z) = (\gamma_N + \delta_N z^{-1}) D_{N-1}(z) = \gamma_N D_{N-1}(z) + \delta_N z^{-1} D_{N-1}(z)$$

El examen de esta secuencia revela que el producto parcial $D_k(z)$ para una iteración k se expande en términos del producto parcial $D_{k-1}(z)$ como:

$$D_k(z) = \gamma_k D_{k-1}(z) + \delta_k z^{-1} D_{k-1}(z)$$

El producto parcial $D_{k-1}(z)$ generará un polinomio de grado $(k-1)$ en z^{-1}:

$$D_{k-1}(z) = \mu_0 (z^{-1})^0 + \mu_1 (z^{-1})^1 + \mu_2 (z^{-2}) + \ldots\ldots + \mu_{k-1} (z^{-1})^{k-1}$$

Los productos $\gamma_k D_{k-1}(z)$ y $\delta_k z^{-1} D_{k-1}(z)$ están dados por

$$\gamma_k D_{k-1}(z) = \gamma_k \mu_0 (z^{-1})^0 + \gamma_k \mu_1 (z^{-1})^1 + \gamma_k \mu_2 (z^{-1})^2 + \ldots\ldots + \gamma_k \mu_{k-1} (z^{-1})^{k-1}$$

$$\delta_k z^{-1} D_{k-1}(z) = \delta_k \mu_0 (z^{-1})^1 + \delta_k \mu_1 (z^{-1})^2 + \delta_k \mu_2 (z^{-1})^3 + \ldots\ldots + \delta_k \mu_{k-1} (z^{-1})^k$$

y $D_k(z)$ esta dada por:

$$D_k(z) = \gamma_k \mu_0 (z^{-1})^0 + (\gamma_k \mu_1 - \delta_k \mu_0)(z^{-1})^1 + (\gamma_k \mu_0 - \delta_k \mu_1)(z^{-1})^2 + \ldots\ldots$$

$$+ (\gamma_k \mu_{k-1} - \delta_k \mu_{k-2})(z^{-1})^{k-1} - \delta_k \mu_{k-1} (z^{-1})^k$$

Por lo tanto, podemos concluir que si μ_n, es el coeficiente para los (z^{-1}) términos en $D_{k-1}(z)$, entonces los coeficientes para $(z^{-1})^n$ términos en $D(z)$, son $(\gamma_k \mu_n + \delta_k \mu_{n-1})$ con la condición que $\mu \equiv 0$ en $D_{k-1}(z)$. El polinomio $D_{k-1}(z)$ se presenta como un arreglo de coeficientes k, con el índice del arreglo correspondiendo al subíndice de μ. Así el elemento del arreglo $mu[0]$ contiene μ_0 el elemento del arreglo $mu[1]$ contiene μ_1 y así en adelante.

Los coeficientes para el producto parcial $D_k(z)$ pueden ser obtenidos de los coeficientes para $D_{k-1}(z)$, como indica el siguiente fragmento de pseudocódigo

```
for(j=k; j>=1;j--)
    {mu[j]=gamma*mu[j]+beta*mu[j-1]}
```

El ciclo es ejecutado en forma decreciente para que los coeficientes puedan ser actualizados "in place". Si se considera la aritmética compleja, para obtener los valores efectivos del algoritmo, el código se puede reescribir como :

```
for(j=k; j>=1;j--)
    {mu[j]=cSub(cMult(gamma.mu[j].cMult(delta.mu[j-1]));}
```

Si este fragmento es colocado dentro de un lazo exterior con k de 1 hasta **numpolos( )**, los valores finales en $mu[n]$ serán los coeficientes a_n para la ecuación 4.20. Similarmente un ciclo puede ser desarrollado para el producto del denominador $N(z)$ dado por:

$$N(z) = \prod_{m=1}^{M} (\alpha_m - \beta_m z^{-1}) \qquad (4.38)$$

donde

$$\alpha_m = \frac{-2}{T} + q_m$$

$$\beta_m = \frac{-2}{T} - q_m$$

El programa completo, desarrollado en C, para los cálculos de la transformación bilineal es mostrado en el listado de programas.

LISTADO DE PROGRAMAS DE DISEÑO DE FILTROS DIGITALES IIR

```
/*                                                                   */
/*                          | Este programa permite obtener la transforma- */
/*   Programa 4.1           | da discreta de Fourier de una funcion       */
/*                          | de transferencia H(z).                      */
/* Respuesta Impulso        |                                             */
/*  Infinito()              |                                             */
/* -----------------------------------------------------------------*/

void respuestaIInfinito( struct complex a[],int bigN,struct complex b[],
                         int bigM,   int numerPuntos, logical dbScala,
                        real magnitude[], real fase[])
{
static struct complex respuesta[MAXPUNTOS];
int k, n, m;
real sumRe, sumIm, phi;

/*                                                              */
/*  calcula la Transformada Discreta de Fourier del numerador de H(z)   */
clrscr();
for( m=0; m<numerPuntos; m++) {
      sumRe = 0.0;
      sumIm = 0.0;
      //printf("\r%d  000",m);
      for(n=0; n<=bigM; n++) {
            printf("\b\b\b%3d",n);
            phi = 2.0 * PI * m * n / (2.0*numerPuntos);
            //printf("b[%d] = (%e, %e)\n",n,b[n].x,b[n].y);
            sumRe += b[n].x * cos(phi) + b[n].y * sin(phi);
            sumIm += b[n].y * cos(phi) - b[n].x * sin(phi);
      }
      respuesta[m] = cmplx(sumRe, sumIm);
      printf("respuesta = (%e, %e)\n",respuesta[m].x, respuesta[m].y);
      getche();
}
/*-----------------------------------------------------------------*/
/*  calcula la Transformada Discreta de Fourier del denominador de H(z) */

for( m=0; m<numerPuntos; m++) {
      sumRe = 1.0;
      sumIm = 0.0;

      for(n=1; n<=bigN; n++) {
            phi = 2.0 * PI * m * n / (2.0*numerPuntos);
            sumRe += -a[n].x * cos(phi) - a[n].y * sin(phi);
            sumIm += -a[n].y * cos(phi) + a[n].x * sin(phi);
            }
      respuesta[m] = cDiv(respuesta[m],cmplx(sumRe, sumIm));
      }
/*  Calcula la magnitud y fase de la  respuesta     */

   for( m=0; m<numerPuntos; m++) {
      fase[m] = arg(respuesta[m]);
      if(dbScala)
            {magnitude[m] = 20.0 * log10(cAbs(respuesta[m]));}
      else
            {magnitude[m] = cAbs(respuesta[m]);}

      printf("\n magnitud = %lf    y   fase = %f   ",magnitude[m], fase[m]);
   }
return;
}
```

```
/*-------------------------------------------------------------------------*/
/*                                  |  Este programa calcula los coeficientes */
/*   Programa 4.2                   |  de una respuesta al impulso            */
/*                                  |                                         */
/*  ImpulsoInvar()                  |                                         */
/*                                  |                                         */
/*-------------------------------------------------------------------------*/


void impulseInvar(      struct complex polo[], int numPolos, struct complex cero[],
                              int numCeros, real hCero, real bigT,
                              struct complex a[], struct complex b[])

{
int k, n, j, maxCoef;
struct complex delta[MAXPOLOS];
struct complex bigA[MAXPOLOS];
struct complex beta, denom, numer, work2;

for(j=0; j<MAXPOLOS; j++) {
      delta[j] = cmplx(0.0,0.0);
      a[j] = cmplx(0.0,0.0);
      b[j] = cmplx(0.0,0.0);
      }
/*-------------------------------------------------------------------------*/
/*  calcula los coeficientes con expansion de fracciones parciles  */

for( k=1; k<=numPolos; k++) {
      numer = cmplx(hCero,0.0);
      for(n=1; n<=numCeros; n++)
            { numer = cMult(numer, cSub(polo[n], cero[n]));}
      denom = cmplx(1.0,0.0);
      for( n=1; n<=numPolos; n++) {
            if(n==k) continue;
            denom = cMult(denom, cSub(polo[k],polo[n]));
            }
      bigA[k] = cDiv(numer,denom);
      }
/*-------------------------------------------------------------------------*/
/*  calcula los coeficientes del numerador      */
/*
for( k=1; k<=numPolos; k++) {
      delta[0] = cmplx(1.0, 0.0);

      for(n=1; n<MAXPOLOS; n++)
            {delta[n] = cmplx(0.0,0.0);}
      maxCoef = 0;
      for( n=1; n<=numPolos; n++) {
            if(n==k) continue;
            maxCoef++;
            beta = sMult(-1.0, cExp(sMult(bigT,pclo[n])));
            for(j=maxCoef; j>=1; j--)
                  { delta[j] = cAdd( delta[j], cMult( beta, delta[j-1]));}
            }
      for( j=0; j<numPolos; j++)
            { b[j] = cAdd(b[j], cMult( bigA[k], delta[j])); }
      }  */
/*-------------------------------------------------------------------------*/
/*  calcula los delta coeficientes del  denominador   */
/*
a[0] = cmplx(1.0,0.0);
for( n=1; n<=numPolos; n++) {
      beta = sMult(-1.0, cExp(sMult(bigT,polo[n])));
      for( j=n; j>=1; j--)
            { a[j] = cAdd( a[j], cMult( beta, a[j-1]));}
      }
for( j=1; j<=numPolos; j++)
      { a[j] = sMult(-1.0,a[j]);}
return;
}
```

```
/*                                                                          */
/*                                  | Este programa permite obtener la transforma */
/*   Programa 4.3                   | da discreta de Fourier de una funcion       */
/*                                  | de transferencia H(z).                      */
/*   PasoInvar()                    |                                             */
/*                                  |                                             */
/* ---------------------------------------------------------------------------*/

void stepInvar(   struct complex polo[], int numPolos, struct complex cero[],
                        int numCeros, real hCero,     real bigT,
                        struct complex a[], struct complex b[])
{
 int k, n, j, maxCoef;
 struct complex delta[MAXPOLOS];
 struct complex bigA[MAXPOLOS];
 struct complex beta, denom, numer, work2;

for(j=0; j<MAXPOLOS; j++) {
      delta[j] = cmplx(0.0,0.0);
      a[j] = cmplx(0.0,0.0);
      b[j] = cmplx(0.0,0.0);
      }
polo[0] = cmplx(0.0,0.0);

/*calcula los coeficientes con expansion de fracciones parciles  */
/*
for( k=0; k<=numPolos; k++) {
      numer = cmplx(hCero,0.0);
      for(n=1; n<=numCeros; n++)
            { numer = cMult(numer, cSub(polo[n], cero[n]));}
      denom = cmplx(1.0,0.0);
      for( n=0; n<=numPolos; n++) {
            if(n==k) continue;
            denom = cMult(denom, cSub(polo[k],polo[n]));
            }
      bigA[k] = cDiv(numer,denom);
}

/*  calcula los coeficientes del numerador      */

for( k=1; k<=numPolos; k++) {
      delta[0] = cmplx(1.0, 0.0);
      for(n=1; n<MAXPOLOS; n++)
            {delta[n] = cmplx(0.0,0.0);}
      maxCoef = 0;
      for( n=0; n<=numPolos; n++) {
            if(n==k) continue;
            maxCoef++;
            beta = sMult(-1.0, cExp(sMult(bigT,polo[n])));
            for(j=maxCoef; j>=1; j--)
                  { delta[j] = cAdd( delta[j], cMult( beta, delta[j-1]));}
            }
      for( j=0; j<numPolos; j++)
            { b[j] = cAdd(b[j], cMult( bigA[k], delta[j])); }

      beta = cmplx(-1.0,0.0);                   /* se multiplica por (-z**(-1))   */
      for(j=numPolos+1; j>=1; j--) {
            b[j] = cAdd(b[j], cMult(beta, b[j-1]));}
}

/*  calcula los coeficientes del denominador      */

a[0] = cmplx(1.0,0.0);
for( n=1; n<=numPolos; n++) {
      beta = sMult(-1.0, cExp(sMult(bigT,polo[n])));
      for( j=n; j>=1; j--)
            { a[j] = cAdd( a[j], cMult( beta, a[j-1]));}
      }
for( j=1; j<=numPolos; j++)
      { a[j] = sMult(-1.0,a[j]);}
return;
}
```

```
/*------------------------------------------------------------------------*/
/*                                  |  Este programa permite diseñar un filtro  */
/*   Programa 4.4                   |  digital por medio del metodo  Bilineal   */
/*                                  |                                           */
/*   Bilineal()                     |                                           */
/*                                  |                                           */
/*------------------------------------------------------------------------*/

void bilinear(    struct complex polo[], int numPolos, struct complex cero[],
                int numCeros, real  hCero, real bigT,
              struct complex a[], struct complex b[])
{
int j,k,m,n, maxCoef;
real hC;
struct complex mu[MAXPOLOS];
struct complex alpha, beta, gamma, delta, eta;
struct complex work, cTwo;

for(j=0; j<MAXPOLOS; j++) {
      mu[j] = cmplx(0.0,0.0);
      a[j] = cmplx(0.0,0.0);
      b[j] = cmplx(0.0,0.0);
      }

/*  calcula el factor constante       */
hC = 1.0;
work = cmplx(1.0,0.0);
cTwo = cmplx(2.0,0.0);
for(n=1; n<=numPolos; n++) {
      work = cMult(work, cSub(cTwo, sMult(bigT,polo[n])));
      hC = hC * bigT;
      }
hC = hCero * hC / work.Re;

/*  calcula los coeficientes del numerador     */

mu[0] = cmplx(1.0, 0.0);
maxCoef = 0;
for( m=1; m<=(numPolos-numCeros); m++) {
      maxCoef++;
      for( j=maxCoef; j>=1; j--)
            { mu[j] = cAdd( mu[j], mu[j-1]);}
      }
for( m=1; m<=numCeros; m++) {
      maxCoef++;
      alpha = cAdd(cmplx( (-2.0/bigT), 0.0), cero[n]);
      beta = cSub(cmplx( (-2.0/bigT), 0.0), polo[n]);
      for(j=maxCoef; j>=1; j--)
            { mu[j] = cAdd( mu[j], cMult( beta, mu[j-1]));}
      }
for( j=0; j<=numPolos; j++) b[j] = sMult(hC, mu[j]);

/*  calcula los coeficientes del denominador    */

mu[0] = cmplx(1.0,0.0);
for(n=1; n<MAXPOLOS; n++)
      {mu[n] = cmplx(0.0,0.0);}
for( n=1; n<=numPolos; n++) {
      gamma = cSub(cmplx( (2.0/bigT), 0.0), polo[n]);
      delta = cSub(cmplx( (-2.0/bigT), 0.0), polo[n]);
      eta = cDiv( delta, gamma);
      for( j=n; j>=1; j--)
            { mu[j] = cAdd( mu[j], cMult(eta, mu[j-1]));}
      }
for( j=1; j<=numPolos; j++)
      { a[j] = sMult(-1.0, mu[j] );}
return;
}
```

BIBLIOGRAFIA

- Digital Filter designer´s handbook
  C. Britton Rorabaugh.
  Ed. McGrawHill

- Designing digital filters
  Charles S. Williams
  Ed. Prentice-Hall

- Introductory digital signal processing with computer applications
  Paul A. Lynn
  Wolfgang Fuerst
  Ed. JohnWiley & Sons.

- Digital signal processing
  Principles, algorithms and applications
  Jonh G Proakis, Dimitris G. Manolakis
  Ed. Macmillan Publishing Company
  2a. edición.

- Digital signal processing a system design approach
  David J Defatta,
  Joseph G. Lucas
  William s. Hodgkisss
  Ed. John Wiley & Sons

- Introduction to signals and systems
  Edward. W. Kamen
  Macwillan Publishing Company

ANEXO I

DEFINICIONES GLOBALES

DEFINICIONES GLOBALES

```
/*                                                      */
/*                                                      */
/*  ANEXO I   Definiciones  globales                    */
/*                                                      */
/*  globDefs.h                                          */
/*                                                      */
/*                                                      */
/*                                                      */

#include <stdio.h>
#include <math.h>
#include <time.h>

#define EOL 10
#define STOP_CHAR 38
#define SPACE 32
#define TRUE 1
#define FALSE 0
#define PI 3.141592653589
#define TWO_PI 6.2831853
#define TEN (double) 10.0
#define MAX_COLUMNS 20
#define MAX_ROWS 20
#define MAXORDEN 20
#define MAXPUNTOS 50
#define DFTSIZE 256

/*  definicion de una estructura compleja simple */
/* struct complex
        {
        float Re;
        float Im;
        }; */
/*  definicion de una estructura compleja doble */
struct complex
        {
        double Re;
        double Im;
        };   */

                        /* se excluyo la definicion debido     */
                        /* a que c++ incorpora la estructura   */
                        /* complex.h con argumentos x, y       */
typedef int logical;
typedef double real;
```

ANEXO II

DEFINICION DE
FUNCIONES LOCALES

DECLARACIÓN DE FUNCIONES

Prototipos para las Funciones en C

```
/*-----------------------------------------------*/
/* funciones empleadas en el programa  principal */
/*-----------------------------------------------*/

void menu(void);
void menu1(void);
void menu2(void);
void menu3(void);
void menu4(void);
void menuBut(void);
void menuChe(void);
void menuBess(void);
void menuEli(void);
void DFT1(void);
void DFT2(void);
void FFT(void);
void rff(void);
void series(void);
void muestreo(void);
void rfi(void);
void impulso(void);
void paso(void);
void bilineal(void);
void respImpulsoInfini(void);
void inicializar(void);
void continuar(void);
void error(void);
void otro(void);
void inicio_fin(void);
void sonido(void);

int respuestaFrecbutterworth( int ordenf,
                              real frecuenciaf,
                              real *magnitudf,
                              real *fasef);

void respuestaImpulsivabutterw(int ordenf,
                               real deltaT,
                               int npts,
                               real yval[]);

void respuestaFrecChebyshev(int ordenf,
                            real rizof,
                            char tiponormalizacion,
                            real frecuencia,
                            real *magnitudf,
                            real *fasef);

void respuestaimpulsivaCheby(int ordenf,
                             real rizof,
                             char tiponormalizacion,
                             real deltaT,
                             int npts,
                             real yval[]);

void estimarOrden(real omegaPaso,
                  real omegaStop,
                  real maxperdidapaso,
                  real minperdidaparo,
                  int *ordenf,
                  real *actualminperidaparo);
```

```
void calcularCoefic(real omegaPaso,
                    real omegaStop,
                    real maxperdidapaso,
                    int ordenf,
                    real aa[],
                    real bb[],
                    real cc[],
                    int *numSecs,
                    real *hZero,
                    real *pZero);

void calcuRespuestaFrec(int ordenf,
                        real aa[],
                        real bb[],
                        real cc[],
                        real hZero,
                        real pZero,
                        real frequenciaf,
                        real *magnitudf,
                        real *fasef);

void reescalarCoef(int ordenf,
                   real aa[],
                   real bb[],
                   real cc[],
                   real *hZero,

                   real *pZero,
                   real alpha);

void coeficientesBessel( int ordenf,
                         char tiponormalizacion,
                         real coef[]);

void respuestaFrecBessel(int ordenf,
                         real coef[],
                         real frecuenciaf,
                         real *magnitudf,
                         real *fasef);

void retardoGrupoBessel( int ordenf,
                        real coef[],
                        real frecuencia,
                        real delta,
                        real *retardoGrupo);

void dft( struct complex x[],
          struct complex xx[],
          int nn);

void dft2( struct complex x[],
           struct complex xx[],
           int nn);

void fft( struct complex x[],
          struct complex xx[],
          int nn);

void respuestaImpulFinita( int tipofiltro,
                       int numbTaps,
                      real hh[],
                       int escaladB,
                       int numeroDPuntos,
                      real hD[]);

void respuestaNormalizada(int escaladB,
                          int numPts,
                          real hh[]);
```

```
void pasaBajasIdeal( int numbTaps,
                     real omegaU,
                     real coefficient[]);

void pasaAltasIdeal( int numbTaps,
                     real omegaL,
                     real coefficient[]);

void pasaBandaIdeal( int numbTaps,
                     real omegaL,
                     real omegaU,
                     real coefficient[]);

void rechazaBandaIdeal( int numbTaps,
                     real omegaL,
                     real omegaU,
                     real coefficient[]);

real respuestaRectangularCont(real freq,
                              real tau,
                              int dbScale);

real respuestaRectangularDisc(real freq,
                              int M,
                              int normalizedAmplitude);

real respuestaTriangularCont( real freq,
                              real tau,
                              int dbScale);

real respuestaTriangularDisc( real freq,
                              int M,
                              int normAmp);

void ventanaTriangular( int N, real Ventana[]);

void makeLagVentana( int N,  real Ventana[],
                     int  center,
                     real outWindow[]);

void makeDataVentana( int N,
                      real Ventana[],
                      real outWindow[]);

void ventanaHann( int nn, real Ventana[]);

void ventanaHamming( int nn, real Ventana[]);

int disenoFiltroMuestreo( int N, int firType,real A[], real h[]);

real findSbPeak( int bandConfig[], int numPts, real hh[]);

real goldenSearch( int firType,int numbTaps,
                   real hD[], real gsTol,int numFreqPts,
                   int bandConfig[],  real *fmin);

void setTrans( int bandConfig[], real x,   real hD[]);

real goldenSearch2( real rhoMin, real rhoMax,
                    int firType, int numbTaps,
                    real hD[],   real gsTol,
                    int numFreqPts, real origins[], real slopes[],
                    int bandConfig[], real *fmin);

void setTransition( real origins[],
                    real slopes[],
                    int bandConfig[],
                    real x,
                    real Hd[]);
```

```
void optimize2(       real yBase,
               int firType, int numbTaps,
               real hD[],  real gsTol,
               int numFreqPts,  int bandConfig[],
               real tweakFactor, real rectComps[]);

void dumpRectComps( real origins[],  real slopes[],
                    int numTransSamps,  real x);

real gridFreq(  real gridParam[], int gI);

real desLpfResp(  real freqP, real freq);

real weightLp( real kk, real freqP, real freq);

void respuestaIIInfinito( struct complex a[],
             int bigN,
             struct complex b[],
             int bigM,
             int numberOfPoints,
             int dbScale,
             real magnitude[],
             real phase[]);

void impulseInvar(struct complex pole[],
                              int numPoles,
                              struct complex zero[],
                              int numZeros,
                              real hZero,
                              real bigT,
                              struct complex a[],
                              struct complex b[]);

void stepInvar(       struct complex pole[],
                              int numPoles,
                              struct complex zero[],
                              int numZeros,
                              real hZero,
                              real bigT,
                              struct complex a[],
                              struct complex b[]);

void bilinear(        struct complex pole[],
                              int numPoles,
                              struct complex zero[],
                              int numZeros,
                              real hZero,
                              real bigT,
                              struct complex a[],
                              struct complex b[]);

struct complex cmplx(  real A, real B);
struct complex cAdd( struct complex A, struct complex B);
struct complex cSub( struct complex A, struct complex B);
real cMag(struct complex A);
real cAbs(struct complex A);
double cdAbs(struct complex A);
real arg(struct complex A);
struct complex cSqrt( struct complex A);
struct complex cMult(  struct complex A, struct complex B);
struct complex sMult( real a, struct complex B);
struct complex cDiv( struct complex numer, struct complex denom);

real sincSqrd( real x);
real sinc( real x);
real acosh( real x);
void pause( int enabled);
int bitRev(    int L, int N);
int log2(  int N );
real ipow(     real x, int k);
```

ANEXO III

FUNCIONES DE ARITMETICA COMPLEJA

Función con Aritmética Compleja

```
#include "globDefs.h"
#include "protos.h"

/*-- ---------- ---------- ----------*/
/*      cmplx()                      */
/*  Dos numeros reales los convierte */
/*  en un numero complejo            */
/*-- -- ---- ---- -- ---- ----------*/

struct complex cmplx(  real A, real B)
{
struct complex result;

result.x = A;
result.y = B;
return( result);
}

/*---------- ---------- ----------*/
/*  cAdd()                       */
/*  suma de dos numero complejos */
/*---- - ---- ---- ----------*/
struct complex cAdd(
                                    struct complex A,
                                    struct complex B)
{
struct complex result;

result.x = A.x + B.x;
result.y = A.y + B.y;
return( result);
}

/*-- -- ---------- -- ----------*/
/*  cSub()                       */
/* resta de dos numeros complejos*/
/*------ -- ---------- ----------*/

struct complex cSub(
                                    struct complex A,
                                    struct complex B)
{
struct complex result;

result.x = A.x - B.x;
result.y = A.y - B.y;
return( result);
}

/*-- -- ---- ---- -- ----------*/
/*  cMag()                       */
/*  Se obtiene la magnitud de un */
/*  numero complejo              */
/*---------- ---------- ----------*/

real cMag(struct complex A)
{
real result;
result = sqrt(A.x*A.x + A.y*A.y);
return( result);
}
```

```
/*                                          */
/*  cAbs()                                  */
/*  Se obtiene el valor absoluto            */
/*                                          */

real cAbs(struct complex A)
{
real result;
result = sqrt(A.x*A.x + A.y*A.y);
return( result);
}

/*                                          */
/*  cdAbs()                                 */
/*  Se obtiene el valor absoluto            */
/*                                          */

double cdAbs(struct complex A)
{
double result;
result = sqrt(A.x*A.x + A.y*A.y);
return( result);
}

/*                                          */
/*  arg()                                   */
/*  Se obtiene el argumento del complejo    */
/*                                          */

real arg(struct complex A)
{
real result;
if( (A.x == 0.0)  && (A.y == 0.0) )  {
        result = 0.0;
}
else    {
        result = atan2( A.y, A.x );
}
return( result);
}

/*                                          */
/*  cSqrt()                                 */
/*                                          */

struct complex cSqrt( struct complex A){
struct complex result;
double r, theta;

r = sqrt(cdAbs(A));
theta = arg(A)/2.0;
result.x = r * cos(theta);
result.y = r * sin(theta);
return( result);
}

/*                                          */
/*  cMult()                                 */
/*  Se obtiene la multiplicacion            */
/*  de dos numero complejos                 */
/*                                          */

struct complex cMult(struct complex A, struct complex B) {
struct complex result;

result.x = A.x*B.x - A.y*B.y;
result.y = A.x*B.y + A.y*B.x;
return( result);
}
```

```
/*                                                    */
/*  sMult()                                           */
/*  se obtiene la multiplicacion de un                */
/*  numero real con un complejo                       */
/*                                                    */

struct complex sMult(real a,  struct complex B){
struct complex result;

result.x = a*B.x;
result.y = a*B.y;
return( result);
}

/*                                                    */
/*  cDiv()                                            */
/*  se obtiene la división de dos                     */
/* numero complejos                                   */
/*                                                    */
struct complex cDiv(struct complex numer, struct complex denom){

real bottom,real_top,imag_top;
struct complex result;

bottom = denom.x*denom.x + denom.y*denom.y;
real_top = numer.x*denom.x + numer.y*denom.y;
imag_top = numer.y*denom.x - numer.x*denom.y;
result.x = real_top/bottom;
result.y = imag_top/bottom;
return( result);
}
```

ANEXO IV

FUNCIONES AUXILIARES

FUNCIONES AUXILIARES

```
/* ---------- ---------- INICIALIZACION ---------- ---------- */

/*                                                              */
/* funcion que inicializa las variables                         */
/* empleadas en el menu principal                               */
/*--------------------------------------------------------------*/

void inicializar(void){


n='\x0';
orden =0;
frecuencia= magnitud= 0;
delta =0;
puntos =0;
rizo =0;
frecuencia_P= frecuencia_R= 0;
A_p=0;
A_r=0;

pasos= 0;
lamdau = lamdal= 0;
valor=0;
tipo=0;
tau=0;
escaladB=0;
M=N=0;
centro=0;

}
```

```
/* ... MISCELANEOS ... */

#include <stdlib.h>
#include <math.h>
#include <ctype.h>
#include "globDefs.h"
#include "protos.h"

/* ... sincSqrd() ... */

real sincSqrd( real x){
real result;
if( x==0.0)
        {
        result = 1.0;
        }
else
        {
        result = sin(x)/x;
        result = result * result;
        }
return(result);
}

/* ... sinc() ... */

real sinc( real x)
{
real result;
if( x==0.0)
        {
        result = 1.0;
        }
else
        {
        result = sin(x)/x;
        }
return(result);
}


/* ... acosh() ... */

real acosh( real x)
{
real result;
result = log(x+sqrt(x*x-1.0));
return(result);
}

/* ... pause() ... */

void pause( logical enabled){
char inputString[20];
if(enabled) {
        printf("enter anything to continue\n");
        gets(inputString);
        }
return;
}
```

```
/*                                                      */
/*    bitRev()                                          */
/*                                                      */

int bitRev(    int L, int N)
{
int work, work2, i, bit;

work2 = 0;
work = N;
for(i=0; i<L; i++) {
        bit = work%2;
        work2 = 2 * work2 + bit;
        work /=2;
        }
return(work2);
}

/*                                                      */
/*    log2()                                            */
/*                                                      */

int log2(  int N )
{
int work, result;

result = 0;
work = N;
for(;;) {
        if(work == 0) break;
        work /=2;
        result ++;
        }
return(result-1);
}

/*                                                      */
/*    ipow()                                            */
/*                                                      */

real ipow(     real x,
                        int k)
{
real result;
int n;
if(k==0)
        {result = 1.0;}
else
        {result = x;
        for( n=2; n<=k; n++)
                { result = result * x;}
        }
return(result);
}
```

ANEXO V

PROGRAMA PRINCIPAL

PROGRAMA PRINCIPAL

```
/*******************************************/
//
//          PROGRAMA  PRINCIPAL
//
//
//             PROYECTO DE
//      INGENIERIA ELECTRONICA I y II
//
//    PROFESOR: JESUS BARRIOS
//
//    ALUMNOS: HERNANDEZ GARCIA IVAN
//             MOLINA SALAS LAURA ANGELICA
//
//
/*******************************************/

#include <math.h>
#include <stdio.h>
#include <conio.h>
#include "llamada.c"
#include "protos.h"     //contiene las rutinas de las funciones desglosadas
#include "inicio.c"

/*-----------------VARIABLES GLOBALES ---------------------*/

char salir = 'n',normalizacion, resp;
char opcion;
int x1,yy;

void sonido(void);

/*-------------FUNCION DE INICIO DEL PROGRAMA---------*/

void inicio_fin(void)
{
   while (salir!= '\x1B')  //Condici¢n <ESC> para salir del programa general
       {
          clrscr();
          inicializar();
          menu();
          gotoxy(5,12);printf ("Para salir oprime <ESC> o cualquier tecla para continuar ");
          salir = getch();
          clrscr();
       }
}

/*-----------------MENU PRINCIPAL-------------------*/

void menu(void)
 {
    opcion='\x0';
    while(opcion!='5'){      //condicion para salir de menu

              clrscr();x1=23;
              gotoxy(30,4);printf (" MENU PRINCIPAL");
              gotoxy(x1,6); printf (" 1. FILTROS ANALOGICOS                ");
              gotoxy(x1,7); printf (" 2. TRANSFORMADA DISCRETA DE FOURIER ");
              gotoxy(x1,8); printf (" 3. DISE¥O DE FILTROS FIR             ");
              gotoxy(x1,9); printf (" 4. DISE¥O DE FILTROS IIR             ");
              gotoxy(x1,10);printf (" 5. SALIR DEL PROGRAMA                ");
              gotoxy(27,12);printf(" Elija su OPCION:                     ");
              gotoxy(45,12);opcion=getch();


                 switch(opcion){
                 case '1':clrscr();
                               menu1();     //  lista los filtros analogicos
                               break;
```

```
                case '2':clrscr();              //  Transformada de Fourier
                                menu2();
                                break;
                case '3':clrscr();
                                menu3();      //  lista opciones para diseño de FIR
                                break;
                case '4':clrscr();
                                menu4();      //  lista opciones para diseño de IIR
                                break;
                case '5':clrscr();  //   salir
                        break;

                default:{
                        gotoxy(x1,15);printf("tecla incorrecta intente de nuevo");
                        //sonido();
                        gotoxy(x1,16);printf("     <oprima cualquier tecla >");
            getch();
            }
          } // fin del switch
        }  // fin del while
      return;
   } //fin de menu

/*--  --  ---------MENU FILTROS ANALOGICOS----------------*/

void menu1(void)
{

    while(opcion!='5'){       //condicion para salir de menu
         clrscr();
         x1=33;
         gotoxy(30,4);printf (" FILTROS ANALOGICOS  ");
         gotoxy(x1,6);printf  (" 1. BUTTERWORTH      ");
         gotoxy(x1,7);printf  (" 2. CHEBYSHEV        ");
         gotoxy(x1,8);printf  (" 3. ELIPTICO         ");
         gotoxy(x1,9);printf  (" 4. BESSEL           ");
         gotoxy(x1,10);printf (" 5. SALIR            ");
         gotoxy(30,12);printf(" Elija su OPCION:    ");
         gotoxy(48,12);opcion=getch();

         switch (opcion){

         case '1': clrscr();            // filtros Butterworth
                        menuBut();
                        break;
         case '2': clrscr();            // filtros Chebyshev
                        menuChe();
                        break;
         case '3': clrscr();            // filtros Elipticos
                        menuEli();
                        break;
         case '4': clrscr();            // filtros Bessel
                        Bess();
                        break;
         case '5': break;

         default:{
                    gotoxy(25,15);printf("tecla incorrecta intente de nuevo");
                    gotoxy(25,16);printf("     <oprima cualquier tecla >");
                    getch();
                  }
         } // fin del switch
       }
       opcion='\x0';
    return;
} //fin de menu1
```

```
/*-----------------MENU DE TRANSFORMADA DISCRETA DE FOURIER-----------------*/

void menu2(void){
     while(opcion!='4'){        //condicion para salir de menu
          clrscr();
          x1=30;
          gotoxy(20,4);printf (" TRANSFORMADA DISCRETA DE FOURIER    ");
          gotoxy(x1,6); printf (" 1. DFT1                            ");
          gotoxy(x1,7); printf (" 2. DFT2                            ");
          gotoxy(x1,8); printf (" 3. FFT  (rapida)                   ");
          gotoxy(x1,9); printf (" 4. SALIR                           ");
          gotoxy(20,11);printf(" Elija su OPCION:                   ");
          gotoxy(38,11);opcion=getch();

          switch (opcion){
                 case '1': clrscr();
                           DFT1();
                           break;
                 case '2': clrscr();
                           DFT2();
                           break;
                 case '3': clrscr();
                           FFT();
                           break;
                 case '4': break;
          default:{
                 gotoxy(18,15);printf("tecla incorrecta intente de nuevo");
                 gotoxy(18,16);printf("     <oprima cualquier tecla >");
                 getch();
          }
        } // fin del switch
      }
         opcion='\x0';
    return;
  } // fin menu2

/*-----------------DISEÑO DE FILTROS FIR (RIF)-----------------*/

void menu3(void){
     while(opcion!='4'){     //condicion para salir de menu
          inicializar();
          clrscr();
          x1=23;
           opcion='\x0';
          gotoxy(x1,5);printf ("     DISE¥O DE FILTROS RIF      ");
          gotoxy(x1,7);printf (" 1. RESPUESTA DE FILTROS RIF    ");
          gotoxy(x1,8);printf (" 2. POR SERIES DE FOURIER       ");
          gotoxy(x1,9);printf (" 3. POR MUESTREO EN FRECUENCIA  ");
          gotoxy(x1,10);printf(" 4. SALIR                       ");
          gotoxy(x1,12);printf("    Elija su OPCION:            ");
          gotoxy(45,12);opcion=getch();

          switch (opcion){
           case '1': clrscr();
                     rff();
                     break;
           case '2': clrscr();
                     series();
                     break;
           case '3': clrscr();
                     muestreo();
                     break;
           case '4': break;
           default:{
                 gotoxy(x1,15);printf("tecla incorrecta intente de nuevo");
                 gotoxy(x1,16);printf("     <oprima cualquier tecla >");
                 getch();
           }
        } // fin del switch
     } // fin del while
        opcion='\x0';
   return;
  }
```

```
/*  ------  --  ---DISEÑO DE FILTROS IIR ------ ---  ---------------*/

void menu4(void){
     while(opcion!='5'){     //condicion para salir de menu
          clrscr();
          x1=24;
           gotoxy(x1,4);printf ("     DISEÑO DE FILTROS IIF    ");
           gotoxy(x1,6);printf (" 1. RESPUESTA DE FILTROS IIF  ");
           gotoxy(x1,7);printf (" 2. RESPUESTA IMPULSIVA       ");
           gotoxy(x1,8);printf (" 3. PASO                      ");
           gotoxy(x1,9);printf (" 4. BILINEAL                  ");
           gotoxy(x1,10);printf(" 5. SALIR                     ");
           gotoxy(x1,12);printf("    Elija su OPCION:          ");
           gotoxy(46,12);opcion=getch();

           switch (opcion){

           case '1': clrscr();
               respImpulsoInfini();
               break;
           case '2': clrscr();
               impulso();
               break;
           case '3': clrscr();
               paso();
               break;
           case '4': clrscr();
               bilineal();
               break;

           case '5': break;

           default:{
              gotoxy(x1,15);printf(" tecla incorrecta intente de nuevo ");
              gotoxy(x1,16);printf("   <oprima cualquier tecla.. >     ");
              getch();
           }
        } // fin del switch
    }
        opcion='\x0';
  return;
 }    // fin menu4


/*---- ---- ---------- ---- PROGRAMA PRINCIPAL----------------------*/

void main(void)
  {
     inicio_fin();
     return 0;
  }
```

PROGRMA AUXILIAR DEL PROGRAMA PRINCIPAL

```
/*-------------------------------------*/
/*                                     */
/* Este programa tiene las funciones   */
/* del menu principal                  */
/*                                     */
/*-------------------------------------*/


#include <math.h>
#include <stdio.h>
#include "globDefs.h"
#include "protos.h"
#include "cmpxmath.c"
#include <conio.h>

/*------Programas importados de la referencia "BRITTON RORABAUGH" ----*/

#include "butter.c"     //Butterworth
#include "cheby.c"      //Chebyshev
#include "eliptic.c"    //Eliptica
#include "bessel.c"     //Bessel
#include "tdf.c"        //TDF
#include "fir.c"        //FIR
#include "ventana.c"    //FIR Fourier (ventanas)
#include "muestreo.c"   //FIR muestreo
#include "iir.c"        //IIR
//#include "bilineal.c"    //IIR Bilineal


/*-------------- Declaracion de  Variables  Globales ----------------*/

real  delta,rizo, frecuencia,fase, magnitud,yval[100], hCero, pCero,alpha;
real  frecuencia_P, frecuencia_R, A_p, A_r, A_r_nueva, valor, valor2;
real  aa[25], bb[25], cc[25], coef[25],hh[25], ventana[30],hd[25];
real  AA[25];
real  retardoGrupo, valor;
real  ventanaout[30],lamdau,lamdal,tau;
real  r,rd,rtc,rtd;

 int  escaladB, pasos, normalizAmplitud,centro, M,N,tipo;
 int  y,puntos,orden,numSecs,dfm;
char  n,normalizacion,resp;
struct complex x[256], xx[256];


//void continuar(void);
//void error (void);
void otro(void);


/*-------------------FILTROS BUTTERWORTH--------------------*/
void menuBut(void){

 int cont;
 resp = 's';
 while (resp == 's' || resp == 'S')
 {
        inicializar();
        clrscr();y=22;
        gotoxy(27,4);printf  (" FILTROS BUTTERWORTH         ");
        gotoxy(y,6);printf   (" 1. Respuesta en frecuencia  ");
        gotoxy(y,7);printf   (" 2. Respuesta impulsiva      ");
        gotoxy(y,8);printf   (" 3. Regresar al menu anterior");
        gotoxy(27,10);printf  (" Elija su OPCION:           ");
        gotoxy(45,10);n=getche();
```

```
        if(n=='1'){
                clrscr();y=18;
                gotoxy(27,4);printf  (" FILTROS BUTTERWORTH        ");
                gotoxy(25,6);printf   (" Respuesta en frecuencia ");
                gotoxy(y,9); printf  ("      Orden del filtro?:  ");
                gotoxy(43,9); scanf  (" %d",&orden);
                gotoxy(y,10); printf  (" Valor de la frecuencia   ");
                gotoxy(y,11); printf  ("          (Normalizada):  ");
                gotoxy(43,11); scanf  (" %lf", &frecuencia);
                clrscr();
                cont=respuestaFrecbutterworth(orden, frecuencia, &magnitud, &fase);
                gotoxy(12,cont+2);printf  ("Y la Magnitud y Fase son: ");
                gotoxy(18,cont+3); printf  ("Magnitud = %lf(dB)   Fase = %lfø",magnitud,fase);
                otro();
        }
        else
        if(n=='2'){
                clrscr();y=12;
                gotoxy(27,4);printf   (" FILTROS BUTTERWORTH        ");
                gotoxy(26,6);printf   (" Respuesta  impulsiva       ");
                gotoxy(y,9);printf   ("              Orden del filtro?:  ");
                gotoxy(45,9);scanf   ("%d",&orden);
                gotoxy(y,10);printf   ("      Dame el valor de delta_T    ");
                gotoxy(y,11);printf   ("(Separaci¢n entre puntos, segs):  ");
                gotoxy(45,11);scanf   ("%lf", &delta);
                gotoxy(y,12);printf   ("       Dame el n£mero de puntos:  ");
                gotoxy(45,12);scanf   ("%d",&puntos);
                clrscr();
                respuestaImpulsivabutterw(orden, delta, puntos, yval);        //La funci¢n se encarga
                otro();
              }
         else
         if (n=='3') break;
         else {
               error(); continuar();
         }
   }
} // fin de menuBut

/*.....----------FILTROS CHEBYSHEV----------*/

void menuChe(void){

 resp = 's';
 while (resp == 's' || resp == 'S')
 {
        clrscr();y=22;
        gotoxy(27,4);printf  (" FILTROS CHEBYSHEV            ");
        gotoxy(y,6);printf   (" 1. Respuesta en frecuencia   ");
        gotoxy(y,7);printf   (" 2. Respuesta impulsiva       ");
        gotoxy(y,8);printf   (" 3. Regresar al menu anterior ");
        gotoxy(27,10);printf  (" Elija su OPCION:             ");
        gotoxy(45,10);n=getche();

        if(n=='1'){
                clrscr();
                gotoxy(28,4);printf   (" FILTROS CHEBYSHEV       ");
                gotoxy(23,6);printf   (" Respuesta en frecuencia ");
                gotoxy(12,9);printf  ("      Orden del filtro: ");
                gotoxy(36,9); scanf  ("%d",&orden);
                gotoxy(12,10);printf  ("   Valor del rizo (dB): ");
                gotoxy(36,10); scanf  ("%lf",&rizo);
                gotoxy(9,11);printf   ("   Normalizaci¢n (para 3dB teclea 3): ");
                gotoxy(48,11);cscanf  ("%c",&normalizacion);
                puts("");
                gotoxy(9,12);printf   ("Valor de la frecuencia (Normalizada): ");
                gotoxy(48,12); scanf  ("%lf", &frecuencia);
                respuestaFrecChebyshev(orden, rizo, normalizacion, frecuencia,
                &magnitud, &fase);
                gotoxy(10,14);printf  ("El resultado es: ");
                gotoxy(10,16); printf ("Magnitud = %lf(dB)   Fase = %lfø",magnitud,fase);
                otro();
        }
```

```
        else
        if(n=='2'){
                clrscr();y=15;
                gotoxy(28,4);printf  (" FILTROS CHEBYSHEV          ");
                gotoxy(25,6);printf   (" Respuesta  impulsiva      ");
                gotoxy(y,9);  printf  ("     Orden del filtro: ");
                gotoxy(38,9);  scanf  ("%d",&orden);
                gotoxy(y,10);  printf  ("  Valor del rizo (dB): ");
                gotoxy(38,10);  scanf  ("%lf",&rizo);
                gotoxy(12,11); printf  ("Normalizaci¢n (para 3dB teclea 3): ");
                gotoxy(47,11); cscanf  ("%c",&normalizacion);
                puts("");
                gotoxy(y,12);  printf  ("      Dame el valor de delta_T     ");
                gotoxy(14,13); printf  ("(Separaci¢n entre puntos, segs):   ");
                gotoxy(47,13);  scanf  ("%lf", &delta);
                gotoxy(13,14); printf  ("       Dame el n£mero de puntos:   ");
                gotoxy(47,14);  scanf  ("%d",&puntos);
                respuestaImpulsivaCheby(orden, rizo, normalizacion, delta,puntos,yval);
                /*Tambi,n la funci¢n imprime sus resultados */
                otro();
        }
        else
        if (n=='3') break;

        else{
          error();continuar();
        }
  }

} //fin de menuChe

/*-------------------------FILTROS ELIPTICO-------------------------*/
void menuEli(void){
int xx;

resp = 's';
while (resp == 's' || resp == 'S')
  {
        clrscr();y=18;
        gotoxy(30,4);printf  (" FILTROS ELIPTICO                                  ");
        gotoxy(y,6);  printf (" 1. Estimar orden para un Filtro El¡ptico          ");
        gotoxy(y,7);  printf (" 2. Calcular coeficientes                          ");
        gotoxy(y,8);  printf (" 3. Regresar al menu anterior                    ");
        gotoxy(27,10); printf ("    Elija su OPCION:                               ");
        gotoxy(48,10);n=getche();

        if (n == '1')
        {
                clrscr();y=13;
                gotoxy(18,4); printf  ("   Estimar orden para un Filtro El¡ptico    ");
                gotoxy(y,6);  printf  ("    Frecuencia l¡mite en la banda de paso:  ");
                gotoxy(57,6);  scanf  ("%lf", &frecuencia_P);
                gotoxy(y,7);  printf  (" Frecuencia l¡mite en la banda de rechazo:  ");
                gotoxy(57,7);  scanf  ("%lf", &frecuencia_R);
                gotoxy(y,8);  printf  ("  Perdida m xima en la banda de paso (dB):  ");
                gotoxy(57,8);  scanf  ("%lf", &A_p);
                gotoxy(y,9);  printf  (" Nivel m¡nimo en la banda de rechazo (dB):  ");
                gotoxy(57,9);  scanf  ("%lf", &A_r);
                estimarOrden(frecuencia_P, frecuencia_R, A_p, A_r, &orden, &A_r_nueva);
                gotoxy(18,12); printf  (" El orden es: %d", orden);
                gotoxy(18,13); printf  (" El nivel m¡nimo recalculado es: %lf(dB)", A_r_nueva);
                otro();
         }
         else
         if (n == '2')
         {
                clrscr();y=9;
                gotoxy(16,4);  printf (" Calcular coeficientes de un Filtro El¡ptico ");
                gotoxy(y,6);   printf ("    Frecuencia l¡mite en la banda de paso:    ");
                gotoxy(52,6);   scanf ("%lf", &frecuencia_P);
                gotoxy(y,7);   printf (" Frecuencia l¡mite en la banda de rechazo:    ");
                gotoxy(52,7);   scanf ("%lf", &frecuencia_R);
```

```
                gotoxy(y,8);   printf ("  Perdida m xima en la banda de paso (dB):    ");
                gotoxy(52,8);   scanf ("%lf", &A_p);
                gotoxy(y,9);   printf ("  Orden del filtro:                          ");
                gotoxy(30,9);   scanf ("%d",&orden);
                calcularCoefic(frecuencia_P, frecuencia_R, A_p, orden, aa, bb, cc,
                 &numSecs, &hCero, &pCero);
                gotoxy(10,11);  printf ("      El n£mero de secciones es:  %d", numSecs);
                gotoxy(10,12);  printf (" El valor de Ho  normalizado es:  %lf", hCero);
                gotoxy(10,13);  printf (" El valor de Po  normalizado es:  %lf", pCero);
                gotoxy(10,15);  printf (" Los coeficientes normalizados de las secciones son: ");
                gotoxy(10,17);  printf (" Secci¢n      ai            bi             ci ");
                for (puntos=1; puntos <= numSecs; puntos++){
                       y=17+puntos;
                       gotoxy(10,y);printf ("   %d      %lf     %lf     %lf
",puntos,aa[puntos],bb[puntos],cc[puntos]);
                       y++;
                }
                gotoxy(15,23);  printf (" Desea reescalar los coeficientes (s/n)?  ");
                gotoxy(57,23); resp = getche();
                if (resp == 's' || resp == 'S')
                {       y=10;
                       clrscr();
                       gotoxy(y,6);  printf (" A qu, valor de frecuencia de referencia?  ");
                       gotoxy(54,6);   scanf ("%lf",&alpha);
                       reescalarCoef(orden, aa, bb, cc, &hCero, &pCero, alpha);
                       gotoxy(y,8); printf (" El valor de Ho reescalado es:  %lf", hCero);
                       gotoxy(y,9); printf (" El valor de Po reescalado es:  %lf", pCero);
                       gotoxy(y,10);printf (" Los coeficientes reescalados de las secciones son:  ");
                       gotoxy(y,12);printf (" Secci¢n      ai            bi                ci ");
                       for (puntos=1; puntos <= numSecs; puntos++){
                         y=12+puntos;
                         gotoxy(10,y);printf ("   %d      %lf      %lf      %lf
",puntos,aa[puntos],bb[puntos],cc[puntos]);
                         y++;
                       }
                }
                resp = 's';
                while (resp == 's' || resp == 'S')
                {      xx=y+1;
               gotoxy(15,23);  printf ("                                              ");
               gotoxy(10,xx); printf (" Desea calcular la respuesta en alguna frecuencia (s/n)? ");
                       gotoxy(68,xx);resp = getche();
                       if (resp == 's' || resp == 'S')
                       {
                              clrscr();
                              gotoxy(y,8); printf (" Cu l es la frecuencia? ");
                              gotoxy(y+25,8);  scanf ("%lf",&frecuencia);
                              calcuRespuestaFrec(orden, aa, bb, cc, hCero, pCero, frecuencia,
                              &magnitud, &fase);
                              gotoxy(15,11); printf (" La magnitud es: %lf(dB)",magnitud);
                              gotoxy(15,12); printf ("     La fase es: %lfø\n",fase);
                       }
                }// fin while interno
        } // fin de if2
        else
        if(n=='3') break;
        else{
        error();continuar();
        }
  }
}// fin de menuEli
```

```
/*-----------------------FILTROS BESSEL---------- ----------------*/

void Bess(void) {

 resp = 's';
 while (resp == 's' || resp == 'S')

 {
        clrscr();y=15;
        gotoxy(30,4); printf (" FILTROS BESSEL                                  ");
        gotoxy(y,6);  printf (" 1. Calcular los coeficientes                    ");
        gotoxy(y,7);  printf (" 2. Calcular la respuesta en frecuencia          ");
        gotoxy(y,8);  printf (" 3. Regresar al menu anterior                    ");
        gotoxy(27,10); printf ("    Elija su OPCION:                             ");
        gotoxy(48,10);n=getche();

        if (n == '3') break;
        else
        if (n == '1'){
              y=16;clrscr();
              gotoxy(24,4); printf (" Calcular los coeficientes ");
              gotoxy(y,6);  printf ("   Orden del filtro (menor a 24):  ");
              gotoxy(50,6);  scanf ("%d",&orden);
              gotoxy(y,8);  printf ("           Tipo de normalizacion   ");
              gotoxy(y,9);  printf ("  (d)esnormalizada/(n)ormalizada:  ");
              gotoxy(50,9); normalizacion = getche();
              clrscr();
              coeficientesBessel(orden, normalizacion, coef);  //la misma funcion se encarga de
              otro();                                          //imprimir el resultado.
        }

        else
        if (n == '2'){
              Uno:
                clrscr();
                y=10;
                gotoxy(20,4);  printf (" Calcular la respuesta en frecuencia       ");
                gotoxy(y,6);   printf ("                          Orden del filtro:  ");
                gotoxy(52,6);   scanf ("%d",&orden);
                gotoxy(y,8);   printf ("    Tipo de aproximaci¢n para el retardo   ");
                gotoxy(y,9);   printf (" de grupo (d)esnormalizada/(n)ormalizada:  ");
                gotoxy(52,9); normalizacion = getche();
                gotoxy(15,10); printf ("     ¨C£al es la frecuencia?: ");
                gotoxy(45,10);  scanf ("%lf",&frecuencia);
                gotoxy(15,11); printf (" ¨C£al es el valor de delta?: ");
                gotoxy(45,11);  scanf ("%lf",&delta);
                respuestaFrecBessel(orden, coef, frecuencia, &magnitud, &fase);
                retardoGrupoBessel(orden, coef, frecuencia, delta, &retardoGrupo);
                gotoxy(18,13); printf ("   La magnitud es:  %lf(dB)",magnitud);
                gotoxy(18,14); printf ("       La fase es:  %lfø",fase);
                gotoxy(18,15); printf (" El retardo de grupo es:  %lf segs.",retardoGrupo);
                gotoxy(15,20); printf ("¨Desea para otra frecuencia (s/n)?");
                gotoxy(51,20); resp = getche();
                if (resp == 's'|| resp== 'S')
                goto Uno;
         }
         else{
         error(); continuar();
         }
  }// fin de while
}// fin de menuBess
```

```
/*  ------------------------FILTROS DFT1-------------------------------*/
void DFT1(void){
int i;
                i=9;
                gotoxy(25,4);  printf (" Tranformada Discreta de Fourier I ");
                gotoxy(15,7);  printf (" ¨C£al es el n£mero de puntos (N)? ");
                gotoxy(50,7);   scanf ("%d",&N);
                for (puntos=0; puntos<N; puntos ++){
                     gotoxy(15,i);    printf (" Dame el valor de x[%d]= ",puntos);
                     gotoxy(40,i);     scanf ("%lf",&valor);
                     x[puntos].x = valor;
                     x[puntos].y = 0;
                     i++;
                }
                dft(x, xx, N);   // manda llamar la funcion
                continuar();clrscr();
                gotoxy(23,6); printf (" Los datos de la transformada son: ");
                i=8;
                for (puntos=0; puntos<N; puntos ++){                    // se introducen las muestras
                    gotoxy(20,i);printf ("xx[%d]= %lf +j %lf",puntos,xx[puntos].x,xx[puntos].y);
                    i++;
                }
                 //fase[puntos]= arg(x[puntos]);
                continuar();
}

/*  ------------------------FILTROS DFT2-------------------------------*/
void DFT2(void){
               int i;          i=9;
                gotoxy(25,4);  printf (" Tranformada Discreta de Fourier II ");
                gotoxy(15,7);  printf (" ¨C£al es el n£mero de puntos (N)? ");
                gotoxy(50,7);   scanf ("%d",&N);
                for (puntos=0; puntos<N; puntos ++){
                       gotoxy(15,i);  printf (" Dame el valor de x[%d]= ",puntos);
                       gotoxy(40,i);   scanf ("%lf",&valor);
                       x[puntos].x = valor;
                       x[puntos].y = 0;
                       i++;
                }
                dft2(x,xx,N);
                continuar();clrscr();
                gotoxy(23,6); printf (" Los datos de la transformada son: ");
                i=8;
                for (puntos=0; puntos<N; puntos ++){                    // se introducen las muestras
                     gotoxy(20,i);printf ("xx[%d]= %lf +j %lf",puntos,xx[puntos].x,xx[puntos].y);
                     i++;
                }
                continuar();
} // fin de DFT2

/*  ------------------------FILTROS FFT -------------------------------*/
void FFT(void){
               int i;          i=9;
                gotoxy(25,4);  printf (" Tranformada Discreta de Fourier ");
                gotoxy(15,7);  printf (" ¨C£al es el n£mero de puntos (N)? ");
                gotoxy(50,7);   scanf ("%d",&N);
                for (puntos=0; puntos<N; puntos ++){
                       gotoxy(15,i);  printf (" Dame el valor de x[%d]= ",puntos);
                       gotoxy(40,i);   scanf ("%lf",&valor);
                       x[puntos].x = valor;
                       x[puntos].y = 0;
                       i++;
                 }
                fft(x, xx, N);
                continuar();clrscr();
                gotoxy(23,6); printf (" Los datos de la transformada son: ");
                i=8;
                for (puntos=0; puntos<N; puntos ++){                    // se introducen las muestras
                       gotoxy(20,i);printf ("xx[%d]= %lf +j %lf",puntos,xx[puntos].x,xx[puntos].y);
                       i++;
                }
                continuar();
}
```

```
/*-------------------------RESPUESTA FILTRO FINITO ----------------*/

void rff(void)
{
 int i;

       y=14;
       gotoxy(23,4);printf (" RESPUESTA AL IMPULSO FINITO   ");
       gotoxy(25,6);   printf (" TIPO DE FILTRO                    ");
       gotoxy(y,8);    printf (" 1. Simetrico  PAR,  longitud IMPAR ");
       gotoxy(y,9);    printf (" 2. Simetrico  PAR,  longitud PAR   ");
       gotoxy(y,10);   printf (" 3. Simetrico IMPAR, longitud IMPAR ");
       gotoxy(y,11);   printf (" 4. Simetrico IMPAR, longitud PAR   ");
       gotoxy(y,12);   printf (" 5. Regresar al menu                ");
       gotoxy(15,14);  printf (" Elija su opcion: ");
       gotoxy(34,14);  tipo=getche();

       if(tipo == '5')
          continuar();

       else{

        gotoxy(y,15);  printf ("  Numero de pasos: ");
        gotoxy(y,26);  scanf ("%d", &pasos);
        gotoxy(y,16);  printf (" Numero de puntos: ");
        gotoxy(y,26);  scanf ("%d",&N);
        clrscr();
        gotoxy(23,4);printf (" RESPUESTA AL IMPULSO FINITO   ");
         i=6;
         for (puntos=0; puntos<N; puntos ++){
              gotoxy(y,i); printf (" Dame el valor del punto x[%d]= ",puntos);
              gotoxy(y,39);scanf ("%lf",&valor);
              hh[puntos]= valor;
              i++;
         }
         gotoxy(y,i+1);printf (""Desea su resultado en escala de dBs <S/N>:? ");
         n=getche();
         if (n == 's' || n == 'S')
             escaladB=1;
         else
             escaladB=0;
         respuestaImpulFinita(tipo,pasos,hh,escaladB,puntos,hd);
         n='\x0';clrscr();
         gotoxy(y,6); printf("Desea normalizar sus resultados <S/N>:? ");
         n=getche();
         if (n == 's'|| n == 'S')
             {respuestaNormalizada(escaladB,puntos,hd);}
         else
            if (n == 'n'|| n == 'N')
              {continuar();}
            else {
              error();continuar();
            }
        } // fin else

 }// fin de rff
```

```
/*------------------------RIF POR SERIES DE FOURIER --------------------*/

void series(void){

  resp = 's';
  while (resp == 's' || resp == 'S')    {
            inicializar();
            clrscr();y=21;
            gotoxy(25,3);printf(" RIF por series de Fourier ");
            gotoxy(y,5);printf("1. Respuestas para filtros ideales      ");
            gotoxy(y,6);printf("2. Respuesta rectangular y triangular  ");
            gotoxy(y,7);printf("3. Ventana rectangular, Hann y Hamming ");
            gotoxy(y,8);printf("4. Regresar al menu                    ");
            gotoxy(y,10);printf("   Elija su opcion: ");
            gotoxy(41,10);n=getche();

            if (n == '4') break;
            else
             if (n == '1'){
                  pasos=0;
                  do{
                   clrscr();y=17;
                   gotoxy(21,4);printf(" Respuestas para filtros ideales    ");
                   gotoxy(y,6);printf ("     Numero de pasos (max. 28): ");
                   gotoxy(50,6);scanf  ("%d",&pasos);
                   if(pasos>28){
                     gotoxy(25,8);printf("n£mero excedido");
                     getch();
                   }
                  }while(pasos>28);
                  gotoxy(y,7);printf (" El valor de lamda u: ");
                  gotoxy(40,7);scanf  ("%lf",&lamdau);
                  gotoxy(y,8);printf (" El valor de lamda l: ");
                  gotoxy(40,8);scanf  ("%lf", &lamdal);
                  clrscr();
                  pasaBajasIdeal(pasos,lamdau,hh);
                  clrscr();
                  pasaAltasIdeal(pasos,lamdal,hh);
                  clrscr();
                  pasaBandaIdeal(pasos,lamdal,lamdau,hh);
                  clrscr();
                  rechazaBandaIdeal(pasos,lamdal,lamdau,hh);
            }
            else
            if (n == '2'){
                    clrscr();escaladB=1;y=10;
                    normalizAmplitud=1;
                    gotoxy(20,4);printf  (" Respuesta Rectangular y Triangular ");
                    gotoxy(y,6); printf  (" Cual es la frecuencia:               ");
                    gotoxy(34,6); scanf  ("%lf",&frecuencia);
                    gotoxy(y,7); printf  ("       El valor de tau:               ");
                    gotoxy(34,7); scanf  ("%lf",&tau);
                    gotoxy(y,8); printf  (" Para ver el resultado en db oprima 1:  ");
                    gotoxy(49,8); scanf  ("%d",&escaladB);
                    gotoxy(y,9); printf  (" Cual es el valor de M:                 ");
                    gotoxy(33,9); scanf  ("%d",M);
                    r=respuestaRectangularCont(frecuencia,tau,escaladB);
                    y=15;
                    gotoxy(y,11);printf(" La respueta  rectangular CONTINUA es: %f", r);
                    getch();
                    rd=respuestaRectangularDisc(frecuencia,M,normalizAmplitud);
                    gotoxy(y,13);printf(" La respuesta rectangular DISCRETA es: %f",rd);
                    getche();
                    rtc=respuestaTriangularCont(frecuencia,tau,escaladB);
                    gotoxy(y,15);printf(" La respuesta  triangular CONTINUA es: %f",rtc);
                    getch();
                    rtd=respuestaTriangularDisc(frecuencia,M,normalizAmplitud);
                    gotoxy(y,17);printf(" La respuesta  triangular DISCRETA es: %f",rtd);
                    getch(); otro();
            }
```

```
            else
            if (n == '3' ){
                    clrscr();y=17;
                    gotoxy(12,4);printf(" VENTANA TRIANGULAR, HANN Y HAMMING ");
                    gotoxy(y,6);printf (" Introducir el valor de N: ");
                    gotoxy(44,6);scanf ("%d",&N);
                    gotoxy(y,7);printf(" Cual es el centro: ");
                    gotoxy(37,7);scanf("%d",&centro);
                    clrscr();
                    gotoxy(19,2);printf(" RESPUESTAS DE LAS DIFERENTES VENTANAS ");
                    ventanaTriangular(N,ventana);
                    clrscr();
                    makeLagVentana(N,ventana,centro,ventanaout);
                    clrscr();
                    makeDataVentana(N,ventana,ventanaout);
                    clrscr();
                    ventanaHann(N,ventana);
                    clrscr();
                    ventanaHamming(N,ventana);
                    getch();otro();
              }
              else{
                  error();
                  continuar();
              }

    } // fin de while

 }  //fin de series

/*-------------------RIF POR METODO DE MUESTREO-------------*/

void muestreo(void){
int i;
        clrscr(); y=12;
        gotoxy(8,4);printf (" DISENO DE FILTRO RIF POR METODO DE MUESTREO");
        gotoxy(25,6);  printf (" TIPO DE FILTRO                    ");
        gotoxy(y,8);   printf (" 1. Simetrico  PAR,  longitud IMPAR ");
        gotoxy(y,9);   printf (" 2. Simetrico  PAR,  longitud PAR   ");
        gotoxy(y,10);  printf (" 3. Simetrico IMPAR, longitud IMPAR ");
        gotoxy(y,11);  printf (" 4. Simetrico IMPAR, longitud PAR   ");
        gotoxy(y,12);   printf (" 5. Regresar al menu ");
        gotoxy(15,14);        printf (" opcion: ");
        gotoxy(24,14);        tipo=getche();

        if( tipo== '5'){
          continuar();}
        else{
          do{
             gotoxy(y,16);    printf (" Numero de puntos: ");
             gotoxy(31,16);   scanf ("%d",&N);
             if(N>28){
              gotoxy(15,18);  printf("n£mero de puntos excedido");
              getch();
              gotoxy(15,18); printf("                              ");
             }
           }while(N>28);

           gotoxy(10,20);printf("oprima una tecla para introducir los valores de los puntos... ");
           getche();clrscr();
           gotoxy(23,4);printf ("DISE¥O POR METODO DE MUESTREO ");
           i=6;

           for (puntos=0; puntos<N; puntos ++){
              gotoxy(y,i); printf (" Valor del punto x[%d]= ",puntos);
              gotoxy(y,39);scanf ("%lf",&valor);
              AA[puntos]= valor;
              i++;
           }

           dfm=disenoFiltroMuestreo(N,tipo,AA,hh);
```

```
            if (dfm >= 1 && dfm<=4){
               i=7;
               gotoxy(17,i);   printf("  La cantidad de valores no corresponde ");
               gotoxy(17,i+1); printf("            a la opcion elegida         ");
               gotoxy(26,18);  printf("  intente de nuevo ...  ");
               getche();
            }
          } //fin else

}

/*-------------------------IIR-------------------------*/

void respImpulsoInfini(void)

{
 int i,num;
       i=9;
       escaladB=0;
       gotoxy(25,3);  printf (" Tranformada Discreta de Fourier de H(z) ");
       gotoxy(15,5);  printf (" ¨C£al es el n£mero de puntos (n)? ");
       gotoxy(50,5);   scanf ("%d",&num);
       gotoxy(15,6);  printf (" numero de entradas previas M: ");
       gotoxy(46,6);   scanf ("%d",&M);
       /*gotoxy(15,7);  printf (" numero de salidas previas N: ");
       gotoxy(45,7);   scanf ("%d",&N);*/
       gotoxy(15,10); printf ("Oprima una tecla para introducir sus datos...");
       getche();
       clrscr();
       gotoxy(25,3);  printf (" Tranformada Discreta de Fourier de H(z) ");
       i=5;
       for (puntos=0; puntos<num; puntos ++){
            gotoxy(15,i);     printf (" Dame el valor de x[%d]= ",puntos);
            gotoxy(40,i);      scanf ("%lf",&valor);
            xx[puntos].x = valor;
            xx[puntos].y = 0;
            i++;
       }
       /*clrscr();
       i=5;
       for (puntos=0; puntos<N; puntos ++){
            gotoxy(15,i);     printf (" Dame el valor de y[%d]= ",puntos);
            gotoxy(40,i);      scanf ("%lf",&valor2);
            xx[puntos].x = valor2;
            i++;
       } */
       respuestaIInfinito(x,N,xx,num,M,escaladB,&magnitud, &fase);
       getche();
}

/*-------------------IIF BILINEAL-------------------*/

void bilineal(void)
{printf("correcto");getche();}


void error (void){
gotoxy(34,13);printf("E R R O R");

}
void continuar(void){
gotoxy(15,22);printf("   Oprima una tecla para continuar ... ");
getche();
}

void otro(void){
gotoxy(15,22);printf(" ¨Desea hacer otro calculo <S/N> ?: "); //de imprimir sus resultados
gotoxy(51,22);resp=getche();
continuar();

}
```