CAPÍTULO 4.
Metodos de diseño de filtros FIR

En este cuarto capítulo del libro vamos a conocer y a aplicar varios métodos o técnicas de diseño de filtros FIR. Partiremos de los filtros ideales y comprenderemos la razón por la cual no son realizables. Posteriormente, conoceremos el método de muestreo en frecuencia, y finalizaremos con el método de ventaneo. De forma simultánea abordaremos esta temática desde el punto de vista teórico, y a nivel de simulación el lenguaje de programación Python.

Al finalizar el capítulo, deberás estar en capacidad de:

1. Explicar la razón por la cual los filtros ideales no son realizables.
2. Explicar el fenómeno de Gibbs a partir del truncamiento de la respuesta al impulso de un filtro ideal.
3. Diseñar filtros (pasa-bajos, pasa-altos, pasa-banda) aplicando el método de muestreo en frecuencia, apoyándose en Python para los cálculos.
4. Diseñar filtros (pasa-bajos, pasa-altos, pasa-banda) aplicando el método de ventaneo, apoyándose en Python para los cálculos.
5. Explicar el comportamiento de los ceros en filtros FIR diseñados por los métodos de promedio y ventaneo.

4.1. FILTROS ANÁLOGOS IDEALES

Para abordar el concepto de filtros ideales, debemos primero repasar la clasificación de los filtros respecto a la respuesta en frecuencia. Los filtros se clasifican en: pasa-bajos, pasa-altos, pasa-banda y rechaza-banda.

En el caso de los filtros pasa-bajos, la banda de paso inicia en los 0 $[Hz]$ y termina en la frecuencia de corte del filtro, denominada f_c. O de forma equivalente, inicia en 0 $[rad/seg]$ y termina en W_c, para $W_c = 2\pi f_c$. A partir de la frecuencia de corte inicia la banda de rechazo, en la cual el filtro idealmente atenúa por completo esas frecuencias de la señal de entrada. Por lo tanto, en el filtro ideal la ganancia (G) en la banda de paso es constante (típicamente $G = 1$), y en la banda de rechazo es cero. En la frecuencia de corte se tiene una caída con pendiente infinita.

La respuesta en frecuencia del filtro pasa-bajo ideal se presenta en la Figura 27.

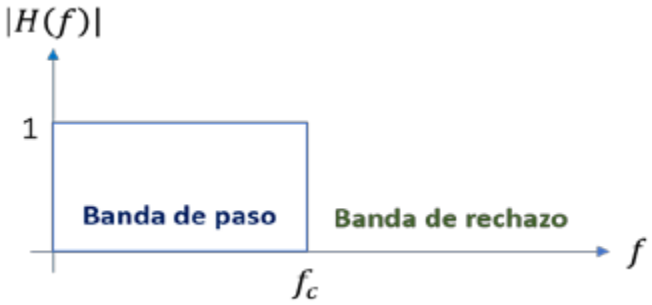

Figura 27. Respuesta en frecuencia de un filtro análogo pasa-bajo ideal.

En el caso del filtro pasa-alto ideal, la banda de rechazo inicia en 0 [Hz] y termina en la frecuencia de corte. La banda de paso corresponde a las frecuencias mayores a la f_c. Tanto el filtro pasa-alto como el filtro pasa-bajo, tienen una sola banda de paso y una sola banda de rechazo. La Figura 28 presenta la respuesta en frecuencia del filtro pasa-altos ideal.

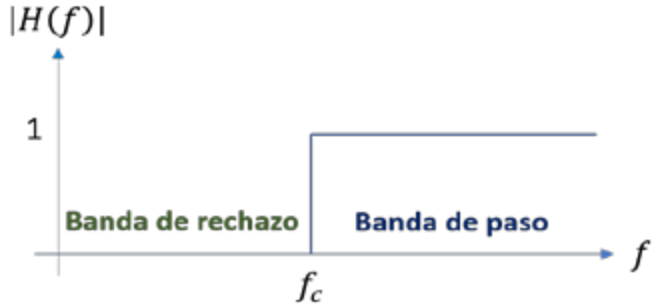

Figura 28. Respuesta en frecuencia de un filtro análogo pasa-alto ideal.

Los otros dos tipos de filtro son pasa-banda y rechaza-banda. El primero, tiene una banda de paso y dos bandas de rechazo (Figura 29). El segundo, tiene dos bandas de paso y una banda de rechazo (Figura 30). En ambos casos, se tienen dos frecuencias de corte, denominadas f_{c1} y f_{c2}.

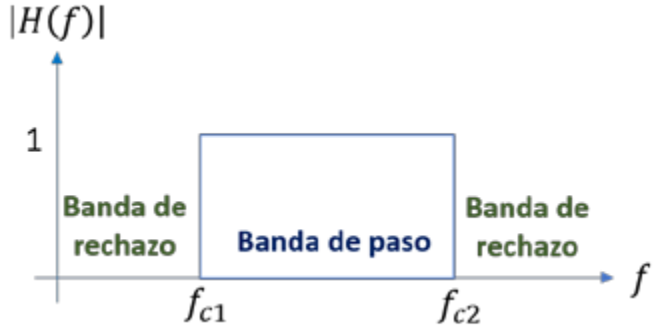

Figura 29. Respuesta en frecuencia de un filtro análogo pasa-banda ideal.

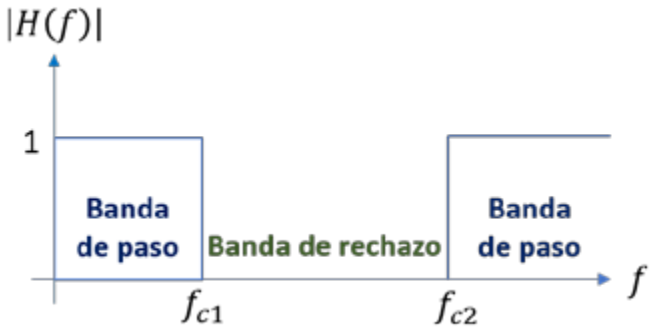

Figura 30. Respuesta en frecuencia de un filtro análogo rechaza-banda ideal.

4.2. FILTROS DIGITALES IDEALES

En el caso de los filtros digitales, la respuesta en frecuencia bilateral la expresamos en el rango [$-\pi \quad \pi$)] con unidades [rad/muestra], o en el rango [$-1 \quad 1$] con unidades [ciclo/muestra].

El filtro digital pasa-bajo ideal se presenta en la Figura 34.

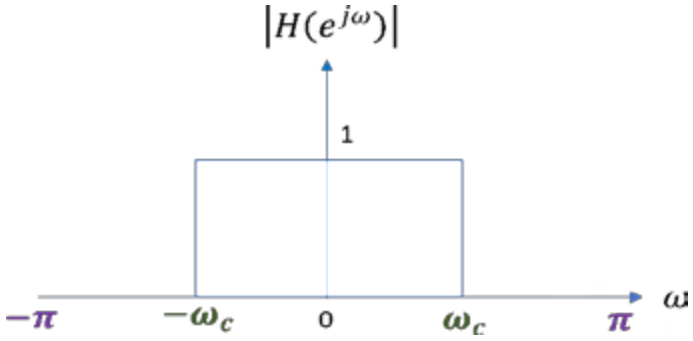

Figura 31. Respuesta en frecuencia del filtro digital pasa-bajo ideal, valores en [rad/muestra].

Matemáticamente, se define como:

$$H(e^{j\omega}) = \begin{cases} 1 & |\omega| \leq \omega c \\ 0 & e.o.c.(en\ otro\ caso) \end{cases} \quad \text{con periodicidad de } 2\pi \qquad \text{Ecuación 24}$$

Las características del filtro, son:

- Banda de paso completamente plana.
- Atenuación infinita en la banda de rechazo.
- Fase cero (sin retraso).

En el dominio del tiempo discreto, la respuesta al impulso del filtro (es decir, la Transformada de Fourier Discreta Inversa: IDTFT), es igual a:

$$h[n] = \frac{sin(\omega_c n)}{\pi n} \qquad \text{Ecuación 25}$$

La cual corresponde a una señal de duración infinita por ambos lados del eje n, conocida como señal sinc.

Revisemos ahora la estabilidad de este filtro pasa-bajos ideal. Recordando la definición de estabilidad presentada anteriormente en este libro (Capítulo 3.5), se tiene que el filtro es estable sí y solo si:

$$\sum_n |h[n]| < L \qquad para\ L<\infty$$

Entonces, el filtro pasa-bajos ideal no es estable, independiente del valor de ω_c que se seleccione, dado que la sumatoria de la magnitud de su respuesta al impulso no es finita.

A partir del concepto anterior, el *primer método* de diseño de filtros FIR corresponde al truncamiento de su respuesta al impulso. De tal forma que, partiendo de un filtro FIR ideal se selecciona un número finito de impulsos (a ambos lados del eje n) para convertirlo en un filtro estable.

4.3. TRUNCAMIENTO DE LA RESPUESTA AL IMPULSO

Este método consiste en limitar la cantidad de muestras de la respuesta al impulso del filtro. Se parte de un $h[n]$ que tiene infinitos impulsos con amplitud distinta a cero, y se llega a un $h[n]$ que tiene un número de impulsos finitos, simétrico respecto al origen.

Cuando se aplica truncamiento a $h[n]$, se hace visible el fenómeno de *Gibbs* en la respuesta en frecuencia del filtro, que consiste en la aparición de pequeñas ondulaciones tanto en la banda de paso como en la banda de rechazo del filtro. La diferencia (error) entre la máxima amplitud del rizado en relación con la amplitud plana del filtro ideal es del 9%, aproximadamente. Este error aparecerá en $H(e^{j\omega})$, independiente de la cantidad de muestras seleccionadas al truncar $h[n]$.

Por ejemplo, supongamos que la señal sinc en el dominio del tiempo discreto de duración infinita la truncamos en el rango $-5 \leq n \leq 5$, cuyo espectro se presenta en la Figura 32.

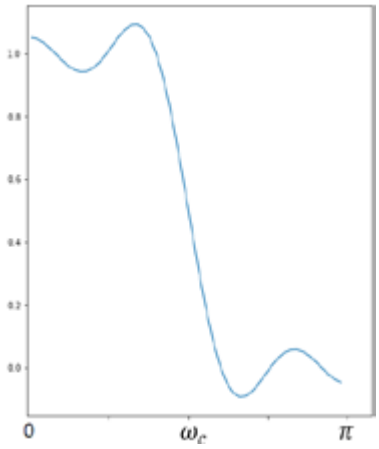

Figura 32. Espectro por truncamiento de h[n] con -5≤n≤5.

Si la misma señal *sinc* la truncamos, pero ahora en el rango $-20 \leq n \leq 20$, obtendremos el espectro de la Figura 33.

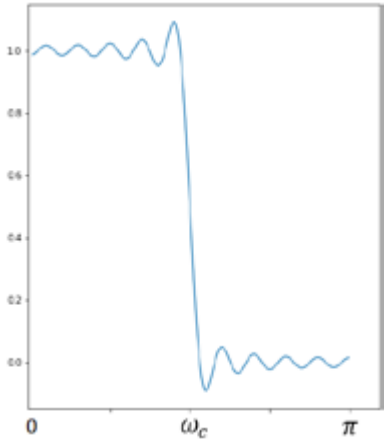

Figura 33. Espectro por truncamiento de h[n] con -20≤n≤20.

Como se aprecia en las figuras anteriores, cuando se realiza truncamiento de $h[n]$ se tiene un efecto de *"rizado"*, tanto en la banda de paso, como en la banda de rechazo. Este rizado se va "compactando" a medida que la cantidad de muestras seleccionadas de $h[n]$ aumenta, pero no desaparece.

4.4. MUESTREO EN FRECUENCIA

Este método de diseño de filtros FIR consiste en muestrear la respuesta en frecuencia de un filtro análogo ideal, y aplicar un conjunto de ecuaciones que nos permiten obtener la respuesta al impulso del filtro digital. Existen dos grupos de ecuaciones

dependiendo de si el filtro tiene una muestra en $\omega = 0$ (es decir, $\alpha = 0$) o no (es decir, $\alpha = 1)/2$). En el primer caso, se diseñan filtros con M impar, mientras que, en el segundo caso M es par.

Utilizaremos los siguientes ejemplos para ilustrar en qué consiste este método de diseño de filtros FIR. Primero, para el caso de $\alpha = 0$; y posteriormente, para $\alpha = 1)/2$.

Ejemplo 1:

Partimos de un filtro pasa-bajo ideal con $\boldsymbol{f_c = 250\ [Hz]}$. La señal de entrada la muestreamos con $\boldsymbol{f_c = 2000\ [Hz]}$ y el filtro análogo lo muestreamos con $\boldsymbol{M = 21}$ (una de sus muestras queda ubicada en la frecuencia $\boldsymbol{f = 0\ [Hz])}$. Para el diseño de este filtro, utilizaremos las ecuaciones correspondientes a $\alpha = 0$.

El valor de espaciamiento en frecuencia, Δf, entre muestras consecutivas del filtro análogo, se calcula con la siguiente ecuación:

$$\Delta f = \frac{\frac{f_s}{2}}{\frac{(M-1)}{2}} \qquad \text{Ecuación 26}$$

Que para este caso es $\boldsymbol{\Delta f = 1000/10 = 100}$, es decir que, cada 100 Hz se toma una muestra del espectro. Las muestras de amplitud distinta a cero se ubican en los siguientes valores de frecuencia $\{\boldsymbol{-200,-100,0,100,200}\}$ $[\boldsymbol{Hz}]$. Aunque la frecuencia de corte deseada está en $\boldsymbol{250\ [Hz]}$, con los valores de M y f_s seleccionados realmente se está diseñando un filtro con frecuencia de corte de $\boldsymbol{200\ [Hz]}$. El filtro muestreado se presenta en la siguiente figura.

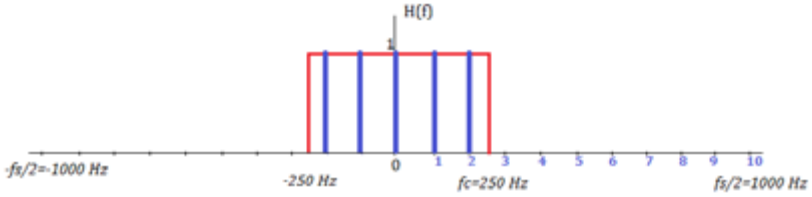

Figura 34. Muestreo en frecuencia del filtro análogo, M=21.

A partir de esta gráfica, se escribe H_r, que corresponde con el filtro muestreado:

$$H_r(k) = \begin{cases} 1 & k = 0,1,2 \\ 0 & k = 3,4,5,6,7,8,9,10 \end{cases}$$

Se debe tener en cuenta que solamente se definen los valores de k del eje de frecuencias positivo (incluido el cero), dado que los otros valores son su espejo.

A partir de H_r se obtiene $G(k)$, utilizando la siguiente ecuación:

$$G(k) = (-1)^k H_r(k) \qquad \text{Ecuación 27}$$

Realizando una alternancia en los signos de H_r así: signo positivo para los valores de k pares; signo negativo para los valores de k impar.

Entonces, para este filtro se tiene que:

$$G(k) = \begin{cases} 1 & k = 0,2 \\ -1 & k = 1 \\ 0 & k = 3,4,5,6,7,8,9,10 \end{cases}$$

Finalmente, se calcula h[n] con la ecuación (para $\alpha = 0$.):

$$h[n] = \frac{1}{M}\left\{G(0) + 2\sum_{k=1}^{U} G(k)cos\left(\left(\frac{2\pi k}{M}\right)\left(n+\frac{1}{2}\right)\right)\right\} \qquad \text{Ecuación 28}$$

La cantidad máxima de términos cosenoidales de la ecuación anterior es $U = (M-1)/2$. Sin embargo, teniendo en cuenta que a partir de k =3 se tiene que $H_r(k) = 0$, entonces solo existen los términos para $k = 1$ y $k = 2$, es decir, dos términos cosenoidales, quedando $h[n]$ expresada así:

$$h[n] = \frac{1}{21}\left\{G(0) + 2\left\{G(1)cos\left(\left(\frac{2\pi}{21}\right)\left(n+\frac{1}{2}\right)\right) + G(2)cos\left(\left(\frac{4\pi}{21}\right)\left(n+\frac{1}{2}\right)\right)\right\}\right\}$$

y al reemplazar los valores de G(k), finalmente se obtiene la siguiente ecuación de $h[n]$:

$$h[n] = \frac{1}{21}\left\{1 + 2\left\{-cos\left(\left(\frac{2\pi}{21}\right)\left(n+\frac{1}{2}\right)\right) + cos\left(\left(\frac{4\pi}{21}\right)\left(n+\frac{1}{2}\right)\right)\right\}\right\}$$

Entonces, $h[n]$ se obtiene en el rango [0 20], dado que $M = 21$.

Podemos utilizar el siguiente código en Python para obtener las 21 amplitudes de los impulsos de $h[n]$:

```
import math
import numpy as np
M=21
G0=1
G1=-1
G2=1
h= np.zeros(M)
pi = math.pi
cos = math.cos

for n in range(M):
     h[n]=1/M*(G0+2*((G1*cos(2*pi/M*(n+0.5)))+(G2*cos(4*pi/M*(n+0.5)))))
print(h)
```

Obteniendo el siguiente resultado:

```
[ 0.04445162  0.02119247 -0.01507826 -0.04761905 -0.05937998 -0.03943817
  0.01259897  0.08580656  0.16110284  0.21731539  0.23809524  0.21731539
  0.16110284  0.08580656  0.01259897 -0.03943817 -0.05937998 -0.04761905
 -0.01507826  0.02119247  0.04445162]
```

Se puede apreciar que el primer término de $h[n]$ (es decir $h[0]$) es igual al último término (es decir $h(M\text{-}1)$); el segundo término es igual al penúltimo, y así sucesivamente. De forma general, siempre que se diseñe un filtro con este método, se cumplirá que:

$$h(0) = h(M\text{-}1)$$

$$h(1) = h(M\text{-}2)$$

$$h(2) = h(M\text{-}3)$$

$$\dots$$

Como en este ejemplo M es impar, entonces el término $h((M - 1)/2)$ no tiene pareja.

Ahora, vamos a graficar la respuesta en frecuencia del filtro que hemos diseñado. Utilizaremos el siguiente código en Python:

```
from scipy import signal
import matplotlib
import matplotlib.pyplot as plt
a=1 # se hace igual a 1 porque el filtro es FIR
w1, v1 = signal.freqz(h, a)
plt.rcParams["figure.figsize"] = (14,8)
plt.plot(w1, np.abs(v1))
```

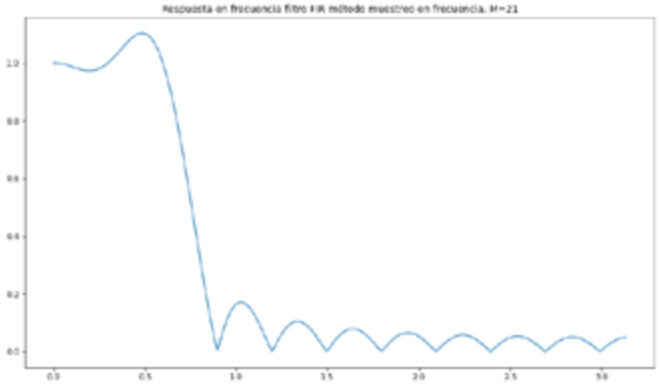

Figura 35. Magnitud de la respuesta en frecuencia método muestreo en frecuencia, M=21.

El siguiente paso consiste en encontrar a partir de la gráfica y de forma teórica la frecuencia de corte del filtro digital. Recordemos que el valor máximo es π [rad/muestra].

Para este método de diseño, la frecuencia de corte se encuentra en la amplitud en la cual se tiene una ganancia de -3 dB en escala logarítmica (o de 0.707 en escala lineal) del valor en estado estable (típicamente es 1). Entonces, de forma visual encontramos que la frecuencia de corte es de aproximadamente 0.7 $[rad/muestra]$. Podemos utilizar el siguiente código en Python para determinar su valor exacto, así:

```
x = np.where(abs(v1) > 0.707)
wcd = np.max(x)*pi/len(w1)
print(wcd)
```

```
0.6994952392758523
```

Finalmente, este valor se normaliza en el rango [0 1], de la siguiente manera:

```
fcn = wcd / pi # frecuencia de corte normalizada en el rango (0 1)
print(fcn)
```

```
0.22265625
```

Por otro lado, la frecuencia de corte normalizada teórica se calcula como:

$$f_{c_N} = \frac{2k}{M-1} \qquad \text{Ecuación 29}$$

Donde k es el máximo valor para el cual H_r es distinto de cero (o el valor mínimo para el cual H_r es distinto de cero, si el filtro es pasa-altos).

En nuestro ejemplo k=2. De tal forma que,

$$f_{c_N} = \frac{2 * 2}{20} = \frac{4}{20} = 0.2$$

El valor experimental es muy cercano el valor teórico, es decir, el filtro diseñado obtenido se aproxima en gran medida al filtro que queríamos diseñar.

Ejemplo 2:

Partimos de un filtro pasa-bajo ideal con $\boldsymbol{f_c = 450\ [Hz]}$, $\boldsymbol{f_s = 1800\ [Hz]}$, y cantidad de muestras $\boldsymbol{M = 18}$. Sin embargo, como M es par, se tiene que $\alpha = 1/2$, lo que significa que no existe muestra en $\boldsymbol{f = 0\ [Hz]}$, sino en $\boldsymbol{f = \Delta f/2\ [Hz]}$.

El valor de espaciamiento en frecuencia, Δf, entre muestras consecutivas del filtro análogo, se calcula con la siguiente ecuación:

$$\Delta f = \frac{\frac{f_s}{2}}{\frac{(M)}{2}} \qquad \text{Ecuación 30}$$

Obteniendo $\boldsymbol{\Delta f = 900/9 = 100\ [Hz]}$, cuyas muestras de valor distinto a cero se ubican en $\boldsymbol{\{-450,-350,-250,-150,-50,50,150,250,350,450\}\ [Hz]}$. El filtro muestreado se presenta en la siguiente figura.

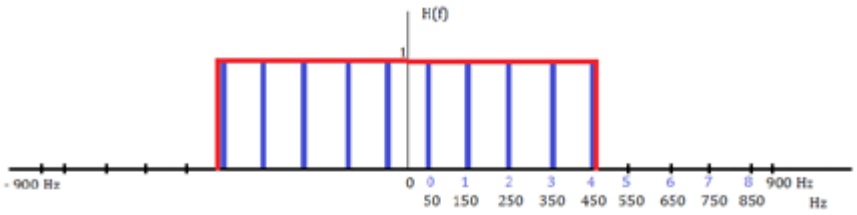

Figura 36. Muestreo en frecuencia del filtro análogo, M=18.

Como siguiente paso, escribiremos el valor de H_r, así:

$$H_r(k) = \begin{cases} 1 & k = 0,1,2,3,4 \\ 0 & k = 5,6,7,8, \end{cases}$$

Y obtenemos $G(k)$ utilizando la ecuación 27,

$$G(k) = \begin{cases} 1 & k = 0,2,4 \\ -1 & k = 1,3 \\ 0 & k = 5,6,7,8,9,10 \end{cases}$$

Y calculamos $h[n]$, a partir de $G(k)$. Se enfatiza que la ecuación cuando M es par se expresa en términos de senoidales, y no de cosenoidales como en el ejemplo anterior.

La ecuación general es:

$$h[n] = \frac{2}{M}\left\{\sum_{k=0}^{U} G(k) sen\left(\left(\frac{2\pi}{M}\right)\left(k+\frac{1}{2}\right)\left(n+\frac{1}{2}\right)\right)\right\} \qquad \textbf{Ecuación 31}$$

La cantidad máxima de términos senoidales de la ecuación anterior es $U = \frac{M}{2} - 1$, sin embargo, teniendo en cuenta que a partir de k=5 se tiene que $h_r(k)$ = 0, solamente se tendrán en este ejemplo cinco términos correspondientes a k = 0,1,2,3 y 4.

Entonces, la respuesta al impulso del filtro se define, así:

$$h[n] = \frac{2}{18}\left\{sen\left(\left(\frac{2\pi}{18}\right)\left(\frac{1}{2}\right)\left(n+\frac{1}{2}\right)\right) - sen\left(\left(\frac{2\pi}{18}\right)\left(\frac{3}{2}\right)\left(n+\frac{1}{2}\right)\right) + sen\left(\left(\frac{2\pi}{18}\right)\left(\frac{5}{2}\right)\left(n+\frac{1}{2}\right)\right) - sen\left(\left(\frac{2\pi}{18}\right)\left(\frac{7}{2}\right)\left(n+\frac{1}{2}\right)\right) + sen\left(\left(\frac{2\pi}{18}\right)\left(\frac{9}{2}\right)\left(n+\frac{1}{2}\right)\right)\right\}$$

Y se pueden obtener sus valores con el siguiente código en Python:

```
import math
import numpy as np

M=18
G0=1
G1=-1
G2=1
G3=-1
G4=1

h= np.zeros(M)
pi = math.pi
sin = math.sin

for n in range(M):
      h[n]=2/M*((G0*sin(2*pi/M*(0.5)*(n+0.5)))+(G1*sin(2*pi/M*(1+0.5)*(n+0.5)))+
               (G2*sin(2*pi/M*(2+0.5)*(n+0.5)))+(G3*sin(2*pi/M*(3+0.5)*(n+0.5)))
+
            (G4*sin(2*pi/M*(4+0.5)*(n+0.5))));
print(h)
```

Obteniendo como resultado,

```
[ 0.04272059  0.02875767 -0.057602   -0.01177696  0.07856742 -0.01681924
 -0.1235279   0.10732509  0.48829857  0.48829857  0.10732509 -0.1235279
 -0.01681924  0.07856742 -0.01177696 -0.057602    0.02875767  0.04272059]
```

De forma similar a lo obtenido en el ejemplo 1, el primer valor de $h[n]$ es igual al último valor, el segundo valor es igual al penúltimo valor, y así sucesivamente. A diferencia del caso anterior, no existe un valor que quede sin pareja, dado que M es par.

Continuaremos, dibujando la respuesta en frecuencia del filtro, con el siguiente código en Python:

```
from scipy import signal
import matplotlib
import matplotlib.pyplot as plt
a=1
w1, v1 = signal.freqz(h, a)
plt.rcParams["figure.figsize"] = (14,8)
plt.plot(w1, np.abs(v1))
```

Obteniendo,

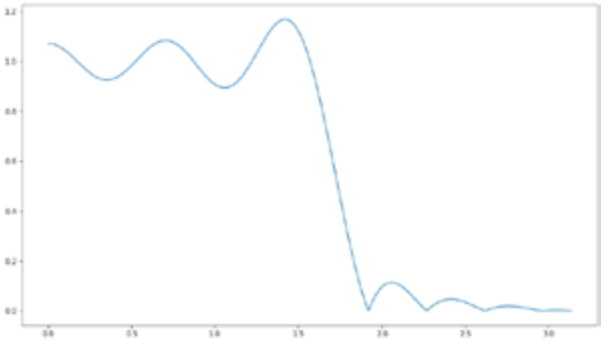

Figura 37. Magnitud de la respuesta en frecuencia método muestreo en frecuencia, M=18.

A partir de la figura anterior, se puede determinar que la frecuencia de corte del filtro digital se encuentra alrededor de 1.7 [$rad/muestra$] (evaluando la frecuencia cuya amplitud es 0.707).

Nos podemos apoyar en Python para encontrar su valor, con el siguiente código:

```
x = np.where(abs(v1) > 0.707)
wcd = np.max(x)*pi/len(w1)
print(wcd)
```

```
1.6689710972195777
```

Ahora, calculamos la frecuencia de corte normalizada del filtro digital, así:

```
fcn = wcd / pi # frecuencia de corte normalizada en el rango (0 1)
print(fcn)
```

```
0.53125
```

Y el valor teórico, por medio de la ecuación:

$$f_{c_N} = \frac{2}{M}\frac{(2k+1)}{2} \qquad \text{Ecuación 32}$$

Obteniendo en este caso,

$$f_{c_N} = \frac{2}{18} * \frac{9}{2} = 0.11 * 4.5 = 0.5$$

Como conclusión, hemos verificado que el filtro quedó diseñado correctamente.

4.5. VENTANEO

Podemos decir que este método se inspiró en el concepto de truncamiento de la respuesta al impulso. Lo que se busca, es limitar la cantidad de impulsos de la señal sinc, para que el filtro sea realizable (es decir, que no requiera de una señal en tiempo discreto de duración infinita por ambos lados del eje n), y, adicionalmente, sea estable. Sin embargo, en este caso no se descartan los coeficientes que estén por fuera del rango de la señal sinc seleccionado, sino que, se multiplica en el dominio del tiempo discreto la señal sinc por una ventana de duración finita. El efecto en el dominio de la frecuencia es el de la convolución entre el espectro de la señal sinc (que corresponde al filtro ideal) y el espectro de la ventana.

Matemáticamente, el concepto anterior lo expresamos así:

Sea $h[n]$ la respuesta al impulso del filtro ideal, y $w[n]$ la ventana discreta de duración finita. Cada una de estas señales tiene su correspondiente espectro, así:

$$h[n] \xrightarrow{DTFT} H(\omega) \qquad \text{Ecuación 33}$$

$$w[n] \xrightarrow{DTFT} W(\omega) \qquad \text{Ecuación 34}$$

Donde DTFT corresponde a la Transformada de Fourier de Tiempo Discreto (*Discrete-Time Fourier Transform*).

Entonces, se multiplica en el dominio del tiempo discreto la señal $h[n]$ de duración infinita con la señal $w[n]$ de duración finita, obteniendo una respuesta al impulso de duración finita, la cual denominaremos $\hat{h}(n)$.

$$\hat{h}[n] = h[n].w[n] \qquad \text{Ecuación 35}$$

El espectro de $\hat{h}[n]$ lo denominaremos $\hat{h}(\omega)$, el cual se obtiene de convolucionar los espectros de las señales h[n] y w[n], es decir,

$$\hat{h}[n] \xrightarrow{DTFT} \hat{H}(\omega) \qquad \text{Ecuación 36}$$

$$\hat{H}(\omega) = \frac{1}{2\pi}\{H(\omega) \circledast W(\omega)\} \qquad \text{Ecuación 37}$$

Donde $\circledast$ es el operador de convolución.

A continuación, se presenta de forma gráfica el proceso de ventaneo, en el dominio del tiempo y de la frecuencia.

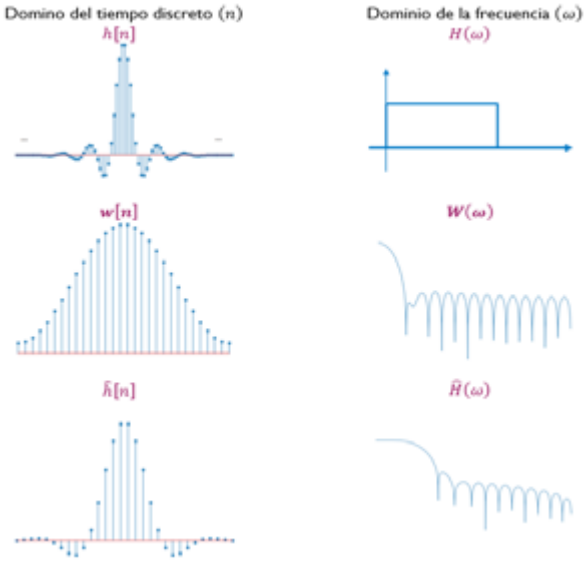

Figura 38. Diseño de filtros FIR utilizando el método de ventaneo.

Algo importante a resaltar, es que existen varios tipos de ventanas. Algunas son más suaves, otras tienen cambios bruscos de amplitud, unas son más puntiagudas, otras más anchas. Cada tipo de ventana tiene su correspondiente espectro, por lo que, el filtro resultante tendrá características diferentes. Por ejemplo, existen ventanas que atenúan de forma significativa en frecuencias distantes a la frecuencia de corte, pero que no atenúan muy bien en frecuencias cercanas a la frecuencia de corte. Otras ventanas tienen un comportamiento casi homogéneo en la zona de rechazo, pero con niveles de atenuación menores que las primeras.

En Python, la librería *scipy* tiene 23 tipos de ventanas[2] . Para diseñarlas, se puede utilizar la instrucción *signal.get_window*, o directamente con el nombre de la ventana.

A continuación, se presenta el código en Python para crear varios tipos de ventanas.

a) Ventana Boxcar (rectangular)

```
import matplotlib.pyplot as plt
from scipy import signal
M=50 # orden del filtro = M-1.
window1 = signal.boxcar(M)
from pylab import rcParams
rcParams['figure.figsize'] = 10, 6
plt.stem(window1)
```

2 https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.get_window.html#scipy.signal.get_window

b.) Ventana Hamming

```
window2 = signal.hamming(M)
from pylab import rcParams
rcParams['figure.figsize'] = 10, 6
plt.stem(window2)
```

c) Ventana Blackman

```
window3 = signal.blackman(M)
from pylab import rcParams
rcParams['figure.figsize'] = 10, 6
plt.stem(window3)
```

d) Ventana Hanning

```
window4 = signal.hann(M)
from pylab import rcParams
rcParams['figure.figsize'] = 10, 6
plt.stem(window4)
```

e) Ventana Triangular

```
window5 = signal.triang(M)
from pylab import rcParams
rcParams['figure.figsize'] = 10, 6
plt.stem(window5)
```

d) Ventana Tukey

```
window6 = signal.windows.tukey(M)
from pylab import rcParams
rcParams['figure.figsize'] = 10, 6
plt.stem(window6)
```

En la Figura 39 se presentan las seis ventanas diseñadas, todas con el mismo orden del filtro, M=50.

La primera ventana, correspondiente a boxcar, es una ventana cuyas muestras son constantes e iguales a uno, de tal forma que, es equivalente a truncar la señal *sinc* cuando se multiplica por esta ventana en el dominio del tiempo discreto. La quinta ventana, *triang*, debe su nombre precisamente a la figura geométrica que generan sus amplitudes. La ventana *tukey* se caracteriza porque tiene una zona creciente seguida de una zona constante y posteriormente una zona decreciente. Las otras tres ventanas que se seleccionaron en este ejemplo son muy similares entre sí, con un cambio de amplitud suave (sin saltos abruptos). Tanto la ventana *blackmann* como la hanning tienen su primera y última muestra de amplitud igual a cero, a diferencia de la ventana hamming que inicia y termina con una amplitud mayor a cero. Adicionalmente, de estas tres ventanas la más "angosta" es la ventana *blackman*.

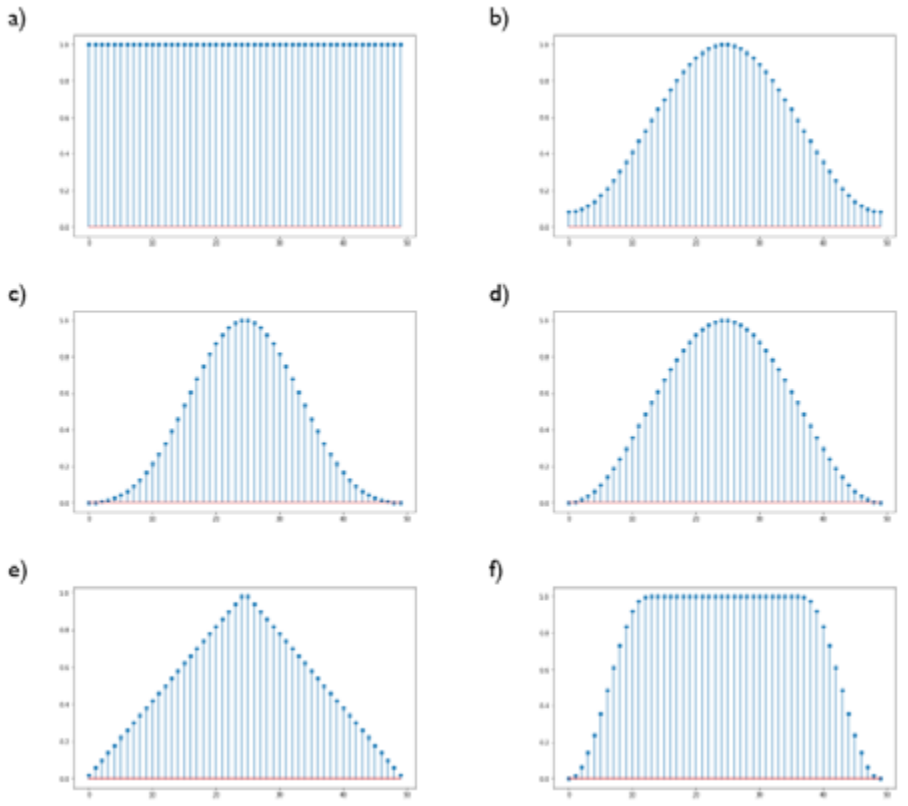

Figura 39. Ejemplos de ventanas, M=50: a) boxcar, b) hamming, c) blackman, d) hanning, e) triangular, f) tukey.

Ahora, compararemos la respuesta en frecuencia de las seis ventanas. Para ello, utilizaremos el siguiente código en Python:

```
from scipy.fft import fft, fftshift
import numpy as np
plt.figure()
window = window1 # se reemplaza para cada una de las ventanas dise-
ñadas previamente
A1 = fft(window, 2048) / (len(window)/2.0)
freq1 = np.linspace(-0.5, 0.5, len(A1))
freq1 = freq1 * 2
response1 = np.abs(fftshift(A1 / abs(A1).max()))
response1 = 20 * np.log10(np.maximum(response1, 1e-10))
from pylab import rcParams
rcParams['figure.figsize'] = 10, 6
N = len(freq1)//2
plt.plot(freq1[N+1:2*N], response1[N+1:2*N])
```

Obteniendo los siguientes espectros:

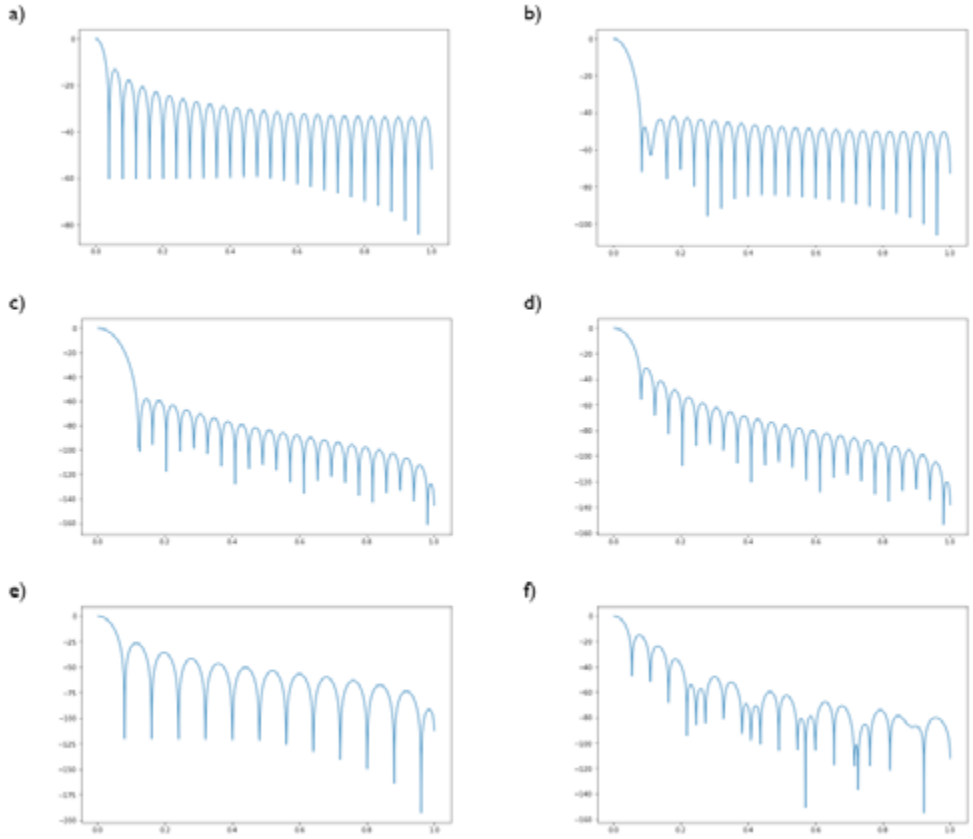

Figura 40. Respuesta en frecuencia para M=50 de las ventanas: a) boxcar, b) hamming, c) blackman, d) hanning, e) triangular, f) tukey.

Como era de esperarse, los espectros obtenidos de las seis ventanas diseñadas difieren entre sí. Empezaremos comentando el espectro de *boxcar*, el cual presenta la menor atenuación en la banda de rechazo, oscilando su ganancia entre -30 dB a -60dB, mientras que otras ventanas como la *blackman* llegan a tener hasta -160 dB de ganancia. En el caso de la ventana *hamming* la ganancia oscila entre -50dB y -90dB.

Como paso final, diseñaremos el filtro FIR con el método de ventaneo. Para ello, debemos seleccionar la ventana por la cual multiplicaremos en el dominio del tiempo la señal *sinc;* mientras que, en el dominio de la frecuencia se realizará la convolución de los dos espectros. En Python utilizamos la instrucción `signal.firwin` de la librería *scipy* para el diseño del filtro FIR por el método de ventaneo.

Los filtros que diseñaremos a continuación son pasa-bajos. Utilizaremos $f_s = 8000\ [Hz]$, y entonces por Nyquist la máxima frecuencia de corte es $fc_{max} = f_s/2 = 4000[Hz]$. Seleccionaremos como frecuencia de corte $fc = 2000[Hz]$, obteniendo que $fc = fc_{max}/2$.

Importe de librerías:

```
from scipy import signal
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
import math
```

Parámetros de diseño (M, frecuencia y tipo de filtro):

```
M=50 # el filtro es de orden M-1
f=2000
pass_zero=True # True corresponde a un filtro pasabajo.
```

Diseño del filtro FIR con la ventana *boxcar* y visualización de la respuesta en frecuencia:

```
h1= signal.firwin(M, f, window='boxcar', fs=8000, pass_zero=pass_
zero)
w1, v1 = signal.freqz(h1,1)
```

Diseño del filtro FIR con la ventana *hamming* y visualización de la respuesta en frecuencia:

```
h2= signal.firwin(M, f, window='hamming', fs=8000, pass_zero=pass_
zero)
w2, v2 = signal.freqz(h2, 1)
```

Diseño del filtro FIR con la ventana *blackman* y visualización de la respuesta en frecuencia:

```
h3= signal.firwin(M, f, window='blackman', fs=8000, pass_zero=pass_
zero)
w3, v3 = signal.freqz(h3, 1)
```

Diseño del filtro FIR con la ventana *hanning* y visualización de la respuesta en frecuencia:

```
h4= signal.firwin(M, f, window='hann', fs=8000, pass_zero=pass_zero)
w4, v4 = signal.freqz(h4, 1)
```

Diseño del filtro FIR con la ventana triangular y visualización de la respuesta en frecuencia:

```
h5= signal.firwin(M, f, window='triang', fs=8000, pass_zero=pass_
zero)
w5, v5 = signal.freqz(h5, 1)
```

Diseño del filtro FIR con la ventana tukey y visualización de la respuesta en frecuencia:

```
h6= signal.firwin(M, f, 200, window='tukey', fs=8000)
w6, v6 = signal.freqz(h6, 1)
```

Primero dibujaremos la respuesta al impulso de los filtros, $\hat{h}(n)$, (resultado de multiplicar en el dominio del tiempo la señal sinc por la ventana), y posteriormente, la respuesta en frecuencia del filtro diseñado, $\hat{H}(\omega)$, (resultado de convolucionar en el dominio de la frecuencia la respuesta en frecuencia de la ventana con la del filtro ideal).

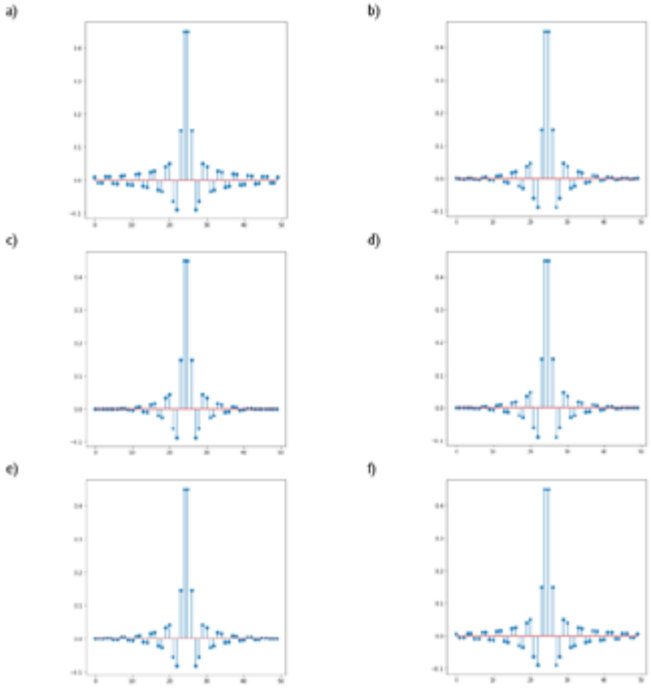

Figura 41. Respuesta al impulso, $\hat{h}(n)$, método de ventaneo, M=50: a) boxcar, b) hamming, c) blackman, d) hanning, e) triangular, f) tukey.

Se utiliza el siguiente código para dibujar las seis respuestas al impulso de los filtros:

```
plt.stem(h) # h= h1, h2, … h6.
```

Al comparar las gráficas, se aprecia que las diferencias se ven más marcadas en los primeros y últimos impulsos de $\hat{h}(n)$, es decir, en las amplitudes más pequeñas de la señal sinc.

Posteriormente, dibujaremos la respuesta en frecuencia de los seis filtros FIR, utilizando escala logarítmica:

```
plt.plot(w, 20*np.log10(np.abs(v))) # w= w1, w2, … w6.  v= v1, v2, … v6.
```

Y obtenemos los siguientes espectros:

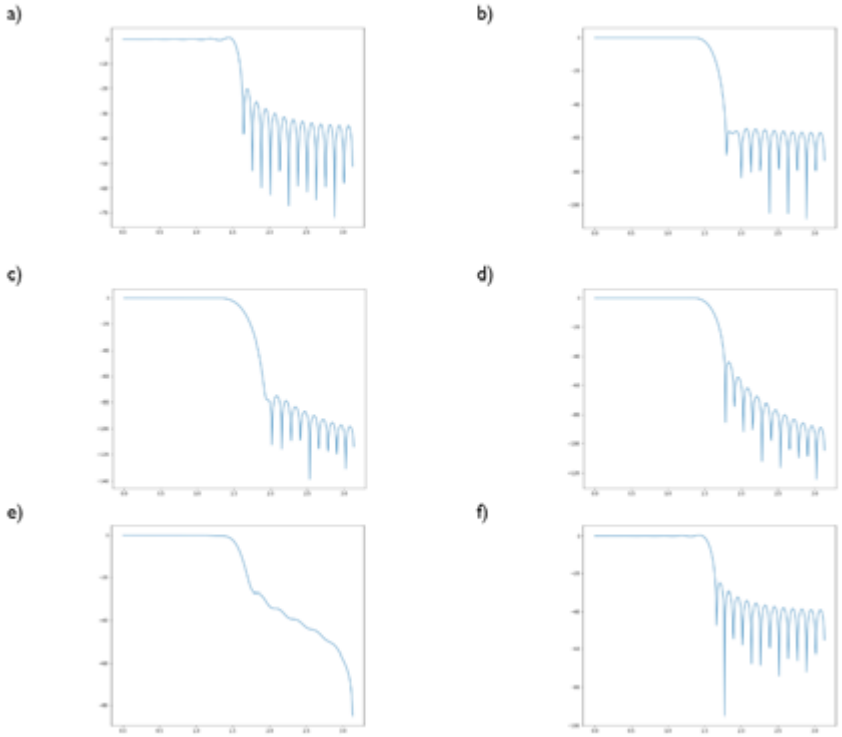

Figura 42. Respuesta en frecuencia de filtros FIR diseñados con ventanas (escala logarítmica), M=50: a) boxcar, b) hamming, c) blackman, d) hanning, e) triangular, f) tukey.

Como se observa, el comportamiento de la respuesta en frecuencia del filtro en la banda de paso cambia de forma significativa entre las ventanas seleccionadas para su diseño. Las mayores atenuaciones (ganancias alrededor de -100dB) se obtienen con las ventanas hamming, blackman, y hanning.

Finalmente, visualizaremos la respuesta en frecuencia de los filtros, pero ahora en escala lineal. El objetivo es poder determinar de forma gráfica la frecuencia de corte del filtro digital obtenido.

Para ello, utilizaremos la siguiente instrucción para cada filtro:

```
plt.plot(w, (np.abs(v))) # w= w1, w2, … w6.  v= v1, v2, … v6.
```

Y obtenemos las siguientes gráficas:

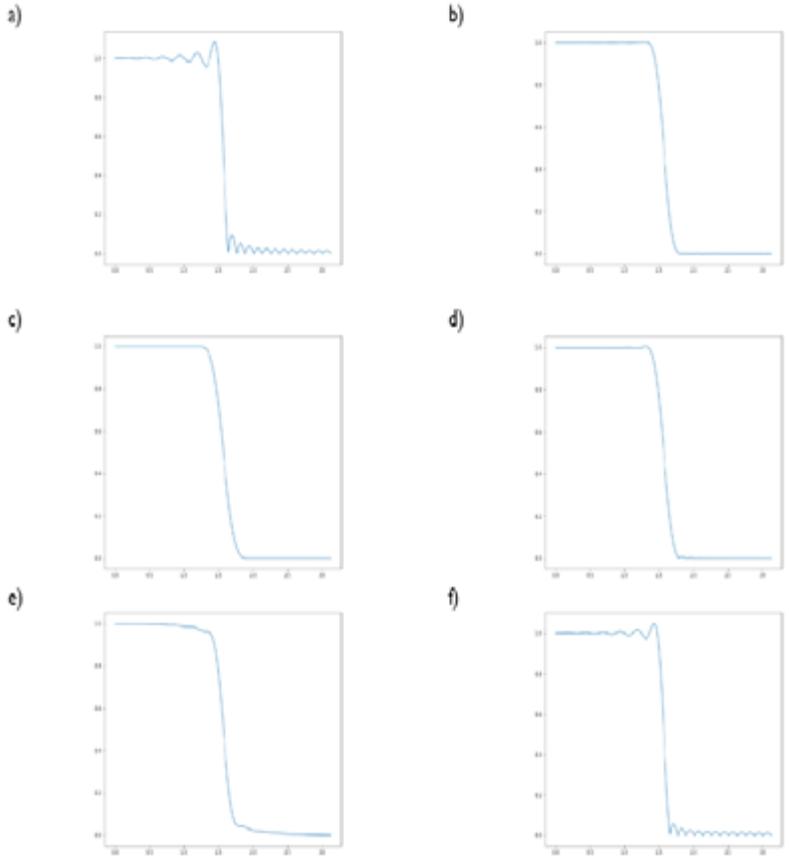

Figura 43. Respuesta en frecuencia de filtros FIR diseñados con ventanas (escala lineal), M=50: a) boxcar, b) hamming, c) blackman, d) hanning, e) triangular, f) tukey.

Podemos observar que tanto en la banda de paso como en la de rechazo, el filtro que presenta mayores ondulaciones es *boxcar* (debido al fenómeno de *Gibbs* que vimos previamente). En segundo lugar, se encuentra el filtro diseñado con la ventana tukey. Los filtros con "mejor" comportamiento, de los evaluados en esta sección, son *hamming, blackman y hanning.* En el caso del filtro diseñado con la ventana triang, su respuesta no es tan *"plana"* en las bandas de paso y de rechazo.

Como paso final, calcularemos la frecuencia de corte del filtro digital y la compararemos con la frecuencia de corte teórica. Como se mencionó en el Capítulo 2, se debe encontrar entre $[0 \quad \pi]$ (en unidades rad/muestra). Como este ejemplo utilizó $fc = fc_{max}/2$, entonces la frecuencia de corte teórica del filtro digital es de $\pi/2$.

Con el siguiente código en Python encontramos la frecuencia de corte experimental (ω_{cd}) de los seis filtros FIR diseñados con las ventanas *boxcar, hamming, blackman, hanning, triangular, y tukey.*

```
pi = math.pi
x = np.where(abs(v) > 0.707) # Para v= v1, v2, … v6.
wcd = np.max(x)*pi/len(w) # Para w= w1, w2, … w6.
print(wcd)
```

Y se obtienen los siguientes resultados:

Ventana	*boxcar*	*hamming*	*blackman*	*hanning*	*triangular*	*tukey*
ω_{cd}	1.5401	1.5155	1.5033	1.5094	1.5155	1.5401

Los cuales son cercanos al valor teórico, correspondiente a 1.5707 [*rad/muestra*].

4.6. CEROS EN FILTROS FIR

Partiendo de la función de transferencia del filtro digital, H(z), que estudiamos en el Capítulo 2 de este libro, tenemos que un filtro FIR se expresa de la siguiente forma:

$$H(z) = \sum_{k=0}^{M-1} a_k z^{-k}$$

Donde M -1 es el orden del filtro. Entonces, la cantidad de términos de $H(z)$ diferentes de cero es M. Esta función de transferencia también se puede escribir como una multiplicatoria (en lugar de una sumatoria) de M -1 términos, a partir de la factorización del polinomio de z, así:

$$H(z) = \prod_{k=1}^{M-1}(z - c_k) \qquad \text{Ecuación 38}$$

Por ejemplo, vamos a suponer que la función de transferencia del filtro FIR es $H(z) = 1 - 2z^{-1} + z^{-2}$, entonces factorizamos el polinomio de segundo orden obteniendo dos términos, así, $H(z) = (1 - z^{-1})(1 - z^{-1})$. Cada uno de los términos representa las raíces del numerador de la función de transferencia, y se conocen como los ceros del filtro digital. Es decir, cada término se iguala a cero y se despeja z para obtener los ceros del filtro.

Para este ejercicio, se tienen dos ceros en la misma posición, ubicados en:

$$(1 - z^{-1}) = 0 \quad \therefore \quad 1 = z^{-1} \quad \therefore \quad z = 1$$

Es decir, c_1=1, c_2 = 1.

> Se resalta que en el caso de los filtros FIR, solamente se tienen raíces en el numerador, es decir, los filtros FIR son sistemas solo-ceros.

En el capítulo 5 se generalizará este concepto a filtros IIR.

A medida que avancemos en el libro conoceremos el *"significado"* de los ceros de un filtro digital. Por ahora, graficaremos su posición en el plano z, apoyándonos en lenguaje de programación Python.

Partiremos con el filtro de promedio que estudiamos en el Capítulo 2 y seguiremos con el método de ventaneo.

Gráfica polos y ceros filtro de promedio:

Lo primero que vamos a realizar es definir el vector de amplitudes del filtro de promedio utilizando `np.ones`. Posteriormente, calculamos los ceros (z), polos (p) y ganancia (k), de la función de transferencia del filtro, por medio de `signal.tf2zpk`. Finalmente, dibujamos el círculo unitario en el plano z con `plt.plot(np.cos(theta),np.sin(theta))`, y ubicamos los ceros con `plt.scatter(np.real(z1),np.imag(z1))`. Se resalta que el filtro de promedio no tiene polos, dado que es un filtro FIR. Este concepto se explica con mayor detalle en el próximo capítulo.

El código completo en Python se presenta a continuación:

```
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
from scipy import signal
import math

M = 2 # M = 2, 3, 4, 5.
b = np.ones(M)/(M)
z, p, k = signal.tf2zpk(b,1)
print(len(z))
theta = np.linspace(-math.pi,math.pi,201)
plt.rcParams["figure.figsize"] = (7,7)
plt.plot(np.cos(theta),np.sin(theta))
plt.scatter(np.real(z),np.imag(z))
plt.show()
```

Y se obtienen las siguientes gráficas:

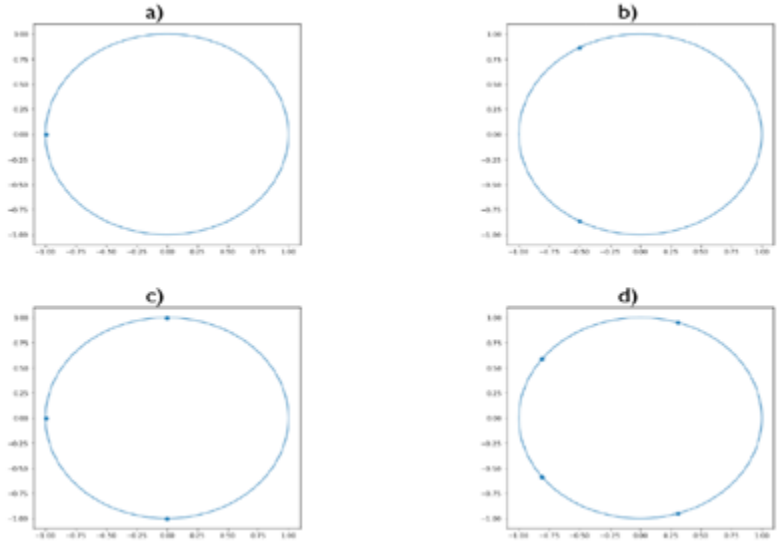

Figura 44. Gráfica de polos y ceros filtro de promedio, para: a) M=2, b) M=3, c) M=4, d) M=5.

De la figura anterior, se pueden enumerar las siguientes conclusiones:

- Todos los ceros de un filtro de promedio se ubican sobre el círculo unitario.
- La cantidad de ceros es igual a M - 1. Es decir, se tienen tantos ceros como el orden del filtro.
- Cuando el valor de M - 1. es par, cada cero tiene su conjugado, es decir, comparten el mismo valor de la parte real y con signo contrario en la parte imaginaria.
- Cuando el valor de M - 1. es impar, se tiene un cero en z = -1.
- Los ceros se concentran en la parte izquierda de la gráfica, como "alejándose" de z = -1

Gráfica polos y ceros filtro diseñado por el método de ventaneo:

Para este método de diseño, la gráfica de polos y ceros es distinta a la obtenida con el filtro de promedio. Se sugiere utilizar M par en filtros pasa-bajos, y M impar en filtros pasa-altos.

Para filtro pasa-bajos y ventana *hamming*, utilizamos el siguiente código en Python:

```
from scipy import signal
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
import math
M=2 # hacer M = 2, 4, 6, 8
f1=2000
pass_zero=True # Si es True corresponde a un filtro pasa-bajo.
h1= signal.firwin(M, f1, window='hamming', fs=8000, pass_zero=pass_zero)
z1, p1, k1 = signal.tf2zpk(h1,1)
theta = np.linspace(-math.pi,math.pi,201)
plt.rcParams["figure.figsize"] = (7,7)
plt.plot(np.cos(theta),np.sin(theta))
plt.scatter(np.real(z1),np.imag(z1))
plt.show()
```

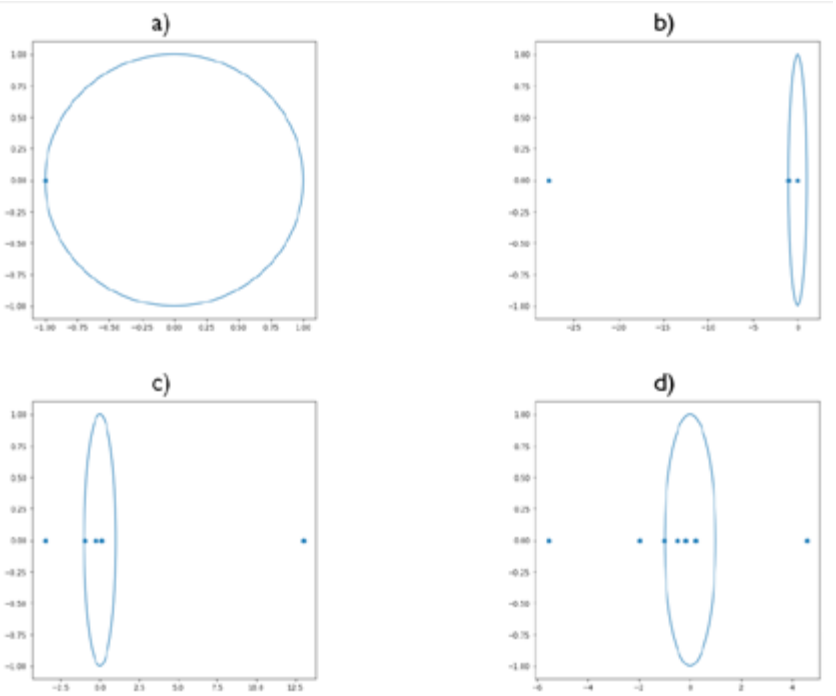

Figura 45. Gráfica de polos y ceros, filtro pasa-bajos diseñado con la ventana hamming: a) M=2, b) M=4, c) M=6, d) M=8.

En este caso, se obtienen ceros por fuera del círculo unitario, principalmente en valores negativos de z, pero eventualmente también en valores positivos. No obstante, se aprecia que uno de los ceros se encuentra en $z = -1$, dado que se diseñó un filtro pasa-bajos.

A continuación, vamos a graficar los polos y ceros, pero ahora de un filtro pasa-altos.

```
M=4
f1=2000
pass_zero=False # Si es False a un filtro pasa alto
h1= signal.firwin(M, f1, window='hamming', fs=8000, pass_zero=pass_zero)
z1, p1, k1 = signal.tf2zpk(h1,1)
theta = np.linspace(-math.pi,math.pi,201)
plt.rcParams["figure.figsize"] = (7,7)
plt.plot(np.cos(theta),np.sin(theta))
plt.scatter(np.real(z1),np.imag(z1))
plt.show()
```

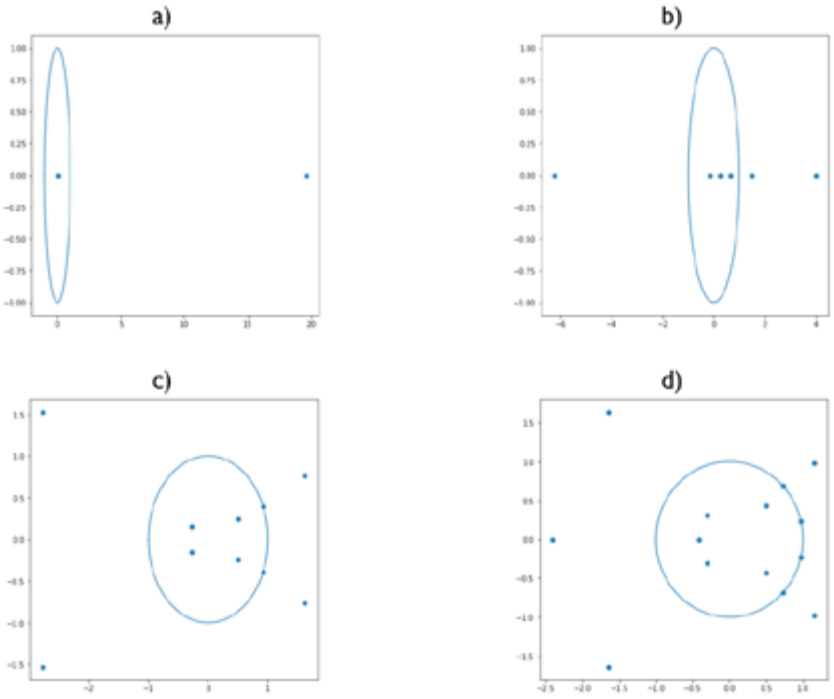

Figura 46. Gráfica de polos y ceros, filtro pasa-altos diseñado con la ventana hamming: a) M=3, b) M=7, c) M=11, d) M=15.

La principal diferencia en el comportamiento de la gráfica de polos y ceros entre filtros pasa-bajos y pasa-altos, es que en los últimos los ceros se concentran alrededor de $z = 1$ (ver Figura 46), mientras que en los primeros se concentran alrededor de $z = -1$. Esta observación es válida independiente del método de diseño del filtro y/o de la ventana seleccionada.