Práctica 3. Filtrado por bloques.

1. Desarrollo de la práctica.

Se han de implementar los dos algoritmos de filtrado por bloques vistos en teoría (overlap-add y overlap-save) sujetos a las siguientes indicaciones:

- La implementación ha de ser lo más general posible.
- Los algoritmos deberán depender únicamente de la longitud de la DFT que se quiere utilizar. La longitud de los bloques y el solape entre ellos se deberán calcular según este parámetro de entrada.
- Se deberá tener en cuenta, evidentemente, la longitud del vector de señal y de la respuesta impulsiva del filtro.
- Aunque se pueden utilizar bucles (for o while) es altamente aconsejable implementar los algoritmos utilizando la función `buffer`.

Inicialmente, para el vector de señal se utilizará un vector de muestras aleatoriamente generadas utilizando la función `randn` (media nula y desviación típica 1) con una longitud del orden de 1000 muestras y para la respuesta impulsiva se utilizará un vector de muestras aleatoriamente generadas utilizando la función `randn` con una longitud del orden de 100 muestras

Queda claro que la longitud de los bloques deberá ser siempre mayor que la longitud de la respuesta impulsiva.

Una vez implementados los algoritmos se deberá comprobar su correcto funcionamiento comparando sus resultados con los obtenidos utilizando la función `conv`.

Cuando el resultado obtenido con su implementación sea exactamente el mismo que el devuelto por la función `conv` (los errores deberían ser del orden de 10^{-6} o inferiores) se debería pasar a trabajar con un vector de señal de más de 10000 muestras y un filtro de más de 200 muestras.

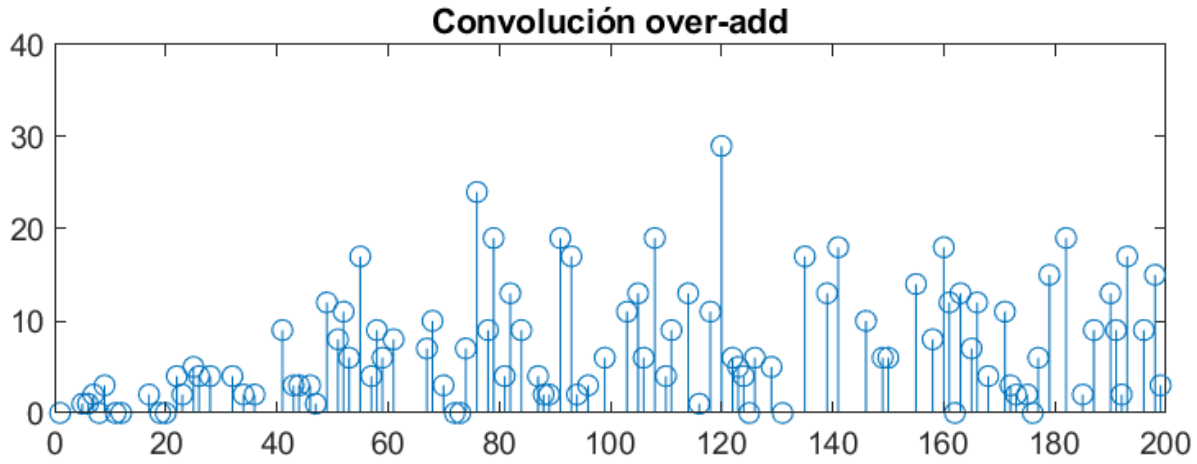
En este caso para la señal se puede seguir trabajando con ruido blan-

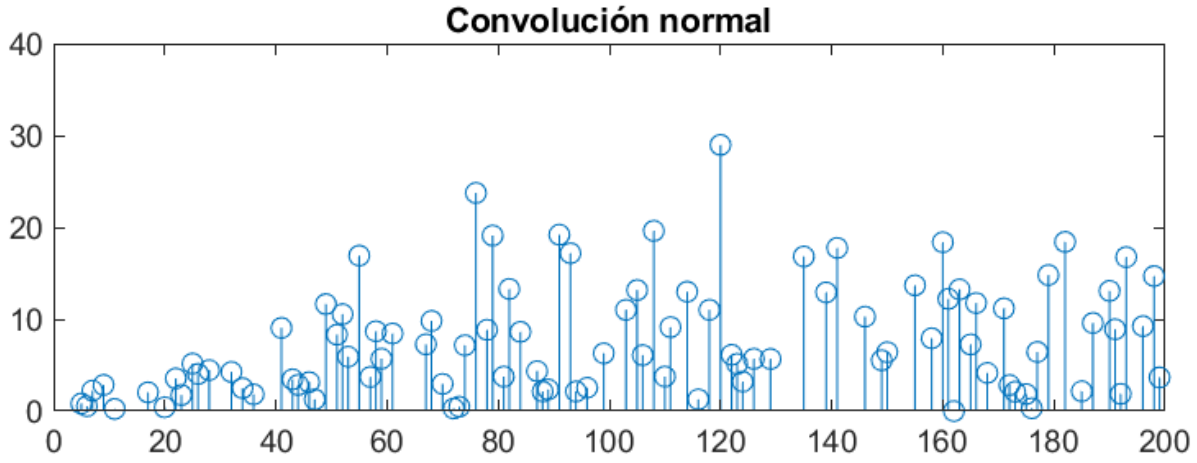
co gaussiano mientras que sería interesante trabajar con un filtro paso bajo para comprobar como modifica el espectro del ruido de entrada. Para el diseño de este filtro utilice la función `fir1` con los siguientes parámetros: `b = fir1(N, Wn);`, donde *N* es el número de muestras menos 1, y *Wn* es la frecuencia de corte comprendida entre 0 y 1 donde el 1 corresponde con la frecuencia digital de π. Elija un valor no mayor de 0,2.

Utilizando las funciones `tic` y `toc` de Matlab puede comprobar la diferencia de tiempo de ejecución entre los dos algoritmos y entre diferentes configuraciones del mismo algoritmo, sobre todo, con el tamaño de los segmentos utilizados en los algoritmos.

2. Algoritmo *overalp-add*: Solapar y sumar

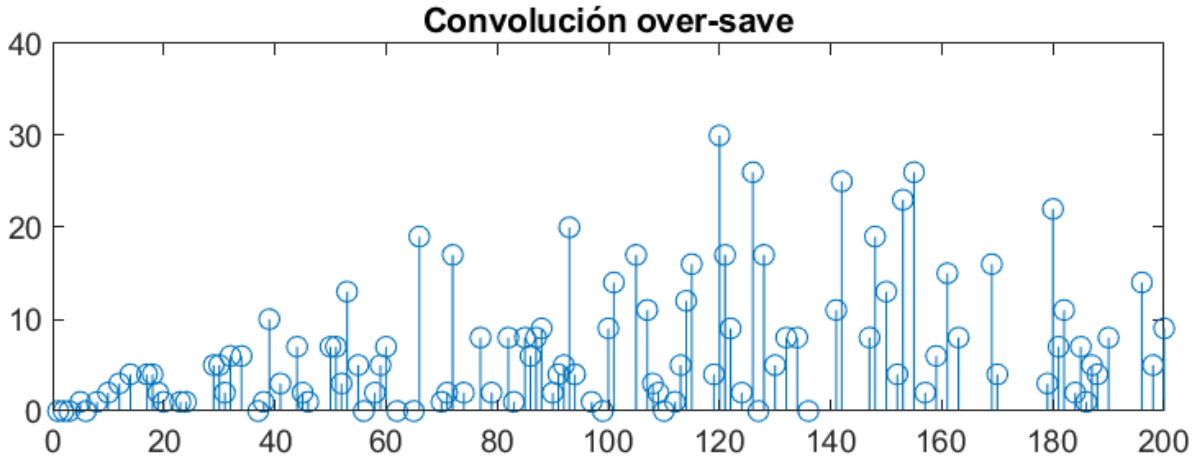
```
% overalp-add

x = randn(1,1000);
h = randn(1,100);

Lx=length(x);

Lh =length(h);

N=Lx+Lh-1;
y=zeros(1,N);

h0=[h zeros(1,Lh-1)];
n0=length(h0);
y=zeros(1,N+n0-Lh);
H=fft(h0);

for i=1:Lh:Lx
   if i<= N
         x1=[x(i:i+n0-Lh) zeros(1,n0-Lh)];
   else
        x1=[x(i:Lx) zeros(1,n0-Lh)];
   end

   XF = fft(x1);
   YFF = XF.*H;
   yc =round(ifft(YFF));

   if (i==1)
       y(1:n0)=yc(1:n0);
   else
       y(i:i+n0-1)=y(i:i+n0-1)+ yc(1:n0);
   end
end

subplot(211);
stem(y(1:N));
axis([0 200 0 40]);
title('Convolución over-add');

subplot(212);
y1 = conv(x,h);
stem(y1(1:N));
axis([0 200 0 40]);
title('Convolución normal');
```

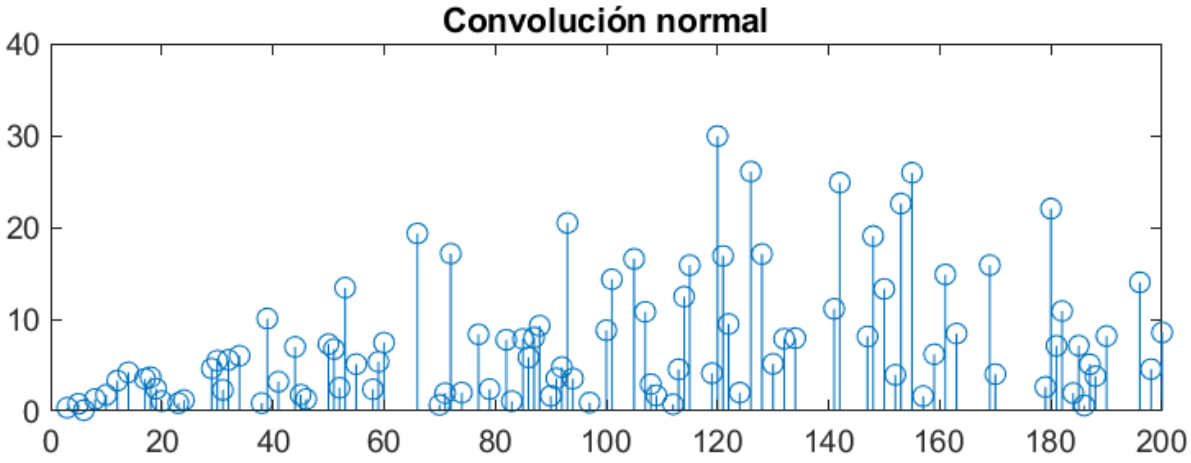

Convolución over-add
40
30
20
10
0
0
20
40
60
80
100
120
140
160
180
200


Convolución normal
40
30
20
10
0
0
20
40
60
80
100
120
140
160
180
200

3. Algoritmo *overlap-save*: Solapar y guardar

```
% overlap-save

x = randn(1,1000);
h = randn(1,100);

Lx=length(x);
Lh =length(h);

N=Lx+Lh-1;

h0=[h zeros(1,Lh-1)];
n0=length(h0);
y=zeros(1,N);
x0=[zeros(1,n0-Lh) x zeros(1,n0)];
H=fft(h0);

for i=1:Lh:Lx
    y1= x0(i:i+(2*(n0-Lh)));
    YF=fft(y1);
    YFF = YF.*H;
    yc = round(ifft(YFF));
    y(i:(i+n0-Lh))= yc(Lh:n0);
end

subplot(211);
stem(y(1:N));
axis([0 200 0 40]);
title('Convolución over-save');

subplot(212);
y2 = conv(x,h);
stem(y2(1:N));
axis([0 200 0 40]);
title('Convolución normal');
```


Convolución over-save
40
30
20
10
0
0
20
40
60
80
100
120
140
160
180
200


Convolución normal
40
30
20
10
0
0
20
40
60
80
100
120
140
160
180
200