Matemáticas Avanzadas para la Ingeniería
Escuela Superior de Cómputo

DENOISING DE AUDIO MEDIANTE FFT Y VALIDACIÓN CON EL TEOREMA DE PARSEVAL

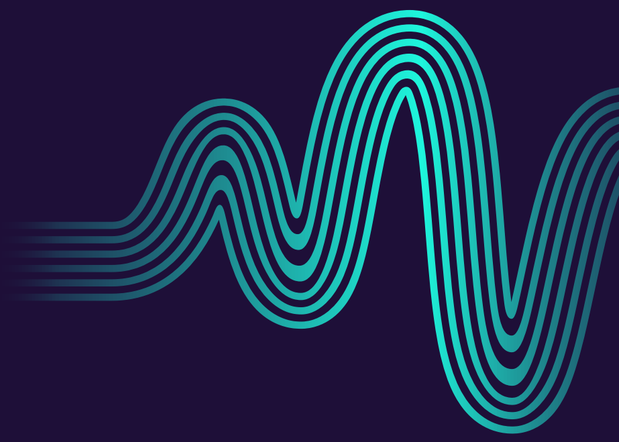
Sánchez González Axel Yael

asanchezg1901@gmail.com

Denoising de audio mediante FFT y validación con el Teorema de Parseval

Motivación del proyecto:

- El audio digital es altamente susceptible al ruido durante su adquisición y transmisión.
- El ruido degrada la calidad perceptual y la información contenida en la señal.
- El procesamiento en el dominio del tiempo resulta limitado para separar señal y ruido.
- El dominio de la frecuencia permite identificar y atenuar componentes no deseadas.



Justificación:

- Permite aplicar de forma directa la Transformada de Fourier discreta.
- Integra conceptos matemáticos con una implementación computacional real.
- Facilita la validación teórica mediante el Teorema de Parseval.

[1], [2]

Planteamiento del problema

El problema que se aborda en este proyecto consiste en eliminar ruido de una señal de audio digital sin destruir la información relevante de la señal original.

Para ello, se asume que la señal es discreta, de duración finita y que puede analizarse tanto en el dominio del tiempo como en el dominio de la frecuencia.

El objetivo no es únicamente obtener un audio perceptualmente mejor, sino demostrar que el proceso de filtrado es matemáticamente consistente, utilizando métricas cuantitativas y la verificación del Teorema de Parseval.

Señales discretas y dominio del tiempo

SEÑAL DE AUDIO DIGITAL:

Una señal de audio digital se representa matemáticamente como una secuencia discreta de valores, donde cada muestra corresponde a la amplitud de la señal en un instante de tiempo específico.

REPRESENTACIÓN MATEMÁTICA:

$$x[n], \quad n=0,1,2,\ldots,N-1$$

donde:

- x[n]: amplitud de la señal en la muestra n
- N: número total de muestras
- n: índice discreto de tiempo

Aunque el dominio del tiempo permite visualizar la forma de la señal, no siempre es suficiente para identificar la presencia de ruido, ya que múltiples componentes frecuenciales pueden coexistir simultáneamente.

Por esta razón, se hace necesario analizar la señal en otro dominio que permita separar sus componentes de manera más clara.

Transformada de Fourier Discreta (DFT)

La Transformada de Fourier permite analizar una señal en términos de sus componentes de frecuencia.

Convierte una señal del dominio del tiempo al dominio de la frecuencia.

Definición matemática (DFT):

$$X[k] = \sum_{n=0}^{N-1} x[n]\ e^{-j2\pi kn/N}$$

donde:

- x[n]: señal en el dominio del tiempo
- X[k]: espectro en frecuencia
- N: número total de muestras
- k: índice de frecuencia discreta
- j: unidad imaginaria

Su objetivo es expresar una señal como la suma de sinusoides complejas de diferentes frecuencias.

En esta ecuación, cada valor del espectro indica qué tanto contribuye una frecuencia determinada a la señal original, lo cual es fundamental para poder identificar y eliminar ruido de manera selectiva.

[3]

FFT: Implementación comútacional

Problema de la DFT directa:

- Complejidad computacional:
- O(N^2)
- Impráctica para señales largas de audio.

Solución: FFT (Fast Fourier Transform):

- Algoritmo optimizado para calcular la DFT.
- Reduce la complejidad a:
- O(NlogN)

Ventajas clave:

- Permite procesamiento de audio en tiempos razonables.
- Hace viable el filtrado en frecuencia.

Uso en el proyecto:

- Se utiliza la función fft() de scipy.fft

Aunque la DFT es fundamental desde el punto de vista matemático, su implementación directa es computacionalmente costosa.

Por esta razón, en aplicaciones reales se utiliza la FFT, que es un algoritmo que calcula exactamente el mismo resultado, pero de forma mucho más eficiente.

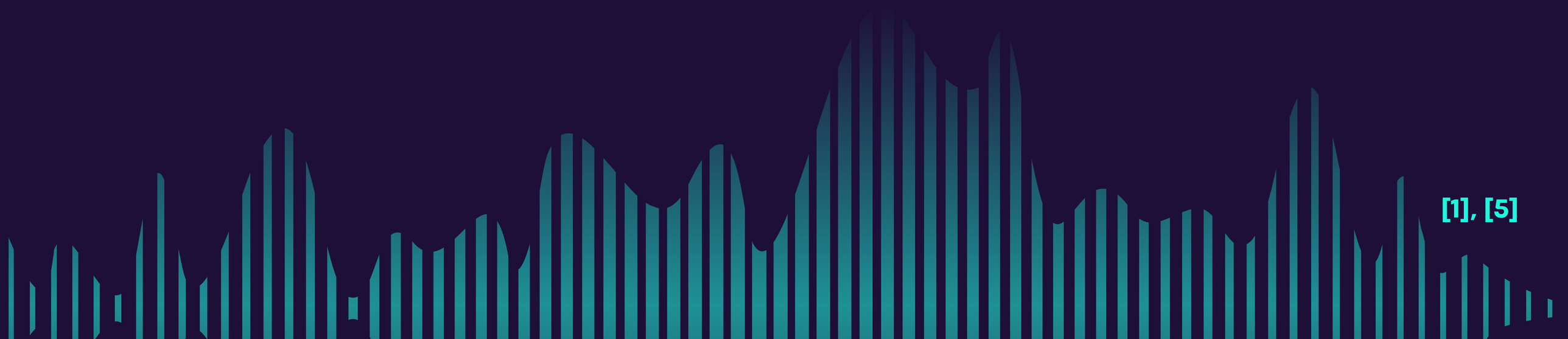
Gracias a la FFT, es posible procesar archivos de audio con miles de muestras sin afectar el desempeño del sistema.

Espectro de Frecuencia y Presencia de Ruido

Una vez que transformamos la señal del dominio del tiempo al dominio de la frecuencia mediante la FFT, obtenemos su espectro de magnitud.

Este espectro nos permite observar cómo se distribuye la energía de la señal en función de la frecuencia. En una señal ideal, la energía estaría concentrada únicamente en las frecuencias que componen la señal original.

Sin embargo, cuando existe ruido, aparecen componentes adicionales en el espectro que no pertenecen a la señal deseada.



[1], [5]

Tipos de ruido identificados en el espetro

En este proyecto se consideran principalmente dos tipos de ruido:

RUIDO BLANCO

Que se caracteriza por tener energía distribuida prácticamente en todo el rango de frecuencias

RUIDO TONAL

Que se manifiesta como picos bien definidos en frecuencias específicas, como el ruido eléctrico de 60 Hz.

Matemáticamente, el espectro obtenido mediante la FFT es un conjunto de coeficientes complejos que indican la contribución de cada frecuencia discreta a la señal original.

Analizando la magnitud de estos coeficientes, es posible decidir qué componentes son relevantes y cuáles corresponden a ruido.

Diseño del Filtro en Frecuencia: Máscaras Espectrales

Una vez identificado el ruido en el espectro de frecuencia, el siguiente paso es diseñar un filtro que permita eliminar dichas componentes no deseadas.

En este proyecto, el filtrado se implementa mediante el uso de máscaras espectrales.

¿Qué es una máscara en frecuencia?

Una máscara en el dominio de la frecuencia es un arreglo de valores reales que toma típicamente valores entre 0 y 1, y que se aplica directamente al espectro de la señal.

Matemáticamente, si X[k] representa el espectro de la señal original y H[k] representa la máscara del filtro, entonces el espectro filtrado se obtiene como:

$$Y[k] = X[k] \cdot H[k]$$

donde:

- X[k]: espectro original obtenido con FFT
- H[k]: función de transferencia del filtro (máscara)
- Y[k]: espectro filtrado

Interpretación física

Cuando la máscara toma el valor de 1, la frecuencia correspondiente se conserva.

Cuando la máscara toma el valor de 0, dicha componente espectral se elimina completamente.

De esta forma, el filtrado se convierte en una operación de selección de frecuencias.

Tipos de filtros implementados

Dependiendo del tipo de ruido que se desea eliminar, se diseñaron diferentes máscaras:

Filtro pasa-bajas:

- Conserva las frecuencias menores a una frecuencia de corte fc.

Filtro pasa-altas:

- Conserva las frecuencias mayores a fc.

Filtro pasa-banda:

- Conserva únicamente un rango de frecuencias [fmin,fmax].

Filtro notch (rechaza-banda):

- Elimina un rango específico de frecuencias no deseadas, como el ruido de 60 Hz.

Reconstrucción de la Señal mediante IFFT

Hasta este punto, el filtrado ha sido realizado completamente en el dominio de la frecuencia.

Sin embargo, para poder escuchar la señal procesada y analizar su comportamiento temporal, es necesario regresar al dominio del tiempo.

Transformada Inversa de Fourier (IFFT)

La Transformada Inversa de Fourier permite reconstruir una señal temporal a partir de su espectro discreto.

Matemáticamente, la IFFT se define como:

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] e^{j2\pi kn/N}$$

donde:

- X[k]: espectro de la señal (ya filtrado)
- x[n]: señal reconstruida en el dominio del tiempo
- N: número total de muestras
- j: unidad imaginaria

[3], [6]

En este proyecto, una vez aplicado el filtro en frecuencia, se obtiene el espectro filtrado Y[k].
A este espectro se le aplica la IFFT para obtener la señal temporal filtrada:

$$y[n]=IFFT\{Y[k]\}$$

Detalle importante: parte real de la señal

Debido a errores numéricos y redondeo en el cálculo computacional, el resultado de la IFFT puede contener una pequeña parte imaginaria, la cual no tiene significado físico en señales de audio reales.
Por esta razón, se toma únicamente la parte real:

$$y[n]=\Re\{IFFT(Y[k])\}$$

Este proceso se implementa en el archivo procesar.py mediante las siguientes líneas:

```
datos_filtrados = np.real(ifft(espectro_filtrado))
```

Aquí:

- espectro_filtrado contiene el resultado de aplicar la máscara
- ifft() reconstruye la señal
- np.real() elimina componentes imaginarias residuales

Métricas de Calidad: Evaluación Cuantitativa del Filtrado

Evaluar numéricamente la calidad del proceso de denoising y comparar la señal original y la señal filtrada.

Métricas utilizadas: Error Cuadrático Medio (MSE) y Relación Señal-Ruido (SNR)

Error Cuadrático Medio (MSE)

REPRESENTACIÓN MATEMÁTICA:

$$\mathrm{MSE} = \frac{1}{N}\sum_{n=0}^{N-1}\left(x[n] - y[n]\right)^2$$

donde:

- x[n]: señal original
- y[n]: señal filtrada
- N: número total de muestras

Interpretación:

- MSE pequeño → señal filtrada cercana a la original
- MSE grande → alta distorsión introducida por el filtrado

Relación Señal-Ruido (SNR)

REPRESENTACIÓN MATEMÁTICA:

$$\mathrm{SNR} = 10 \log_{10} \left(\frac{P_{\text{señal}}}{P_{\text{ruido}}} \right)$$

donde:

- x[n]: señal original
- y[n]: señal filtrada
- N: número total de muestras

$$P_{\text{señal}} = \frac{1}{N} \sum y[n]^2$$
$$P_{\text{ruido}} = \frac{1}{N} \sum (x[n] - y[n])^2$$

Interpretación:

- SNR alto → mejor calidad de la señal
- SNR bajo → ruido dominante

Estas métricas se calculan en el archivo procesar.py dentro de la función:

def calcular_metricas(original, procesada):

```
mse = np.mean((original - procesada) ** 2)

ruido = original - procesada
potencia_senal = np.mean(procesada ** 2)
potencia_ruido = np.mean(ruido ** 2)

snr = 10 * np.log10(potencia_senal / potencia_ruido)
```

Validación del Teorema de Parseval

El Teorema de Parseval establece que la energía total de una señal es la misma, independientemente de si la analizamos en el dominio del tiempo o en el dominio de la frecuencia.

En su forma discreta, esto significa que la suma de los cuadrados de las muestras de la señal en el tiempo debe ser igual (salvo errores numéricos) a la suma de los cuadrados del espectro en frecuencia, escalada por el número de muestras.

En el proyecto calculamos ambas energías: primero directamente desde la señal temporal y después desde el espectro obtenido con la FFT. Posteriormente comparamos estos valores y calculamos un error porcentual.

El hecho de que este error sea muy pequeño confirma que la FFT y la IFFT están correctamente implementadas y que el filtrado no introduce ni elimina energía artificialmente, lo cual valida tanto la implementación computacional como la coherencia matemática del proceso.

Teorema de Parseval (forma discreta):

REPRESENTACIÓN MATEMÁTICA:

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

Interpretación física

- El lado izquierdo representa la energía de la señal en el tiempo.
- El lado derecho representa la energía de la señal en frecuencia.
- Ambos valores deben ser aproximadamente iguales.

Cualquier diferencia se debe a errores numéricos de punto flotante.

Importancia en el proyecto

- Garantiza que la FFT e IFFT están correctamente implementadas.
- Confirma que el filtrado no introduce ni elimina energía artificialmente.
- Refuerza la correcta aplicación del marco matemático del curso.

Implementación en el código

Esta validación se implementa en procesar.py mediante la función:

def verificar_parseval(señal_tiempo, espectro_frecuencia):
Fragmento clave:

```
energia_tiempo = np.sum(señal_tiempo ** 2)
energia_frecuencia = np.sum(np.abs(espectro_frecuencia) ** 2) / N
error_porcentual = 100 * abs(energia_tiempo - energia_frecuencia) / energia_tiempo
```

[8], [9]

Conclusión

Este proyecto logró desarrollar una herramienta computacional funcional que aplica directamente los conceptos de Matemáticas Avanzadas para la Ingeniería al problema real del denoising de audio. Mediante la implementación práctica de la FFT, pudimos analizar señales en el dominio de la frecuencia, diseñar filtros espectrales específicos y reconstruir la señal limpia mediante IFFT.

La validación mediante el Teorema de Parseval confirmó la consistencia matemática del proceso, demostrando que la energía se conserva entre los dominios del tiempo y la frecuencia. Las métricas cuantitativas (MSE y SNR) permitieron evaluar objetivamente la efectividad de cada filtro, completando así un ciclo completo de análisis, procesamiento y validación.

Más allá del resultado técnico, este trabajo consolida la conexión entre teoría matemática y aplicación computacional, mostrando cómo conceptos abstractos encuentran utilidad práctica en la solución de problemas de ingeniería contemporáneos.

Referencias

[1] A. V. Oppenheim and R. W. Schafer, Discrete-Time Signal Processing, 3rd ed. Pearson, 2010. [En línea]. Obtenido de: https://studylib.net/doc/26288079/-prentice-hall-signal-processing-series--alan-v.-oppenhei. [Accedido: 4 de enero de 2026].

[2] S. W. Smith, The Scientist and Engineer's Guide to Digital Signal Processing. California Technical Publishing, 1997. [En línea]. Obtenido de: https://img.anfulai.cn/bbs/97312/The%20Scientist%20and%20Engineer%20Guide%20to%20DSP.pdf. [Accedido: 4 de enero de 2026].

[3] A. V. Oppenheim, A. S. Willsky, and S. H. Nawab, Signals and Systems, 2nd ed. Prentice Hall, 1996. [En línea]. Obtenido de: https://www.cedric-richard.fr/assets/files/Signals_and_Systems_2nd_Edition_by_Oppen.pdf. [Accedido: 4 de enero de 2026].

[4] J. W. Cooley and J. W. Tukey, “An algorithm for the machine calculation of complex Fourier series,” Math. Comput., vol. 19, no. 90, pp. 297–301, 1965. [En línea]. Obtenido de: https://www.ams.org/journals/mcom/1965-19-090/S0025-5718-1965-0178586-1/S0025-5718-1965-0178586-1.pdf. [Accedido: 4 de enero de 2026].

[5] J. G. Proakis and D. G. Manolakis, Digital Signal Processing: Principles, Algorithms, and Applications, 4th ed. Pearson, 2006. [En línea]. Obtenido de: https://www.srecwarangal.ac.in/ece-downloads/Digital_Signal_Processing___Proakis_and_Manolakis.pdf. [Accedido: 4 de enero de 2026].

[6] B. P. Lathi, Linear Systems and Signals, 2nd ed. Oxford University Press, 2005. [En línea]. Obtenido de: https://umairbfrend.wordpress.com/wp-content/uploads/2015/01/text-book.pdf. [Accedido: 4 de enero de 2026].
[7] P. C. Loizou, Speech Enhancement: Theory and Practice, 2nd ed. CRC Press, 2013. [En línea]. Obtenido de: https://api.pageplace.de/preview/DT0400.9781466504226_A37880723/preview-9781466504226_A37880723.pdf. [Accedido: 4 de enero de 2026].
[8] C. R. Harris et al., "Array programming with NumPy," Nature, vol. 585, pp. 357–362, 2020. [En línea]. Obtenido de: https://doi.org/10.1038/s41586-020-2649-2. [Accedido: 4 de enero de 2026].
[9] P. Virtanen et al., "SciPy 1.0: Fundamental algorithms for scientific computing in Python," Nat. Methods, vol. 17, pp. 261–272, 2020. [En línea]. Obtenido de: https://doi.org/10.1038/s41592-019-0686-2. [Accedido: 4 de enero de 2026].
[10] S. Haykin, Communication Systems, 4th ed. Wiley, 2001. [En línea]. Obtenido de: https://ggnindia.dronacharya.info/Downloads/Sub-info/RelatedBook/4thSem/Communication-System-text-book-6.pdf. [Accedido: 4 de enero de 2026].
[11] R. N. Bracewell, The Fourier Transform and Its Applications, 3rd ed. McGraw-Hill, 2000. [En línea]. Obtenido de: https://see.stanford.edu/materials/lsoftaee261/book-fall-07.pdf. [Accedido: 4 de enero de 2026].