CAPÍTULO 5.
Metodos de diseño de filtros IIR

En este quinto capítulo del libro abordaremos el diseño de filtros de respuesta al impulso infinita (IIR), a partir del diseño de filtros análogos y aplicando mapeo entre el dominio Laplaciano y el dominio Z.

Al finalizar el capítulo, deberás estar en capacidad de:

1. Encontrar la TZ de señales de duración finita e infinita, así como su región de convergencia.
2. Diseñar filtros digitales aplicando Transformada Bilineal, con ayuda de Python.
3. Diseñar filtros Butterworth digitales, con ayuda de Python.
4. Encontrar la relación entre la frecuencia de corte del filtro análogo con la frecuencia de corte del filtro digital (o frecuencia de resonancia, en el caso de filtros pasa-banda de banda angosta).
5. Explicar el comportamiento de polos y ceros en filtros IIR.
6. Filtrar señales 1D con filtros IIR.

Una diferencia importante en el diseño de filtros digitales FIR con los IIR radica en que los segundos se diseñan a partir de un mapeo entre el dominio *Laplaciano* y el dominio Z. Teniendo en cuenta que la función de transferencia de los filtros análogos contiene un polinomio en el denominador dependiente de s, entonces, los filtros digitales obtenidos por el mapeo entre estos dos dominios contendrán en su función de transferencia un polinomio en el denominador dependiente de z. Por lo tanto, con esta técnica de diseño, siempre se obtendrán filtros IIR.

En este capítulo repasaremos algunos conceptos básicos de la TZ y posteriormente abordaremos dos ecuaciones de mapeo entre los dominios *Laplaciano* y z, una correspondiente a la aproximación en derivadas, y la otra, a la Transformada Bilineal. Aunque en la práctica la aproximación en derivadas es un método que no se utiliza por las limitaciones que tiene, permite entender el concepto de mapeo entre ambos dominios y facilita comprender en qué consiste la Transformada Bilineal. Por esa razón, la incluiremos en este capítulo.

5.1. CONCEPTOS BÁSICOS DE LA TZ

En el Capítulo 2.1 se presentó una breve introducción a la Transformada Z. En este Capítulo abordaremos el concepto de Región de Convergencia (ROC) de la TZ de la señal discreta, así:

> *"La ROC es el conjunto de todos los valores de z para los cuales la TZ de $x[n]$ es finita, es decir, que $X(z)$ converge a un valor".*

Si no se logra satisfacer la condición anterior con ningún valor de z, entonces se dice que la TZ de la señal no existe.

A continuación, aplicaremos el concepto de ROC a varios casos. Inicialmente para señales de duración finita y posteriormente para señales de duración infinita.

Caso 1: señal de duración finita causal.

Supongamos que $x[n] = \delta[n] + 2\delta[n - 1] + \delta[n - 2]$. Entonces, la TZ de la señal es $X(z) = z^0 + 2z^{-1} + z^{-2}$, ó de forma equivalente, $X(z) = 1 + \frac{2}{z} + \frac{1}{z^2}$.

Ahora bien, ¿existe algún valor o un conjunto de valores de z para los cuales $X(z)$ no sea finita? Específicamente, si $z = 0$ entonces $X(z) = 1 + \frac{2}{0} + \frac{1}{0^2} \cdot \rightarrow \infty$, es decir, $X(z)$ no converge, y entonces ese valor queda por fuera de la ROC.

Por lo cual, la ROC de la señal se expresa así:

$$ROC = todo\ plano\ z - \{z = 0\}.$$

Para $z \neq 0$ se tiene que $X(z)$ es finita.

Caso 2: señal de duración finita anti-causal

Utilizaremos la señal $x[n] = \delta[n + 2] + 5\delta[n + 1]$, cuya TZ es $X(z) = z^2 + 5z^1$. Para este caso, cuando $z = 0$ se tiene que $X(z)$ es finita, y entonces hace parte de su ROC. Analicemos entonces si para otro valor de z se tendría que $X(z)$ no es finita. Específicamente, si $z = \infty$ se tiene que $X(z) = \infty^2 + 5\infty^5 \rightarrow \infty$, por lo cual se debe excluir este valor de la ROC, quedando expresada de la siguiente manera:

$$ROC = todo\ plano\ z - \{z = \infty\}.$$

Caso 3: señal de duración infinita causal

Partiremos de la señal

$$x[n] = \alpha^n\, u[n]$$

Para conocer su comportamiento, dibujaremos la señal para algunos valores

de n teniendo en cuenta que esta señal inicia en $n = 0$ y termina en $n = \infty$. En la Figura 47 se presenta un ejemplo de la señal para algunos valores de n y $\alpha = 1/2$.

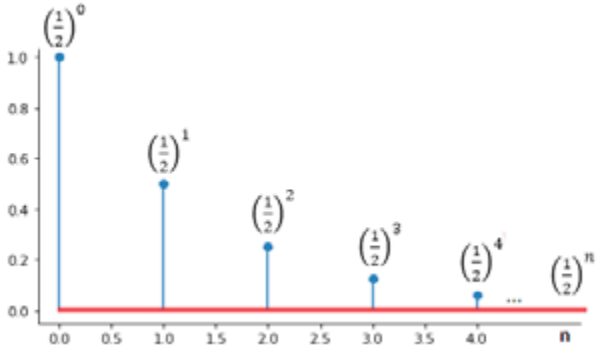

Figura 47. Gráfica de la señal $(1/2)^n\, u[n]$.

De forma general, si $0 < \alpha < 1$ la señal es decreciente, pero si $\alpha > 0$ es entonces creciente. Si α es negativa, tendrá un comportamiento oscilatorio (valores positivos y negativos alternados).

El código en Python para dibujar n muestras de la señal es:

```
import numpy as np
import matplotlib.pyplot as plt
a=0.5
n=5
n= np.linspace(0, n-1, n)
x = a **n
plt.stem(n, x)
```

La TZ de esta señal es:

$$X(z) = \alpha^0 z^0 + \alpha^1 z^{-1} + \alpha^2 z^{-2} + \alpha^3 z^{-3} + \cdots + \alpha^\infty z^{-\infty}$$

Teniendo en cuenta que la cantidad de términos de la expresión anterior es infinita, nos apoyamos en la siguiente serie matemática:

$$1 + A + A^2 + A^3 + \cdots + A^\infty = \frac{1}{1-A} \quad \leftrightarrow \quad |A| < 1$$

De tal forma que, al comparar las dos ecuaciones anteriores encontramos una similitud entre ellas cuando $A=\alpha/z$. La ROC quedaría entonces como $|\alpha/z| < 1$, o de forma equivalente $|z| > |\alpha|$.

De tal forma que, podemos reescribir la TZ de la señal, así:

$$X(z) = \frac{1}{1-\left(\frac{\alpha}{z}\right)} = \frac{1}{1-\alpha z^{-1}} \quad \leftrightarrow \quad ROC: |z| > |\alpha|$$

Para el caso específico de $\alpha = 0.5$, se tiene que su TZ es:

$$X(z) = \frac{1}{1-0.5z^{-1}} \quad \leftrightarrow \quad ROC: |z| > 0.5$$

Por lo cual, la ROC de esta señal causal de duración infinita es el exterior de un círculo de radio α.

Caso 4: señal de duración infinita anti-causal

Partiremos de la señal

$$x[n] = -\beta^n u[-n-1]$$

Esta señal existe desde n= -∞ hasta n = -1. Para los demás valores de n, su amplitud es cero. La siguiente figura presenta su comportamiento para algunos valores de n, y con β = 2.

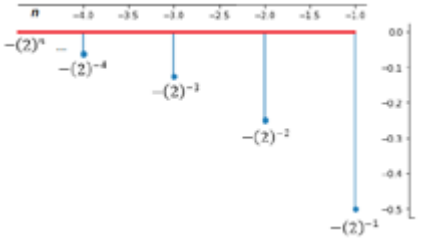

Figura 48. Gráfica de la señal -(2)^n u[-n-1].

Se debe tener en cuenta que el signo negativo está por fuera de la potencia n, de tal forma que toda la amplitud de la señal se invierte. Ahora bien, si $\beta > 1$, entonces se tiene una señal que disminuye en amplitud a medida que se aleja del origen en valores negativos de n.

El código en Python para dibujar n muestras de la señal es:

```
import numpy as np
import matplotlib.pyplot as plt
a=2
n=4
n= np.linspace(-n, -1, n)
x = -(a **n)
plt.stem(n, x)
```

La TZ de esta señal es:

$$X(z) = -\{\beta^{-1}z^1 + \beta^{-2}z^2 + \beta^{-3}z^3 + \cdots + \beta^{-\infty}z^{\infty}\}$$

De forma similar al caso anterior, nos apoyamos en la siguiente serie matemática:

$$1 + A + A^2 + A^3 + \cdots + A^{\infty} = \frac{1}{1-A} \quad \leftrightarrow \quad |A|<1$$

Pasando el valor de 1 al lado derecho de la ecuación, tenemos que:

$$A + A^2 + A^3 + \cdots + A^{\infty} = \frac{1}{1-A} - 1 \quad \leftrightarrow \quad |A|<1$$

Y resolviendo,

$$A + A^2 + A^3 + \cdots + A^{\infty} = \frac{A}{1-A} \qquad \leftrightarrow \qquad |A|<1$$

Finalmente, multiplicamos a ambos lados de la ecuación por -1, obteniendo que:

$$-\{A + A^2 + A^3 + \cdots + A^{\infty}\} = \frac{A}{A-1} \qquad \leftrightarrow \qquad |A|<1$$

La TZ de la señal se parece a la serie anterior cuando $A = z/\beta = \beta^{-1}z$. Y la ROC quedaría como $|\beta^{-1}z| < 1$, o de forma equivalente, $|z| < |\beta|$.

Entonces, podemos reescribir la TZ de la señal, así:

$$X(z) = \frac{\beta^{-1}z}{\beta^{-1}z-1}$$

O de forma equivalente,

$$\frac{1}{1-\beta z-1} \leftrightarrow \quad ROC: |z|<|\beta|$$

Para el caso específico de $\beta = 2$, se tiene que su TZ es:

$$X(z) = \frac{1}{1-2z^{-1}} \leftrightarrow \quad ROC: |z| < 2$$

Entonces, la ROC de esta señal anti-causal de duración infinita es el interior de un círculo de radio β.

Caso 5: señal de duración infinita por ambos lados de *n*

Partiremos de la señal

$$x[n] = \alpha^n u[n] - \beta^n u[-n-1]$$

Para $x_1[n] = \alpha^n u[n]$, y $x_2[n] = -\beta^n u[-n-1]$, es decir que, $x[n] = x_1[n] + x^2[n]$.

Teniendo en cuenta lo presentado en el Caso 3 y Caso 4 de este Capítulo, se tiene que:

$$X(z) = \frac{1}{1-\alpha z^{-1}} + \frac{1}{1-\beta z^{-1}}$$

Con ROC: $|z| > |\alpha| \cap |z| < |\beta|$

De tal forma que, la TZ $\exists \leftrightarrow \beta > \alpha$. En caso contrario, $\nexists$.

5.2. APROXIMACIÓN EN DERIVADAS

El concepto que vamos a aplicar en esta subsección y la siguiente es el de mapeo. Pero ¿qué significa exactamente mapear dos dominios? Según Britannica[3], la defini-

3 Disponible en: https://www.britannica.com/science/mapping

ción de mapeo es *"cualquier forma prescrita de asignar a cada objeto en un conjunto a un objeto en particular en otro (o el mismo) conjunto"*. Entonces, para nuestro caso, el mapeo permite relacionar el dominio Laplaciano con el dominio z a través de una función.

En el caso del método de aproximación en derivadas, se mapea la función de transferencia $H(s)$ con la función de transferencia $H(z)$, correspondiente a la derivada. En el dominio Laplaciano la función de transferencia de la derivada es $H(s) = s$, mientras que en el dominio z es $H(z) = (1 - z^{-1})/T_s$, donde T_s corresponde al periodo de muestreo del sistema (es decir, el espaciamiento entre muestras consecutivas, sabiendo que $T_s = 1/f_s$).

La Figura 49 nos permite ilustrar el concepto de derivada. Supongamos que queremos calcular la derivada de una señal discreta en un tiempo específico n, denominada $m(n)$, la cual se define como el incremento en amplitud de la señal dividido en el periodo de muestreo, T_s, de la forma:

$$m(n) = \frac{x(n) - x(n-1)}{T_s} \qquad \text{Ecuación 39}$$

Por ejemplo, para $n = 7$, tendremos que su derivada es $m(7) = \{x(7) - x(6)\}/T_s$.

Entonces, si la salida del sistema es la derivada de la señal de entrada, para todo valor de n tendremos la siguiente ecuación de entrada-salida:

$$y[n] = \frac{x[n] - x[n-1]}{T_s} \qquad \text{Ecuación 40}$$

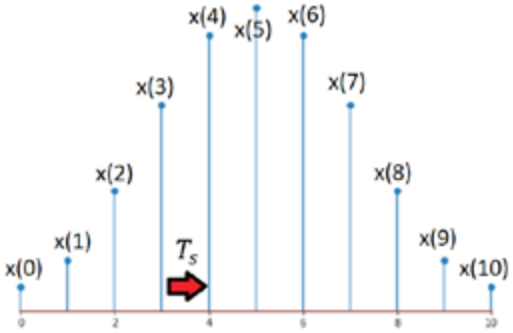

Figura 49. Señal discreta: concepto de derivada.

Aplicando la TZ a cada uno de los términos de la ecuación anterior y la propiedad de desplazamiento de la TZ, tendremos que:

$$Y(z) = \frac{X(z) - z^{-1}X(z)}{T_s} = \frac{X(z)\{1 - z^{-1}\}}{T_s} \qquad \text{Ecuación 41}$$

De tal forma que la función de transferencia nos queda así:

$$\frac{Y(z)}{X(z)} = H(z) = \frac{\{1-z-1\}}{T_s} \qquad \text{Ecuación 42}$$

Relacionando las dos funciones de transferencia $H(s)$ con la de $H(z)$, obtenemos el mapeo entre el dominio s y el dominio z:

$$s = \frac{(1-z^{-1})}{T_s} \qquad \text{Ecuación 43}$$

Despejando z de la ecuación anterior, obtenemos:

$$z = \frac{1}{1-(sT_s)} \qquad \text{Ecuación 44}$$

Finalmente, al reemplazar $s = j\Omega$, obtendremos:

$$z = \frac{1}{1-j\Omega T_s} = \frac{1}{1-j\Omega T_s} * \frac{1+j\Omega T_s}{1+j\Omega T_s} = \frac{1+j\Omega T_s}{1-(j\Omega T_s)^2} = \frac{1+j\Omega T_s}{1-j^2\Omega^2 {T_s}^2}$$

Entonces,

$$z = \frac{1}{1+\Omega^2 {T_s}^2} + \frac{j\Omega T_s}{1+\Omega^2 {T_s}^2} \qquad \text{Ecuación 45}$$

Donde Ω corresponde a la frecuencia de corte del filtro análogo. Al variar Ω en el rango {-∞, ∞} se obtiene una correspondencia en el plano z de un círculo de $r = 0.5$ y centro en $z = 0.5$. De tal forma que un filtro análogo estable (el cual tiene sus polos en el semiplano izquierdo), se transforma en un filtro digital estable (el cual tiene sus polos dentro del círculo unitario). La principal desventaja de este método de diseño de filtros IIR consiste en que la ubicación de los polos en ese pequeño círculo corresponde a frecuencias bajas. De tal forma que solamente se pueden diseñar filtros con valores de ΩT_s pequeños.

5.3. TRANSFORMADA BILINEAL

La Transformada Bilineal es una mejora del método de aproximación en derivadas, dado que se mapea todo el semi-plano izquierdo del dominio Laplaciano, aprovechando todo el interior del círculo unitario. Como consecuencia, se pueden diseñar filtros de cualquier frecuencia de corte, superando la limitación que tenía el método de aproximación en derivadas.

La ecuación que nos permite mapear ambos dominios es:

$$s = \frac{2}{T_s}\left(\frac{1-z^{-1}}{1+z^{-1}}\right) \qquad \text{Ecuación 46}$$

Con esta función de mapeo, todo el semiplano izquierdo en el dominio s se corresponde con el interior del círculo unitario en el dominio z.

En el dominio Laplaciano, un filtro pasa bajo tiene un cero en $s = \infty$. Cuando se aplica la ecuación 46, el cero del filtro digital queda ubicado en en $z = -1$.

Adicionalmente, la correspondencia entre la frecuencia de corte del filtro análogo

(Ω_a) con la del filtro digital (ω_d) no es lineal, sino una relación de tipo tangencial, dado por la ecuación:

$$\Omega_a \cong \frac{2}{T_s} tg\left(\frac{\omega_d}{2}\right) \qquad \text{Ecuación 47}$$

A partir de la ecuación anterior, podemos encontrar la frecuencia de corte (o de resonancia) del filtro digital a partir de la del filtro análogo, así:

$$\omega_d \cong 2tg^{-1}\left(\frac{\Omega_a * T_s}{2}\right) \qquad \text{Ecuación 48}$$

Como siguiente paso, necesitamos recordar las funciones de transferencia de filtros análogos. Trabajaremos con filtros de segundo orden.

Tipo de filtro	Función de transferencia	Variables de diseño
Pasa-bajo	$H(s) = \frac{G{\Omega_c}^2}{s^2 + (2\varsigma\Omega_c)s + {\Omega_c}^2}$	G: ganancia del filtro Ω_c: frecuencia de corte del filtro análogo ς: factor de amortiguamiento
Pasa-alto	$H(s) = \frac{Gs^2}{s^2 + (2\varsigma\Omega_c)s + {\Omega_c}^2}$	G: ganancia del filtro Ω_c: frecuencia de corte del filtro análogo ς: factor de amortiguamiento
Pasa-banda	$H(s) = \frac{G\left(\frac{\Omega_r}{Q}\right)s}{s^2 + \left(\frac{\Omega_r}{Q}\right)s + {\Omega_r}^2}$	G: ganancia del filtro Ω_r: frecuencia de resonancia del filtro análogo Q: factor de calidad

Se aclara que las frecuencias de los filtros análogos de las funciones de transferencia de la tabla anterior están en unidades de $[rad/seg]$.

Con el siguiente ejemplo se ilustra el método de diseño de filtro IIR con la Transformada Bilineal, apoyado en Python.

Ejemplo 1: filtro pasa-altos

Se quiere diseñar un filtro digital utilizando Transformada Bilineal, a partir de un filtro análogo pasa-alto, con $\Omega_c = 100[Hz]$, $\varsigma = 1$, y $G = 1$. La frecuencia de muestreo, f_s, es 10 veces la frecuencia de corte del filtro análogo.

El primer paso consiste en convertir la frecuencia de corte que inicialmente está en $[Hz]$ en unidades $[rad/seg]$. Posteriormente, escribir la función de transferencia en el dominio análogo, teniendo en cuenta el tipo de filtro, así:

$$H(s) = \frac{1*s^2}{s^2 + (2 * 1 * 100 * 2 * \pi)s + (100 * 2 * \pi)^2}$$

A partir de $H(s)$ se escribe el siguiente código en Python:

i

```
mport numpy as np
from scipy import signal
import matplotlib.pyplot as plt
f = 100
frad = 2*3.14*f
amort = 1
G = 1
nums=np.array([G, 0, 0])
dens=np.array([1, 2*amort*frad, frad*frad])
fs=10 *f
ws, hs = signal.freqs(nums, dens)
```

Del código anterior, f es la frecuencia de corte del filtro análogo en unidades $[Hz]$, *frad* es la frecuencia de corte del filtro análogo en unidades $[rad/seg]$, *amort* es el factor de amortiguamiento, y G es la ganancia del filtro. Adicionalmente, *nums* es el vector del polinomio del numerador de $H(s)$, *dens* es el vector del polinomio del denominador de $H(s)$, y fs es la frecuencia de muestreo del sistema. Teniendo en cuenta que ambos polinomios (numerador y denominador) son de segundo orden, entonces cada vector contiene tres valores, el primero asociado a s^2, el segundo a s^1 y el tercero a s^0.

Con la instrucción *signal.freqs* se calcula la respuesta en frecuencia del filtro análogo. La salida ws corresponde al vector de frecuencias; mientras que, hs es el vector de amplitudes de $H(s)$.

Para graficar la respuesta en frecuencia, escribimos el siguiente código:

```
plt.plot(ws, (np.abs(hs)), label=r'$|H(s)|$')
plt.legend()
plt.xlabel('Frecuencia [rad/seg]')
plt.ylabel('Magnitud')
plt.title('Respuesta en frecuencia del filtro análogo')
plt.grid()
```

Obteniendo la siguiente gráfica:

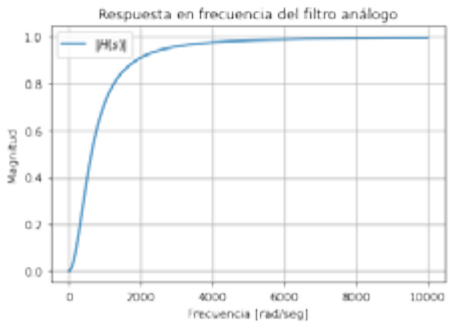

Figura 50. Respuesta en frecuencia filtro análogo pasa-alto, Ωc=200π $[rad\diagup seg]$.

Como siguiente paso, convertimos $H(s)$ en $H(z)$, aplicando Transformada Bilineal. Para ello, utilizamos la instrucción **signal.bilinear* y posteriormente creamos el sistema LTI con la instrucción *signal.dlti.* A continuación, con *signal.freqz* calculamos

la respuesta en frecuencia del filtro digital (a partir de los vectores del numerador y denominador de $H(z)$), y la graficamos.

```
filtz = signal.dlti(*signal.bilinear(nums, dens, fs))
wz, hz = signal.freqz(filtz.num, filtz.den)
plt.plot(wz, (np.abs(hz)), label=r'$|H(z)|$')
plt.legend()
plt.xlabel('Frecuencia normalizada [rad/muestra]')
plt.ylabel('Magnitud')
plt.title('Respuesta en frecuencia del filtro digital')
plt.grid()
```

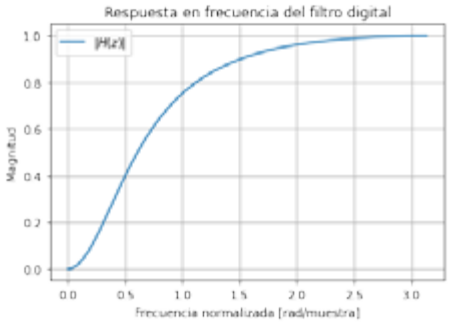

Figura 51. Respuesta en frecuencia filtro digital pasa-alto, $\omega_d = 0.6$ $[rad/muestra]$ $\varsigma = 1$.

Teniendo en cuenta que $\varsigma = 1$, entonces la frecuencia de corte corresponde a la ganancia de 0.5. Al revisar la Figura anterior, este valor se encuentra en 0.6 $[rad/muestra]$, aproximadamente.

A nivel teórico, calculamos la frecuencia de corte con el siguiente código:

```
wd =2*np.arctan(frad/(fs*2))
wd
```

Obteniendo

```
0.608501664475969
```

Coincidiendo el valor teórico con el encontrado a partir de la gráfica del filtro digital.

Por otro lado, podemos escribir la función de transferencia del filtro digital, a partir de los vectores `filtz.num` y `filtz.den`.

Sabiendo que,

```
filtz.num
```

```
array([ 0.57917428, -1.15834857, 0.57917428])
```

```
filtz.den
```

```
array([ 1. , -1.04414003, 0.2725571 ])
```

Entonces, escribimos $H(z)$, partiendo de z^0 en el primer término del polinomio, tanto del numerador como del denominador, obteniendo que:

$$H(Z) = \frac{0.57917428 - 1.15834857z^{-1} + 0.57917428z^{-2}}{1 - 1.04414003z^{-1} + 0.2725571z^{-2}}$$

Como siguiente paso, se calculan los ceros, polos y ganancia del filtro digital, utilizando `signal.tf2zpk`, así:

```
z, p, k = signal.tf2zpk(filtz.num,filtz.den)
print(z)
print(p)
print(k)
```

Obteniendo:

```
[1. 1.]
[0.52207002 0.52207002]
0.5791742828084856
```

Finalmente, se grafican los polos y ceros del filtro digital:

```
theta = np.linspace(-np.pi,np.pi,201)
plt.rcParams["figure.figsize"] = (5,5)

plt.plot(np.cos(theta),np.sin(theta))
plt.scatter(np.real(z),np.imag(z), marker='o')
plt.scatter(np.real(p),np.imag(p), marker='x')
plt.title('Gráfica polos y ceros filtro digital')
```

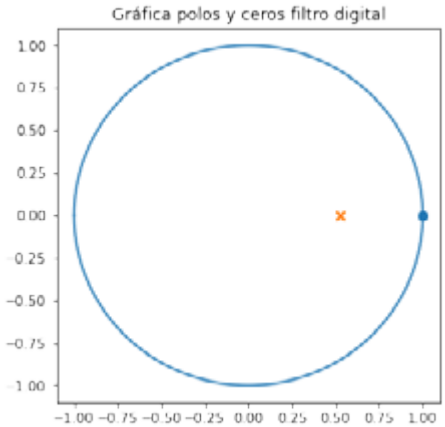

Figura 52. Gráfica de polos y ceros del filtro digital pasa-alto, ω_d=0.6 $[rad/muestra]$.

A partir de la gráfica anterior, se pueden extraer las siguientes conclusiones:

1.Si el filtro es pasa-alto, los dos ceros se ubican en z=1.

2.Los polos están relacionados con la frecuencia de corte del filtro digital. En este caso se ubican en el semicírculo derecho del plano z, dado que $(\omega_d < \pi)/2$.

Ejemplo 2: filtro pasa-bajos

Se quiere diseñar un filtro digital utilizando Transformada Bilineal, a partir de un filtro análogo pasa-bajo, con $\Omega_c = 100\ [Hz]$, ς=0.707, y G=1. La frecuencia de muestreo es 4 veces la frecuencia de corte del filtro análogo.

El primer paso consiste en convertir la frecuencia de corte que inicialmente está en $[Hz]$, en unidades $[rad/seg]$. Posteriormente, escribir la función de transferencia en el dominio análogo, teniendo en cuenta el tipo de filtro, así:

$$H(s) = \frac{1 * (100 * 2 * \pi)^2}{s^2 + (2 * 0.707 * 100 * 2 * \pi)s + (100 * 2 * \pi)^2}$$

A partir de $H(s)$ se escribe el siguiente código en Python:

```
import numpy as np
from scipy import signal
import matplotlib.pyplot as plt
f = 100
frad = 2*3.14*f
amort = 0.707
G = 1
nums=np.array([0, 0, G*frad*frad])
dens=np.array([1, 2*amort*frad, frad*frad])
ws, hs = signal.freqs(nums, dens)
plt.plot(ws, (np.abs(hs)), label=r'$|H(s)|$')
plt.legend()
plt.xlabel('Frecuencia [rad/seg]')
plt.ylabel('Magnitud')
plt.title('Respuesta en frecuencia del filtro análogo')
plt.grid()
```

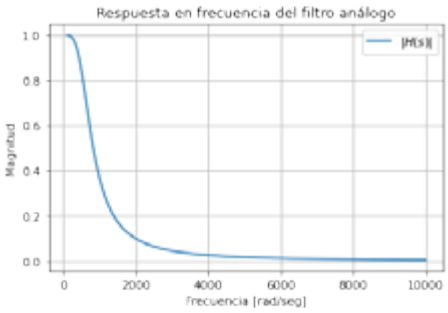

Figura 53. Respuesta en frecuencia filtro análogo pasa-bajo, $\Omega_c = 200\pi$ [$[rad/seg]$.].

Posteriormente, se utiliza la Transformada Bilineal para encontrar la función de transferencia del filtro digital, con el siguiente código:

```
fs=4 *f
filtz = signal.dlti(*signal.bilinear(nums, dens, fs))
wz, hz = signal.freqz(filtz.num, filtz.den)
plt.plot(wz, (np.abs(hz)), label=r'$|H(z)|$')
plt.legend()
plt.xlabel('Frecuencia normalizada [rad/muestra] ')
plt.ylabel('Magnitud')
plt.title('Respuesta en frecuencia del filtro digital')
plt.grid()
```

Obteniendo,

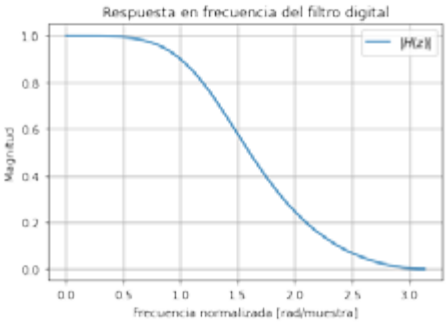

Figura 54. Respuesta en frecuencia filtro digital pasa-bajo, $\omega_d = 0.133$ [*rad/muestra*], $\varsigma = 0.707$.

La ganancia en la frecuencia de corte es 0.707, dado que $\varsigma = 0.707$. Entonces la frecuencia de corte del filtro digital a partir de la gráfica es:

```
x = np.where(abs(hz) > 0.707)
wcd = np.max(x)*3.14/len(wz)
print(wcd)
```

```
1.3246875
```

A nivel teórico, la frecuencia de corte la encontramos con la siguiente ecuación:

```
wd =2*np.arctan(frad/(fs*2))
wd
```

```
1.3310548874510058
```

Los dos valores anteriores son muy cercanos, entonces el filtro quedó bien diseñado.

Como siguiente paso, encontramos las constantes de los polinomios del numerador y denominador de $H(z)$, así:

```
filtz.num
```

```
array([0.22603683, 0.45207366, 0.22603683])
```

```
filtz.den
```

```
array([ 1. , -0.28154419, 0.18569152])
```

Y escribimos la función de transferencia del filtro digital:

$$H(z) = \frac{0.22603683 + 0.45207366z^{-1} + 0.22603683z^{-2}}{1 - 0.28154419z^{-1} + 0.18569152z^{-2}}$$

Los ceros, polos y ganancia de $H(z)$, la encontramos con el siguiente código:

```
z, p, k = signal.tf2zpk(filtz.num,filtz.den)
print(z)
print(p)
print(k)
```

Obteniendo:

```
[-1. -1.]
[0.1407721+0.40727722j 0.1407721-0.40727722j]
0.22603683128439978
```

Es decir, el filtro tiene dos ceros en z = -1, y dos polos muy cercanos al eje vertical del plano z.

La gráfica de polos y ceros del filtro se obtiene con el siguiente código:

```
theta = np.linspace(-np.pi,np.pi,201)
plt.rcParams["figure.figsize"] = (5,5)
plt.plot(np.cos(theta),np.sin(theta))
plt.scatter(np.real(z),np.imag(z), marker='o')
plt.scatter(np.real(p),np.imag(p), marker='x')
plt.title('Gráfica polos y ceros filtro digital')
```

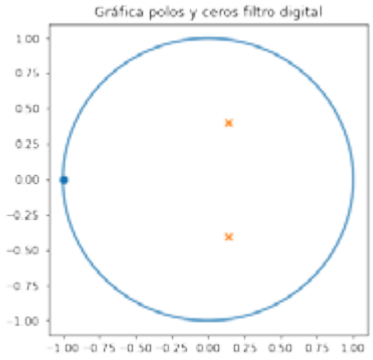

Figura 55. Gráfica de polos y ceros del filtro digital pasa-bajo, ω_d=1.33 [*rad/muestra*].

A partir de la gráfica anterior, se pueden extraer las siguientes conclusiones:

1. Si el filtro es pasa-bajo, los dos ceros se ubican en z = -1.
2. Los dos polos se ubican en el semicírculo derecho del plano z (muy cerca del eje vertical), dado que la frecuencia de corte, ω_d, es ligeramen te menor a $\pi/2$.

Ejemplo 3: filtro pasa-banda de banda angosta

Se quiere diseñar un filtro digital utilizando Transformada Bilineal, a partir de un filtro análogo pasa-banda, con $\Omega_r = 100\ [Hz]$, $Q = 2$, y $G = 1$. La frecuencia de muestreo es 3 veces la frecuencia de corte del filtro análogo.

El primer paso consiste en convertir la frecuencia de corte que inicialmente está en $H(z)$ en unidades $[rad/seg]$. Posteriormente, escribir la función de transferencia en el dominio análogo, teniendo en cuenta el tipo de filtro, así:

$$H(s) = \frac{1 * \frac{100 * 2 * \pi}{2}}{s^2 + \left(\frac{100 * 2 * \pi}{2}\right) s + (100 * 2 * \pi)^2}$$

A partir de $H(s)$ se escribe el siguiente código en Python:

```
import numpy as np
from scipy import signal
import matplotlib.pyplot as plt
f = 100
frad = 2*3.14*f
Q = 2
G = 1
nums=np.array([0, G*frad/Q, 0])
dens=np.array([1, frad/Q, frad*frad])

ws, hs = signal.freqs(nums, dens)
plt.plot(ws, (np.abs(hs)), label=r'$|H(s)|$')
plt.legend()
plt.xlabel('Frecuencia [rad/seg]')
plt.ylabel('Magnitud')
plt.title('Respuesta en frecuencia del filtro análogo')
plt.grid()
```

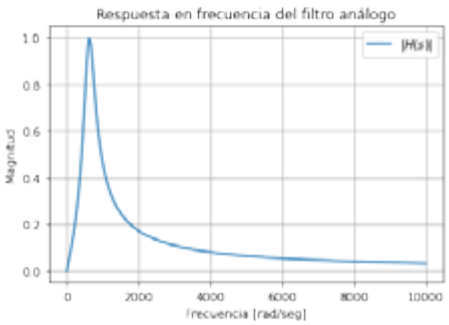

Figura 56. Respuesta en frecuencia filtro análogo pasa-banda, $\Omega_r = 200\pi\ [rad/seg]$.

Y se convierte el filtro análogo en digital con la Transformada Bilineal, así:

```
fs=3 *f
filtz = signal.dlti(*signal.bilinear(nums, dens, fs))
wz, hz = signal.freqz(filtz.num, filtz.den)
plt.plot(wz, (np.abs(hz)), label=r'$|H(z)|$')
plt.legend()
plt.xlabel('Frecuencia normalizada [rad/muestra]')
plt.ylabel('Magnitud')
plt.title('Respuesta en frecuencia del filtro digital')
plt.grid()
```

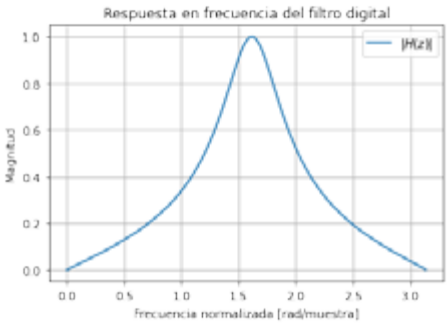

Figura 57. Respuesta en frecuencia filtro digital pasa-banda, $\omega_d = 1.61$ $[rad/muestra]$, $Q = 2$.

La frecuencia de resonancia la encontramos a partir de la gráfica anterior, con el siguiente código en Python:

```
x = np.where(abs(hz) > 0.999)
wcd = np.max(x)*3.14/len(wz)
print(wcd)
```

```
1.6251953125
```

A nivel teórico, la frecuencia de resonancia del filtro digital es:

```
wd =2*np.arctan(frad/(fs*2))
wd
```

```
1.6163910321996993
```

Los valores anteriores son muy similares entre sí, entonces hemos verificado que el filtro quedó bien diseñado.

Las constantes de los polinomios del numerador y denominador de la función de transferencia del filtro digital son:

```
filtz.num
```

```
array([ 0.19983368, 0. , -0.19983368])
```

```
filtz.den
```

```
array([1. , 0.07294142, 0.60033263])
```

De tal forma que H(z) es:

H(z)=(0.19983368+⟦-0.19983368z⟧^(-2))/(1-0.07294142z^(-1)+⟦0.60033263z⟧^(-2))

$$H(z) = \frac{0.19983368 + 0.19983368z^{-2}}{1 - 0.07294142z^{-1} + 0.60033263z^{-2}}$$

Los ceros, polos y ganancia del filtro digital se calculan con el siguiente código en Python:

```
z, p, k = signal.tf2zpk(filtz.num,filtz.den)
print(z)
print(p)
print(k)
```

```
[-1.  1.]
[-0.03647071+0.77395253j -0.03647071-0.77395253j]
0.1998336840676125
```

Y se grafican con el siguiente código:

```
theta = np.linspace(-np.pi,np.pi,201)
plt.rcParams["figure.figsize"] = (5,5)

plt.plot(np.cos(theta),np.sin(theta))
plt.scatter(np.real(z),np.imag(z), marker='o')
plt.scatter(np.real(p),np.imag(p), marker='x')
plt.title('Grafica polos y ceros filtro digital')
```

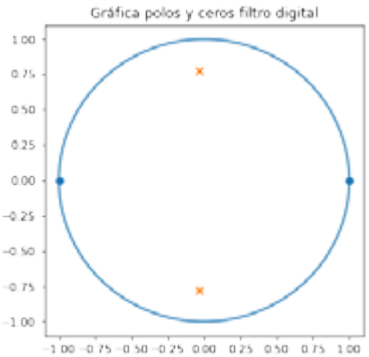

Figura 58. Gráfica de polos y ceros del filtro digital pasa-banda, ω_d=1.61 [*rad/muestra*].

A partir de la gráfica anterior, se pueden extraer las siguientes conclusiones:

1. En este caso, existe un cero en z = -1 y otro en z = 1, dado que el filtro es pasa-banda.
2. Los dos polos se ubican en el semicírculo izquierdo del plano z, dado que la frecuencia de resonancia, ω_d, es mayor a $\pi/2$.

5.4. FILTRO BUTTERWORTH

En esta última sección de diseño de filtros IIR, trabajaremos con los filtros Butterworth, los cuales se caracterizan por:

- Respuesta plana en la banda de paso.
- En la frecuencia de corte tiene una ganancia de -3 dB en escala logarítmica, o de $\sqrt{2}/2$ en escala lineal, respecto a la amplitud en la banda de paso.
- $H(s)$ solamente posee polos.

Apoyándonos en Python tenemos dos estrategias para diseñar los filtros Butterworth, las cuales son:

Diseñar un filtro análogo Butterworth y aplicar Transformada Bilineal.

Diseñar directamente el filtro digital Butterworth.

A continuación, exploraremos las dos estrategias de diseño, a partir de ejemplos.

Ejemplo 1: filtro Butterworth análogo y Transformada Bilineal

Se quiere diseñar un filtro Butterworth pasa-bajo, a partir de un filtro análogo y aplicando Transformada Bilineal, para diferentes valores de orden del filtro (específicamente, $N = 2,4,6,8,10$). La frecuencia de corte del filtro análogo es $\Omega_c = 100\ [Hz]$.

- Graficar la respuesta en frecuencia del filtro análogo, para $N = 2,4,6,8,10$.
- Escribir $H(s)$ cuando $N = 2$.
- Calcular el filtro digital a partir del filtro análogo aplicando Transformada Bilineal, con $f_s = 10 * \Omega_c$. Graficar la respuesta en frecuencia del filtro digital Butterworth, para $N = 2,4,6,8,10$
- Escribir $H(z)$ cuando $N = 2$.
- Obtener los polos y ceros del filtro digital Butterworth, para $N = 2,4,6,8,10$ Graficar los polos y ceros del filtro digital Butterworth, para $N = 2,4,6,8,10$.

Respuesta en frecuencia del filtro análogo, $N = 2,4,6,8,10$:

```
import numpy as np
from scipy import signal
import matplotlib.pyplot as plt
f = 100
wn = f * 2 * np.pi
```

```
N = 2
b2,a2 = signal.iirfilter(N, wn, btype='lowpass', analog=True,
ftype='butter')
ws2, hs2 = signal.freqs(b2, a2)
wsHz2=ws2/(2*np.pi)
plt.rcParams["figure.figsize"] = (14,8)
plt.plot(wsHz2, (np.abs(hs2)), label=r'$|H(s)| con N=2$')
plt.legend()
plt.xlabel('Frecuencia [Hz]')
plt.ylabel('Magnitud')
plt.title('Respuesta en frecuencia filtro pasa bajo')
plt.grid()
```

```
N = 4
b4,a4 = signal.iirfilter(N, wn, btype='lowpass', analog=True,
ftype='butter')
ws4, hs4 = signal.freqs(b4, a4)
wsHz4=ws4/(2*np.pi)
plt.plot(wsHz4, (np.abs(hs4)), label=r'$|H(s)| con N=4$')
plt.legend()
```

```
N = 6
b6,a6 = signal.iirfilter(N, wn, btype='lowpass', analog=True,
ftype='butter')
ws6, hs6 = signal.freqs(b6, a6)
wsHz6=ws6/(2*np.pi)
plt.plot(wsHz6, (np.abs(hs6)), label=r'$|H(s)| con N=6$')
plt.legend()
```

```
N = 8
b8,a8 = signal.iirfilter(N, wn, btype='lowpass', analog=True,
ftype='butter')
ws8, hs8 = signal.freqs(b8, a8)
wsHz8=ws8/(2*np.pi)
plt.plot(wsHz8, (np.abs(hs8)), label=r'$|H(s)| con N=8$')
plt.legend()
```

```
N = 10
b10,a10 = signal.iirfilter(N, wn, btype='lowpass', analog=True,
ftype='butter')
ws10, hs10 = signal.freqs(b10, a10)
wsHz10=ws10/(2*np.pi)
plt.plot(wsHz10, (np.abs(hs10)), label=r'$|H(s)| con N=10$')
plt.legend()
plt.show()
```

Nota: Tener en cuenta que el vector de frecuencias, *ws*, se divide entre 2π, para que la gráfica quede en [*Hz*].

Obteniendo las siguientes respuestas en frecuencia del filtro análogo:

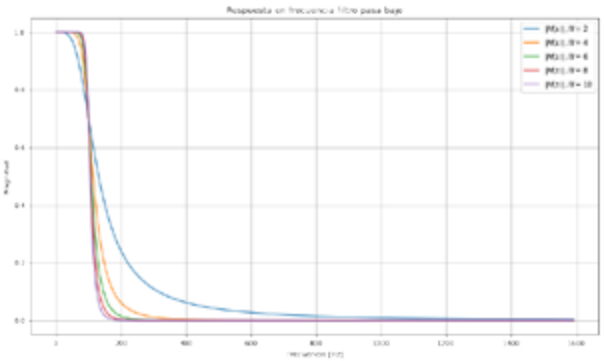

Figura 59. Respuesta en frecuencia filtro análogo Butterworth, $\Omega_c = 100[Hz]$ y $N = 2,4,6,8,10$.

De la figura anterior se puede identificar que independiente del orden del filtro, la ganancia en la frecuencia de corte es la misma, correspondiente a 0.707. Es decir, todas las curvas cruzan por el mismo valor de ganancia cuando $\Omega_c = 100[Hz]$. Adicionalmente, a medida que el valor de N aumenta, entonces la caída entre la banda de paso y la banda de rechazo se hace más pronunciada, es decir, mayor atenuación en las frecuencias cercanas a la de corte (se aproxima en mayor medida al filtro ideal).

Función de transferencia del filtro análogo, para $N = 2$:

Previamente se han encontrado las constantes de los polinomios tanto del numerador como del denominador del filtro análogo, en las variables *"b"* y *"a"*. Para el caso de $N = 2$, se utilizan $b2$ y $a2$.

```
b2
```

```
array([394784.17604357])
```

```
a2
```

```
array([1.00000000e+00, 8.88576588e+02, 3.94784176e+05])
```

A partir de los resultados anteriores, se tiene que:

$$H(s) \cong \frac{39.47*10^4}{s^2 + 8.88 * 10^2\, s + 3.94 * 10^5}$$

Nota: por simplicidad se expresó $H(s)$ solamente con dos cifras decimales.

Cálculo de $H(z)$ y respuesta en frecuencia del filtro digital:

Se utiliza `*signal.bilinear` para realizar el mapeo entre el filtro análogo y el filtro digital, `y signal.freqz` para la respuesta en frecuencia del filtro digital.

```
fs= 10*f # frecuencia de muestreo en Hz
filtz2 = signal.dlti(*signal.bilinear(b2, a2, fs))
wz2, hz2 = signal.freqz(filtz2.num, filtz2.den)
plt.plot(wz2, (np.abs(hz2)), label=r'$|H(z)|$') # se repite para los
demás valores de N
plt.legend()
plt.xlabel('Frecuencia normalizada [rad/muestra]')
plt.ylabel('Magnitud')
plt.title('Respuesta en frecuencia del filtro digital')
plt.grid()
```

Obteniendo las siguientes respuestas en frecuencia del filtro digital:

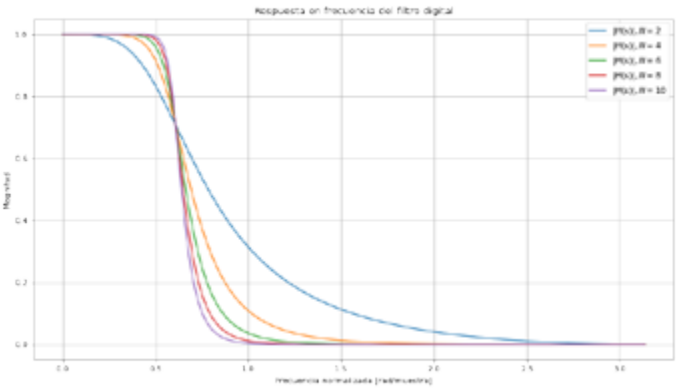

Figura 60. Respuesta en frecuencia filtro digital Butterworth, $\omega_d = 0.6$ [*rad/muestra*] y $N = 2,4,6,8,10$.

Teniendo en cuenta que se aplicó Transformada Bilineal para el diseño del filtro digital, entonces se utiliza la ecuación que relaciona la frecuencia del filtro análogo con la del filtro digital que se presentó en la sección 5.3, así:

```
fcdigital= 2 * np.arctan(wn/(2*fs))
print(fcdigital)
```

```
0.6087915947292302
```

Función de transferencia del filtro digital, para *N*=2:

A partir de los vectores filtz.num y filtz.den se encuentran las constantes de los polinomios del numerador y denominador de $H(z)$, respectivamente. Específicamente para $N = 2$, se utiliza `filtz2.num` y `filtz2.den`.

```
filtz2.num
```

```
array([0.06396438, 0.12792877, 0.06396438])
```

```
filtz2.den
```

```
array([ 1. , -1.16826067, 0.42411821])
```

Y entonces,

$$H(z) \cong \frac{0.0639 + 0.1279z^{-1} + 0.0639z^{-2}}{1 - 1.1683z^{-1} + 0.4241z^{-2}}$$

Nota: por simplicidad se expresó $H(z)$ solamente con cuatro cifras decimales.

Cálculo y gráfica de los polos y ceros del filtro digital, para N = 2,4,6,8,10:

```
z2, p2, k2 = signal.tf2zpk(filtz2.num,filtz2.den)
z4, p4, k4 = signal.tf2zpk(filtz4.num,filtz4.den)
z6, p6, k6 = signal.tf2zpk(filtz6.num,filtz6.den)
z8, p8, k8 = signal.tf2zpk(filtz8.num,filtz8.den)
z10, p10, k10 = signal.tf2zpk(filtz10.num,filtz10.den)
```

```
theta = np.linspace(-np.pi,np.pi,201)
plt.rcParams["figure.figsize"] = (5,5)

plt.plot(np.cos(theta),np.sin(theta))
plt.scatter(np.real(z2),np.imag(z2), marker='o')
plt.scatter(np.real(p2),np.imag(p2), marker='x')
plt.title('Gráfica polos y ceros filtro digital, N=2')
```

…

```
plt.plot(np.cos(theta),np.sin(theta))
plt.scatter(np.real(z10),np.imag(z10), marker='o')
plt.scatter(np.real(p10),np.imag(p10), marker='x')
plt.title('Gráfica polos y ceros filtro digital, N=10 ')
```

Y se obtienen las gráficas que se presentan en la Figura 61. Se puede apreciar que independiente del orden del filtro todos los ceros se ubican en z = -1 (por ser un filtro pasa-bajos), y que todos los polos se ubican en el semicírculo derecho (dado que $\omega_d < \pi/2$).

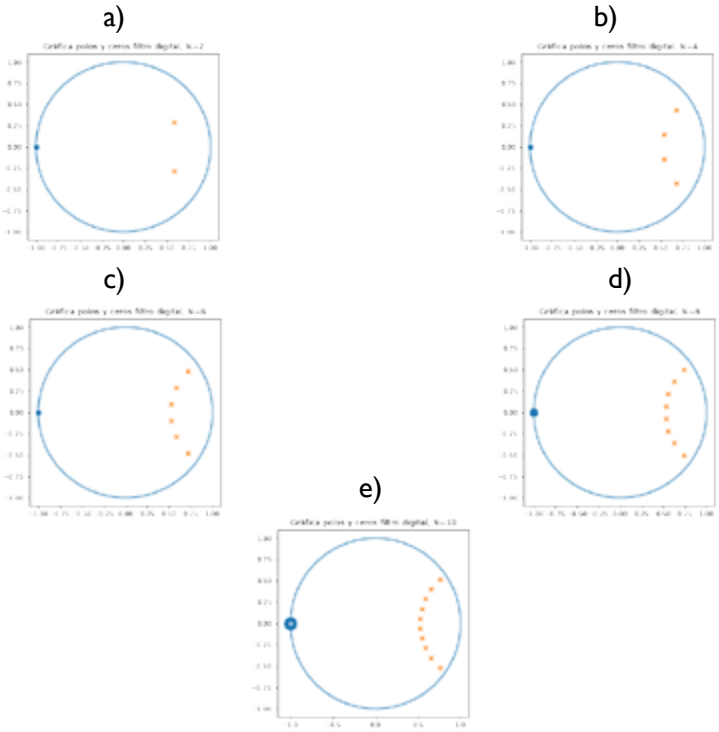

Figura 61. Gráfica de polos y ceros del filtro pasa pasa-bajo Buttherworth digital, ω_d = 0.6 [*rad/muestra*] y N = 2,4,6,8,10. Estrategia de diseño # 1.

Ejemplo 2: filtro Butterworh digital

Se quiere diseñar directamente un filtro Butterworth digital, correspondiente con un filtro análogo con $\Omega_c = 100\ [Hz]$, $f_s = 10 * \Omega_c$, y diferentes valores de orden del filtro, específicamente $N = 2,4,6,8,10$.

- Calcular Hz y graficar el filtro digital Butterworth, para $N = 2,4,6,8,10$.
- Escribir Hz cuando $N = 2$.
- Obtener los polos y ceros cuando $N = 2,4,6,8,10$. Graficar los polos y ceros cuando $N = 2,4,6,8,10$.

Como primer paso, debemos encontrar la frecuencia normalizada del filtro digital, la cual la podemos expresar como:

$$w_n = \frac{\Omega_c}{f_s/2} \qquad \text{Ecuación 49}$$

Que, en este caso es:

$$w_n = \frac{100}{\frac{1000}{2}} = 0.2$$

Nota: tener en cuenta que $0 < w_n < 1$.

Posteriormente, utilizamos la instrucción `signal.iirfilter`, haciendo `analog=False`.

Cálculo de $H(z)$ y gráfica de la respuesta en frecuencia del filtro digital:

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

N = 2
f = 100
fs = 10*f
wn = f/(fs/2)
b2, a2 = signal.iirfilter(N, wn, btype='lowpass', analog=False,
ftype='butter')
wz2, hz2 = signal.freqz(b2, a2, fs)
plt.rcParams["figure.figsize"] = (14,8)
plt.plot(wz2, (np.abs(hz2)), label=r'$|H(z)|, N=2$')
plt.legend()
plt.xlabel('Frecuencia normalizada [rad/muestra]')
plt.ylabel('Magnitud')
plt.title('Rta frecuencia filtro digital Butterworth')
plt.grid()
```

```
N = 4
b4, a4 = signal.iirfilter(N, wn, btype='lowpass', analog=False,
ftype='butter')
wz4, hz4 = signal.freqz(b4, a4, 4000)
plt.plot(wz4, (np.abs(hz4)), label=r'$|H(z)|, N=4$')
plt.legend()
```

```
N = 6
b6, a6 = signal.iirfilter(N, wn, btype='lowpass', analog=False,
ftype='butter')
wz6, hz6 = signal.freqz(b6, a6, 4000)
plt.plot(wz6, (np.abs(hz6)), label=r'$|H(z)|, N=6$')
plt.legend()
```

```
N = 8
b8, a8 = signal.iirfilter(N, wn, btype='lowpass', analog=False,
ftype='butter')
wz8, hz8 = signal.freqz(b8, a8, 4000)
plt.plot(wz8, (np.abs(hz8)), label=r'$|H(z)|, N=8$')
plt.legend()
```

```
N = 10
b10, a10 = signal.iirfilter(N, wn, btype='lowpass', analog=False,
ftype='butter')
wz10, hz10 = signal.freqz(b10, a10, 4000)
plt.plot(wz10, (np.abs(hz10)), label=r'$|H(z)|, N=10$')
plt.legend()
plt.show()
```

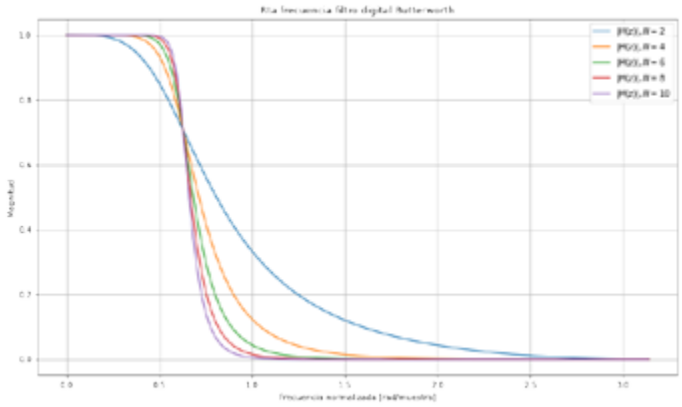

Figura 62. Respuesta en frecuencia filtro Butterworth digital pasa pasa-bajo, $\omega_N = 0.2$ y $N = 2,4,6,8,10$. Estrategia de diseño #2.

Función de transferencia del filtro digital, para $N = 2$:

A partir de los vectores b y a se encuentran las constantes de los polinomios del numerador y denominador de $H(z)$, respectivamente. Específicamente para $N = 2$, se utiliza `b2` y `a2`.

```
b2
```

```
array([0.06745527, 0.13491055, 0.06745527])
```

```
a2
```

```
array([ 1. , -1.1429805, 0.4128016])
```

Y entonces,

$$H(z) \cong \frac{0.0674 + 0.1349z^{-1} + 0.0674z^{-2}}{1 - 1.1429z^{-1} + 0.4128z^{-2}}$$

Nota 1: por simplicidad se expresó $H(z)$ solamente con cuatro cifras decimales.

Nota 2: se puede comparar este valor de $H(z)$ con el obtenido en el ejemplo #1 de esta sección.

Cálculo y gráfica de los polos y ceros del filtro digital, para N=2,4,6,8,10:

```
z2, p2, k2 = signal.tf2zpk(b2, a2)
z4, p4, k4 = signal.tf2zpk(b4, a4)
z6, p6, k6 = signal.tf2zpk(b6, a6)
z8, p8, k8 = signal.tf2zpk(b8, a8)
z10, p10, k10 = signal.tf2zpk(b10, a10)
```

Se utiliza el mismo código que del ejemplo # 1 de esta sección para graficar los polos y ceros de los filtros digitales. Las gráficas se presentan en la Figura 63.

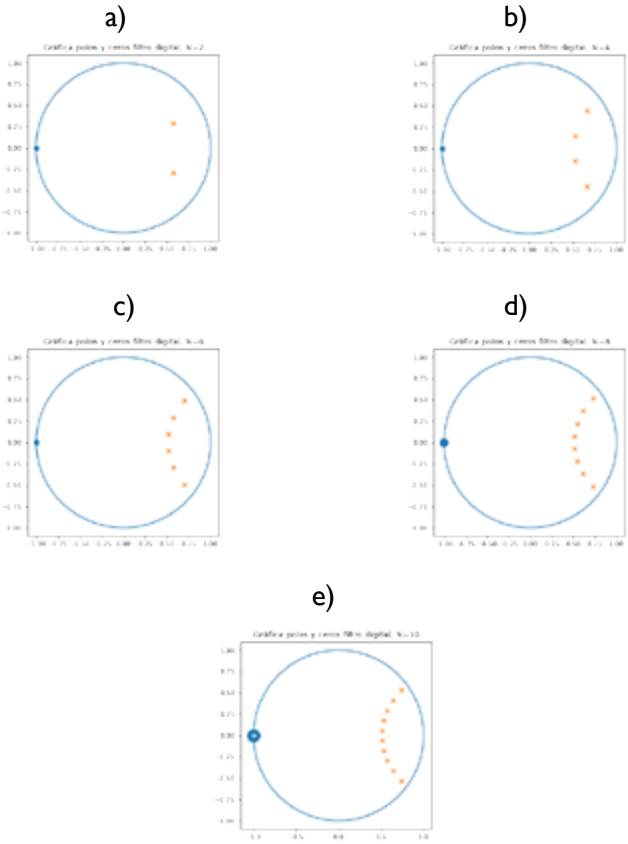

Figura 63. Gráfica de polos y ceros del filtro pasa pasa-bajo, $\omega_N = 0.2$ y N = 2,4,6,8,10. Estrategia de diseño # 2.

Al comparar la Figura 63 con la Figura 61, se aprecia que la ubicación de los polos y ceros es muy similar, por lo que las dos estrategias de diseño de esta sección permiten llegar al "mismo resultado".

5.5. FILTRADO DE SEÑALES CON FILTROS IIR

Para finalizar esta sección de filtros IIR, vamos a filtrar una señal con un filtro IIR diseñado con el método de Transformada Bilineal, a partir de un filtro análogo.

Para ello, utilizaremos la siguiente señal:

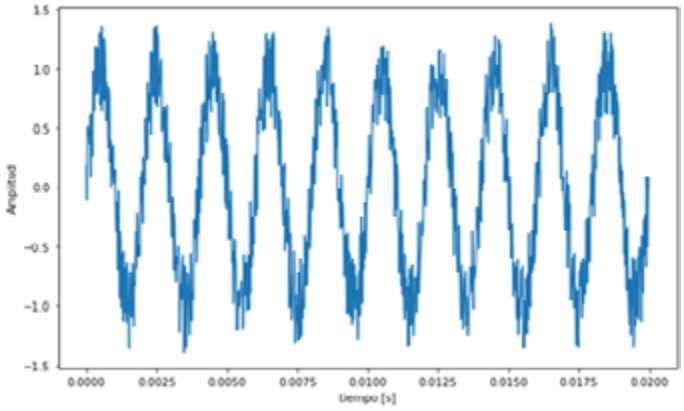

Figura 64. Señal en el dominio del tiempo, xnoise[n].

Esta señal se ha generado con el siguiente código en Python,

```
#Paso 1:  importar librerías de trabajo
import numpy as np
import scipy as sp
import matplotlib.pyplot as plt
from scipy import signal
import math

#Paso 2:  generar una señal sin ruido
f = 500 # Hz
fs = 100 * f
step = 1/fs
frad = f * 2 * math.pi
t = np.arange(0,10/f,step)
x = np.sin(frad*t)

#Paso 3:  generar ruido aleatorio
samples = len(x)
An= 0.8
noise = An*np.random.rand(samples) - An/2

# Paso 4:  sumar la señal senoidal con la señal de ruido
xnoise = x +  noise
plt.plot(t,xnoise)
plt.xlabel('tiempo [s]')
plt.ylabel('Amplitud')
```

Visualizando $xnoise[n]$, ésta contiene dos señales, una correspondiente a $x[n]$, y otra a $noise[n]$. Específicamente, $x[n]$ es una señal senoidal; mientras que, $noise[n]$, es un ruido de fondo. Sin embargo, para poder tener información más puntual del comportamiento en frecuencia tanto de $x[n]$ como de $noise[n]$, es necesario realizar un análisis espectral de la señal $xnoise[n]$.

Entonces, utilizamos el código en Python que vimos en el Capítulo 1, para el cálculo y gráfica de la Transformada de Fourier de la señal.

```
import scipy.fftpack as  fourier
L=len(xnoise)
transformada = fourier.fft(xnoise)
magnitud = abs(transformada)
magnitud_lateral = magnitud[0:L//2]
fase = np.angle(transformada)
frecuencias = fs*np.arange(0, L//2)/L
plt.plot(frecuencias, magnitud_lateral)
plt.xlabel('Frecuencia (Hz)', fontsize='10')
plt.ylabel('|FFT|', fontsize='10')
plt.show()
```

Obteniendo el siguiente espectro de $xnoise[n]$

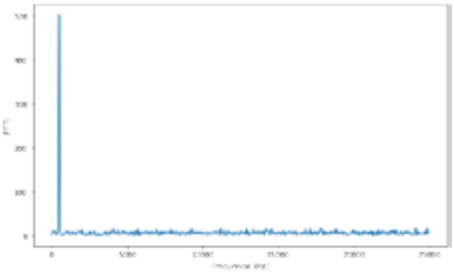

Figura 65. Espectro de $xnoise[n]$.

Observamos que existe un tono (señal de frecuencia pura) correspondiente a la señal senoidal, y que el ruido se encuentra en todos los valores de frecuencia, hasta $f_s/2$ (es decir $25K$ $[Hz]$).

Para determinar la frecuencia exacta correspondiente a la señal senoidal, vamos a apoyarnos en el siguiente código en Python:

```
np.max(magnitud_lateral)
```

```
496.67405522769
```

```
x = np.where(abs(magnitud_lateral) == np.max(magnitud_lateral))
f_tono = np.min(x)*(fs/2)/len(magnitud_lateral)
print(f_tono)
500.0
```

De tal forma que, el tono se encuentra ubicado en los 500 $[Hz]$, de amplitud 496.67.

Teniendo en cuenta que queremos filtrar el ruido que abarca todas las frecuencias, y que la señal de interés se encuentra únicamente en la frecuencia de 500 [Hz], lo

más conveniente en este caso es diseñar un filtro pasa-banda de banda angosta, con $Q > 0.5$ (ej. $Q = 1$), y $\Omega_r = 500\ [Hz]$.

Entonces, la función de transferencia del filtro análogo queda de la siguiente forma:

$$H(s) = \frac{1 * \frac{500 * 2 * \pi}{1}}{s^2 + \left(\frac{500 * 2 * \pi}{1}\right) s + (500 * 2 * \pi)^2}$$

A partir de H(s) se escribe el siguiente código en Python:

```
import numpy as np
from scipy import signal
import matplotlib.pyplot as plt
f = 500
frad = 2*3.14*f
Q = 1
G = 1
nums=np.array([0, G*frad/Q, 0])
dens=np.array([1, frad/Q, frad*frad])

ws, hs = signal.freqs(nums, dens)
plt.plot(ws, (np.abs(hs)), label=r'$|H(s)|$')
plt.legend()
plt.xlabel('Frecuencia [rad/seg]')
plt.ylabel('Magnitud')
plt.title('Respuesta en frecuencia del filtro análogo')
plt.grid()
```

Y obtenemos la siguiente respuesta en frecuencia del filtro análogo,

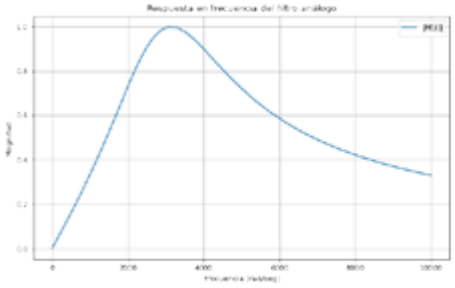

Figura 66. Respuesta en frecuencia filtro análogo pasa-banda, $\Omega_r = 1000\pi\ [rad/seg]$.

Y aplicamos Transformada Bilineal, para obtener $H(z)$, así:

```
filtz = signal.dlti(*signal.bilinear(nums, dens, fs))
wz, hz = signal.freqz(filtz.num, filtz.den)
plt.plot(wz, (np.abs(hz)), label=r'$|H(z)|$')
plt.legend()
plt.xlabel('Frecuencia normalizada [rad/muestra]')
plt.ylabel('Magnitud')
plt.title('Respuesta en frecuencia del filtro digital')
plt.grid()
```

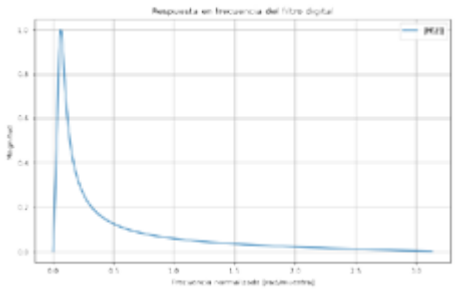

Figura 67. Respuesta en frecuencia filtro digital pasa-banda, $\omega_d = 0.061$ $[rad/muestra]$.

La frecuencia de corte del filtro digital la obtenemos tanto a partir de la gráfica de la respuesta en frecuencia del filtro, como de la ecuación teórica que relaciona la frecuencia del filtro análogo con la frecuencia del filtro digital.

Utilizamos el siguiente código en Python:

```
np.max(abs(hz))
```

```
0.9958317963050908
```

```
x = np.where(abs(hz) == np.max(abs(hz)))
wcd = np.max(x)*3.14/len(wz)
print(wcd)
```

```
0.061328125000000004
```

```
wd =2*np.arctan(frad/(fs*2))
wd
```

```
0.06277937277186546
```

Verificamos que los valores son similares, entonces el filtro quedó bien diseñado.

Posteriormente, encontramos las constantes de los polinomios del numerador y denominador de $H(z)$, así:

```
filtz.num
```

```
array([ 0.01544233, 0. , -0.01544233])
```

```
filtz.den
```

```
array([ 1. , -1.96523623, 0.96911534])
```

Y escribimos $H(z)$ de la siguiente manera,

$$H(z) = \frac{0.015 - 0.015z^{-2}}{1 - 1.965z^{-1} + 0.969z^{-2}}$$

Nota: por simplicidad se han utilizado solamente tres cifras decimales en $H(z)$.

A partir de $H(z)$ filtramos la señal con `signal.filtfilt`, con el siguiente código:

```
filtrada = signal.filtfilt(filtz.num, filtz.den, xnoise)
plt.rcParams["figure.figsize"] = (10,6)
plt.plot(t,filtrada)
```

Nota: a diferencia del caso de filtros FIR, ahora el parámetro "a" de la instrucción `signal.filtfilt` no es una constante de valor igual a uno, sino que también es un vector. Específicamente, con el nombre de las variables que hemos utilizado, corresponde a `filtz.den`.

Obteniendo como resultado:

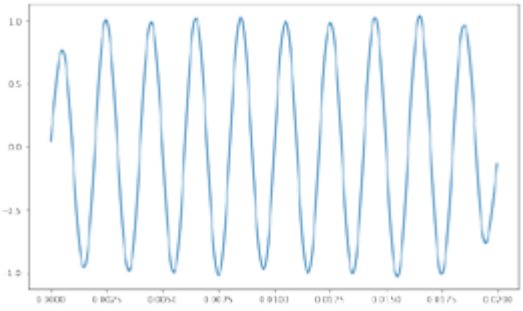

Figura 68. Señal filtrada en el dominio del tiempo.

Adicionalmente, podemos verificar que el espectro de la señal filtrada no contiene el ruido de fondo, con el siguiente código en Python:

```
transformada2 = fourier.fft(filtrada)
magnitud2 = abs(transformada2)
magnitud_lateral2 = magnitud2[0:L//2]
fase2 = np.angle(transformada2)
frecuencias2 = fs*np.arange(0, L//2)/L
plt.rcParams["figure.figsize"] = (10,6)
plt.plot(frecuencias2, magnitud_lateral2)
plt.xlabel('Frecuencia (Hz)', fontsize='10')
plt.ylabel('|FFT|', fontsize='10')
plt.show()
```

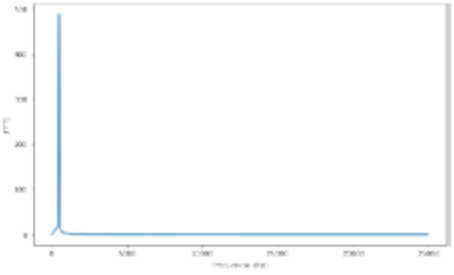

Figura 69. Espectro de la señal filtrada.