Escuela Universitaria Ingeniería Técnica Industrial ZARAGOZA

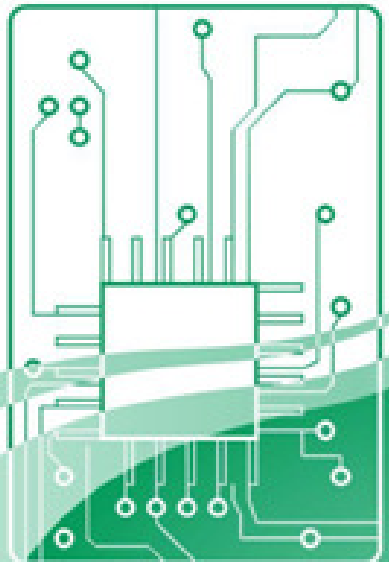
Universidad Zaragoza 1542

Efectos musicales digitales. Programación alternativa mediante el uso de la Transformada Wavelet Continua Compleja (CCWT)

Proyecto Fin de Carrera

Anexos *(scripts de programación)*

Alumno: **Óscar Gil Bailo**
Especialidad: **Electrónica Industrial**
Director: **Jesús Ponce de León Vázquez**
Convocatoria: **Septiembre 2011**



En este documento de anexos se adjuntan los scripts de programación en *Matlab* de cada uno de los efectos musicales analizados en la memoria del proyecto. Para comprobar el resultado de estos algoritmos de programación se han generado una serie de archivos de audio. Debido a que en el repositorio de documentos electrónicos de la Universidad de Zaragoza (*Zaguán*) sólo pueden cargarse archivos en formato *pdf*, los archivos de audio se encuentran a disposición del lector en el siguiente enlace:
http://www.megaupload.com/?d=UTR6I1ZP

ÍNDICE

1. CWAS inicio

```
%%%% Previamente se ejecuta el script frame2frame para obtener la
%%%% matriz CWT de cada sonido.
UMBRAL=0.995; %Energía a partir de la cual se desechan los
% parciales: max=1.
GAP=2^8; %Tamaño del miniframe: si se altera, el MM/GAP=16 de la
% función.
            %SELECCION habrá que cambiarlo adecuadamente.
LEN=4; %Longitud mínima en miniframes de un parcial para ser tenido
% en cuenta.

MAXPARTIALS=2000; %Número máximo de parciales con los que
% trabajaremos para que la matriz de máscaras sea tan grande como la
% matriz cwt, podemos trabajar con un número máximo de parciales de
% duracion_señal*185(bandas)/(3*(2+num_tot_littleframes)), más de
% 10000 en general. La idea sería no borrar nunca. Con esto
% obtenemos que la etiqueta de la máscara coincida con la posición
% SIEMPRE (mientras no borremos nada).

TAMLF=16; %Número de miniframes por frame. Revísese si GAP o MM
cambian de
            %256 y 4095 respectivamente.

[file_name, file_path] = uigetfile('*.cwt', 'Select input .cwt file
...');
mask=zeros(3,2,MAXPARTIALS);
%mask0=zeros(3,2,MAXPARTIALS);
cwt=[];
synthetic=[];
fid = fopen([file_path file_name],'rb');   %open file.
[filename, mode, machineformat]=fopen(fid);
cfile=fread(fid, 1, 'int8', machineformat);
binname = fread(fid, cfile, 'char',machineformat);
wavname = char(binname');
[signal,Fs]=wavread(wavname);
M=fread(fid, 1, 'int32','ieee-le'); % File length.
N=fread(fid, 1, 'int16', machineformat); % Number of bands.
freqs=fread(fid, N, 'float32',machineformat);
frames=fread(fid,1,'int32',machineformat);

%cwt=zeros(M,N); %Si reservamos espacio para cwt ganamos en
% velocidad. Hay que redefinir el modo en que se escriben los datos,
% en tal caso (línea  cwt=[cwt;data]);

tic
timein=0;
t1=tic;
%Barra de progreso.
str2=num2str(frames);%str2=num2str(frames);
str=strcat(' (',num2str(1),[blanks(1) 'of' blanks(1) str2],')');
id_waitbar = waitbar(0, strcat('Frame' ,str, '...'));

for i=1:frames-1,
    MM=fread(fid,1,'int32',machineformat);
    %read the real part.
    datar = fread(fid, [MM,N], 'float32',machineformat);
    %read the imag part.
    datai = fread(fid, [MM,N], 'float32',machineformat);
```

```
    data=complex(datar,datai);
    lframes=floor(MM/GAP)+1;

    maskt=zeros(3,lframes,MAXPARTIALS);
    mask=[mask maskt];

      % Intento de sacar el tracking como función externa. Comenta
      % desde AQUI
%
%     if i==1,
%         [mask,nop,ultimoframe1,ultimomodule1,ultimoscalogram1,...
%          ultimobmin1,ultimobmax1,ultimopico1,timedur,...
%          escalogramatotal]=tracking(GAP,Fs,i,lframes,data,mask);
%     else
%         [mask,nop,ultimoframe1,ultimomodule1,ultimoscalogram1,...
%          ultimobmin1,ultimobmax1,ultimopico1,timedur,...
%          escalogramatotal]=tracking(GAP,Fs,i,lframes,...
%
data,mask,ultimoframe1,ultimomodule1,ultimoscalogram1,...
%
ultimobmin1,ultimobmax1,ultimopico1,timedur,escalogramatotal);
%     end
%
%     % A AQUI

    %Comienza el tracking de parciales: refrescaremos
    %resultados cuando se barra un frame completo (4095 muestras).

    if i==1,
        lsup=lframes-1;
    else
        lsup=lframes;
    end

    for lf=1:lsup,
        %Leeremos los frames en trozos de 256 muestras.
        %Trabajaremos con 2 segmentos por paso.
        if (i==1),
            ini2=GAP*lf+1;
            fin2=GAP*(lf+1);
        else
            ini2=GAP*(lf-1)+1;
            fin2=GAP*lf;
        end

        %Inicio del tracking.
        if ((i==1)&&(lf==1)), %Arranque de la señal.
            littleframe1=data(GAP*(lf-1)+1:GAP*lf,:);
            %Obtenemos las variables clásicas de estudio para el
            %frame 1.
            module1=abs(littleframe1);
            scalogram1=sum(module1,1);
            [bmin1,bmax1,pico1]=varcalc(scalogram1);
            littleframe2=[];
        else
            clear littleframe1 module1 scalogram1 bmin1 bmax1 pico1;
            %Obtenemos las variables clásicas de estudio para el
            %frame 1
            %copiando la informacion del frame 2 antes de borrarla.
            littleframe1=littleframe2;
            module1=module2;
```

```
    scalogram1=scalogram2;
    bmin1=bmin2;
    bmax1=bmax2;
    pico1=pico2;
end
if (lf==lsup),
    clear littleframe2;
    littleframe2=data(ini2:end,:);
else
    clear littleframe2;
    littleframe2=data(ini2:fin2,:);
end


%Variable que marca la duración en muestras de cada
%miniframe. Se empleará para reconstruir los parciales
%después del tracking.
if ((i==1)&&(lf==1)),
    timedur=size(littleframe1,1);
    timedur=[timedur size(littleframe2,1)];
else
    timedur=[timedur size(littleframe2,1)];
end

%Obtenemos las variables clásicas de estudio para el frame
%2.
clear module2 scalogram2 bmin2 bmax2 pico2;
module2=abs(littleframe2);
scalogram2=sum(module2,1);
[bmin2,bmax2,pico2]=varcalc(scalogram2);

%Obtenemos los límites en banda y la energía para cada uno
%de los parciales del miniframe 2.
[bandaminima2,posicion2,bandamaxima2,energia2]...
    =bandmasker(scalogram2,pico2,bmin2,bmax2);

%Calculamos el escalograma acumulado y la energia
%total acumulada de la señal.
if ((i==1)&&(lf==1)), %Arranque de la señal.
    escalogramatotalframe=scalogram1+scalogram2;
    escalogramatotal=scalogram1+scalogram2;
elseif (lf==1),
    escalogramatotalframe=scalogram2;
    %Energía de la señal en el frame actual.
    energiatotalframe=sum(escalogramatotalframe);
    escalogramatotal=escalogramatotal+scalogram2;
else
    %Escalograma de la señal en el frame actual.
    escalogramatotalframe=escalogramatotalframe+scalogram2;
    %Energía de la señal en el frame actual.
    energiatotalframe=sum(escalogramatotalframe);
    %Escalograma de la señal acumulado.
    escalogramatotal=escalogramatotal+scalogram2;
end
%Energía de la señal acumulada.
energiatotal=sum(escalogramatotal);

%Iniciamos las variables de la máscara (etiqueta,
%energía,inicio)para el primer miniframe de la señal.
if ((i==1)&&(lf==1)), %Arranque de la señal.
    maskini=iniciador(scalogram1,pico1,bmin1,bmax1);
```

```
            mask(:,1:3,1:length(pico1))=maskini;
        end

        %Buscamos los picos del escalograma 2 que siguen a los picos
        %del escalograma 1.
        salida=buscawinner(Fs,pico1,bmin1,bmax1,littleframe1,...
            pico2,bmin2,bmax2,littleframe2,1);

        %Determinamos los parciales nuevos, muertos y activos en el
            %miniframe 2 y sacamos las máscaras actualizadas.
        [mask,nop]=seleccion5(salida,mask,bandaminima2,posicion2,...
            bandamaxima2,energia2,i,lf);

        %Fin del tracking del frame
    end

    % Fusionado de parciales de baja energía y corta duración
    % con parciales importantes. Descomentar para Fx.
    [mask]=funde_parciales(mask);

%     % Extracción de datos importantes del frame. Comentar para Fx.
%
[parcialframe,ifrequencyframe,imoduleframe,notpframe,energiasord_fra
me]...
%     =extractor2(TAMLF,GAP,mask,data,i,frames,Fs);

%     %Análisis armónico. Comentar para Fx.
%     [n_fuentes,cjto_fuente,frecs_fuente,fuente_sep]=...
%     analisis_armonico(notpframe,energiasord_frame,parcialframe,...
%
ifrequencyframe,imoduleframe,Fs,umbral_modulo,umbral_frecuencia);

    % Update progress bar
    str=strcat(' (',num2str(i),[blanks(1) 'of' blanks(1) str2],')');
    waitbar(i/frames,id_waitbar,strcat('Frame' ,str, '...'));

%     timedursignal=sum(timedur); %Muestras de duración de la señal
(acumulado).
%     timedurframe=size(data,1); %Tamaño del frame actual
(muestras).
%
%     %Tomamos los datos correspondientes a las máscaras del frame
%     %en el que estemos trabajando. Sólo máscaras no nulas.
%     if (i<frames),
%         masksframe=mask(:,2+(i-
1)*TAMLF+1:2+i*TAMLF,mask(1,1,:)>0);
%     else
%         masksframe=mask(:,2+(i-1)*TAMLF+1:end,mask(1,1,:)>0);
%     end

    cwt=[cwt;data]; % Descomentar para Fx.
end
close(id_waitbar);

% Variables globales de la señal. Descomentar para Fx.
[notp,parcial,ifrequency,imodule,cwtout,signalout,energiasordenadas]
=...
    muestra_todo(mask,cwt,timedur,Fs);

% Separación de fuentes: análisis armónico global y sin separar
    % superpuestos. Comentar para Fx.
```

```
[num_fuentes,conjunto_fuente,frecuencias_fuente,fundamental,...

ifrequencymedian]=analisis_armonico(notp,energiasordenadas,parcial,.
..
    ifrequency,imodule,0.05,0.10);
%
% % Los parámetros p1 y p2 controlan la importancia relativa de
% % información frecuencial y modular, respectivamente. Deben sumar
% %1.
% p1=0.9;
% p2=0.1;
% [fuente_compleja,envolventes]=execute_separation(notp,parcial,...
%
ifrequency,imodule,Fs,0.05,num_fuentes,conjunto_fuente,fundamental,.
..
%     ifrequencymedian,p1,p2);
% % % Para escuchar las fuentes, descomentar esto:
%timedursignal=length(ifrequency);
%for xxx=1:num_fuentes,
%     soundsc(real(fuente_compleja(:,xxx)),Fs);
%     pause ((timedursignal/Fs)+1);
% end

% %Dibujamos el escalograma y el análisis armónico
% fundam=ifrequencymedian(fundamental(1));
% loco=find(abs(freqs-fundam)==min(abs(freqs-fundam)));
% n=(2:1:20);
% arms=fundam.*n;
% escalogram=sum(abs(cwt),1);
% figure;
% plot(freqs,20*log10(scalogram))
% hold on
% plot(fundam,20*log10(scalogram(loco)),'pr')
% plot(arms,20*log10(scalogram(loco)),'ok')
% axis tight;

fundam=ifrequencymedian(fundamental(1));


%%%%Añadido por Óscar Gil para guardar la variable "parcial" de cada
%%%%sonido en un archivo para su uso en los efectos de morphing.
parcialx=parcial;
nombre=strcat(file_name(1:end-4),'_parcial');%para escribir en un
%archivo la matriz parcial.
save(nombre,'parcialx');
%%%%%%%%%%%%lo que hacemos es guardar parcial en parcialx.

%%%%%Añadido por Óscar Gil para guardar los escalogramas de las
%%%%% notas de un instrumento y poder interpolar luego.
escalogramatotalx=escalogramatotal;
nombre=strcat(file_name(1:end-4),'_escalograma');%para escribir en
% un archivo el espectrograma.
save(nombre,'escalogramatotalx');
%%%%%%%%%%%%guardamos el escalograma en escalogramatotalx



telapsed=toc(t1);
%view_cwt_cas(signalout,cwtout,freqs,Fs,100);
time = [0:1/Fs:(length(signal)-1)/Fs]';
```

```
soundsc(signalout,Fs)
strsout=strcat(file_name(1:end-4),'_synthetic','.wav');
wavwrite(signalout,Fs,strsout);
```

2. Slapback-echo

```
%%Efecto para producir eco y slapback.

%%%Se basa en la estructura FIR por lo que se va a producir
%%%únicamente una repetición.
mode=input('Para producir un Slapback pulse 1; Para producir un Echo
pulse 2              ');
if mode==1
    tiempo_delay=input('introduce el tiempo de slapback(entre 10 y
25 ms) en ms              ') ;
elseif mode==2
    tiempo_delay=input('Introduce el tiempo de Echo(mayor de 50ms)
en ms                ');
else
    disp('ERROR');
end
tam=size(parcial);
delay=(tiempo_delay/1000)*Fs;
delay=round(delay);
g=1;%%%se puede cambiar al gusto
tono=1.01;%%%se puede cambiar al gusto
fase_delay=zeros(tam(1,1),tam(1,2));
mod_delay=zeros(tam(1,1),tam(1,2));
mod=abs(parcial);
fase=unwrap(angle(parcial));
tic
for n=1:tam(1,1)
    if n<=delay
        mod_delay(n,:)=0;
        fase_delay(n,:)=0;
    else
        mod_delay(n,:)=g*mod(n-delay,:);
        fase_delay(n,:)=tono*fase(n-delay,:);
    end

end
sig_delay=sum(mod_delay.*cos(fase_delay),2)+sum(mod.*cos(fase),2);
soundsc(sig_delay,Fs);
toc

%%%Primero lo he realizado con una copia exacta del Dafx, pero al
%%%tratarse de un algoritmo que va copiando de uno en uno, para un
%%%delay de 50ms le costaba 90 segundos de cálculo, mientras que con
%%%mi algoritmo le cuesta menos de un segundo.
```

3. Chorus

```
%%Efecto Chorus que trabaja con matriz parcial.

tam=size(parcial);
mod_f=abs(parcial);
fase_f=unwrap(angle(parcial));

md=ceil((25/1000)*Fs);%Empleamos 25 porque un chorus debe tener
                      %variaciones entre 10 y 25 ms.
z=zeros(md,tam(1,2));
m=max(abs(mod_f));
mod_delay=[mod_f;z];%zero padding al modulo original
mod=mod_delay;
mod=mod.*0;
fase_delay=[fase_f;z];
fase=fase_delay;
fase=fase.*0;
sig_delay=sum(mod_delay.*cos(fase_delay),2);
voces=2;%para establecer el número de repeticiones.

%%%%He observado que si aumentamos mucho el número de repeticiones,
%%%%el efecto pasa de parecer un chorus a convertirse en una especie
%%%%de reverb, por tanto para conseguir un chorus más eficaz es
%%%%recomendable utilizar pocas repeticiones, por ejemplo 1 o 2.
%b=zeros(voces,1);
%sig=0;

for i=1:voces
    b=round((1102-221)*rand+221);%1102 son los samples de delay que
%le corresponden para d=50ms y 221 para d=10ms con esta forma hago
%que el delay varíe aleatoriamente entre 10  y 50ms.
    z=zeros(b,tam(1,2));
    zi=zeros(md-b,tam(1,2));
    mod=[z;mod_f;zi];%zero padding a las voces

    %%para que cada fase no sea exactamente igual.
    tono=0.95+0.1*rand;
    fase=tono.*[z;fase_f;zi];


    %for j=1:tam(1,1)+md
    %    mod(j+md-b(i),:)=mod_delay(j,:);
    %
    %    fase(j+md-b(i),:)=fase_delay(j,:).*(1-i/50);%para que cada
    %voz no sea exactamente igual y de la sensación de ser distintas
    %voces.
    %end
    sig=sum(mod.*cos(fase),2);
    sig_delay=sig_delay+sig;

end

%%%%La ventaja de hacerlo así frente a hacerlo directo con la
%%%%salida es que podemos variar el tono de cada voz para que no
%%%%suene exactamente igual.

soundsc(sig_delay,Fs);
```

4. Vibrato

```
%%%Script del efecto vibrato

%%%Este vibrato es un vibrato basado en parte en lo realizado en el
%%%Dafx, pero un poco distinto.
%%%Lo que hace es cambiar la afinación oscilando ente el valor
%%%original y +- la senoidal creada con la longitud del sonido.
tic
tam=size(parcial);
frecuencia_vib=5;
amplitud=2;%%al igual que en el Dafx, para conseguir un efecto
           %%vibrato, lo ideal es movernos entre 5 y 14 hz.
m=frecuencia_vib/Fs;
senoidal=ones(tam(1,1),tam(1,2));
%%pruebas para generar una senoidal
for i=1:tam(1,1)
    senoidal(i,:)=amplitud*sin(m*2*pi*i);
%%%la amplitud marca cuantos tonos oscila el sonido respeto al
%%%original, a más altas amplitudes más se aleja del sonido original
end

m_vib=abs(parcial);
f_vib=(angle(parcial));
%m_vib=m_vib.*mod;
f_vib=f_vib+senoidal;

sig_vib=sum(m_vib.*cos(f_vib),2);

toc
soundsc(sig_vib,Fs);
```

5. Time stretching

```
%%Script del efecto time-stretching.

%Primero definimos el incremento o decremento temporal en %

incremento=4;%Define el número de veces que se hace más lento o más
             %rápido. Si es positivo se alarga el sonido, si es
             %negativo se acorta.
             %Si el incremento es negativo lo que se hace es
             %multiplicar la señal por 1/incremento.
s=size(parcial);
mod_t=abs(parcial);
tam=size(parcial);
fase_t=unwrap(angle(parcial));

der=[diff(fase_t);zeros(1,tam(1,2))];
% alargar o acortar
dur_original=length(signalout);%Sacamos la duración del vector
%signal_out, el sonido sintetizado por cwas.
if incremento<=0 %si es menor de cero se acorta el sonido
    incremento=abs(incremento);
    a=1;
    dur_nueva=(dur_original/incremento);%empleo ceil para redondear
%al entero más próximo ya que sino puede haber problemas con algunos
%números.
else
    incremento=abs(incremento);
    %alargar=incremento*dur_original/100;
    %dur_nueva=dur_original+alargar;
    dur_nueva=dur_original*incremento;
    a=0;
end
%Hasta aquí hemos sacado lo que debe durar la nueva señal y funciona
%correctamente.
factor=dur_nueva/dur_original;
%pasamos factor a un numero racional; factor=n1/n2;
[n1,n2]=rat(factor);

%interpolar por el numerador;
%reservamos el tamaño para la matriz
der1=zeros(dur_original*n1,s(1,2));
mod1=zeros(dur_original*n1,s(1,2));
for j=1:dur_original% en este caso es la duración de la mtz original
%es el número de filas de la matriz
    for i=1:n1% en este caso  n1 es el número de veces que se quiere
%repetir cada fila.
        der1(i+(j-1)*n1,:)=[der(j,:)];
    end
end
%%%he trabajado sobre las derivadas (las pendientes) ya que así al
%%%repetir los valores y luego "integrar' se comete menos error que
%%%si repito valores directamente.

%%%sobre la fase.....al hacerlo sobre la fase, se producía aliasing
%%%y salían frecuencias que en el original no estaban.
for j=1:dur_original% en este caso es la duración de la mtz original
% es el número de filas de la matriz.
    for i=1:n1% en este caso es el número de veces que se quiere
%repetir cada fila.
```

```
        mod1(i+(j-1)*n1,:)=[mod_t(j,:)];
    end
end
%La razón de interpolar por los mods y las fases por separado es
%que, si lo hace directamente sobre la mtz parcial, le cuesta mucho
%más tiempo de computación.

%interpolamos por el denominador
der2=der1(1:n2:dur_original*n1,:);
mod2=mod1(1:n2:dur_original*n1,:);
%pasamos del la derivada a la fase
fase2=zeros(dur_nueva,tam(1,2));
fase2(1,:)=fase_t(1,:);%damos el valor original al primer termino de
%la fase.

for i=1:(dur_nueva-1)
    fase2(i+1,:)=fase2(i,:)+der2(i,:);
end


if a==1
if rem(incremento,2)==0
fase_n=[fase2;zeros(1,tam(1,2))];
else
 fase_n=fase2;
end
else
 fase_n=fase2;
end
%Toda esta sección está hecha porque sino, al hacer el sonido más
%corto, había errores con algunos valores de "incremento".

mod_n=mod2;

signal_time=sum(mod_n.*cos(fase_n),2);
soundsc(signal_time,Fs);
```

6. Robotización

```
%%Efecto de robotización que consiste en crear una frecuencia fija
%%para el sonido.

tam=size(parcial);
m_robot=abs(parcial);

f_robot=unwrap(angle(parcial));
%%Robotizar una voz consiste en darle un tono fijo a una señal.
frecuencia=280;%frecuencia que queremos darle al sonido en Herzios.

%%Calculamos la fase a partir de la frecuencia.
fase=(2*pi*frecuencia)/(Fs);
fase2=zeros(tam(1,1),1);
for i=1:tam(1,1)
    fase2(i,1)=fase*(i);
end
for j=1:15%hasta 15 porque ahí esta la información más importante
    if j==1
    f_robot(:,j)=fase2(:,1);
    else
    f_robot(:,j)=fase2(:,1)*(j/2);
    end
end



sig_rob=sum(m_robot.*cos(f_robot),2);


soundsc(sig_rob,Fs);
```

7. Denoising

```
%%Script del efecto denoising que consiste en quitar el ruido a un
%%sonido.

mod_noise=abs(parcial);
fase_noise=(angle(parcial));
parcial_noise=parcial;
tam=size(parcial);

coef=0.01;

%como hay términos de los módulos que son=0, debemos tener esto en
%cuenta ya que sino nos pueden aparecer NaN al operar.
for j=1:tam(1,2)
    for i=1:tam(1,1)
        if mod_noise(i,j)==0
        mod_noise(i,j)=0;
        else

mod_noise(i,j)=(mod_noise(i,j)./(mod_noise(i,j)+coef)).*mod_noise(i,
j);
        end
    end
end


%%%Sometemos la señal a un filtrado que consiste en eliminar los
%%%parciales cuyas frecuencias medias son próximas a 50 y 60 Hz.

for i=1:tam(1,2)
    if (ifrequencymedian(i,1)>49)&&(ifrequencymedian(i,1)<51)
        mod_noise(:,i)=0;
    elseif (ifrequencymedian(i,1)>59)&&(ifrequencymedian(i,1)<61)
        mod_noise(:,i)=0;
    end
end


sig_noise=sum(mod_noise.*cos(fase_noise),2);
signal2=signalout-sig_noise;
soundsc(sig_noise,Fs);
soundsc(signal2,Fs);
```

8. Pitch shifting

```
%Script del efecto Pitch_shifting simple.

%Debido a que cambiar el tono de un sonido equivale a modificar
%su frecuencia instantánea, lo que vamos a hacer es multiplicar su
%fase por un factor de transposición.

%iphase para obtener la fase instantánea de cada parcial
%Cuando se sube una octava (12 semitonos) equivale a multiplicar la
%frecuencia fundamental de la nota por 2.
%Cada vez que se aumenta un semitono la nota original se multiplica
%la frecuencia de dicha nota por 1.0595 de forma sucesiva de manera
%que, por ejemplo, si queremos desplazar la nota 4 semitonos se
%tendrá que multiplicar la frecuencia por un ratio de 1.0595^4.


numero_semitonos= input(' introduce el numero de semitonos que
quieres desplazar la nota(en + o -) maximo 12');
%El motivo de poner máximo 12 semitonos es porque, aunque sigue
%funcionando si ponemos más de 12, se desnaturaliza mucho el sonido.

if numero_semitonos>0
    ratio=1.0595^numero_semitonos;
elseif numero_semitonos<0
    numero_semitonos=abs(numero_semitonos);
    ratio=1/(1.0595^numero_semitonos);
else
    ratio=1;
end
%Hasta aquí lo que hemos hecho es obtener una relación entre el
%ratio por el que vamos a multiplicar la fase (y por tanto la
%frecuencia) y los tonos/semitonos musicales que queremos subir o
%bajar.
imod_pichi=abs(parcial);
iphase_pi=unwrap(angle(parcial));
iphase_pichi=iphase_pi.*ratio;
signal_pichi=sum(imod_pichi.*cos(iphase_pichi),2);
soundsc(signal_pichi,Fs);
```

9. Pitch shifting mejorado

```
%%Script del Pitch shifting mejorado.

%%%Lo que intentamos hacer es sacar el escalograma de un instrumento
%%%del que conocemos diferentes notas, de manera que hacemos una
%%%interpolación para intentar sacar su comportamiento.

%%Lo primero de todo tenemos 7 muestras de sonido (hornC2,E2,...)de
%%las que vamos a cargar su espectograma en una matriz para hacer la
%%interpolación.
tam=size(parcial);
escalogramas=zeros(189,7);
escalogramafinal=zeros(11025,7);
load HornC2_escalograma;
escalogramas(:,1)=escalogramatotalx;
load HornE2_escalograma;
escalogramas(:,2)=escalogramatotalx;
load HornG2_escalograma;
escalogramas(:,3)=escalogramatotalx;
load HornC3_escalograma;
escalogramas(:,4)=escalogramatotalx;
load HornE3_escalograma;
escalogramas(:,5)=escalogramatotalx;
load HornG3_escalograma;
escalogramas(:,6)=escalogramatotalx;
load HornC4_escalograma;
escalogramas(:,7)=escalogramatotalx;

for i=1:7
    %%%%%-----------------------------------%%%%%%%%
    %%%%Pasamos el plot del escalograma a una variable para hacernos
    %%%%más sencillos los cálculos.
    x=freqs;
    y=escalogramas(:,i);
    xi=[1:11025];
    yi=interp1(x,y,xi);
    escalogramafinal(:,i)=yi';
    %%%%%-----------------------------------%%%%%%%%%%%%%%%%%%%
end
for j=1:22
    escalogramafinal(j,:)=0;%para evitar los NaN que salen ya que el
    %barrido del escalograma se realiza desde los 22.6hz hasta los
    %11025hz;
end
clear x xi y yi

interpolacion=zeros(11025,25);

for i=1:11025
    x=[1,5,8,13,17,20,25];%%%%dentro de los 25 semitonos que
    %componen 2 octavas conocemos el numero 1, 5, 8, 13...
    xi=[1:25];
    y=escalogramafinal(i,:);
    yi=interp1(x,y,xi);
    interpolacion(i,:)=yi;
end

%%nos quedan almacenadas en la matriz interpolacion las gráficas de
```

```
%%los espectogrmas estimados de cada nota, de manera que si queremos
%%representar el espectograma de una nota que esté 2 semitonos
%%sobre la fundamental(C2) tenemos que representar
%%interpolacion(:,3)

%%%una vez que tenemos el escalograma de cada nota de la escala
%%%almacenado en la matriz, nos interesa encontrar una relación
%%%entre la matriz parcial y el escalograma del sonido original.

imodulemedian=zeros(tam(1,2),1);
for i=1:tam(1,2)
    suma=sum(imodule(:,i));
    imodulemedian(i,1)=suma/tam(1,1);%de esta manera en
%imodulemedian nos queda una media de la amplitud de cada parcial.

    %Esta media nos será de utilidad para localizar los parciales
    %más significativos a partir del escalograma y aumentar los que
    %pasen a ser más significativos al cambiar de nota.
end

%%%Si sólo multiplicamos las fases por 1.095^(número de semitonos a
%%%desplazar), lo que hacemos es mover el escalograma a la derecha
%%%desplazándolo.

%%%Sin embargo, si miramos el escalograma interpolado de la nota que
%%%queremos generar, su escalograma no es exactamente el escalograma
%%%desplazado.

numero_semitonos=input(' introduce el numero de semitonos que
quieres desplazar la nota(en + o -)maximo 12');
numsemis2=numero_semitonos;

if numero_semitonos>0
    ratio=1.0595^numero_semitonos;
elseif numero_semitonos<0
    numero_semitonos=abs(numero_semitonos);
    ratio=1/(1.0595^numero_semitonos);
else
    ratio=1;
end
%Hasta aquí lo que hemos hecho es obtener una relación entre el
%ratio por el que vamos  a multiplicar la fase (y por tanto la
%frecuencia) y los tonos/semitonos musicales que queremos subir o
%bajar.

%Calculamos las nuevas frecuencias instantáneas y los nuevos módulos
%medios para retocar según nuestras necesidades.

frecuencias_pichi=ifrequencymedian;

%Ahora tenemos que localizar en el escalograma correspondiente los
%picos de cada frecuencia.

escalo_ini=interpolacion(:,13);
escalo_inter=interpolacion(:,13+numsemis2);

%%En el escalograma inicial y en el interpolado, buscamos todos los
%%picos de cierta importancia para quedarnos solo con ellos.
```

```
picos_ini=zeros(tam(1,2),2);
picos_inter=zeros(tam(1,2),2);
for i=1:tam(1,2)
    [maxi1,num1]=max(escalo_ini);
    [maxi2,num2]=max(escalo_inter);
    picos_ini(i,:)=[maxi1,num1];
    picos_inter(i,:)=[maxi2,num2];
    for j=1:100
        escalo_ini((num1-50)+j,1)=0;
        escalo_inter((num2-50)+j,1)=0;
    end
end%%%con esta estructura me voy quedando con cada uno de los
   %%%máximos que me interesan.
%%%%%%%_____________________________________________%%%%%%%%%%%%%%



%Ahora vuelvo a reconstruir escalo_ini pero sólo con los picos
%%realmente esto no sería necesario, pero lo hago para comprobar que
%%detecta los picos buenos.
escalo_ini=zeros(11025,1);
escalo_inter=zeros(11025,1);
for i=1:30%con los 30 primeros picos tenemos suficiente
    num1=picos_ini(i,2);
    num2=picos_inter(i,2);
    escalo_ini(num1,1)=picos_ini(i,1);
    escalo_inter(num2,1)=picos_inter(i,1);
end

%%%%==============================================
%%%%==============================================

%como tenemos el espectograma interpolado, simplemente vamos a mirar
%de trasladar los picos del espectograma a los módulos.

%primero multiplicamos las fases por el ratio.
fase_pichi=unwrap(angle(parcial));
modulo_pichi=abs(parcial);

fase_pichi=fase_pichi.*ratio;
frecuencias_pichi=frecuencias_pichi.*ratio;
frecuencias=zeros(30,1);
picos_inter2=zeros(30,2);
for i=1:30
    frecuencias(i,1)=frecuencias_pichi(i,1);
    picos_inter2(i,:)=picos_inter(i,:);
end
diferencias=zeros(tam(1,2),1);%hacemos este vector para encontrar
%para cada frecuencia de los picos del escalograma, la frecuencia
%real más próxima
logicas=zeros(tam(1,2),1);%%%para almacenar con un 1 los parciales
%que son retocados.
for i=1:30%por retocar solo los 30 más importantes

    frec=picos_inter2(i,2);
    diferencias=abs(frecuencias-frec);

    [mini,lugar]=min(diferencias);
    logicas(lugar,1)=1;
    ampli=picos_inter2(i,1)/34500;
```

```
    multi=ampli/imodulemedian(lugar,1);
    modulo_pichi(:,lugar)=modulo_pichi(:,lugar).*multi;
    %frecuencias(lugar,1)=1000000;


end
%%%Originalmente retocamos solo los 30 primeros picos que serán los
%%%más importantes, pero el resto no los retocamos, por lo que en
%%%casos de descensos o ascensos marcados, esto puede llevar a que
%%%se escuchen partes no deseadas con más volumen del que se
%%%deberían escuchar.

%%%por tanto, vamos a introducir una compensación para el resto de
%%%frecuencias que consiste en ampliar o atenuar dichas frecuencias
%%%en proporción al máximo pico de cada espectograma.

compensacion=max(escalo_ini)/max(escalo_inter);
for i=1:tam(1,2)
    if logicas(i,1)==0

        modulo_pichi(:,i)=modulo_pichi(:,i)./compensacion;

    end
end


senal_pichi=sum(modulo_pichi.*cos(fase_pichi),2);
soundsc(senal_pichi,Fs);




%%%%Funciona bien, pero siempre van a aparecer problemas al sacar
%%%%una nota de su tono natural que vienen de que la ejecución de
%%%%una nota real una octava arriba y una octava abajo no son
%%%%exactamente iguales, al subir una nota de tono, se hacen
%%%%perceptibles defectos del sonido que en la nota original no se
%%%%aprecian.
```

10. Sustain

```
%%Script para encontrar y modificar la región sustain del sonido.

%%%Nos vamos a centrar en encontrar tramos estables.
%%%Lo primero es que vamos a tratar de buscar en cada parcial un
%%%tramo de una duración de 1000 samples que varía muy poco
.
%%%Lo buscamos sobre los módulos de cada parcial.


tam=size(parcial);
mod_adsr=abs(parcial);
fase_adsr=unwrap(angle(parcial));

%%%creamos una matriz con tam(1,2)(nº parciales) filas y 2 columnas
%%%que corresponderán al inicio y fin de cada tramo estable.

regiones=zeros(tam(1,2),2);

%%%ahora tenemos que encontrar para cada parcial, el tramo de 1000
%%%samples que tiene una menor variación.
%%%para ello tomamos como referencia el valor máximo de cada
%%%parcial.

region_lo=logical(regiones);
maximos=zeros(tam(1,2),1);
for i=1:tam(1,2)
    maximos(i,1)=max(mod_adsr(:,i));
end
diferencia=zeros(1,1);

%variamos para que no de problema con el clarinete: pongo el valor
%50 que consume más tiempo de cálculo, pero aseguramos que no queda
%ninguna región sin detectar.

for contador=1:0.01:50%%%con este método realizo varias pasadas para
%%% que las regiones que no detecte a la primera las detecte en
%%%sucesivas pasadas.
for j=1:tam(1,2)
if region_lo(j,1)==0
for i=tam(1,1):-1:1001

    diferencia=abs(mod_adsr(i,j)-mod_adsr(i-1000,j));

    if diferencia < ((maximos(j,1)/1000)*contador)

        diferencia=abs(mod_adsr(i,j)-mod_adsr(i-500,j));
            if diferencia < ((maximos(j,1)/1000)*contador)
%si se cumple en el tramo inicial y final y en el punto medio, se
%guardan los valores.
            regiones(j,1)=i-1000;
            regiones(j,2)=i;
            end

    end
end
end
region_lo=logical(regiones);
```

```
end
end

%%%Una vez que tenemos almacenado en "regiones" cada sector más
%%%estable de cada parcial, procedemos a repetir "n" veces la
%%%duración.

%Ahora lo que vamos a hacer es almacenar en una matriz de
%1000x"numero %de parciales" todos los sectores estables.

estables=zeros(1001,tam(1,2));
estables_fase=zeros(1001,tam(1,2));
derivada_fase=zeros(1001,tam(1,2));
for i=1:tam(1,2)
    for j=1:1001
        estables(j,i)=mod_adsr(regiones(i,1)-1+j,i);
        estables_fase(j,i)=fase_adsr(regiones(i,1)-1+j,i);

    end
end
%nos queda almacenado en estables cada región estable de cada
%parcial.


%%%Para las fases nos resulta más cómodo y con una transición mucho
%%%más suave, trabajar con las derivadas, por tanto, vamos a hacer
%%%las derivadas de la matriz que acabamos de hacer.
derivada_fase=[zeros(1,tam(1,2));diff(estables_fase)];

%Ahora simplemente debemos pegar esas regiones repetidas veces.
%Antes vamos a hacer una matriz con el tamaño final y con la
%secciones repetidas en ella.
repeticiones=input('duracion de la señal');

sustain=zeros(repeticiones*1001,tam(1,2));
sustain_derivada=zeros(repeticiones*1001,tam(1,2));
sustain_fase=zeros(repeticiones*1001,tam(1,2));
for i=1:tam(1,2)
    for k=1:repeticiones
        for j=1:1001
        sustain(j+1001*(k-1),i)=estables(j,i);
        sustain_derivada(j+1001*(k-1),i)=derivada_fase(j,i);
%%%de esta forma tenemos un pequeño problema, que nos aparecen
%%%valores con cero debido al primer cero de la mtz.
        end
    end
end


%%%Como nos aparecen ceros indeseados en  la matriz sustain fase,
%%%vamos a eliminarlos.

for j=1:tam(1,2)
    for i=1:repeticiones*(1001)-1
        if sustain_derivada(i,j)==0
            sustain_derivada(i,j)=sustain_derivada(i+1,j);
        end
    end
end
```

```
%tenemos que tener en cuenta que al pasar de derivada a fase habrá
%que sumar desde la segunda fila.

%%%Ahora pasamos sustain derivada a sustain_fase.
for j=1:tam(1,2)
    for i=1:repeticiones*(1001)
        if i==1
        sustain_fase(i,j)=estables_fase(1,j);
        else
        sustain_fase(i,j)=sustain_fase(i-1,j)+sustain_derivada(i,j);
        end
    end
end
%ya tenemos la fase en sustain_fase, ahora realizamos lo mismo que
%con los módulos.

%añadimos la matriz sustain a los módulos.

%primero haremos una matriz con el tamaño necesario.

compensacion=zeros(1,tam(1,2));
for i=1:tam(1,2)
    compensacion(1,i)=sustain_fase(repeticiones*1001,i)-
fase_adsr(regiones(i,2),i);
end
modulo_sustain=zeros(tam(1,1)+(repeticiones-1)*1001,tam(1,2));
fase_sustain=zeros(tam(1,1)+(repeticiones-1)*1001,tam(1,2));

for i=1:tam(1,2)
    for j=1:regiones(i,1)
        modulo_sustain(j,i)=mod_adsr(j,i);
        fase_sustain(j,i)=fase_adsr(j,i);
    end
    for j=regiones(i,1):regiones(i,1)+(repeticiones*1001)-1
        modulo_sustain(j,i)=sustain(j+1-regiones(i,1),i);
        fase_sustain(j,i)=sustain_fase(j+1-regiones(i,1),i);
    end

    for j=regiones(i,1)+(repeticiones*1001):tam(1,1)+(repeticiones-
1)*1001
        modulo_sustain(j,i)=mod_adsr(j-(repeticiones-1)*1001,i);

%Hay un pequeño problema con el último tramo, necesito la diferencia
%de lo que ha variado la fase y sumárselo al último tramo----->lo
%he hecho en el vector "compensación".

        fase_sustain(j,i)=fase_adsr(j-(repeticiones-
1)*1001,i)+compensacion(1,i);
    end
end

%Funciona perfectamente y al realizar el plot de la suma de todos
%los módulos, se aprecia un resultado muy suave.....a simple vista
%con los módulos no hay que realizar ningún suavizado extra.

senal=sum(modulo_sustain.*cos(fase_sustain),2);
soundsc(senal,Fs);
```

11. Vocoder simple

```
%%%%%Efecto Vocoder basado en morphing simple que consiste en
%%%%%realizar un vocoder (controlar con los parámetros de un sonido
%%%%%la ejecución de otro).


load redwheel_parcial

%%%Para cargar la matriz parcial de la voz que se utiliza (en este
%%%caso el archivo redwheel).


tam=size(parcial);

tam2=size(parcialx);
%%%queremos saber qué matriz tiene mayor número de parciales para
%%%darle las dimensiones en columnas de la mayor.
if tam2(1,2)<=tam(1,2)
    tam(1,2)=tam(1,2);
else
    tam(1,2)=tam2(1,2);
end

if tam2(1,1)<=tam(1,1)
%%la función del bucle if es mirar cuál de las 2 señales tiene mayor
%%duración y adaptar una a la otra
parcial2=zeros(tam(1,1),tam(1,2));
%%reservamos una matriz de ceros del tamaño de parcial para colocar
%%dentro de ella parcial redwheel
    for i=1:tam2(1,1)
        for j=1:tam2(1,2)
            parcial2(i,j)=parcialx(i,j);
        end
    end
else
parcial3=zeros(tam2(1,1),tam(1,2));
%%reservamos una matriz de ceros del tamaño de parcial_redwheel para
%%colocar dentro de ella parcial.
    for i=1:tam(1,1)
        for j=1:tam(1,2)
            parcial3(i,j)=parcial(i,j);
        end
    end
parcial=parcial3;
parcial2=parcialx;
end
%%%Con esto metemos "parcial_redwheel" en la matriz de ceros de
%%%tamaño "parcial" o si la señal de control tiene mayor duración,
%%%metemos parcial en una matriz de ceros de tamañoo
%%%parcial_redwheel.
mod_vocoder=abs(parcial2);

env_vocoder=sum(mod_vocoder,2);
mod=abs(parcial);
tam=size(parcial);
%de esta manera se refrescan los valores de tam para %el último
%paso.

maxis_voc=max(env_vocoder);
```

```
%con esto vamos a hacer que el máximo de la amplitud sea uno con el
%objetivo de multiplicar la amplitud mod vocoder por mod.

%%teniendo los máximos de la señal original y de la señal con la que
%%queremos modular dicha señal, lo que hacemos es meter la señal de
%%control en ese rango.
compensar=1./maxis_voc;
env_vocoder=env_vocoder.*compensar;
%conseguimos que el máximo sea 1 y así multiplicamos por la amplitud
%de la señal original.
fase_vocoder=unwrap(angle(parcial));
for i=1:tam(1,2)
    mod(:,i)=mod(:,i).*env_vocoder(:,1);
end

sig_vocoder=sum(mod.*cos(fase_vocoder),2);
soundsc(sig_vocoder,Fs);


%%%%Finalmente lo que hemos realizado es un sumatorio de todas las
%%%%amplitudes de redwheel y luego comprimirlas a uno para
%%%%multiplicar por la propia amplitud de la señal original.
%%%%De esta manera lo que hacemos es modular la amplitud del sonido
%%%%original mediante la amplitud del sonido que usamos para
%%%%controlar, sin embargo, no se articulan las palabras del sonido
%%%%de voz (en el caso de las pruebas hemos usado el sonido
%%%%redwheel).
```

12. Vocoder mejorado

```
%%%%%Este efecto está basado en el vocoder simple pero con la
%%%%%diferencia de que en este queremos que se entiendan las
%%%%% palabras.

load redwheel_parcial

%%%Para cargar la matriz parcial de la voz que se utiliza (en este
%%%caso el archivo redwheel).


tam=size(parcial);
n_parciales=tam(1,2);%número de parciales del sonido a controlar
tam2=size(parcialx);
%%%queremos saber qué matriz tiene mayor número de parciales para
%%%darle las dimensiones en columnas de la mayor.

if tam2(1,2)<=tam(1,2)
    tam(1,2)=tam(1,2);
else
    tam(1,2)=tam2(1,2);
end

if tam2(1,1)<=tam(1,1)%%la función del bucle if es mirar cual de las
%%2 señales tiene mayor duración y adaptar una a la otra.
parcial2=zeros(tam(1,1),tam(1,2));%%reservamos una matriz de ceros
%%del tamaño de parcial para colocar dentro de ella parcial redwheel
    for i=1:tam2(1,1)
        for j=1:tam2(1,2)
            parcial2(i,j)=parcialx(i,j);
        end
    end
    parcial3=zeros(tam(1,1),tam(1,2));
        for i=1:tam(1,1)
            for j=1:n_parciales
                parcial3(i,j)=parcial(i,j);
            end
        end
    parcial=parcial3;
else
parcial3=zeros(tam2(1,1),tam(1,2));%%reservamos una matriz de ceros
%%del tamaño de parcial_redwheel para colocar dentro de ella parcial

    for i=1:tam(1,1)
        for j=1:tam(1,2)
            parcial3(i,j)=parcial(i,j);
        end
    end

parcial=parcial3;
parcial2=zeros(tam2(1,1),tam(1,2));

    for i=1:tam2(1,1)
        for j=1:tam2(1,2)
            parcial2(i,j)=parcialx(i,j);
        end
    end
```

```
end
%%%Con esto metemos "parcial_redwheel" en la matriz de ceros de
%%%tamaño "parcial" o si la señal de control tiene mayor duración,
%%%metemos parcial en una matriz de ceros de tamaño
%%%parcial_redwheel.

%%%%En este efecto vocoder mejorado, tenemos que formar un sonido
%%%%nuevo directamente con la amplitud de un sonido y la fase del
%%%%otro.

%%%%Los problemas que nos vamos a encontrar son que el número de
%%%%parciales de ambos sonidos no tienen porque coincidir.
%%%%Por tanto, vamos a desarrollar un método para no perder
%%%%información dando prioridad al número de parciales de la fase.

mod_vocoder=abs(parcial2);

mod_sonido=abs(parcial);

%%%%_____________________original
mod_suma1=sum(mod_vocoder,2);
maxi1=max(mod_suma1);%sacamos los máximos de cada suma para adaptar
%la señal de la voz en amplitud a la de la señal original.
mod_suma2=sum(mod_sonido,2);
maxi2=max(mod_suma2);
comp=maxi2/maxi1;%para realizar esa compensación que hemos dicho
%antes.
mod_vocoder=comp.*mod_vocoder;
%%%%%%_____________________________



fase_vocoder=unwrap(angle(parcial));

sig_vocoder=sum(mod_vocoder.*cos(fase_vocoder),2);

soundsc(sig_vocoder,Fs);

%Funciona perfectamente, lo único que trabaja mejor sobre señales
%que sean una sola nota y donde el ruido no sea muy elevado, ya que
%sino aparece bastante ruido.
```

13. Morphing

```
%%%%%Este efecto esta basado en el morphing, lo que queremos hacer
%%%%%es mezclar dos sonidos diferentes combinándolos en distintos
%%%%%porcentajes.


%%%Lo primero que debemos realizar es adecuar la matriz parcialx a
%%%las dimensiones del sonido que queremos controlar.

%%%Por ahora, la forma de cargar el sonido de voz que va a modular
%%%la ejecución del sonido principal, es mediante el comando load
%%%xxxxxxxx_parcial

load Clarinet_note_parcial


tam=size(parcial);
n_parciales=tam(1,2);%número de parciales del sonido a controlar
tam2=size(parcialx);
%%%queremos saber qué matriz tiene mayor número de parciales para
%%%darle las dimensiones en columnas de la mayor.
if tam2(1,2)<=tam(1,2)
    tam(1,2)=tam(1,2);
else
    tam(1,2)=tam2(1,2);
end

if tam2(1,1)>=tam(1,1)%%la función del bucle if es mirar cual de las
%%2 señales tiene menor duración y adaptar el sonido más largo al
%%más corto.
parcial2=zeros(tam(1,1),tam(1,2));%%reservamos una matriz de ceros
%%del tamaño de parcial para colocar dentro de ella parcial redwheel
    for i=1:tam(1,1)
        for j=1:tam2(1,2)
            parcial2(i,j)=parcialx(i,j);
        end
    end
    parcial3=zeros(tam(1,1),tam(1,2));
    for k=1:n_parciales
        parcial3(:,k)=parcial(:,k);
    end
    parcial=parcial3;
else

parcial3=zeros(tam2(1,1),tam(1,2));%%reservamos una matriz de ceros
%%del tamaño de parcial_redwheel para colocar dentro de ella parcial
    for i=1:tam2(1,1)
        for j=1:tam(1,2)
            parcial3(i,j)=parcial(i,j);
        end
    end
parcial=parcial3;
parcial2=zeros(tam2(1,1),tam(1,2));
    for i=1:tam2(1,1)
        for j=1:tam2(1,2)
            parcial2(i,j)=parcialx(i,j);
        end
    end
```

```
end

%%%%Hasta aquí la parte de adaptar la longitud de las señales la una
%%%%a la otra, nos quedarán para trabajar las matrices parcial y
%%%%parcial2.
moda=abs(parcial);
fasea=unwrap(angle(parcial));
modb=abs(parcial2);
faseb=unwrap(angle(parcial2));

tamano=size(parcial);%el tamaño de las matrices una vez adaptadas la
%una a la otra.
samples=tamano(1,1);%para saber el número de samples.
columnas=tamano(1,2);

%z=zeros(samples,1);

mezcla=input('porcentaje de mezcla         ');
inversa=1-mezcla;
mod_m=mezcla.*modb+inversa.*moda;
fase_m=mezcla.*faseb+inversa.*fasea;

sig_m=sum(mod_m.*cos(fase_m),2);

soundsc(sig_m,Fs);

%%%%La versión final se hace mezclando los sonidos en porcentaje
%%%%como se explica en el DAFX.
```

14. Pseudorobot

```
%%Script que pretende robotizar la voz.

%Consiste en cambiar la fase del  archivo y darle el valor cero en
%algunas partes.
%Lo primero que vamos a hacer es obtener el módulo y fase del sonido
mod_r=abs(parcial);
fase_r=unwrap(angle(parcial));
%ahora dividimos fase en n1 partes
n1=167;
w1=hanningz(n1);%se va a emplear el hanningz porque en las pruebas
%realizadas se observa que si se realiza un cambio brusco en las
%amplitudes suena con mucho ruido y poco natural, sin embargo de
%esta manera el sonido aparece más progresivamente.
tam=length(parcial);
a=rem(tam,n1);
tam2=tam-a;
n2=tam2/n1;%con esto sacamos el número de partes completas de n1 en
           %tam

for i=1:n2
    if rem(i,2)==0
        for j=1:n1
            fase_r((i-1)*n1+j,:)=0;
        end
    else
        for j=1:n1
            fase_r((i-1)*n1+j,:)=1.3*fase_r((i-1)*n1+j,:);
%el valor 1.3 se puede sustituir por el que se desee, el objetivo es
%obtener distintos tonos de voz de robot.
        end
    end
end
%con esta parte lo que hago es una parte sí y otra no, darle el
%valor cero a la fase.
for i=1:n2
    if rem(i,2)==0
        for j=1:n1
            mod_r((i-1)*n1+j,:)=0;
        end
    else
        for j=1:n1
            mod_r((i-1)*n1+j,:)=mod_r((i-1)*n1+j,:)*w1(j);
        end
    end
end
%damos valor cero en las amplitudes también y multiplicamos por w1
%para que el sonido aparezca más progresivamente y sin ruido.

sig_robot=sum(mod_r.*cos(fase_r),2);
soundsc(sig_robot,Fs);
%wavwrite(sig_robot, Fs, 'xxxxx.wav');
```

A continuación se incluye el código de la función Hanningz a la que se llama para realizar el suavizado de los módulos:

```
function w = hanningz(n)
%HANNING HANNINGZ(N) returns the N-point Hanning window in a column
vector.

%   Copyright (c) 1984-94 by The MathWorks, Inc.
%       $Revision: 1.4 $  $Date: 1994/01/25 17:59:15 $

w = .5*(1 - cos(2*pi*(0:n-1)'/(n)));
```

15. Armonizador

```
%%%Script para obtener el efecto Armonizador. Al tratarse de un
%%%efecto directamente relacionado con el pitch_sifht simple, el
%%%primer bloque del script es igual que el de dicho efecto.

%iphase para obtener la fase instantánea de cada parcial
%Cuando se sube una octava (12 semitonos) equivale a multiplicar la
%frecuencia fundamental de la nota por 2.

%Cada vez que se aumenta un semitono la nota original se multiplica
%la frecuencia de dicha nota por 1.0595 de forma sucesiva de manera
%que por ejemplo si queremos desplazar la nota 4 semitonos se tendrá
%que multiplicar la frecuencia por un ratio de 1.0595^4

volumen=0.3;% se puede modificar al gusto del usuario.

numero_semitonos=7;

if numero_semitonos>0
    ratio=1.0595^numero_semitonos;
elseif numero_semitonos<0
    numero_semitonos=abs(numero_semitonos);
    ratio=1/(1.0595^numero_semitonos);
else
    ratio=1;
end
%Hasta aquí lo que hemos hecho es obtener una relación entre el
%ratio por el que vamos a multiplicar la fase (y por tanto la
%frecuencia) y los tonos/semitonos musicales que queremos subir o
%bajar.

imod_pichi=abs(parcial);
iphase_pi=unwrap(angle(parcial));
iphase_pichi=iphase_pi.*ratio;
signal_pichi=sum(volumen.*imod_pichi.*cos(iphase_pichi),2)+
sum(imod_pichi.*cos(iphase_pi),2);
soundsc(signal_pichi,Fs);

%soundsc(signalout,Fs);%para comprobar el sonido procedente del
%pitch con el original.
```

16. Reverb

```
%%%Vamos a tratar de hacer una reverb room basada en el retardo que
%%%tiene el sonido en llegar a las 4 paredes de una habitación.

%%%Consideraremos para ello que la velocidad del sonido es de 340m/s
%%%y que estamos en una habitación de dimensiones 50x50m (se puede
%%%modificar al gusto del usuario cambiando el parámetro dim).

%%%%Una de las primeras cosas a considerar es la posición de la
%%%%fuente sonara dentro de la habitación

%%La definiremos mediante x e y;
%====================inicializamos==============================
dim=50;%dimensiones de la sala de 50x50m
x=10;%respecto al eje
y=15;

%%%Consideramos 4 ondas sonoras (una por cada pared) que se sumarán
%%%a la onda original de manera que tendremos que tener en cuenta
%%%la ida y vuelta del sonido.

%%A su vez el factor g deberá ser proporcional a la longitud
%%recorrida por cada onda

l1=(dim-x)*2;%longitud que recorre la onda 1.
l2=x*2;
l3=(dim-y)*2;
l4=y*2;
%%%pasamos las longitudes a un vector
lon=[l1;l2;l3;l4];


%%%%relacionamos la atenuación con la longitud
g=1-lon/100;
%%%%calculamos el tiempo en retardo de cada onda
t=lon/340;%tiempo en segundos

%%pasamos el tiempo a numero de samples
s=t*Fs;
%%%redondeamos al entero más próximo
s=round(s);
maxi=max(s);

tam=size(parcial);
mod=abs(parcial);
fase=unwrap(angle(parcial));

nueva_dur=tam(1,1)+maxi;%indica la nueva duración en samples del
%sonido eligiendo el delay máximo para no tener problemas.



%=================comienza el script============================


blanco=zeros(maxi,tam(1,2));%espacio en blanco a dejar para que se
%repita la señal.
```

```
mod_delay1=[mod;blanco];
fase_delay1=[fase;blanco];
mod_delay2=g(1,1).*[zeros(s(1,1),tam(1,2));mod;zeros(maxi-
s(1,1),tam(1,2))];
mod_delay3=g(2,1).*[zeros(s(2,1),tam(1,2));mod;zeros(maxi-
s(2,1),tam(1,2))];
mod_delay4=g(3,1).*[zeros(s(3,1),tam(1,2));mod;zeros(maxi-
s(3,1),tam(1,2))];
mod_delay5=g(4,1).*[zeros(s(4,1),tam(1,2));mod;zeros(maxi-
s(4,1),tam(1,2))];
fase_delay2=[zeros(s(1,1),tam(1,2));1.*fase;zeros(maxi-
s(1,1),tam(1,2))];
fase_delay3=[zeros(s(2,1),tam(1,2));1.*fase;zeros(maxi-
s(2,1),tam(1,2))];
fase_delay4=[zeros(s(3,1),tam(1,2));1.*fase;zeros(maxi-
s(3,1),tam(1,2))];
fase_delay5=[zeros(s(4,1),tam(1,2));1.*fase;zeros(maxi-
s(4,1),tam(1,2))];

sig_delay1=sum(mod_delay1.*cos(fase_delay1),2);
sig_delay2=sum(mod_delay2.*cos(fase_delay2),2);
sig_delay3=sum(mod_delay3.*cos(fase_delay3),2);
sig_delay4=sum(mod_delay4.*cos(fase_delay4),2);
sig_delay5=sum(mod_delay5.*cos(fase_delay5),2);

sig_delay_out=sig_delay1+sig_delay2+sig_delay3+sig_delay4+sig_delay5
;

soundsc(sig_delay_out,Fs);


%%%Al trabajar sobre los módulos y fases, se pueden introducir
%%%cambios de tono como se ha visto en otros efectos sin más que
%%%multiplicar las fases por un parámetro.
```

17. Trémolo

```
%%Script del efecto de trémolo.

%%Generamos una senoidal y multiplicamos por la amplitud de los
%%parciales.

%%%Generamos la senoidal con longitud la duración del sonido.

tam=size(parcial);
amplitud=0.5;
frecuencia_seno=7;%frecuencia en herzios. Podemos variar la
%frecuencia al gusto para variar la velocidad del efecto.
onda=zeros(tam(1,1),1);
for n=1:tam(1,1)
onda(n,1)=amplitud*sin(2*pi*frecuencia_seno*n/Fs)+0.5;
end
mod_t=abs(parcial);
fase_t=unwrap(angle(parcial));
for i=1:tam(1,1)
mod_t(i,:)=mod_t(i,:).*onda(i,1);
end
s_tremolo=sum(mod_t.*cos(fase_t),2);
soundsc(s_tremolo,Fs);

%%%%como curiosidad, destacar que mientras hacemos el efecto a una
%%%%frecuencia entre 1 y 10 hz se produce el efecto deseado,
%%%%mientras que si aumentamos la frecuencia, se produce un cambio
%%%%en el tono del sonido original.
```

18. Overdrive

```
%%Script del efecto overdrive con simulación valvular.

%%%%Tratamos de conseguir el efecto overdrive  y además aumentamos
%%%%la amplitud de los armónicos 2,3 y 5 para conseguir un efecto
%%%%más "musical" y similar al que se obtiene al aplicar estos
%%%%efectos con un amplificador a válvulas.

tam=size(parcial);
fase_o=(angle(parcial));
mod_o=abs(parcial);
gain1=3;%%%para ampliar los armónicos 2,3,5

%%%como sabemos mediante fundam(que se obtiene del Script
%%%CWAS_inicio) cuál es la frecuencia fundamental, vamos a buscar
%%%los parciales que corresponden los armónicos 2,3 y 5

f_fundamental=fundam;
contador=0;
for i=1:tam(1,2)
    %if ifrequencymedian(i,1)==f_fundamental
    %   contador=contador+1;
    %   armonicos(contador)=i;
    %end %tras varias pruebas he decidido que el primer parcial es
        %mejor no aumentarlo más.
        if ifrequencymedian(i,1)>=f_fundamental*2
        if ifrequencymedian(i,1)<(f_fundamental*2+10)
            contador=contador+1;
            armonicos(contador)=i;
        end
    end
    if ifrequencymedian(i,1)>=f_fundamental*3
        if ifrequencymedian(i,1)<(f_fundamental*3+10)
            contador=contador+1;
            armonicos(contador)=i;
        end
    end
    if ifrequencymedian(i,1)>=f_fundamental*5
        if ifrequencymedian(i,1)<(f_fundamental*5+10)
            contador=contador+1;
            armonicos(contador)=i;
        end
    end
end%%%una vez realizado esto, hemos encontrado los parciales que
   %%%corresponden a los armónicos 1, 2, 3 y 5

for j=1:length(armonicos)
    numero=armonicos(j);
    mod_o(:,numero)=mod_o(:,numero).*gain1;
end
gain=0.5;%%%para un efecto de overdrive, es mejor una ganancia
%%%máximo de 1 a 5 ya que sino la distorsión comienza a ser
%%%excesiva.

%%%0.5 para tener exactamente el mismo nivel de ganancia que en el
%%%DAfx.
signalover=sum(mod_o.*cos(fase_o),2);
maxi=max(signalover);
```

```
aumento=1./maxi;%%para hacer que la señal este en el rango de 0 a 1

x=gain*signalover*aumento;



N=tam(1,1);
th=1/3;% th es el término que se emplea para separar los tramos de
       %la señal


%%%%Esta parte está adaptada directamente del Script del DAFX, no
%%%%hay cambios.
for i=1:1:N,
   if abs(x(i))< th, y(i)=2*x(i);end;
   if abs(x(i))>=th,
      if x(i)> 0, y(i)=(3-(2-x(i)*3).^2)/3; end;
      if x(i)< 0, y(i)=-(3-(2-abs(x(i))*3).^2)/3; end;
  end;
   if abs(x(i))>2*th,
      if x(i)> 0, y(i)=1;end;
      if x(i)< 0, y(i)=-1;end;
   end;
end;

soundsc(y,Fs);

%soundsc(signalout,Fs);
```

19. Distorsión

```
%%Script del efecto distorsión con simulación valvular.

%%Vamos a utilizar la no linealidad que nos dan en el Dafx.
tam=size(parcial);
fase_o=(angle(parcial));
mod_o=abs(parcial);

gain1=3;%%%para ampliar los armónicos 2,3,5


f_fundamental=fundam;
contador=0;
for i=1:tam(1,2)
    %if ifrequencymedian(i,1)==f_fundamental
    %   contador=contador+1;
    %   armonicos(contador)=i;
    %end%%%tras varias pruebas he decidido que el 1er armónico no es
        %%%necesario realzarlo, ya esta bastante realzado por
        %%%naturaleza.
        if ifrequencymedian(i,1)>=f_fundamental*2
        if ifrequencymedian(i,1)<(f_fundamental*2+10)
            contador=contador+1;
            armonicos(contador)=i;
        end
    end
    if ifrequencymedian(i,1)>=f_fundamental*3
        if ifrequencymedian(i,1)<(f_fundamental*3+10)
            contador=contador+1;
            armonicos(contador)=i;
        end
    end
    if ifrequencymedian(i,1)>=f_fundamental*5
        if ifrequencymedian(i,1)<(f_fundamental*5+10)
            contador=contador+1;
            armonicos(contador)=i;
        end
    end
end%%%una vez realizado esto, hemos encontrado los parciales que
   %%%corresponden a los armónicos 1, 2, 3 y 5

for j=1:length(armonicos)
    numero=armonicos(j);
    mod_o(:,numero)=mod_o(:,numero).*gain1;
end
gain=4;
x=sum(mod_o.*cos(fase_o),2);%señal de salida

q=x*gain/max(abs(x));
z=sign(-q).*(1-exp(sign(-q).*q));
y=z*max(abs(x))/max(abs(z));
y=y*max(abs(x))/max(abs(y));
%soundsc(x,Fs);
soundsc(y,Fs);
```