

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
FACULTAD DE CIENCIAS E INGENIERÍA
INGENIERÍA DE TELECOMUNICACIONES



PUCP

**Ingeniería de las
Telecomunicaciones**

PROCESAMIENTO DIGITAL DE SEÑALES – IEE352
TRABAJO ACADÉMICO

HORARIO: H791

NOMBRE Y APELLIDOS: Hineill David Céspedes Espinoza

CÓDIGO: 20213704

P1: 6.0
P2: 7.5
P3: 4.5
Total: 18.0

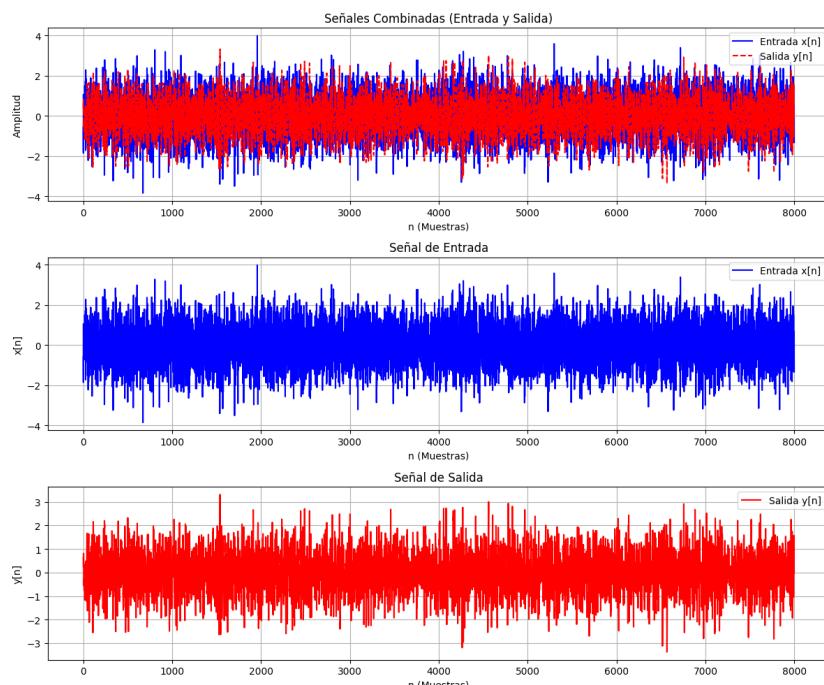
PREGUNTA 1: IDENTIFICACIÓN DEL SISTEMA

6 p

Se desea reconocer la respuesta al impulso del sistema a partir de la entrada y la salida ruidosa del sistema. Para lograr eso, emplearemos el método de gradiente descendente, es decir diseñar un filtro de Wiener y otro LMS (con coeficientes variantes en el tiempo debido a su adaptación). De tal forma se realizará una comparativa entre ambos para medir su performance o si es coherente aplicarlo a un caso donde el sistema podría no ser LTI.

1. Sistema LTI:

3 p Extraemos la data de la entrada y salida del caso del sistema LTI.



1.1 Análisis usando Filtro Wiener:

Luego aplicamos el algoritmo de gradiente descendente para el error cuadrático medio de tal forma hallamos los coeficientes del filtro de Wiener. En particular se utiliza la relación matricial que resulta de aplicar la gradiente descendente, como se muestra a continuación:

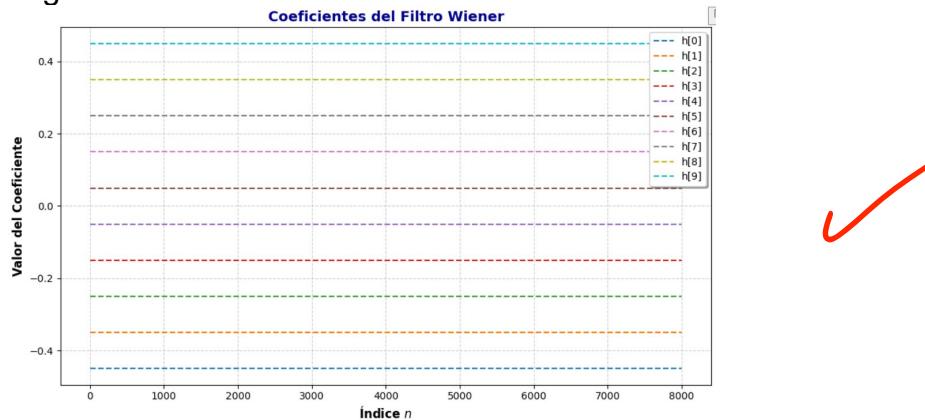
$$\mathbf{R}_x \bar{\mathbf{w}} = \bar{\mathbf{r}}_{dx} \implies \bar{\mathbf{w}}_{\text{opt}} = \mathbf{R}_x^{-1} \bar{\mathbf{r}}_{dx} .$$

```

#BASADO EN EL CODIGO DE LA CLASE
M = 10 # Número de coeficientes del filtro Wiener
N = len(input_lti) # Longitud de la señal de entrada `input_lti`
# Inicializamos la matriz de autocorrelación `Rx` y el vector de autocorrelación `rx`
Rx = np.zeros((M, M)) # Matriz de autocorrelación (M x M)
rx = np.zeros(M) # Vector de autocorrelación (M)
# Calculamos los valores de autocorrelación para el vector `rx`
for k in range(M):
    rx[k] = np.sum(input_lti[:k] * input_lti[:N-k]) / N
# Llenamos la matriz de autocorrelación `Rx` usando el vector `rx`
for i in range(M):
    for j in range(M):
        Rx[i, j] = rx[np.abs(i-j)]
# Calculamos el vector `rdx` para la autocorrelación cruzada entre entrada y salida
rdx = np.zeros(M)
for k in range(M):
    rdx[k] = np.sum(output_lti[k:] * input_lti[:N-k]) / N
# Calculamos los coeficientes del filtro Wiener usando la ecuación matricial
W = np.dot(np.linalg.inv(Rx), rdx)
# Mejoramos el estilo de la gráfica de los coeficientes del filtro Wiener
plt.figure(figsize=(10, 6)) # Tamaño de la figura
for i in range(M):
    plt.plot([0, N], [W[i], W[i]], linestyle="--", label=f'h[{i}]', linewidth=1.5)

```

Obtenemos los siguientes resultados:

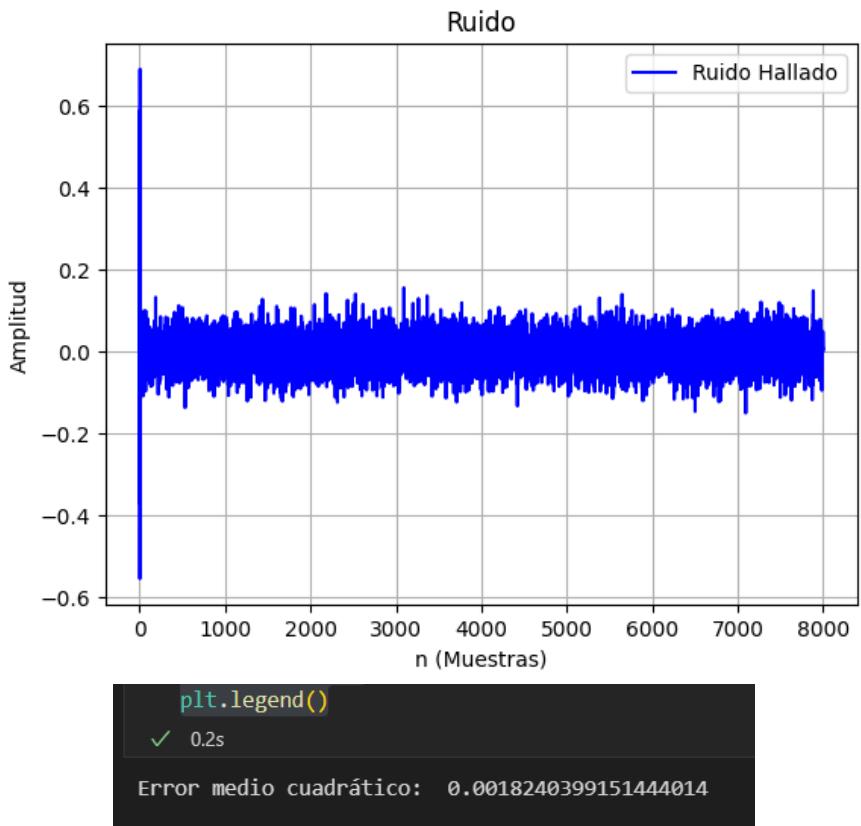


Dado el escenario en el que se tomaron las muestras, $y[n]$ ya se encuentra contaminada por ruido. De forma indirecta, el filtro Wiener al estimar lo más posible la salida ruidosa, se approximó a $y[n]$ solamente y no a $y[n] + \text{ruido}$. Este filtrado del ruido se debe a que el filtro de Wiener opera con las autocorrelaciones y correlaciones cruzadas, como cada muestra del ruido solo está correlacionada con su misma muestra, el filtro Wiener tiende a separar este valor. Para verificar lo antes mencionado, aplicaremos el filtro con los coeficientes hallados para estimar la salida limpia y calcularemos su error. En este caso el error entre el estimado y la señal y (contaminada) debería devolvernos al ruido.

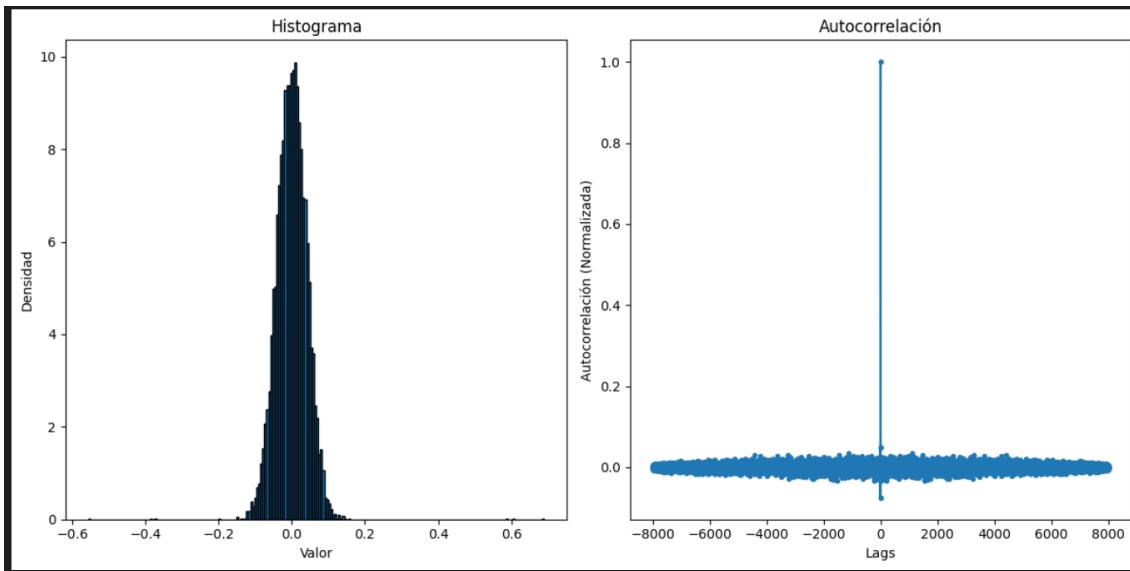
```

salida_aproximada = sc.signal.lfilter(W,[1],input_lti) #Libre de ruido
print('Error medio cuadrático: ',np.mean((output_lti-salida_aproximada)**2))
ruido = salida_aproximada - output_lti
plt.plot(ruido, label='Ruido Hallado', color='blue')
plt.title('Ruido')
plt.xlabel('n (Muestras)')
plt.ylabel('Amplitud')
plt.grid(True)
plt.legend()

```



Ahora para verificar que el error en realidad es ruido blanco analizamos sus estadísticas:

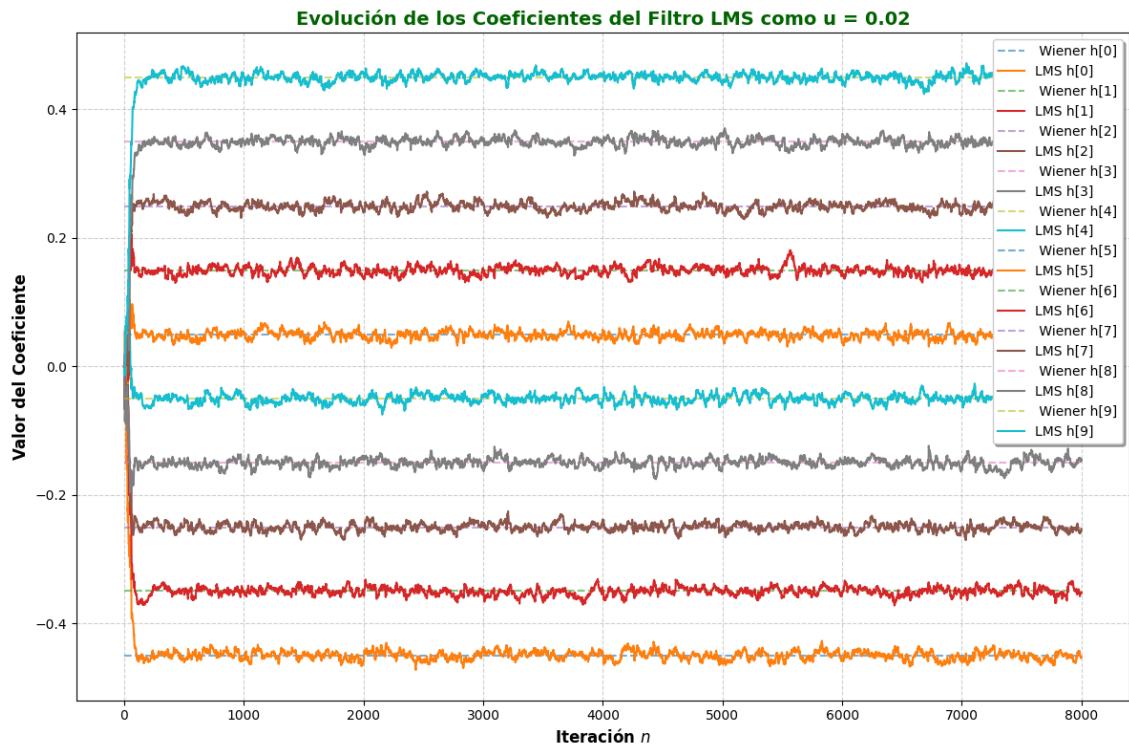


En efecto, el error calculado es del ruido blanco ya que el histograma se aproxima a una Gaussiana y está fuertemente autocorrelacionado (una muestra del error es solo similar consigo misma). Por lo que podríamos afirmar que el filtro de Wiener fue capaz de aproximarse mejor a la señal limpia a pesar de no tenerla previamente. Además, el error calculado antes debería ser menor de lo que en realidad es ya que el error (verdadero) es respecto a la señal limpia y el ruido agrega un error adicional que no debería ser considerado. De todas formas, para el error calculado antes se puede afirmar que tiene un buen performance.

1.2 Análisis usando LMS:

El método LMS al ser un filtro adaptivo, se basa en las muestras obtenidas de tal forma que va estimando mejor el valor de los coeficientes.

```
# Definimos parámetros iniciales
M = 10 # Número de coeficientes del filtro LMS
N = len(input_lti) # Longitud de la señal de entrada `input_lti`
h_LMS = np.zeros((M, N)) # Matriz para almacenar los coeficientes del filtro LMS en cada iteración
d_LMS = np.zeros(N) # Vector para las salidas estimadas del filtro
mu = 0.02 # Tasa de aprendizaje (step size) del algoritmo LMS
xx = np.zeros(M) # Ventana deslizante para almacenar las últimas `M` muestras de entrada
# Iteración para calcular los coeficientes LMS
for n in range(N-1):
    # Desplazamos la ventana de muestras y añadimos la nueva muestra de entrada
    xx = np.roll(xx, 1) # Desplazamos los valores hacia la derecha
    xx[0] = input_lti[n] # Insertamos el nuevo valor al inicio de la ventana
    # Calculamos la salida estimada usando los coeficientes actuales
    d_LMS[n] = np.dot(h_LMS[:, n], xx)
    # Calculamos el error entre la salida real y la estimada
    en = output_lti[n] - d_LMS[n]
    # Actualizamos los coeficientes del filtro LMS
    h_LMS[:, n+1] = h_LMS[:, n] + 2 * mu * en * xx
```



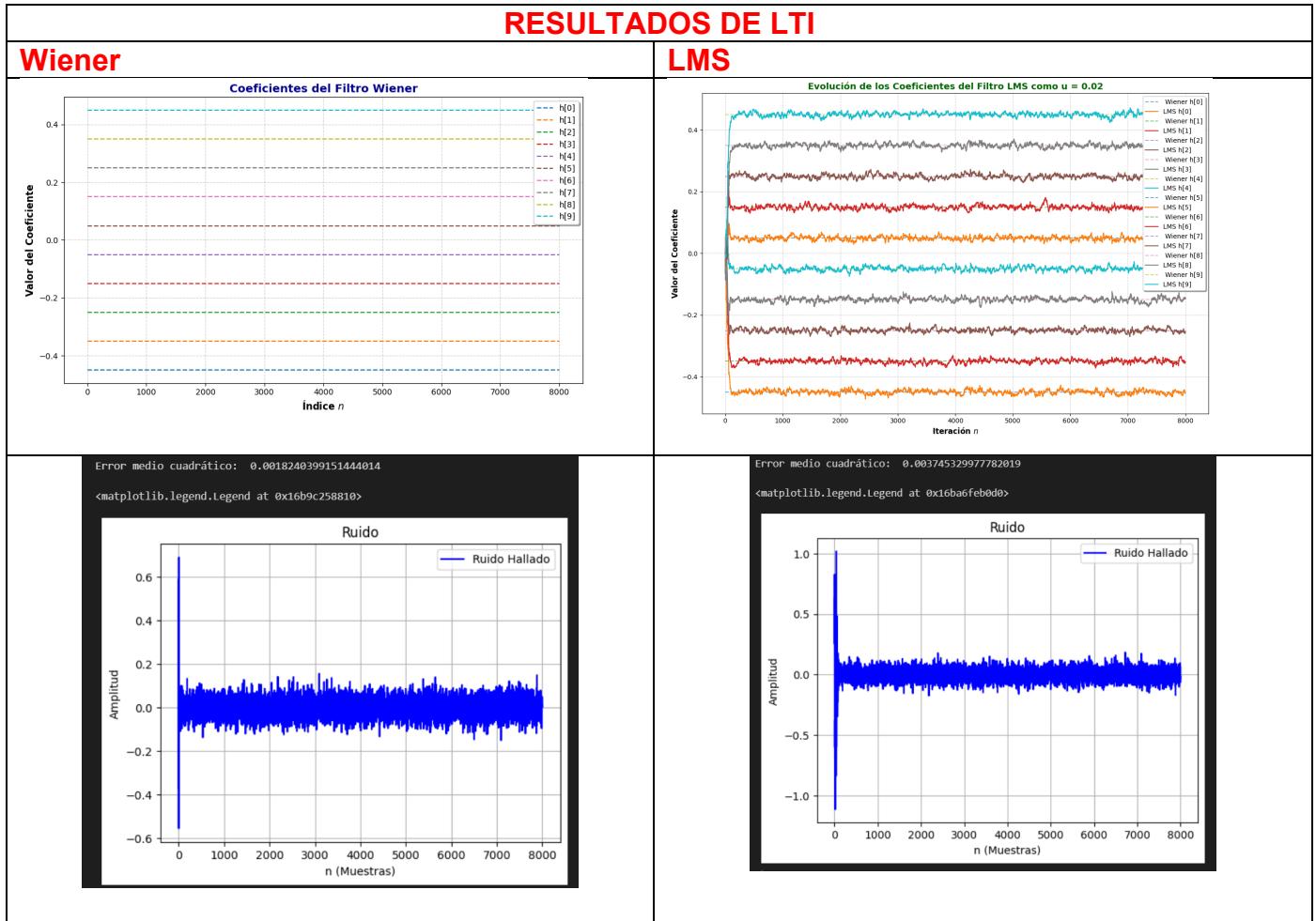
```
#Calcular Error
print('Error medio cuadrático: ',np.mean((output_lti-d_LMS)**2))
ruido_LMS = output_lti - d_LMS
plt.plot(ruido_LMS, label='Ruido Hallado', color='blue')
plt.title('Ruido')
plt.xlabel('n (Muestras)')
plt.ylabel('Amplitud')
plt.grid(True)
plt.legend()
```

0.3s

Error medio cuadrático: 0.003745329977782019

El algoritmo LMS ajusta los valores de los coeficientes basándose en el error provocado en cada iteración y ajustando los valores de los coeficientes basándose en un paso ($u = 0.02$). Por simple inspección de las gráficas podemos determinar que a medida que aumentan las iteraciones el algoritmo LMS

converge a las soluciones de Wiener esto se debe a que Wiener en realidad representa la solución optima a la que debería de llegar el algoritmo LMS; es decir la solución de Wiener es el valor convergente al que llega el algoritmo LMS. En resumen:

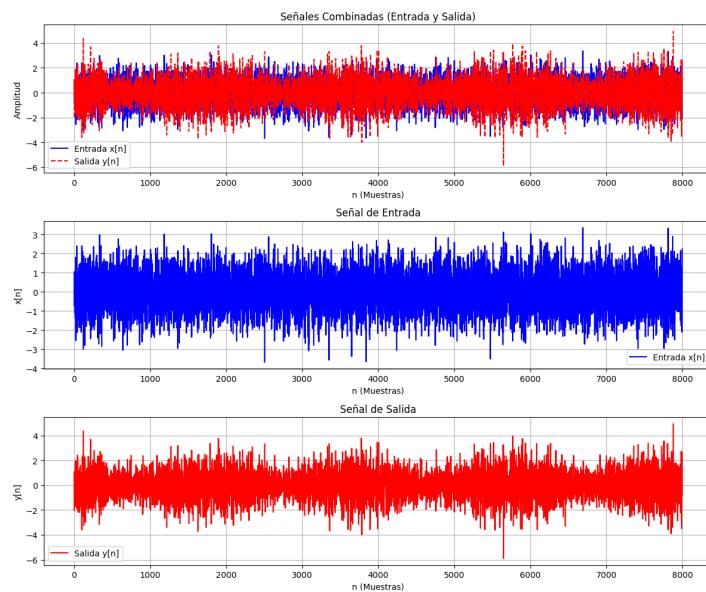


En conclusión, para el sistema LTI, es mejor aproximar su respuesta al impulso usando un filtro Wiener en lugar de LMS porque con Wiener se obtiene menos error lo cual se demostró con las estadísticas mostradas (error Wiener < Error LMS). Este mejor performance de Wiener se debe a que al algoritmo LMS le toma una cierta cantidad de muestras lograr converger a una solución optima y es durante este proceso de convergencia que se llega obtiene un gran error que al promediarse con el resto genere un menor performance comparado a Wiener. Esto también se denota en las gráficas porque para el caso de LMS el ruido (error) inicial captado es mucho mayor que el de Wiener. Cabe mencionar que las estadísticas del error en LMS podrían cambiar dependiendo del tamaño del paso definido; el error podría aumentar si escogemos un paso pequeño (ya que se demoraría mucho en converger) pero si escogemos uno demasiado grande podría no acercarse lo suficiente al óptimo produciendo más error. Por lo tanto, la correcta elección de un tamaño de paso es crítica para evaluar el Throughput del sistema.

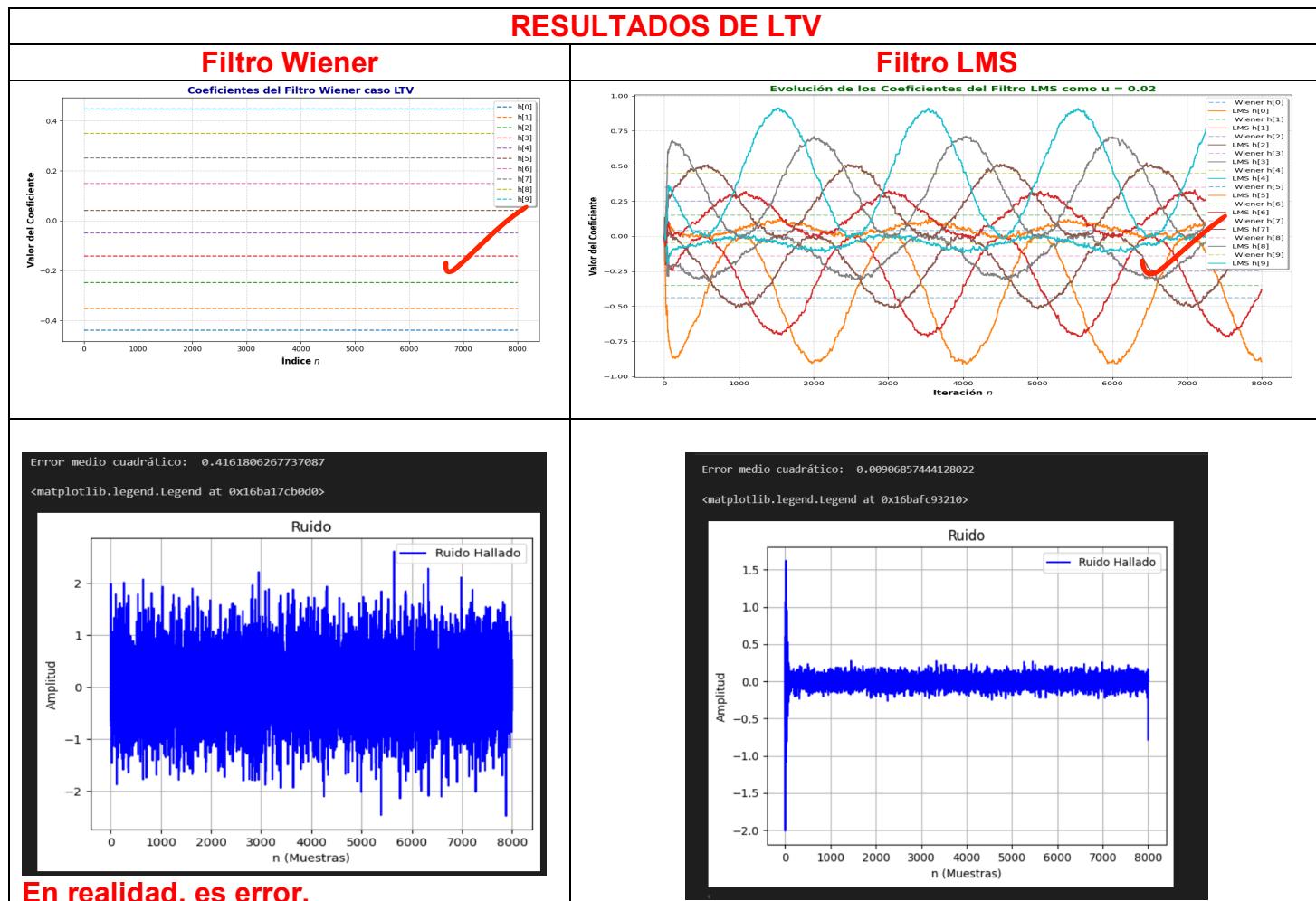
2. Sistema LTV:

3p

Exportamos los datos del archivo CSV:



Obtenemos los resultados de forma similar que en el caso del sistema LTI:



¿Qué puede concluir sobre las características del sistema LTV? ¿Qué opina sobre la performance del algoritmo LMS comparado con la del filtro Wiener?

En el caso de LTV, como la salida depende del momento en el que se registra la entrada (definición de invariancia) ya no permite que el cálculo de los coeficientes óptimos de Wiener sean la mejor solución. Se podría entender que como es un LTV; su respuesta al impulso es variante en el tiempo por lo que los coeficientes que identifican a su sistema también varían (como una prueba de concepto) por lo que al aplicar los coeficientes óptimos de Wiener que son estáticos no son capaces de estimar correctamente la salida del sistema. Esto se evidencia en el cálculo del error para Wiener; obtenemos un error bastante alto. Por lo antes expuesto lo más coherente, es que para aproximarse a un sistema cuyos coeficientes varíen en el tiempo, los del filtro también puedan realizar este cambio, por tanto, conceptualmente esperaría que el LMS tenga mejor performance que el filtro Wiener. Después en los cálculos e implementación podemos corroborar eso, los gráficos de sus coeficientes muestran una variación caótica (no convergen a un valor) a medida que avanza el tiempo; esta variación caótica de los coeficientes es justo lo que permite que los coeficientes del filtro LMS logré una mejor estimación de la señal deseada y se obtenga menor error que con el filtro Wiener.

En conclusión, para un sistema LTV, los filtros LMS tienen un mejor performance que el filtro Wiener, dado que los coeficientes que representen a ese sistema varían con el tiempo por tanto solo un filtro adaptativo podría ser capaz de realizar una aproximación eficaz a las salidas de ese sistema.

PREGUNTA 2: DISEÑO DE FILTROS DIGITALES 7,5p

Se desea diseñar filtros pasabajos para las aplicaciones de radar. Considere que contamos con un sistema de recepción de señales de radar basado en un dispositivo SDR que adquiere datos a una velocidad de muestreo de 100 MHz. La señal digitalizada pasa por un demodulador para llevar la señal a banda base, por un filtro pasabajos y por un decimador para reducir la tasa de muestreo a 1 MHz. Por ende, la frecuencia de corte del filtro a diseñar debe de ser de 1 MHz.

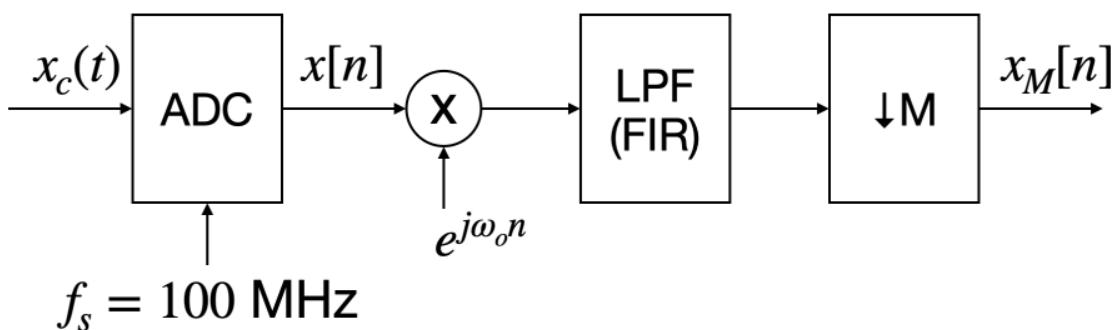


Figura 1: Diagrama de bloques a seguir

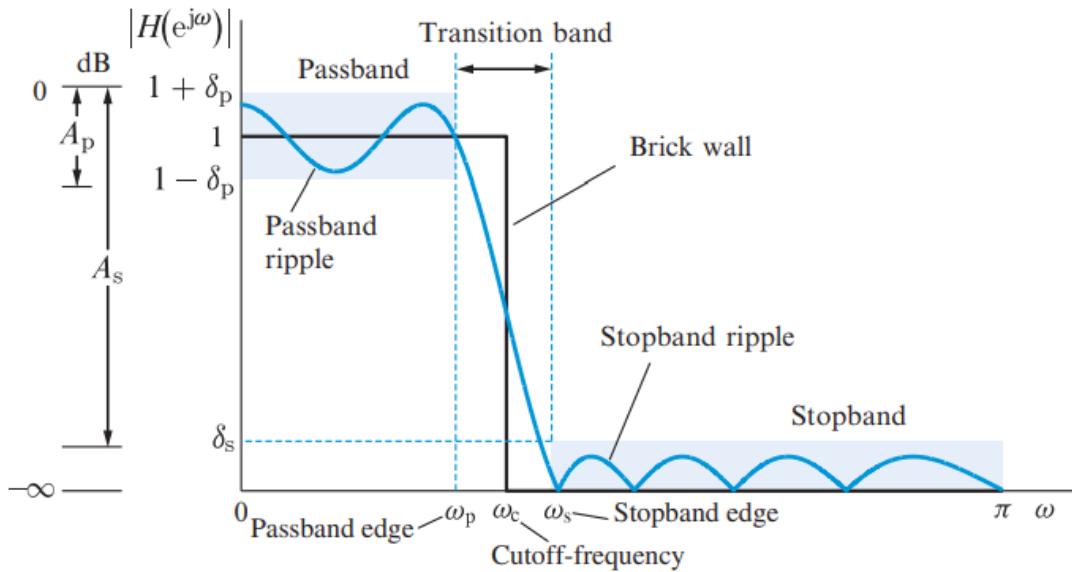


Figura 2: Parámetros a considerar en el diseño del filtro pasabajos

1. Diseño del filtro de Kaiser:

De acuerdo a la bibliografía recomendada, el diseño de un filtro con ventana Kaiser del tipo FIR implica el uso de determinados parámetros que nos permitirán calcular los valores de los coeficientes de la respuesta al impulso del filtro. Para ello debemos emplear la siguiente fórmula:

$$w[n] = \begin{cases} \frac{I_0 \left[\beta \sqrt{1 - [(n - \alpha)/\alpha]^2} \right]}{I_0(\beta)}, & 0 \leq n \leq M \\ 0, & \text{otherwise} \end{cases}$$

Figura 3: Relación entre los coeficientes del filtro Kaiser y los parámetros que definen sus características, extraída de Proakis Manolakis.

Los parámetros dentro de la fórmula nos permiten alterar las distintas propiedades del filtro Kaiser a continuación se explicarán las relaciones y el significado.

>> Parámetro α : Este parámetro permite realizar el desplazamiento en los parámetros de tal forma que lo que se encontraba centrado, luego este lo suficientemente desplazado como para que quede alineado con la forma del filtro ideal inicial del FIR (sinc) y se introduzca correctamente la fase lineal.

$$\alpha = M/2,$$

Donde:

$$A = -20 \log_{10} \delta$$

$$M = \frac{A - 8}{2.285\Delta\omega}.$$

Con delta igual al rizado en la banda de paso.

>> Parámetro β : Este parámetro controla el equilibrio entre el ancho del lóbulo principal y la altura de los lóbulos secundarios en la respuesta en frecuencia del filtro. Un β más alto aumenta la atenuación en la banda de rechazo, reduciendo el ripple fuera de la banda pasante, pero a costa de ensanchar la banda de transición, haciendo el filtro menos abrupto. Por el contrario, un β más bajo reduce el ancho del lóbulo principal, estrechando la banda de transición, pero incrementa el ripple en la banda de rechazo. Esto nos permite personalizar el diseño del filtro según las necesidades de atenuación y banda de transición. **Se determina usando la siguiente fórmula:**

$$\beta = \begin{cases} 0, & A < 21 \\ 0.5842(A - 21)^{0.4} + 0.07886(A - 21), & 21 \leq A \leq 50 \\ 0.1102(A - 8.7), & A > 50 \end{cases}$$

>> Funciones de Bessel de primer orden (I_0) : Estas funciones permiten aproximar, de manera eficiente, las **funciones esferoidales prolatas**, que son soluciones óptimas al problema de maximizar la concentración de energía de una función en el dominio de tiempo y frecuencia. En el contexto del diseño de filtros, esta optimización garantiza un balance adecuado entre el ancho del lóbulo principal (relacionado con la banda de transición) y la altura de los lóbulos secundarios (que afecta la atenuación en la banda de rechazo).

$$I_0(x) = 1 + \sum_{m=1}^{\infty} \left[\frac{(x/2)^m}{m!} \right].$$

Después de todo lo explicado previamente, se concluye que la implementación y uso de filtros con ventana Kaiser es bastante útil para determinar los coeficientes de un filtro FIR que cumpla con características específicas (esto se logra mediante las relaciones antes explicadas).

Entonces implementamos de manera computacional el filtro FIR con ventana kaiser teniendo en cuenta las siguientes características:

Característica	Valor
Atenuación en la banda de rechazo	60 dB
Banda de transición	10% * frecuencia de muestreo (luego de decimación)

Para facilitar los cálculos y determinar los parámetros, usaremos la librería “SCIPY” para los cálculos intermedios, a continuación, se mostrará el código, los resultados y los comentarios de cada sección solicitada:

```

from scipy import signal
# --- Parámetros básicos del filtro ---
sampling_frequency = 100e6 # Frecuencia de muestreo en Hz (100 MHz)
nyquist_frequency = sampling_frequency / 2 # Frecuencia de Nyquist (50 MHz)
cutoff_frequency = 1e6 # Frecuencia de corte en Hz (1 MHz)
transition_bandwidth = 0.08 * cutoff_frequency # Banda de transición (80 kHz)
stopband_attenuation = 60 # Atenuación deseada en la banda de rechazo (60 dB)
# --- Cálculo de parámetros para la ventana Kaiser ---
attenuation_in_db = stopband_attenuation # Guardar la atenuación deseada en dB
normalized_transition_bandwidth = (transition_bandwidth * np.pi) / nyquist_frequency # Ancho de la banda en radianes
# Calcular el parámetro beta de la ventana Kaiser basado en la atenuación
kaiser_beta = signal.kaiser_beta(attenuation_in_db)
# --- Calcular el orden del filtro ---
# Fórmula del orden del filtro FIR utilizando la aproximación de Kaiser
filter_order = int((attenuation_in_db - 8) / (2.285 * normalized_transition_bandwidth))
# Ajustar el orden para que sea ímpar (necesario para un filtro FIR simétrico)
if filter_order % 2 == 0:
    filter_order += 1
# --- Diseño del filtro FIR utilizando la ventana Kaiser ---
# Generar los coeficientes del filtro con la función firwin
fir_coefficients = signal.firwin(
    filter_order, # Orden del filtro calculado
    cutoff_frequency / nyquist_frequency, # Frecuencia de corte normalizada
    window='kaiser', kaiser_beta), # Usar la ventana Kaiser con el parámetro beta
    pass_zero=True # Especifica que es un filtro pasa-bajos
)
# --- Calcular la respuesta en frecuencia del filtro ---
# freqz calcula las frecuencias y las magnitudes de la respuesta del filtro
frequencies, frequency_response = signal.freqz(
    fir_coefficients, # Coeficientes del filtro
    worN=8000 # Número de puntos en el eje de frecuencia para mayor resolución
)
print("-----Parámetros de la ventana Kaiser-----")

```

1) Los resultados obtenidos son:

2,5 P

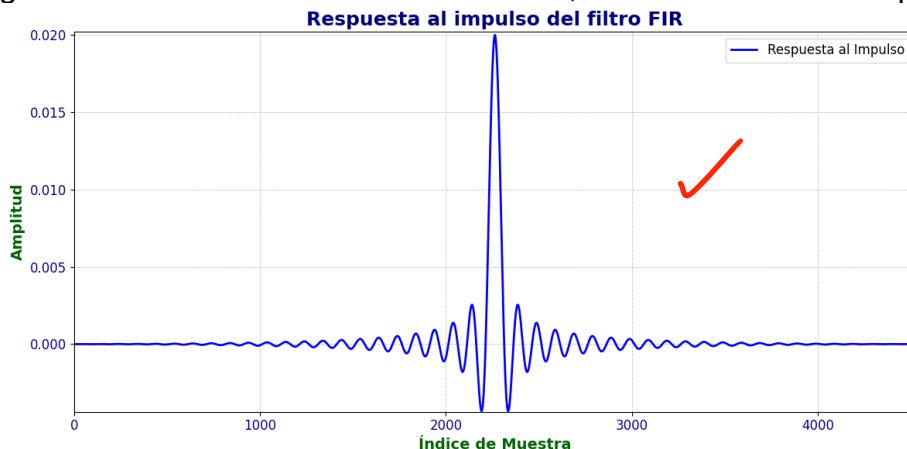
```

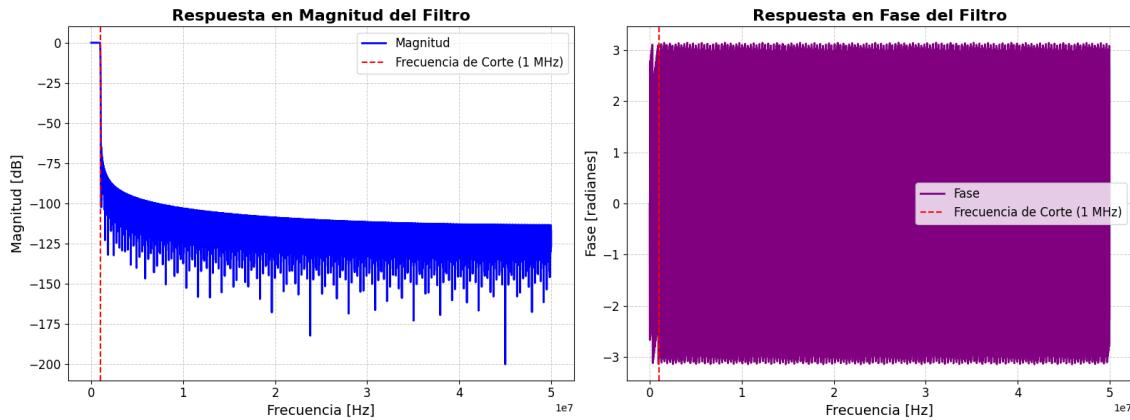
-----Parámetros de la ventana Kaiser-----
La longitud del filtro FIR es de 4527
El valor del Beta del filtro FIR es de 5.65326
Ya que usamos funciones determinadas de la librería de SCIPY , solo fue necesario calcular
beta usando la función signal.kaiser_beta(attenuation_in_db) y calcular la longitud del filtro FIR
Estos valores eran necesarios como argumentos de la función que calcula los coeficientes

```

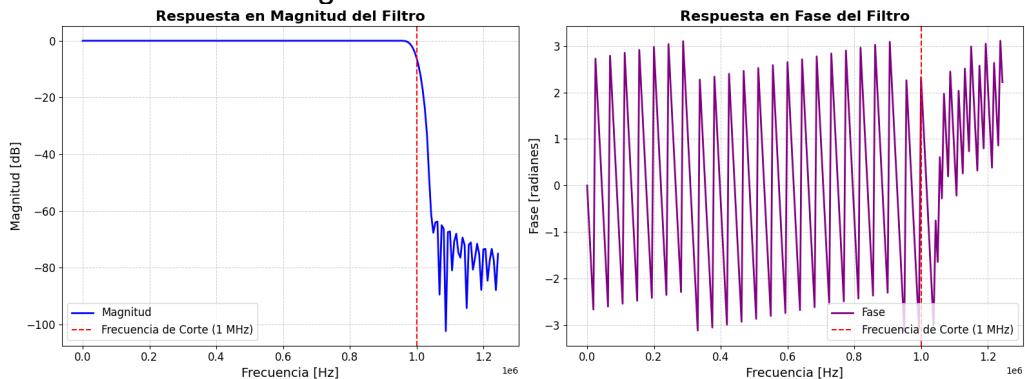
Ahora visualizamos la respuesta al impulso del filtro y su respuesta en frecuencia, para luego verificar que se hayan alcanzado las características solicitadas:

Se plotearon las gráficas correspondientes (el detalle de estos métodos se encuentra en el archivo TA_pregunta2.ipynb) Aunque en parte del último extracto de código se mostró el cálculo de estos valores, lo nuevo solo sería el ploteo.





Realizamos un zoom a la gráfica



Características del filtro:

Parámetros del Filtro

Parámetro	Valor
Longitud del filtro	4527
Rizado en la banda de paso	-7.470078705464153 dB
Rizado en la banda de rechazo	-3.0105816373201466 dB
Atenuación en la banda de rechazo	60 dB
Ancho de banda de transición	100.00 kHz

Comentario:

En general, se consiguieron las características del filtro deseadas respecto a la atenuación y frecuencia de corte; se puede corroborar tanto de manera visual como mediante los cálculos correspondientes. Los parámetros en la configuración de un filtro de Kaiser son los que facilitan la implementación de las características planteadas para esta etapa. Más adelante, se discutirá sobre esto, pero cabe mencionar que dado que se está aplicando un filtro con frecuencia de corte de 1MHz a una señal muestreada de 100MHz; computacionalmente, esto implica la pérdida de componentes frecuenciales (por su atenuación) o también podríamos mencionar que se pierde resolución de la gráfica. Esto lo podemos apreciar en el la gráfica de magnitud y fase con zoom. En el siguiente inciso trataremos de mejorar la resolución mediante la integración

de filtros en cascada y compararemos los resultados de ambas implementaciones.

2. Diseño del filtro usando CIC:

El objetivo de utilizar Filtros en cascada junto a downsamplers es no perder resolución ante un filtrado donde se pierdan muchas muestras en frecuencia. Dado que estamos ante un caso donde filtramos de 100MHz a 1MHz. Deberíamos conseguir ese resultado después de aplicar todas las etapas del filtro.

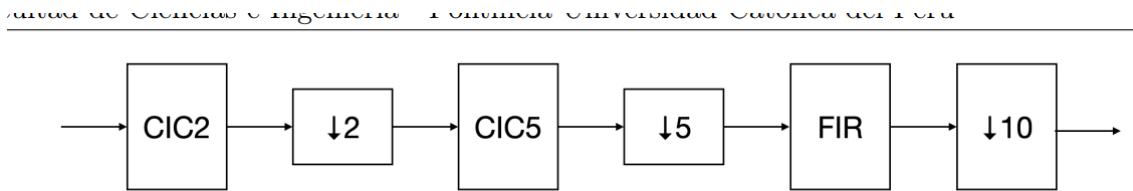
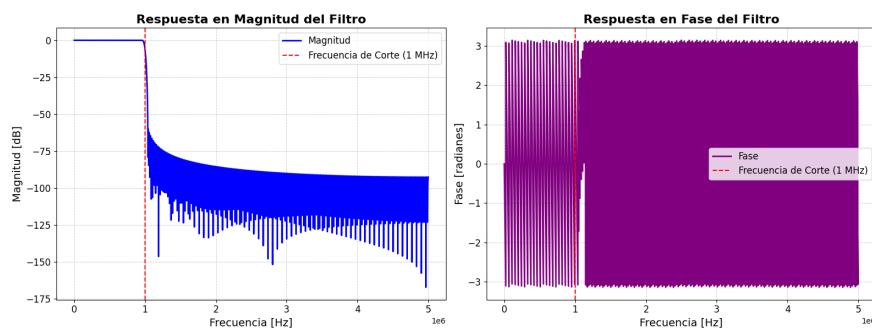
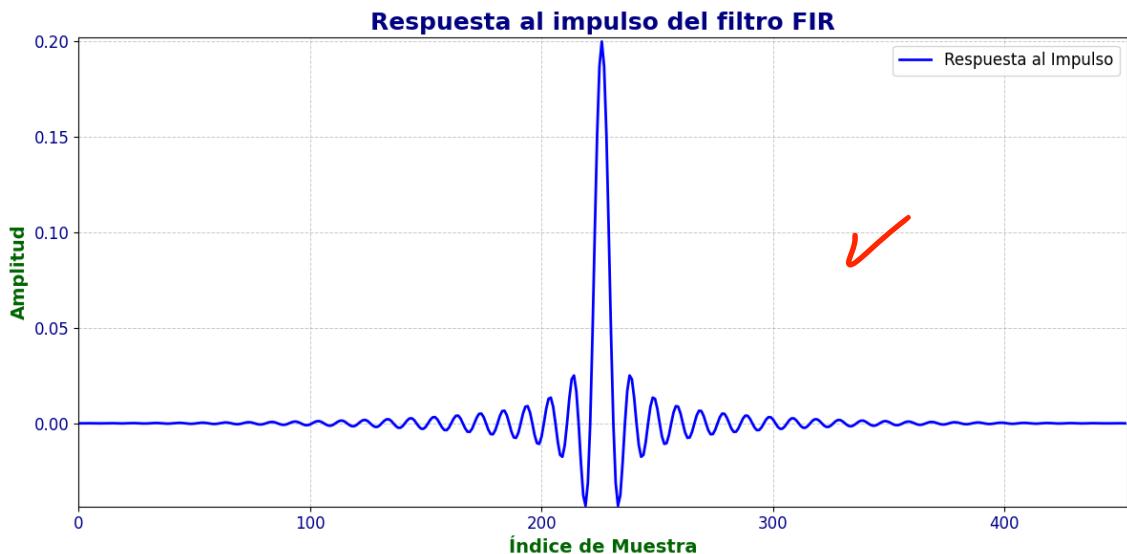


Figura 3: Diagrama de bloques filtros CICs en cascada con filtro FIR.

Para resumir los criterios del diseño y el procedimiento para plantear los algoritmos, CIC2 tendrá una frecuencia de muestreo de 100MHz, CIC5 tendrá una frecuencia de muestreo de 50MHz y el FIR (kaiser) tendrá una frecuencia de muestreo de 10MHz. Para observar ampliamente la respuesta en frecuencia del conjunto de filtros del sistema, replicaremos los espectros de FIR y CIC5 tantas veces sea necesario para poder encajar en el espectro CIC2. De tal forma, podremos analizar si el resultado de aplicar los filtros en conjunto ofrece una mejor resolución que aplicar directamente un filtro con ventana Kaiser.

2.1. Rediseño de Filtro Kaiser:





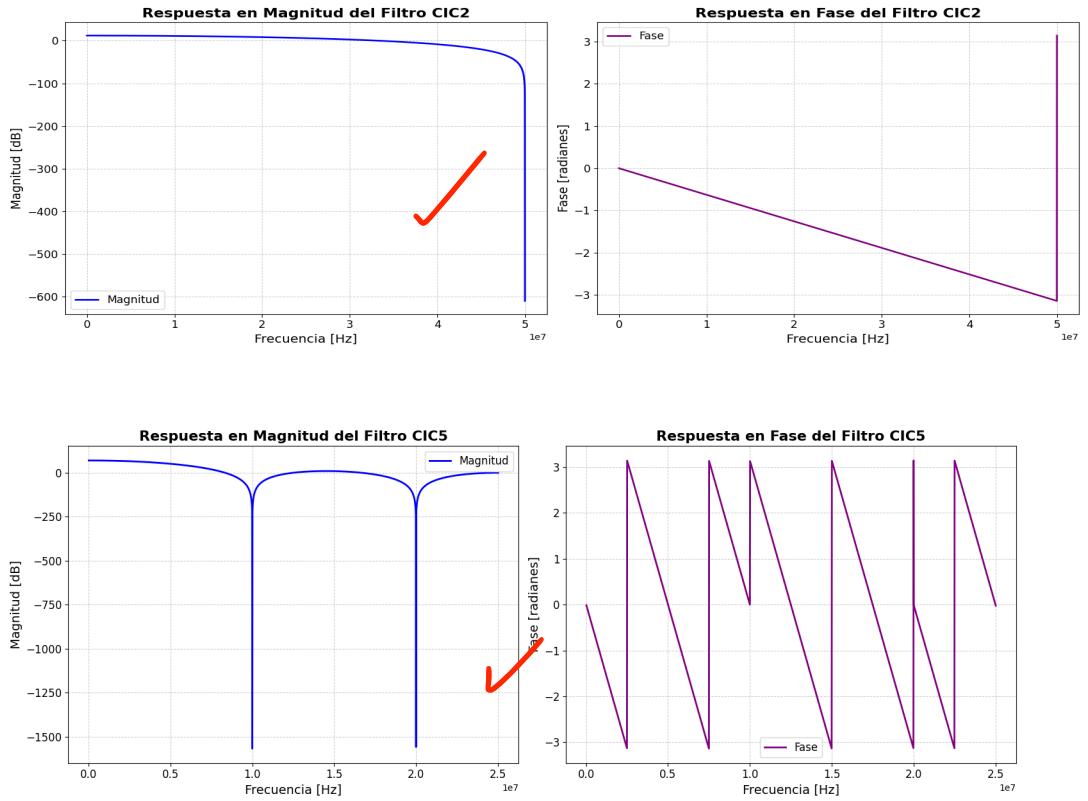
Parámetros del Filtro

Parámetro	Valor
Longitud del filtro	453
Rizado en la banda de paso	-6.139936285728006 dB
Rizado en la banda de rechazo	-3.13854581815531 dB
Atenuación en la banda de rechazo	60 dB
Ancho de banda de transición	100.00 kHz

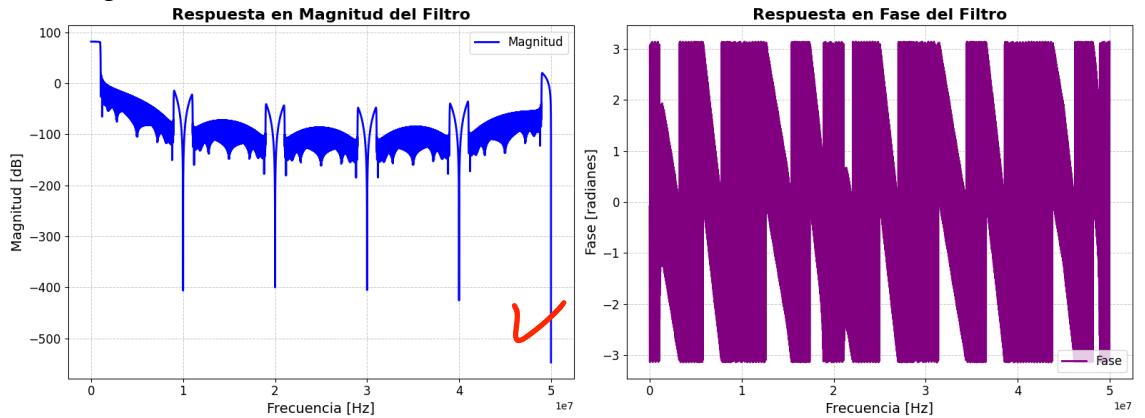
En resumen, seguimos el mismo criterio explicado en la sección 2.1 para el diseño del filtro de Kaiser, pero considerando una frecuencia de muestreo 10MHz, debido a los downsamplers. Cabe notar que, dada la reducción de la frecuencia de muestreo, se redujeron a la decima parte la cantidad de coeficientes del filtro, como se puede observar en la respuesta al impulso y características del filtro nuevo diseñado.

3p 2.2. Diseño de Filtros CIC:

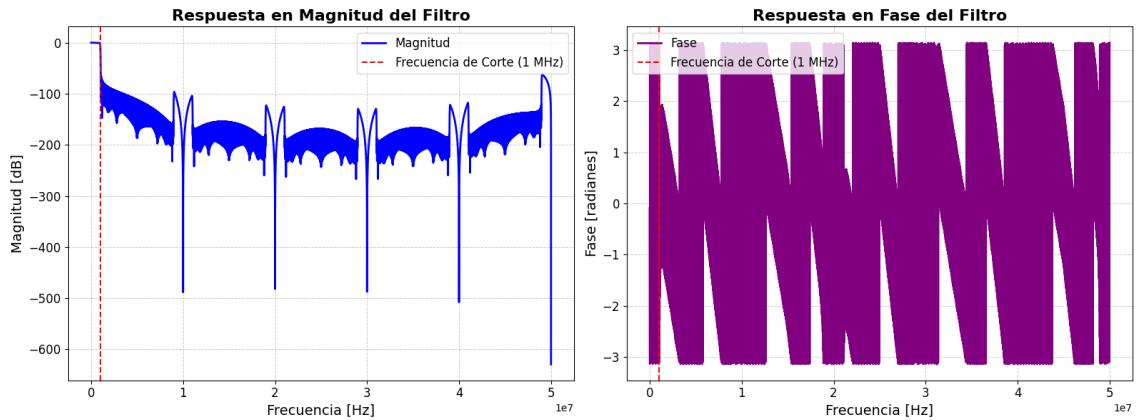
Diseñamos los filtros en cascada utilizando su respuesta en frecuencia directamente reemplazamos $z = \exp(jw)$ en las expresiones y realizamos los gráficos correspondientes, como con los otros filtros. En particular, se aprovecha esta forma de generación para de una vez obtener los valores de CIC5 tantas veces sea necesario para llegar hasta 50MHz para poder combinar las respuestas de los filtros después.



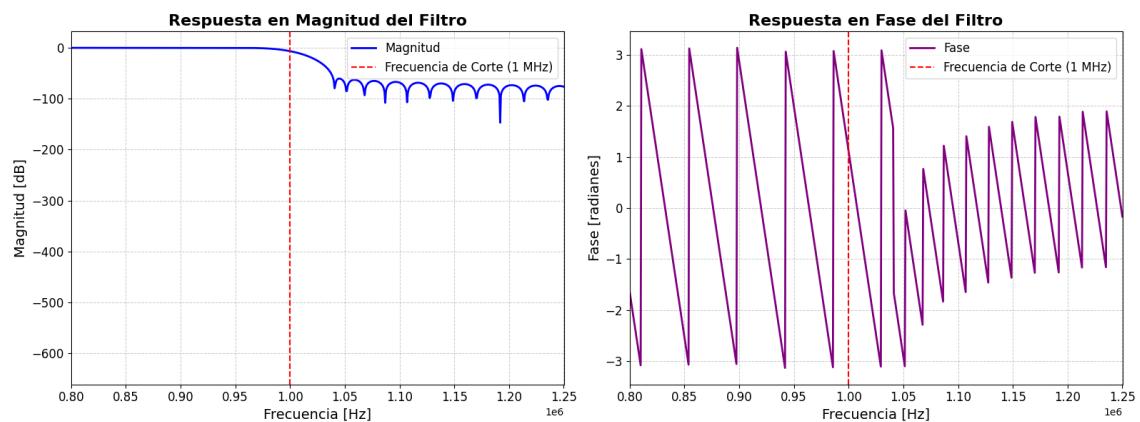
Finalmente, combinamos las respuestas en frecuencia de todos los filtros y obtenemos los siguientes resultados:



Dado que se realizaron downsamplers en distintas etapas de la implementación esto generó una ganancia en los filtros que debe ser corregida para ello se dividió a todas las muestras entre la muestra con el máximo valor y obtuvimos lo siguiente:



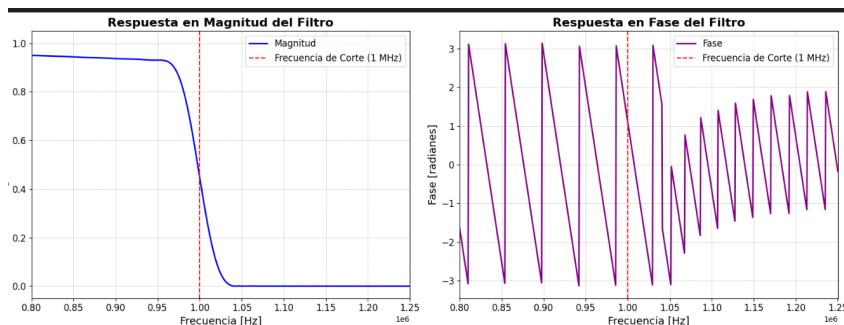
Como se puede apreciar, este filtro ahora ya no posee ganancia en la banda de paso por lo que ya se encuentra corregido.



Al realizar un zoom a las gráficas, se puede afirmar que estas presentan una fase lineal en la banda de paso y sí es notorio que cumple con la frecuencia de corte solicitada (vista en magnitud)

Parámetros del Filtro

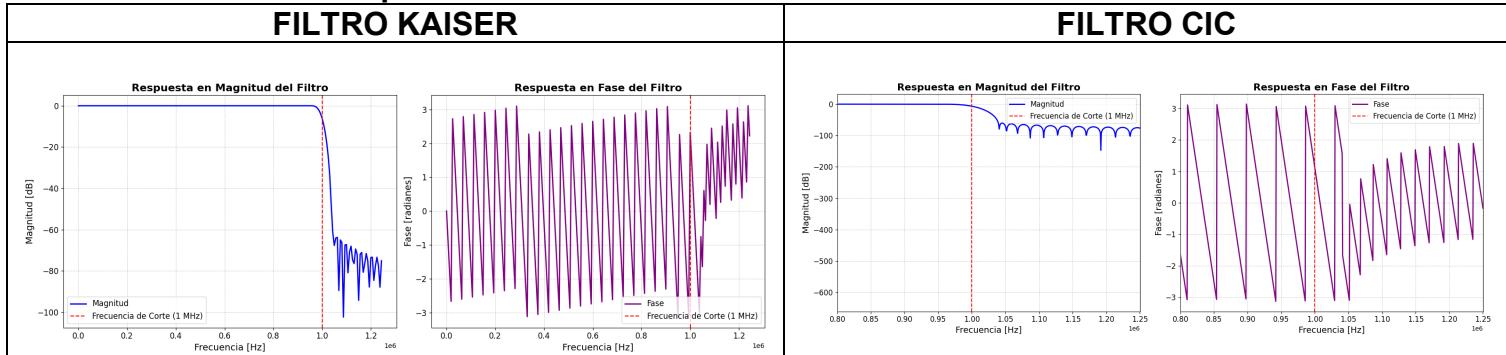
Parámetro	Valor
Rizado en la banda de paso	-1.12 dB
Rizado en la banda de rechazo	-20.123 dB
Atenuación en la banda de rechazo	80 dB



Además, se puede apreciar que el rizado en la banda de rechazo disminuyó.

ZP

2.3. Comparación de resolución:



Podemos apreciar una gráfica del espectro de magnitud y fase con mejor resolución utilizando los filtros en cascada ya que a medida que estos van filtrando se deshacen de muestras de manera progresiva, de cierta forma al actuar de forma selectiva y realizar este tipo de filtrado de forma gradual se logra una mejor claridad en la obtención final de las muestras. Por lo que la implementación con filtros integradores en cascada junto al FIR Kaiser resultan conservar de mejor manera las componentes frecuenciales ante un filtrado con una relación grande entre frecuencia de corte a frecuencia de muestreo en comparación a solo aplicar el filtro de Kaiser.

PREGUNTA 3: Comparación de Algoritmos LMS

4,5P

En esta sección se elaborarán los algoritmos solicitados para el cálculo de los coeficientes del filtro.

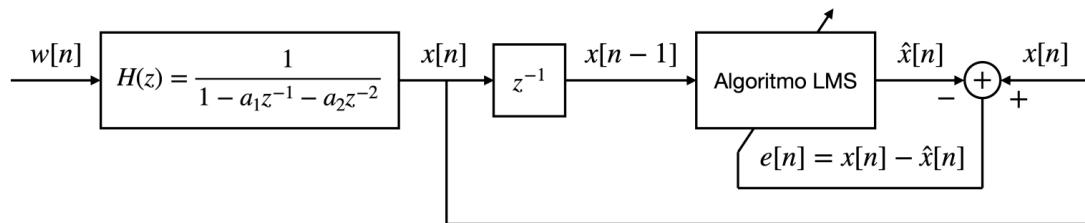
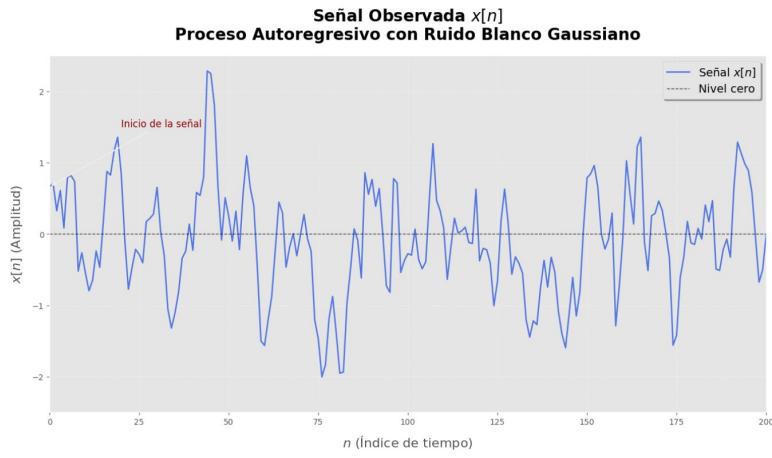


Figura 4: Predictor Lineal

Paso 1: Generar la secuencia del primer filtro que será nuestra data para las entradas al filtro LMS.

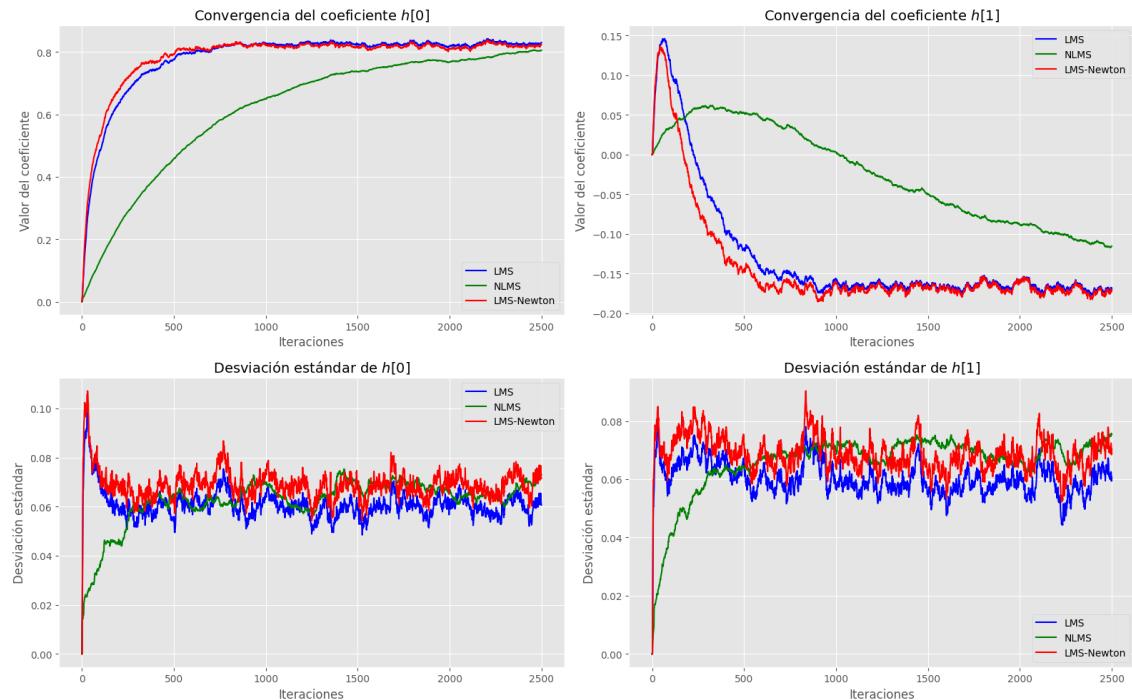
$$x[n] = \sum_{m=1}^M a[m]x[n-m] + w[n],$$



**Vamos directamente al paso 3 , para aplicar las iteraciones de una vez
Paso 3:**

- 2,5 P**
- a) Comparación de la velocidad de convergencia de los algoritmos. Para ello vamos a seleccionar el factor de adaptación de cada algoritmo (μ) tal que la varianza de los coeficientes estimados (calculada usando las 100 realizaciones de cada algoritmo) sea la misma. Grafique los coeficientes promediados en función del índice de tiempo (número de iteración) para cada algoritmo. ¿Qué algoritmo tiene la mayor velocidad de convergencia? Justifique su respuesta

Para lograr esto aplicamos ensayos y errores hasta encontrar valores de paso para los cuales las varianzas son iguales , de tal forma se lograron alcanzar los siguientes resultados:



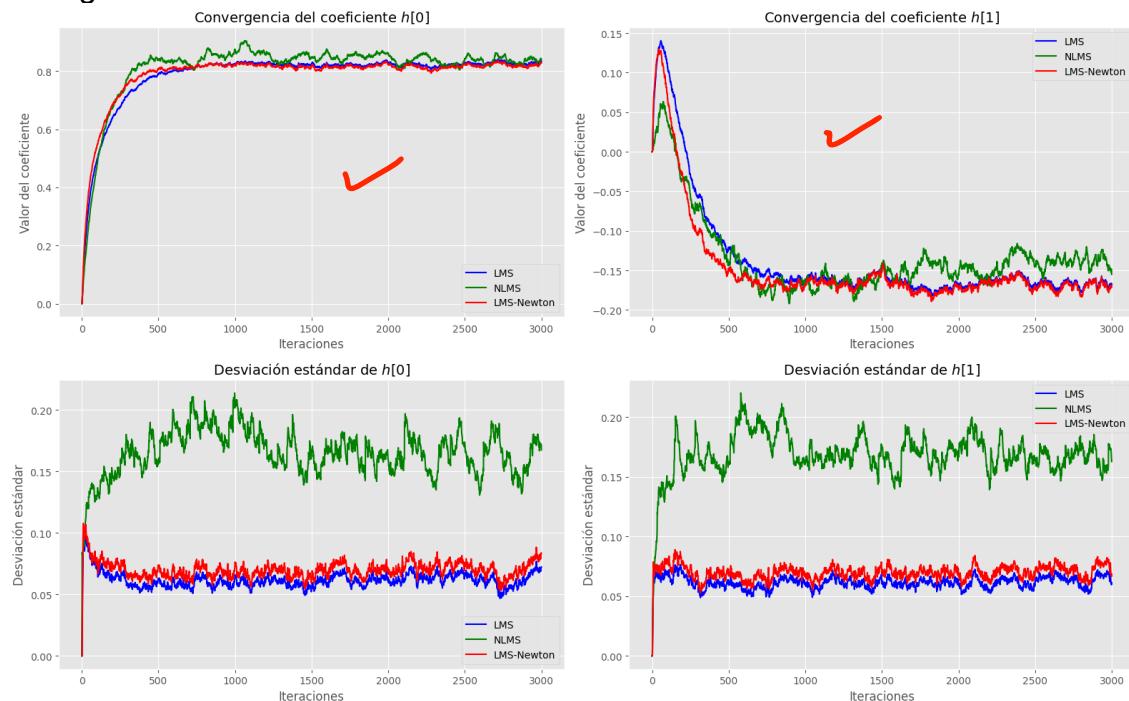
Se usaron los siguientes pasos $\mu_{\text{LMS}} = 0.0161$, $\mu_{\text{NLMS}} = 0.00192$, $\mu_{\text{LMSNewton}} = 0.0211$.

NLMS tiene problemas de implementación.

Por simple inspección podemos determinar que el algoritmo LMS-Newton es el que tiene una mayor velocidad de convergencia ya que tiene una pendiente más pronunciada a nivel de coeficientes, probablemente esto se debe a que es el algoritmo más complejo de los 3 implementados y por ende tiene mayor capacidad de adaptación de cualquier forma es el que de lejos mantiene un mejor performance. Por otro lado, el algoritmo NLMS es el q más tarda en converger , probablemente esto se debe a que no escogí un tamaño de paso correcto que aproveche mejor las virtudes de este algoritmo para mejorar su tiempo de convergencia (En una secuencia más larga se ve su final).

- 2 P** b) Comparación de la varianza de los coeficientes estimados. En este caso vamos a seleccionar el factor de adaptación de cada algoritmo (μ) tal que la velocidad de convergencia de la secuencia promediada (obtenida luego de promediar las 100 realizaciones de cada algoritmo) sea la misma para cada algoritmo. Grafique los coeficientes promediados en función del índice de tiempo para cada algoritmo. ¿Qué algoritmo tiene la mayor varianza de los coeficientes estimados?

Para lograr esto aplicamos ensayos y errores hasta encontrar valores de paso para los cuales las convergencias sean iguales, de tal forma se lograron alcanzar los siguientes resultados:



Se usaron los siguientes pasos $\text{Mu_LMS} = 0.0161$, $\text{Mu_NLMS} = 0.01$, $\text{Mu_LMSNewton} = 0.0211$.

Al analizar las gráficas resultantes, se puede ver que el algoritmo LMS es el que posee menor varianza y el algoritmo NLMS es el q posee mayor varianza. Además LMS-Newton sigue siendo el algoritmo que ofrece un tiempo de convergencia menor.

LMS Newton debería tener la menor varianza en comparación con LMS.
NLMS debería tener una varianza comparable a LMS.