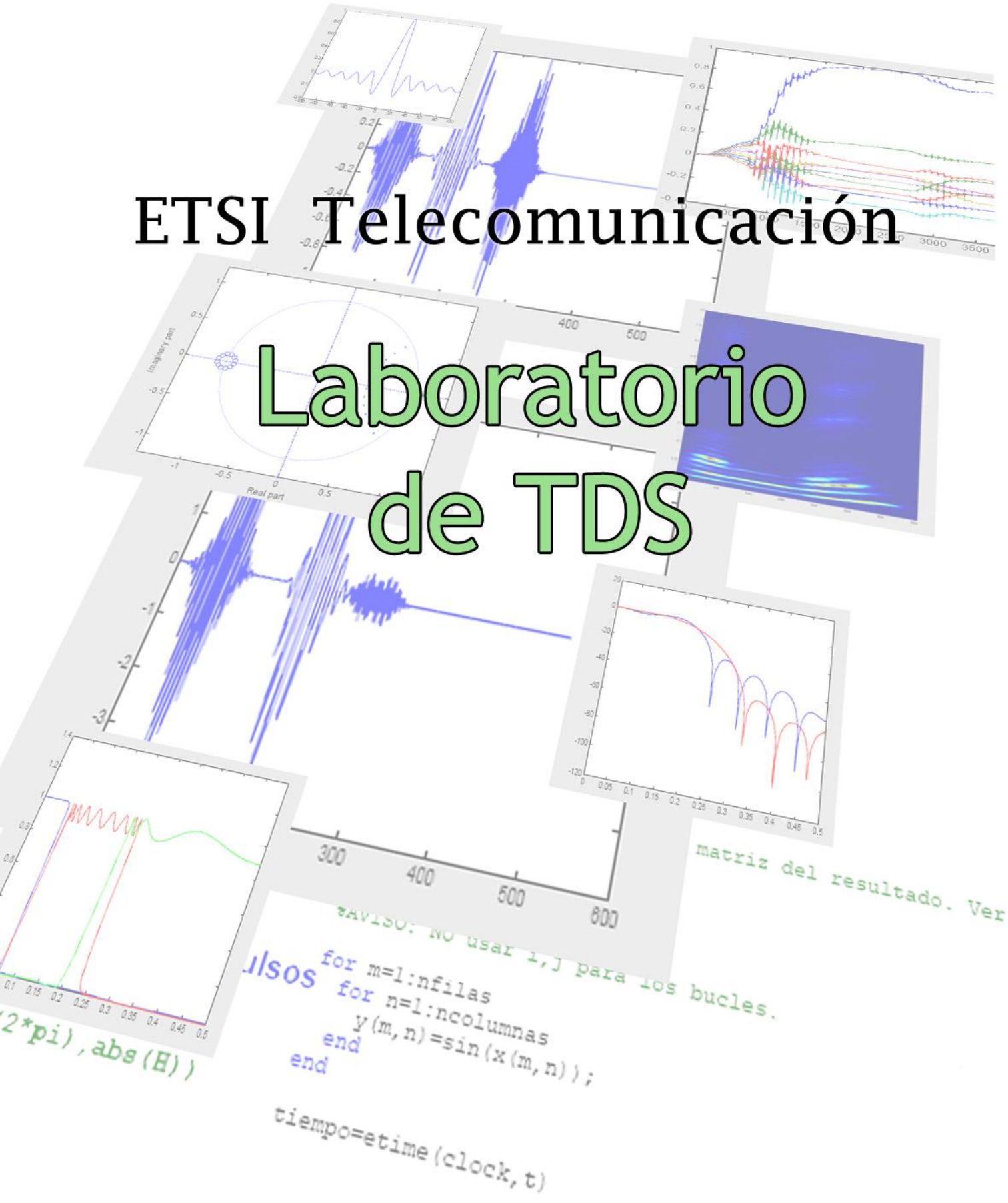
ETSI Telecomunicación

Laboratorio de TDS



Apuntes de Pak

Laboratorio de Tratamiento Digital de la Señal

Apuntes de Pak (Francisco José Rodríguez Fortuño)
ETSI Telecomunicación. Universidad Politécnica de Valencia.
Primer cuatrimestre de 4º curso
Curso 2006/2007

Contenido

- Resumen de las prácticas

Fecha de última actualización: 26 Agosto 2007

PRÁCTICA 1. Matlab y las señales discretas

- La aplicación Notebook
  - Matlab junto a Word
  - CNTRL + ENTER
  - Procura usar ';'

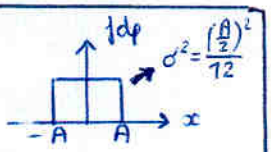
- Representación gráfica
  - plot (t, x, '*')
  - stem (n, x)

- Tiempo empleado
  - t = clock;
  - [.. y=sin(x)...]
  - tiempo = etime (clock, t)
    - ↑ elapsed time

- señales discretas en Matlab
  - Equiespaciados: n = 0:0.5:3 ; t = linspace (0, 3, 20)
  - Espaciados progresión geométrica: f = logspace (log10(1), log10(2), 13) ; f2 = 440 * f; ← tonos: ver p. 5
  - notas $u(t) \cdot e^{-t/\tau}$ ← p. 7

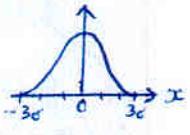
Generar ruido:

- Ruido blanco UNIFORME entre ±A



  - r1 = rand (1000, 1) ∈ [0, 1] (↑ fila, ↑ col)
  - r2 = (r1 - 0'5) * 2 * A;

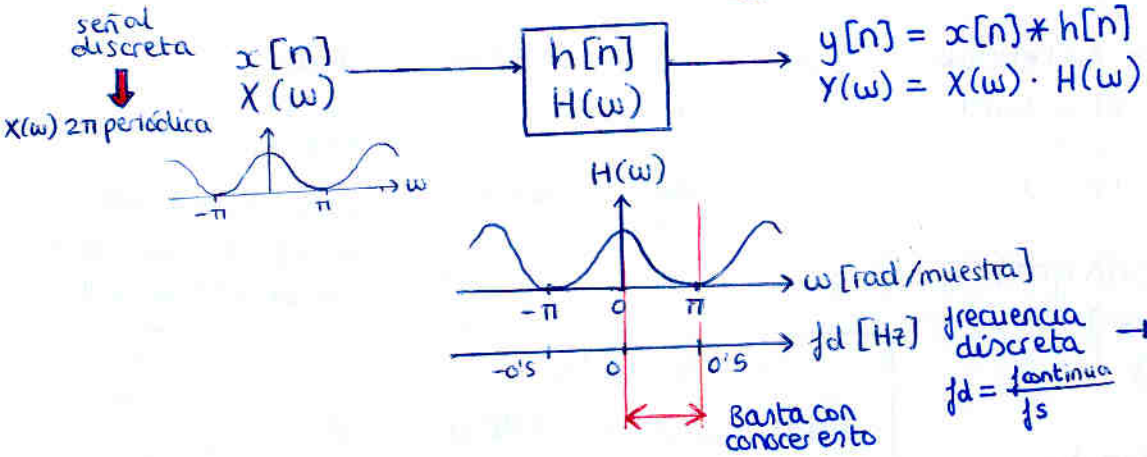
- Ruido blanco GAUSSIANO ↔ media m, var σ^2 ; $\sigma^2 = \overbrace{E[X^2]}^{\text{potencia}} + m^2$
  - desv_estandar = sqrt (var)
  - r1 = randn (1000, 1) (↑ fila, ↑ col) ↔ mean: 0, variance: 1



  - r2 = desv_estandar .* r1
  - r3 = r2 + media;
  - hist (r3)

PRÁCTICA 2. La transformada Z y sus aplicaciones

Función de transferencia $H(\omega)$ ≡ Respuesta en frecuencia del filtro



No confundir con la frecuencia NORMALIZADA que pide Matlab en algunas funciones y que va de 0 → 1

$f_{norm} = 2 * f_d = \frac{f_{cont}}{(f_s/2)}$

mar frec para que no haya aliasing

$H(\omega)$ se obtiene con freqz

$$[H, W] = freqz(B, A, N)$$

↑ nº puntos que devuelve

[0, π]

$$H(z = e^{j\omega}) = \frac{B(z)}{A(z)} = \frac{b(1) + b(2)z^{-1} + b(3)z^{-2} + \ldots + b(n_b+1)z^{-n_b}}{1 + a(2)z^{-1} + a(3)z^{-2} + \ldots + a(n_a+1)z^{-n_a}}$$

Módulo en dB: `plot(W/(2*pi), 20*log10(abs(H)));`

Fase en rad/muestra: `plot(W/(2*pi), angle(H));`

Si quisieramos generar el vector W manualmente, cuidado, debe ir de 0 a pi

N muestras ej N = 512

$W \not= 2\pi * \frac{1}{N} * (0 : N-1)$ ✗

$W = 2\pi * \frac{1}{2N} * (0 : N-1)$ ✓

y NO OLVIDAR $W = W(:)$

Nota: Polinomios y raíces en Matlab

$c = [c_1, c_2, c_3, c_4] \equiv c_1x^3 + c_2x^2 + c_3x + c_4$

↑ interpretación como polinomio

$r = roots(c)$ ← columna con las raíces de c ⟷ $c = poly(r)$ ← raíces

↑ coeficientes del polinomio

Polos y ceros (desarrollo teórico)

$$y[n] = b_0 x[n] + b_1 x[n-1] + \dots + b_{Nb} x[n-Nb] + a_1 y[n-1] + a_2 y[n-2] + \dots + a_{Na} y[n-Na]$$

$$\Downarrow$$

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \dots + b_{Nb} z^{-Nb}}{1 - a_1 z^{-1} - a_2 z^{-2} - \dots - a_{Na} z^{-Na}}$$

$$= b_0 \cdot \frac{(1 - c_1 z^{-1}) \cdot (1 - c_2 z^{-1}) \cdot \dots \cdot (1 - c_{Nb} z^{-1})}{(1 - d_1 z^{-1}) \cdot (1 - d_2 z^{-1}) \cdot \dots \cdot (1 - d_{Na} z^{-1})}$$

c_i : ceros
d_i : polos

(en Matlab)

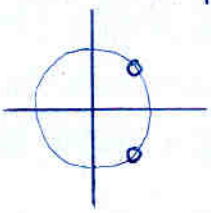
```
B = [b0, b1, b2, ..., bNb];
A = [1, -a1, -a2, ..., -aNa];
```

← sus elementos son los mismos (mismo signo) que aparecen en el denominador

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \dots}{1 - a_1 z^{-1} - a_2 z^{-2} - \dots}$$

a_1 → A[2], a_2 → A[3]

```
c = roots(B); % ceros
d = roots(A); % polos
zplane(c, d);
```



Respuesta impulsional h(n) ← impz

$h[n]$

no confundir con H(z) freqz

- mediante filtrado de $\delta[n]$

```
delta = [1 zeros(1,30)];
y = filter(B, A, delta);
stem(y);
```

- mediante impz

```
[h, n] = impz(B, A)
stem(n, h)
```

Filtrado de señales

```
y = filter(B, A, x);
```

NOTA: Si un filtro es FIR

$$H(\omega) = \frac{B(z)}{A(z)_{=1}} = B(z)$$

$$H(\omega) = b_0 + b_1 z^{-1} + b_2 z^{-2} + \dots$$

ENTONCES

$$h[n] = [b_0, b_1, b_2, \dots]$$

$b_0 \delta(n) + b_1 \delta(n-1) + \dots$

en vectores de Matlab

$B \equiv h$

Cuidado, H(w) no es B, es un vector de 512 ptos con $B(e^{j\omega})$

por ej H(0) = sum(B)

Efectos del filtrado

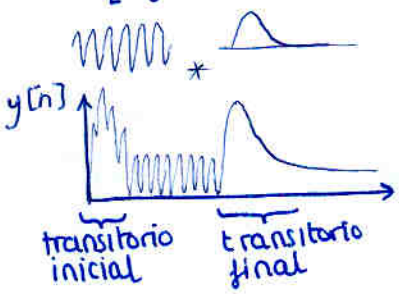
$x[n] \xrightarrow[X(\omega)]{} H(\omega) \xrightarrow[Y(\omega)]{} y[n]$

- En frecuencia:

$$Y(\omega) = X(\omega) \cdot H(\omega) \begin{cases} |Y(\omega)| = |X(\omega)| \cdot |H(\omega)| \\ \angle Y(\omega) = \angle X(\omega) + \angle H(\omega) \end{cases}$$

En tiempo

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



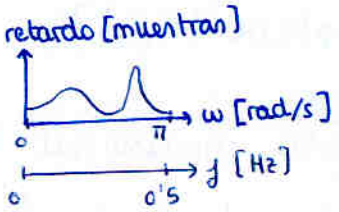
→ Por estar 'entrando' / 'saliendo' h[n] al desplazarse para hacer la convolución

Retardo de grupo:

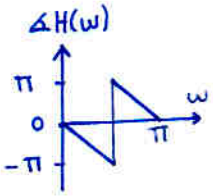
$$G_d(\omega) = -\frac{d(\angle H(\omega))}{d\omega} \quad [\text{muestras}]$$

nº de muestras que se retarda la pulsación ω al pasar por el filtro

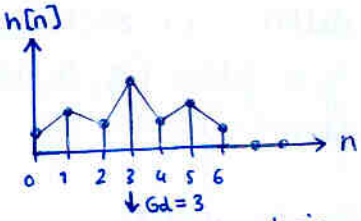
```
[Gd, W] = grpdelay(B, A)
plot(W/(2*pi), Gd)
```



Fase lineal



H(ω) fase lineal ⟺ Respuesta impulsiva simétrica h[n]



⇩

El retardo de grupo es constante de valor igual al índice de la muestra del centro de simetría

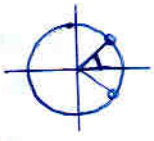
Diseño de Filtros Notch (colocando ceros y polos)

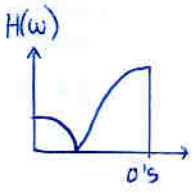
Filtros FIR

sólo ceros en $e^{\pm j\omega}$

frec discreta que queramos eliminar

```
arg = 2*pi*f_elim
ceros = [1*exp(j*arg); 1*exp(-j*arg)];
B = poly(ceros);
[H,W] = freqz(B,[1]);
plot(W/2*pi, abs(H));
```



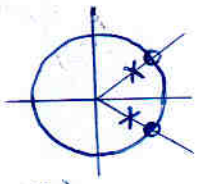


Filtros IIR

ceros en $e^{\pm j\omega} = e^{\pm j 2\pi f_{elim}}$

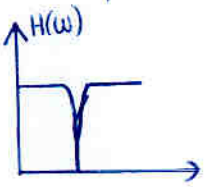
polos en $A \cdot e^{\pm j\omega}$ con $A<1$

cuanto mas $A \to 1$ más estrecho será el filtro notch, ya que cualquier $e^{j\omega}$ casi es equidistante al polo y al cero



$$H(z) = b_0 \cdot \frac{(1-c_1 z^{-1})(1-c_2 z^{-1})}{(1-d_1 z^{-1})(1-d_2 z^{-1})}$$

$\simeq 1$ $\simeq 1$



PRACTICA 3. La transformada rápida de Fourier

ver referencia rápida LabTDS pag II-3, 4

Diezmado en tiempo

$N = 2^{\alpha} \rightarrow$ $DFT_N = DFT_{\frac{N}{2}}$ muestras pares $+$ $DFT_{\frac{N}{2}}$ muestras impares $\cdot W_N^k$ ($e^{-j\frac{2\pi k}{N}}$)

ver mfft.m en cuarol3

Nota: fftdt() está en el directorio de la práctica se encarga de generar los fasores y llamar a mfft

$N = 3^{\alpha} \rightarrow$ $DFT_N = DFT_{\frac{N}{3}}^{3r} + DFT_{\frac{N}{3}}^{3r+1} \cdot W_N^k + DFT_{\frac{N}{3}}^{3r+2} W_{2N}^k$

ver mfft3.m

Diezmado en frecuencia (fftdf)

$N = 2^{\alpha} \rightarrow$ $DFT_N(k) = \begin{cases} DFT_{\frac{N}{2}} \text{ de } x[n] + x[n+\frac{N}{2}] & k \text{ par} \\ DFT_{\frac{N}{2}} \text{ de } (x[n] - x[n+\frac{N}{2}]) \cdot W_N^k & k \text{ impar} \end{cases}$

Recuerda FFT ≡ DFT calculada eficientemente

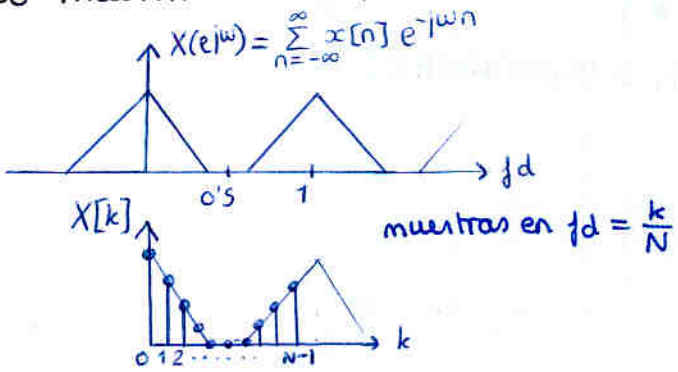
DFT La DFT resultar ser 'muestras' de X(ω)

$$X[k] = \sum_{n=0}^{N-1} x[n]\, e^{-j\frac{2\pi k}{N}n}$$

$$x[n] = \frac{1}{N}\sum_{k=0}^{N-1} X[k]\, e^{j\frac{2\pi k}{N}n}$$

nº muestras de x[n] = nº muestras de X[k]

otra interpretación es que X[k] es la X(ω) de x[n] repetida periodicamente (salen deltas)



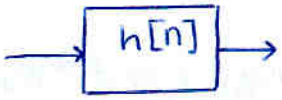
FFT implementada en Matlab → ya compilada

No es interpretada (los .m si lo son) y por tanto no tiene que compilarse cada vez que se llama a la función, por tanto es mucho más rápida

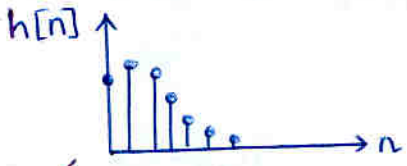
PRACTICA 4. Aplicaciones de la DFT

Obtención de H[ω] a partir de h[n]

- **Filtro FIR**

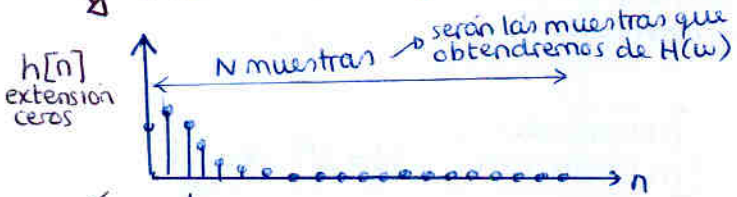


conocemos $h[n] \equiv B$ (FIR ↑, coef del numerador ↑)



extensión con ceros (para obtener un muestreo más fino de H(ω))

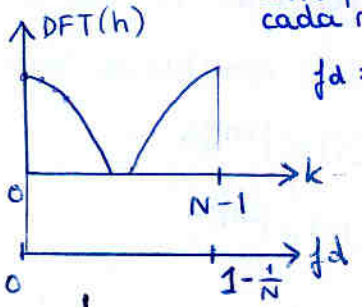
```
function [H,f]=respfrecfir(h,N)

N=2^(nextpow2(N));
% N=2^(ceil(log2(N)));

H=fft(h,N);

H=fftshift(H); % "Swapea" las dos mitades de la FFT

f=-0.5:1/N:0.5-(1/N);
```



DFT → DFT_N

La frec digital correspondiente a cada muestra será

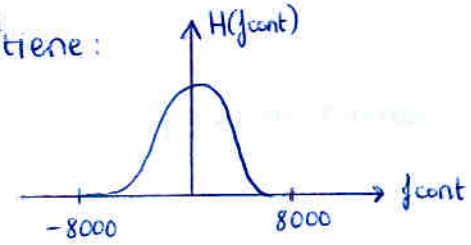
$f_d = \frac{k}{N}$ $k = 0, 1, \ldots, N-1$



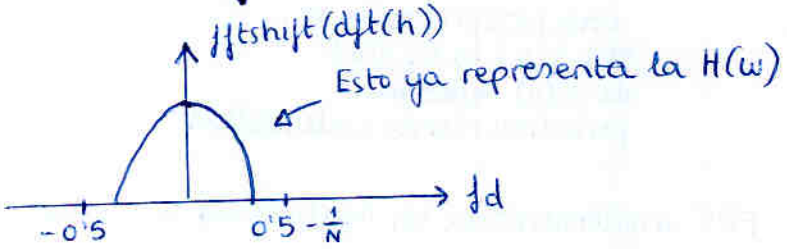
con la salida se puede hacer

```
fs = 16000;
fcont = fs * f
plot (fcont, 20log10(abs(H)))
```

se obtiene:



FFTSHIFT swapea el vector



- **Filtro IIR**

$$H(\omega) = \frac{B(z) \longleftrightarrow H_B(\omega)}{A(z) \longleftrightarrow H_A(\omega)}$$

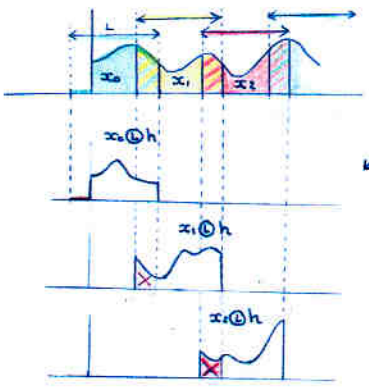
- Calculamos $H_B(\omega)$ y $H_A(\omega)$ = la respuesta en frec de los "filtros" FIR del numerador y denominador
- Dividimos muestra a muestra

$$H(\omega) = H_B(\omega) ./ H_A(\omega)$$

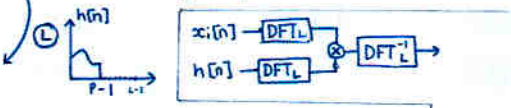
```
function [H,f]=respfreciir(B,A,N)

N=2^nextpow2(N);

BB=fft(B,N); %En un fir, los coeficientes son las muestras
% H(z) = A + Bz^-1 + Cz^-2 + ....
% h(n) = Ad(n) + Bd(n-1) + ...

AA=fft(A,N);

% Y curiosamente, la FFT de la división, coincide con la division
% muestra a muestra de los filtros FIR en numerador y denominador
H=BB./AA;

H=fftshift(H);    % Asi reordenamos las muestras de H
%f=(0:N-1)/N - 0.5;
f=-0.5:1/N:0.5-(1/N);
```

Filtrado lineal usando DFT's : Solape y almacenamiento

- Solape y almacenamiento



1. Trocear señal en bloques longitud L que se solapan P-1 muestras con el anterior (que el 1º bloque se solape con P-1 ceros del principio)



2. Hacer la convolución circular de cada trozo

$$x_i[n] \textcircled{L} h[n]$$

sin tener que insertar ceros

NOTA: h[n] ya la guardo con los ceros

sabemos que el resultado tiene duración L (igual que el trozo) y que las P-1 primeras muestras [0, P-2] son erróneas.

3. Simplemente deshechamos las primeras muestras erróneas y las sustituimos por las muestras correctas del trozo anterior

Se puede utilizar filtro para eliminar ruido

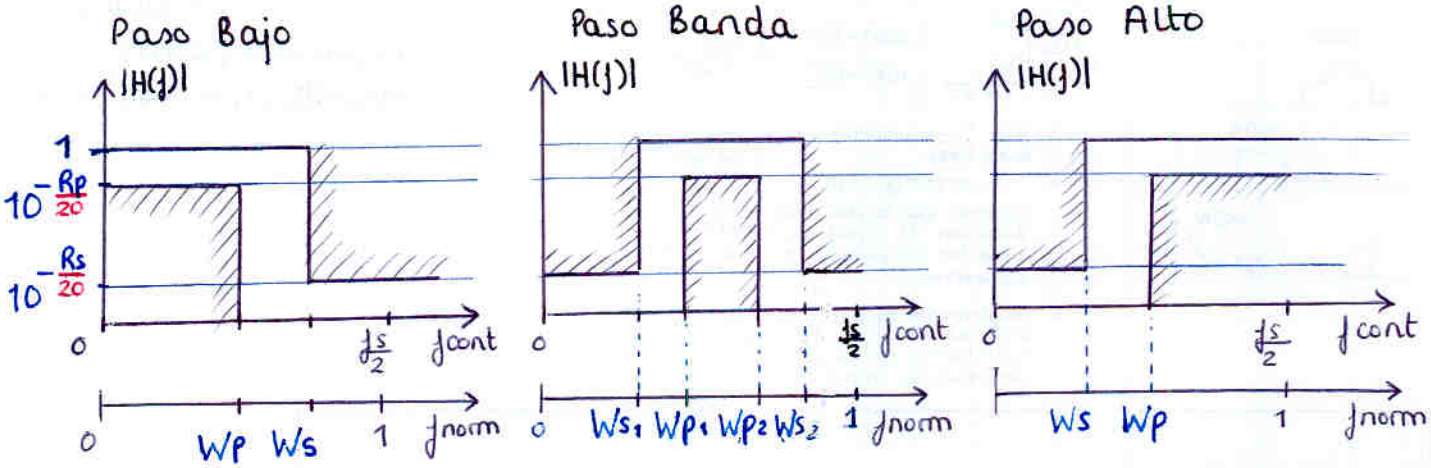
```
load h.mat
[x,fs,nbits]= wavread
                ('audio1.wav');
y=oversave(x,h);
wavwrite(y,fs,nbits,'result.wav');
```

```
function y=oversave(x,h)

% Poner los vectores de entrada en columnas
x=x(:);
h=h(:);

% Calcular longitudes L y P
L=max(512,2^nextpow2(length(h)))
P=length(h)
H=fft(h,L);

% Punteros iniciales :
puntero_inicio_x = 1 - (P-1);
puntero_fin_x = puntero_inicio_x + (L-1);
puntero_inicio_y = 1;
puntero_fin_y = puntero_inicio_y + (L-1) - (P-1);


while(puntero_inicio_x<=length(x)) % Hasta que el trozo de x no se haya
salido del todo

    % Preparamos el trozo de X
    num_ceros_iniciales = -(puntero_inicio_x-1);
    num_ceros_finales = puntero_fin_x-length(x);
    indice_inicio_x = max(1,puntero_inicio_x);
    indice_fin_x = min(length(x),puntero_fin_x);
    x_trozo = [zeros(num_ceros_iniciales,1);
               x(indice_inicio_x:indice_fin_x);
               zeros(num_ceros_finales,1)];

    % Hacemos la convolucion circular modulo L
    X_trozo = fft(x_trozo);
    Y_trozo = X_trozo.*H;
    y_trozo = ifft(Y_trozo);

    % Quitamos la parte de Y_trozo que no interesa
    % es decir, las (P-1) primeras muestras
    % que son el resultado erroneo de la
    % convolucion circular.
    y_trozo = y_trozo(P:end);

    % Insertamos el trozo de y en el lugar correspondiente de y
    y(puntero_inicio_y:puntero_fin_y,1) = y_trozo;

    % Actualizo los punteros para el trozo siguiente
    puntero_inicio_x = (puntero_fin_x+1)-(P-1);
    puntero_fin_x = puntero_inicio_x + (L-1);
    puntero_inicio_y = (puntero_fin_y+1);
    puntero_fin_y = puntero_inicio_y + (L-1) - (P-1);

end
```

PRACTICA 5. Diseño de Filtros Digitales con Matlab

Diseño de Filtros IIR (Digitales) usamos funciones de Matlab

1. Especificaciones



- Las funciones que vamos a estudiar necesitan los siguientes datos
  - R_p : passband ripple decibels (máxima atenuación en banda de paso)
  - R_s : stopband ripple decibels (mínima atenuación en banda eliminado)
  - Dos vectores Wp y Ws
    - Éstos vectores le dirán a la función si es paso bajo, paso banda (o paso alto añadiendo 'high') según el tamaño del vector
    - A las funciones se les pasa primero Ws (1º parámetro) y luego Wp (2º parámetro). Cuidado con ésto porque lo cambiaron en una actualización y se les olvidó cambiar la ayuda (por tanto hay ayudas incorrectas en Matlab)
    - Las unidades de éstos vectores W (a pesar de llamarse W) son de frecuencia normalizada. La frecuencia normalizada se la define Matlab como un valor entre 0 y 1, correspondiente a 0 y $f_s/2$. Cuidado, no es igual a la frecuencia discreta de la asignatura TDS que va de 0 a 0'5, es justo el doble.

$$\text{frec normalizada de Matlab } W = \frac{f_{continua}}{(f_s/2)} = 2 \cdot \frac{f_{continua}}{f_s \text{ frec muestreo}} = 2 \cdot f_{discreta}$$

Entonces, habiendo ya 'calculado' (Wp y Ws) ó (Wp1, Wp2, Ws1 y Ws2) construimos los vectores

Vector	Paso Bajo	Paso Banda	Paso Alto
Wp =	[Wp]	[Wp1, Wp2]	[Wp]
Ws =	[Ws]	[Ws1, Ws2]	[Ws]

2. Cálculo del orden

Con los parámetros anteriores hacemos: (en algunas versiones de Matlab es al revés, y en otras el help está mal (en versión 7 ES AL REVES))

```
[N, Wn] = tiporespuestord ( Ws, Wp, Rp, Rs )
```

- N → orden del filtro
- Wn → frec normalizada de resonancia (vector 1x2 si Ws y Wp eran tb 1x2)

} necesario para cumplir las especificaciones

3. Cálculo de los coeficientes

```
[B, A] = tiporespuesta ( N, Rp, Rs, Wn, { nada | 'low' | 'pass' | 'high' | 'stop' } )
```

- Rp, Rs: algunos tipos de respuesta no necesitan uno o dos de estos parámetros
- último parámetro: permite distinguir el tipo de filtro que queremos

		último parámetro
si Wn es vector 1x1	Paso bajo	nada o 'low'
	Paso alto	'high'
si Wn es vector 1x2	Paso banda	nada o 'bandpass'
	Elim. banda	'stop'

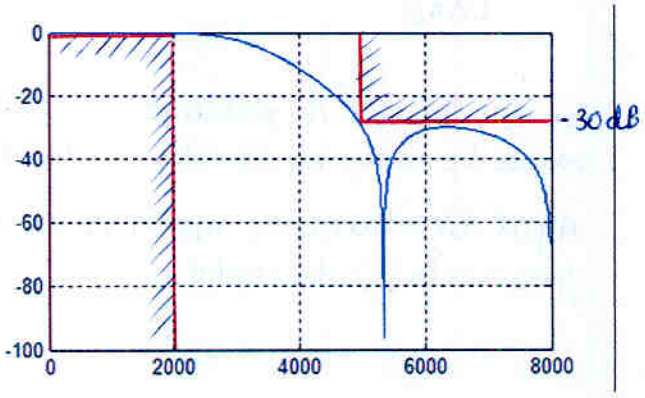
Las funciones para calcular el orden y los coef tienen distintos nombres para los distintos tipos de respuesta → ver tabla

Tipo Respuesta	Cálculo del Orden [N, Wn] =	Cálculo de los coeficientes [B, A] =
Butterworth	buttord (Ws, Wp, Rp, Rs)	butter (N, Wn, ..)
Chebyshev tipo I	cheb1ord (Ws, Wp, Rp, Rs)	cheby1 (N, Rp, Wn, ..)
Chebyshev tipo II	cheb2ord (Ws, Wp, Rp, Rs)	cheby2 (N, Rs, Wn, ..)
Eliptico	ellipord (Ws, Wp, Rp, Rs)	ellip (N, Rp, Rs, Wn, ..)

(en versiones nuevas es al revés)

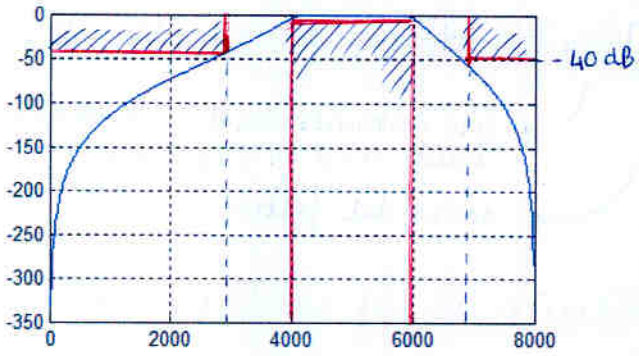
ejemplo: filtro paso bajo

```
>> fs = 16000; % frec muestreo
>> fp = 2000; % fin de la banda de paso
>> fstop = 5000; % inicio de banda atenuada
>> Wp = fp/(fs/2);
>> Ws = fstop/(fs/2);
>> Rp = 1; % 1dB
>> Rs = 30; % 30dB
>> [N,Wn] = cheb2ord(Wp,Ws,Rp,Rs);
>> [B,A] = cheby2(N,Rs,Wn,'low');
>> [H,W] = freqz(B,A);
>> f_discreta = W/(2*pi);
>> f_continua = f_discreta * fs;
>> plot(f_continua,20*log10(abs(H)))
>> grid
```



ejemplo: paso banda

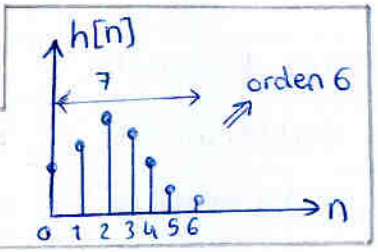
```
>> % Pasobanda con fs/2 = 8000 Hz
>> fs = 16000; % frec muestreo
>> % Banda de paso [4kHz,6kHz]
>> fp1 = 4000;
>> fp2 = 6000;
>> % Banda atenuada
>> fstop1 = 3000;
>> fstop2 = 7000;
>> fmax = fs/2; %Frec a la que normalizamos
>> Wp = [fp1/fmax, fp2/fmax];
>> Ws = [fstop1/fmax, fstop2/fmax];
>> Rp = 5; % 5dB
>> Rs = 40; % 40dB
>>
>> [N,Wn]=buttord(Wp,Ws,Rp,Rs);
>> [B,A]=butter(N,Wn,'bandpass');
>> [H,W] = freqz(B,A);
>> f_discreta = W/(2*pi);
>> f_continua = f_discreta * fs;
>> plot(f_continua,20*log10(abs(H)))
Warning: Log of zero.
>> grid
```



Filtros FIR digitales. Método de las ventanas

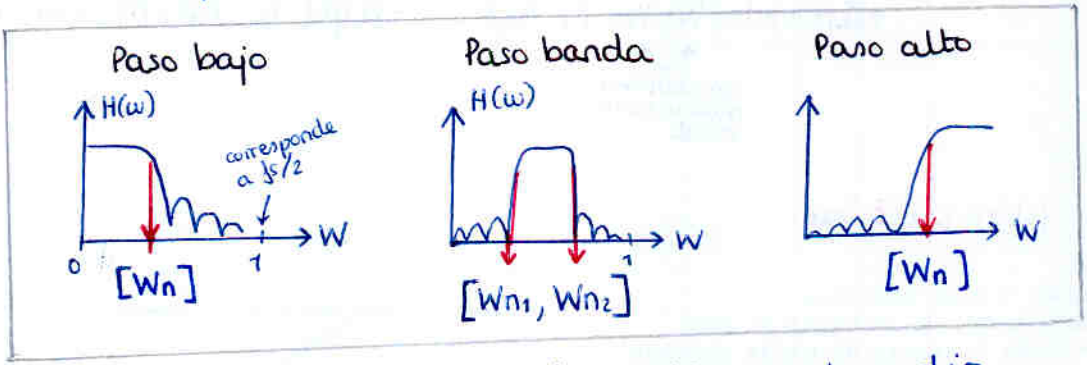
- Mucha resolución (bajo ALP) (reducir banda de paso) → Interesa número de coeficientes elevado
  (ALP: ancho lóbulo principal)
- Margen dinámico grande (bajo Rs) (baja atenuación mínima en banda eliminada) → Aumentar el nº de coeficientes no sirve de nada; hay que jugar con la ventana

1. Parámetros:
   - Orden del filtro: $N = num\_coef - 1 = L - 1$
     ↑ recuerda: en FIR h[n] es igual a los coeficientes B



   - Frecuencias Wn
     ↑ frec normalizada de Matlab entre 0 y 1

$$Wn = \frac{f_{continua}}{(f_s/2)} = 2 \cdot f_{discreta}$$



En el filtro FIR no podemos ser tan pejigueros de pedir banda de paso, banda atenuada, atenuación mínima, ...

Aquí directamente metemos en el vector Wn la(s) frecuencia(s) de corte.

2. Obtención de los coeficientes $B \equiv h[n]$

```
[B] = fir1 ( N, Wn, 'low' / 'bandpass' / 'high' , ventana )
```

```
h = B;
stem(h);
[H, W] = freqz (B, 1);
plot ( W/2*pi, 20 log10(abs(H)) );
```

- ventana: `ones(N+1, 1)` → ventana rectangular
- ventana: `hamming(N);` (por defecto si no pones nada)

Comparando ventanas:

- **rectangular**:
  - menor banda de transición (cae más rápido) lo cual no tiene mucho mérito porque la banda de transición es menor cuantos más coeficientes usemos
- **hamming**:
  - menor rizado
  - menor R_s (atenuación en banda eliminada)

Filtro FIR mediante método de optimización

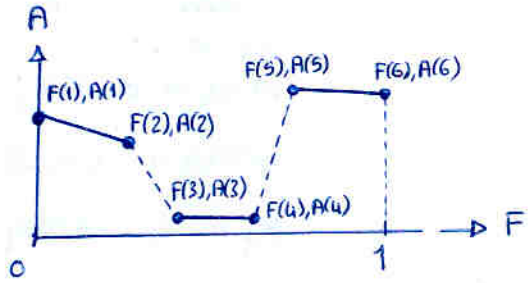
Necesitamos 2 vectores del mismo tamaño

F = vector con los límites de las bandas de frecuencia en orden ascendente y en frecuencia normalizada (de 0 a 1)

A = vector con la amplitud deseada en cada frecuencia de F

La respuesta deseada está formada por las líneas que unen los puntos $\{F(k), A(k)\}$ con $\{F(k+1), A(k+1)\}$ para k impar; para k par se considera banda de transición y no se intenta aproximar esa línea

La respuesta minimiza el error en cada banda con los coef de que dispone

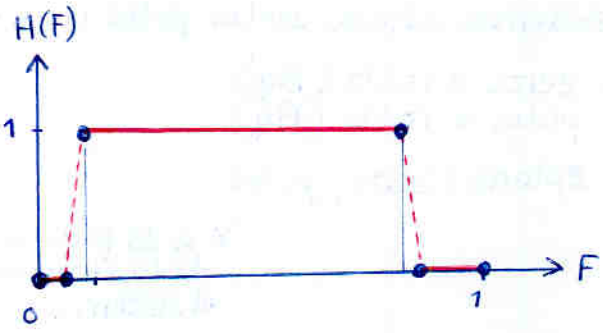



```
B = remez (N, F, A)
```

- B: tendrá long. N+1
- N: orden = nºcoef - 1

ejemplo:

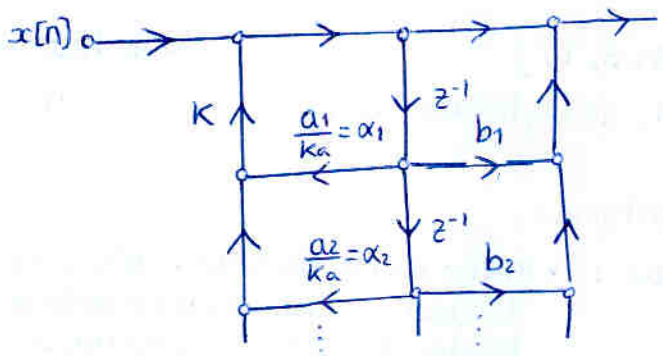
```
F = [0  0'05  0'075  0'85  0'875  1]
A = [0   0     1     1     0     0]
```



PRÁCTICA 6. Efectos de precisión finita en filtros digitales

- Tenemos un filtro: $H(z) = \frac{B(z)}{A(z)}$

  → si $\max(A) > 1$, no se pueden almacenar los coef en coma fija, y lo que se almacena es $\alpha = A./K_a$



siendo
$K_a = \text{ceil}(\max(A))$;

  → si $\max(B) > 1$, ocurre lo mismo y lo que se almacena entonces es $\beta = B./K_b$

- Al almacenar α y β, éstos se cuantificarán (precisión finita) dando lugar a α_q y β_q.
  El filtro que entonces obtenemos tiene unos coeficientes $A_q = \alpha_q \cdot K_a$, $B_q = \beta_q \cdot K_b$
  y tendremos $H_q(z) = \frac{B_q(z)}{A_q(z)}$

  Éste nuevo filtro puede haberse convertido en inestable

Cómo cuantificar coeficientes en Matlab tenemos A y B

Cuantificar coeficientes denominador	numerador
max(A) ans = 73'8659 Ka = 74; % Ka = ceil(max(A)); alfa = A/Ka; **alfaq = cuantif(alfa, 16)** (16 ↑ nº bits) Aq = Ka * alfaq	max(B) ans = 0'032 Kb = 1; beta = B/Kb; betaq = cuantif(beta, 16) Bq = Kb · betaq

Podemos ahora ver los polos y ceros del nuevo filtro

```
zerosq = roots(Bq)
polosq = roots(Aq)
zplane(zerosq, polosq);
```

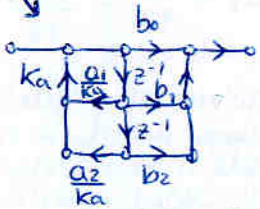
↳ si los polos se salen de la circunferencia unidad, el sistema es inestable

Comparando con el original
zeros = roots(B)
polos = roots(A)
zplane(zeros, polos)

Descomposición en secciones de segundo orden

Para solucionar el problema anterior

cada uno tendrá sus coeficientes que habrá que cuantificar



Cada filtro de orden 2 tiene 5 coeficientes

Hay funciones en Matlab que ayudan a descomponer en SOS (second order sections)

```
[z, P, K] = tf2zp(B, A);   ← transfer function to zeros, poles
SOS = zp2sos(z, P, K);     ← zeros poles to 2nd order section
```

sos = matriz 6x6 (1 fila para los coef de cada filtro)

```
% Para cuantificarlo, en la práctica se dice que se considere el mismo
  factor k para todas las secciones de orden 2. Obviamente en la
  práctica cuantificaríamos por separado numerador y denominador
  de cada filtro
```

```
K = ceil(max(sos));
sigma = sos./K;
sigmaq = cuantif(sigma, 16);
sosq = sigmaq * K;
```

← cuantificamos los filtros de segundo orden.

```
[Bq, Aq] = sos2tf(sosq);   ← second order section to transfer function
```

Ruido cuantificación — Recordemos la teoría

• Redondeo en operaciones

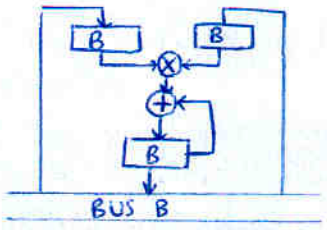
Al multiplicar 2 números se aumenta la precisión, y al final esta debe perderse, introduciendo un error de

$$Pote = \frac{\Delta^2}{12} = \frac{\left(\frac{2}{2^B}\right)^2}{12} = \frac{2^{-2B}}{3}$$

Movimiento fuentes de ruido:
El lugar dónde se suma un ruido puede ir moviéndose por el camino de la 'señal' mientras el camino sea único. (única salida)

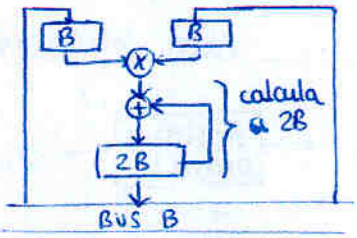
Agrupación fuentes de ruido:
Dos fuentes de ruido sumando en mismo punto pueden agruparse en potencia.

Acumulador Simple



Hay ruido tras cada multiplicación por fraccionario.

Acumulador Doble

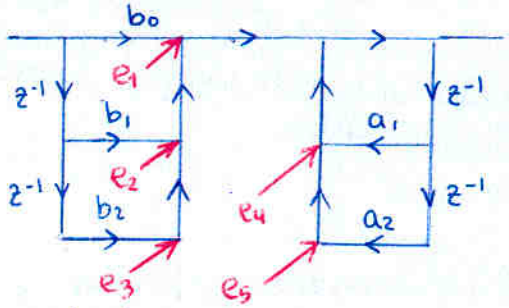


Sólo hay ruido al sacar datos al bus
- resultado final
- acumulador debe remultiplicarse
- acumulador debe almacenarse (z^{-1})

no hay ruido en sucesivas operaciones MACD

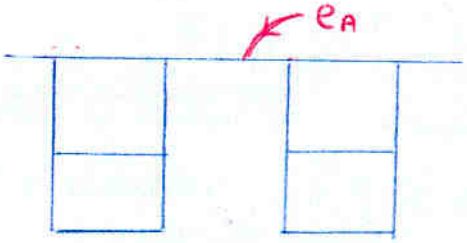
Forma Directa I

Acumulador simple

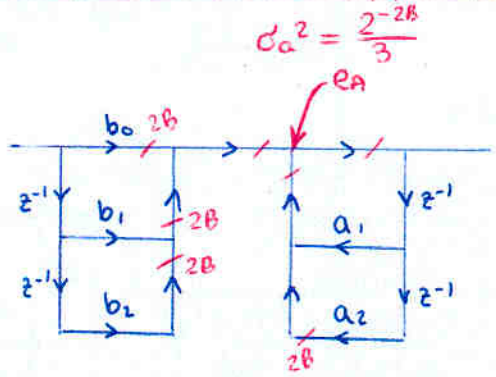


Movimiento y agrupación ⇩

$$\sigma_A^2 = 5\,\frac{2^{-2B}}{3}$$

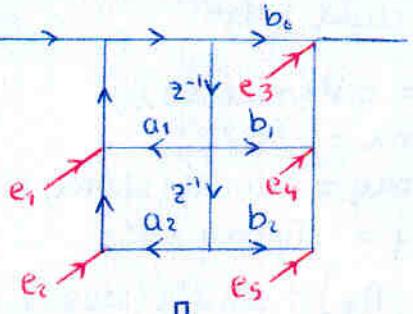


Acumulador Doble



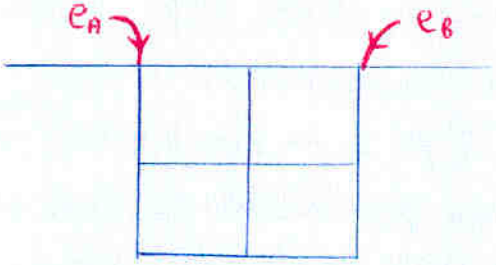
Forma Directa II

Acumulador simple



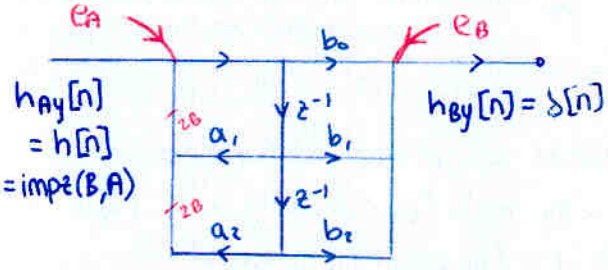
⇩

$$\sigma_a^2 = 2\cdot\frac{2^{-2B}}{3} \qquad \sigma_b^2 = 3\,\frac{2^{-2B}}{3}$$



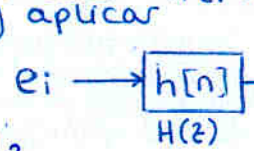
Acumulador Doble

$$\sigma_a^2 = \frac{2^{-2B}}{3} \qquad \sigma_b^2 = \frac{2^{-2B}}{3}$$



Solo falta por calcular $He_i \to Y$ y aplicar

$$H_{eY}(z) = \frac{b_0 + b_1 z^{-1} + \dots}{1 - a_1 z^{-1} - \dots}$$



$$\sigma_y^2 = \sigma_x^2 \cdot \sum |h[n]|^2$$

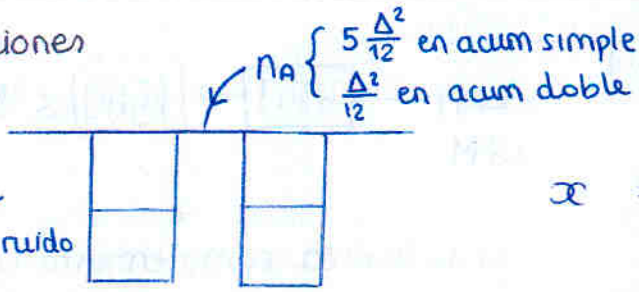
en Matlab:

```
h = impz([b0, b1, ..], [1, -a1, -a2, ...]);
S = sum(h.*h);
Nout = Nin * S
```

Hacerlo para cada fuente y sumar

Observaciones

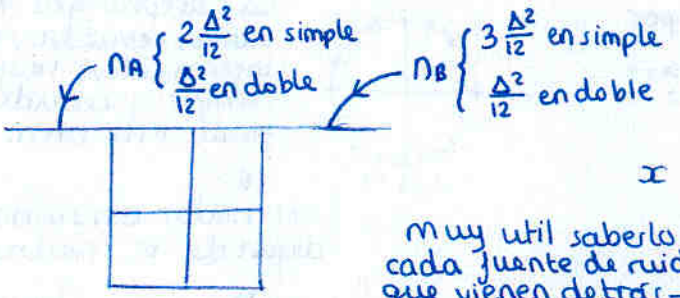
FD I:



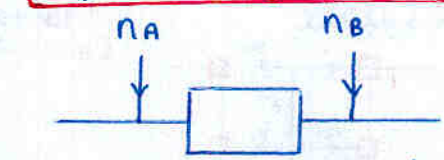
se reduce a única fuente de ruido

FD II:

se reduce a dos fuentes de ruido



h_{Ay} = impz(B,A) | h_{By} = [1];



muy util saberlo para realización cascada: cada fuente de ruido atraviesa sólo los filtros que vienen detrás → interesa ganancias grandes al principio

Cálculo en Matlab [DSP con acumulador de ancho doble
Ruido cuantificación [Forma directa 2

Dos fuentes de ruido, una al principio y otra al final

Estudio analítico

```
Bits = 8;
ruido = 2^(-2*Bits))/3;

hxy = impz(B,A);
Ssquaredxy = sum(hxy.* hxy);
Nout1 = ruido * Ssquaredxy;

hyy = 1;
Ssquaredyy = sum(hyy.* hyy);
Nout2 = ruido * Ssquaredyy;

Nout = Nout1 + Nout2;
```

Simulacion

Suponiendo que ya tenemos funciones para filtrado [con → filtraq / sin → filtra] cuantificación (ver en cuasol)

señal entrada: señal aleatoria 500 muestras con f.d.p uniforme, blanca, de media nula, y valor máximo 0'3Amax, siendo Amax la amplitud máxima para que no haya saturación

```
signal = 0'3 * Amax * (2 * rand(1,500)
y ideal = filter(B,A,signal);
y quantif = filtraq(B,A,signal,Bits)
error = yquantif - yideal
potinst = error.* error;
potmediaerror = mean(error);
```

¿sería igual a filter(Bq,Aq,signal)?

NO. La cuantificación en las operaciones no tiene NADA que ver con la cuantificación de coeficientes

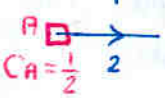
• **Saturación**: Recordemos la teoría

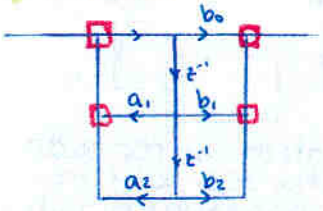
- No podemos codificar números > 1
- Habrá que regular la entrada para que en los nodos críticos no se supere un valor máximo

$x[n] \leq M$ → $h[n]$ → $|y[n]| \leq M \cdot \Sigma|h[n]|$

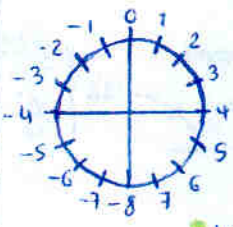
¿Nodos críticos?

- resultado de sumas va a hacerse productos por enteros > 1
- salida





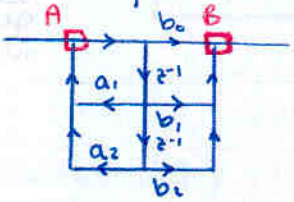
Si se utiliza complemento a 2



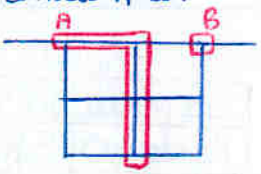
Ca2 acepta que, haciéndose sumas seguidas, en los pasos intermedios haya desbordam. siempre y cuando el resultado final esté bien.

⇓

Los nodos con sumas parciales dejan de ser nodos críticos



Nota: en realidad el nodo A es:



A cada nodo crítico se le asigna una cota máxima

C_i { normalmente 1; si cualquiera de las salidas del nodo se va a multiplicar por entero E, entonces $C_i = \frac{1}{E}$

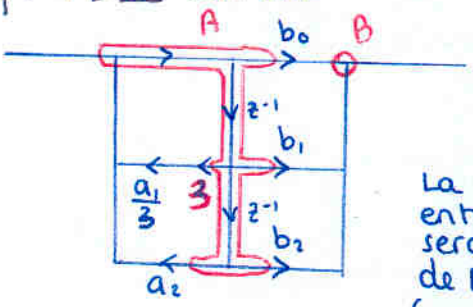
Para cada nodo crítico: Calculamos { $H_i(z)$ desde entrada hasta el nodo crítico, y con $H_i(z)$ obtenemos $h_i[n]$ y de ahí $\Sigma|h_i[n]|$

exigimos: $\Sigma|h[n]| \cdot M_i \leq C_i$

→ $M_i \leq \frac{C_i}{\Sigma|h_i[n]|}$

se hace para cada nodo crítico, y se toma como cota máxima de la entrada la menor de las M_i

ej: FD II con Ca2



$\begin{cases} C_A = \frac{1}{3} \\ C_B = 1 \end{cases}$

La cota a la entrada X será la menor de M_A y M_B (suele limitar M_A)

- Para A:

$$H_{XA}(z) = \frac{1}{1 - a_1 z^{-1} - a_2 z^{-2}}$$

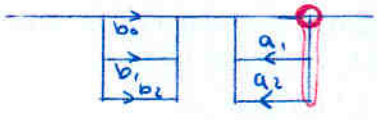
```
h = impz([1],[1,-a1,-a2])
S = sum(abs(h));
MA = CA / S
```

- Para B:

$$H_{XB}(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 - a_1 z^{-1} - a_2 z^{-2}}$$

```
h = impz([b0,b1,b2],[1,-a1,-a2]);
S = sum(abs(h));
MB = CB / S
```

NOTA: En FDI con Ca2 son todos sumas parciales y sólo hay un nodo crítico.



Con esto aseguramos que en el PEOR CASO (entrada igual a h[n]) no habrá saturación. Garantizado 100%

Condición más relajada: Exigir que no haya saturación para un tono a la entrada.

tono A_1 → $H(e^{j\omega})$ → tono $A_2 = A_1 \cdot |H(e^{j\omega})|$

Hay que suponer el peor tono posible; el que esté a la ω donde $|H(e^{j\omega})|$ es max.

$$M_i \leq \frac{C_i}{\max|H(e^{j\omega})|}$$

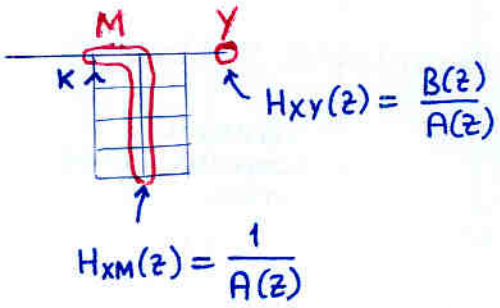
ej: $H_{XA} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 - a_1 z^{-1} - a_2 z^{-2}}$

```
HXA = freqz([b0,b1,b2],[1,-a1,-a2], 1024)
Hmax = max(abs(HXA))
M = C / Hmax
```

a mayor resolución, más cerca de pillar el máximo.

Estudio analítico Saturación

Nodos críticos: M e Y




```
K = ceil(max(A));      } nodo M
Mmax = 1/K;
Ymax = 1;              } nodo Y
```

- **Señal entrada arbitraria**

```
hxm = impz(1, A);
sumxm = sum(abs(hxm));
INmaxm = Mmax / sumxm;          → max para nodo M

hxy = impz(B, A);
sumxy = sum(abs(hxy));
INmaxy = Ymax / sumxy;          → max para nodo Y

INmax = min(INmaxm, INmaxy);
```

si utilizamos este criterio, aseguramos el peor caso (entrada = respuesta impulsiva)
Para un ruido típico que cumpla este INmax, cumplimos la saturación DE SOBRA
ej: en la práctica en la simulación con ruido a la entrada, en el nodo crítico ni llegaba a 0's

- **Tono a la entrada**

```
Hxm = freqz(1, A);
Hxmmax = max(abs(Hxm));
INtonemaxm = Mmax / Hxmmax;

Hxy = freqz(B, A);
Hxymax = max(abs(Hxy));
INtonemaxy = Ymax / Hxymax;

INtonemax = min(INtonemaxm, INtonemaxy);
```

Simulacion

ruido uniforme, media nula, entre ±1

```
signal limitada = INmax * 0'95 * (2*rand(1,500) - 1)
signal sin limitar =  1   * 0'95 * (2*rand(1,500) - 1)
```

⚠ cuidado con ese 2

```
yideallimitada = filter(B, A, signallimitada);
yideal sinlimitar = filter(B, A, signalsin limitar);
yquantiflimitada = filterq(B, A, signallimitada, Bits);
yquantif sinlimitar = filterq(B, A, signalsinlimitar, Bits);
```

ⓘ Podemos ver la señal en M usando filter(1, A, ...) filterq(1, A, ...)

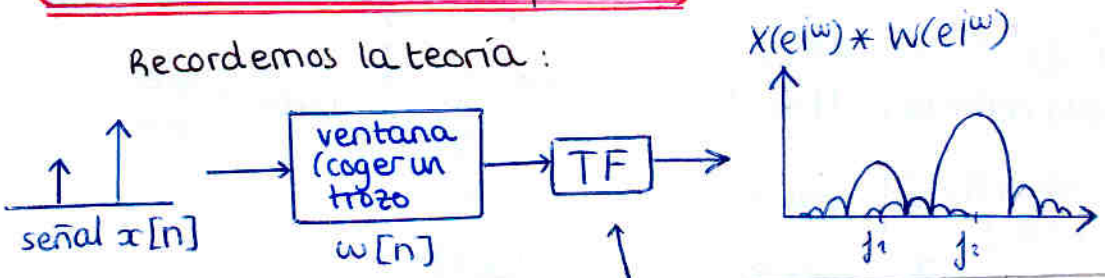
se puede comprobar que

yquantifsinlimitar ≠ yideal sin limitar a causa de la saturacion
sin embargo
yquantiflimitada = yideallimitada

buena forma de ver si $y_1 = y_2$: stem(y_1, y_2)

PRACTICA 7. Análisis espectral

Recordemos la teoría:



Conceptos importantes:

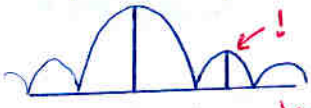
- Resolución ≃ A_{LP}(Hz)

  mínima separación entre tonos de igual amplitud para distinguirlos

  solución → aumentar L

- Margen dinámico ≃ N_{LPS}(dB)

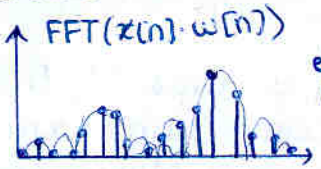
  Relación de amplitudes a partir de la cual se pueden distinguir tonos más separados que la resolución



  aumentar L no la mejora

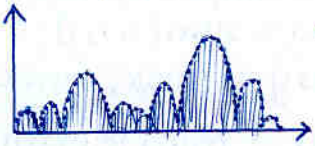
  solución: usar otra ventana

En la práctica hacemos la FFT y lo que tendremos serán muestras:



el muestreo puede ser muy malo y engañoso

conviene rellenar con ceros después de ventana, antes de aplicar la FFT para muestrear más fino



CUIDADO: NO MEJORAMOS la resolución

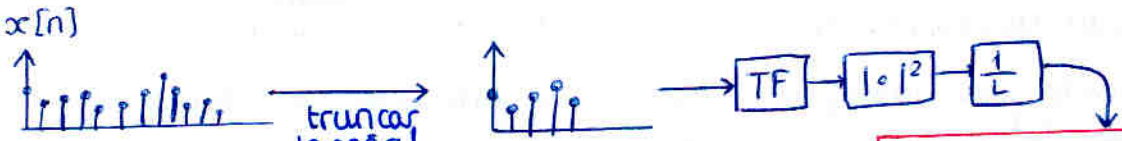
Ventanas w[n]

· Rectangular		$A_{LP} = \frac{2}{L}$ Hz	$N_{LS} = 13$dB
· Triangular		$A_{LP} = \frac{4}{L}$ Hz	$N_{LS} = 26$dB
· Hamming		$A_{LP} = \frac{4}{L}$ Hz	$N_{LS} = 42$ dB

Por tanto, primero escogemos la ventana que cumpla nuestro margen dinámico y luego hallamos el L necesario (mayor L → mayor tiempo necesario

$$T = L/f_s \ [s]$$

Periodograma $\hat{\phi}_x(e^{j\omega})$: se usa para estimar la d.e.p $\equiv \phi_x(e^{j\omega})$



$$\hat{\phi}_x(e^{j\omega}) = \frac{1}{L}\left|\sum_{n=0}^{L-1} x[n]e^{-j\omega n}\right|^2$$

```
function [P ,f]=periodograma(x, N)
% [P f]=periodograma(x,N)
% Calcula N muestras del periodograma de
% las muestras en x
% N debe ser > que length(x)
%
% P: Periodograma propiamente
% f frecuencias correspondientes

x = x(:);
L=length(x);
f = (0 : N-1)/N;
f = f(:)
%x = [x; zeros(N-L,1)];
%P=(1/L)*(abs(fft(x)).^2);

% La FFT ya se encarga de añadir
% ceros o truncar la señal convenietemente

P = (1/L)*(abs(fft(x,N)).^2);
```

$$E\left\{\hat{\phi}_x(e^{j\omega})\right\} = \phi_x(e^{j\omega}) * \frac{1}{L}\left|W_R(e^{j\omega})\right|^2$$

en general:
- el periodograma es sesgado
- sesgo disminuye si L aumenta

para ruido blanco, $\phi_x(e^{j\omega}) = cte$
- el periodograma es insesgado ya que $\frac{1}{L}|W_R(e^{j\omega})|^2$ tiene area 1

→ salida: f: frec discreta de toda la vida con 0's equivalente a $f_s/2$ sale en el rango [0, 1] i.e. copia innecesaria de semiperiodo negativo

P: periodograma (estimación de la d.e.p.)

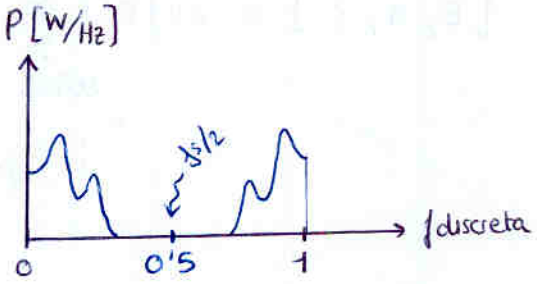
Son [W/Hz] por tanto hacer 10 log10(abs(P))

Periodograma modificado

[P, f] = periodmodif (x, ventana, N)

ventana: ones(L,1) | hamming(L) | recomend. N>2L

el L calculado para la resolución deseada



Para leer valores usar

[F, Y] = ginput(2) ← para tomar 2 muestras

$f_{continua} = f_{discreta} \cdot f_s$

ejemplo: separación mínima 80 Hz

separación $f_d = 80/f_s$ ⇒ Lmin

hamming: $A_{LP} = \frac{4}{L} < 80/f_s$

rectangular: $A_{LP} = \frac{2}{L} < 80/f_s$

recuerda que una L demasiado grande limita la mínima duración del tono que queramos detectar $T_{min} = L/f_s$

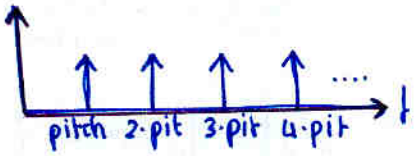
→ Detección de códigos DTMF
- se codifican los números con parejas de tonos
- La ALP será la mínima separación entre tonos que PUEDAN existir simultaneamente (i.e. que formen pareja
- ver cuasol tema 7, todo bien explicado

Juego de los espias
- se juega con resolución y margen dinámico
- recuerda que si no importa el MD la mejor ventana es la rectangular → mucha resolucion con mínima L

Análisis espectral de señales de voz

pitch: frecuencia fundamental de los sonidos sonoros — mujer 150–200 Hz, hombre 80–100 Hz

- como los sonidos son cuasi-periódicos, están formados por deltas a multiplos del pitch o frec fundamental



Las variaciones del pitch dentro de una frase determinan la entonación de ésta

↔ resolución mínima en caso peor 80 Hz ⇒

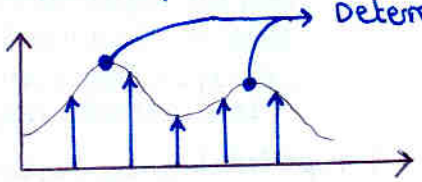
rectangular: $\text{resolucion} = \frac{2}{L} < \frac{80}{f_s}$

↳ $L_{min} = \text{ceil}(2 * f_s / 80)$

hamming: $\text{resolucion} = \frac{4}{L} < \frac{80}{f_s}$

↳ $L_{min} = \text{ceil}(4 * f_s / 80)$

formantes: los picos de la envolvente espectral

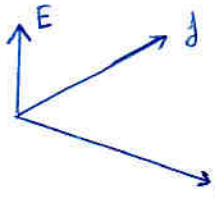


Determinan la letra (ej: 'a' o 'e')

Transformada de Fourier de tiempo corto

$[E, n, f] = \text{stft}(x, \text{ventana}, N, \text{avance})$

- x: señal
- ventana: ej: hamming(Lmin)
- N: nº puntos de la FFT (ej: >2L)
- avance: ej: ceil(2 * Lmin / 5)



Para visualizar:

```
[N, F] = meshgrid(n, f)
pcolor(n, f, abs(E));
shading interp
```

recuerda:

```
[y, fs, nbits] = wavread(file);
wavwrite(y, fs, nbits, file);
```

Estima de la autocorrelación usando el periodograma

Periodograma (ventana rectangular) $\xrightarrow{DFT^{-1}}$ Estimador sesgado de la autocorrelación

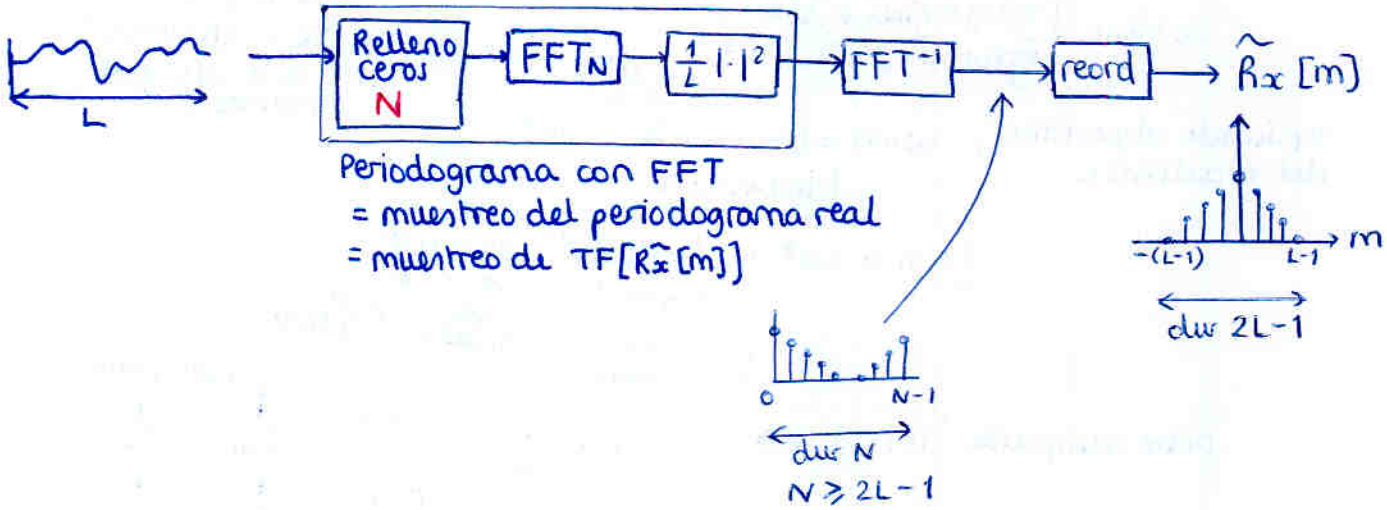
↑ ventana triangular

Autocorrelación

```
function R = correlperiod(x)
% Cuidado, el estimador sesgado de
  la autocorrelación (que es igual
  a la autocorrelación pasada por una ventana triangular) no
  hay que calcularlo con un periodograma modificado con
  ventana triangular, sino con el periodograma normal de
  ventana rectangular

L = length(x);
N = 2*L-1; % Muestras necesarias para el periodograma para
             insertar ceros

[P, f] = periodograma [f, N]

R = ifft(P, N)

R = [R(floor(N/2)+2 : end);     Para pasar la
     R(1 : floor(N/2)+1 ];      mitad derecha
                                a la izquierda
```

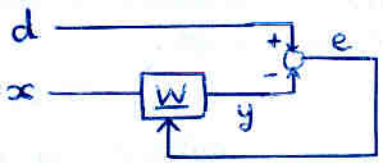
la salida es muy similar (casi idéntica) a `xcorr(x, 'biased')` de Matlab.

Recuerda:



PRACTICA 8. Filtrado adaptativo usando LMS

Recuerda:



El filtrado es:

$$\underline{X}[n] \rightarrow \boxed{\underline{W}[n] \text{ FIR Ncoef}} \rightarrow y[n] = \underline{X}^T \cdot \underline{W}$$

$$\underline{X}[n] = \begin{pmatrix} x[n] \\ x[n-1] \\ x[n-2] \\ x[n-3] \end{pmatrix} \qquad \underline{W} = \begin{pmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \end{pmatrix}$$

recuerda, FIR
Orden = Ncoef − 1

De las señales de entrada definimos

$$\underline{\underline{R}} = \begin{pmatrix} Rx[0] & Rx[1] & Rx[2] & Rx[3] \\ Rx[1] & Rx[0] & Rx[1] & Rx[2] \\ Rx[2] & Rx[1] & Rx[0] & Rx[1] \\ Rx[3] & Rx[2] & Rx[1] & Rx[0] \end{pmatrix}$$

cada elemento $Rx[m] = E\{x[n] \cdot x[n-m]\}$

en matlab: $R = \text{matcor}(x, N)$ en este caso $N = 4$

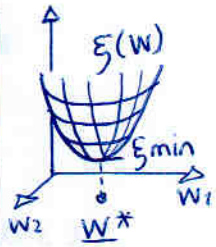
$$\underline{P} = \begin{pmatrix} Rdx[0] \\ Rdx[1] \\ Rdx[2] \\ Rdx[3] \end{pmatrix} = \begin{pmatrix} E\{d[n]x[n]\} \\ E\{d[n]x[n-1]\} \\ E\{d[n]x[n-2]\} \\ E\{d[n]x[n-3]\} \end{pmatrix}$$

se obtuvo de teoría que la potencia de error viene dada por

$$\text{Pot}(e) = \xi(\underline{W}) = E\{d^2\} + \underline{W}^T \underline{\underline{R}}\, \underline{W} - 2\underline{P}^T \underline{W} = \xi_{min} + (\underline{W} - \underline{W}^*)\, \underline{\underline{R}}\, (\underline{W} - \underline{W}^*)$$

$$\nabla\xi = -2\underline{P} + 2\underline{\underline{R}}\,\underline{W} = 2\underline{\underline{R}}\,(\underline{W} - \underline{W}^*)$$

siendo $\boxed{\underline{W}^* = \underline{\underline{R}}^{-1} \cdot \underline{P}}$ coeficientes óptimos



Algoritmo del gradiente

se hace $\underline{W}_{n+1} = \underline{W}_n - \mu \nabla\xi(W_n)$

tamaño de los pasos proporcional a μ y proporcional a $\nabla\xi = 2\underline{\underline{R}}(\underline{W} - \underline{W}^*)$ (i.e. a lo lejos que estamos)

Caso unidimensional

se tiene $\begin{cases} \xi(w) = \xi_{min} + \lambda(w - w^*) \\ \nabla\xi(w) = 2\lambda(w - w^*) \end{cases}$

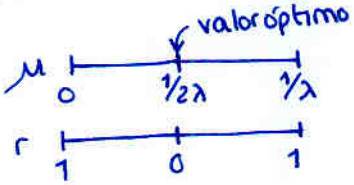
aplicando algoritmo del gradiente: $w_{n+1} = w_n - \overbrace{2\mu\lambda(w - w^*)}^{\mu\nabla\xi}$

↓ inducción

$$w_n = w^* + (1 - 2\mu\lambda)^n (w_0 - w^*)$$

razón de convergencia r · pesos iniciales · pesos óptimos

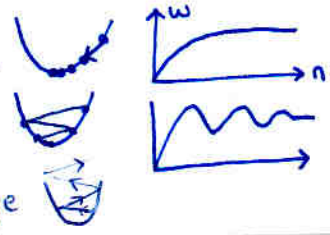
Debe cumplirse $\boxed{|r| < 1 \iff 0 < \mu < \frac{1}{\lambda}}$



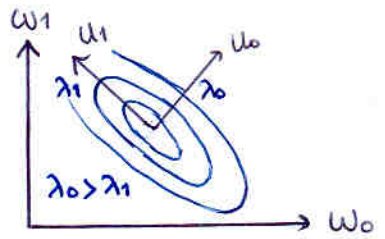
$r \in [0, 1) \rightarrow$

$r \in [-1, 0] \rightarrow$

$|r| > 1$ diverge



Caso multidimensional



λ_i : autovectores de $\underline{\underline{R}}$

u_i : autovector asociado a λ_i

entonces se tiene:

$$\begin{cases} u_0[n] = u_0^* + (1-2\lambda_0\mu)^n (u_0 - u_0)^* \\ u_1[n] = u_1^* + (1-2\lambda_1\mu)^n (u_0 - u_0)^* \\ \vdots \end{cases}$$

en cada dirección se tiene distinta razón de convergencia, pero todas comparten el mismo μ, por tanto hay que tomar

$$\mu < \frac{1}{\lambda_{max}}$$

por culpa de esto, en las que tengan λ pequeña

$r = (1 - \frac{\lambda_{min}}{\lambda_{max}})$ muy lento

```
R = matcor(x, Ncoef);
lambda = eig(R);
mu_max = 1./max(lamibda);
rmin = 1 - min(lamdda)/max(lambda);
```

Algoritmo LMS

Es el algoritmo del gradiente $W_{n+1} = W_n - \mu \hat{\nabla}\xi$

pero estimando el gradiente $\hat{\nabla}\xi = -2\, e[n] \cdot \underline{X}[n]$

se puede demostrar que $E\{\hat{\nabla}\xi\} = \nabla\xi$

- Paso a paso:

0: comienza instante: Llegan $x[n]$ y $d[n]$
1: Actualizar $\underline{X}[n]$ (meter $x[n]$ empujando los demás)
2: Filtrar $y[n] = \underline{W}_n^T \cdot \underline{X}[n]$
3: Obtener $e[n] = d[n] - y[n]$
4: Actualizar $\underline{W}_{n+1} = \underline{W}_n + 2\mu\, e[n]\, \underline{X}[n]$ ← en el instante anterior ya tenemos los pesos del siguiente

- Elección de μ

Teórico: $\mu < \frac{1}{\lambda_{max}}$ Práctico: $\mu < \frac{1}{\Sigma\lambda} = \frac{1}{Ncoef \cdot Potx}$

```
L = orden del filtro
Ncoef = L + 1;
Potx = mean(x.* x)
mu_max = 1/(Ncoef* Potx)
```

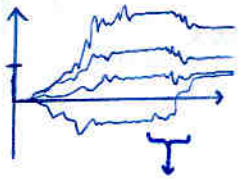
si la potencia instantánea $x.*x$ va cambiando (x no estacionaria) o bien cogemos la máxima prevista o vamos monitorizando

- Desajuste de los pesos finales:
  - $\hat{\nabla}\xi \neq \nabla\xi$ → Por tanto LMS va un poco "borracho"
  - Al llegar al óptimo $\nabla\xi = 0$ y ya no debería moverse, pero $\hat{\nabla}\xi \neq 0$ y por tanto se queda bailando en el óptimo ⇒ exceso de potencia de error
    μ grande ⇒ converge rápido pero al final baila mucho
  - Caso especial: si $\xi_{min} = 0 \rightarrow \hat{\nabla}\xi = 0$ al final → los pesos no bailan

LMS en Matlab

[ERROR, PESOS] = LMS (d, x, coef_iniciales, mu)

- ERROR: Pot instantanea de error → ERROR.*ERROR
  Pot media de error → mean(ERROR.*ERROR)
  plot(ERROR)
- PESOS: Representar la evolución de los pesos → plot(PESOS)
- coef_iniciales: típicamente zeros(1, Ncoef) ; L+1
- mu: En el enunciado de la práctica se pide 0'15 de su valor máximo → 0'15 * mumax



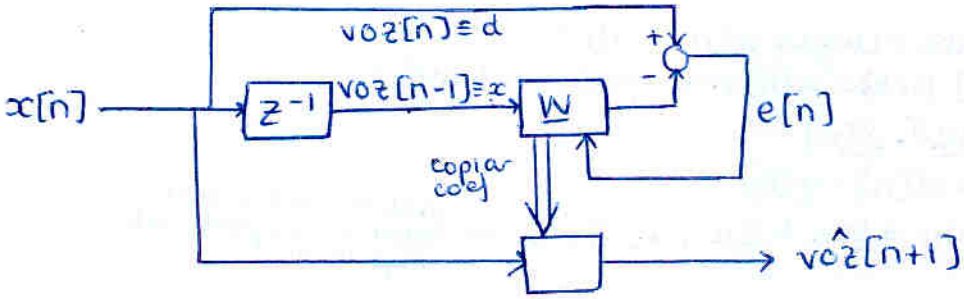
si cambia la pot de entrada (no estacionaria) cambian los pesos óptimos y deben reajustarse

Nota: en general no se ANULA el error, sino que se minimiza

Xi_min = min(ERROR.*ERROR)

Pesos optimos = PESOS(end, :);

Predicción adaptativa



en Matlab:

```
x = [0 ; voz];
d = [voz];
```

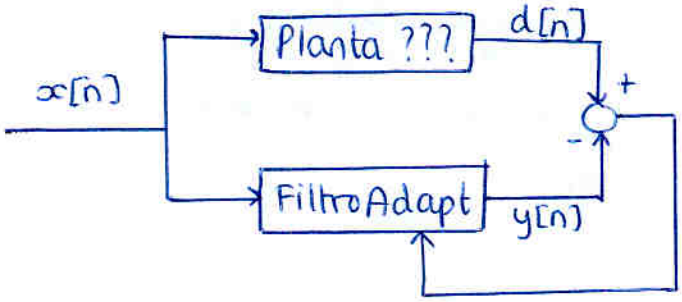
```
L = orden del filtro
Ncoef = L + 1
Potvoz = mean(voz.* voz)
mumax = 1/(Ncoef * Potvoz);
```

```
[ERROR, PESOS] = LMS (d, x, zeros(1, Ncoef), A * mumax);
                                              ↑ deben decirlo
```

El error "rebrota" y los pesos deben ajustarse cuando hay cambios en la potencia de la voz

La convergencia será más lenta cuanto menor sea
rmin = min(lambda)/max(lambda)

Identificación de sistemas

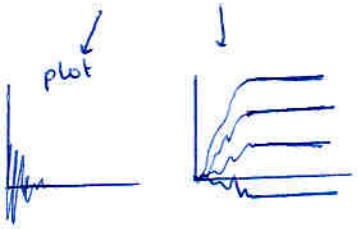


El filtro adaptativo se esforzará, con los coeficientes de que disponga, de imitar los coeficientes de la planta.

Mejor dicho, intentará imitar la H(ω) de la planta en aquellas frecuencias en las cuales exista x[n]

L = orden filtro,
Ncoef = L+1;
Potx = mean(x.* x)
numax = 1/(Ncoef * Potx)

[ERROR, PESOS] = LMS (d, x, zeros (1, Ncoef), 0'1 * numax);



se dan en la práctica (load idsist)

Efecto dispersión de autovalores

repetir anterior pero con x2 y d2

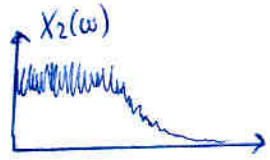
```
R2= matcor (x2, Ncoef);
lambda2= eig(R2);
```

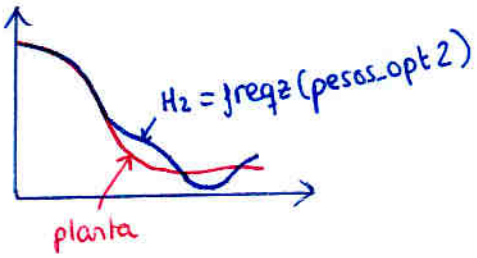
$\frac{\max(\text{lambda2})}{\min(\text{lambda2})} = 8'5 \cdot 10^3$ ← MUY lento

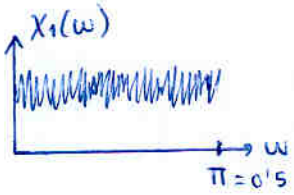
```
R = matcor (x, Ncoef);
lambda = eig(R);
```

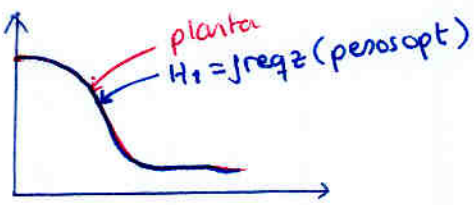
$\frac{\max(\text{lambda})}{\min(\text{lambda})} = 1'43$ ← muy bien

Influencia del espectro de entrada









con WOSA se puede representar el espectro de los ruidos de entrada de forma mucho mas bonita

$$[E, f] = \text{wosa}(x, L, N, \text{solape}, \text{ventana})$$

- E: hacer $10\log(\text{abs}(E))$
- f: fd $[0,0's]$
- L: duracion ventana 30
- N: nº ptos fft 256
- solape: 10
- ventana: hamming(30)

Trabajo Lab TDS

Decodificador de audio para C+

$f_s = 48\,kHz$

1. Generación de taps.hex para el FILTRO FIR paso bajo a 12'8 kHz

```
fs = 48000;
nbits = 16;
num_coef = 57;
N = num_coef - 1;
fdc = 12800 / fs; % frec. corte digital   (fc/fs, fc hasta fs/2)
Wn = 2 * fdc % matlab trabaja con w ∈ [0,1]
B = FIR1 ( N, Wn, 'low', hamming(N+1));
coefs = B * 2^(nbits-1); ← multiplicar por 2^15
coefs = round (coefs); ← redondear a enteros
save coefs.txt coefs -ascii
```

en MSDOS : asc2hex coefs.txt coefs.hex

2. Generación del tono de 12'8 kHz

usaremos el archivo seno.hex de la practica 9 que contiene $t[k] = \text{sen}\left(\frac{2\pi k}{4096}\right)$ $k = 0, 1, \dots, 4096$

y cogeremos una de cada M muestras $y[n] = \text{sen}\left(\frac{2\pi M n}{4096}\right)$

de forma que $f = f_s \frac{M}{4096} = 12'8\,kHz$

$$M = \frac{4096 \cdot 12'8\,kHz}{f_s = 48\,kHz} = 1092'2\overline{6} \simeq 1092$$

i0 } ocupados
i1 }

Datos

i2: muestras l_1 m2
i3: muestras l_2 m2
i2: muestras r_1 m2
i3: muestras r_2 m2

Programa

i4: coefs m4 (-1)
i5: tablaseno m5 (1092)
m6 (0)

Recálculo de M para no tener que reordenar

$$t[k] = \text{sen}\left(\frac{2\pi M}{A}\right)$$

$$f = f_s \frac{M}{A} = 12'8k$$

$$48k \cdot \frac{M}{A} = 12'8k$$

$$\frac{M}{A} = \frac{4}{15}$$

ejemplo:
A = 15000
M = 4000

⚠ No hacen falta tantos; si lo piensas un poco, con 15 el resultado es el mismo

```
k = 0:1:15-1 ;
t = sin(2*pi*k/15 );
t = t.*2^15 ;   t=round(t);   ← No olvidar x2^15
save seno.txt t -ascii

asc2hex seno.txt seno.hex
```

```
{*****************************************************************************

                        [ ..... ]

{ Variables para el filtro FIR }
{Cambiar "57" por el n£mero de coeficientes del filtro.}

.var/dm/ram/circ                    muestrasl_1[57];
.var/dm/ram/circ                    muestrasr_1[57];
.var/pm/ram/circ                    taps[57];
.var/dm/ram/circ                    muestrasl_2[57];
.var/dm/ram/circ                    muestrasr_2[57];

.var/dm                         puntL1;
.var/dm                         puntL2;
.var/dm                         puntR1;
.var/dm                         puntR2;

{Cambiar "fich.hex" por el nombre del fichero con los coeficientes      }
{en hexadecimal                                                         }
.init taps: <coefs.hex>;

{variables para multiplicar por tono}

.var/pm/circ                    tablaseno[15];
.init tablaseno: <seno.hex>;
.var/dm                          mm;
.var/dm                         dos;

.init mm: 4;          {frec_gen= fs * mm / 4096 }
                        {mm = round(frec_gen / fs * 4096)}
.init dos: 65536; {2 por 2 a la 15}

{ Ejemplo: Para generar aprox. 1 Khz. (1.00129 Khz.) con fs=44.1 --> mm = 93}


  **************        [ ..... ]     ****************


       0xc85c,     {  * LAB_TDS * Cambiar aqui la frecuencia de muestreo
                                  modificando el £ltimo caracter hexadecimal
                                  segun la tabla de abajo.
              ######## Frecuencias de muestreo ################
                       b0-3: 0=  8.
                             1=  5.5125
                             2= 16.
                             3= 11.025
                             4= 27.42857
                             5= 18.9
                             6= 32.
                             7= 22.05
                             8=   .
                             9= 37.8
                             a=   .
                             b= 44.1
                             c= 48.
                             d= 33.075
                             e=  9.6
                             f=  6.615

        ********       [ ......... ] ****************

       i2=^muestrasl_1;
       i3=^muestrasl_2;
       dm(puntL1)=i2;
       dm(puntL2)=i3;
       i2=^muestrasr_1;
       i3=^muestrasr_2;
  dm(puntR1)=i2;
  dm(puntR2)=i3;


       i4=^taps;
  i5=^tablaseno;

       m2=1;
       m4=-1;
  m5=4; {dm(mm); ¿y si pongo directamente 4?}
  m6=0;

       l2=%muestrasl_1;
       l3=%muestrasr_1;
       l4=%taps;
```

```
    l5=%tablaseno;

{-------------------------------------------------------------------------------
 -  * LAB_TDS *
 -  LO EJECUTADO CADA VEZ QUE SE QUIERE SACAR UNA MUESTRA (de cada canal L R)
 -
 -------------------------------------------------------------------------------}
input_samples:

 {
       i4=^taps;            i4: Apunta a los coefic. del filtro.
  i5=^tablaseno;

       m2=1;              m2: Para incrementar direcci¢n;
  m5=dm(mm);         m5: Para incrementar puntero del seno;
                         ¿y si pongo directamente 1092?
  m6=0;              m6: Para no incrementar la direccion;
 }


                    {i0,i1,m0,m1,l0,l1: NO se deben usar pues los usa el autobuffering de
                                salida  y  entrada}

  { ------------------------------------------------------------}
       { CANAL L ----------------------------------------------------------}
  { ------------------------------------------------------------}

  i2=dm(puntL1);
  i3=dm(puntL2);

  { Como i0 e i1 estaban ya utilizadas, no teniamos suficientes punteros
       de memoria de datos para los dos filtros de cada canal. Por tanto
       hemos utilizado variables (punteros) en memoria de datos.
              Utilizando cada vez sólo dos punteros i2 e i3 para el
       primer y segundo filtro, simplemente al inicio de cada canal
       cargamos las variables puntX1 y puntX2 (con X = L o R)
       en i2 e i3, y al finalizar el canal actualizamos esas variables
       con el valor con que se hayan
       quedado los punteros, para que la proxima muestra al cargarlas
       esté en el lugar correcto}

  { PRIMER FILTRO }
  { ------------- }

       ax0 = dm (rx_buf + 1);      {Leer nueva muestra de A/D}
       dm(i2,m2)=ax0;              {Guardar nueva muestra en memoria. }
                         {Tras la escritura de la muestra que
                              acabamos de adquirir, el puntero i2
                         queda apuntando a la muestra más antigua
                         del buffer}

       mr=0, mx0=dm(i2,m2), my0=pm(i4,m4);
                         {Inicializamos el bucle, para ello ponemos a
                              cero el acumulador de sumas, y cargamos los
                              registros con los valores del primer
                              producto a realizar. todo esto se hace
                              en una unica instruccion.}

       cntr=dm(taps_1);    {El registro cntr sirve para realizar bucles.
                              Se carga con el número de iteraciones
                              que queremos realizar (en este caso el
                              numero de coeficientes del filtro menos 1).}

       do filt_l1 until ce;
  filt_l1:           mr=mr+mx0*my0 (ss),mx0=dm(i2,m2),my0=pm(i4,m4);
                         {Bucle en que se realizan todos los productos y
                              sumas menos uno. En la misma instruccion
                              realizamos el producto y acumulación y
                              cargamos los registros para la siguiente
                              operacion}

       mr=mr+mx0*my0 (ss);       {Realizamos el ultimo producto y acumulacion sin
                         traernos mas datos de memoria}
                         {Al final, por cada muestra que entra se accede a
                              memoria de muestras un numero de veces igual
                              al numero de coeficientes del filtro mas uno:
                              una al leer la nueva muestra y el resto durante
                              los productos y sumas). Esto hace que cuando
                              llegue una nueva muestra del D/A el puntero
                              este apuntando a la muestra mas antigua del
                              buffer (la que se tiene que eliminar)
```

```
{ MULTIPLICACION POR TONO }
{ ---------------------- }

my0=pm(i5,m6);              {Leer muestra del seno al MAC (my0) y no modificar
                             el puntero (m6 = 0) a la tabla del seno
                             ya que aun hay que multiplicar
                             por el seno en el canal R, allí ya se
                             incrementará en M (m5) para preparar
                             el seno para la siguiente muestra}

mr=mr1*my0 (ss);            {Usar el MAC para multiplicar mr (que contenia
                             la muestra ya pasada por el primer
                             filtro) por my0 (que contiene la
                             muestra correspondiente del seno) }


{ SEGUNDO FILTRO }
{ -------------- }

dm(i3,m2)=mr1;              { Leer el resultado de la multiplicacion
                             por el seno y almacenar en memoria de
                             muestras del segundo filtro
                             sustituyendo a la muestra mas antigua}

mr=0, mx0=dm(i3,m2), my0=pm(i4,m4);  {Inicializamos el bucle}
                             {Nota: utilizaremos los mismos coeficientes
                                    con el mismo puntero que el primer
                                    filtro, ya que tras
                                    el filtro, el puntero se queda
                                    apuntando a la misma muestra que
                                    al incio, (i.e. se queda igual)
                                     por lo que los dos filtros
                                    no se estorbarán entre ellos}
                             { cuidado con de dónde saco las muestras:
                                    Las saco de i6 = ^muestras1_2, e
                                    incremento ciclicamente con m7, que
                                    contiene un 1 (no vale reusar m2 que
                                    tambien contenía un 1 porque el
                                    registro I y el M deben estar en el
                                    mismo DAG (Address Generator).
                                    Receurda que uno tiene los registros
                                     i,l,m = 0,1,2,3 y el otro 4,5,6,7 y
                                    no se pueden mezclar) }

cntr=dm(taps_1);

do filt_l2 until ce;
filt_l2:        mr=mr+mx0*my0 (ss),mx0=dm(i3,m2),my0=pm(i4,m4);

mr=mr+mx0*my0 (ss);


{my0=0x0001;                        Multiplicar por dos para compensar
mr=mr1*my0 (ss);                           la multiplicacion por el coseno
                                           y el posterior filtrado de
                                           la mitad superior del espectro}
                             { Hay que multiplicar por 0x0001 y coger
                                    el resultado de mr0, en lugar
                                    de mr1 }



      dm (tx_buf + 1) = mr1;          {Enviar resultado a D/A}


dm(puntL1)=i2;
dm(puntL2)=i3;

{ ------------------------------------------------------------}
      { CANAL R ---------------------------------------------------}
{ ------------------------------------------------------------}

i2=dm(puntR1);
i3=dm(puntR2);

{ PRIMER FILTRO }
{ ------------- }


      ax0 = dm (rx_buf + 2);     {Leer nueva muestra de A/D}
```

```
        dm(i2,m2)=ax0;              {Guardar nueva muestra en memoria}


        mr=0, mx0=dm(i2,m2), my0=pm(i4,m4);

        cntr=dm(taps_1);
        do filt_r1 until ce;
filt_r1:        mr=mr+mx0*my0 (ss),mx0=dm(i2,m2),my0=pm(i4,m4);
        mr=mr+mx0*my0 (ss);



{ MULTIPLICACION POR TONO }
{ ----------------------- }


my0=pm(i5,m5);          {Traer valor correspondiente de la tabla del seno
                                y preparar puntero para que apunte al
                                siguiente valor para la siguiente muestra que
                                venga i.e. añadirle m5 = mm = 1092}
mr=mr1*my0 (ss);              {Usar el MAC para multiplicar mx0 (que contenia
                                la muestra ya pasada por el primer
                                filtro) por my0 (que contiene la
                                muestra correspondiente del seno) }

{ SEGUNDO FILTRO }
{ -------------- }

dm(i3,m2)=mr1;                { Leer el resultado de la multiplicacion
                                por el seno y almacenar en memoria
                                sustituyendo a la muestra mas antigua.
                                Esta vez usamos i3 = muestrasr_2 }

mr=0, mx0=dm(i3,m2), my0=pm(i4,m4);  {Inicializamos el bucle}


cntr=dm(taps_1);

do filt_r2 until ce;
filt_r2:        mr=mr+mx0*my0 (ss),mx0=dm(i3,m2),my0=pm(i4,m4);

mr=mr+mx0*my0 (ss);

{my0=0x0001;
mr=mr1*my0 (ss);}


        dm (tx_buf + 2) = mr1;          {Enviar resultado a D/A}

dm(puntR1)=i2;
dm(puntR2)=i3;

        rti;


.endmod;
```