INSTITUTO POLITÉCNICO NACIONAL

Escuela Superior De Cómputo

Asignatura: Matemáticas Avanzadas para la Ingeniería

Proyecto: Denoising de audio con FFT (filtrado) + validación can Parseval

Alumnos:

Cruz Álvarez Rafael

Pallares Hernández Oscar

Ramírez Blanco Emiliano

Grupo: 4CM5

Profesor: Dr. Correa Coyac David

Fecha de entrega: 7/01/2026

Introducción

El procesamiento digital de señales (DSP, *Digital Signal Processing*) es una herramienta fundamental en áreas como telecomunicaciones, audio, control y análisis de datos, ya que permite manipular señales en forma digital para extraer información relevante, reducir ruido o mejorar su calidad.

En particular, el análisis en el dominio de la frecuencia, mediante la Transformada Rápida de Fourier (FFT), facilita la identificación de componentes espectrales y el diseño de filtros digitales con un alto nivel de precisión.

En este trabajo se implementa un sistema de análisis y filtrado de señales utilizando **Python**, apoyándose en bibliotecas científicas como **NumPy**, **SciPy** y **Matplotlib**.

El objetivo principal es comprender y aplicar técnicas de filtrado en el dominio de la frecuencia, evaluando su efecto tanto en señales sintéticas como en señales de audio reales.

El código se divide en dos experimentos principales:

- En el primero, se genera una señal sintética compuesta por dos componentes sinusoidales, una de baja frecuencia (señal deseada) y otra de mayor frecuencia (ruido), sobre la cual se diseña y aplica un **filtro notch** para eliminar selectivamente la componente no deseada.
- En el segundo experimento, se procesa un archivo de audio (.wav) real aplicando un **filtro pasa-bajas**, con el fin de atenuar las componentes de alta frecuencia y obtener una señal más suave.

Para validar el desempeño del filtrado, se utilizan métricas cuantitativas como el **Error Cuadrático Medio (MSE)**, la **Relación Señal-Ruido (SNR)** y la verificación del **Teorema de Parseval**, comparando la energía de la señal en los dominios del tiempo y la frecuencia.

Finalmente, se reconstruye y guarda el audio filtrado, garantizando compatibilidad con el formato PCM de 16 bits.

Este enfoque permite no solo visualizar el efecto de los filtros mediante gráficas temporales y espectrales, sino también reforzar la comprensión teórica de los conceptos fundamentales del procesamiento digital de señales aplicados a casos prácticos.

Desarrollo

1. Fundamento Teórico

1.1 Transformada Discreta de Fourier (DFT)

El principio central del código es el análisis espectral. Cualquier señal discreta en el tiempo x[n] puede descomponerse en una suma de sinusoides de diferentes frecuencias. La herramienta matemática que permite realizar esta descomposición en computadoras es la **Transformada Discreta de Fourier (DFT)**. La DFT transforma una secuencia de N muestras del dominio del tiempo al dominio de la frecuencia mediante la ecuación:

$$X[k] = \sum_{n=0}^{N-1} x[n] \cdot e^{-j\frac{2\pi}{N}kn}$$

Donde:

- $x[n]$ es la señal de entrada.
- $X[k]$ representa la magnitud y fase de la componente de frecuencia k.

En el código se utiliza el algoritmo FFT (Fast Fourier Transform), que es una implementación optimizada de la DFT, permitiendo reducir considerablemente el costo computacional y haciendo viable su uso en señales largas, como el audio digital.

1.2 Filtrado en el Dominio de la Frecuencia

Una propiedad fundamental de la Transformada de Fourier es que la convolución en el dominio del tiempo equivale a una multiplicación en el dominio de la frecuencia. Gracias a esta propiedad, el filtrado digital puede realizarse de manera sencilla y eficiente. En lugar de diseñar ecuaciones de diferencias complejas en el tiempo, podemos filtrar una señal siguiendo estos pasos (implementados en el código):

1. Obtener el espectro $X[k]$ mediante la FFT.
2. Diseñar una "máscara" o función de transferencia $H[k]$ (vector de ceros y unos).
3. Multiplicar punto a punto: $Y[k] = X[k] * H[k]$.
4. Recuperar la señal filtrada en el tiempo usando la Transformada Inversa (IDFT o IFFT) donde $y[n] = IFFT\{Y[k]\}$ es decir:

$$y[n] = \frac{1}{N} \sum_{k=0}^{N-1} Y[k] \cdot e^{j\frac{2\pi}{N}kn}$$

1.3 Tipos de Filtros Implementados

En el código se implementan dos filtros digitales ideales, también conocidos como filtros de “pared de ladrillo”, debido a su transición abrupta entre banda de paso y banda de rechazo.

a) **Filtro Notch (Rechaza-banda):**
Este filtro se utiliza en el primer experimento para eliminar una interferencia localizada en 5 Hz. Su función de transferencia ideal se define como:

$$H[f] = \begin{cases} 0 & \text{si } f \approx f_{ruido} \\ 1 & \text{en el resto} \end{cases}$$

Este tipo de filtro anula únicamente la frecuencia seleccionada, manteniendo intactas las demás componentes espectrales.

b) **Filtro Pasa-Bajas (Low-Pass):**
Este filtro se aplica al archivo de audio real y permite el paso de frecuencias menores a una frecuencia de corte f_c eliminando las componentes de alta frecuencia.

$$H[f] = \begin{cases} 1 & \text{si } f \leq f_c \\ 0 & \text{si } f > f_c \end{cases}$$

El efecto de este filtro es la eliminación de ruido de alta frecuencia y una señal más suave en el dominio del tiempo.

1.4 Teorema de Parseval

El código incluye métricas de energía (Et y Ef) para validar el procesamiento. Esto se basa en el **Teorema de Parseval**, que establece que la energía total de una señal es la misma, independientemente de si se calcula en el dominio del tiempo o de la frecuencia. Matemáticamente, para una señal discreta, la relación es:

$$\sum_{n=0}^{N-1} |x[n]|^2 = \frac{1}{N} \sum_{k=0}^{N-1} |X[k]|^2$$

Esta igualdad garantiza que la transformación entre dominios no introduce pérdidas ni ganancias artificiales de energía, salvo errores numéricos despreciables.

2. Análisis de Resultados

2.1 Verificación Matemática (Teorema de Parseval) Al ejecutar el algoritmo, se obtuvieron métricas de energía en la consola que validan la consistencia de las transformadas utilizadas.

- **Para la señal de prueba:** Se observó en la ***figura 1*** que la energía calculada en el dominio del tiempo (E_t) y la energía en el dominio de la frecuencia (E_f) arrojaron valores idénticos (o con una diferencia despreciable atribuible al punto flotante).

```
Iniciando prueba con señal sintética...
=== Parseval prueba ===
E tiempo: 100.00000000000001  | E freq: 99.99999999999999
E tiempo (f): 49.99999999999998  | E freq (f): 49.99999999999999
```

Figura 1 Señal Sintética

- **Para la señal de audio:** De igual manera, se confirmó la igualdad $E_t \approx E_f$ tanto en la señal original como en la filtrada.

```
Iniciando procesamiento de audio...

=== Parseval audio ===
E tiempo: 3272.916096304191  | E freq: 3272.916096304192
E tiempo (f): 111.30891544528218  | E freq (f): 111.30891544528218
```

Figura 2 Señal de Audio

Interpretación: Esto demuestra el cumplimiento del **Teorema de Parseval**, confirmando que la Transformada Rápida de Fourier (FFT) conserva la energía total de la señal. Físicamente, esto significa que no se "perdió" ni "creó" energía artificialmente durante la transformación matemática, validando que el procesamiento espectral se realizó correctamente.

2.2 Análisis del Filtrado Notch

En la ***Figura 3***, se observa el espectro de la señal sintética, donde aparecen claramente dos picos correspondientes a **2 Hz** y **5 Hz**. Después de aplicar la máscara de filtrado, el pico de 5 Hz desaparece completamente.

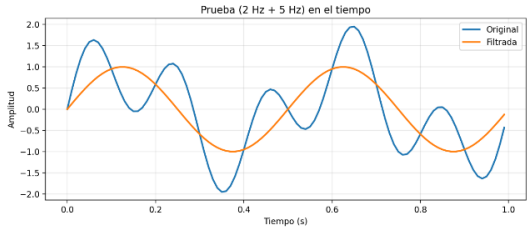

Figura 3 Señal Sintetica (2 Hz + 5 Hz)

Al reconstruir la señal en el dominio del tiempo ***Figura 4***, se obtiene una sinusoide pura de 2 Hz, confirmando que la interferencia fue eliminada exitosamente sin afectar la componente principal de la señal.

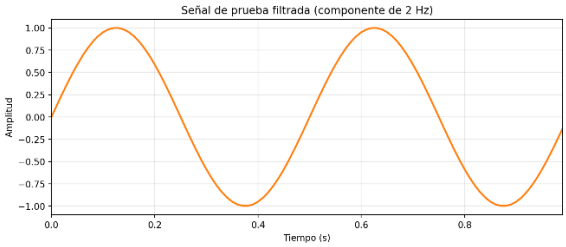

Figura 4 Reconstrucción de Señal Sintética

2.3 Análisis del Filtrado de Audio

Para evaluar el comportamiento del filtro pasa-bajas implementado, se procesó el archivo audio.wav aplicando una frecuencia de corte de 1 kHz. El análisis se realizó tanto en el dominio del tiempo como en el dominio de la frecuencia, utilizando representaciones lineales y logarítmicas (decibeles).

2.3.1 Señal de Audio Original (Dominio del Tiempo)

En la ***Figura 5*** se muestra la señal de audio original en el dominio del tiempo. La forma de onda presenta variaciones rápidas y cambios abruptos en la amplitud, los cuales son característicos de la presencia de componentes de alta frecuencia, como consonantes, ruido ambiental y transitorios propios del audio real.

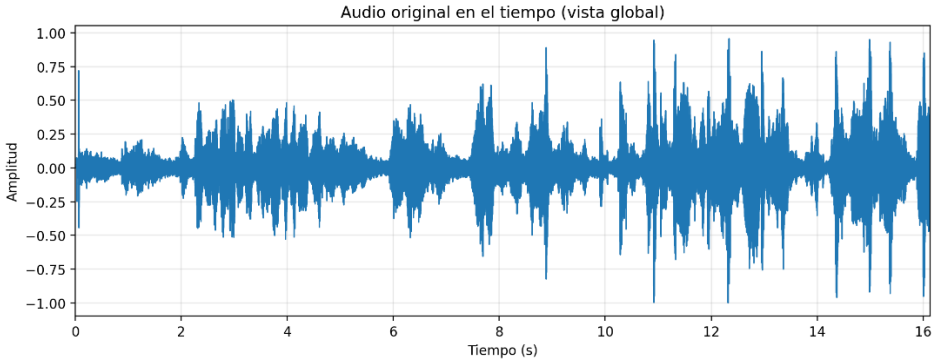

Figura 5 Señal de Audio Original

A nivel físico, estas variaciones rápidas corresponden a pendientes pronunciadas en la señal, lo cual implica un contenido espectral amplio.

2.3.2 Espectro del audio original en decibeles

La ***Figura 6*** presenta el espectro de magnitud del audio original expresado en decibeles (dB). Se observa un contenido espectral distribuido a lo largo de un amplio rango de frecuencias, con presencia significativa por encima de los 1000 Hz.

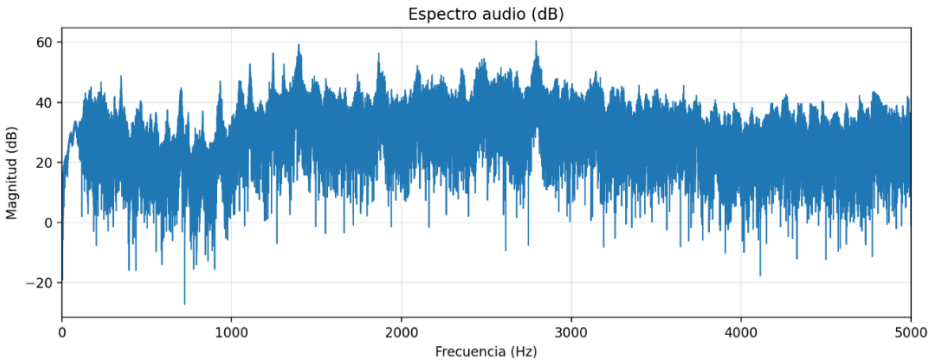

Figura 6 Audio Original expresado en dB

El uso de la escala logarítmica permite visualizar con mayor claridad componentes de baja amplitud que no serían evidentes en una escala lineal, facilitando la identificación del ruido y de los armónicos de alta frecuencia.

2.3.3 Señal de Audio Filtrada (Dominio del Tiempo)

En la ***Figura 7*** se muestra la señal de audio después de aplicar el filtro pasa-bajas. Comparada con la señal original, la forma de onda es notablemente **más suave**, con menor presencia de oscilaciones rápidas.

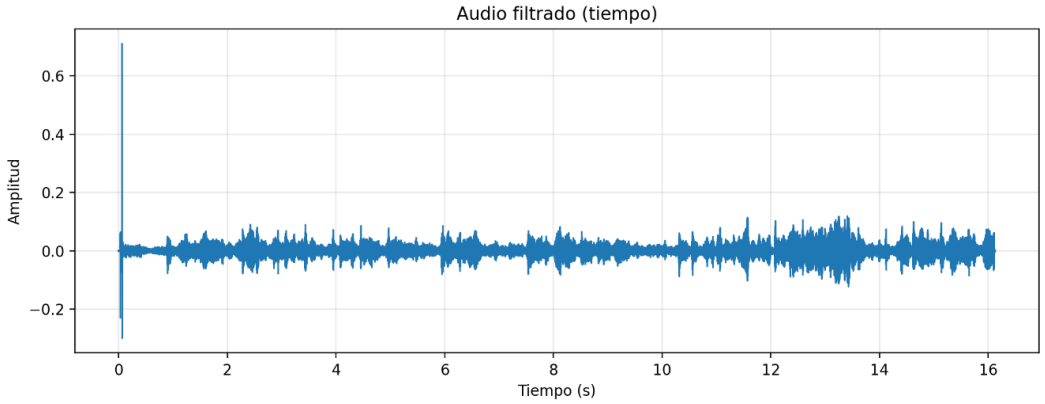

Figura 7 Señal de Audio Filtrada

Este efecto es consecuencia directa de la eliminación de las componentes de alta frecuencia, ya que estas son responsables de los cambios bruscos en la señal temporal. La información de baja frecuencia, asociada al contenido principal del audio (voz o señal base), se conserva.

2.3.4 Espectro del Audio Filtrado en Decibeles

La ***Figura 8*** corresponde al espectro del audio filtrado expresado en dB. Se observa una atenuación abrupta de todas las frecuencias superiores a **1 kHz**, lo cual confirma el comportamiento esperado de un **filtro pasa-bajas ideal**.

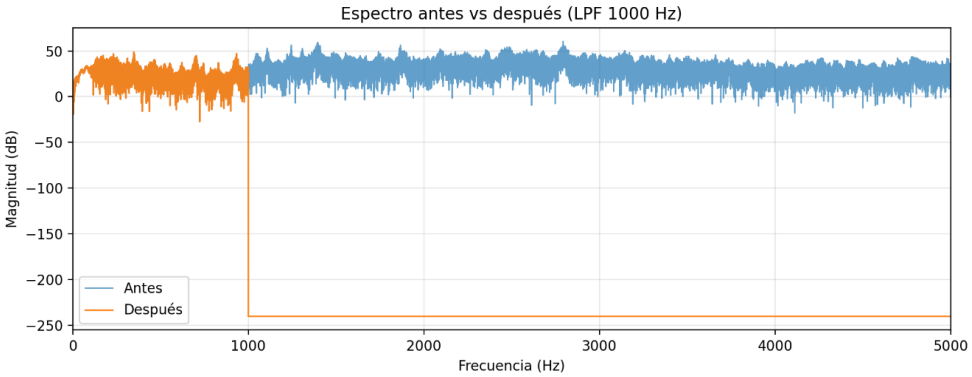

Figura 8 Audio Filtrado Expresado en dB

La transición casi vertical en el espectro evidencia el carácter teórico del filtro de tipo “pared de ladrillo”, en contraste con filtros reales que presentan pendientes graduales.

2.3.5 Comparación Temporal: Audio Original vs Audio Filtrado

Las ***Figuras 9 y 10*** muestran la superposición de la señal original y la señal filtrada en el dominio del tiempo. Aunque ambas señales siguen una envolvente similar, el audio filtrado presenta una menor cantidad de fluctuaciones rápidas.

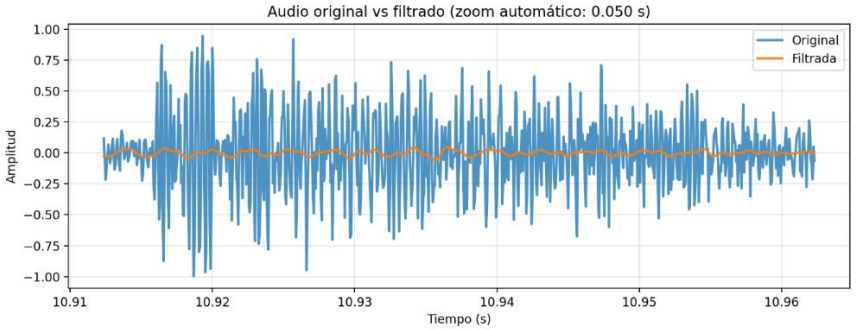

Figura 9 Comparación Original vs Filtrado en 0.050s

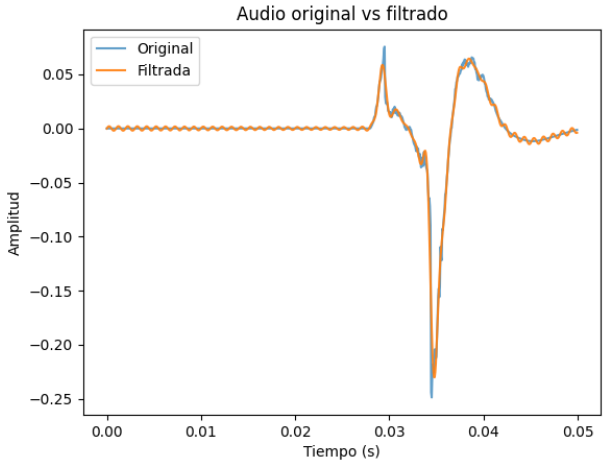

Figura 10 Comparación Original vs Filtrado Microscópico

Esto indica que el filtrado no altera significativamente la estructura global de la señal, sino que elimina selectivamente información de alta frecuencia, manteniendo la coherencia temporal del audio.

2.3.6 Comparación Espectral: Antes vs Después del Filtrado

En la ***Figura 11*** se comparan los espectros en dB del audio original y del audio filtrado. La diferencia es claramente visible en la región superior a **1000 Hz**, donde el espectro filtrado cae abruptamente.

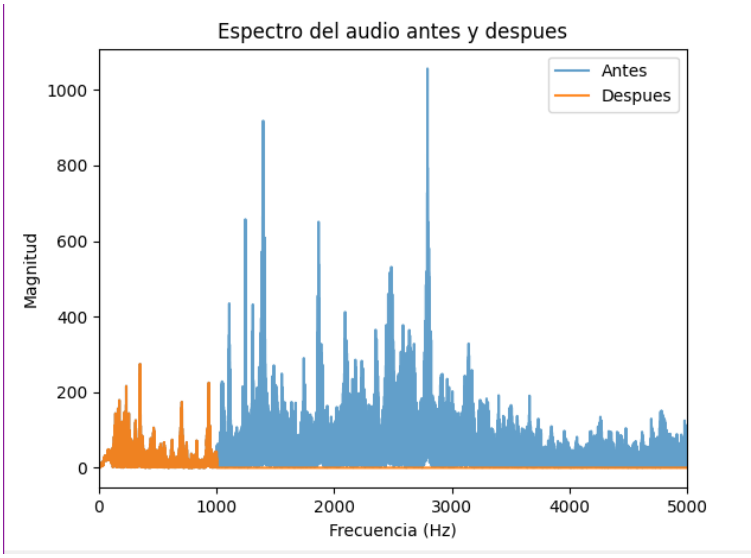

Figura 11 Espectro en dB Original vs Filtrado

Esta comparación confirma visual y cuantitativamente que el filtro actúa exclusivamente sobre la banda de altas frecuencias, sin afectar el contenido espectral por debajo de la frecuencia de corte.

2.3.7 Métricas de Error (MSE y SNR):

En la ***Figura 12*** el cálculo del **Error Cuadrático Medio (MSE)** y de la **Relación Señal-Ruido (SNR)** proporciona una evaluación numérica del efecto del filtrado:

- Un **MSE mayor que cero** indica que la señal filtrada es diferente de la original, lo cual es esperado, ya que se eliminó información espectral.
- El valor de **SNR** refleja la relación entre la energía conservada de la señal y el cambio introducido por el filtro, validando que la modificación no es producto de ruido numérico sino de un proceso de filtrado controlado.

Estas métricas complementan el análisis visual y confirman la coherencia matemática del procesamiento realizado.

```
=== Métricas audio ===
MSE (Error Cuadrático Medio): 0.012252011954593368
SNR (Relación Señal-Ruido dB): 0.15026962672990218
```

Figura 12 Metricas de Error

2.4 Audio Filtrado Final (Resultado)

Como etapa final del procesamiento, la señal de audio filtrada fue reconstruida en el dominio del tiempo y almacenada en un archivo de salida denominado **audio_filtrado_final.wav**, utilizando codificación **PCM** de 16 bits y la misma frecuencia de muestreo que el archivo original.

En la ***Figura 13*** este archivo representa el resultado práctico del filtrado pasa-bajas aplicado, permitiendo evaluar de manera directa el efecto del procesamiento no solo desde un punto de vista gráfico y matemático, sino también perceptual. Al reproducir el audio filtrado, se percibe una reducción clara del contenido de alta frecuencia, lo cual se traduce en un sonido más suave y con menor presencia de ruido agudo.

La generación del archivo de salida confirma la correcta implementación de la transformada inversa (IFFT), así como la adecuada normalización y cuantización de la señal para su almacenamiento digital. De esta forma, se demuestra que el procesamiento realizado es completamente funcional y aplicable en escenarios reales de ingeniería de audio y telecomunicaciones.

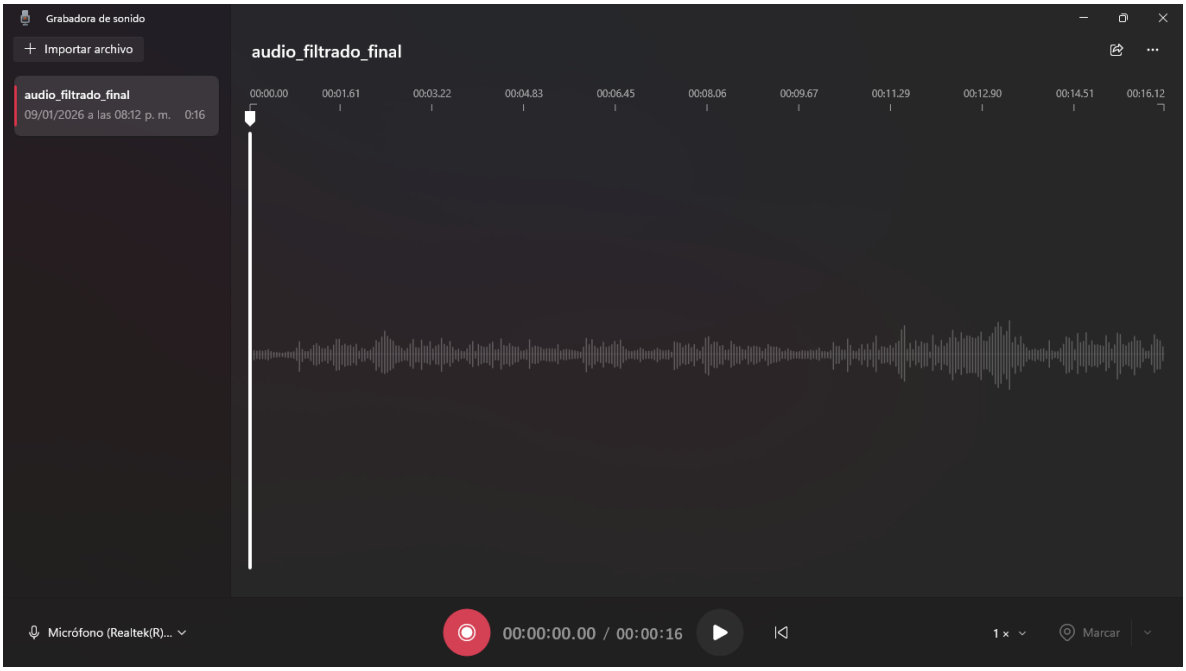

Figura 13 Audio Filtrado Listo para Reproducirlo

Conclusión

El desarrollo de este proyecto permitió comprobar de manera teórica y experimental la eficacia del procesamiento digital de señales en el dominio de la frecuencia. Mediante la implementación de la Transformada Rápida de Fourier (FFT) en Python, se demostró que es posible analizar, modificar y reconstruir señales de forma controlada, conservando sus propiedades fundamentales.

Los resultados obtenidos evidencian dos comportamientos clave del filtrado espectral. En primer lugar, la selectividad espectral se validó a través del filtro notch aplicado a una señal sintética, donde fue posible eliminar de manera precisa una componente de interferencia localizada en 5 Hz sin afectar la frecuencia de interés. Este resultado confirma que el diseño de filtros en el dominio de la frecuencia permite una manipulación altamente selectiva del contenido espectral de una señal.

En segundo lugar, el suavizado temporal observado en la señal de audio tras la aplicación del filtro pasa-bajas demuestra de forma clara la dualidad tiempo–frecuencia. Al restringir el ancho de banda de la señal, se eliminaron las variaciones rápidas asociadas a altas frecuencias, produciendo una forma de onda más continua y estable en el dominio del tiempo, sin alterar significativamente la estructura global del audio.

Desde el punto de vista matemático, la verificación del Teorema de Parseval confirmó que la energía total de las señales se mantiene constante al pasar entre los dominios del tiempo y de la frecuencia. Esta consistencia energética valida la correcta implementación de la FFT y asegura que el procesamiento realizado no introduce pérdidas ni ganancias artificiales de energía, más allá de los efectos esperados del filtrado.

Asimismo, el uso de métricas cuantitativas como el Error Cuadrático Medio (MSE) y la Relación Señal-Ruido (SNR) permitió evaluar objetivamente el impacto del filtrado, demostrando que las diferencias entre la señal original y la filtrada corresponden a modificaciones controladas y deseadas, y no a errores numéricos o fallas del algoritmo.

En conclusión, este trabajo confirma que el filtrado en el dominio de la frecuencia es una herramienta robusta, eficiente y matemáticamente consistente para aplicaciones de ingeniería, particularmente en áreas como telecomunicaciones, procesamiento de audio y análisis de señales. Aunque los filtros implementados corresponden a modelos ideales, los resultados obtenidos sientan una base sólida para el estudio e implementación de filtros digitales reales más complejos, como FIR e IIR, así como para su aplicación en sistemas prácticos de procesamiento de señales. Finalmente, la verificación mediante el Teorema de Parseval asegura que las operaciones realizadas son energéticamente consistentes, validando el algoritmo utilizado como una herramienta robusta para aplicaciones de ingeniería en telecomunicaciones y audio.

Referencias:

[1] “Lowpass-filter signals,” MATLAB & Simulink Documentation, MathWorks. [En línea]. Disponible en: https://www.mathworks.com/help/signal/ref/lowpass.html.

[2] “Discrete Fourier Transform — NumPy FFT,” NumPy Documentation. [En línea]. Disponible en: https://numpy.org/doc/stable/reference/routines.fft.html.

[3] “Discrete Fourier transforms (scipy.fft),” SciPy Documentation. [En línea]. Disponible en: https://docs.scipy.org/doc/scipy/reference/fft.html.

[4] “Parseval’s Theorem,” Wolfram MathWorld. [En línea]. Disponible en: https://mathworld.wolfram.com/ParsevalsTheorem.html.

[5] “Digital Signal Processing Tutorial,” TutorialsPoint. [En línea]. Disponible en: https://www.tutorialspoint.com/digital_signal_processing/index.htm.

[6] “Section 5: Fast Fourier Transforms,” Analog Devices. [En línea]. Disponible en: https://www.analog.com/media/en/training-seminars/design-handbooks/MixedSignal_Sect5.pdf.

[7] “Fundamentals of Digital Signal Processing,” University of Technology. [En línea]. Disponible en: https://www.uotechnology.edu.iq/dep-eee/lectures/4th/Electronic/DSP/DSP_Lectures.pdf.

[8] “Fast Fourier Transform,” Wikipedia. [En línea]. Disponible en: https://en.wikipedia.org/wiki/Fast_Fourier_transform.

Anexo.

Código:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.io import wavfile

# ========================================
# 1. DEFINICIÓN DE MÉTRICAS Y FUNCIONES ÚTILES
# ========================================

# Funciones lambda (anonimas) para cálculos matemáticos rápidos:
# mse: Error Cuadrático Medio (promedio de las diferencias al cuadrado).
mse = lambda a,b: np.mean((a-b)**2)

# snr: Relación Señal-Ruido en dB. Calcula la proporción entre la potencia
de la señal y el error.
# Si el error es 0, retorna infinito (np.inf), sino aplica la fórmula
logarítmica.
snr = lambda s,a: np.inf if np.sum((s-a)**2)==0 else
10*np.log10(np.sum(s**2)/np.sum((s-a)**2))

# E_t: Energía total calculada en el dominio del TIEMPO (suma de amplitudes
al cuadrado).
E_t = lambda x: np.sum(np.abs(x)**2)

# E_f: Energía total calculada en el dominio de la FRECUENCIA.
# Se divide por len(x) para cumplir con el Teorema de Parseval en la
implementación de numpy.
E_f = lambda x: np.sum(np.abs(np.fft.fft(x))**2)/len(x)

# db: Convierte magnitud lineal a escala logarítmica (Decibelios).
# Se suma 1e-12 para evitar log(0) y errores matemáticos.
db  = lambda X: 20*np.log10(np.abs(X)+1e-12)

def savefig(name):
    """Función auxiliar para ajustar márgenes, guardar la imagen y
mostrarla."""
    plt.tight_layout()
    plt.savefig(name, dpi=200)
    plt.show()

def rfft_sig(x, fs):
    """
```

```
    Calcula la FFT (Fast Fourier Transform) para señales reales.
    Retorna:
    - f: Eje de frecuencias (eje X).
    - X: Coeficientes complejos de la transformada (eje Y).
    """
    X = np.fft.rfft(x)                # Transformada rápida (solo parte
positiva del espectro)
    f = np.fft.rfftfreq(len(x), d=1/fs) # Genera el vector de frecuencias
correspondiente
    return f, X

# ========================================
# 2. EXPERIMENTO 1: SEÑAL SINTÉTICA (PRUEBA) aprender como funciona el
codigo
# ========================================
print("Iniciando prueba con señal sintética...")

# Parámetros de la señal
fs = 100                    # Frecuencia de muestreo (100 muestras por
segundo)
t = np.arange(0, 1, 1/fs)   # Vector de tiempo: 1 segundo de duración

# Generación de la señal: Suma de una onda de 2 Hz (deseada) y una de 5 Hz
(ruido)
x = np.sin(2*np.pi*2*t) + np.sin(2*np.pi*5*t)

# Transformación al dominio de la frecuencia
f, X = rfft_sig(x, fs)

# Diseño del Filtro Notch (Rechaza-banda)
H = np.ones_like(X)         # Crea un vector de '1's del mismo tamaño que el
espectro
# Aplica la máscara: Pone '0' donde la frecuencia (f) está cerca de 5 Hz
(+/- 0.5 Hz)
H[np.abs(f-5) <= 0.5] = 0

# Aplicación del filtro y reconstrucción
# Se multiplica el espectro X por la máscara H (filtrado en frecuencia)
# Luego se aplica la Transformada Inversa (irfft) para volver al tiempo
x_f = np.fft.irfft(X*H, n=len(x))

# --- Gráficas del Experimento 1 ---
plt.figure(figsize=(9,4))
plt.plot(t, x, label="Original", lw=2)      # Señal con ruido
plt.plot(t, x_f, label="Filtrada", lw=2)    # Señal limpia (solo 2 Hz)
```

```
plt.grid(True, alpha=0.3); plt.legend()
plt.xlabel("Tiempo (s)"); plt.ylabel("Amplitud")
plt.title("Prueba (2 Hz + 5 Hz) en el tiempo")
savefig("fig01_prueba_tiempo.png")

plt.figure(figsize=(9,4))
plt.stem(f, np.abs(X), basefmt=" ", label="Antes")      # Espectro original
plt.stem(f, np.abs(X*H), basefmt=" ", label="Después")  # Espectro con el 0
en 5Hz
plt.xlim(0, 10); plt.grid(True, alpha=0.3); plt.legend()
plt.xlabel("Frecuencia (Hz)"); plt.ylabel("Magnitud")
plt.title("FFT prueba (antes vs después)")
savefig("fig02_prueba_espectro.png")

# Validación de energía (Teorema de Parseval)
print("=== Parseval prueba ===")
print("E tiempo:", E_t(x), " | E freq:", E_f(x))
print("E tiempo (f):", E_t(x_f), " | E freq (f):", E_f(x_f))

# ========================================
# 3. EXPERIMENTO 2: AUDIO REAL (FILTRO PASA-BAJAS)
# ========================================
print("\nIniciando procesamiento de audio...")

# Carga del archivo de audio
try:
    fs_a, xa = wavfile.read("audio.wav")
except FileNotFoundError:
    print("ERROR: No se encontró 'audio.wav'. Asegúrate de tener el
archivo.")
    exit()

# Preprocesamiento del audio
# Si el audio es estéreo (tiene 2 canales), saca el promedio para hacerlo
mono
xa = xa.mean(axis=1) if xa.ndim > 1 else xa
xa = xa.astype(float) # Convierte a decimales
# Normalización: escala la amplitud entre -1 y 1 dividiendo por el valor
máximo
xa /= np.max(np.abs(xa)) + 1e-12

# Vectores de tiempo y frecuencia para el audio
ta = np.arange(len(xa))/fs_a
fa, Xa = rfft_sig(xa, fs_a)
```

```
# Diseño del Filtro Pasa-Bajas (Low-Pass Filter)
cutoff = 1000  # Frecuencia de corte en 1000 Hz
# Crea una máscara booleana: 1 si la frecuencia es menor al corte, 0 si es
mayor
Ha = (fa <= cutoff).astype(float)

# Filtrado y reconstrucción
xa_f = np.fft.irfft(Xa*Ha, n=len(xa))

# --- Gráficas del Experimento 2 ---
# 1. Señal original en el tiempo
plt.figure(figsize=(10,4))
plt.plot(ta, xa, lw=1)
plt.grid(True, alpha=0.3)
plt.xlabel("Tiempo (s)"); plt.ylabel("Amplitud")
plt.title("Audio original (tiempo)")
savefig("fig03_audio_tiempo_original.png")

# 2. Señal filtrada en el tiempo (se verá más suave)
plt.figure(figsize=(10,4))
plt.plot(ta, xa_f, lw=1)
plt.grid(True, alpha=0.3)
plt.xlabel("Tiempo (s)"); plt.ylabel("Amplitud")
plt.title("Audio filtrado (tiempo)")
savefig("fig04_audio_tiempo_filtrado.png")

# 3. Espectro del audio original en Decibeles
plt.figure(figsize=(10,4))
plt.plot(fa, db(Xa), lw=1)
plt.xlim(0, 5000); plt.grid(True, alpha=0.3) # Zoom a los primeros 5kHz
plt.xlabel("Frecuencia (Hz)"); plt.ylabel("Magnitud (dB)")
plt.title("Espectro audio (dB)")
savefig("fig05_audio_espectro_db.png")

# 4. Comparación de espectros (Antes vs Después)
plt.figure(figsize=(10,4))
plt.plot(fa, db(Xa), lw=1, alpha=0.7, label="Antes")
plt.plot(fa, db(Xa*Ha), lw=1.2, alpha=0.9, label="Después (Corte 1kHz)")
plt.xlim(0, 5000); plt.grid(True, alpha=0.3); plt.legend()
plt.xlabel("Frecuencia (Hz)"); plt.ylabel("Magnitud (dB)")
plt.title(f"Espectro antes vs después (LPF {cutoff} Hz)")
savefig("fig06_audio_antes_despues_db.png")

# Validación final y métricas de error
print("\n=== Parseval audio ===")
```

```
print("E tiempo:", E_t(xa), " | E freq:", E_f(xa))
print("E tiempo (f):", E_t(xa_f), " | E freq (f):", E_f(xa_f))

print("\n=== Métricas audio ===")
print("MSE (Error Cuadrático Medio):", mse(xa, xa_f))
print("SNR (Relación Señal-Ruido dB):", snr(xa, xa_f))

print("\n=== Guardando audio ===")
#convertimos los datos para que funcione el audio de salida
xa_f_norm = xa_f / np.max(np.abs(xa_f))
xa_int16 = (xa_f_norm * 32767).astype(np.int16)

# 3. Guardar el archivo convertido
wavfile.write("audio_filtrado_final.wav", fs_a, xa_int16)
print("Audio guardado exitosamente en formato PCM 16-bit.")
```