Universidad de Costa Rica
Facultad de Ingeniería
Escuela de Ingeniería Eléctrica

TRABAJO FINAL DE GRADUACIÓN
LICENCIATURA

ANÁLISIS COMPARATIVO DE FUNCIONES *WAVELET* PARA EL MÉTODO DE ELIMINACIÓN DE RUIDO EN SEÑALES DE VOZ

Por:

Josué David Elizondo Murillo

Ciudad Universitaria Rodrigo Facio, Costa Rica

Junio de 2021

FICHA RESUMEN DEL TEMA PROPUESTO

- **TÍTULO:** ANÁLISIS COMPARATIVO DE FUNCIONES *WAVELET* PARA EL MÉTODO DE ELIMINACIÓN DE RUIDO EN SEÑALES DE VOZ
- **ESTUDIANTE:** Josué David Elizondo Murillo
  - **Carné:** B42354
  - **Teléfono:** +506 8497 8688
  - **Correo Electrónico:** `josue.elizondo@ucr.ac.cr`
- **COMITÉ ASESOR PROPUESTO**
  - **Director:** Ing. Jorge Arturo Romero Chacón, PhD.
  - **Asesor:** Ing. Marvin Coto Jiménez, PhD
  - **Asesor:** Ing. Diego Dumani Jarquín, PhD
- **MODALIDAD:** Tesis
- **CONTRIBUCIÓN PERSONAL:** Se aplican técnicas de procesamiento digital de señales para eliminación de ruido en señales de voz mediante el método de recorte de coeficientes en el dominio *wavelet*. El método depende de la función *wavelet* seleccionada, de acuerdo con los criterios establecidos para tal propósito, como por ejemplo el tipo de señal, el ruido a suprimir y el umbral de recorte.

  Se pretende aplicar el método considerando cinco funciones *wavelet* distintas. De los resultados obtenidos se busca determinar cuál función *wavelet* y su respectiva transformada ofrece mejor rendimiento para eliminación de ruido en señales de voz de acuerdo a los ruidos seleccionados para la investigación.

La presente investigación se ha elaborado de conformidad con lo establecido en el Reglamento de Trabajos Finales de Graduación de la Universidad de Costa Rica.

Josué David Elizondo Murillo
Estudiante proponente

Fecha: _____

Avalo el contenido de la presente investigación de tema para trabajo final de graduación.

VB°_____
Director del Comité Asesor

Índice general

Índice de figuras

Índice de cuadros

Acrónimos

CWT	Transformada *Wavelet* Continua (Continuous *Wavelet* Transform)
DSP	Procesamiento Digital de Señales (Digital Signal Processing)
DWT	Transformada *Wavelet* Discreta (Discrete *Wavelet* Transform)
FIR	Respuesta Finita al Impulso (Finite Impulse Response)
MSE	Error Cuadrático Medio (Mean Square Error)
$PESQ$	Evaluación Porcentual de la Calidad de Habla (Percentual Evaluation of Speech Quality)
QMF	Filtros de Espejo en Cuadratura (Quadrature Mirror Filter)
SNR	Relación Señal a Ruido (Signal to Noise Ratio)

1 Introducción

1.1. Antecedentes

En la actualidad el estudio de las funciones *wavelet*, desde la perspectiva del procesamiento digital de señales, ha disminuido considerablemente con respecto a las investigaciones desarrolladas a finales de los años noventa. Tal y como se describe en Meyer (1993), el desarrollo de la teoría matemática que en la actualidad se conoce sobre las funciones *wavelet* fue un estudio progresivo y que contó con una gran serie de colaboradores exclusivos del área de las matemáticas, tal es el caso de los descubrimientos llevados a cabo por Alfred Haar, J.E. Littlewood, Raymond Paley e Ingrid Daubechies. Fue cerca de 1985 cuando los estudios de Stephane Mallat (Meyer, 1993) sintetizaron la mayor parte de las investigaciones previas y se forjó la base de la aplicación de las funciones *wavelet* en el procesamiento digital de señales.

En el campo de la ingeniería, la teoría matemática desarrollada funcionó como base para llevar a cabo investigaciones principalmente en procesamiento de señales e imágenes, codificación de señales, mecánica cuántica, análisis numérico y otros campos de estudio (Daubechies, 1992). En procesamiento digital de señales, el principal motivo de investigación consistía en encontrar una herramienta superior a la transformada de Fourier para el tratamiento de singularidades como el caso del ruido (Meyer, 1993). A partir de este punto, diversos métodos y algoritmos fueron desarrollados para estandarizar los procedimientos de supresión de ruido mediante transformadas con funciones *wavelet*.

Uno de los primeros aportes en este campo fue el desarrollado por David L. Donoho, donde se sintetizan los detalles más importantes del método de eliminación de ruido mediante recorte de coeficientes en el dominio *wavelet* (Donoho, 1995). En este estudio, Donoho hace uso del término *de-noising* que en adelante será clave para definir los algoritmos para eliminación de ruido mediante la transformada *wavelet*. Donoho (1995) presenta el método de recorte suave de coeficientes, que actualmente es uno de los procedimientos más reconocidos para eliminación de ruido mediante esta transformada.

En adelante, se han desarrollado diversos estudios relacionados con el método de eliminación de ruido donde, para el caso de señales de voz, se puede

citar el llevado a cabo por Nongpiur (2008), en el cual se presentan las ventajas de la transformada *wavelet* para eliminación de ruido gracias a su versatilidad para representar matemáticamente los puntos transitorios de las señales, puntos donde generalmente se encuentra el ruido y la transformada de Fourier se queda corta para su representación. Otro ejemplo similar es el desarrollado por Xiaoli y Mao (2004), donde se presenta el típico caso de supresión de ruido mediante transformada *wavelet* y recorte suave de coeficientes. Por su parte, Yong y Qiang (2010) explican claramente el algoritmo de la eliminación del ruido mediante la transformada *wavelet*. En su estudio, presentan diagramas de flujo con el algoritmo y un pseudo-código para implementación en algún lenguaje de programación.

En otros aportes, se han desarrollado investigaciones relacionadas con la selección del umbral para el recorte de coeficientes. Por ejemplo, Zhigang et al. (2013) ofrecen un nuevo umbral para el recorte de los coeficientes de la señal procesada con la transformada *wavelet*. Los métodos de recorte suave, medio y fuerte tienen definido su umbral para el recorte de coeficientes. Zhigang et al. (2013) presentan un nuevo valor de umbral que procura mejores resultados experimentales. De igual forma, Lin y Cai (2010) proponen otro umbral para el recorte de los coeficientes en el proceso de eliminación de ruido mediante la transformada *wavelet*. Estos estudios analizan el umbral propuesto y verifican su desempeño mediante simulaciones y resultados experimentales. Cabe destacar que Lin y Cai (2010) comparan el rendimiento de los umbrales propuestos mediante la medición de la relación señal a ruido (SNR).

Otras investigaciones se han delimitado con respecto a la selección de la función *wavelet* adecuada. Como referencia, Long et al. (2004) hacen un estudio sobre las siguientes *wavelets*: Haar, Daubechies, Meyer, Biortogonal, Coiflets y Symlets. En su estudio, la selección de la función *wavelet* se hace a partir del error obtenido al reconstruir la señal después de haber aplicado la transformada *wavelet* y la transformada *wavelet* inversa, considerando cinco niveles de descomposición. Similarmente, Deng y Jiang (2012) presentan algunas bases para determinar el tipo de *wavelet* más indicado para el proceso de eliminación de ruido considerando el tipo de señal que se desea procesar.

Como bien se indicaba anteriormente, los desarrollos con funciones *wavelet* en los años recientes han sido menos comunes. Razones por las que se podría deducir su motivo consisten en la dificultad de implementar la matemática de la transformada en plataformas actuales, comparado con otros métodos más flexibles como el filtro Wiener. A su vez influye también el descubrimiento de nuevas metodologías como los algoritmos *deep learning*, los cuáles son más afines con la tecnología computacional actual y gozan de mayor popularidad.

Otra razón podría deberse a que el apogeo de la investigación *wavelet* sucedió a finales de los años noventa y principios de los dos mil. Esto no quiere decir que no exista valor en su investigación, sino que las investigaciones pueden ser más específicas con el propósito de afinar detalles. En el ámbito nacional, con respecto a estudios de esta índole, se cuenta con la tesis de maestría de Delgado (2010) que utiliza el método de eliminación de ruido mediante transformada *wavelet* para restaurar obras musicales antiguas. Este estudio presenta un ejemplo claro de una aplicación del método de eliminación de ruido mediante funciones *wavelet* en la vida real.

1.2. Justificación

Actualmente, la mayor parte de las señales en los sistemas de información son digitalizadas mediante algún método de conversión analógico a digital. Es inevitable que durante el proceso de conversión la señal deseada se mezcle con diversos tipos de ruido. Por esta razón, el profesional en ingeniería eléctrica debe ser capaz de llevar a cabo métodos por los cuales se pueda reducir el nivel de ruido, de forma tal, que la señal procesada sea lo más similar a la señal deseada, es decir, con el nivel de ruido suficientemente despreciable para su respectivo funcionamiento.

El ruido está presente de una u otra forma en las señales. En el procesamiento digital de señales influye desde el ruido blanco, y de otros colores según su espectro en frecuencia como el ruido rosa, hasta componentes indeseables que son recopiladas en la etapa de muestreo, como el ruido *babble*. El nivel de ruido de una señal puede ser suficiente para degradar la misma y que su posterior procesamiento o transmisión se vea considerablemente afectado. Este escenario puede presentarse en cualquier sistema que tome señales como entradas, pero cuando se consideran señales de voz, se puede delimitar a telecomunicaciones, grabaciones musicales y sistemas de procesamiento de voz.

Esta investigación, en particular, permite determinar la función *wavelet* que produce mejores resultados para el método de eliminación de ruido en señales de voz. Delimita el tipo de señal a señales de voz y estudia las bases teóricas y prácticas para la selección de las funciones *wavelets* que se analizan. Así mismo, considera diversos métodos de recorte de coeficientes, se aplican para cada función *wavelet* y se analiza su rendimiento. Los resultados de esta investigación se pueden llevar a la práctica en aplicaciones como sistemas de reconocimiento de voz, donde se desea que la señal de entrada esté idealmente libre de ruido. De la misma forma en sistemas de comunicaciones donde se emplean señales de voz.

Las investigaciones que han sido desarrolladas en esta área comúnmente consideran una única función *wavelet*, estudian como optimizar el umbral de recorte de coeficientes y además se enfocan en un solo tipo de ruido a suprimir. Por su parte, esta investigación considera cinco funciones *wavelets* distintas, dado que la eficacia del método depende de la transformada *wavelet* aplicada, para obtener la función adecuada según el tipo de ruido a suprimir. Así mismo, se desea considerar los tres umbrales de recorte de coeficientes tradicionales conocidos como suave, medio y fuerte. Se consideran los ruidos blanco, rosa y *babble*. Los resultados obtenidos permiten determinar la combinación de función *wavelet* y el umbral de recorte que brinda mejores resultados al aplicar el método según el ruido a suprimir seleccionado. Dado que se trabaja con señales de voz, se pretende considerar la calidad de la señal procesada en términos cualitativos y cuantitativos. Finalmente, se desea comparar los resultados del método mediante transformada *wavelet* contra los filtros Wiener, siendo el segundo mejor conocido en el estado del arte con señales de voz.

1.3. Planteamiento del problema

Desde la perspectiva de la ingeniería, el ruido es una cualidad inherente de las señales. En el campo del procesamiento de señales, el ruido puede aparecer tanto en el proceso de captación, como en el de muestreo o de procesamiento. Es imposible diseñar algoritmos que supriman por completo el ruido de las señales, sin embargo, existen métodos que permiten eliminar la mayor parte de sus componentes adheridas a tales señales.

En esta investigación se estudia el método de procesamiento digital de señales conocido como eliminación de ruido mediante recorte de coeficientes en el dominio *wavelet*. Principalmente, se hace un análisis de las funciones *wavelet* que presentan mejor desempeño para la aplicación del método considerando señales de voz, diferentes umbrales de recorte y ruidos a suprimir. Se selecciona la función *wavelet* que logra la mejor supresión de componentes de ruido según cada caso de estudio.

1.4. Objetivos

1.4.1 Objetivo General

Analizar el desempeño proporcionado por cinco funciones *wavelets* para eliminación de ruido en señales de voz mediante el método de recorte de coeficientes

en el dominio *wavelet*, con el fin de seleccionar la función *wavelet* que ofrece mejor desempeño para el algoritmo, las señales y los tipos de ruido delimitados.

1.4.2 Objetivos específicos

- Determinar cinco funciones *wavelet* adecuadas para procesamiento digital de señales de voz aplicando criterios de caracterización de señales.
- Generar las señales de voz de prueba con los ruidos seleccionados para suprimir mediante el algoritmo de estudio.
- Aplicar el método de eliminación de ruido mediante recorte de coeficientes considerando las funciones *wavelets* determinadas, los tipos de ruido de estudio y los criterios de recorte de coeficientes, realizando a su vez, mediciones de la relación señal a ruido y del error cuadrático medio a las señales procesadas.
- Seleccionar la función *wavelet* que ofrece mejor desempeño para eliminación de ruido con el algoritmo empleado y según el tipo de ruido suprimido, considerando la calidad de la señal de voz y comparando los resultados obtenidos contra los proporcionados por los filtros Wiener.

1.5. Alcances

El enfoque de este trabajo radica en eliminar ruido de señales mediante métodos de procesamiento digital de señales. Bajo esta línea de investigación, otros autores han demostrado que la transformada *wavelet* presenta mejores resultados para la supresión de ruido a diferencia de la transformada de Fourier (Meyer, 1993). Por esta razón, desde el punto de vista del procesamiento digital de señales es importante conocer acerca de los métodos que se han desarrollado para tal propósito.

Dentro de los métodos desarrollados para eliminación de ruido en señales digitalizadas se dispone del que emplea la transformada *wavelet* para recorte de coeficientes. En este estudio se desea aplicar este método considerando cinco funciones *wavelet* distintas pero adecuadas al tratamiento de señales de voz. Se sabe que el método se puede aplicar a señales de otra naturaleza, como imágenes o de descarga en líneas de transmisión, sin embargo se busca aplicar el método considerando específicamente señales de voz.

El método de recorte de coeficientes usualmente considera con tres tipos de recorte: suave, medio y fuerte (Delgado, 2010). En este estudio se pretende

tomar en cuenta los tres métodos de recorte descritos en la literatura tradicional. No se pretende considerar los umbrales de recorte definidos por otros autores. Mientras que con respecto a los ruidos a suprimir, se desea considerar naturales como por ejemplo el blanco y rosa. De la misma forma se desea estudiar ruidos artificiales como el *babble*.

Se pretende realizar mediciones del error cuadrático medio y de la relación señal a ruido para verificar la eficacia del método, así mismo se desea medir la evaluación porcentual de la calidad de habla (PESQ) para estudiar la calidad de la señal de voz procesada. A modo de comparación se desea aplicar el método de eliminación de ruido mediante filtros Wiener, algoritmo comúnmente utilizado para suprimir ruido en señales de voz, y a la vez comparar su desempeño con el método de estudio.

1.6. Hipótesis

En este estudio se pretende determinar la función *wavelet* que proporciona los mejores resultados con el método de eliminación de ruido mediante recorte de coeficientes. Para el procedimiento influyen diversas variables a considerar, como el ruido a suprimir, la función *wavelet* y el umbral de recorte. A partir del desarrollo de esta investigación se pretende demostrar que la eficacia del método depende principalmente de la función *wavelet* fundamental seleccionada para aplicar el algoritmo de supresión de ruido. De esta forma se comprueba que la selección de la función *wavelet* es un paso crítico y que no todas las funciones *wavelet* son aceptables para el algoritmo.

1.7. Metodología

- Dentro de los primeros pasos a considerar se cuenta con el estudio de las generalidades teóricas sobre las funciones *wavelet*. Estas funciones, desde la perspectiva del DSP son una herramienta matemática para aplicar el método de supresión de ruido. No obstante, se debe tener claro las principales características matemáticas que influyen en el tratamiento de señales mediante la transformada *wavelet*.

- Para la selección de las funciones *wavelet* que se aplicarán con el método de eliminación de ruido se debe conocer las características deseables en las mismas. Para esta selección se pueden considerar criterios como el de la distorsión de la señal al ser procesada por la transformada de cada función *wavelet*. Se debe considerar que la función *wavelet* a seleccionar se ve influenciada por el tipo de señal que se desea procesar.

- Posteriormente, se debe estudiar las formas en que el ruido se puede presentar en señales de voz particularmente. Este estudio pretende considerar el ruido blanco, el ruido rosa como un caso más práctico del ruido blanco y el ruido *babble* o de múltiples hablantes.

- En este punto se deben preparar las señales de prueba para la aplicación del método de eliminación de ruido. Para tal propósito, se toma un conjunto de señales de voz y una señal para cada tipo de ruido de estudio. Para este propósito se puede hacer uso del software Audacity.

- Las siguientes etapas se pueden considerar más prácticas, puesto que una vez seleccionadas las funciones *wavelet* del estudio y los ruidos que se desean eliminar, se procede a ejecutar el método tomando como entrada las señales de prueba generadas. Se aplica el método considerando las funciones *wavelet* seleccionadas y los criterios de recorte de coeficientes suave, medio y fuerte para el análisis. Para el procesamiento se pretende utilizar como herramienta el software libre Scilab y su librería sobre *wavelets*.

- Se pretende realizar mediciones de la relación señal a ruido y del error cuadrático medio para cada procesamiento. Así mismo, se desea mostrar curvas de las señales antes y después de ser procesadas. Además, se pretende brindar las muestras de audio procesadas.

- Dado que se trata con señales de voz, se desea verificar la calidad de las señales de procesadas, donde se puede hacer uso de la medición del algoritmo de la evaluación porcentual de la calidad de habla (PESQ).

- Se desea comparar el rendimiento del método que emplea *wavelets* con los filtros Wiener, donde el segundo es uno de los algoritmos más usados para procesamiento de señales de voz.

- Finalmente se puede pasar al análisis de resultados, considerando las mediciones realizadas hasta este punto.

1.8. Procedimiento de evaluación

Para el procedimiento de evaluación se decide estudiar el desempeño de la investigación de acuerdo a los archivos entregados al finalizar el estudio.

- Se deben generar archivos de audio .mp3 correspondientes a las señales de voz de prueba, es decir, contando con la señal de voz más los ruidos de estudio: blanco, rosa y *babble*.

- Se deben entregar las funciones desarrolladas en el entorno Scilab que realizan el procesamiento de las señales de audio. Estas funciones deben considerar las funciones *wavelet* delimitadas para el estudio y los umbrales de recorte de coeficientes seleccionados.
- Las funciones de Scilab deben procesar las señales de audio ruidosas y exportar las correspondientes señales de audio con los componentes de ruido atenuados.
- Se deben generar las funciones en Scilab para el procesamiento mediante filtros Wiener así como las señales de audio procesados con este método.

2 Marco Teórico

2.1. Generalidades teóricas sobre ruido

En el procesamiento digital de señales, como indica van Exter (2003), el ruido se modela como un evento aleatorio que se caracteriza en términos de sus propiedades probabilísticas. En términos de notación, se define una señal con ruido $S(t)$ como

$$S(t) = V(t) + N(t), \tag{2.1}$$

donde la señal de interés se representa con $V(t)$ y el ruido con $N(t)$.

Generalmente, para la mayoría de estudios se dice que el ruido cuenta con dos características especiales, presenta una distribución probabilística Gaussiana y sus propiedades estadísticas son estacionarias, es decir, que no dependen del tiempo (van Exter, 2003).

2.1.1 Ruido blanco

El ruido blanco, como indica Vaseghi (2008), se define como un proceso aleatorio sin correlación y que cuenta con magnitud de potencia constante en todo su espectro de frecuencia. Si bien es cierto, la definición del ruido blanco es principalmente un concepto teórico, en la práctica se presenta cuando aparece ruido con magnitud de potencia aproximadamente constante en una banda limitada de frecuencias. En la figura 2.1 se muestra una señal de ruido blanco, su autocorrelación y su densidad de potencia espectral.

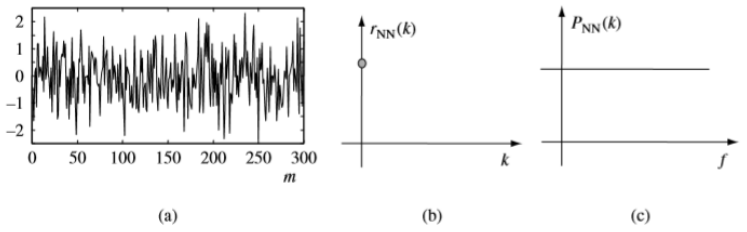

Figura 2.1: (a) Señal de ruido blanco. (b) Función de autocorrelación del ruido blanco. (c) Densidad de potencia espectral del ruido blanco. Disponible en (Vaseghi, 2008)

La función de autocorrelación del ruido blanco se define como

$$r_{NN} = E[N(t)N(t+\tau)] = \sigma^2\delta(\tau), \tag{2.2}$$

donde r_{NN} representa la autocorrelación, $N(t)$ la señal de ruido blanco y σ la varianza. Mientras que la densidad espectral de potencia se define como

$$P_{NN} = \int_{-\infty}^{\infty} r_{NN} e^{-2\pi ftj} dt = \sigma^2. \tag{2.3}$$

donde P_{NN} representa la densidad de potencia espectral y se obtuvo al calcular la transformada de Fourier de la función de autocorrelación.

2.1.2 Ruido rosa

Como se explicó en la sección anterior, el ruido blanco representa una idea sobre todo teórica, en la práctica se asocian ciertos tipos de ruido con colores de acuerdo con su espectro de frecuencia, en una idea similar a los colores presentes en el espectro visible (Vaseghi, 2008). El ruido rosa presenta la forma espectral presentada en la figura 2.2(b), donde se aprecia como la magnitud decrece conforma aumenta la frecuencia, a diferencia del ruido blanco que teóricamente presenta magnitud constante. Es apreciable de la figura 2.2(b) como la mayor parte de la potencia del ruido se encuentra en las frecuencias bajas, por lo que se deduce que este ruido afecta usualmente a señales de audio y voz (Vaseghi, 2008).

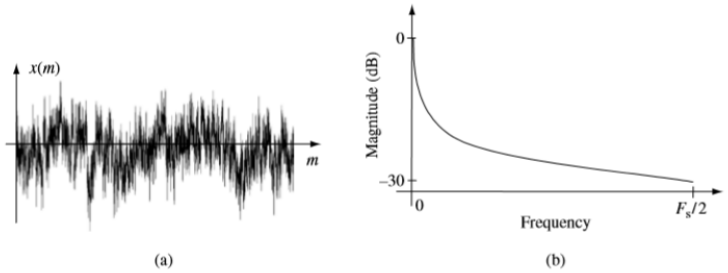

Figura 2.2: (a) Señal de ruido rosa. (b) Densidad espectral del ruido rosa. Disponible en (Vaseghi, 2008)

2.1.3 Ruido *babble*

El ruido *babble* como describen Krishnamurthy y Hansen (2008), consiste en un tipo de ruido que por sus características presenta gran dificultad para supresión en las señales de voz. Este tipo de ruido generalmente es de interés cuando se trabaja con sistemas de procesamiento de voz, donde se modelan las

señales en condiciones adversas para validar la efectividad del sistema. El ruido *babble* consiste en múltiples hablantes en un tiempo y lugar específico. Como explican Krishnamurthy y Hansen (2008), dentro de sus dificultades se cuenta con su estructura no estacionaria y su similitud con la señal deseada, dado que son de la misma naturaleza. Finalmente, el ruido *babble* está directamente relacionado con la cantidad de hablantes presentes en el ambiente, es decir, entre más hablantes se cuenta con mayor variabilidad acústica.

2.2. Perspectiva histórica sobre *wavelets*

El estudio de las funciones *wavelet* data desde principios del siglo XX, cuando el término *wavelet* o su propio concepto ni siquiera existía como se conoce actualmente (Meyer, 1993). Dentro de los primeros descubrimientos se resalta el desarrollado por Alfred Haar alrededor de 1910. Haar, matemático alemán, motivado por los descubrimientos previos de Fourier, en el que se podía representar cualquier función como una suma de funciones ortogonales trigonométicas, basó su investigación en la búsqueda de otros sistemas ortogonales (Meyer, 1993). De esta forma se definió la actual función de Haar y su matriz asociada, la cual considera la traslación y el desplazamiento sobre un vector base. Debido a este vector base, los vectores de la matriz de Haar son perpendiculares entre sí y de esta forma fue posible el desarrollo de matrices ortogonales tomando como referencia el vector base (Bömers, 2000).

Los estudios posteriores trataron de solventar las carencias que presentaba la transformada de Fourier para el análisis de señales. Por ejemplo, como se indica en Meyer (1993), el sistema trigonométrico presenta limitaciones para representar las singularidades de las señales y la distribución de potencia de la señal, ya sea si se encuentra concentrada en puntos especiales o a lo largo de toda su distribución. En este punto es donde se desarrolla el concepto de separación del fenómeno en partes más simples o niveles, al cuál se le denomina comúnmente *separación atómica* gracias al concepto de una *wavelet madre* (Bömers, 2000). En adelante el concepto de una *wavelet madre* (o *wavelet* base) será utilizado por las investigaciones posteriores. En la literatura usualmente se define la *wavelet* base como Ψ. Estos aportes fueron liderados principalmente por J.E. Littlewood y Raymond Paley alrededor de 1930 y posteriormente por Alberto Calderón y Antoni Zygmund alrededor de 1960 (Meyer, 1993).

Posteriormente, cerca de 1980 los estudios de A. Grossmann and J. Morlet definieron el concepto de *wavelets* en el campo de la mecánica cuántica basados en los resultados de Calderón–Zygmund (Meyer, 1993). Por su parte

Jan-Olov Strömberg, como se indica en Bömers (2000), descubre la primer función *wavelet* ortogonal, que a diferencia de la de Haar se caracteriza por presentar suavidad en su curva. Cabe destacar que Grossmann y Morlet, físico e ingeniero respectivamente, con interés en procesamiento de señales e imágenes, realizan la primer definición de la palabra *wavelet*, una onda pequeña, y del concepto de *transformada wavelet* (Meyer, 1993).

Finalmente, como se indica en Bömers (2000), a finales de 1980 y 1990, de las investigaciones destacables resaltan las de Stephane Mallat, especialista en procesamiento de imágenes. De igual forma, los estudios de Yves Meyer, desarrollando las primeras *wavelet* no triviales y el desarrollo de Ingrid Daubechies, mejorando los estudios de Haar en la definición de *wavelets* ortogonales. Estos aportes llevaron al desarrollo posterior de técnicas como los filtros de espejo en cuadratura para procesamiento de señales o los algoritmos de pirámide para el procesamiento de imágenes (Meyer, 1993). En este punto es donde el desarrollo matemático previo, desde Haar hasta Daubechies, pasó a ser aplicado en el procesamiento digital de señales.

2.3. Transformada *Wavelet* y su aplicación en el DSP

El significado de *wavelet* representa literalmente onda pequeña. Como se ha comentado anteriormente, las funciones *wavelet* en el procesamiento digital de señales permiten representar cierto grupo de señales de forma más eficiente que la transformada de Fourier. La transformación *wavelet*, como se verá a continuación, permite generar una función dependiente de los parámetros de compresión y traslación a partir de una *wavelet* base (Pathak, 2009).

Desde el punto de vista matemático, como define Pathak (2009), una función ψ que satisface la condición:

$$\int_{-\infty}^{\infty} \psi(t)dt = 0, \tag{2.4}$$

representa una función *wavelet* e implica que cambia de signo en $(-\infty, \infty)$ y desaparece en $-\infty$ y $+\infty$.

Lo anterior también se puede resumir en palabras, como indican Teolis y Benedetto (1998), de la siguiente forma:

- Contiene su energía finita concentrada en el dominio del tiempo.
- Presenta cierta oscilación en el tiempo.

Por ejemplo, la función *wavelet* de Haar, mostrada en la figura 2.3, se define como:

$$\Psi(t) = \begin{cases} 1 & si \quad 0 \leq t \leq \frac{1}{2} \\ -1 & si \quad \frac{1}{2} \leq t \leq 1 \\ 0 & cualquier\ otro\ caso \end{cases} \tag{2.5}$$

Al obtener la trasformada de Fourier de la *wavelet* de Haar (2.5), como muestran Debnath y Shah (2017) y Pathak (2009), se deduce que una función *wavelet* debe cumplir además:

$$C_\psi = \int_{-\infty}^{\infty} \frac{|\hat{\psi}(\omega)|^2}{|\omega|} d\omega < \infty, \tag{2.6}$$

donde $\hat{\psi}(\omega)$ corresponde a la transformada de Fourier de $\psi(t)$. Esta condición (2.6) se conoce como admisibilidad. Para cualquier función ψ que cumpla con (2.6) se puede definir como una función *wavelet base o madre* (Pathak, 2009).

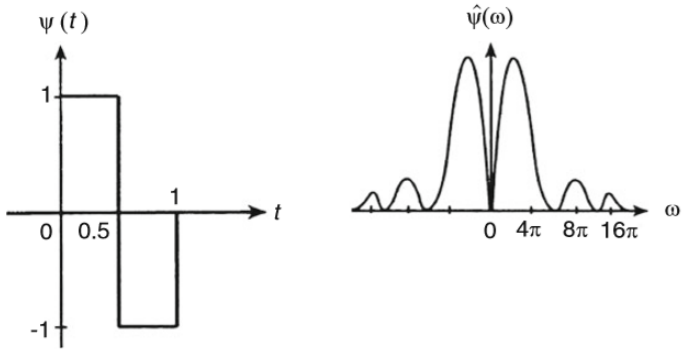

Figura 2.3: Función *wavelet* de Haar y su transformada de Fourier. Disponible en (Debnath y Shah, 2017)

2.3.1 Transformada *wavelet* continua (CWT)

Para la definición de la transformada *wavelet* continua, se toma como referencia la descripción brindada por Shukla y Tiwari (2013):

Considérese una función real o compleja de tiempo continuo $\psi(t)$ que puede describir un espacio de vectores para señales de energía finita que satisface la condición:

$$\int_{-\infty}^{\infty} |\psi(t)|^2 dt < \infty, \tag{2.7}$$

la función $\psi(t)$ describe una *wavelet* por lo que su valor promedio debe ser igual a cero y debe ser de corta duración. Esta condición, descrita anteriormente como admisibilidad, implica que:

$$\int_{-\infty}^{\infty} \psi(t) dt = 0. \tag{2.8}$$

De esta forma, la transformada continua *wavelet* (CWT) se define como:

$$CWT(s,\tau) = W(s,\tau) = \int_{-\infty}^{\infty} x(t)\psi^*_{s,\tau}(t) dt < \infty, \tag{2.9}$$

donde:

$$\psi_{s,\tau}(t) = |s|^{-\frac{1}{2}} \psi\left(\frac{t-\tau}{s}\right). \tag{2.10}$$

En la relación (2.10) la función $\psi(t)$ representa una *wavelet* base, conocida como *wavelet madre*, el operador * describe el complejo conjugado, mientras que los parámetros s, $\tau \in \mathbb{R}$ denotan los coeficientes de compresión y traslación respectivamente.

La CWT, de forma similar que la transformada de Fourier, asigna una función de una dimensión a otra función dependiente de las variables s y τ. De esta forma, mediante una función *wavelet* madre, que pueda representar adecuadamente a la señal deseada, se obtiene una nueva representación de la señal en la que los parámetros de compresión y traslación influyen. Por ejemplo, tomando una función dependiente de τ para un s constante, la transformada *wavelet* $\psi[f(t)](s,\tau)$ representa la información contenida en la señal $f(t)$ a la escala s. En la figura 2.4 se muestran algunos ejemplos de la influencia de los parámetros de compresión y traslación sobre representaciones *wavelet*.

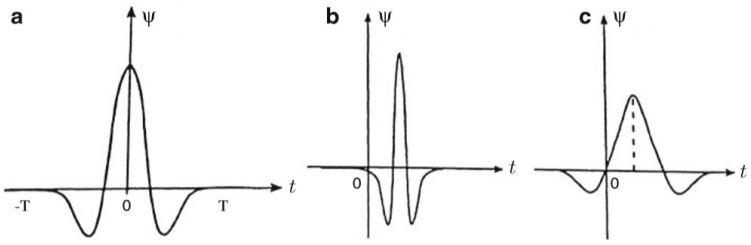

Figura 2.4: (a) Función *wavelet* base. (b) Función *wavelet* comprimida y trasladada con $0 < s << 1$ y $\tau > 0$. (c) Función *wavelet* comprimida y trasladada con $s >> 1$ y $\tau > 0$. Disponibles en (Debnath y Shah, 2017)

Basado en la definición propuesta mediante (2.9) y (2.10) se pueden anotar las siguientes propiedades, tal y como sugieren (Debnath y Shah, 2017) y (Debnath y Shah, 2015):

Sean f, g funciones reales y $W[f](s,\tau)$ la transformada *wavelet* de f con parámetros s, τ.

a. Linealidad: Sean a, b coeficientes reales:

$$W[af+bg](s,\tau) = aW[f](s,\tau) + bW[g](s,\tau) \tag{2.11}$$

b. Traslación: Sea T_c el operador de traslación tal que $T_c f(t) = f(t-c)$:

$$W[T_c f](s,\tau) = W[f](s,\tau - c) \tag{2.12}$$

c. Compresión: Sea D_c el operador de compresión tal que $D_c f(t) = \frac{1}{c} f\left(\frac{t}{c}\right)$ con $c > 0$:

$$W[D_c f](s,\tau) = \frac{1}{\sqrt{c}} W[f](\frac{s}{c}, \frac{\tau}{c}) \tag{2.13}$$

d. Paridad: Sea P el operador de paridad tal que $Pf(t) = f(-t)$:

$$W[Pf](s,\tau) = W[f](s,-\tau) \tag{2.14}$$

Es importante destacar que la transformada *wavelet* no representa por completo a la señal. Por esta razón, se hace uso de una transformación complementaria que hace uso de la función de escalamiento ϕ definida como

$$\psi_{s,\tau}(t) = |s|^{-\frac{1}{2}} \phi\left(\frac{t-\tau}{s}\right), \tag{2.15}$$

donde se obtienen los componentes de baja frecuencia con la expresión

$$V(s,\tau) = \int_{-\infty}^{\infty} x(t)\phi^*_{s,\tau}(t)dt < \infty. \tag{2.16}$$

Se dice entonces que para una señal al aplicar la transformada (2.9) se obtienen los componentes de alta frecuencia o detalles de la señal y complementariamente con la transformada (2.16) se obtienen los componentes de baja frecuencia o aproximaciones (Delgado, 2010).

2.3.2 Transformada *wavelet* discreta (DWT)

En la sección anterior se describió la transformada *wavelet* continua, ahora bien, en escenarios prácticos es conveniente considerar que las aplicaciones de procesamiento digital de señales se deben caracterizar mediante un número finito de valores. De acuerdo con lo expuesto por Debnath y Shah (2015), tomando los valores fijos s_0 y τ_0 y partiendo de la relación:

$$\psi_{m,n}(t) = s_0^{-m/2}\psi(s_0^{-m}t - n\tau_0) \tag{2.17}$$

donde m y n corresponden a enteros. Si se discretizan los parámetros tal que $s = s_0^m$ y $\tau = n\tau_0 s_0^m$ se define la transformada *wavelet* discreta de f como:

$$DWT(m,n) = W(m,n) = \int_{-\infty}^{\infty} f(t)\psi^*_{m,n}(t)dt = s_0^{-m/2}\int_{-\infty}^{\infty} f(t)\psi(s_0^{-m}t - n\tau_0). \tag{2.18}$$

Como definen Debnath y Shah (2015), esta transformada representa a una función mediante una serie finita de coeficientes que corresponden a puntos sobre un plano escala-tiempo indexados por los parámetros m y n. Para lograr mayor eficiencia en algoritmos de computación usualmente se define $s_0 = 2$ y $\tau_0 = 1$, por lo que la relación (2.17) se simplifica como

$$\psi_{m,n}(t) = 2^{-m/2}\psi(2^{-m}t - n) \tag{2.19}$$

que se conoce como la transformación *wavelet* discreta tipo Dyadic (Debnath y Shah, 2017) (Delgado, 2010).

De la relación (2.19) el término $2^{-m/2}$ representa el escalamiento de la función mientras que el término n representa el desplazamiento en el tiempo. La máxima cantidad de detalles se denomina como M y corresponde a la cantidad contenida por la señal original. De la misma forma, conforme el valor de m aumenta, la señal se hace más angosta en el tiempo y por ende adquiere mayor resolución en frecuencia (Delgado, 2010).

Tomando una función $f(t)$ cuadrado integrable y aplicando la transformación de (2.19), se puede obtener su representación como:

$$f(t) = \sum_{m,n} b_{m.n}2^{-m/2}\psi(2^{-m}t - n), \tag{2.20}$$

donde la matriz $B = [b_{m.n}]$ representa los coeficientes de la transformación *wavelet* de la función $f(t)$. Dicho de otra forma, la DWT suma todas las *wavelets* dilatadas, desplazadas y pesadas por los coeficientes $B = [b_{m.n}]$, es decir:

$$f(t) = \sum_{m,n} b_{m.n}2^{-m/2}\psi(t). \tag{2.21}$$

2.3.3 Bancos de filtros

El concepto de bancos de filtros, como se describe en Bömers (2000), consiste en un bloque que dada una señal de entrada es separada en dos o más señales con diferente rango de frecuencias. El caso más simple corresponde a un filtro pasa bajos con un filtro pasa altos que descomponen una señal en dos, una con frecuencias bajas y la otra con frecuencias altas. En la figura 2.5 se muestra un ejemplo de un banco de filtro simple. Generalmente, a la señal de baja frecuencia se le conoce como las aproximaciones de la señal original mientras que la señal de alta frecuencia se asocia con los detalles (Delgado, 2010). La importancia del estudio de los bancos de filtros es que permiten implementar la transformación *wavelet* discreta como se verá a continuación.

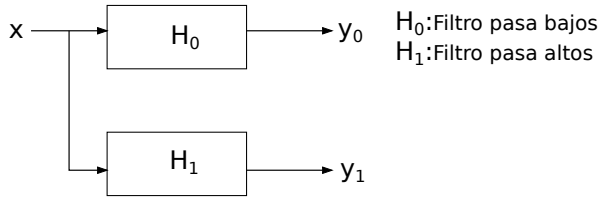

Figura 2.5: Banco de filtro simple. Disponible en (Bömers, 2000)

De la figura 2.5, si se considera la señal x como una señal digitalizada, las señales y_0 y y_1 al pasar por los filtros tendrían la misma cantidad de muestras que la señal original y por ende se tendría el doble de datos. Por esta razón, a la salida de los filtros, se aplica una operación conocida como *decimación* (del anglicismo *decimation*), cuyo símbolo se representa como $\downarrow 2$. El proceso de decimación permite obtener una señal con la mitad de la cantidad de muestras pero en el mismo rango de tiempo que la señal original (Bömers, 2000). Dicho de otro modo, permite que el vector resultante de cada descomposición tenga la mitad de la longitud del vector original.

Para los efectos prácticos, como se define en Delgado (2010), se implementa una serie de bancos de filtros recursivos donde se obtiene una descomposición en múltiples niveles de la señal original (Bömers, 2000). Si se definen $c[k]$ y $d[k]$ los vectores de coeficientes asociados a la función de escalamiento ϕ y la función *wavelet* ψ respectivamente. El filtro con los coeficientes $c[k]$ se encarga de transformar las bajas frecuencias mientras que el filtro con los coeficientes $d[k]$ realiza la transformación de altas frecuencias. Es decir,

$$\phi(t) = \sum_k c[k]\phi(2t-k) \tag{2.22}$$

$$\psi(t) = \sum_k d[k]\psi(2t-k). \tag{2.23}$$

Estos bancos de filtros se consideran como el algoritmo de aplicar la transformada *wavelet* biortognal, que consiste en filtrado recursivo de la señal mediante filtros pasa bajo y pasa alto. Se obtiene entonces N coeficientes *wavelet* y de baja frecuencia de tal forma que la representación de la señal en el dominio *wavelet* de una señal discreta ocupa el mismo espacio en disco que la señal original (Delgado, 2010).

2.3.4 Reconstrucción de la señal

La celda básica de transformación *wavelet* se muestra en la figura 2.5. Cuando las funciones *wavelet* consideradas para aplicar la transformación son finitas, los filtros H_0 y H_1 se pueden implementar mediante filtros de respuesta finita al impulso (FIR del inglés *Finite Impulse Response*). De acuerdo con el algoritmo mostrado en la figura 2.6, cada celda reduce el número de muestras a la mitad debido a la operación de decimación, de esta forma se procede aplicando el método recursivo hasta que se obtienen vectores con una única muestra. Es aquí donde se ha producido la descomposición en niveles de la señal original mediante el algoritmo de descomposición en pirámide de Mallat (Delgado, 2010).

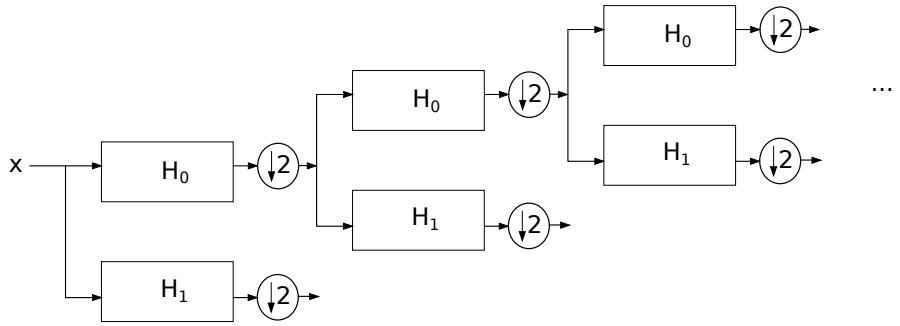

Figura 2.6: Bancos de filtros recursivos. Disponible en (Shukla y Tiwari, 2013)

Como indican Delgado (2010) y Shukla y Tiwari (2013), el banco de filtros de la figura 2.5 puede ser reversible, es decir, se puede volver a obtener la señal original. Sin embargo, debido a la operación de decimación aparece el fenómeno del *aliasing* que dificulta la reconstrucción perfecta de la señal. Entonces, se dice que la reconstrucción de la señal dependerá en gran medida de los filtros que se utilicen para implementar la transformación.

Para la reconstrucción de la señal se aplica el esquema mostrado en la figura 2.7. Como se puede apreciar, considerando el caso más simple, para reconstruir la señal se utilizan dos filtros digitales y la operación de expansión (inversa de la decimación). Con esta operación se agregan ceros en los

vectores después de cada muestra. Los filtros de reconstrucción permiten suavizar los efectos producidos por la operación de expansión sobre cada vector de coeficientes. En la figura 2.7 se muestra el proceso completo de transformación *wavelet* de la señal x mediante el algoritmo de pirámide de Mallat, se agrega un bloque de procesamiento en el dominio *wavelet* y posteriormente los bloques de reconstrucción de la señal x^* (Shukla y Tiwari, 2013).

En la figura 2.7 se ha utilizada una notación distinta para la señal reconstruida ya que se sabe que la reconstrucción no necesariamente es perfecta. Para recobrar la señal de forma perfecta, al menos teóricamente, según exponen Delgado (2010), Meyer (1993) y Shukla y Tiwari (2013), los filtros H_0, H_1, G_0 y G_1 deben cumplir las siguientes condiciones en el dominio z:

$$G_0(z)H_0(z) + G_1(z)H_1(z) = 2z^{-1} \tag{2.24}$$

$$G_0(z)H_0(-z) + G_1(-z)H_1(z) = 0. \tag{2.25}$$

Los filtros que cumplen las condiciones (2.24) y (2.25) se los conoce como filtros de espejo en cuadratura (*QMF: Quadrature Mirror Filter*) y son por lo tanto un banco de filtros ortogonales. Cuando se implementan QMF para el algoritmo de la figura 2.7 se garantiza además que los efectos del *aliasing* son cancelados (Delgado, 2010).

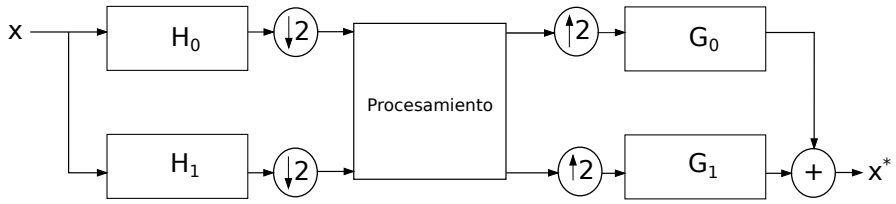

Figura 2.7: Bancos de filtros para transformación y reconstrucción. Disponible en (Shukla y Tiwari, 2013)

2.4. Criterios para selección de la función *wavelet*

Para la selección de las funciones *wavelet* adecuadas para este estudio se consideran algunos de los aspectos resaltados por Delgado (2010) y Bömers (2000). En estos estudios previos se consideran las funciones *wavelet* para el análisis de señales musicales, sin embargo las señales de voz guardan relación con las mismas por lo que sus argumentos son válidos igualmente. Tanto las señales musicales como las de voz se asemejan en que su forma de onda se puede describir como ondas suaves y su espectro de frecuencia es similar.

Como indica Bömers (2000) para aplicar la transformada *wavelet* es importante considerar la calidad de la descomposición. Si se toma como referencia una función *wavelet* que no puede representar las características de la señal a descomponer, sus coeficientes no serán una representación válida, es decir, la mayoría de coeficientes serían iguales a cero. Por lo tanto, en este caso la representación de la señal mediante la transformada *wavelet* podría considerarse no válida y consecuentemente cualquier procesamiento que se le aplique. De acuerdo con lo expuesto anteriormente, deben considerarse funciones *wavelet* que muestren transiciones suaves en su onda.

Se desea que las funciones *wavelet* seleccionadas posean una fase lineal (Delgado, 2010). Los filtros encargados de descomponer la señal mediante la transformada *wavelet* deben tener una fase lineal, de lo contrario algunas de las frecuencias podrían sufrir retrasos en el dominio *wavelet*. En este escenario entra en consideración la distorsión de la señal, dado que al aplicar posteriormente la transformada inversa algunos coeficientes podrían haber recibido alguna modificación debido a las frecuencias con retraso (Bömers, 2000).

2.5. *Wavelets* comunes y sus propiedades

En esta sección se describen algunas de las funciones *wavelet* más comunes de acuerdo con las propiedades que definen Delgado (2010) y Bömers (2000).

2.5.1 *Wavelets* de Daubechies

Estas funciones *wavelet* fueron descritas por Ingrid Daubechies alrededor de 1980. Su construcción involucra el uso de filtros ortogonales con respuestas planas alrededor de cero y de la mitad de la frecuencia de muestreo. La mayoría de estas funciones presentan asimetría. Para efectos prácticos, se suelen utilizar órdenes altos para mejorar la separación de las bandas de frecuencia. Se implementan mediante filtros FIR por lo que su respuesta en fase es lineal.

2.5.2 *Wavelets* de Coiflets

Las funciones *wavelet* de Coiflets fueron de igual forma desarrolladas por Ingrid Daubechies para R. Coifman. En términos generales, presentan las mismas características que las de Daubechies, sin embargo con restricciones adicionales en la función de escalamiento donde se cuenta con más momentos de desvanecimiento. Debido a esta propiedad los coeficientes de aproximación pueden obtenerse mediante las mismas muestras de la señal pero con la limitante de que los filtros requieren órdenes mayores y se pierde eficiencia.

2.5.3 *Wavelets* de Symmlets

Las funciones de Symmlets fueron desarrolladas por Ingrid Daubechies y presentan propiedades muy similares a las funciones *wavelet* del mismo nombre. Específicamente presentan mejoras en la simetría y la fase lineal.

2.5.4 *Wavelets* Bi-ortogonales

Estas funciones *wavelet* presentan características como fase lineal así como funciones *wavelet* y de escalamiento distintas, tanto para descomposición como para reconstrucción. Permiten crear transformaciones simétricas. Para efectos prácticos, los filtros de descomposición y reconstrucción pueden ser diseñados bajo diferentes especificaciones.

2.6. Reducción de ruido en el dominio *wavelet*

2.6.1 Algoritmo para eliminación de ruido

El algoritmo para la eliminación de ruido, mediante el método de recorte de coeficientes en el dominio *wavelet*, se puede representar en los siguientes pasos, tal y como describen Donoho y Johnstone (1994), Delgado (2010), Lin y Cai (2010). Así mismo, se plantea en el esquema de la figura 2.8.

1. Aplicar la transformada *wavelet*, considerando alguna función *wavelet* base, a la señal con ruido para obtener los coeficientes w_k.

2. Aplicar algún método de recorte de coeficientes tal que suprima aquellos correspondientes a las componentes del ruido.

3. Realizar la transformación *wavelet* inversa a la señal y obtener la misma sin las componentes de ruido iniciales.

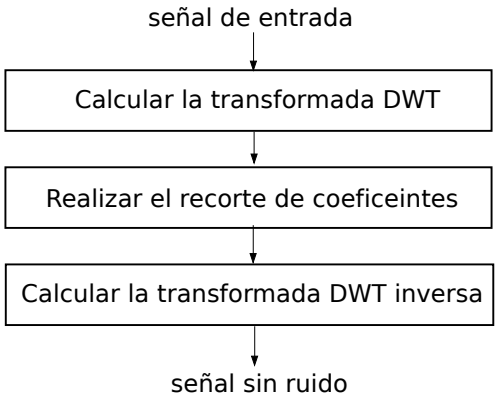

Figura 2.8: Esquema del algoritmo de recorte de coeficientes. Disponible en (Hidayat et al., 2018)

2.6.2 Recorte fuerte de coeficientes

La técnica de recorte fuerte de coeficientes o conocida también como *hard thresholding* se define de la siguiente forma (Donoho y Johnstone, 1994):

$$\eta_H(w_k, T) = \begin{cases} w_k & si \quad |w_k| \geq T \\ \\ 0 & si \quad |w_k| < T \end{cases} \tag{2.26}$$

donde en la ecuación (2.26) el límite para recorte de coeficientes se representa con T y los coeficientes k-ésimos de la señal con w_k.

2.6.3 Recorte suave de coeficientes

La técnica de recorte suave de coeficientes o conocida también como *soft thresholding* se define de la siguiente forma (Donoho y Johnstone, 1994):

$$\eta_S(w_k, T) = \begin{cases} w_k - T & si \quad w_k \geq T \\ \\ 0 & si \quad |w_k| < T \\ \\ w_k + T & si \quad w_k \leq -T \end{cases} \tag{2.27}$$

donde en la ecuación (2.27) el límite para recorte de coeficientes se representa con T y los coeficientes k-ésimos de la señal con w_k.

2.6.4 Recorte medio de coeficientes

La técnica de recorte medio de coeficientes o conocida también como *mid thresholding* se define de la siguiente forma (Walden et al., 1998):

$$\eta_M(w_k, T) = \begin{cases} 2(|w_k| - T)_* & si \quad |w_k| < 2T \\ \\ |w_k| & otros\ casos \end{cases} \tag{2.28}$$

donde en la ecuación (2.28) el límite para recorte de coeficientes se representa con T, los coeficientes k-ésimos de la señal con w_k, y la función $(|w_k| - T)_*$ como:

$$(|w_k| - T)_* = \begin{cases} |w_k| - T & si \quad |w_k| > T \\ \\ 0 & otros\ casos \end{cases} \tag{2.29}$$

2.6.5 Definición del umbral de recorte

En las ecuaciones (2.26), (2.27), (2.28) se denominó el umbral de recorte como T, este valor fue definido como el límite universal por Donoho y Johnstone (1994) y se describe para una señal de longitud N como:

$$T = \sigma_n \sqrt{2log(N)}, \tag{2.30}$$

donde σ_n corresponde al nivel de ruido definido como una aproximación de la desviación estándar de los coeficientes *wavelet* de la señal, expresado matemáticamente según Delgado (2010) como:

$$\sigma_n = \frac{MAD}{0,6745}, \tag{2.31}$$

y el factor MAD hace referencia a la mediana de la desviación absoluta estimada a la escala más fina.

2.7. Filtros Wiener

Los filtros Wiener, como indica Vaseghi (2008), son una de las técnicas más comunes para eliminación de ruido en señales de voz. Para estudiar su teoría, primero se considera la salida del filtro como

$$\hat{x}(m) = \sum_{i=0}^{P} \omega(i) y(m - i) \tag{2.32}$$

donde $\omega(k)$ representa los coeficientes del filtro, $y(m)$ la señal de voz y $x(m)$ la señal de voz estimada sin ruido. Los coeficientes del filtro se obtienen como

$$\omega = R_{yy}^{-1} r_{xy} \tag{2.33}$$

donde R_{yy} indica la auto-correlación de la señal con ruido y, mientras que r_{xy} determina la correlación cruzada entre la señal sin ruido x y la señal ruidosa y.

Para el caso de señales de voz y ruido no correlacionados, los coeficientes del filtro se aproximan con

$$\omega = [R_{xx} + R_{nn}]^{-1} r_{xx} \tag{2.34}$$

con R_{xx} la correlación de la señal de voz, R_{nn} la correlación del ruido y R_{xx} la autocorrelación de la señal de voz.

Considerando el análisis en el dominio de la frecuencia, la relación (2.34) se expresa como

$$W(k) = \frac{P_{XX}(k)}{P_{XX}(k) + P_{NN}(k)} \tag{2.35}$$

donde $P_{XX}(k)$ y $P_{NN}(k)$ indica la densidad de potencia espectral de la señal y el ruido mientras que $W(k)$ es la respuesta del filtro con la variable discreta k.

La relación (2.35) se puede expresar en términos del SNR si se divide por $P_{NN}(k)$, es decir,

$$W(k) = \frac{SNR(k)}{SNR(k) + 1}. \tag{2.36}$$

En la figura 2.9 se muestra el esquema en bloques del algoritmo de los filtros Wiener para eliminación de ruido en señales de voz.

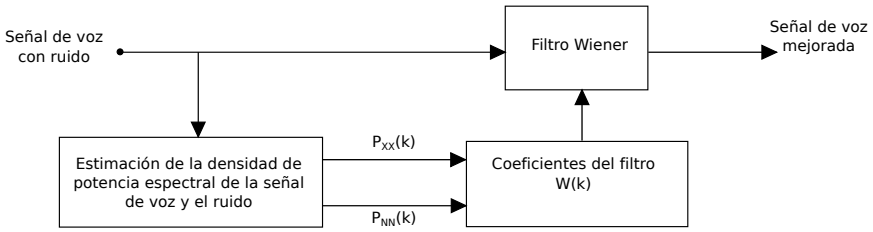

Figura 2.9: Diagrama de bloques del algoritmo del filtro Wiener. Disponible en (Vaseghi, 2008)

2.8. Relación señal a ruido, error cuadrático medio y evaluación porcentual de la calidad de voz

2.8.1 Error Cuadrático Medio

El error cuadrático medio o por sus siglas en inglés MSE, constituye un índice de medición del promedio de las diferencias entre el valor real y el valor estimado. Matemáticamente, se puede definir según la relación (2.37) descrita en (Zhang et al., 2018) como

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(S - S^*)^2 \tag{2.37}$$

de la cual n representa el número de muestras, S el valor real y S^* el valor estimado.

Para el MSE se desea encontrar el valor más cercano a cero, entre más cercano a cero sea el valor obtenido, mejor desempeño tiene el estimador.

2.8.2 Relación Señal a Ruido

La relación señal a ruido usualmente consiste en un medidor de la calidad de una señal con respecto al ruido. Para su cálculo usualmente basta con conocer la potencia de la señal y la potencia del ruido asociada a ese señal. El SNR, por sus siglas en inglés, es una medición de suma importancia para la calidad de los sistemas de comunicación que se utiliza para diseñar la potencia requerida por los transmisores del sistema (Haykin y Moher, 2006). La relación señal a ruido se puede definir teóricamente como

$$SNR = \frac{E[s^2(t)]}{E[n^2(t)]} \tag{2.38}$$

en donde, en la relación (2.38), la señal se representa con $s(t)$, el ruido con $n(t)$ y el operador promedio con $E[.]$.

De una forma más práctica, el SNR se puede calcular también en dB como indica Zhang et al. (2018), descrito en la ecuación (2.39)

$$SNR(dB) = 10log\left(\frac{\sum_{i=1}^{N} s^2(k)}{\sum_{i=1}^{N} n^2(k)}\right) \tag{2.39}$$

donde se hace la relación de la potencia de la señal s y la potencia del ruido n.

Al contrario que el MSE, es deseable obtener valores altos de SNR, lo cual indica que la potencia de la señal es mucho mayor que el ruido y por lo tanto no se verá afectada por el mismo.

2.8.3 Evaluación Porcentual de la Calidad de Habla

Los parámetros anteriores como el MSE y el SNR permiten medir objetivamente la efectividad de algoritmos así como de redes de comunicaciones. Para el caso en que se cuenta con señales de voz es necesario aplicar otra medición centrada en la calidad de la señal, ya que el objetivo es contar con una señal capaz de transmitir el mensaje correctamente. La medición del PESQ (*Perceptual Evaluation of Speech Quality*), tal y como se describe en UIT-T (2001), proviene en gran parte de experimentos subjetivos de la calidad de escucha.

PESQ es un estándar, recomendación ITU P.862, para medición objetiva de la calidad de señales de voz de forma tal que predice los resultados considerando pruebas de escucha subjetivas gracias a su modelo matemático de predicción. Para la medición de PESQ se utiliza un modelo sensitivo que compara la señal original con la señal degrada a la salida del canal de comunicación (Praveen, 2012). De esta forma, al final del procesamiento del algoritmo de medición PESQ se obtiene una nota con rango -0,5 a 4,5, la cual expresa la calidad de la señal de voz del canal de comunicaciones, o en el caso de esta investigación, del algoritmo de procesamiento de voz. Entre más alto el valor de la nota de PESQ mejor es la calidad de la señal de voz procesada.

Para esta investigación se sale del enfoque de estudio dar una descripción detallada del algoritmo PESQ dado la complejidad del mismo, para mayor detalle se pueden consultar (UIT-T, 2001) y (Praveen, 2012).

2.9. Sistemas de procesamiento de señales de voz

En los sistemas de procesamiento de la voz humana generalmente las señales son representadas de forma digital, por esta razón, estos sistemas pasan a ser objeto de estudio del procesamiento digital de señales como un caso especial de la disciplina. Para el procesamiento, se utilizan diversas representaciones y algoritmos matemáticos que permiten modelar los comportamientos deseados. El procesamiento de señales de voz se puede clasificar en etapas como reconocimiento, síntesis, mejoramiento y codificación (Farouk, 2014).

Algunas aplicaciones de los sistemas de procesamiento de voz, como describe Farouk (2014), suelen ser sistemas de comunicaciones basados en voz, sistemas de traducción automáticos y sistemas de lectura y compresión. Para llevar a cabo este tipo de procesamiento usualmente se aplica alguna transformada a la señal de voz, de esta forma se cambia su naturaleza y se exploran otras de sus cualidades. Tal es el caso de realizar el procesamiento de la señal en el dominio de la frecuencia o el dominio *wavelet*.

Los sistemas de reconocimiento de voz tratan sobre la identificación de palabras u oraciones mediante alguna máquina desarrollando algoritmos específicos (Hidayat et al., 2018). En estos sistemas, antes de llevar a cabo el proceso de reconocimiento de voz, se requiere que la señal esté libre de perturbaciones de ruido que afecten la identificación de su contenido. Los estudios en este campo han propuesto aplicar algún método de eliminación de ruido para garantizar precisión en el sistema de reconocimiento de voz (Daubechies, 1992) (Hidayat et al., 2018) (Farouk, 2014). Uno de los métodos empleados consiste en la eliminación de ruido mediante recorte de coeficientes en el dominio *wavelet*.

3 Funciones *Wavelet* para procesamiento de señales de voz

3.1. Características deseables en *wavelets* para procesamiento de señales de voz

Es importante destacar que no todas las funciones *wavelet* permiten obtener una transformación válida para el análisis de señales (Meyer, 1993). A continuación se describen las características deseables en una función *wavelet* para el caso de análisis de señales de voz. Estas propiedades se basan, específicamente, en los estudios desarrollados por (Delgado, 2010), (Deng y Jiang, 2012) y (Long et al., 2004).

- En primer lugar se desea que la función *wavelet* cuente con forma similar a una señal de voz. En este punto se habla de transiciones curvas suaves recordando que las señales de voz son similares a las funciones sinusoidales con una fundamental y armónicos. Por ejemplo en la figura 3.1(a) se muestra la función *wavelet* de Haar como una función con transición brusca y poco apta para tratamiento de señales de voz. De la misma forma, en la figura 3.1(b), con la función db5 se presenta una forma aconsejable para procesamiento de señales de voz.

- Es deseable que cuenten con respuesta de fase lineal, de lo contrario algunas frecuencias pueden sufrir atrasos en el dominio *wavelet* durante el procesamiento.

- Se recomienda que cuente con mayor concentración de energía en los primeros niveles de descomposición o bien en algunos pocos, por lo que se tiene menos coeficientes significativos, lo que implica que a escalas mayores los coeficientes son muy cercanos a cero. Además, al contar con menores niveles se hace más eficiente el procesamiento. Por ejemplo, la figura 3.2 muestra la relación entre la potencia y los niveles de descomposición de una señal de audio mediante la transformada *wavelet* considerando las funciones db2, db5 y db10, como se observa, la mayoría de la potencia se presenta en pocas escalas.

- La cantidad de coeficientes de cada *wavelet* por cada nivel es otro elemento que puede influir si el tiempo de procesamiento es un factor de

peso. A menores coeficientes menor tiempo de procesamiento. Esta propiedad también se observa en la figura 3.2, donde de acuerdo al estudio de Delgado (2010), la función db2 cuenta con 4 coeficientes, db5 cuenta con 10 coeficientes y db10 cuenta con 20 coeficientes.

- *Wavelets* con grados mayores usualmente dan mejores resultados contra el ruido.

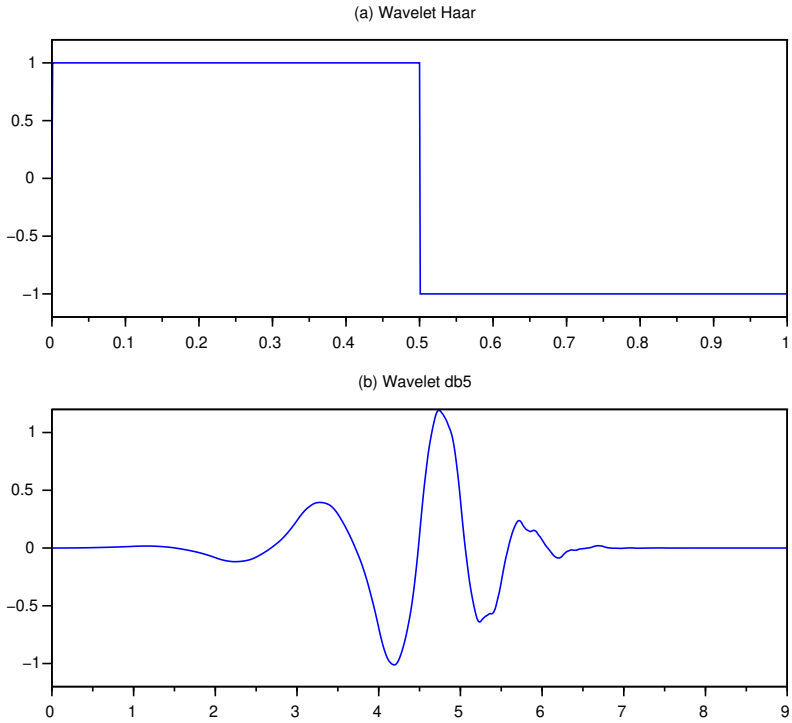

Figura 3.1: (a) Función *wavelet* de Haar (b) Función *wavelet* de Daubechies orden 5

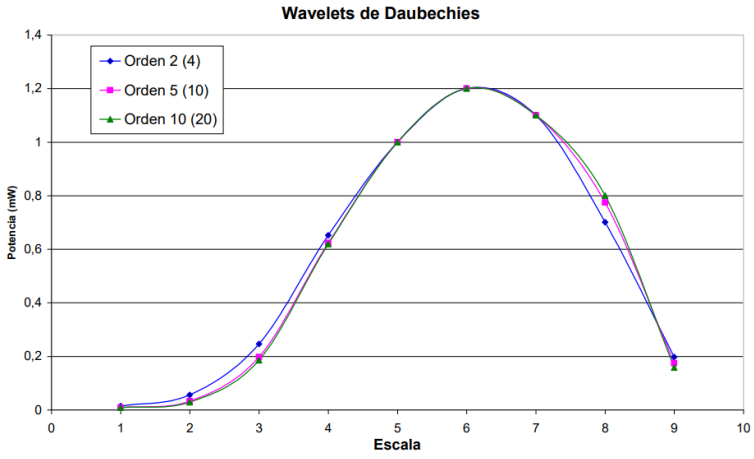

Figura 3.2: Potencia en términos de la escala para las funciones *wavelets*: db2, db5 y db10. Disponible en (Delgado, 2010)

3.2. *Wavelets* recomendadas en otros estudios

Estudios previos han implementado la transformada en el dominio *wavelet* para aplicar el método de eliminación de ruido en señales de audio y/o voz. Por ejemplo, se puede mencionar el caso de la investigación llevada a cabo por Delgado (2010) denominada *Restauración de grabaciones de audio antiguas mediante el método de recorte de coeficientes en el dominio wavelet*, el cual considera como posibles funciones *wavelet* para tratamiento de señales de audio: db10, coif5, sym8 y bior3.3. En este estudio, coif5 termina siendo la función seleccionada para su algoritmo.

Por su parte, Deng y Jiang (2012) en su estudio *Selection of optimal wavelet basis for signal denoising* (Selección de la función *wavelet* óptima para eliminación de ruido en señales) recomiendan el uso de las siguientes funciones: sym4, sym8, db5, db10, bior3.7, bior6.8. Se resalta que sym8 y bior6.8 son siempre recomendadas para señales de baja frecuencia.

En el estudio de Long et al. (2004) *Selection of the best wavelet base for speech signal* (Selección de la mejor *wavelet* base para señales de voz) se consideran las funciones: haar, db2, dmey, bior2.4, coif3, sym4. Su principal conclusión indica que la función *wavelet* dmey otorga los mejores resultados.

Es importante destacar que cada uno de estos autores han considerado ciertos parámetros en su investigación, por lo que sus resultados no necesariamente implican que son aplicables en cualquier escenario. En el caso de (Delgado, 2010) se trabaja con señales de audio correspondientes a música de orquesta, con un comportamiento similar a las señales de voz pero con anchos

de banda mayores y gran presencia de tonalidades. En el caso de (Deng y Jiang, 2012) se analiza la curva de umbral y energía de las *wavelet* de estudio, en donde la curva expone la relación entra la energía de la transformada *wavelet* de la señal y la cantidad de coeficientes para su representación, donde es deseable la mayor parte de la energía en las primeras escalas y menos coeficientes significativos. A su vez, (Long et al., 2004), determinan la mejor *wavelet* midiendo el error máximo y el error promedio de cada nivel de descomposición cuando se aplica la transformada *wavelet* a una señal de voz que se descompone en cinco niveles.

3.3. Descripción del algoritmo para determinar las *wavelets* a considerar

3.3.1 Determinación de los niveles de descomposición

Un factor importante, al igual que la función *wavelet*, corresponde a los niveles en que se descompone la señal mediante la transformada *wavelet*. La cantidad de niveles está directamente relacionado con el ancho de banda de la señal que se procesa. Como se indica en Rémy y Letamendia (2014), el ancho de banda de una señal de voz en alta definición (voz HD en LTE) corresponde a 7 kHz, para este estudio se toma ese valor como referencia y se define el ancho de banda aproximado de la señal de voz como 8 kHz. Considerando la tabla 3.1 se observa que al hacer una descomposición de cuatro niveles, mediante la transformada *wavelet*, se obtiene los coeficientes de aproximación `cA4` y los coeficientes de detalle `cD4, cD3, cD2, cD1` referenciados con respecto al ancho de banda que encierra cada uno.

Cuadro 3.1: Separación en frecuencia para cuatro niveles de descomposición

Coeficientes		**cA4**		**cD4**		**cD3**		**cD2**		**cD1**	
Frecuencia (kHz)	0		0,5		1		2		4		8

Como se observa del cuadro 3.1, los coeficientes `cA4` contienen la aproximación de baja frecuencia de la señal comprendida entre 0 y 500 Hz. Mientras que los detalles `cD4, cD3, cD2, cD1` contienen la información comprendida entre 500 Hz y 8 kHz. Si se aplica otra descomposición más, alcanzando cinco niveles, la aproximación de baja frecuencia se encontraría entre 0 y 250 Hz, intervalo que probablemente no representa fielmente una señal de voz, como se mostrará en el capítulo siguiente.

3.3.2 Experimento para selección de las funciones *wavelet*

Para determinar las cinco funciones *wavelet* que se implementan en el método de eliminación de ruido de este estudio, se lleva a cabo previamente un experimento de selección. Las funciones *wavelet* que se consideran para este experimento, tomando las recomendaciones de los estudios de Delgado (2010), Deng y Jiang (2012) y Long et al. (2004), son:

- db2
- db5
- db10
- coif3
- coif5
- sym4
- sym8
- bior2.4
- bior3.3
- bior3.7
- bior6.8
- dmey

El experimento consiste en tomar un conjunto de cincuenta señales de voz, aplicar la transformada *wavelet* considerando cada una de estas funciones, aplicar la transforma inversa y reconstruir la señal. Como se cuenta con la señal original y la señal reconstruida se puede determinar parámetros de medición del error como el MSE. Se realiza las mediciones para las funciones *wavelet* listadas anteriormente y se selecciona la función *wavelet* que cuente con menor medición en promedio de MSE por cada familia, de esta forma se cuenta con variabilidad con lo que respecta a las familias de funciones *wavelet*. En la tabla 3.2 se muestra la lista de funciones *wavelet* y sus mediciones ordenadas de forma ascendente por familia. Se muestra el valor promedio, la desviación estándar, el valor máximo y el valor mínimo medidos considerando las cincuenta señales de voz.

Cuadro 3.2: Medición del MSE con cuatro niveles de descomposición

Wavelet	**Promedio MSE**	**Desv Est MSE**	**Max MSE**	**Min MSE**
bior3.3	$4,81x10^{-34}$	$1,34x10^{-34}$	$8,45x10^{-34}$	$2,17x10^{-34}$
bior3.7	$6,42x10^{-34}$	$1,89x10^{-34}$	$1,19x10^{-33}$	$3,16x10^{-34}$
bior2.4	$1,11x10^{-33}$	$3,18x10^{-34}$	$2,00x10^{-33}$	$5,22x10^{-34}$
bior6.8	$1,25x10^{-27}$	$3,52x10^{-28}$	$2,31x10^{-27}$	$6,42x10^{-28}$
coif5	$1,72x10^{-33}$	$4,93x10^{-34}$	$3,07x10^{-33}$	$7,91x10^{-34}$
coif3	$1,73x10^{-33}$	$5,11x10^{-34}$	$3,38x10^{-33}$	$7,91x10^{-34}$
db5	$7,99x10^{-34}$	$2,35x10^{-34}$	$1,49x10^{-33}$	$3,64x10^{-34}$
db2	$1,16x10^{-33}$	$3,51x10^{-34}$	$2,22x10^{-33}$	$5,30x10^{-34}$
db10	$1,42x10^{-33}$	$4,26x10^{-34}$	$2,85x10^{-33}$	$6,43x10^{-34}$
sym8	$1,13x10^{-27}$	$3,14x10^{-28}$	$1,91x10^{-27}$	$4,81x10^{-28}$
sym4	$6,50x10^{-27}$	$1,97x10^{-27}$	$1,31x10^{-26}$	$2,99x10^{-27}$
dmey	$5,63x10^{-07}$	$1,56x10^{-07}$	$9,51x10^{-07}$	$2,50x10^{-07}$

3.4. Funciones *wavelets* seleccionadas

De acuerdo con lo expuesto en la sección anterior, específicamente en la tabla 3.2, las cinco funciones seleccionadas para aplicar en el método de eliminación de ruido mediante recorte de coeficientes en el dominio *wavelet* son:

- bior3.3
- coif5
- db5
- sym8
- dmey

En la figura 3.3 se muestran las curvas de las funciones *wavelet* seleccionadas. Se destaca que para las funciones *wavelet* biortogonales, por definición, se cuenta con una función para descomposición y otra función para reconstrucción, por lo que esta *wavelet* se separa en dos como en el caso con bior3.3. Para las funciones, se puede observar como su forma con suave curvatura es recomendable para procesamiento de las señales de voz.

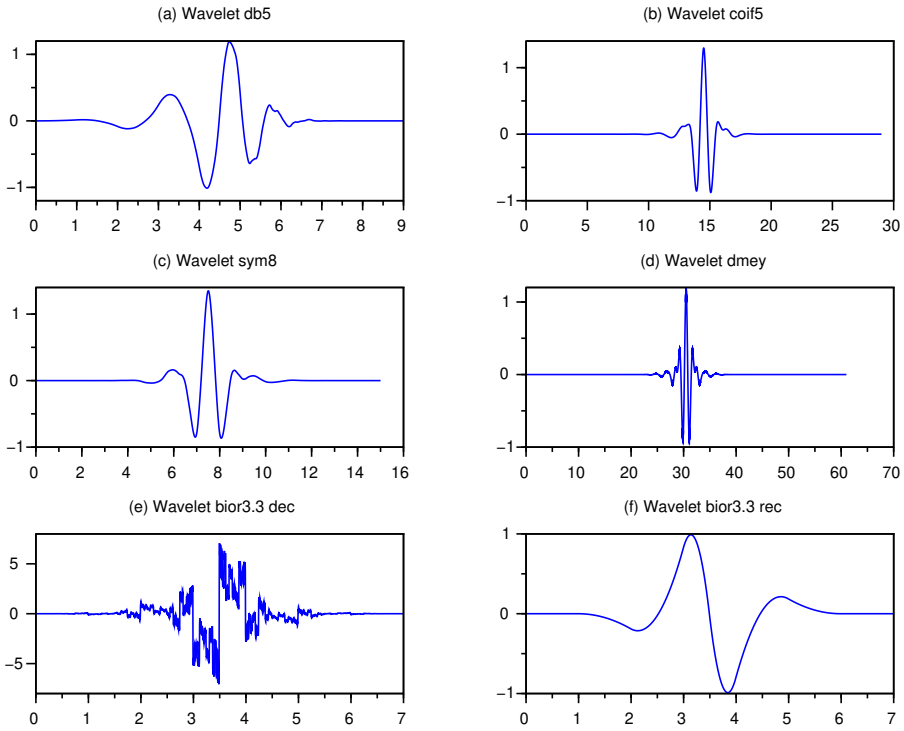

Figura 3.3: (a) *Wavelet* db5. (b) *Wavelet* coif5. (c) *Wavelet* sym8. (d) *Wavelet* dmey. (e) *Wavelet* bior3.3 descomposición. (f) *Wavelet* bior3.3 reconstrucción

4 Desarrollo de señales de voz de prueba

4.1. Descripción de las muestras de señal de voz

Para esta investigación se hizo uso de una biblioteca de cincuenta señales de voz disponibles en (Festvox, 2003). Esta biblioteca es un fragmento de una serie de bases de datos de señales de voz desarrollados por el Instituto de Tecnologías de Lenguaje de la Universidad Carnegie Mellon.

Como describen Kominek y Black (2003), estas bibliotecas fueron diseñadas con el objetivo de que puedan ser un estándar en investigaciones concernientes a señales de voz. Las señales de un único hablante, masculino o femenino, fueron grabadas bajo condiciones de estudio por lo que cuentan con excelente calidad. Las señales contemplan el balance fonético adecuado de las frases comunes en el idioma inglés. Además, su distribución es software libre.

Para esta investigación se utilizaron cincuenta de estas señales de voz de un hablante masculino en inglés. Se consideró una base satisfactoria para cumplir los objetivos de la investigación de acuerdo con la recomendación del comité asesor.

4.2. Consideraciones para la señal de voz

Existen algunas consideraciones teórico-prácticas que se toman en cuenta para el desarrollo de las señales de voz de prueba, las cuáles se describen a continuación:

(a) Se ha tomado una base de cincuenta señales de voz con frases comunes en inglés. Estas señales se pueden encontrar en (Festvox, 2003) y forman parte de una biblioteca de señales dedicadas para investigaciones en el campo de procesamiento de señales de voz.

(b) La base de cincuenta señales de voz se encuentra en formato .wav (*Waveform Audio Format*) lo cual facilita el procesamiento con los software Audacity y Scilab.

(c) La duración de las señales es variada, sin embargo ninguna sobre pasa los 10 segundos. La frecuencia de muestreo de las señales es de 16 kHz y la velocidad de bits 256 kbit/s.

(d) El espectro en frecuencia de una de las señales de voz se muestra en la figura 4.1.

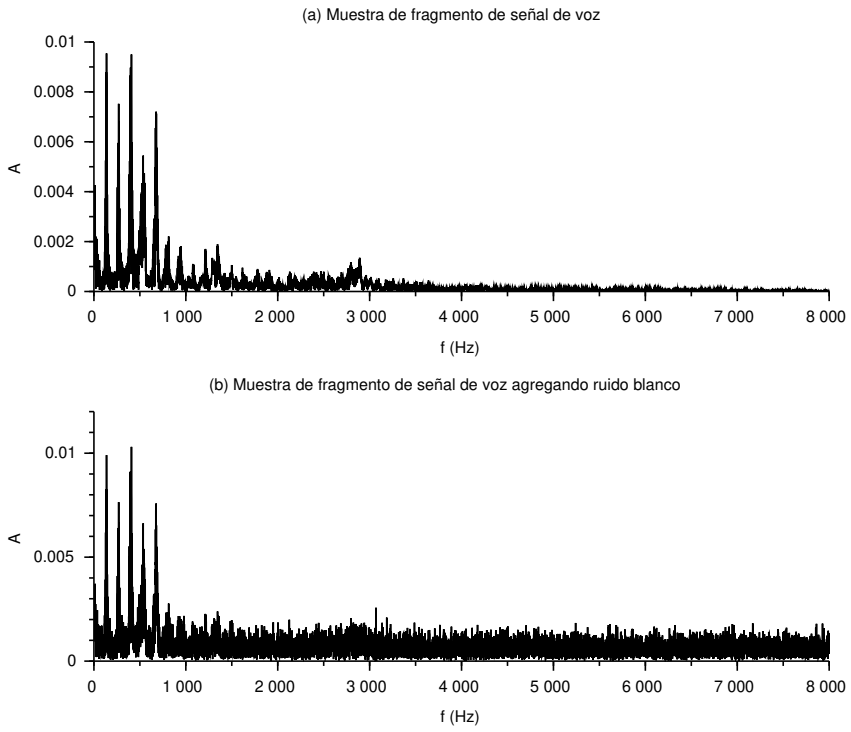

Figura 4.1: (a) Espectro de frecuencia de un fragmento de muestra de señal de voz sin ruido. (b) Espectro de frecuencia de un fragmento de muestra de señal de voz con ruido blanco agregado

4.3. Ruidos de estudio seleccionados

Como se estableció anteriormente se ha decidido considerar tres tipos de ruido particulares: blanco, rosa y *babble*. Sobre los mismos se puede dar la siguiente descripción:

(a) El ruido *babble* se toma del sitio web (Koenig, 2017) y se procesa mediante Audacity para cumplir con el criterio de formato .wav. Su respuesta en frecuencia se muestra en la figura 4.2(a).

(b) El ruido blanco en formato .wav se toma del software Audacity, el cual cuenta con sonidos muestra donde se encuentra disponible el ruido blanco. Su espectro de frecuencia se muestra en la figura 4.2(b).

(c) El ruido rosa en formato .wav se toma igualmente del software Audacity. Su espectro de frecuencia se muestra en la figura 4.2(c).

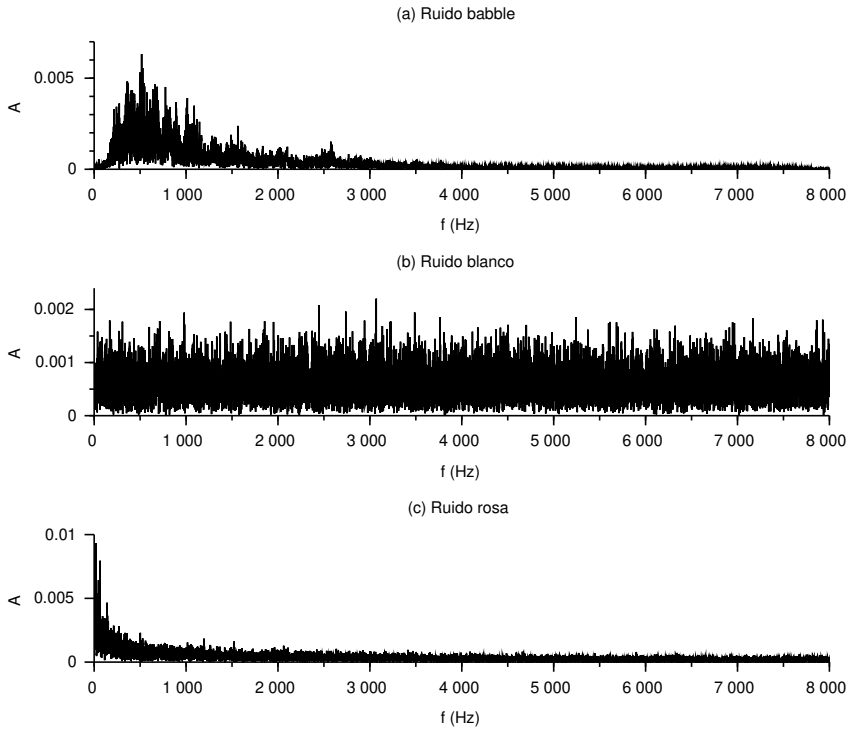

Figura 4.2: Espectro de frecuencia de las señales de ruido en formato .wav. (a) Ruido *babble* (b) Ruido blanco (c) Ruido rosa

4.4. Implementación práctica

Se parte los archivos .wav correspondientes a las señales de voz y a las señales de los ruidos blanco, rosa y *babble* y se desea obtener la mezcla de cada señal de voz con cada tipo de ruido.

4.4.1 Mediante software Audacity

El software Audacity se implementó en esta investigación principalmente para manipular los ruidos de estudio. De tal forma que las señales .wav de los ruidos blanco, rosa y *babble* fueron generadas en este entorno. En la figura 4.3 se muestra como ejemplo las señales de los ruidos en el software Audacity.

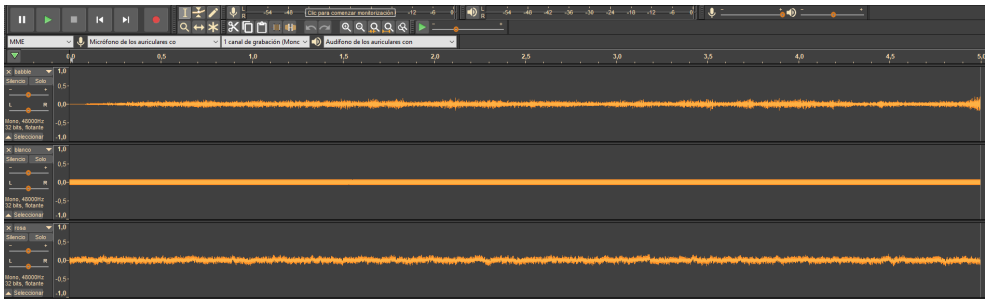

Figura 4.3: Captura de pantalla del proceso de manipulación de señales de ruido en Audacity

Entonces, en resumen, el software Audacity permite manipular fácilmente los archivos de audio y generar los .wav para su posterior procesamiento con el algoritmo de estudio mediante el software Scilab.

4.4.2 Mediante software Scilab

Dado que se desea contar con la señal de voz inicial y cada señal de ruido por separado, se decidió realizar la mezcla mediante el software Scilab. Para este proceso se debe considerar el siguiente código:

```
1  // señales y ruidos .wav
2  my_voz_wav = senal;
3  my_blanco_wav = "C:\Users\Admin\Documents\Tesis\Audio_signals\50_frases_ingles\blanco.wav";
4  my_rosa_wav = "C:\Users\Admin\Documents\Tesis\Audio_signals\50_frases_ingles\rosa.wav";
5  my_babble_wav = "C:\Users\Admin\Documents\Tesis\Audio_signals\50_frases_ingles\babble.wav";
6
7  // numero de muestras que depende de la señal de voz
8  n = length(s_voz)
9
10 // mezcla de ruido y senal de voz
11 [s_blanco,Fs,bits] = wavread(my_blanco_wav,n);
12 gain_voz = get_gain_voz(s_voz, s_blanco, snr_in);
13 s_voz = gain_voz .* s_voz;
14 s_n = s_blanco + s_voz;
```

Como se observa, mediante el entorno Scilab se hace la lectura de las señales de audio .wav con la función `wavread`. De esta forma se tiene una variable para la señal de voz y los tres ruidos de estudio. Se determinan las muestras mediante la señal de voz seleccionada. Finalmente, la mezcla de cada

señal de voz se realiza una vez que se determina la ganancia que se debe aplicar a la señal para obtener el SNR deseado para la aplicación del algoritmo. En este caso, el código mostrado es un ejemplo del procesamiento llevado a cabo mediante las funciones diseñadas para la ejecución del algoritmo. En el capítulo siguiente se brindan más detalles sobre cada función.

5 Descripción y resultados experimentales del método propuesto

5.1. Descripción de la implementación práctica del algoritmo

En esta sección se detallan los componentes esenciales sobre la lógica de procesamiento, considerada el eje central de este estudio. Como se ha indicado, se hace uso del entorno Scilab debido a que cuenta con su propia librería sobre funciones *wavelets*, además es un entorno software libre. Vale la pena destacar que se buscó desarrollar una función suficientemente flexible y automática tal que sólo bastara cambiar los parámetros y señales iniciales para obtener los resultados deseados según cada caso de estudio.

En el siguiente extracto se muestra la función desarrollada en Scilab para llevar a cabo el algoritmo de eliminación de ruido en el dominio *wavelet*.

```
1 function [ss, s_n, Fs, bits] = wavelet_denoising(senal, ruido, wavelet,
  ↪  recorte, snr_in, graph_enable, output_wav_enable)
```

De la función anterior se describen sus argumentos y resultados como:

- `ss`: señal de salida procesada
- `s_n`: señal de entrada
- `Fs`: frecuencia de muestreo de las señales
- `bits`: resolución en bits de las señales
- `senal`: señal de entrada
- `ruido`: tipo de ruido a procesar
- `wavelet`: función *wavelet*
- `recorte`: tipo de recorte

- `snr_in`: SNR entre la señal de entrada y el ruido seleccionado
- `graph_enable`: activar gráficos de las señales de entrada/salida
- `output_wav_enable`: generar audios de las señales de entrada/salida

A partir de la función `wavelet_denoising()` es posible aplicar el algoritmo según las señales de entrada y los parámetros deseados. Otra ventaja de desarrollar esta función es la facilidad para correr el algoritmo de forma cíclica alternando las señales de entrada y parámetros. A continuación se muestran los detalles más relevantes de esta función.

Como se observa en las siguientes líneas de código, inicialmente se hace el llamado a las demás funciones necesarias para aplicar el algoritmo. A continuación se definen las rutas de acceso a los ruidos de estudio. Se definen los niveles de descomposición y el límite de recorte a implementar. Posteriormente, se efectúa la lectura de la señal de voz así como su cantidad de muestras. Las lecturas de las señales en formato .wav se realizan mediante la función `wavread()`.

```
1  //funciones sci
2  exec('C:\Users\Admin\Documents\Tesis\Scripts\limit_escala.sci');
3  exec('C:\Users\Admin\Documents\Tesis\Scripts\limit_universal.sci');
4  exec('C:\Users\Admin\Documents\Tesis\Scripts\recorte_fuerte.sci');
5  exec('C:\Users\Admin\Documents\Tesis\Scripts\recorte_suave.sci');
6  exec('C:\Users\Admin\Documents\Tesis\Scripts\recorte_medio.sci');
7  exec('C:\Users\Admin\Documents\Tesis\Scripts\get_gain_voz.sci');
8
9  // señales y ruidos .wav
10 my_voz_wav = senal;
11 my_blanco_wav =
   ↪  "C:\Users\Admin\Documents\Tesis\Audio_signals\50_frases_ingles\blanco.wav";
12 my_rosa_wav =
   ↪  "C:\Users\Admin\Documents\Tesis\Audio_signals\50_frases_ingles\rosa.wav";
13 my_babble_wav =
   ↪  "C:\Users\Admin\Documents\Tesis\Audio_signals\50_frases_ingles\babble.wav";
14
15 // valores soportados: [1 2 3 4]
16 decomp_levels = [1 2 3 4];
17 //valores soportados: 'universal', 'escala'
18 my_limit = 'universal';
19
20 // informacion en consola
21 printf('[INFO] Señal de voz seleccionada: %s\n', senal);
```

```
22 printf('[INFO] Wavelet seleccionada: %s\n', wavelet);
23 printf('[INFO] Procesando con ruido: %s\n', ruido);
24 printf('[INFO] Procesando con umbral de recorte: %s\n',recorte);
25
26 // Voz
27 [s_voz,Fs,bits] = wavread(my_voz_wav);
28
29 // numero de muestras depende de la señal de voz
30 n = length(s_voz)
```

Posteriormente se toman decisiones de acuerdo con el tipo de ruido que se desea suprimir. En este punto, como se observa en las líneas de código siguientes, se suman la señal de voz y la señal del ruido. No obstante, antes de la mezcla se determina la ganancia que se debe aplicar a la señal de voz para cumplir con el requisito de SNR inicial especificado por el usuario. Para tal propósito se hace uso de la función `get_gain_voz()`. Esta función, al no ser parte teórica del algoritmo, se describe en el anexo B.

```
1  //seleccion del ruido a eliminar
2  // s_n es la senal a procesar
3  if ruido == 'blanco' then
4      [s_blanco,Fs,bits] = wavread(my_blanco_wav,n);
5      gain_voz = get_gain_voz(s_voz, s_blanco, snr_in);
6      s_voz = gain_voz .* s_voz;
7      voz_rms = sqrt(mean(s_voz.^2));
8      blanco_rms = sqrt(mean(s_blanco.^2));
9      SNR_med = 20*log10(voz_rms/blanco_rms)
10     s_n = s_blanco + s_voz;
11
12 elseif ruido == 'rosa' then
13     [s_rosa,Fs,bits] = wavread(my_rosa_wav,n);
14     gain_voz = get_gain_voz(s_voz, s_rosa, snr_in);
15     s_voz = gain_voz .* s_voz;
16     voz_rms = sqrt(mean(s_voz.^2));
17     rosa_rms = sqrt(mean(s_rosa.^2));
18     SNR_med = 20*log10(voz_rms/rosa_rms);
19     s_n = s_rosa + s_voz;
20
21 elseif ruido == 'babble' then
22     [s_babble,Fs,bits] = wavread(my_babble_wav,n);
23     gain_voz = get_gain_voz(s_voz, s_babble, snr_in);
```

```
    s_voz = gain_voz .* s_voz;
    voz_rms = sqrt(mean(s_voz.^2));
    babble_rms = sqrt(mean(s_babble.^2));
    SNR_med = 20*log10(voz_rms/babble_rms);
    s_n = s_babble + s_voz;

elseif ruido == 'ninguno' then
    s_n = s_voz;
else
    disp("[ERROR] Indique el tipo de ruido")
end
```

La función `wavedec` se encarga de realizar la transformada *wavelet* de la señal deseada `s_n` de acuerdo con los niveles de descomposición seleccionados y la función *wavelet* determinada. Dicha función genera dos vectores, el vector `C` que almacena los coeficientes de aproximación y de detalles y el vector `L` sobre la longitud asociada a estos coeficientes. Mediante estos vectores y las funciones `appcoef()` y `detcoef()` se pueden obtener los coeficientes de aproximación `cA` y detalle `cD` a los niveles de descomposición seleccionados, tal y como sucede en el extracto de código siguiente.

```
// descomposicion en niveles
[C,L] = wavedec(s_n,length(decomp_levels),wavelet);

// coeficientes de aproximacion
cA_1 = appcoef(C,L,wavelet,1);
cA_2 = appcoef(C,L,wavelet,2);
cA_3 = appcoef(C,L,wavelet,3);
cA_4 = appcoef(C,L,wavelet,4);

// coeficientes de detalle
cD_1 = detcoef(C,L,1);
cD_2 = detcoef(C,L,2);
cD_3 = detcoef(C,L,3);
cD_4 = detcoef(C,L,4);

end
```

Continuando con el análisis del procesamiento, se muestra el siguiente extracto, en el que se hace la selección entre los diversos modos de recorte de coeficientes. Es importante destacar que el recorte se aplica para cada nivel

de descomposición de los coeficientes de detalle y el límite de recorte establecido por la función `limit_escala()` o bien `limit_universal()`. Además, se define el factor de realce de sonido `c_amp`, factor que amplifica los coeficientes correspondientes a la señal y que no serán recortados por el método.

```
if my_limit == 'escala' then
    T_j1 = limit_escala(length(s_n), cD_1)
    T_j2 = limit_escala(length(s_n), cD_2)
    T_j3 = limit_escala(length(s_n), cD_3)
    T_j4 = limit_escala(length(s_n), cD_4)
else
    T_u = limit_universal(length(s_n), cD_1);
    T_j1 = T_u;
    T_j2 = T_u;
    T_j3 = T_u;
    T_j4 = T_u;
end

//realce de sonido en cA
c_amp = 1.15;

// aplicar recorte fuerte de coeficientes
if recorte == 'fuerte' then
    cD_1 = recorte_fuerte(cD_1,T_j1,c_amp);
    cD_2 = recorte_fuerte(cD_2,T_j2,c_amp);
    cD_3 = recorte_fuerte(cD_3,T_j3,c_amp);
    cD_4 = recorte_fuerte(cD_4,T_j4,c_amp);

elseif recorte == 'medio' then
    cD_1 = recorte_medio(cD_1,T_j1,c_amp);
    cD_2 = recorte_medio(cD_2,T_j2,c_amp);
    cD_3 = recorte_medio(cD_3,T_j3,c_amp);
    cD_4 = recorte_medio(cD_4,T_j4,c_amp);

elseif recorte == 'suave' then
    cD_1 = recorte_suave(cD_1,T_j1,c_amp);
    cD_2 = recorte_suave(cD_2,T_j2,c_amp);
    cD_3 = recorte_suave(cD_3,T_j3,c_amp);
    cD_4 = recorte_suave(cD_4,T_j4,c_amp);

elseif recorte == 'ninguno' then
    disp("[INFO] Sin recorte de coeficientes")
```

```
38 else
39     disp("[ERROR] Indique el tipo de recorte")
40 end
```

A continuación se muestra en detalle la función `limit_universal()` encargada de calcular el límite del umbral de recorte de coeficientes para las escalas de descomposición. En las líneas siguientes se muestra su cálculo, que como se puede observar, consiste en la aplicación de la ecuaciones (2.30) y (2.31).

```
1 function [T_u]=limit_universal(N, coef)
2
3     // nivel de ruido universal
4     sigma_u = mad(coef) / 0.6745
5
6     // limite universal
7     T_u = sigma_u * sqrt(2 * log10(N))
8
9 endfunction
```

De la misma forma seguidamente se muestra en detalle las funciones de recorte de coeficientes fuerte, suave y medio. La función de recorte de coeficientes `recorte_fuerte()`, descrita por las siguientes líneas de código, toma una serie de coeficientes y los convierte a cero si su valor es menor que el umbral `T`, tal y como se describió en la ecuación (2.26).

```
1 function [cD]=recorte_fuerte(coef, T)
2
3     // contador para recorrer el array
4     for i = 1:length(coef)
5         if abs(coef(i)) >= T then
6             coef(i) = c_amp * coef(i);
7         else
8             coef(i) = 0;
9         end
10    end
11
12    cD = coef;
13
14 endfunction
```

La función de recorte de coeficientes `recorte_suave()`, implementada por la siguiente lógica, toma un vector de coeficientes y aplica las relaciones expresadas en la ecuación (2.27). Tal y como se puede observar, los valores de los coeficientes después de aplicado el recorte pueden ser cero, su valor original más o menos el umbral o bien permanece sin alteración.

```
function [cD]=recorte_suave(coef, T)

    // contador para recorrer el array
    for i = 1:length(coef)
        if coef(i) >= T then
            coef(i) = coef(i) - T;
        elseif abs(coef(i)) < T then
            coef(i) = 0,
        elseif coef(i) <= -T then
            coef(i) = coef(i) + T;
        else
            coef(i) = c_amp * coef(i);
        end
    end

    cD = coef;

endfunction
```

La función de recorte de coeficientes `recorte_medio()`, descrita por las ecuaciones (2.28) y (2.29), se muestra implementada por las isguientes líneas.

```
function [cD]=recorte_medio(coef, T)

    // contador para recorrer el array
    dosT = 2*T;
    for i = 1:length(coef)
        if abs(coef(i)) < dosT then
            if abs(coef(i)) > T then
                coef(i) = 2 * (abs(coef(i)) - T)
            else
                coef(i) = 0
            end
        else
            coef(i) = c_amp * abs(coef(i));
```

```
        end
    end

    cD = coef;

endfunction
```

Como se mostró en las funciones anteriores, el parámetro `c_amp` se multiplica por los coeficientes sin ruido, el valor determinado de forma práctica para esta investigación fue 1,15. Para su determinación se consideró el rango establecido por (Delgado, 2010), se probaron valores dentro del rango y se determinó que 1,15 daba buenos resultados.

Para regresar al dominio del tiempo se reconstruye el vector de coeficientes `C`, que contiene los coeficientes de aproximación `cA` de la última escala y los coeficientes de detalles `cD` por cada escala. Es importante destacar que solamente a los coeficientes de detalle se les aplica el recorte en el dominio *wavelet*. La reconstrucción de la señal es posible gracias a la función `waverec()` que aplica la transformada *wavelet* inversa.

```
// unir vector coeficientes
C_deno = [cA_4 cD_4 cD_3 cD_2 cD_1];

// reconstruccion de la señal
ss = waverec(C_deno,L,wavelet);
```

Finalmente, al aplicar la función desarrollada se obtiene el resultado en consola siguiente. Donde se imprime la señal de voz actual, la función *wavelet* seleccionada, el tipo de ruido a suprimir, el tipo de recorte aplicado, el SNR tras agregar el ruido, el SNR después de aplicar el procesamiento y por último el MSE.

```
--> my_voz =
↪  "C:\Users\Admin\Documents\Tesis\Audio_signals\50_frases_ingles\arctic_a0002.wav";
--> [ss, s_n, Fs, bits] = wavelet_denoising(my_voz, "babble", "db5", "suave",
↪  0, "on", "off");

[INFO] Señal de voz seleccionada:
↪  C:\Users\Admin\Documents\Tesis\Audio_signals\50_frases_ingles\arctic_a0002.wav
[INFO] Wavelet seleccionada: db5
[INFO] Procesando con ruido: babble
```

```
[INFO] Procesando con umbral de recorte: suave
[INFO] Inicialmente SNR (dB) = -0.102567
[INFO] SNR de señal procesada (dB) = 3.282964
[INFO] MSE = 0.00775839
```

En el anexo A, se muestra en detalle la función `wavelet_denoising()` que ha sido explicada en detalle en esta sección. De la misma forma, se muestra el *script* desarrollado para realizar las mediciones de forma cíclica fácilmente.

5.2. Descripción de la medición de parámetros

5.2.1 Error Cuadrático Medio (MSE)

Se implementó el error cuadrático medio como índice medidor de calidad ya que compara las diferencias entre la señal inicial y la señal procesada. El cálculo del MSE se describe en las siguientes líneas, donde la señal de voz original se representa con `s_voz` y la señal de voz procesada con `ss`. Como se muestra, se utiliza la fórmula (2.37) para el cálculo práctico del MSE. El parámetro `n` corresponde al número de muestras de las señales.

```
// error señal original vs procesada
subs_error = s_voz - ss;
mse = 1/n .* sum(subs_error.^2);
```

5.2.2 Relación señal a ruido (SNR)

Para la medición de la relación señal a ruido se hizo uso de una aproximación. Se cuenta con la señal original `s_voz` y la señal procesada `ss`. Por lo tanto, para obtener el ruido presente en la señal reconstruida se le resta la señal original. Se obtiene los valores rms de la señal procesada y el ruido, por lo que se puede calcular el valor del SNR según como se indica en las ecuación (2.39), con la única diferencia de que se multiplica por veinte ya que se está trabajando con los valores rms de las señales y no con potencias.

```
// SNR = |ss| / |ss - s_voz|
ss_rms = sqrt(mean(ss.^2));
n_ss = ss - s_voz;
n_rms = sqrt(mean(n_ss.^2));
SNR_ss = 20*log10(ss_rms/n_rms);
```

5.2.3 Relación porcentual de la calidad de habla (PESQ)

Para la medición del parámetro PESQ se utiliza la librería en Python desarrollada por Miao (2019). Los detalles se muestran en el anexo C. Se decidió aprovechar una librería externa dado que el PESQ es una medición compleja de desarrollar, por lo que su eventual desarrollo estaría fuera del alcance de la investigación. La sintaxis de la función en Python se muestra en la siguientes líneas de código.

```
# lecturas .wav
rate, proc = wavfile.read(my_voz_procesada)
rate, deg = wavfile.read(my_voz_sucia)
rate, clean = wavfile.read(senal_limpia)

# calculo PESQ sucia vs limpia
pesq_limpia_sucia = pesq(rate, clean, deg, 'wb')

# calclulo PESQ procesada vs limpia
pesq_procesada_sucia = pesq(rate, clean, proc, 'wb')

delta_pesq = pesq_limpia_sucia - pesq_procesada_sucia
```

5.3. Mediciones experimentales para ruido *babble*

En la tabla 5.1 se muestran las mediciones del parámetro SNR, en la tabla 5.2 las mediciones del MSE y en la tabla 5.3 las mediciones del PESQ para el caso con ruido *babble*. Se destaca que en el caso de la medición del PESQ se realiza una resta entre el valor calculado considerando la señal en limpio y la señal ruidosa y el valor calculado considerando la señal procesada y la señal ruidosa. Esto se debe a que el PESQ es una medición muy sensible por lo que al calcular esta diferencia se puede observar mejor el efecto del método de eliminación de ruido sobre la calidad del audio procesado. Las mediciones realizadas se efectuaron con un SNR inicial de aproximadamente 0 dB.

Es importante destacar que las mediciones de las tablas 5.1, 5.2 y 5.3 consideran las cinco funciones *wavelet* de estudio con sus tres casos de recorte de coeficientes. Se utiliza una base de cincuenta señales de audio disponibles en (Festvox, 2003), por lo que en las tablas se muestra resultados estadísticos de valor promedio, desviación estándar, mínimo y máximo. De esta forma se puede tener una mejor idea del comportamiento de cada método seleccionado en un grupo de señales.

Con respecto al SNR, como se muestra en la tabla 5.1, en promedio el recorte suave de coeficientes brinda los resultados más efectivos, mientras que el recorte medio brinda lo menos deseables. Al mismo tiempo, es apreciable como las cinco funciones *wavelet* presentan resultados bastante similares según cada tipo de recorte. Dicho de otro modo, para el grupo de señales procesadas, las distribuciones de SNR presentan similitudes entre sí. Estas observaciones son de igual forma válidas para el parámetro MSE, tal y como se puede corroborar con los datos de la tabla 5.2.

Para el parámetro PESQ, mostrado en la tabla 5.3, se mantiene como mejor caso el recorte suave de coeficientes. Inclusive, se puede observar como la distribución de datos de este tipo de recorte no muestra valores negativos. El recorte medio presenta los resultados menos efectivos y el recorte fuerte en algunos casos presenta deltas negativos. Las funciones *wavelet* entre sí siguen presentando resultados similares para este tipo de ruido. Vale la pena destacar que las mediciones de PESQ son muy sensibles para casos en que se agrega ruido a la señal (UIT-T, 2001), por esta razón los valores numéricos son muy bajos.

Cuadro 5.1: Medición del SNR después de aplicar el algoritmo con ruido *babble* sobre cincuenta señales de prueba

Wavelet	**Recorte**	**Promedio SNR**	**Desv Est SNR**	**Min SNR**	**Max SNR**
bior3.3	fuerte	2,846	0,045	2,727	2,950
	medio	1,433	0,289	0,746	1,870
	suave	3,269	0,058	3,090	3,393
coif5	fuerte	2,837	0,047	2,736	2,957
	medio	1,219	0,334	0,604	1,893
	suave	3,379	0,060	3,217	3,514
db5	fuerte	2,848	0,045	2,740	2,965
	medio	1,176	0,300	0,457	1,817
	suave	3,421	0,070	3,226	3,549
dmey	fuerte	2,827	0,047	2,734	2,949
	medio	1,284	0,331	0,535	1,986
	suave	3,352	0,056	3,213	3,496
sym8	fuerte	2,839	0,046	2,735	2,948
	medio	1,208	0,321	0,510	1,857
	suave	3,392	0,063	3,219	3,522

Cuadro 5.2: Medición del MSE después de aplicar el algoritmo con ruido *babble* sobre cincuenta señales de prueba

Wavelet	**Recorte**	**Promedio MSE**	**Desv Est MSE**	**Min MSE**	**Max MSE**
bior3.3	fuerte	0,011	0,0001	0,011	0,012
	medio	0,014	0,0011	0,012	0,017
	suave	0,008	0,0000	0,008	0,008
coif5	fuerte	0,011	0,0001	0,011	0,012
	medio	0,016	0,0014	0,014	0,019
	suave	0,008	0,0001	0,007	0,008
db5	fuerte	0,011	0,0001	0,011	0,011
	medio	0,016	0,0013	0,014	0,020
	suave	0,007	0,0001	0,007	0,007
dmey	fuerte	0,012	0,0001	0,011	0,012
	medio	0,016	0,0014	0,014	0,020
	suave	0,008	0,0001	0,007	0,008
sym8	fuerte	0,011	0,0001	0,011	0,012
	medio	0,016	0,0014	0,014	0,020
	suave	0,007	0,0001	0,007	0,008

Cuadro 5.3: Medición del PESQ después de aplicar el algoritmo con ruido *babble* sobre cincuenta señales de prueba

Wavelet	**Recorte**	**Promedio PESQ**	**Desv Est PESQ**	**Min PESQ**	**Max PESQ**
bior3.3	fuerte	0,016	0,011	-0,049	0,036
	medio	-0,085	0,030	-0,181	-0,009
	suave	0,042	0,012	0,016	0,082
coif5	fuerte	0,012	0,007	-0,005	0,036
	medio	-0,085	0,032	-0,170	-0,004
	suave	0,052	0,022	0,021	0,178
db5	fuerte	0,008	0,007	-0,011	0,025
	medio	-0,099	0,033	-0,186	-0,026
	suave	0,050	0,044	0,017	0,346
dmey	fuerte	0,015	0,008	-0,014	0,040
	medio	-0,074	0,030	-0,163	-0,012
	suave	0,054	0,020	0,026	0,161
sym8	fuerte	0,011	0,007	-0,011	0,027
	medio	-0,088	0,032	-0,179	-0,022
	suave	0,051	0,022	0,020	0,176

5.4. Mediciones experimentales para ruido blanco

En la tabla 5.4 se muestran las mediciones del parámetro SNR, en la tabla 5.5 las mediciones del MSE y en la tabla 5.6 las mediciones del PESQ para el caso

ruido blanco. De igual forma se muestran resultados estadísticos considerando que el método se aplicó sobre el mismo grupo de cincuenta señales de voz. Las mediciones realizadas se efectuaron con un SNR inicial de aproximadamente 0 dB.

De la tabla 5.4 se puede observar como para este ruido de estudio existe mayor variación en resultados de acuerdo con la función *wavelet*. La función bior3.3 presenta resultados apreciablemente distintos con respecto al grupo de funciones de estudio. Dejando de lado este aspecto, el recorte fuerte brinda los resultados más efectivos en promedio para el grupo de funciones *wavelet*, evidentemente exceptuando la función bior3.3. Además, el recorte medio continúa siendo el caso menos deseable. Para este ruido, observando su desviación estándar, su mínimo y su máximo se puede corroborar como su distribución es bastante amplia comparada con los demás tipo de recorte. Estas observaciones, nuevamente son válidas para el parámetro MSE, como se observa en la tabla 5.5.

Con respecto al PESQ, tabla 5.6, contrariamente la función bior3.3 con recorte suave es la que brinda mejores resultados promedio. Esto es una prueba del balance entre la efectividad del método de supresión de ruido y la calidad de las señales de audio. Las demás funciones, según el tipo de recorte, brindan resultados bastante similares entre sí.

Cuadro 5.4: Medición del SNR después de aplicar el algoritmo con ruido blanco sobre cincuenta señales de prueba

Wavelet	**Recorte**	**Promedio SNR**	**Desv Est SNR**	**Min SNR**	**Max SNR**
bior3.3	fuerte	4,516	0,146	3,908	4,766
	medio	2,826	0,690	1,252	4,120
	suave	6,229	0,477	5,198	7,064
coif5	fuerte	7,638	0,499	6,352	8,686
	medio	3,538	1,220	1,285	5,832
	suave	6,152	0,697	4,865	7,564
db5	fuerte	7,453	0,496	5,984	8,438
	medio	3,374	1,077	0,944	5,569
	suave	5,857	0,674	4,458	7,267
dmey	fuerte	7,770	0,494	6,490	8,699
	medio	3,775	1,283	1,185	6,693
	suave	6,380	0,746	4,933	7,860
sym8	fuerte	7,588	0,501	6,213	8,487
	medio	3,378	1,196	1,052	5,746
	suave	6,087	0,685	4,774	7,369

Cuadro 5.5: Medición del MSE después de aplicar el algoritmo con ruido blanco sobre cincuenta señales de prueba

Wavelet	**Recorte**	**Promedio MSE**	**Desv Est MSE**	**Min MSE**	**Max MSE**
bior3.3	fuerte	0,0052	0,0001	0,0049	0,0059
	medio	0,0063	0,0011	0,0045	0,0092
	suave	0,0021	0,0001	0,0018	0,0026
coif5	fuerte	0,0017	0,0001	0,0014	0,0022
	medio	0,0043	0,0012	0,0024	0,0070
	suave	0,0020	0,0002	0,0016	0,0025
db5	fuerte	0,0018	0,0001	0,0014	0,0023
	medio	0,0043	0,0011	0,0026	0,0074
	suave	0,0021	0,0002	0,0017	0,0026
dmey	fuerte	0,0017	0,0001	0,0014	0,0021
	medio	0,0042	0,0012	0,0021	0,0073
	suave	0,0020	0,0002	0,0015	0,0024
sym8	fuerte	0,0017	0,0001	0,0014	0,0022
	medio	0,0044	0,0012	0,0025	0,0074
	suave	0,0020	0,0002	0,0016	0,0025

Cuadro 5.6: Medición del PESQ después de aplicar el algoritmo con ruido blanco sobre cincuenta señales de prueba

Wavelet	**Recorte**	**Promedio PESQ**	**Desv Est PESQ**	**Min PESQ**	**Max PESQ**
bior3.3	fuerte	0,069	0,020	0,039	0,150
	medio	0,109	0,031	0,065	0,232
	suave	0,255	0,066	0,071	0,429
coif5	fuerte	0,043	0,026	-0,032	0,124
	medio	0,048	0,038	-0,066	0,149
	suave	0,210	0,099	0,033	0,499
db5	fuerte	0,084	0,032	0,024	0,179
	medio	0,054	0,040	-0,072	0,197
	suave	0,238	0,098	0,068	0,523
dmey	fuerte	0,035	0,022	-0,029	0,089
	medio	0,056	0,040	-0,047	0,162
	suave	0,221	0,102	0,046	0,566
sym8	fuerte	0,052	0,025	-0,010	0,133
	medio	0,048	0,034	-0,048	0,142
	suave	0,216	0,096	0,050	0,485

5.5. Mediciones experimentales para ruido rosa

En la tabla 5.7 se muestran las mediciones del parámetro SNR, en la tabla 5.8 las mediciones del MSE y en la tabla 5.9 las mediciones del PESQ para el

caso ruido rosa. De igual forma se muestran resultados estadísticos considerando que el método se aplicó sobre el grupo de cincuenta señales de voz. Las mediciones realizadas se efectuaron con un SNR inicial de aproximadamente 0 dB.

Para este tipo de ruido, el parámetro SNR mostrado en la tabla 5.7, presenta resultados similares tanto para el grupo de funciones *wavelet* de estudio así como para los recortes fuerte y suave. El recorte medio sigue siendo el que muestra los resultados menos efectivos y con distribuciones más amplias. La tabla 5.8 muestra que para el MSE las mismas observaciones son válidas al menos para este tipo de ruido.

No obstante, según se observa en la tabla 5.9, el tipo de recorte suave presenta la distribución más amplia, inclusive más que el recorte medio, a diferencia de los casos anteriores. A pesar de esta peculiaridad, en promedio el recorte suave presenta los mejores resultados de PESQ.

Cuadro 5.7: Medición del SNR después de aplicar el algoritmo con ruido rosa sobre cincuenta señales de prueba

Wavelet	**Recorte**	**Promedio SNR**	**Desv Est SNR**	**Min SNR**	**Max SNR**
bior3.3	fuerte	3,193	0,069	2,990	3,328
	medio	1,536	0,320	0,765	2,112
	suave	3,148	0,108	2,837	3,374
coif5	fuerte	3,444	0,075	3,231	3,591
	medio	1,603	0,457	0,674	2,483
	suave	3,083	0,140	2,764	3,337
db5	fuerte	3,441	0,077	3,216	3,571
	medio	1,528	0,404	0,519	2,394
	suave	3,032	0,146	2,673	3,301
dmey	fuerte	3,461	0,070	3,267	3,600
	medio	1,686	0,468	0,588	2,635
	suave	3,149	0,141	2,843	3,389
sym8	fuerte	3,446	0,077	3,229	3,591
	medio	1,567	0,448	0,575	2,464
	suave	3,066	0,142	2,747	3,327

Cuadro 5.8: Medición del MSE después de aplicar el algoritmo con ruido rosa sobre cincuenta señales de prueba

Wavelet	**Recorte**	**Promedio MSE**	**Desv Est MSE**	**Min MSE**	**Max MSE**
bior3.3	fuerte	0,009	0,0001	0,009	0,010
	medio	0,013	0,0011	0,011	0,016
	suave	0,007	0,0001	0,007	0,008
coif5	fuerte	0,008	0,0001	0,008	0,008
	medio	0,012	0,0014	0,009	0,015
	suave	0,007	0,0002	0,007	0,008
db5	fuerte	0,008	0,0001	0,008	0,008
	medio	0,012	0,0013	0,009	0,015
	suave	0,007	0,0002	0,007	0,008
dmey	fuerte	0,008	0,0001	0,008	0,008
	medio	0,011	0,0014	0,009	0,015
	suave	0,007	0,0002	0,007	0,008
sym8	fuerte	0,008	0,0001	0,008	0,008
	medio	0,012	0,0014	0,009	0,015
	suave	0,007	0,0002	0,007	0,008

Cuadro 5.9: Medición del PESQ después de aplicar el algoritmo con ruido rosa sobre cincuenta señales de prueba

Wavelet	**Recorte**	**Promedio PESQ**	**Desv Est PESQ**	**Min PESQ**	**Max PESQ**
bior3.3	fuerte	0,066	0,021	0,037	0,142
	medio	0,044	0,024	-0,032	0,127
	suave	0,180	0,057	0,060	0,334
coif5	fuerte	0,116	0,032	0,059	0,194
	medio	0,051	0,028	-0,049	0,118
	suave	0,232	0,081	0,051	0,420
db5	fuerte	0,125	0,032	0,069	0,238
	medio	0,037	0,022	-0,054	0,082
	suave	0,217	0,078	0,043	0,394
dmey	fuerte	0,116	0,038	0,045	0,265
	medio	0,071	0,034	-0,035	0,163
	suave	0,250	0,089	0,058	0,471
sym8	fuerte	0,123	0,033	0,070	0,192
	medio	0,050	0,027	-0,036	0,132
	suave	0,228	0,082	0,050	0,439

6 Selección de la función *wavelet* con mejor desempeño

6.1. Función *wavelet* óptima para cada ruido de estudio

En el capítulo anterior se mostraron los resultados de SNR, MSE y PESQ en detalle del método de eliminación de ruido de esta investigación. En esta sección se analizan los resultados obtenidos para determinar cual función *wavelet* y que tipo de recorte de coeficientes produce el mejor resultado. Para tal propósito se resume cada ruido de estudio en la tabla 6.1, donde se muestra el mejor caso según el parámetro medido. Es importante destacar el enfoque de cada parámetro, pues como se observa en la tabla 6.1, el mejor caso de estudio varía para cada ruido y dentro de cada ruido varía según el parámetro medido. En términos generales el SNR se utiliza para medir la reducción de ruido, el MSE para medir la diferencia entre la señal original y la señal procesada y finalmente el PESQ permite describir la calidad auditiva de la señal de audio procesada.

Al observar la tabla 6.1 se deduce que para el ruido *babble* el mejor caso de función *wavelet* y recorte de coeficientes es db5 o dmey con recorte suave. Para el ruido blanco, dmey con recorte fuerte o bior3.3 con recorte suave. Finalmente, para el ruido rosa serían dmey con recorte fuerte, coif5 con recorte suave o bien dmey con recorte suave.

Cuadro 6.1: Mejor caso de estudio para cada ruido según los parámetros medidos

Ruido	**Parámetro**	***wavelet***	**Recorte**
bable	SNR	db5	suave
bable	MSE	db5	suave
bable	PESQ	dmey	suave
blanco	SNR	dmey	fuerte
blanco	MSE	dmey	fuerte
blanco	PESQ	bior3.3	suave
rosa	SNR	dmey	fuerte
rosa	MSE	coif5	suave
rosa	PESQ	dmey	suave

Ahora bien, antes de definir el mejor caso por ruido de estudio, se puede hacer una comparación de los valores medidos para determinar la diferencia entre escoger un caso u otro. Dicho esto, si se observa la tabla 6.2 para el ruido *babble*, tanto la función db5 como dmey, con recorte suave, producen resultados muy similares de SNR, MSE y PESQ. La diferencia principal radica en el parámetro PESQ donde en promedio los valores medidos con ambas funciones son similares, dando mejores resultados para dmey, mientras que para db5 la distribución se muestra más amplia, tal y como se observa en su desviación estándar, su valor mínimo y su valor máximo.

Cuadro 6.2: Comparación de parámetros medidos para los mejores casos del método de eliminación de ruido considerando ruido *babble*

Parámetro	**Dato**	**db5-suave**	**dmey-suave**
SNR	Promedio	3,4209	3,3522
	DesvEst	0,0700	0,0567
	Min	3,2261	3,2135
	Max	3,5494	3,4963
MSE	Promedio	0,0077	0,0083
	DesvEst	0,0001	0,0001
	Min	0,0074	0,0079
	Max	0,0079	0,0086
PESQ	Promedio	0,0504	0,0543
	DesvEst	0,0448	0,0207
	Min	0,0179	0,0260
	Max	0,3469	0,1615

En la tabla 6.3, se muestra la comparación de los valores medidos para el ruido blanco con dmey recorte fuerte y bior3.3 recorte suave. En este caso si es evidente la diferencia en el valor del SNR y el MSE entre ambos métodos, donde se aprecia como la función dmey brinda mejores resultados en estos parámetros. Sin embargo, contrariamente para el parámetro PESQ la función bior3.3 es la que brinda mejores resultados en promedio. Esto es un ejemplo del balance entre la eficacia del método para reducir el ruido contra la calidad de la señal de audio procesada.

Cuadro 6.3: Comparación de parámetros medidos para los mejores casos del método de eliminación de ruido considerando ruido blanco

Parámetro	**Dato**	**bior3.3-suave**	**dmey-fuerte**
SNR	Promedio	6,2292	7,7699
	DesvEst	0,4778	0,4948
	Min	5,1985	6,4900
	Max	7,0641	8,6999
MSE	Promedio	0,0021	0,0017
	DesvEst	0,0001	0,0001
	Min	0,0018	0,0014
	Max	0,0026	0,0021
PESQ	Promedio	0,2550	0,0355
	DesvEst	0,0666	0,0229
	Min	0,0711	-0,0296
	Max	0,4290	0,0895

Para el caso de ruido rosa, al comparar los casos, coif5 con recorte suave, dmey con recorte fuerte y dmey con recorte suave, resultados de la tabla 6.4, se observa un comportamiento particular. En este caso la función dmey con recorte fuerte brinda mejores resultados de SNR, no obstante la función coif5 brinda mejores resultados de MSE y nuevamente la función dmey brinda mejores resultados de PESQ pero al aplicar recorte suave. Nuevamente se puede apreciar el balance entre SNR y PESQ, reducción de ruido contra calidad del audio. No obstante, las diferencias entre estos tres casos no son tan significativas como en el caso de ruido blanco.

Cuadro 6.4: Comparación de parámetros medidos para los mejores casos del método de eliminación de ruido considerando ruido rosa

Parámetro	**Dato**	**coif5-suave**	**dmey-fuerte**	**dmey-suave**
SNR	Promedio	3,0829	3,4619	3,1494
	DesvEst	0,1399	0,0707	0,1418
	Min	2,7648	3,2675	2,8432
	Max	3,3374	3,6000	3,3893
MSE	Promedio	0,0076	0,0084	0,0076
	DesvEst	0,0002	0,0001	0,0002
	Min	0,0073	0,0081	0,0073
	Max	0,0081	0,0087	0,0081
PESQ	Promedio	0,2324	0,1167	0,2504
	DesvEst	0,0817	0,0384	0,0892
	Min	0,0517	0,0452	0,0580
	Max	0,4200	0,2652	0,4710

Finalmente, tras esta revisión de resultados de los mejores casos por pa-

rámetros para los tres ruidos de estudio, se puede determinar le mejor combinación *wavelet* y tipo de recorte de coeficientes. Dado que el método busca suprimir el ruido, se define que el mejor parámetro para medir su efectividad es el SNR, por esta razón se va considerar el mejor caso de acuerdo con este parámetro. Otro motivo importante a considerar es que las mediciones de PESQ pueden resultar confusas dado que se trabaja con señales que tienen ruido añadido. Además, las diferencias entre las mediciones de PESQ no son inmensas por lo que se podría asumir que la calidad de la señal de audio no se ve impactada significativamente entre seleccionar una función *wavelet* , un tipo de recorte u otro de los casos listados en las tablas 6.2, 6.3 y 6.4. Para el caso de MSE, generalmente su tendencia es muy similar al SNR.

El análisis previo permite tomar con mayor conocimiento la decisión de seleccionar el mejor caso de acuerdo con el SNR. Sin embargo, como se ha destacado a lo largo de la investigación es importante no perder de vista parámetros como el MSE y el PESQ, que brindan otro tipo de información, como bien lo es el error entre las señales original y procesada así como la calidad del audio final. En la tabla 6.5 se muestra la selección del mejor caso de función *wavelet* y tipo de recorte de coeficientes para cada ruido de estudio.

Cuadro 6.5: *Wavelet* y tipo de recorte óptimo para cada ruido de estudio

Ruido	***Wavelet***	**Recorte**
babble	db5	suave
blanco	dmey	fuerte
rosa	dmey	fuerte

En la figura 6.1 se muestra la curva en el tiempo de la señal ruidosa y la señal procesada para el caso ruido blanco y la función *wavelet* dmey con recorte fuerte. Es apreciable como el algoritmo suprime las componentes ruidosa. La figura 6.2 muestra el mismo caso en el dominio de la frecuencia, nuevamente se observa la reducción del ruido sobre la señal procesada. Se añaden las curvas del ruido blanco ya que la apreciación visual del ruido y de su eliminación es más evidente que en los casos ruido rosa y ruido *babble*. Similarmente, la figura 6.3 muestra un espectrograma para una señal de muestra considerando el caso ruido blanco con el software Audacity. Se muestra la señal original, la señal tras agregar ruido y finalmente la señal después de ser procesada por el algoritmo.

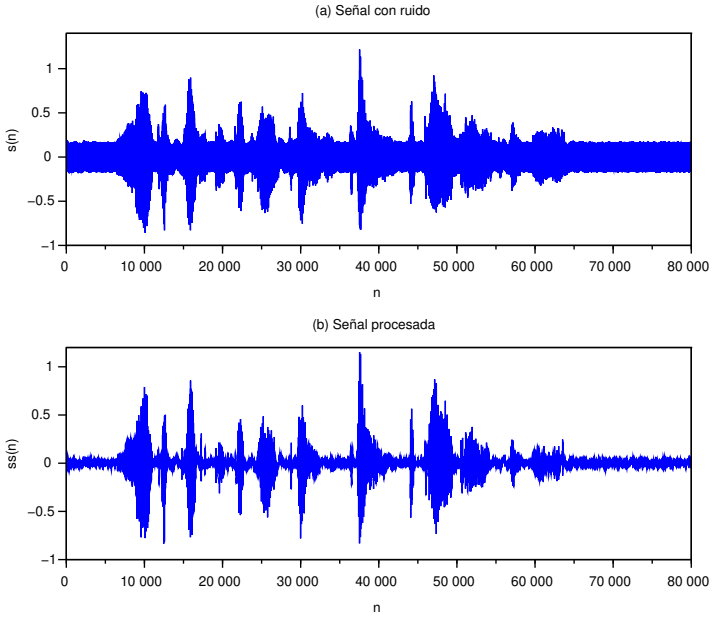

Figura 6.1: Forma de onda en el tiempo para ruido blanco. (a) Señal de voz ruidosa. (b) Señal de voz procesada

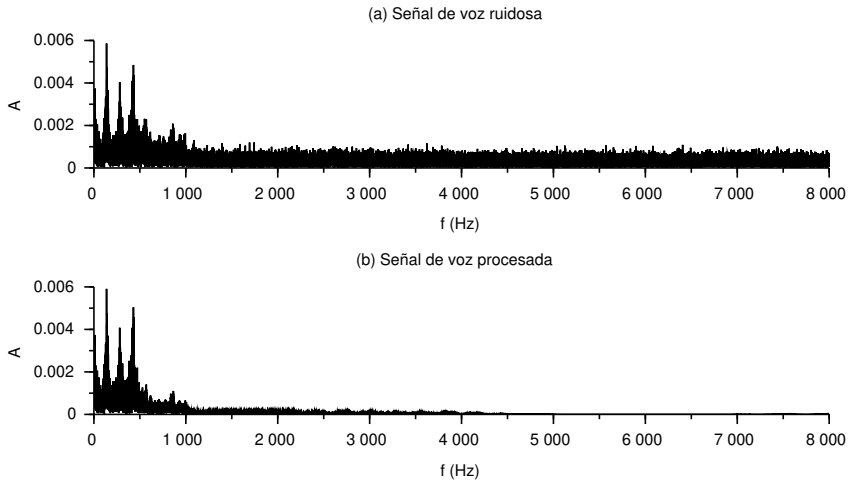

Figura 6.2: Espectro de frecuencia para ruido blanco. (a) Señal de voz ruidosa. (b) Señal de voz procesada

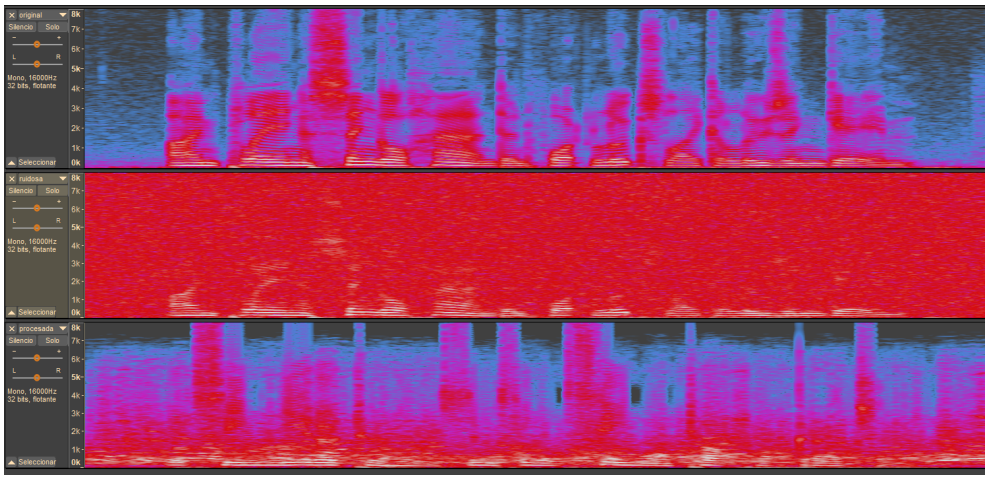

Figura 6.3: Espectrograma de la señal original, ruidosa y procesada para el caso ruido blanco considerando una señal de muestra mediante el software Audacity

6.2. Eficacia del método de acuerdo con la función *wavelet*

Otro detalle interesante al analizar los datos recolectados es la variación de los parámetros medidos, considerando el mismo grupo de señales, el mismo tipo de recorte de coeficientes y únicamente variando la función *wavelet*. Para simplificar solo se va a realizar el análisis considerando el SNR. En la tabla 6.6 tenemos el caso de ruido *babble* y recorte suave, como se puede observar la variación de resultados es muy baja. Las cinco funciones *wavelets* de estudio muestran en promedio resultados muy similares de la relación señal a ruido.

Cuadro 6.6: Comparación del SNR de la señal procesada producido por cada función *wavelet* para ruido *babble* y recorte suave

Wavelet	**Promedio SNR**	**Desv Est SNR**	**Min SNR**	**Max SNR**
bior3.3	3,269	0,058	3,090	3,393
coif5	3,379	0,060	3,217	3,514
db5	3,420	0,070	3,226	3,549
dmey	3,352	0,056	3,213	3,496
sym8	3,392	0,063	3,219	3,522

Sin embargo, si se considera otro caso de estudio, específicamente el ruido blanco y recorte fuerte, cuyos resultados se muestran en la tabla 6.7, la variación del SNR medido cambia según la función *wavelet* . Como se observa, la función bior3.3 presenta una diferencia de aproximadamente 3 dB con respecto a las demás funciones *wavelet* .

Cuadro 6.7: Comparación del SNR de la señal procesada producido por cada función *wavelet* para ruido blanco y recorte fuerte

Wavelet	**Promedio SNR**	**Desv Est SNR**	**Min SNR**	**Max SNR**
bior3.3	4,516	0,146	3,908	4,766
coif5	7,638	0,499	6,352	8,686
db5	7,453	0,495	5,984	8,438
dmey	7,769	0,494	6,490	8,699
sym8	7,588	0,501	6,213	8,487

No obstante, del caso anterior si solo se cambia el método de recorte, resultados mostrados en la tabla 6.8, la variación ahora es menor. Inclusive, como se puede observar en este caso, ahora es la función db5 la que muestra una diferencia apreciable en el valor del SNR medido.

Cuadro 6.8: Comparación del SNR de la señal procesada producido por cada función *wavelet* para ruido blanco y recorte suave

Wavelet	**Promedio SNR**	**Desv Est SNR**	**Min SNR**	**Max SNR**
bior3.3	6,229	0,477	5,198	7,064
coif5	6,152	0,697	4,865	7,564
db5	5,857	0,674	4,458	7,267
dmey	6,380	0,746	4,933	7,860
sym8	6,087	0,685	4,774	7,369

Entonces, con base en las observaciones realizadas sobre los datos de las tablas 6.6, 6.7 y 6.8, se puede deducir que la variación de los resultados medidos no depende exclusivamente de la función *wavelet* como se plantea en la hipótesis de esta investigación. Factores como el ruido a suprimir y el tipo de recorte de coeficientes también impactan directamente los resultados del método de eliminación de ruido. Es decir, no basta con seleccionar adecuadamente la función *wavelet* para cada ruido de estudio sino que además se debe determinar cuidadosamente el tipo de recorte de coeficientes en el dominio *wavelet* .

Se debe aclarar apropiadamente que esta conclusión es válida principalmente porque las cinco funciones *wavelet* seleccionadas para reducción de ruido son recomendadas para señales de voz. Si se consideran otras funciones no aptas para señales de voz la variación de resultados sería evidente según la función *wavelet* . Este caso se muestran en la tabla 6.9, donde se compara la función *wavelet* dmey, la cual presenta el mejor caso para estos dos ruidos de estudio, contra la función haar, no recomendada para señales de voz. Se aprecia la diferencia de resultados para ruido blanco y ruido rosa.

Cuadro 6.9: Comparación del SNR de la señal procesada producido por función dmey contra función haar con recorte suave

Ruido	***Wavelet***	**Promedio SNR**	**Desv Est SNR**	**Min SNR**	**Max SNR**
blanco	dmey	6,13	0,73	5,15	7,34
	haar	3,78	0,54	3,17	4,92
rosa	dmey	3,13	0,15	2,95	3,38
	haar	2,50	0,18	2,27	2,87

6.3. Comparación de resultados con el filtro Wiener

Es importante destacar la importancia de comparar el método de eliminación de ruido en el dominio *wavelet* con un método común del área de estudio

como lo es el filtro Wiener. La descripción del filtro Wiener implementado en esta investigación se muestra en el anexo D. Las mediciones se realizan con la misma base de cincuenta señales de voz, los tres ruidos de estudio y el SNR inicial de 0 dB. El orden del filtro se seleccionó en 500, número determinado en forma práctica considerando tiempo de procesamiento y eficacia del método para suprimir ruido. Los detalles sobre esta selección también se añaden en el anexo D.

En la tabla 6.10 se muestran los valores estadísticos de SNR de las mediciones con filtro Wiener y el mejor caso del método con funciones *wavelet* para los tres ruidos de estudio. Para el ruido *babble* y el ruido blanco se observa como en promedio el método con funciones *wavelet* brinda mejores resultados, sin embargo para el ruido rosa el método con filtro Wiener produce una ligera mejoría. Otra observación es que el método con filtro Wiener produce distribuciones más amplias, por lo que se deduce que depende en mayor medida de la señal de entrada.

Cuadro 6.10: Comparación del SNR de la señal procesada entre el método con *wavelet* y el filtro Wiener

Ruido	**Método**	**Promedio SNR**	**Desv Est SNR**	**Min SNR**	**Max SNR**
babble	db5 suave	3,42	0,07	3,22	3,54
babble	Filtro Wiener	2,41	0,57	1,40	4,00
blanco	dmey fuerte	7,77	0,49	6,49	8,69
blanco	Filtro Wiener	5,35	0,49	4,29	6,53
rosa	dmey fuerte	3,46	0,07	3,26	3,60
rosa	Filtro Wiener	3,61	0,37	2,98	4,75

En la tabla 6.11 se muestran las mediciones correspondientes al MSE. Como se observa, en promedio para el ruido *babble* y el ruido rosa el filtro Wiener presenta mejores resultados, no obstante para el ruido blanco el método con funciones *wavelet* es ligeramente superior.

Cuadro 6.11: Comparación del MSE de la señal procesada entre el método con *wavelet* y el filtro Wiener

Ruido	**Método**	**Promedio MSE**	**Desv Est MSE**	**Min MSE**	**Max MSE**
babble	db5 suave	0,007	0,0001	0,0074	0,0079
babble	Filtro Wiener	0,003	0,0003	0,0027	0,0041
blanco	dmey fuerte	0,001	0,0001	0,0014	0,0021
blanco	Filtro Wiener	0,002	0,0001	0,0017	0,0026
rosa	dmey fuerte	0,008	0,0001	0,0081	0,0087
rosa	Filtro Wiener	0,002	0,0001	0,0024	0,0032

En la tabla 6.12 se muestran las mediciones para el parámetro PESQ. Es

importante recordar que en este parámetro se hace una resta entre el valor calculado considerando la señal en limpio y la señal ruidosa y el valor calculado considerando la señal procesada y la señal ruidosa. Para el caso de ruido *babble*, según las mediciones, no hay mejoría en la calidad del audio con el filtro Wiener. Contrariamente, para el ruido blanco, el filtro Wiener presenta mejores resultados que el método de funciones *wavelet*. Sin embargo, para el ruido rosa el método con recorte en el dominio *wavelet* presenta mejores resultados en cuanto a la calidad del audio.

Cuadro 6.12: Comparación del PESQ de la señal procesada entre el método con *wavelet* y el filtro Wiener

Ruido	**Método**	**Promedio PESQ**	**Desv Est PESQ**	**Min PESQ**	**Max PESQ**
babble	db5 suave	0,050	0,044	0,017	0,346
babble	Filtro Wiener	-0,061	0,027	-0,155	-0,010
blanco	dmey fuerte	0,035	0,022	-0,029	0,089
blanco	Filtro Wiener	0,054	0,025	0,027	0,144
rosa	dmey fuerte	0,116	0,038	0,045	0,265
rosa	Filtro Wiener	0,068	0,040	0,026	0,239

Como se muestra, nuevamente existe variación de acuerdo con los parámetros medidos inclusive al utilizar el método con filtro Wiener. Entonces si se restringe al valor del SNR, midiendo solamente la efectividad para suprimir ruido, el método con funciones *wavelet* brinda mejores resultados claramente para ruido *babble* y ruido blanco. Mientras que para ruido rosa ambos métodos presentan resultados muy similares. La ganancia de utilizar el método con funciones *wavelet* sería que se obtienen resultados eficaces para los tres ruidos de estudio.

6.4. Efectividad del método a diferentes SNR iniciales

En esta sección se compara la efectividad del mejor caso seleccionado para el método con funciones *wavelet*, así como el filtro Wiener a diferentes valores iniciales de SNR. En la figura 6.4 se muestra la curva del SNR inicial contra el SNR después del procesamiento para el caso ruido *babble*. Se puede observar como el método con la función db5 presenta un comportamiento tipo exponencial, con mejores resultados a valores bajos de SNR inicial. Por su parte el filtro Wiener muestra un comportamiento lineal. Para el rango de valores medidos, el método *wavelet* presenta mejores resultados.

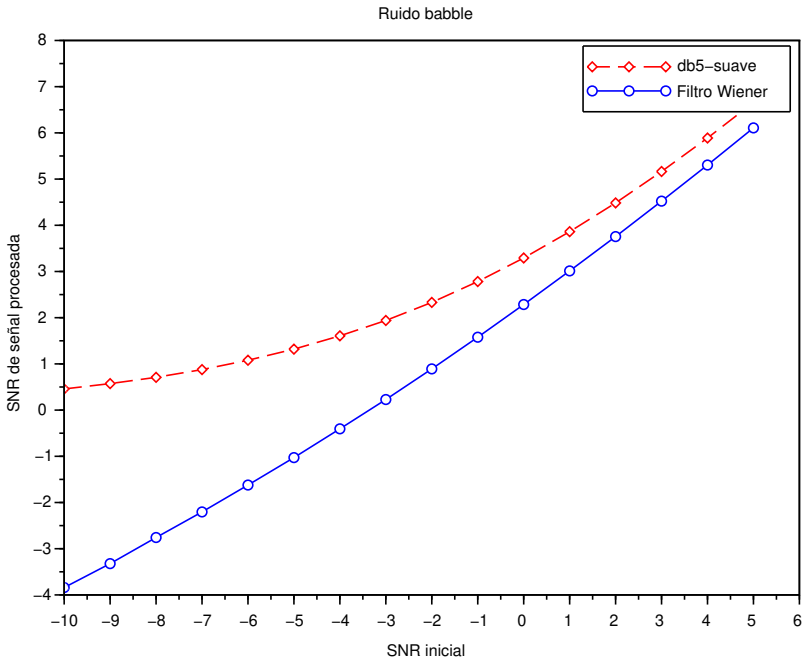

Figura 6.4: Curvas de la relación SNR inicial vs SNR de señal procesada para el caso de ruido *babble*

En la figura 6.5 para el caso ruido blanco, tanto el filtro Wiener como el método con la función dmey muestran una respuesta lineal. Sin embargo, a los diferentes valores de SNR iniciales probados el método *wavelet* presenta mejores resultados.

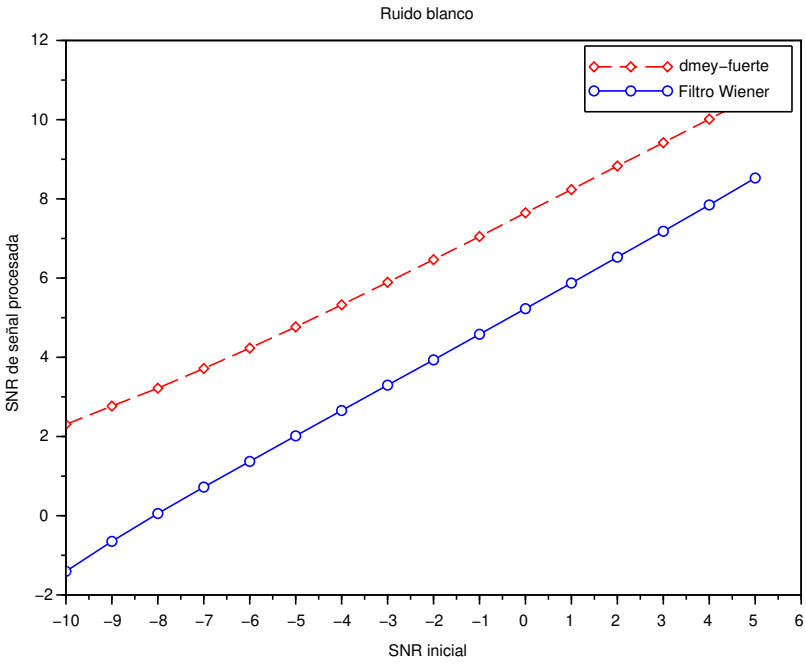

Figura 6.5: Curvas de la relación SNR inicial vs SNR de señal procesada para el caso de ruido blanco

Para el ruido rosa, mostrado en la figura 6.6, nuevamente el método con la función *wavelet* presenta un comportamiento curvilíneo y el filtro Wiener lineal. Inclusive, ambas curvas se cruzan en aproximadamente 0 dB. Por lo que se deduce que para valores de SNR iniciales menores a 0 dB es recomendable el método con la función dmey. Sin embargo, para valores de SNR iniciales mayores a 0 dB resulta más efectivo el método con filtro Wiener. No obstante, se debe considerar los casos en los que se desea eliminar ruido sobre una señal con niveles de SNR superiores a 0 dB, por lo que se podría decir que el método con funciones *wavelet* sería más efectivo en la práctica ya que cubre los rangos más problemáticos de SNR inicial.

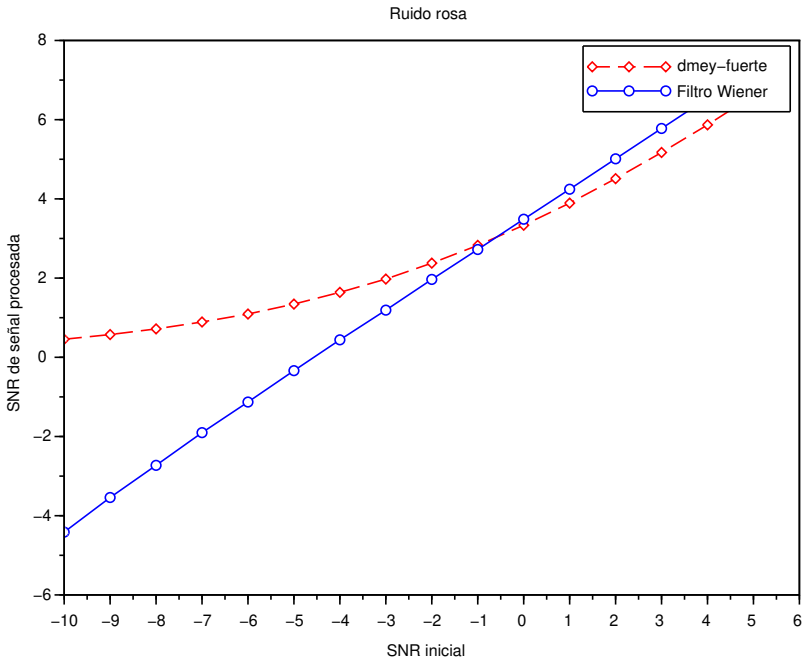

Figura 6.6: Curvas de la relación SNR inicial vs SNR de señal procesada para el caso de ruido rosa

Finalmente, se puede observar de nuevo como el método con funciones *wavelet* presenta mejor respuesta al compararse con el filtro Wiener. En este caso, como se mostró en las curvas anteriores, el método con funciones *wavelets* resulta más efectivo a diferentes valores de SNR inicial. Esto con la excepción del ruido rosa a valores superiores a cero de SNR inicial, considerándolo un caso poco probable en materia de reducción de ruido como se comentó antes.

No obstante, a pesar de que el método con funciones *wavelet* presenta mejores resultados, el filtro Wiener tiene sus ventajas sobre el primero. La flexibilidad de desarrollo en prácticamente cualquier plataforma hace que el filtro Wiener sea más accesible a los desarrolladores. Por otra parte se puede jugar con el parámetro del orden del filtro, así se decide el balance entre efectividad del método contra tiempo de procesamiento según las necesidades de cada aplicación.

7 Conclusiones

La hipotesis de esta investigación plantea la importancia de la selección de la función *wavelet* para el método de eliminación de ruido en su dominio. Tal y como se ha demostrado en esta investigación, la selección de esta función impacta directamente la efectividad del algoritmo, como se observó en los resultados de la relación señal a ruido obtenidos al comparar la función dmey contra la función haar. No obstante, también se comprobó que la función *wavelet* no es el único factor determinante en la efectividad del método, influyen también la naturaleza del ruido así como el tipo de recorte de coeficientes.

Cómo se plateó a lo largo de la investigación, el tipo de ruido a suprimir implica diferencias tanto en los resultados obtenidos como en la función *wavelet* recomendada. La investigación consideró dos ruidos artificiales como lo son el ruido blanco y el ruido rosa así como un ruido natural como lo es el *babble*. Los dos primeros, de similar naturaleza, son tratados adecuadamente por la función *wavelet* dmey. Para el caso del ruido *babble*, la función *wavelet* con mejores resultados fue db5. En este hecho se observa el impacto de la naturaleza del ruido sobre el algoritmo.

El tipo de recorte de coeficientes en el dominio *wavelet* es otro factor determinante para la efectividad del método. Como se observó de las mediciones realizadas, el tipo de recorte puede variar los resultados obtenidos de forma considerable. Inclusive, aplicando la misma función *wavelet* y variando únicamente el tipo de recorte el valor del SNR medido puede variar hasta 3 dB. Se observó además que tanto el recorte fuerte como el suave brindan resultados aceptables, no obstante, el recorte medio siempre fue inferior con respecto a los dos anteriores. El recorte medio fue recomendado para señales de grabaciones musicales, si embargo, para señale de voz su efectividad no es apreciable en esta investigación.

De la comparación de resultados con el filtro Wiener se obtienen varias conclusiones interesantes. Si se considera la relación señal a ruido, el método con funciones *wavelet* brinda mejores resultados con lo que se puede deducir que su capacidad para suprimir ruido es superior. Inclusive, al comparar resultados cambiando el SNR inicial, de igual forma el método con funciones *wavelets* es mejor. Sin embargo, el filtro Wiener es superior el algunos casos, por ejemplo considerando ruido rosa, el filtro produce señales con menos error

entre la señal original y la procesada. Otro aspecto importante es que si bien se determinó que el método con funciones *wavelet* produce mejores resultados en cuanto a la eliminación de ruido, su programación es más complicada que el filtro Wiener. Para el caso de la investigación se contó con la librería de la transformada *wavelet* en Scilab, por lo que el desarrollo del algoritmo fue sencillo. No obstante, si se desea aplicar este método en alguna otra plataforma se debería partir de cero incluyendo el desarrollo de la misma transformada *wavelet* así como su transformada inversa. Por esta razón, el filtro Wiener se puede considerar como un método más fácil de desarrollar en diversas plataformas, por lo que da mayor flexibilidad al desarrollador. No obstante, estudios recientes como el caso de Lee et al. (2019) permiten aplicar la transformada *wavelet* en Python, de esta forma se reduce drásticamente la complejidad de aplicar el método de eliminación en el dominio *wavelet* con un lenguaje de programación de alto nivel.

Otro aspecto a considerar es la calidad del audio. A lo largo del desarrollo de la investigación este aspecto fue el más problemático. Cada vez que se trabaja con ruido y con métodos para su eliminación se presenta el caso en que se distorsiona la señal original durante el proceso de filtrado. El método en el dominio *wavelet*, a pesar de que teóricamente la transformada separa las componentes de la señal original de las ruidosas, impacta al igual que muchos otros métodos la calidad del audio procesado. Las mediciones mostraron que existen casos en que hay que determinar el balance entre la eficacia del método para eliminar ruido contra la calidad de la señal procesada. Como en muchas áreas científicas, no se puede obtener los dos beneficios al mismo tiempo. Para la investigación, se le dio mayor prioridad a la eliminación de ruido, sin embargo, se trató de no abandonar la calidad de la señal procesada.

De las mediciones realizadas sin duda alguna la de mayor complejidad fue la del parámetro PESQ. La importancia de esta medición es que permite no dejar de lado la calidad del audio procesado, ya que como se comentó anteriormente, el algoritmo desarrollado impacta la calidad de la señal procesada. La complejidad no se trató del desarrollo del algoritmo para su medición, pues bien se utilizó una librería externa, sino en los resultados obtenidos. Las primeras mediciones realizadas considerando la señal procesada contra la señal ruidosa mostraban resultados poco realistas. Debido a la comparación que realiza el algoritmo, las señales procesadas menos exitosas, es decir, con ruido considerable, al compararse con la señal ruidosa mostraban valores cercanos a cinco, que indicaba que la calidad de la señal era muy buena, sin embargo es un resultado contradictorio porque si bien las dos señales son muy similares para el algoritmo, desde el punto de vista de la supresión de ruido es un caso poco exitoso. Por esta razón, se decidió realizar la medición en dos

etapas, considerando la medición entre la señal original y la señal ruidosa así como entre la señal procesada y la señal ruidosa. De esta forma la medición del PESQ, es la diferencia entre estas dos mediciones y se analizó aumentos o disminuciones tras aplicar el método. Los resultados mostraron valores muy pequeños, considerando que la medición del PESQ es una calificación de uno a cinco, de mala a excelente calidad. Sin embargo, estos números son esperados ya que se trabajó con señales que cuentan con ruido agregado y en este caso la medición se vuelve muy sensible.

Finalmente, tal y como se plantea en los objetivos de esta investigación se logra comparar cinco funciones *wavelets* y su efectividad para eliminación de ruido en su dominio. Se logra desarrollar el algoritmo de procesamiento de forma automática y fácil de aplicar en la plataforma Scilab, tal que cualquier usuario con dicha plataforma podría utilizarlo e inclusive modificar a su conveniencia. Se determinó la mejor función *wavelet*, con su respectivo tipo de recorte para los tres ruidos de estudio. La efectividad del método se comparó con el filtro Wiener y además se determinó que su efectividad es superior, no obstante, se comprobó que el filtro es más flexible para desarrollar en otras plataformas.

El aporte de esta investigación al campo de estudio reside en la viabilidad de considerar este método en aplicaciones comunes donde se requiere suprimir ruido. Los resultados mostraron que seleccionando la función *wavelet* y el tipo de recorte adecuado se pueden obtener resultados superiores al filtro Wiener. Para simplificar tal tarea, la presente investigación se encargó de determinar cuales serían las funciones *wavelet* recomendadas así como sus tipos de recorte según los tradicionales. Ahora bien, como se ha mencionado habrá que explorar el traslado de este método en otras plataformas. Investigaciones futuras podrían explorar el desarrollo de este método en plataformas como Python, el cual es muy flexible para el manejo de expresiones matemáticas. De la misma forma, los tipos de recorte actuales son variados y existen más de los tres considerados en este estudio.

Bibliografía

Bömers, F. (2000). Wavelets in real time digital audio processing: Analysis and sample implementations. Tesis de Maestría, University of Manheim, Dept. of Computer Science IV.

Daubechies, I. (1992). *Ten Lectures on Wavelets*. CBMS-NSF Regional Conference Series in Applied Mathematics. Society for Industrial and Applied Mathematics.

Debnath, L. y Shah, F. (2017). *Lecture Notes on Wavelet Transforms*. Springer.

Debnath, L. y Shah, F. A. (2015). *Wavelet transforms and their applications*. Springer.

Delgado, A. (2010). Restauración de grabaciones de audio antiguas mediante el método de recorte de coeficientes en el dominio wavelet. Tesis de Maestría, Universidad de Costa Rica, San José.

Deng, N. y Jiang, C. (2012). Selection of optimal wavelet basis for signal denoising. En *2012 9th International Conference on Fuzzy Systems and Knowledge Discovery*, páginas 1939–1943. IEEE.

Donoho, D. L. (1995). De-noising by soft-thresholding. *IEEE Transactions on Information Theory*, 41(3):613–627.

Donoho, D. L. y Johnstone, I. M. (1994). Threshold selection for wavelet shrinkage of noisy data. En *Proceedings of 16th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, volumen 1, páginas A24–A25 vol.1.

Farouk, M. H. (2014). *Application of wavelets in speech processing*. Springer.

Festvox (2003). *CMU ARCTIC speech synthesis databases*. `http://www.festvox.org/cmu_arctic/index.html`.

Haykin, S. y Moher, M. (2006). *An Introduction to Analog and Digital Communications, 2nd Edition*. Wiley Textbooks.

Hidayat, R., Bejo, A., Sumaryono, S., y Winursito, A. (2018). Denoising speech for mfcc feature extraction using wavelet transformation in speech recognition system. En *2018 10th International Conference on Information Technology and Electrical Engineering (ICITEE)*, páginas 280–284. IEEE.

Koenig, M. (2017). Party crowd sounds: Effects: Sound bites: Sound clips from soundbible.com. `http://soundbible.com/2163-Party-Crowd.html`.

Kominek, J. y Black, A. (2003). Databases for speech synthesis. Reporte técnico, Carnegie Mellon University, Pittsburgh, USA.

Krishnamurthy, N. y Hansen, J. H. L. (2008). Speech babble: Analysis and modeling for speech systems. En *2008 IEEE International Conference on Acoustics, Speech and Signal Processing*, páginas 4505–4508.

Lee, G. R., Gommers, R., Waselewski, F., Wohlfahrt, K., y O., A. (2019). Pywavelets: A python package for wavelet analysis. *Journal of Open Source Software*, 4(36):1237.

Lin, Y. y Cai, J. (2010). A new threshold function for signal denoising based on wavelet transform. En *2010 International Conference on Measuring Technology and Mechatronics Automation*, volumen 1, páginas 200–203. IEEE.

Long, Y., Gang, L., y Jun, G. (2004). Selection of the best wavelet base for speech signal. En *Proceedings of 2004 International Symposium on Intelligent Multimedia, Video and Speech Processing, 2004.*, páginas 218–221. IEEE.

Meyer, Y. (1993). Wavelets-algorithms and applications. *Wavelets-Algorithms and applications Society for Industrial and Applied Mathematics Translation., 142 p.*

Miao, W. (2019). *PESQ Wrapper for Python Users*. `https://github.com/ludlows/python-pesq`.

Nongpiur, R. C. (2008). Impulse noise removal in speech using wavelets. En *2008 IEEE International Conference on Acoustics, Speech and Signal Processing*, páginas 1593–1596. IEEE.

Pathak, R. S. (2009). *The wavelet transform*, volumen 4. Springer Science & Business Media.

Praveen, L. (2012). Measuring speech quality of laptop microphone system using pesq. Tesis de Maestría, Blekinge Institute of Technology, Blekinge, Karlskrona.

Rémy, J. y Letamendia, C. (2014). *LTE Standards*. Iste Series. Wiley.

Shukla, K. K. y Tiwari, A. K. (2013). *Efficient algorithms for discrete wavelet transform: with applications to denoising and fuzzy inference systems*. Springer Science & Business Media.

Teolis, A. y Benedetto, J. J. (1998). *Computational signal processing with wavelets*, volumen 182. Springer.

UIT-T (2001). Serie p: Calidad de transmision telefonica, instalaciones telefonicas y redes locales. Reporte técnico, UIT-T, Ginebra, Suiza.

van Exter, M. (2003). Noise and signal processing. Reporte técnico, Universiteit Leiden, Leiden, Netherlands.

Vaseghi, S. (2008). *Advanced Digital Signal Processing and Noise Reduction*. Wiley.

Walden, A. T., Percival, D. B., y McCoy, E. J. (1998). Spectrum estimation by wavelet thresholding of multitaper estimators. *IEEE Transactions on Signal Processing*, 46(12):3153–3165.

Xiaoli, C. y Mao, T. (2004). Implementation of wavelet shrinkage de-noising based on dsp. En *2004 IEEE International Conference on Semiconductor Electronics*, páginas 5–pp. IEEE.

Yong, T. y Qiang, W. (2010). The realization of wavelet threshold noise filtering algorithm in dsp. En *2010 International Conference on Measuring Technology and Mechatronics Automation*.

Zhang, P., Ni, Y., y Wang, J. (2018). Research on signal denoising method based on adaptive lifting wavelet transform. En *2018 International Conference on Sensor Networks and Signal Processing (SNSP)*, páginas 306–310. IEEE.

Zhigang, D., Jingxuan, Z., y Chunrong, J. (2013). An improved wavelet threshold denoising algorithm. En *2013 Third International Conference on Intelligent System Design and Engineering Applications*, páginas 297–299. IEEE.

A Función sobre eliminación de ruido en el dominio *wavelet*

A.1. Función del algoritmo

En las siguientes líneas se muestra la función desarrollada para aplicar el algoritmo principal de esta investigación. Los detalles principales se comentaron en el capítulo 5, por lo que se incluye en esta sección la función completa.

```
function [ss, s_n, Fs, bits]=wavelet_denoising(senal, ruido, wavelet, recorte,
↪  snr_in, graph_enable, output_wav_enable)

    //funciones sci
    exec('C:\Users\Admin\Documents\Tesis\Scripts\limit_escala.sci');
    exec('C:\Users\Admin\Documents\Tesis\Scripts\limit_universal.sci');
    exec('C:\Users\Admin\Documents\Tesis\Scripts\recorte_fuerte.sci');
    exec('C:\Users\Admin\Documents\Tesis\Scripts\recorte_suave.sci');
    exec('C:\Users\Admin\Documents\Tesis\Scripts\recorte_medio.sci');
    exec('C:\Users\Admin\Documents\Tesis\Scripts\get_gain_voz.sci');

    // señales y ruidos .wav
    my_voz_wav = senal;
    my_blanco_wav =
    ↪  "C:\Users\Admin\Documents\Tesis\Audio_signals\50_frases_ingles\blanco.wav";
    my_rosa_wav =
    ↪  "C:\Users\Admin\Documents\Tesis\Audio_signals\50_frases_ingles\rosa.wav";
    my_babble_wav =
    ↪  "C:\Users\Admin\Documents\Tesis\Audio_signals\50_frases_ingles\babble.wav";

    // valores soportados: [1 2 3 4]
    decomp_levels = [1 2 3 4];
    //valores soportados: 'universal', 'escala'
    my_limit = 'universal';

    // informacion en consola
    printf('[INFO] Señal de voz seleccionada: %s\n', senal);
    printf('[INFO] Wavelet seleccionada: %s\n', wavelet);
```

```
25      printf('[INFO] Procesando con ruido: %s\n', ruido);
26      //printf('[INFO] Procesando con %i niveles de
        ↪  descomposición\n',length(decomp_levels));
27      printf('[INFO] Procesando con umbral de recorte: %s\n',recorte);
28
29      // Voz
30      [s_voz,Fs,bits] = wavread(my_voz_wav);
31
32      // numero de muestras depende de la señal de voz
33      n = length(s_voz)
34
35      //seleccion del ruido a eliminar
36      // s_n es la senal a procesar
37      if ruido == 'blanco' then
38          [s_blanco,Fs,bits] = wavread(my_blanco_wav,n);
39          gain_voz = get_gain_voz(s_voz, s_blanco, snr_in);
40          s_voz = gain_voz .* s_voz;
41          voz_rms = sqrt(mean(s_voz.^2));
42          blanco_rms = sqrt(mean(s_blanco.^2));
43          SNR_med = 20*log10(voz_rms/blanco_rms)
44          s_n = s_blanco + s_voz;
45
46      elseif ruido == 'rosa' then
47          [s_rosa,Fs,bits] = wavread(my_rosa_wav,n);
48          gain_voz = get_gain_voz(s_voz, s_rosa, snr_in);
49          s_voz = gain_voz .* s_voz;
50          voz_rms = sqrt(mean(s_voz.^2));
51          rosa_rms = sqrt(mean(s_rosa.^2));
52          SNR_med = 20*log10(voz_rms/rosa_rms);
53          s_n = s_rosa + s_voz;
54
55      elseif ruido == 'babble' then
56          [s_babble,Fs,bits] = wavread(my_babble_wav,n);
57          gain_voz = get_gain_voz(s_voz, s_babble, snr_in);
58          s_voz = gain_voz .* s_voz;
59          voz_rms = sqrt(mean(s_voz.^2));
60          babble_rms = sqrt(mean(s_babble.^2));
61          SNR_med = 20*log10(voz_rms/babble_rms);
62          s_n = s_babble + s_voz;
63
64      elseif ruido == 'ninguno' then
65          s_n = s_voz;
66      else
```

```
67          disp("[ERROR] Indique el tipo de ruido")
68      end
69
70      // descomposicion en niveles
71      [C,L] = wavedec(s_n,length(decomp_levels),wavelet);
72
73      // coeficientes de aproximacion
74      cA_1 = appcoef(C,L,wavelet,1);
75      cA_2 = appcoef(C,L,wavelet,2);
76      cA_3 = appcoef(C,L,wavelet,3);
77      cA_4 = appcoef(C,L,wavelet,4);
78
79      // coeficientes de detalle
80      cD_1 = detcoef(C,L,1);
81      cD_2 = detcoef(C,L,2);
82      cD_3 = detcoef(C,L,3);
83      cD_4 = detcoef(C,L,4);
84
85      if my_limit == 'escala' then
86          T_j1 = limit_escala(length(s_n), cD_1)
87          T_j2 = limit_escala(length(s_n), cD_2)
88          T_j3 = limit_escala(length(s_n), cD_3)
89          T_j4 = limit_escala(length(s_n), cD_4)
90      else
91          T_u = limit_universal(length(s_n), cD_1);
92          T_j1 = T_u;
93          T_j2 = T_u;
94          T_j3 = T_u;
95          T_j4 = T_u;
96      end
97
98      // realce de sonido en cA
99      c_amp = 1.15;
100
101     // aplicar recorte fuerte de coeficientes
102     if recorte == 'fuerte' then
103         cD_1 = recorte_fuerte(cD_1,T_j1,c_amp);
104         cD_2 = recorte_fuerte(cD_2,T_j2,c_amp);
105         cD_3 = recorte_fuerte(cD_3,T_j3,c_amp);
106         cD_4 = recorte_fuerte(cD_4,T_j4,c_amp);
107
108     elseif recorte == 'medio' then
109         cD_1 = recorte_medio(cD_1,T_j1,c_amp);
```

```
110         cD_2 = recorte_medio(cD_2,T_j2,c_amp);
111         cD_3 = recorte_medio(cD_3,T_j3,c_amp);
112         cD_4 = recorte_medio(cD_4,T_j4,c_amp);
113
114     elseif recorte == 'suave' then
115         cD_1 = recorte_suave(cD_1,T_j1,c_amp);
116         cD_2 = recorte_suave(cD_2,T_j2,c_amp);
117         cD_3 = recorte_suave(cD_3,T_j3,c_amp);
118         cD_4 = recorte_suave(cD_4,T_j4,c_amp);
119
120     elseif recorte == 'ninguno' then
121         //disp("[INFO] Sin recorte de coeficientes")
122     else
123         disp("[ERROR] Indique el tipo de recorte")
124     end
125
126     // unir vector coeficientes
127     C_deno = [cA_4 cD_4 cD_3 cD_2 cD_1];
128
129     // reconstruccion de la señal
130     ss = waverec(C_deno,L,wavelet);
131
132     if graph_enable == 'on' then
133         subplot(2,1,1)
134         plot(s_n)
135         xlabel('n')
136         ylabel('s(n)')
137         title('(a) Señal con ruido')
138
139         subplot(2,1,2)
140         plot(ss)
141         xlabel('n')
142         ylabel('ss(n)')
143         title('(b) Señal procesada')
144     end
145
146     // medicion de SNR y MSE señal procesada
147     // error señal original vs procesada
148     subs_error = s_voz - ss;
149
150     //mse = mean( (s_voz(:)-ss(:)).^2);
151     mse = 1/n .* sum(subs_error.^2);
152
```

```
153     //error max y promedio como referencia
154     mean_error = abs(mean(subs_error))
155     max_error = max(subs_error)
156
157     //si NO se aplica recorte cambia la info en consola y no hace falta
        ↪  calcular SNR
158     if recorte ~= 'ninguno' then
159         printf('[INFO] Inicialmente SNR (dB) = %f\n', SNR_med);
160         // SNR = |ss| / |ss - s_voz|
161         ss_rms = sqrt(mean(ss.^2));
162         n_ss = ss - s_voz;
163         n_rms = sqrt(mean(n_ss.^2));
164         SNR_ss = 20*log10(ss_rms/n_rms);
165
166         printf('[INFO] SNR de señal procesada (dB) = %f\n', SNR_ss);
167         printf('[INFO] MSE = %.8f\n', mse);
168     else
169         printf('[INFO] MSE = %.9g\n', mse);
170         printf('[INFO] Error promedio = %.9g\n', mean_error);
171         printf('[INFO] Error máximo = %.9g\n', max_error);
172     end
173
174     //audio de voz sucia y voz procesada con el algoritmo
175     senal_name = part(senal, 63:length(senal)-4)
176     if output_wav_enable == 'on' then
177         voz_sucia = "C:\Users\Admin\Desktop\outputs_wavelet\voz_sucia_" +
            ↪  senal_name + "_" + ruido + "_" + wavelet + "_" + recorte + "_" +
            ↪  string(snr_in) + ".wav";
178         wavwrite(s_n, Fs, bits, voz_sucia)
179         voz_procesada =
            ↪  "C:\Users\Admin\Desktop\outputs_wavelet\voz_procesada_" +
            ↪  senal_name + "_" + ruido + "_" + wavelet + "_" + recorte + "_" +
            ↪  string(snr_in) + ".wav";
180         wavwrite(ss, Fs, bits, voz_procesada)
181     end
182
183 endfunction
```

A.2. Test para mediciones cíclicas

El desarrollo de la función con argumentos en el entorno Scilab permite agilizar el proceso de mediciones para grupos de señales, ruidos, funciones *wavelets*,

entre otros. A continuación se muestra el script desarrollado para la aplicación del algoritmo sobre el grupo de señales de estudio considerando todos los demás parámetros. Como se observa, basta con hacer un barrido tras otro para obtener en una sola ejecución los resultados de todos los casos de estudio.

```
exec('C:\Users\Admin\Documents\Tesis\Scripts\wavelet_denoising.sci');
my_path = "C:\Users\Admin\Documents\Tesis\Audio_signals\50_frases_ingles"

noise_vect = ['blanco', 'rosa', 'babble'];
wavelet_vect = ['dmey','coif5','sym8','db5','bior3.3'];
recorte_vect = ['fuerte', 'suave', 'medio'];
snr_vect = [0];

for senal = 1:1:50
    for ruido = 1:3
        for wavelet = 1:5
            for recorte = 1:3
                for snr = 1:1
                    if senal < 10 then
                        frase = "\arctic_a000" + string(senal) + ".wav";
                    else
                        frase = "\arctic_a00" + string(senal) + ".wav";
                    end
                    senal_voz = my_path + frase;
                    [ss, s_n, Fs, bits] = wave-
                    ↪ let_denoising(senal_voz,noise_vect(ruido),wavelet_vect(wavelet),recorte_vect(recorte)
                end
            end
        end
    end
end
```

B Función sobre ganancia de la señal de voz

En las siguientes líneas se muestra la lógica desarrollada para la función `get_gain_voz()`. Esta función permite definir la ganancia que se debe aplicar a la señal de voz de entrada para que al realizar la mezcla con el ruido se obtenga el SNR inicial deseado.

Como se observa, los argumentos necesarios son la señal de voz, la señal del ruido y el SNR deseado. Se hace un barrido sobre el parámetro `a`, representando la ganancia de la señal de voz y se calcula el SNR para cada valor del barrido. Cuando la diferencia entre el SNR actual y el SNR deseado es menor a 0,2 dB se rompe el ciclo ya que se encontró el valor `a` necesario para cumplir aproximadamente el SNR solicitado por el usuario.

El barrido sobre el parámetro `a` se selecciona de 0 a 8 con pasos de 0,02. Estos valores fueron seleccionados de forma práctica considerando cubrir un rango de SNR deseado de -15 dB a 10 dB para las señales y los ruidos de estudio. De igual forma el valor de la diferencia de SNR igual a 0,2 dB se seleccionó de forma práctica. El tiempo de procesamiento fue también otro aspecto considerado por lo que para los valores actuales el cálculo toma en el peor de los casos unos tres segundos. Es importante destacar que tanto el barrido como el valor de la diferencia se pueden modificar según otros casos de estudio, no obstante impactaría el tiempo de procesamiento.

```
function [gain_senal]=get_gain_voz(s_senal, s_ruido, SNR_target)

    a_found = 0;
    for a = 0:0.02:8
        s_senal = a .* s_voz;
        senal_rms = sqrt(mean(s_senal.^2));
        ruido_rms = sqrt(mean(s_ruido.^2));
        SNR_meas = 20*log10(senal_rms/ruido_rms);
        delta_SNR = abs(SNR_meas - SNR_target);
        //disp(SNR_meas)
        if delta_SNR < 0.2 then
            a_found = 1;
            break;
        end
```

```
15      end
16
17      if a_found ~= 1 then
18          disp('[ERROR] Parametro gain_senal no encontrado, verifique su SNR deseado');
19      end
20
21      gain_senal = a;
22
23  endfunction
```

C Medición de PESQ

Para realizar las mediciones de PESQ se utiliza la librería de Python elaborada por Miao (2019), la cual combina métodos de Python con las funciones en C desarrolladas por la UIT para medición del PESQ según el estándar P862.2.

Como establece Miao (2019), para hacer uso de las funciones de su librería solo basta con instalar las librerías de Python Numpy, Cython así como el compilador de C (GCC usualmente). Una vez que se cumplen los requisitos, el cálculo de PESQ es posible tal y como se describe en las siguientes líneas.

Para facilidad de esta investigación se desarrolló una función en Python que agiliza el proceso de medición del PESQ tomando el grupo de señales de estudio. Para esta función se parte de que los audios .wav de la señales con ruido añadido (señal sucia), la señal original (señal limpia) y la señal después de aplicar el algoritmo (señal procesada) se encuentran en alguna ruta del sistema. Posteriormente, basta con aplicar el algoritmo de medición desarrollado por Miao (2019) haciendo un barrido sobre las señales, los ruidos de estudio, las funciones *wavelet* y los tipos de recorte.

Se hace de nuevo la aclaración de que la medición del parámetro PESQ se realiza en dos etapas, primero considerando la señal limpia contra la señal ruidosa y luego considerando la señal procesada contra la señal ruidosa. En el análisis final se verifica la diferencia entre estas dos mediciones. Realizar ambas permite de forma más realista observar el efecto del algoritmo de eliminación de ruido sobre la calidad del audio.

Para las señales procesadas con el filtro Wiener se aplica una función muy similar dado que esos audios también se guardan en ruta definida del sistema.

```
1  import os, sys
2  from scipy.io import wavfile
3  from pesq import pesq
4
5  def test_pesq_wavelet():
6      """
7      Medicion del PESQ considerando el metodo de eliminacion de ruido en el
   ↪  dominio wavelet.
8      """
9      my_clean_signals_dir =
       ↪  r"C:\Users\Admin\Documents\Tesis\Audio_signals\50_frases_ingles"
10     my_output_dir = r"C:\Users\Admin\Desktop\outputs_wavelet"
```

```
11
12      noise_vect = ['blanco', 'rosa', 'babble']
13      wavelet_vect = ['dmey','coif5','sym8','db5','bior3.3']
14      recorte_vect = ['fuerte', 'suave', 'medio']
15      snr_vect = [0]
16
17      for senal in range(1,51):
18          for ruido in noise_vect:
19              for wavelet in wavelet_vect:
20                  for recorte in recorte_vect:
21                      for snr_in in snr_vect:
22
23                          if senal < 10:
24                              frase = "arctic_a000" + str(senal)
25                          else:
26                              frase = "arctic_a00" + str(senal)
27
28                          # construccion de path senales
29                          procesada = "voz_procesada_" + frase + "_" + ruido +
                            ↪  "_" + wavelet + "_" + recorte + "_" + str(snr_in)
                            ↪  + ".wav"
30                          sucia = "voz_sucia_" + frase + "_" + ruido + "_" +
                            ↪  wavelet + "_" + recorte + "_" + str(snr_in) +
                            ↪  ".wav"
31                          limpia = frase + ".wav"
32
33                          # print each case
34                          print("[INFO] Señal de estudio : {}".format(limpia))
35                          print("[INFO] Wavelet : {}".format(wavelet))
36                          print("[INFO] Ruido : {}".format(ruido))
37                          print("[INFO] Recorte : {}".format(recorte))
38                          print("[INFO] SNR inicial = {}".format(str(snr_in)))
39
40                          # unir path + senal
41                          my_voz_procesada = os.path.join(my_output_dir,
                            ↪  procesada)
42                          my_voz_sucia = os.path.join(my_output_dir, sucia)
43                          senal_limpia = os.path.join(my_clean_signals_dir,
                            ↪  limpia)
44
45                          print("[INFO] Señal procesada:
                            ↪  {}".format(my_voz_procesada))
```

```
46                      print("[INFO] Señal degradada:
                        ↪  {}".format(my_voz_sucia))
47                      print("[INFO] Señal limpia: {}".format(senal_limpia))
48
49                      # lecturas .wav
50                      rate, proc = wavfile.read(my_voz_procesada)
51                      rate, deg = wavfile.read(my_voz_sucia)
52                      rate, clean = wavfile.read(senal_limpia)
53
54                      # calculo PESQ sucia vs limpia
55                      my_wb_pesq = pesq(rate, clean, deg, 'wb')
56                      print("[INFO] PESQ limpia-sucia wideband medido es =
                        ↪  {:10.4f}".format(my_wb_pesq))
57
58                      # calclulo PESQ procesada vs limpia
59                      my_wb_pesq = pesq(rate, clean, proc, 'wb')
60                      print("[INFO] PESQ limpia-procesada wideband medido es
                        ↪  = {:10.4f}".format(my_wb_pesq))
```

D Filtro Wiener

D.1. Función del filtro Wiener

En las siguientes líneas se muestra la lógica desarrollada para la implementación del filtro Wiener, tomando en consideración las ecuaciones (2.32) y (2.33). Este procesamiento se define como una función en el entorno Scilab, donde sus argumentos son la señal de voz, el tipo de ruido, el orden del filtro, el SNR inicial deseado y las banderas para activar o desactivar los gráficos y los archivos .wav de salida. La estructura de la función es muy similar a la elaborada para el método con funciones *wavelets*, esto con el propósito de facilitar el procesamiento cuando se desea repetir cíclicamente diversas señales, ruidos y valores de SNR iniciales.

```
1  function [ss, s_n, Fs, bits]=wiener_filter(senal, ruido, filter_order, snr_in,
   ↪  graph_enable, output_wav_enable)
2
3      // funciones sci
4      exec('C:\Users\Admin\Documents\Tesis\Scripts\get_gain_voz.sci');
5
6      // señales y ruidos .wav
7      my_voz_wav = senal;
8      my_blanco_wav =
       ↪  "C:\Users\Admin\Documents\Tesis\Audio_signals\50_frases_ingles\blanco.wav";
9      my_rosa_wav =
       ↪  "C:\Users\Admin\Documents\Tesis\Audio_signals\50_frases_ingles\rosa.wav";
10     my_babble_wav =
       ↪  "C:\Users\Admin\Documents\Tesis\Audio_signals\50_frases_ingles\babble.wav";
11
12     //printf('[INFO] Procesando con filtro Wiener de orden %d\n',
       ↪  filter_order);
13     printf('[INFO] Señal de voz seleccionada: %s\n', senal);
14     printf('[INFO] Procesando con ruido: %s\n', ruido);
15     printf('[INFO] Orden del filtro: %i\n', filter_order);
16
17     // Voz
18     [s_voz,Fs,bits] = wavread(my_voz_wav);
19
```

```
20      // numero de muestras depende de la señal de voz
21      n = length(s_voz)
22
23      //seleccion del ruido a eliminar
24      // s_n es la senal a procesar
25      if ruido == 'blanco' then
26          [s_blanco,Fs,bits] = wavread(my_blanco_wav,n);
27          gain_voz = get_gain_voz(s_voz, s_blanco, snr_in);
28          s_voz = gain_voz .* s_voz;
29          voz_rms = sqrt(mean(s_voz.^2));
30          blanco_rms = sqrt(mean(s_blanco.^2));
31          SNR_med = 20*log10(voz_rms/blanco_rms)
32          s_n = s_blanco + s_voz;
33      elseif ruido == 'rosa' then
34          [s_rosa,Fs,bits] = wavread(my_rosa_wav,n);
35          gain_voz = get_gain_voz(s_voz, s_rosa, snr_in);
36          s_voz = gain_voz .* s_voz;
37          voz_rms = sqrt(mean(s_voz.^2));
38          rosa_rms = sqrt(mean(s_rosa.^2));
39          SNR_med = 20*log10(voz_rms/rosa_rms);
40          s_n = s_rosa + s_voz;
41      elseif ruido == 'babble' then
42          [s_babble,Fs,bits] = wavread(my_babble_wav,n);
43          gain_voz = get_gain_voz(s_voz, s_babble, snr_in);
44          s_voz = gain_voz .* s_voz;
45          voz_rms = sqrt(mean(s_voz.^2));
46          babble_rms = sqrt(mean(s_babble.^2));
47          SNR_med = 20*log10(voz_rms/babble_rms);
48          s_n = s_babble + s_voz;
49      elseif ruido == 'ninguno' then
50          s_n = s_voz;
51      else
52          disp("[ERROR] Indique el tipo de ruido")
53      end
54
55      // Ryy^-1 autocorrelacion de la señal ruidosa
56      r = xcorr(s_n);
57      Ryy = toeplitz(r(n:n+filter_order-1));
58
59      // r_xy correlación cruzada entre señal ruidosa y limpia
60      r_xy_temp = xcorr(s_n,s_voz);
61      r_xy = r_xy_temp(n:n+filter_order-1);
62
```

```
63      // w coeficientes del filtro
64      w = Ryy \ r_xy';
65
66      // filtrando la señal
67      ss = filter(w,1,s_n);
68
69      if graph_enable == 'on' then
70          subplot(2,1,1)
71          plot(s_n)
72          xlabel('n')
73          ylabel('s(n)')
74          title('Señal con ruido')
75
76          subplot(2,1,2)
77          plot(ss)
78          xlabel('n')
79          ylabel('s*(n)')
80          title('Señal procesada')
81      end
82
83
84      // medicion de SNR y MSE señal procesada
85      // error señal original vs procesada
86      subs_error = s_voz - ss;
87
88      //mse = mean( (s_voz(:)-ss(:)).^2);
89      mse = 1/n .* sum(subs_error.^2);
90
91      //error max y promedio como referencia
92      mean_error = abs(mean(subs_error))
93      max_error = max(subs_error)
94
95      printf('[INFO] Inicialmente SNR (dB) = %f\n', SNR_med);
96      // SNR = |ss| / |ss - s_voz|
97      ss_rms = sqrt(mean(ss.^2));
98      n_ss = ss - s_voz;
99      n_rms = sqrt(mean(n_ss.^2));
100     SNR_ss = 20*log10(ss_rms/n_rms);
101
102     printf('[INFO] SNR de señal procesada (dB) = %f\n', SNR_ss);
103     printf('[INFO] MSE = %.8f\n', mse);
104
105     //audio de voz sucia y voz procesada con el algoritmo
```

```
106     senal_name = part(senal, 63:length(senal)-4)
107     if output_wav_enable == 'on' then
108         voz_sucia = "C:\Users\Admin\Desktop\outputs_wiener\voz_sucia_" +
            ↪  senal_name + "_" + ruido + "_" + string(snr_in) + "_" +
            ↪  "wiener_filter" + ".wav";
109         wavwrite(s_n, Fs, bits, voz_sucia)
110         voz_procesada = "C:\Users\Admin\Desktop\outputs_wiener\voz_procesada_"
            ↪  + senal_name + "_" + ruido + "_" + string(snr_in) + "_" +
            ↪  "wiener_filter" + ".wav";
111         wavwrite(ss, Fs, bits, voz_procesada)
112     end
113
114 endfunction
```

D.2. Selección del orden del filtro

Para la selección del orden del filtro se realizó un barrido de 1 a 991 con pasos de 10 considerando una señal de voz. De esta forma al graficar para los tres ruidos de estudio, el SNR de la señal procesada contra el orden del filtro se obtiene la figura D.1. De la figura se observa como en aproximadamente el orden 150, el filtro Wiener alcanza su respuesta estacionaria. No obstante, para mejorar muy ligeramente la eficacia del filtro ante las diferentes señales de estudio, se decidió utilizar orden 500, el cual es más de tres veces el orden necesario para alcanzar la respuesta estacionaria. También, de forma práctica el orden 500 tampoco impacta gravemente el tiempo de procesamiento por lo que se considera un buen valor para esta investigación.

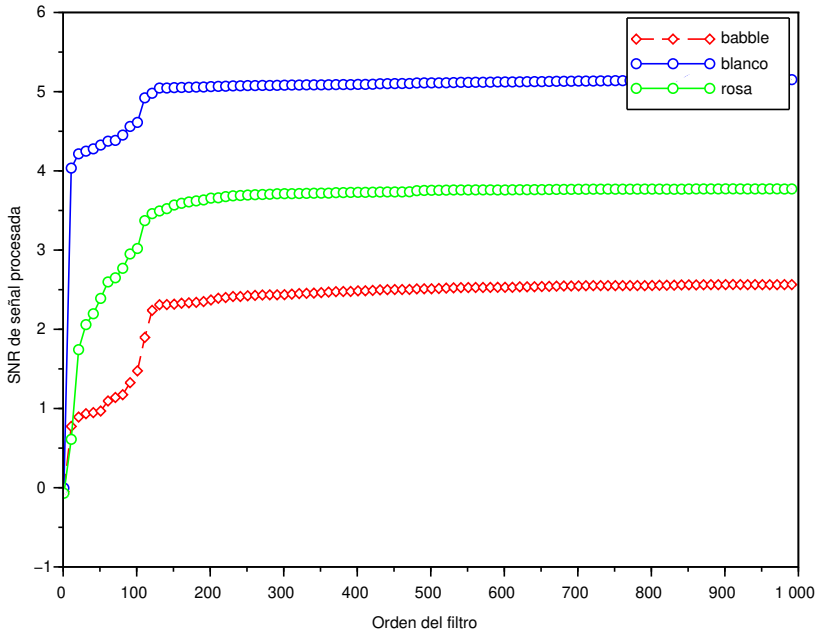

Figura D.1: Curvas del SNR de la señal procesada contra el orden del filtro Wiener