Anexo A

Método Lundeby para obtener parámetros acústicos

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ldbparam.m
% Esta función evalúa los diversos parámetros acústicos de una sala, con el método de procesamiento de señales de Lundeby. La entrada es la Respuesta Impulsiva del Recinto y la frecuencia de muestreo. La salida es un archivo de texto con los valores de los diversos parámetros de cada banda de frecuencia.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [salida]=ldbparam(IR,fs)

banda = filtros(IR,fs);
t = size(banda,2);  % Bandas de octavas
for n = 1:t
    s(1,n) = ceil(1000*2^(n-5));
    comeco = inicio(banda(:,n));
    fim = lundeby(banda(comeco:end,n),fs,1);
    title(['Punto de Cruce - Banda ',num2str(s(1,n))])
        if n == t-2
            title('Punto de Cruce - Compensación A ')
        elseif n == t-1
            title('Punto de Cruce - Compensación C ')
        elseif n == t
            title('Punto de Cruce - Lineal ')
        end
    aux = banda(comeco:fim,n).^2;
    [s(2,n),s(3,n),s(4,n),s(5,n),s(6,n)] = energeticos(aux,fs);
    [s(7,n),s(8,n),s(9,n),s(10,n)] = reverberacao(aux,fs,0);
    title(['Curva de Decaimiento - Banda ',num2str(s(1,n))])
        if n == t-2
            title('Curva de Decaimiento - Compensación A ')
        elseif n == t-1
            title('Curva de Decaimiento - Compensación C ')
        elseif n == t
            title('Curva de Decaimiento - Lineal ')
        end
end
salida = s;
salida(1,t-2) = (['A']);
salida(1,t-1) = (['C']);
salida(1,t) = (['L']);
tabla(s,size(banda,2))
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% energeticos.m
% Esta función calcula la energía relacionada con los parámetros acústicos de una sala como la claridad, la definición y el tiempo central. La respuesta impulsiva cuadrática de la sala debe ir a la entrada, con su %respectiva frecuencia de muestreo.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [C50,C80,D50,D80,CT]=energeticos(energia,Fs);

t50 = round(0.05*Fs);
t80 = round(0.08*Fs);

%Clarity = Razón entre la energía inicial de la señal y la energía remanente.
C50 = 10*log10(integral(energia(1:t50),Fs)/integral(energia(t50:end),Fs));
C80 = 10*log10(integral(energia(1:t80),Fs)/integral(energia(t80:end),Fs));

%Definition = Razón entre la energía inicial de la señal y la energía total de la señal.
E = integral(energia,Fs);
D50 = integral(energia(1:t50),Fs)/E*100;
D80 = integral(energia(1:t80),Fs)/E*100;

%Tempo Central= Equivalente al centro de gravedad de la curva de energía.
x=(0:length(energia)-1)/Fs;
CT = integral(energia(:).*x(:),Fs)/E;
end

function P = integral(p,Fs)
P = (sum(p)-(p(1)+p(end))/2)/Fs;
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% reverberacion.m
% Esta función evalúa el tiempo de decaimiento en un recinto, dada la respuesta impulsiva.
% [EDT,T20,T30,T40] = reverberacion(IR^2,Fs,flag)
% La entrada es la respuesta impulsiva cuadrática y su frecuencia de muestreo, el retardo de esta respuesta impulsiva no deberá estar presente; la variable bandera (flag) especifica si la función grafica (1) o no (2) las curvas de decaimiento, la salida son los tiempos de decaimiento T20, T30, T40 y EDT
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [EDT,T20,T30,T40] = reverberacion(varargin)

ir = varargin{1};
Fs = varargin{2};
E(length(ir):-1:1) = (cumsum(ir(length(ir):-1:1))/sum(ir));

if find(E < 0)
   E(min(find(E < 0)):end) = [];
   E=10*log10(E);
else
   E=10*log10(E);
end

if nargin == 3
   flag = varargin{3};
```

```
else
   flag = 0;
end
x = (0:length(E)-1)/44100;

%Calcula el Early Decay Time (EDT) de la señal. La curva de Schroeder siempre debería haber sido obtenida de una respuesta al impulso sin ruido desde el principio.
t10   = min(find(E < -15));
[A10,B10] = intlinear(x(1:t10),E(1:t10));
EDT = (-60)/(B10);

% Calcula los tiempos de reverberación de la respuesta impulsiva (T20 y T30) a partir de la curva de Schroeder (db) prevista en el argumento de entrada

if flag == 1
   %gráfico de salida
   figure, plot(x,E,'LineWidth',1.5);
end

begin = min(find(E < -5));
t25   = min(find(E < -25));    %Si la curva no es monótona,  el primer punto donde la curva llega a
                                    %-25dB y -35dB
t35   = min(find(E < -35));    %Los límites de la región de iteración
t45   = min(find(E < -45));

%Usando 20dB
if ~isempty(t25)
   [A20,B20] = intlinear(x(begin:t25),E(begin:t25));
   T20 = (-60)/(B20);
else
   T20=NaN;          %Caso en que la respuesta impulsiva no presenta la dinámica suficiente
end

%Usando 30dB
if ~isempty(t35)
   [A30,B30] = intlinear(x(begin:t35),E(begin:t35));
   T30 = (-60)/(B30);
else
   T30=NaN;          %Caso en que la respuesta impulsiva no presenta la dinámica suficiente
end

if nargout == 4
   %Usando 40dB
   if ~isempty(t45)
      [A40,B40] = intlinear(x(begin:t45),E(begin:t45));
      T40 = (-60)/(B40);
   else
      T40=NaN;
   end
else
   T40=NaN;          %Caso en que la respuesta impulsiva no presenta la dinámica suficiente
end

if flag == 1
   title('Aproximación de los tiempos de decaimiento');
```

```
    ylim([-70 0]);
    ylimit = ylim;
    xlabel('tiempo (s)'), ylabel('dB')
    xlim([0 max([T20 T30 T40])*1.1]);
    xlimit=xlim;
    line([0,(-60-A10)/(B10)],[A10,-60],'Color','m','LineWidth',.5);
    line([0,(-60-A20)/(B20)],[A20,-60],'Color','r','LineWidth',.5);
    line([0,(-60-A30)/(B30)],[A30,-60],'Color','g','LineWidth',.5);
        if nargout == 4
            line([0,(-60-A40)/(B40)],[A40,-60],'Color','y','LineWidth',.5);
        end
    line([xlimit(1),xlimit(2)],[-60,-60],'Color',[.4,.4,.4],'LineWidth',.5);
    legend('Curva de Schroeder',['EDT (ms) = ',num2str(EDT*1000)],['T_2_0 (ms) =
',num2str(T20*1000)],...
     ['T_3_0 (ms) = ',num2str(T30*1000)],['T_4_0 (ms) = ',num2str(T40*1000)])
end
```

Anexo B

Algoritmos Adaptativos

B.1. Algoritmo LMS Matlab

```
%   Author(s): S.C. Douglas
%   Copyright 1999-2002 The MathWorks, Inc.
%   $Revision: 1.9 $  $Date: 2002/10/16 16:13:49 $
%   Algoritmo LMS adaptativo
%% function [y,e,W] = lms(x,d,M,STEP,LEAKAGE,COEFFS)

function [y,e,W] = lms(x,d,M,STEP,LEAKAGE,COEFFS)
[Mx,Nx] = size(x);
[Md,Nd] = size(d);

%% Inicialización de variables
ntr = max([Mx Nx]);                                  %  número de iteraciones temporal
if nargin > 2, L = M;       else L=10;  end          %  número de coeficientes
if nargin > 3, mu=STEP;     else mu=1;  end          %  paso de convergencia
if nargin > 4, lam=LEAKAGE; else lam=1; end          %  asignación de fuga
if nargin > 5, W = COEFFS;  else W = zeros(1,L); end %  inicialización del vector de coeficientes
y = zeros([Mx Nx]);                % inicialización del vector de salida
e = y;                             % inicialización del vector error
X = zeros(L,1);                    % inicialización del buffer temporal de la señal de entrada
X(1:L-1)= zeros(L-1,1);            % asignación del buffer de la señal de entrada

%% Bucle principal
for n=1:ntr,
   X(2:L) = X(1:L-1);              %  cambio temporal de la caída de buffer de la señal de entrada
   X(1) = x(n);                    %  asignación actual de la muestra de entrada
   y(n) = W*X;                     %  cálculo y asignación actual de la muestra de salida
   e(n) = d(n) - y(n);             %  cálculo y asignación actual de la muestra de la señal de error
   W = lam*W + mu*e(n)*X';         %  actualización de los coeficientes del filtro
end
```

B.2. Algoritmo NLMS Matlab

```
%   Author(s): S.C. Douglas
%   Copyright 1999-2005 The MathWorks, Inc.
%   $Revision: 1.10.4.2 $  $Date: 2005/12/22 18:04:47 $
%   Algoritmo nlms
```

```
function [y,e,W] = nlms(x,d,M,STEP,LEAKAGE,COEFFS)
[Mx,Nx] = size(x);
[Md,Nd] = size(d);

%% Inicialización de variables
ntr = max([Mx Nx]);                                  %  número de iteraciones temporal
if nargin > 2, L = M;       else L=10;  end          %  número de coeficientes
if nargin > 3, mu=STEP;     else mu=1;  end          %  paso de convergencia
if nargin > 4, lam=LEAKAGE; else lam=1; end          %  asignación de fuga
if nargin > 5, W = COEFFS;  else W = zeros(1,L); end %  inicialización del vector de coeficientes
y = zeros([Mx Nx]);              %  inicialización del vector de salida
e = y;                           %  inicialización del vector error
X = zeros(L,1);                  %  inicialización del buffer temporal de la señal de entrada
X(1:L-1)= zeros(L-1,1);          %  asignación del buffer de la señal de entrada

%% Bucle principal
for n=1:ntr,
   X(2:L) = X(1:L-1);            %  cambio temporal de la caída de buffer de la señal de entrada
   X(1) = x(n);                  %  asignación actual de la muestra de entrada
   normX = X'*X + eps;           %  actualización vector normalizado de entrada
   y(n) = W*X;                   %  cálculo y asignación actual de la muestra de salida
   e(n) = d(n) - y(n);           %  cálculo y asignación actual de la muestra de la señal de error
   W = lam*W + mu/normX*e(n)*X'; %  actualización de los coeficientes del filtro
end
```

B.3. Algoritmo RLS Matlab

```
%   Author(s): S.C. Douglas
%   Copyright 1999-2002 The MathWorks, Inc.
%   $Revision: 1.11 $  $Date: 2002/10/15 20:13:18 $
%   Algoritmo RLS
%%  function [y,e,W] = rls(x,d,M,LAMBDA,INVCOV,COEFFS,STATES)

function [y,e,W] = rls(x,d,M,LAMBDA,INVCOV,COEFFS,STATES)
[Mx,Nx] = size(x);
[Md,Nd] = size(d);

%% Inicialización de variables
y = zeros([Mx,Nx]);                          %  inicialización del vector de salida
ntr = max([Mx,Nx]);                          %  número de iteraciones temporales
if nargin > 2, L = M; else, L=10; end        %  número de coeficientes
X = zeros(L,1);                              %  inicialización del buffer de la señal de entrada temporal
if nargin > 3, lam=LAMBDA; else, lam =1; end %  asignación del factor de olvido
if nargin > 4, P=INVCOV; else, P=1e3*eye(L); end
if nargin > 5, W = COEFFS; else, W=  zeros(1,L); end %  vector de coeficientes iniciales
if nargin > 6, X(1:L-1)=STATES; else, X(1:L-1)=zeros(L-1,1); end %  asignación del buffer de entrada
e = y;                                       %  inicialización del vector de error
invlam = 1/lam;                              %  asignación del factor de olvido inverso

%% Bucle principal
for n=1:ntr,
%  Actualización de la señal al buffer de entrada
```

```
   X(2:L) = X(1:L-1);              %  selección temporal del buffer de bajada de la señal de entrada
   X(1) = x(n);                    %  asignar actual muestra de la señal de entrada
   % Carga de la matriz de covarianza
   XP = X'*P;
   PX = P*X;
   invden = 1/(lam + XP*X);
   K = invden*PX;                  %  cálculo del vector de ganancia Kalman
   P = invlam*(P - K*XP);          %  actualización de la matriz de covarianza
   % Cálculo de la señal de salida, señal de error, y carga de coeficientes
   y(n) = W*X;                     %  cálculo y asignación del la muestra de la salida actual
   e(n) = d(n) - y(n);             %  cálculo y asignación de la señal de muestreo del error actual
   W = W + e(n)*K';                %  actualización del vector de coeficientes
end
```

B.4. Algoritmo FDAF Matlab

```
%   Author(s): S.C. Douglas
%   Copyright 1999-2003 The MathWorks, Inc.
%   $Revision: 1.18.4.2 $  $Date: 2004/04/12 23:15:31 $
%   Algoritmo FDAF
%%  function [y,e,h] =
fdaf(x,d,M,STEP,LEAKAGE,DELTA,LAMBDA,BLOCKLEN,OFFSET,COEFFS,STATES)

function [y,e,h] =
fdaf(x,d,M,STEP,LEAKAGE,DELTA,LAMBDA,BLOCKLEN,OFFSET,COEFFS,STATES)
[Mx,Nx] = size(x);
[Md,Nd] = size(d);

%% Inicialización de Variables
ntr = length(x);                   %  longitud del vector de entrada iteración temporal
if nargin > 2,                     %  M
   L=M;                            %  número de coeficientes del filtro
else,
   L=10;                           %  por defecto 10 coeficientes
end
if nargin > 7,                     %  BLOCKLEN
   N=BLOCKLEN;
else,
   N = L;                          %  Por defecto de igual longitud al filtro.
end
if (rem(length(x),N)~=0),          %comprueba si la dimensión de la señal es múltiplo del bloque
   error('La longitud de la señal de entrada debe poder ser divisible entre la longitud del bloque.')
end
ntrB = floor(length(x)/N);         %  número de iteraciones bloque temporal
y = zeros(size(x));                %  inicialización del vector de salida
e = y;                             %  inicialización de la señal de error
X = zeros(L+N,1);                  %  inicialización temporal del buffer de la señal de entrada
E = zeros(L+N,1);                  %  inicialización temporal del buffer de la señal de error
Ef = zeros(L+N,1);                 %  inicialización temporal del buffer de error
nnL = 1:L;                         %  variable índice utilizado para el búfer de la señal de entrada
nnLpN = N+1:N+L;                   %  variable índice utilizado para el búfer de la señal de entrada
nnNpL = L+1:L+N;                   %  variable índice utilizado para el búfer de la señal de entrada
```

```
nnLpNr = L+N:-1:N+1;            %  variable índice utilizada para la actualización de los coeficientes
if nargin > 9,                  %  COEFFS
   COEFFS = COEFFS(:).';        %  Coeficientes son un vector fila
          if length(COEFFS)~= L,
          error(sprintf(['El numero de coeficientes en el dominio del tiempo'debe ser igual a la \n',...
          'longitud del filtro.']));
          end
   FFTW=(fft(COEFFS,L+N)).';
else,
   FFTW=(zeros(1,L+N)).';
end
if nargin > 5,                          %  DELTA
   normFFTX =DELTA*ones(L+N,1);         %  Inicializa y asigna la intensidad a la señal de entrada FFT
else,
   normFFTX =ones(L+N,1);
end
if nargin > 10,                         %  STATES
   X(nnLpNr) =STATES;                   % asigna el buffer de la señal de entrada
else,
   X(nnLpNr) =zeros(L,1);
end
if nargin > 3,                          %  STEP
   mu =STEP;                            %  paso adaptativo del filtro
else,
   mu =1;                               %  asigna el paso del filtro
end
if nargin > 6,                          %  LAMBDA
   bet=LAMBDA;                  % Factor promedio para calcular la ventana exponencial de la señal
                                %de intensidad FFT para la actualización de los coeficientes.
else
   bet=0.9;
end
ombet = 1 - bet;                % calcula (1 - h.AvgFactor)
if nargin > 4,                  %  LEAKAGE
   lam =LEAKAGE;                % parámetro de fuga del algoritmo entre cero y uno.
else
   lam =1;
end
if nargin > 8,                  %  OFFSET
   Offset = OFFSET;              % compensación para términos normalizados
else
   Offset =eps;
end
%%
ZN = zeros(N,1);                 %  asignación N-dimensional de vector cero

%% Bucle principal
for n=1:ntrB,
   nn = ((n-1)*N+1):(n*N);      %  Índice de los bloques de señal actual
   X(nnL) = X(nnLpN);           %  cambio almacenamiento temporal  de la señal de entrada
   X(nnNpL) = x(nn);            %  asignación del vector de entrada actual
   FFTX = fft(X);               %  cálculo de la FFT en el vector señal entrada
   Y = ifft(FFTW.*FFTX);        %  cálculo del vector de la salida actual
```

```
   y(nn) = Y(nnNpL);              %  asignación actual del bloque de salida de la señal
   e(nn) = d(nn) - y(nn);         %  asignación actual del bloque de la señal de error
   E(nnNpL) = mu*e(nn);           %  asignación actual del bloque de la señal de error
   FFTE = fft(E);                 %  cálculo de la FFT del vector de error
   normFFTX = bet*normFFTX + ombet*real(FFTX.*conj(FFTX));
                                  %  actualización de la fft de la señal de
                                  %  potencia de entrada
   G = ifft(FFTE.*conj(FFTX)./(normFFTX + Offset));
                                  %  cálculo del vector gradiente
   G(nnNpL) = ZN;                 %  imponer la restricción de gradiente
   FFTW = lam*FFTW + fft(G);      %  actualización de los coeficientes en el dominio de la frecuencia
end
h=ifft(FFTW);
if isreal(x) && isreal(d),
   y = real(y);                   %  limitar la señal de salida para ser de valor real
   e = real(e);                   %  limitar la señal de error a ser de valor real
end                               %  fin del programa
```

B.5. Algoritmo XLMS Matlab

```
%   Author(s): S.C. Douglas
%   Copyright 1999-2002 The MathWorks, Inc.
%   $Revision: 1.15 $  $Date: 2002/10/17 22:40:23 $

function [y,e,W] = xlms(x,d,Le,STEPSIZE,LEAKAGE,PATHCOEFFS,PATHEST,FSTATES,PSTATES,
COEFFS,STATES)

[Mx,Nx] = size(x);
[Md,Nd] = size(d);

%% Inicialización de variables
ntr = length(x);                  %  número temporal de muestras procesadas
if nargin > 2,L= Le;else, L=10; end               % longitud del filtro
if nargin > 5,                                    % PATHCOEFFS
   SecondaryPathCoeffs=PATHCOEFFS;                % coeficientes del filtro secundario
                                                  % para el sensor del actuador del sensor secundario
   SecondaryPathEstimate=PATHCOEFFS;
end
if nargin > 6,    % PATHEST
   SecondaryPathEstimate=PATHEST;                 % Estimación del filtro secundario
end
M = length(SecondaryPathCoeffs);                  % número de coeficientes del filtro secundario
N = length(SecondaryPathEstimate);                % número de coeficientes estimados
mLN = max([L N]);                 %  máximo entre L y N
nnL = 1:L-1;                      %  variable índice utilizado para el búfer de entrada de señal
nnLp1 = nnL + 1;                  %  variable índice utilizado para el búfer de entrada de señal
nnLL = 1:L;                       %  variable índice utilizado para el búfer de entrada de señal
nnNN = 1:N;                       %  variable índice utilizado para el búfer de entrada de señal
nnM = 1:M-1;                      %  variable índice utilizado para el búfer de entrada de señal
nnMp1 = nnM + 1;                  %  variable índice utilizado para el búfer de entrada de señal
H = SecondaryPathCoeffs;          %  inicializa el modelo de la ruta secundaria
Hhat = SecondaryPathEstimate;%  inicializa y asigna la estimación del modelo de ruta secundaria
```

```
pathord=length(SecondaryPathCoeffs)-1;
Npath = pathord;
pathestord=length(SecondaryPathEstimate)-1;
Nest =  pathestord;
y = zeros(size(x));                 %  inicializa vector de salida
e = y;                              %  inicializa vector de error
X = zeros(mLN,1);                   %  inicializa buffer temporal de entrada
Y = zeros(M,1);                     %  inicializa buffer temporal de salida
F = zeros(L,1);                     %  inicializa buffer temporal de la señal de entrada filtrada
if nargin > 3,mu = STEPSIZE;else, mu=0; end
if nargin > 4,lam= LEAKAGE;else, lam=1; end
if nargin > 7,                      %  asignación del buffer de entrada filtrado.
   FilteredInputStates=FSTATES;
   F(nnL)=FilteredInputStates;
else,
   F(nnL)=zeros(1,L-1);
end
if nargin > 8,                      %  asignación de estados de buffer secundario
   SecondaryPathStates=PSTATES;
   Y(nnM)= SecondaryPathStates;
else,
   Y(nnM)= zeros(1,Npath);
end
 all zeros.
if nargin > 9,                      % COEFFS coeficientes de entrada
% Asegúrese de que los coeficientes son una fila
   COEFFS = COEFFS(:).';
   if length(COEFFS)~= L,
     error(sprintf(['El numero de coeficientes \n',...
            'debe ser igual a la longitud del filtro.']));
   end
   W=COEFFS;
else,
   W=zeros(1,L);                    %  por defecto vector de ceros de igual longitud del filtro
end
if nargin > 10,X(1:mLN-1)= STATES;
else,         X(1:mLN-1)= zeros(1,max(L-1,Nest));
end

%% Bucle principal
for n=1:ntr,
   X(2:mLN) = X(1:mLN-1);           %  selecciona temporalmente el buffer de bajada de señal de entrada
   X(1) = x(n);                     %  asigna la muestra actual de entrada
   Y(nnMp1) = Y(nnM);               %  selecciona temporalmente el buffer de bajada de la señal de salida
   Y(1) = W*X(nnLL);                %  calcula la muestra actual de señal de salida
   y(n) = Y(1);                     %  asigna la muestra de señal de salida actual
   e(n) = d(n) + H*Y;               %  calcula y asigna la muestra actual de la señal de error
   F(nnLp1) = F(nnL);               %  selecciona temporalmente el buffer de bajada de la señal de
                                    %  entrada filtrada
   F(1) = Hhat*X(nnNN);             %  asigna la muestra de señal de entrada actual filtrada
   W = lam*W - mu*e(n)*F';          %  actualiza el vector de coeficientes del filtro
end
```

Anexo C

Implementación en Matlab de un Ecualizador Paramétrico digital de orden superior

```
%% Ecualizador paramétrico Butterworth
%% Copyright (c) 2005 by Sophocles J. Orfanidis

% N : Orden del filtro analógico
% G0: Ganancia de referencia
% G : Ganancia pico en f0
% GB: Ganancia de Ancho de banda
% f0: frecuencia central
% Bw: Ancho de Banda
% fs: frecuencia de muestreo

function [B,A] = EcuaParButter(N,G0,G,GB,f0,Bw,fs)

w0 = 2*pi*f0/fs;
Dw = 2*pi*Bw/fs;
G0 = 10^(G0/20);              % convierte las ganancias de dB a unidades absolutas
G  = 10^(G/20);
GB = 10^(GB/20);

% N es el orden del filtro Analogico
r=N-(fix(N./2)).*2;           %fix redondea al entero mas cercano a 0
L =(N-r)/2;
if G==G0,                     %Si la ganancia de referencia es igual a la
                              %ganancia del filtro no realiza filtrado
  B  = G0*[1 0 0 0 0]; A  = [1 0 0 0 0];
  return;
end

c0 = cos(w0);
if w0==0,    c0=1;  end       % casos especiales
if w0==pi/2, c0=0;  end
if w0==pi,   c0=-1; end

WB=tan(Dw/2);
e =sqrt((G^2-GB^2)/(GB^2-G0^2));
g = G^(1/N); g0 = G0^(1/N);

a = e^(1/N);
b = g0*a;

if r==0, Ba(1,:)=[1,0,0]   ; Aa(1,:)=[1,0,0]; end
if r==1, Ba(1,:)=[g*WB,b,0]; Aa(1,:)=[WB,a,0];end
```

```
if L>0,
   i = (1:L)';
   ui = (2*i-1)/N;
   ci = cos(pi*ui/2); si = sin(pi*ui/2);
   v = ones(L,1);
       Ba(1+i,:) = [g^2*WB^2*v, 2*g*b*si*WB, b^2*v];
       Aa(1+i,:) = [WB^2*v, 2*a*si*WB, a^2*v];
end

[B,A] = blt(Ba,Aa,w0);              % transformada bilineal
%% Las secciones del filtro de orden n son la convolución de sus secciones
K = size(B,1);                      % tamaño de la sección del numerador
L = size(A,1);                      % tamaño de la sección del denominador
M = max(K,L);                       % extendido a un numero común de secciones
B(K+1:M,:)=0; B(K+1:M,1)=1;         % secciones tradicionales son triviales, p.e., [1,0,0,...,0]
A(L+1:M,:)=0; A(L+1:M,1)=1;
bb=B(1,:);
aa=A(1,:);
for i=2:M,
   bb=conv(bb,B(i,:));
   aa=conv(aa,A(i,:));
end
B=bb;                               %numerador
A=aa;                               %denominador


%% Transformada bilinear
%% Copyright (c) 2005 by Sophocles J. Orfanidis

function [B,A,Bhat,Ahat] = blt(Ba,Aa,w0)

if nargin==0, help blt; return; end
K = size(Ba,1);                     % número de secciones de filtro

B = zeros(K,5); A = zeros(K,5);
Bhat = zeros(K,3); Ahat = zeros(K,3);

B0 = Ba(:,1); B1 = Ba(:,2); B2 = Ba(:,3);          % notación simplificada
A0 = Aa(:,1); A1 = Aa(:,2); A2 = Aa(:,3);          % A0 no debe ser cero

c0 = cos(w0);

if w0==0,    c0=1;  end;            % casos especiales
if w0==pi;   c0=-1; end;
if w0==pi/2; c0=0;  end;

i = find((B1==0 & A1==0) & (B2==0 & A2==0))        % encontrar las secciones de orden-0th

Bhat(i,1) = B0(i)./A0(i);
Ahat(i,1) = 1;
```

```
B(i,1) = Bhat(i,1);
A(i,1) = 1;

i = find((B1~=0 | A1~=0) & (B2==0 & A2==0));    % encontrar las secciones analógicas de primer orden

D = A0(i)+A1(i);
Bhat(i,1) = (B0(i)+B1(i))./D;
Bhat(i,2) = (B0(i)-B1(i))./D;
Ahat(i,1) = 1;
Ahat(i,2) = (A0(i)-A1(i))./D;

B(i,1) = Bhat(i,1);
B(i,2) = c0*(Bhat(i,2)-Bhat(i,1));
B(i,3) = -Bhat(i,2);
A(i,1) = 1;
A(i,2) = c0*(Ahat(i,2)-1);
A(i,3) = -Ahat(i,2);

i = find(B2~=0 | A2~=0);          % encontrar las secciones analógicas de segundo orden
D = A0(i)+A1(i)+A2(i);
Bhat(i,1) = (B0(i)+B1(i)+B2(i))./D;
Bhat(i,2) = 2*(B0(i)-B2(i))./D;
Bhat(i,3) = (B0(i)-B1(i)+B2(i))./D;
Ahat(i,1) = 1;
Ahat(i,2) = 2*(A0(i)-A2(i))./D;
Ahat(i,3) = (A0(i)-A1(i)+A2(i))./D;

B(i,1) = Bhat(i,1);
B(i,2) = c0*(Bhat(i,2)-2*Bhat(i,1));
B(i,3) = (Bhat(i,1)-Bhat(i,2)+Bhat(i,3))*c0^2 - Bhat(i,2);
B(i,4) = c0*(Bhat(i,2)-2*Bhat(i,3));
B(i,5) = Bhat(i,3);

A(i,1) = 1;
A(i,2) = c0*(Ahat(i,2)-2);
A(i,3) = (1-Ahat(i,2)+Ahat(i,3))*c0^2 - Ahat(i,2);
A(i,4) = c0*(Ahat(i,2)-2*Ahat(i,3));
A(i,5) = Ahat(i,3);

if c0==1 | c0==-1                 % LP o HP filtro shelving
  B = Bhat;                       % B,A son de segundo orden
  A = Ahat;
  B(:,2) = c0*B(:,2);             % cambia el signo si w0=pi
  A(:,2) = c0*A(:,2);
  B(:,4:5) = 0;
  A(:,4:5) = 0;
end
```