UNIVERSITAT JAUME I

Grado en Matemática Computacional

Estancia en Prácticas y Trabajo Final de Grado

Wavelets de Haar y Daubechies y sus aplicaciones.

Autor:
Nahuel Olivera Rodriguez

Supervisor:
Sergio Aguado González
Tutor académico:
Marina Murillo Arcila

Fecha de lectura: 27 de septiembre de 2018
Curso académico 2017/2018

Resumen

El análisis de señales se aplica a diferentes campos de estudios y una forma de extraer y tratar la información de estas señales es a través de las wavelets. En este documento se detallan algunos aspectos básicos sobre la teoría de wavelets, más concretamente se detallan las de Haar y las de Daubechies.

La estructura del trabajo se divide en dos partes. En la primera parte del trabajo se detalla la estancia en prácticas desarrolladas en la empresa Soluciones Cuatroochenta. En ella se explican las tecnologías y las tareas desarrolladas durante la misma.

En la segunda sección, se desarrolla el trabajo fin de grado que trata sobre las wavelets y sus aplicaciones. En primer lugar, se introducen algunos conceptos previos necesarios para poder entender el trabajo en profundidad. A continuación, se describen y analizan las wavelets de Haar y, seguidamente, se comparan con las de Daubechies. Finalmente, se describen algunas de las aplicaciones que tienen las wavelets como son la compresión y reducción del ruido, y haciendo uso del programa Matlab se proporcionan ejemplos y aplicaciones sobre el tratamiento de algunas señales usando este tipo de wavelets.

Palabras clave

Big data, Cassandra, Señales, Wavelet, Haar, Daubechies, Análisis de multirresolución (MRA), Transformada, Matlab

Keywords

Big data, Cassandra, Signals, Wavelet, Haar, Daubechies, Multirresolution Analysis (MRA), Transform, Matlab

Índice general

1. Estancia en Prácticas **9**

1.1. Introducción 9

1.2. La empresa 9

1.3. Objetivos de la estancia en prácticas 10

1.4. Software y herramientas utilizadas 11

1.4.1. Apache Cassandra [2] & DataStax DevCenter [5] 11

1.4.2. Apache Spark [3] 12

1.4.3. Node.JS [11] 12

1.4.4. Git [6] & Sourcetree [14] 12

1.4.5. JetBrains: WebStorm [7] 13

1.4.6. Postman [12] 14

1.4.7. D3.JS [4] 15

1.5. Planificación temporal de las tareas 16

1.5.1. Primer Informe 16

1.5.2. Segundo Informe 16

1.5.3. Tercer Informe 19

1.5.4. Cuarto Informe 21

1.6. Grado de consecución de los objetivos propuestos 23

2. Memoria TFG **25**

2.1. Conceptos previos 27

2.2. Wavelets de Haar 29

2.2.1. Scaling y wavelets 30

2.2.2. Análisis de multirresolución (MRA) 34

2.2.3. Transformada de Haar 38

2.3. Familia de wavelets ortogonales 40

2.3.1. Wavelets de Daubechies 40

2.4. Compresión de señales y reducción del ruido 47

2.4.1. Cuantización y umbralizado 47

2.4.2. Codificación 50

2.4.3. Medidas de aproximación 52

2.5. Aplicaciones en Matlab 53

2.5.1. Wavelets de Haar 53

2.5.2. Compresión de señales 58

2.5.3. Wavelets de Daubechies 60

2.5.4. Reducción de ruido 62

3. Conclusiones **65**

3.1. Estancia en prácticas 65

3.2. Memoria TFG 66

A. Algunos programas de la Estancia en Prácticas **71**

A.1. generateData.py 71

A.2. paths.js 75

A.3. JSONToCSVConvertor.js 75

A.4. saveJSON.js 77

B. Programas de Matlab sobre la Memoria del TFG **79**

B.1. funcionG.m 79

B.2. obtenerValoresFuncionG.m 79

B.3. analisisSenyal.m 79

B.4. mapaSignificancia.m 81

B.5. energia.m 81

B.6. energiaAcumulada.m 81

B.7. rms.m 81

B.8. comprimir.m 82

B.9. analisisSenyalDb.m 82

B.10. hacerRuido.m 84

B.11. eliminarRuido.m 84

Capítulo 1

Estancia en Prácticas

1.1. Introducción

En este capítulo se desarrolla la memoria técnica de mi estancia en prácticas de la asignatura MT1030 (Prácticas externas y Proyecto de Final de Grado) que he realizado en la empresa **Soluciones Cuatroochenta S.L.**[13], ubicada en el edificio Espaitec 2 de la Universitat Jaume I (Castellón). La persona encargada de supervisar mi estancia fue Sergio Aguado, CTO de la empresa.

En primer lugar, se dará una visión general de la empresa. Después, se detallará mi objetivo a realizar durante la estancia en prácticas. Posteriormente, se analizarán el software y las herramientas que se han necesitado durante estas semanas. Y, finalmente, se explicarán con detalle las labores y tareas que he llevado a cabo durante estos días.

1.2. La empresa

Soluciones Cuatroochenta S.L. es una empresa especializada en el desarrollo de aplicaciones para smartphones y tablets. Para ello, cuenta con profesionales del campo de la programación, desarrollo, diseño gráfico, marketing y comunicación. Actualmente cuenta con varios proyectos

en marcha (tanto a nivel nacional como internacional), más de 60 aplicaciones ya lanzadas al mercado y otras más en proceso de desarrollo. Algunas de sus aplicaciones ya lanzadas son: una app para Consum, Legends, CTN - Adidas para latinoamérica y una app para el Villarreal C.F., entre muchas otras.

También cuenta con aplicaciones propias como *Moviltik*, una app para pagar el parking de la zona azul u otras de estacionamiento regulado (ORA) desde el móvil, o *Sefici*, una aplicación que permite reportar y gestionar incidencias para uso empresarial.

El nombre de la empresa proviene de la resolución de pantalla del primer iPhone, *480 x 320*, que abrió una nueva oportunidad de mercado para difundir contenido. Más tarde, y con la llegada de Android también al mercado, emergen nuevas oportunidades para las empresas de llegar a más clientes.

1.3. Objetivos de la estancia en prácticas

El objetivo de mi estancia en prácticas es crear una base de datos no relacional, utilizando la tecnología de Apache Cassandra. Este tipo de bases de datos nos permite almacenar una gran cantidad de información en ella sin perjudicar el rendimiento de las consultas.

Una vez creada, extraeré y analizaré la información que se encuentre almacenada para poder obtener conclusiones y estadísticos (a través de consultas). Estos datos se los pasaremos a una compañera que se unirá más tarde al proyecto; a través de unos *web services* que crearé con la ayuda de Sergio Aguado, utilizando la tecnología *Node.JS*, que utiliza el lenguaje *JavaScript*. Ella será la encargada de crear el *front-end* de la aplicación.

Finalmente, también estudiaré la representación gráfica de los datos con *D3.JS* con el fin de mostrar estos datos a través de diagrama de barras, mapas y otros tipos de gráficos que se pueden crear con esta tecnología y así poder colaborar también en la parte de front-end.

1.4. Software y herramientas utilizadas

1.4.1. Apache Cassandra [2] & DataStax DevCenter [5]

Apache Cassandra permite crear una base de datos no relacional, la cual es muy útil cuando se necesita escalabilidad, es decir, almacenar una gran cantidad de datos, y alta disponibilidad. Esto último se consigue duplicando los datos en distintos *clusters* por lo que si uno dejase de funcionar en algún momento se puede seguir teniendo acceso a los datos sin comprometerlos.

El lenguaje propio de esta base de datos es **CQL** (*Cassandra Query Language*). En resumidas cuentas se trata de un lenguaje de consultas similar a **SQL** pero con la diferencia de que no se permiten hacer *joins*, que es lo que caracteriza a las bases de datos relacionales.

Una herramienta de desarrollo útil que permite crear y ejecutar CQL es **DataStax DevCenter**. Se pueden crear keyspaces, tablas y ejecutar consultas y verlo de una forma interactiva en lugar de hacerlo a través de un terminal o consola.

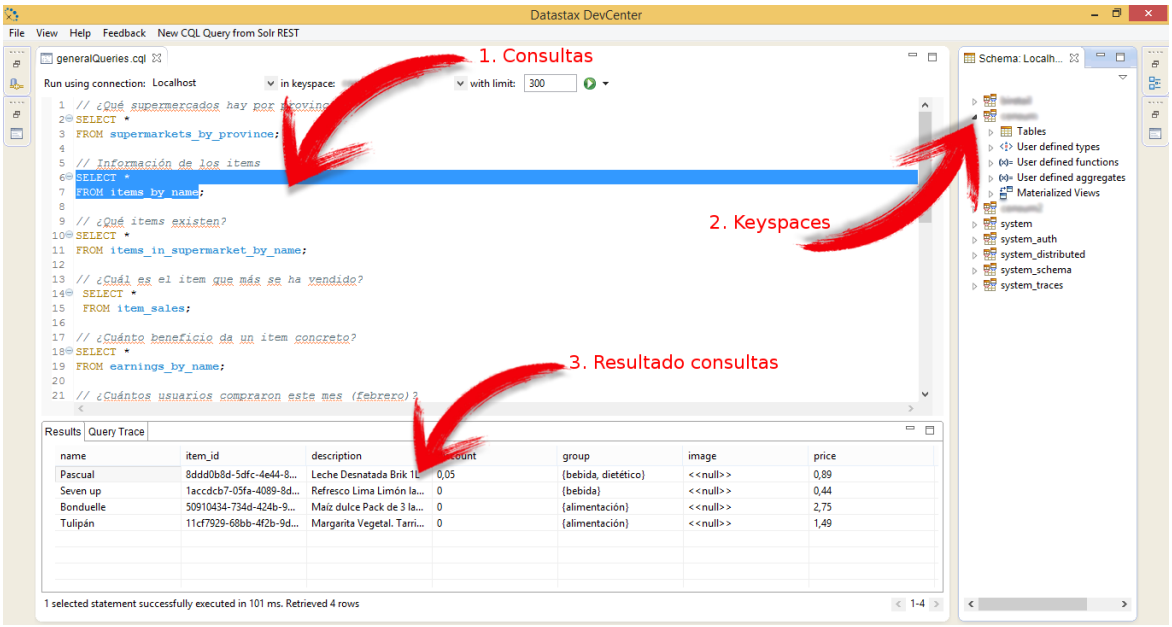

Figura 1.1: Captura de pantalla de DataStax DevCenter ejecutando una consulta.

En la Figura 1.1, se observa que en (1) hay un espacio para escribir el código CQL para realizar consultas, crear tablas, etc. En (2) se ven los keyspaces disponibles y dentro de cada uno existen las tablas. Y en (3) se visualiza el resultado de la consulta. También existe una segunda pestaña donde se puede ver en detalle cómo se obtienen los datos a devolver.

1.4.2. Apache Spark [3]

Apache Spark es un framework de computación en cluster de código abierto. Se encuentra orientado a la velocidad y proporciona APIs para poder usarlo en distintos lenguajes de programación como **Java**, **Python**, **R** y **Scala**.

Se puede conectar con Cassandra a través de un nodo trabajador y el driver y derivar toda la carga de trabajo fuera de la base de datos. Esto permitiría aligerar los cálculos y no sobrecargar la base de datos.

1.4.3. Node.JS [11]

Node.JS es una herramienta usada para construir aplicaciones web como un evento asíncrono basado en **JavaScript** (*JS*). Utiliza entrada/salida no bloqueante y es ligero y eficiente. Su principal uso está en el diseño de aplicaciones de redes. Por ejemplo, para crear un servidor web.

1.4.4. Git [6] & Sourcetree [14]

Git es una herramienta utilizada para realizar control de versiones. En particular y a diferencia de otras herramientas, Git permite trabajar offline y hacer copias de seguridad locales y, cuando se tenga acceso a Internet, actualizar las copias en el servidor. De esta manera, se puede trabajar de forma asíncrona y luego sincronizarse si fuese necesario.

Sourcetree es una interfaz gráfica, **GUI** (*Graphical User Interface*), que permite interaccionar con Git sin la necesidad de utilizar líneas de comandos.

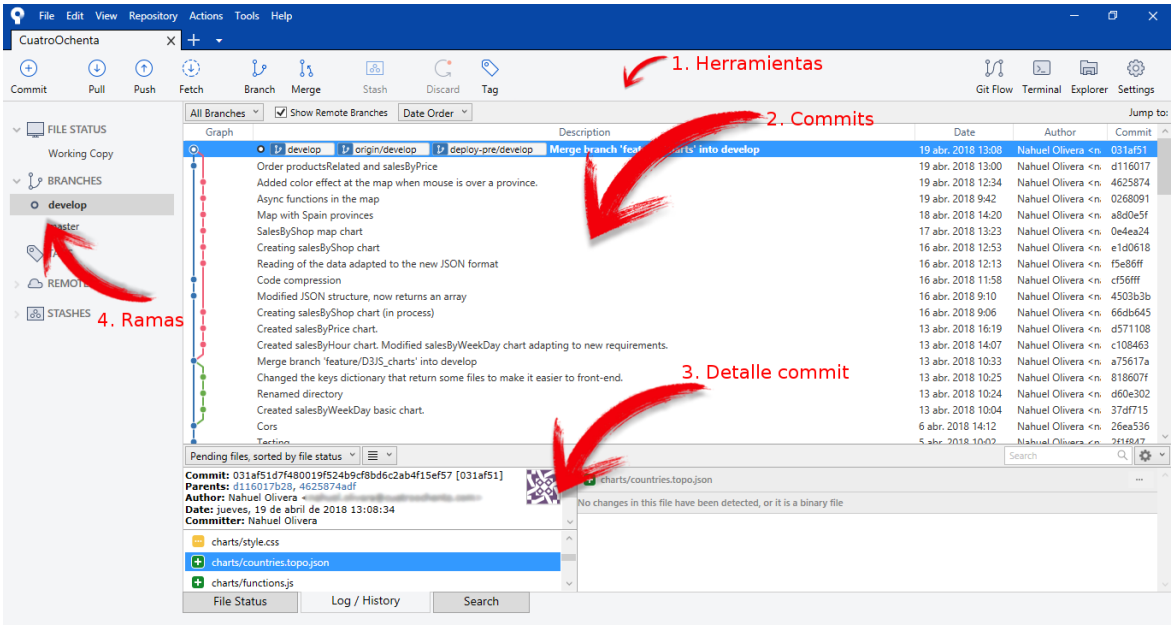

Figura 1.2: Captura de pantalla de SourceTree donde se ve el árbol gráfico de commits hechos.

En la Figura 1.2 se aprecian cuatro partes diferenciadas. En (1) se encuentran las herramientas para hacer *commits*, subir los cambios al servidor, bajarse los commits hechos por otros desarrolladores, crear ramas, hacer un *merge* (juntar) entre dos ramas existentes, etc. En la parte central, (2) se ve de forma gráfica la estructura del árbol y de los commits que se hacen. Los detalles de cada commit se pueden ver en la parte inferior (3); y haciendo click en cada uno de los archivos, se detalla qué línea se modificó. Finalmente, en el lado izquierdo (4) se pueden ver cuántas ramas existen en ese proyecto.

1.4.5. JetBrains: WebStorm [7]

Webstorm, de JetBrains, es un editor de programación, **IDE** (*Integrated Development Environment*), muy útil cuando se programa en JavaScript o enfocado a la programación web en general. También trae consigo un control de versiones que se puede sincronizar con el repositorio Git, además de otras características que facilitan la tarea de programación pues te sugiere autocompletado de código y te informa de posibles errores o erratas en tiempo real.

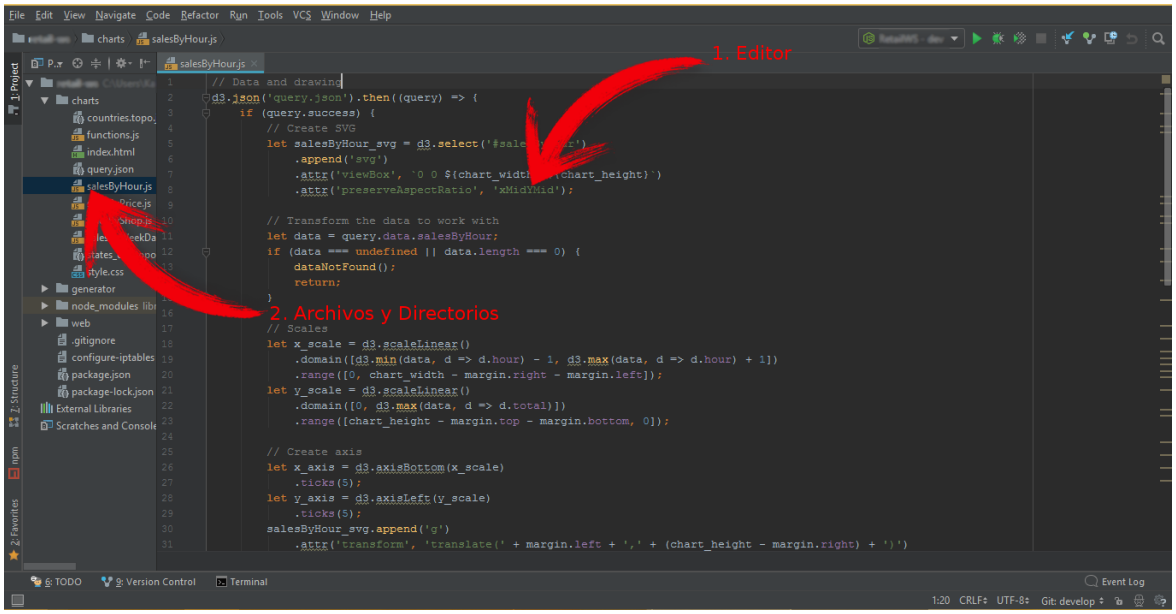

Figura 1.3: Captura de pantalla del editor WebStorm.

En la Figura 1.3 se observa el editor de programación con una temática oscura. Se pueden modificar los colores al gusto de cada desarrollador y ofrece un ventana de edición (1) y también los directorios y carpetas donde se encuentra el proyecto (2) de manera que no tienes que estar navegando por tu ordenador y arrastrando los ficheros uno a uno. Además, como se puede sincronizar con Git, muestra en colores los ficheros que fueron modificados y no se ha hecho commit sobre ellos. También diferencia sobre los que sí se guardaron los cambios pero no se hizo un *push*, es decir, los commits que no se subieron al servidor remoto.

1.4.6. Postman [12]

Postman es un entorno de desarrollo de APIs, **ADE** (*API Development Environment*) que, entre otras funciones, permite hacer *debug* de tu código, es decir, permite testear las APIs, examinar las respuestas y enviar peticiones.

En la Figura 1.4 se observa el programa Postman donde se realiza una petición web a una ruta específica (1) con unos parámetros de entrada (2). Estos datos conectan con el servidor en cuestión y si la entrada es correcta, se devuelve una salida (3) con un formato que especifica el

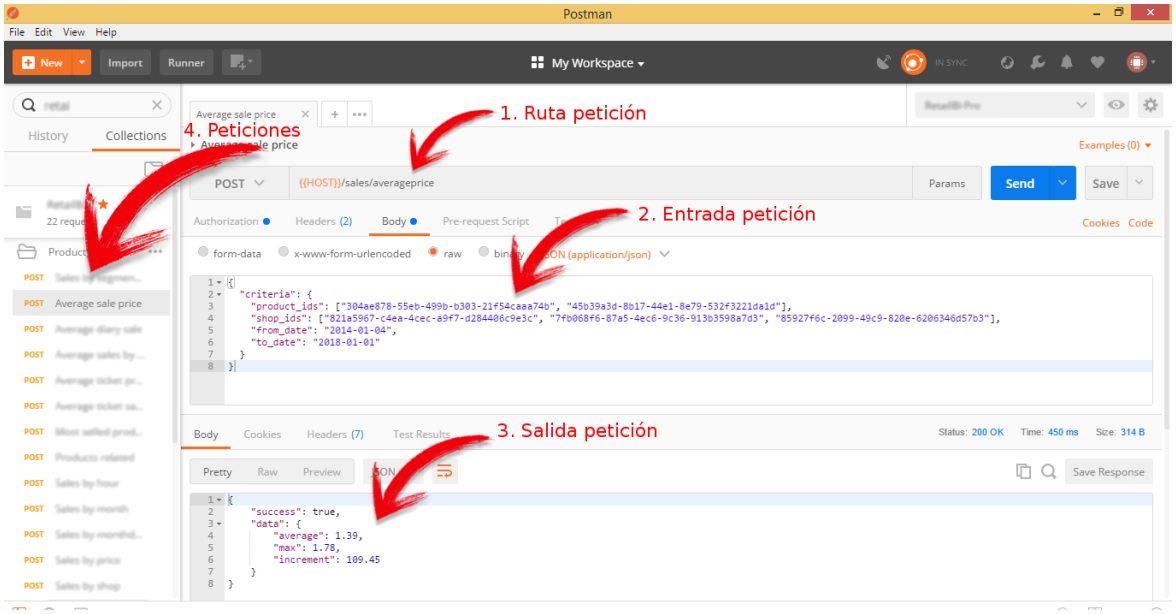

Figura 1.4: Captura de pantalla de Postman con entrada y salida de una petición web.

desarrollador del web service. Además, se pueden guardar las peticiones en un listado (4) para no tener que escribirlas siempre y poder hacer pruebas, por ejemplo, más rápidamente.

1.4.7. D3.JS [4]

D3.JS es una librería de JavaScript que permite la visualización de datos en los navegadores web. Para ello utiliza **SVGs**, **HTML5** y **CSS** y, además, puede hacerlo de forma dinámica e interactiva. Con esta librería, por ejemplo, se pueden crear mapas y gráficos.

1.5. Planificación temporal de las tareas

1.5.1. Primer Informe

Durante el primer día, tanto los compañeros que han entrado en ese momento a hacer prácticas (distintas a la mía pero en la misma compañía) como a mi, se nos presenta al resto de trabajadores de la empresa. Además, se nos proporciona un usuario (como a los demás trabajadores) para poder acceder a servicios internos de la compañía. Se nos detalla también el funcionamiento general de la empresa y se nos hace una visita guiada por las instalaciones.

Se me explican los objetivos a realizar los primeros días en la empresa: aprender qué es **Apache Cassandra**, cómo funciona y su lenguaje de programación. Una vez adquiridos los conocimientos básicos sobre esta nueva base de datos no relacional, continuaré con la formación en la misma página web, Datastax[5], sobre otro lenguaje de programación que me permita hacer consultas más complejas (pues Cassandra es útil para almacenar muchos datos pero no para consultas) fuera de Cassandra. Este lenguaje es *Scala* que utiliza el framework **Apache Spark**. Durante las lecciones teóricas que se hacen en el curso, también fui haciendo una serie de ejercicios y retos que se proponen para poner en práctica lo explicado.

Finalmente, se decidió que aprendiese también **Node.JS** que utiliza JavaScript enfocado a los servidores web y así poder ejecutar las consultas en un servidor externo y poder derivar toda la carga computacional a un tercer equipo y no a los propios, personales o de la empresa. En esta ocasión, seguí un curso de Udemy Academy[15] en el que también se proponen retos y ejercicios que fui haciendo conforme se pedían.

1.5.2. Segundo Informe

Durante los siguientes días, mi tarea fue la de crear una base de datos nueva con datos inventados pero que simulasen situaciones reales. Para ello, creé una nueva base de datos de un supermercado cualquiera. Una compañera me facilitó un archivo CSV con datos de unos productos (nombre, marca, precio, etc.) y tiendas reales (ubicación, si tenía parking, si se podía comprar por Internet, etc.). Además, busqué en Internet una pequeña base de datos con nombres y apellidos de personas para poder simular a los clientes. En Mockaroo [10] generé tickets de

compras, con fechas, horas, cantidades, descuentos, etc. siguiendo una distribución normal (pues es la que, a groso modo y en tamaños muy grandes, supone una buena aproximación de los datos reales).

Para hacer esta simulación de los tickets, inserté un peso a cada producto manualmente, fila por fila; dándole mayor valor a aquellos que consideré que la gente compraría más porque eran productos más cotidianos o básicos. Y, también de forma manual, aunque con un poco más de aleatoriedad, también le asigné un peso a los supermercados. En este caso, me basé en su tamaño, si tenían o no parking, el pueblo donde se encontraban, etc. De esta forma, algunos datos tendrían más probabilidad de aparecer repetidos que otros. Así podíamos simular una situación más o menos real y, cuando se analizasen los datos, dieran conclusiones lógicas.

Una vez creadas estas tablas, las guardé todas en un archivo CSV, cada uno independiente, que posteriormente importaría a la base de datos. Para ello, creé un nuevo espacio de trabajo con estas nuevas tablas (ya no trabajaría con el mismo keyspace que había hecho inicialmente para testear). La pecularidad de Cassandra es que no es una base de datos relacional. Por ello, tuve que crear una nueva tabla para cada consulta que quería hacer a partir de las tablas básicas (supermercados, clientes, tickets, líneas de tickets y productos).

Con el objetivo de crear estas tablas combinadas, una de la otra, creé un programa en Python (se puede ver en el Anexo A.1) con distintas funciones, cada una según el objetivo que buscaba. Pero, todas leían de un CSV y lo convertían de nuevo a un CSV. Más tarde, Sergio me sugirió hacerlo con NodeJS (algunas de las funciones utilizadas se pueden encontrar en el Anexo A.2 y A.4) y continué creando los CSVs con este lenguaje de programación. En este lenguaje tuve que manejar los datos con archivos JSON. Por ello, creé una función que transformaba los datos de CSV a este formato y, a la vez que lo hacía, los ajustaba de forma que los combinaba como me interesaban; ya que Cassandra no permite hacer "joins" entre sus tablas. Posteriormente, hice la transformación inversa con otra función que también creé (de JSON a CSV), para volver a importar los datos a Cassandra. La función que realizaba tal tarea se puede encontrar en el Anexo A.3.

Por ejemplo, de las tablas iniciales de items y línea de tickets creé dos nuevas tablas derivadas: una que fuese la cantidad de items que se vendieron en un periodo concreto de tiempo (así si se quisiera saber cuántos artículos se vendieron en Navidad lo buscaríamos en esta tabla) y otra que indicase la cantidad de items que se vendieron en un supermercado concreto (para saber

cuántas ventas produce una tienda en particular). Pero también creé otra tabla que solo contenía la cantidad de ventas que se hicieron de un producto concreto en todos los supermercados y desde cualquier fecha.

Una vez consideré que con los CSVs que tenía podía cubrir perfectamente las consultas que me pedían y sacar conclusiones de esos datos, me propuse exportarlos a la base de datos que había hecho, en local, creando nuevas tablas y comprobando que funcionaban bien las consultas.

Una vez acabada la parte más esencial para trabajar, me dispuse a investigar cómo conectarme a Cassandra de forma remota, pues hasta entonces trabajé en local (en mi ordenador) y, para trabajar con Sergio, necesitábamos tener ambos acceso a la misma base de datos con el fin de poder modificarla simultáneamente, acceder a los mismos datos, etc.

Logrado el objetivo, se me sugirió leerme la documentación sobre **Git** que es el programa que se utiliza en la empresa para hacer control de versiones y poder trabajar todos juntos. En ese momento, una nueva compañera se unió al equipo para encargarse de la parte de front-end del proyecto. Mi tarea ahora consistía en preparar los datos para que la compañera, al hacer una petición en un servidor web, pudiese leer los datos que yo había volcado en ella.

Para evitar trabajar con línea de comandos directamente sobre Git, me sugirieron instalarme **Sourcetree**, que implementaba una interfaz gráfica para ello y me explicaron cómo funcionaba. Además, instalé **WebStorm** para trabajar como entorno de programación y editor que también viene con un control de versiones. Y, finalmente, también instalé y se me explicó cómo funciona **Postman** para hacer pruebas de las peticiones web y poder verificar que los programas que iba a hacer funcionasen correctamente.

Llegados a este punto, subí lo que había hecho los últimos días en la base de datos a nuestro repositorio en Git para que todos los miembros del equipo tuviésemos los mismos datos y pudiésemos trabajar sincronizadamente.

1.5.3. Tercer Informe

Una vez llegamos al punto en que subí todo lo que tenía al repositorio y Sergio supervisó lo que había creado, nos dimos cuenta de que algunas tablas no respondían exactamente a las consultas que se nos pedían. Entonces, tuvimos que generar nuevas tablas, pues en Cassandra cada consulta requiere de una tabla para ello. Por ejemplo, en lugar de hacer una tabla combinada en la que se mostrasen los productos vendidos cada día de la semana y en un periodo de tiempo y supermercado concreto, había considerado la situación más general, es decir, desde que existen registros y sin diferenciar entre supermercados. Este tipo de detalles fueron los que se tuvieron que modificar. Se puede observar en la Figura 1.5 cómo se mejora una tabla antigua en otras dos con más detalles y que respondiese mejor a las consultas.

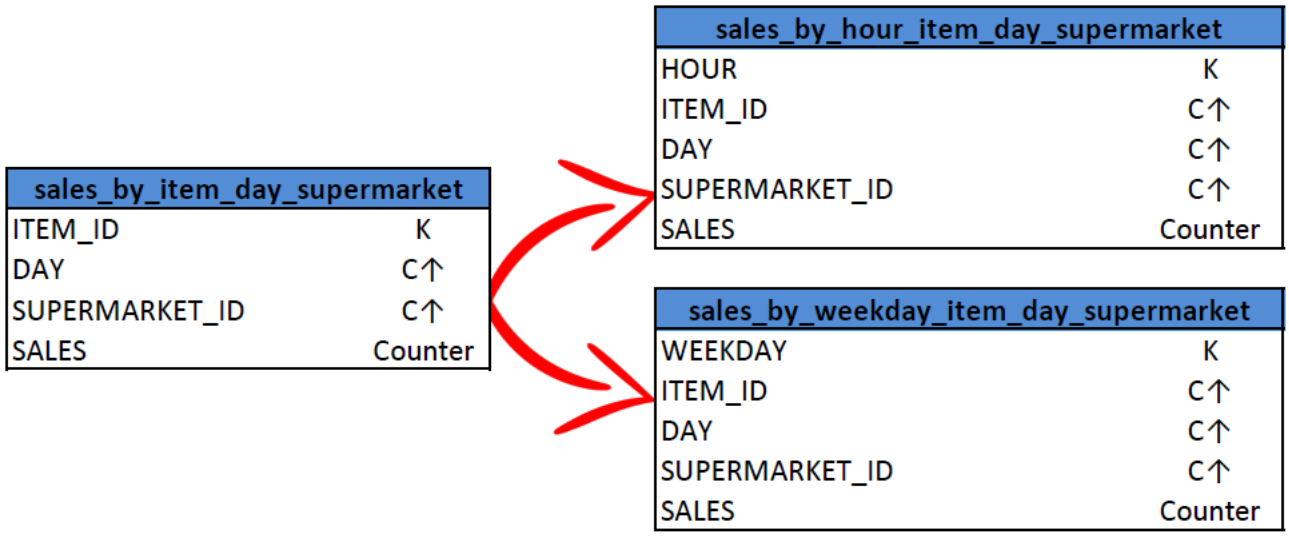

Figura 1.5: Esquema de cómo una tabla se deriva en otras dos con más detalles.

A la vez, también fuimos creando los programas para recibir peticiones web, web services, y devolver los datos que nos pedían.

Una vez creadas las nuevas tablas corregidas, subí todo el código y tablas al repositorio común que teníamos los miembros del equipo para poder trabajar con los mismos datos. Después, continué creando nuevas tablas para seguir respondiendo a las consultas que se nos pedían y aún nos quedaba por cubrir.

Cuando testeé por primera vez la base de datos remota me di cuenta de que la versión de Cassandra era anterior a la que yo trabajaba en local (y me daba algunos errores con algunas importaciones) por lo que tuvimos que actualizarla. Una vez subidos los datos surgieron un par de errores menores en la configuración de los ficheros del servidor (como no tener bien el nombre del keyspace o apuntar a una dirección errónea de la base de datos remota) que solucionamos cambiando unos parámetros.

Durante los siguientes días, generé más datos aleatorios para tener más datos con los que tratar. De esta forma, pasamos de tener casi 3.000 líneas de tickets a más de 13.500.

En esta ocasión la generación de datos fue mucho más rápida pues tenía ya creadas las funciones que me permitían obtener los datos en el formato que yo quería; además de saber qué datos me interesaban para las tablas ya creadas. De forma simultánea aproveché para optimizar y limpiar el código, eliminando funciones que no me harían falta en ningún otro momento y comentando el programa para que cualquiera pueda ejecutarlo en un futuro y entenderlo sin ninguna dificultad.

Una vez creadas las nuevas tablas con los nuevos datos y habiendo testeado el código en local, subí la nueva versión al remoto. En esta ocasión, también subí, a otro servidor distinto a aquel en el que se encontraba la base de datos, los web services para hacer las peticiones y obtener las consultas.

Una vez acabado mi principal objetivo de simular datos aleatorios y coherentes para poder hacer consultas en Cassandra, preparar las tablas para ello y también las peticiones web para que mi compañera pudiera usarlas y leer de la base de datos para luego tratar con los datos, se me propuso formarme en **D3.JS**. El objetivo de formarme en D3.JS era poder ayudar a mi compañera a mostrar una visualización de los datos (con gráficas). Esta librería de JavaScript permite crear gráficos a partir de datos almacenados en texto plano, CSV o JSON. Precisamente, los datos que devuelven los web services que creé son JSON.

La formación, igual que la anterior de Node.JS, la realicé a través de un curso online de Udemy Academy en la que se proporcionan también ejercicios prácticos.

1.5.4. Cuarto Informe

Durante los últimos días de mi estancia en prácticas en Cuatroochenta, preparé gráficos para la parte front-end del proyecto con D3.JS. Para ello, simulé la entrada de datos que recibirían desde la parte de cliente a través de unas peticiones web que deberían hacer. De esta forma, no tenía que preocuparme del buen funcionamiento de las comunicaciones entre servidor y cliente pues de ello se encargaba mi compañera, y podía centrarme en el correcto funcionamiento del código.

Así, si había que depurar algún error, encontrar algún fallo o testear, resultaba mucho más fácil pues solo dependía de mi código y no de otros.

Las gráficas que hice fueron dos diagramas de barras, uno de líneas y un mapa interactivo. En los primeros representaba en el eje de abscisas las horas o días predefinidos y en el eje de ordenadas la cantidad de ventas de un determinado producto. Sin embargo, con este gráfico era difícil saber el precio de venta de los productos (y su visualización) pues podía variar en el tiempo. Por ello, utilicé uno de líneas para este caso. Finalmente, en el mapa representé las ciudades que han vendido un determinado producto y la cantidad de veces que lo vendieron con un sistema de coordenadas real (gracias a los datos que me pasaron cuando empecé la estancia). Ejemplos de estos gráficos se pueden ver en la Figura 1.6 donde se muestra un dato concreto al pasar el ratón por encima de uno de los puntos.

Finalmente, durante las últimas horas de la estancia, volví a repasar todos los web services y limpié el código, eliminando líneas repetidas y comprimiéndolo más pero sin dificultar su lectura. También hice alguna pequeña modificación y agregué comentarios para facilitar la lectura y el trabajo de mi compañera.

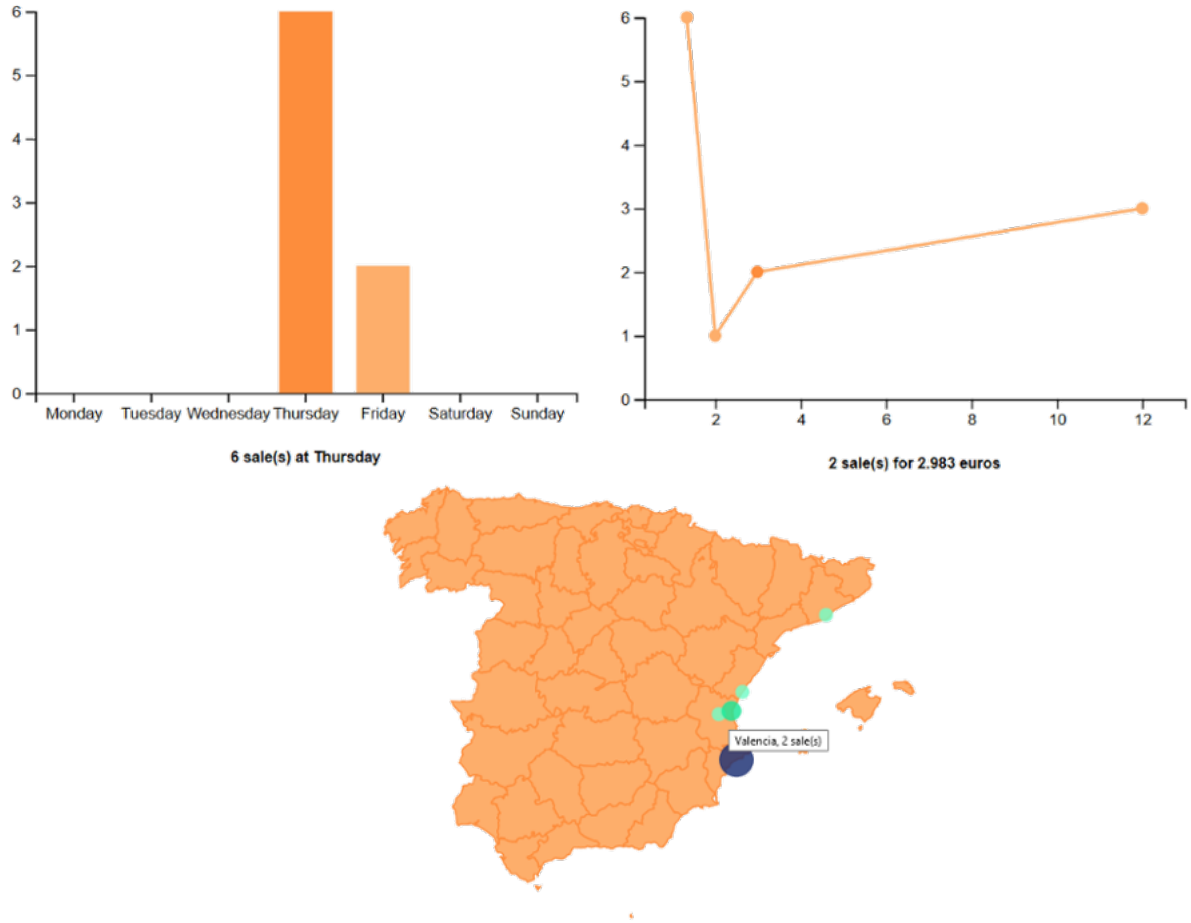

Figura 1.6: Ejemplos de los diseños hechos con D3.JS. A la izquierda un diagrama de barras, a la derecha un gráfico de líneas y abajo un mapa de España.

1.6. Grado de consecución de los objetivos propuestos

El objetivo principal de mi estancia en prácticas era crear una base de datos no relacional utilizando Cassandra para almacenar una gran cantidad de datos en ella. Una vez logrado, tenía que preparar una serie de web services para que la parte encargada de dar servicio al cliente pudiese recibir los datos, representarlos y poder sacar conclusiones a partir de ellos.

La primera parte la conseguí en un lapso de tiempo relativamente corto si se tiene en cuenta la dificultad que suponía, mientras que la segunda me llevó más tiempo pues tuve que rehacer cambios y estar pendiente de que la base de datos devolviese en las consultas los datos que, precisamente, necesitaba y quería.

Además de haber cumplido con los objetivos propuestos, también logré ayudar en la parte del cliente pues me sobraron unas semanas para completar las horas que la asignatura requería.

Toda la tecnología que he utilizado durante estos días no la había visto en la carrera por lo que me tocó aprenderla de cero pero fue una experiencia agradable. No solo he aprendido nuevos lenguajes de programación sino también nuevas metodologías de trabajo; y esta experiencia me ha servido para ver cómo funciona una empresa de este sector desde dentro.

Capítulo 2

Memoria TFG

Hoy en día, el análisis de señales se aplica a muchos campos como la medicina (por ejemplo en el tratamiento de imágenes de resonancias magnéticas), ingeniería eléctrica, música, predicción de terremotos e incluso para resolver ecuaciones en derivadas parciales, entre otras muchas aplicaciones. Las wavelets, inventadas por J. Morlet y A. Grossmann, proporcionan un conjunto de herramientas flexibles para ello.

La transformada wavelet es muy eficiente para extraer información de señales no periódicas. Además, existe una amplia familia de wavelets que permite tratar distintos tipos de señales según lo que convenga en cada situación.

El tratamiento de señales e imágenes, la reducción del ruido y la compresión de señales (por ejemplo, en electrocardiogramas, vibraciones de motores, etc.) son problemas que se pueden afrontar bien con las wavelets pues presentan una buena localización en tiempo y frecuencia; a diferencia de la transformada de Fourier que tiene una muy buena localización en frecuencia pero muy pobre en tiempo. [19]

La ventaja de las wavelets frente a la transformada de Fourier es su soporte compacto, es decir, se encuentra definida en un intervalo finito. No podemos aplicar la transformada de Fourier en un intervalo finito de una señal continua pues la función no será derivable en todos los puntos, pero sí se podrán aplicar wavelets. Además, la transformada de Fourier requiere de un mayor número de operaciones. Veremos como las wavelets son más eficientes también en este campo.

Nos centraremos en las wavelets discretas y unidimensionales relacionadas con el análisis de multirresolución. Cuando la familia de wavelets es ortogonal, la computación es más rápida pues permite la transformada inversa de forma tan rápida como la directa.

La transformada de wavelets discreta divide la señal en dos. Por una parte, la tendencia que es una copia de la señal original pero con menor resolución. Y, por otro lado, las fluctuaciones que guardan información sobre la diferencia entre la señal modificada y la original. De esta forma, se elimina ruido, el cual se comporta siguiendo un modelo de probabilidad (cada uno puede ser distinto).

Las wavelets continuas, por su parte, requieren de un elevado número de integrales. Sin embargo, permiten adaptarse a situaciones donde la discreta no da un buen resultado. Este tipo de wavelets no serán objeto de estudio en este trabajo.

2.1. Conceptos previos

En esta primera sección se introducirán algunos conceptos básicos de álgebra lineal que serán necesarios para el desarrollo de nuestro trabajo.

Definición 1. Sea f una señal discreta, es decir, $f = (f_1, f_2, ..., f_N) \in \mathbb{R}^N$ con $N = 2^k$, donde k es lo suficientemente grande.

- Definimos la **energía** de una señal f como $E(f) := f_1^2 + f_2^2 + \cdots + f_N^2$.
- El **soporte** de una señal f se denota por $sop(f) := \{m : f_m \neq 0\}$.
- El **valor medio** se define como $\frac{f_1+f_2+\cdots+f_N}{|sop(f)|}$, donde $|sop(f)|$ es el cardinal del conjunto soporte.

A continuación, vamos a definir dos nuevas señales, aunque tomamos las definiciones como las del "primer nivel". Es necesario aclarar que existen más niveles como se verá más adelante.

Definición 2. La señal de **promedio de Haar a primer nivel** se puede definir como la nueva señal, A^1, que se obtiene al promediar los elementos de la señal f dos a dos y sustituyendo los valores que se promedian por sus respectivos promedios.

Por ejemplo, si $f = (2, 5, 7, 3)$, el promedio de Haar a primer nivel es la señal $A^1 = (3.5, 3.5, 5, 5)$. Es fácil ver que esta nueva señal es de *menor resolución* que la original.

Definición 3. La señal de **detalle de Haar a primer nivel** podemos definirla de la siguiente forma: $D^1 := f - A^1$, es decir, es la diferencia entre la señal original y el valor promedio calculado anteriormente.

Siguiendo el ejemplo anterior, la señal de detalle de Haar de primer nivel es $D^1 = (-1.5, 1.5, 2, -2)$.

Una vez obtenidas estas dos nuevas señales, el siguiente paso consistiría en promediar los cuatro elementos de f o bien los de A^1. De esta forma, obtendríamos el promedio de Haar a nivel 2 que es una señal con menor resolución que la de nivel 1: $A^2 = (4.25, 4.25, 4.25, 4.25)$.
En este caso, la señal en detalle de Haar a segundo nivel es: $D^2 = A^1 - A^2 = (-0.75, -0.75, 0.75, 0.75)$.

Si quisiéramos volver a obtener la señal original podemos obtenerla de la siguiente forma: $f = A^2 + D^2 + D^1$.

Llegados a este punto, recordemos ahora algunas nociones básicas de álgebra lineal.

Definición 4. Sea $B = \{v_1, v_2, \ldots, v_m\} \subseteq \mathbb{R}^N$. Los vectores, v_i se dicen que son **linealmente independientes** si existen $a_i \in \mathbb{R}$ tales que $a_1v_1 + a_2v_2 + \cdots + a_mv_m = 0$ implica que $a_i = 0$ para todo $1 \leq i \leq m$. En caso contrario, se dirá que son **linealmente dependientes**.

Definición 5. Sea V un espacio vectorial y sea $B = \{v_1, v_2, \ldots, v_m\} \subseteq \mathbb{R}^N$. Se dice que B es un **sistema generador** del espacio vectorial V si cualquier vector de V puede expresarse como combinación lineal de vectores de B. Un sistema de generadores de B linealmente independiente se llama **base** del espacio vectorial V.

Con estos conceptos básicos podemos introducir las siguientes definiciones:

Definición 6. Sea $B = \{v_1, v_2, \ldots, v_m\} \subseteq \mathbb{R}^N$. Dado $v = a_1v_1 + a_2v_2 + \cdots + a_mv_m$, los números $a_i, 1 \leq i \leq m$, se llaman **coordenadas del vector** v respecto a la base B.

Definición 7. Sea $B = \{v_1, v_2, \ldots, v_m\} \subseteq \mathbb{R}^N$, se llama *envoltura lineal* o **conjunto generado por un sistema de vectores** de B al subespacio vectorial V formado por las combinaciones lineales de los elementos de B y se denota por: $V := lin\{v_1, v_2, \ldots, v_m\}$.

Definición 8. El **producto escalar** de dos vectores $u = (u_1, \ldots, u_N)$, $w = (w_1, \ldots, w_N) \in \mathbb{R}^N$ se define como: $u \cdot w := u_1w_1 + u_2w_2 + \cdots + u_Nw_N$.

Esta última operación nos permite definir otros conceptos:

Definición 9. Dos vectores u y w son **ortogonales** entre sí cuando su producto escalar es nulo y se denotará como $u \perp w$; es decir, $u \perp w$ si y solo si $u \cdot w = 0$.

Definición 10. Sea $u \in \mathbb{R}^N$ un vector. Definimos su **norma euclídea** como la raíz cuadrada de su energía, es decir, $\| u \| := \sqrt{u \cdot u} = \sqrt{u_1^2 + u_2^2 + \cdots + u_N^2}$.

Definición 11. Un sistema de vectores $B = \{v_1, v_2, \ldots, v_m\} \subseteq \mathbb{R}^N$ es **ortonormal** si cada vector del sistema tiene norma 1 y son ortogonales dos a dos.

Por último, introducimos algunas definiciones más que serán necesarias para el desarrollo de nuestro trabajo.

Definición 12. Sean V y W dos subespacios vectoriales. Se dicen que son **ortogonales entre sí** y se denota como $V \perp W$, si $v \cdot w = 0$ para todo $v \in V$ y para todo $w \in W$. En este caso, se denota el espacio suma de V y W como $V \oplus^{\perp} W$.

Definición 13. Sean $v \in \mathbb{R}^N$ y $W \subseteq \mathbb{R}^N$. Se define la **proyección ortogonal** de v sobre W como el único $w \in W$ tal que $(v - w) \perp W$.

Definición 14. Sean V y W dos espacios vectoriales sobre el mismo cuerpo K. Una **aplicación lineal** es una aplicación $T : V \to W$ si para todo par de vectores $v, w \in V$ y para todo escalar $\lambda \in K$ se satisface que:

- $T(v + w) = T(v) + T(w)$
- $T(\lambda v) = \lambda T(v)$

Definición 15. Sea $T : \mathbb{R}^N \to \mathbb{R}^N$ una aplicación lineal. Se dice que T es una **transformación ortogonal** si conserva el producto escalar, es decir, si $Tv \cdot Tw = v \cdot w$ para todo par $v, w \in \mathbb{R}^N$.

Esto equivale a que sea una isometría[1], es decir, conserva la norma ($\| Tv \| = \| v \|$, para todo $v \in \mathbb{R}^N$). En particular una transformación ortogonal preserva la energía, al ser el cuadrado de la norma. T es transformación ortogonal si y solo si su expresión matricial en una (cualquier) base ortonormal de $\mathbb{R}^N$ es tal que sus filas (columnas) son una base ortonormal.

2.2. Wavelets de Haar

El tipo de wavelets que vamos a introducir se caracterizan por su sencillez. Fueron definidas por Alfred Haar en 1910 ([21]) pero no fueron reconocidas como funciones de ondas hasta la definición de las wavelets de Daubechies (que se verá más adelante).

[1] Una isometría es una aplicación entre dos espacios métricos que conserva las distancias entre los puntos.[17]

2.2.1. Scaling y wavelets

Definición 16. Sea N la dimensión del espacio donde nos enontramos, entonces, el número de **wavelets de Haar de primer nivel** es $N/2$ y se definen como:

$$\begin{aligned} w_1^1 &:= (\frac{1}{\sqrt{2}}, \frac{-1}{\sqrt{2}}, 0, \ldots, 0) \\ w_2^1 &:= (0, 0, \frac{1}{\sqrt{2}}, \frac{-1}{\sqrt{2}}, 0, \ldots, 0) \\ &\vdots \\ w_{N/2}^1 &:= (0, \ldots, 0, \frac{1}{\sqrt{2}}, \frac{-1}{\sqrt{2}}). \end{aligned}$$

Podemos observar que cada w_j^1, con $j > 1$, se obtiene desplazando $2j$ posiciones las entradas no nulas de la wavelet inicial, w_1^1.

Además, este conjunto de vectores forman un sistema ortonormal. Para verlo, comprobemos primero que son ortogonales dos a dos.

$$\begin{aligned} w_1^1 \cdot w_2^1 &= (\frac{1}{\sqrt{2}}, \frac{-1}{\sqrt{2}}, 0, \ldots, 0) \cdot (0, 0, \frac{1}{\sqrt{2}}, \frac{-1}{\sqrt{2}}, 0, \ldots, 0) = \frac{1}{\sqrt{2}} \cdot 0 + \frac{-1}{\sqrt{2}} \cdot 0 + \cdots + 0 = 0 \\ w_1^1 \cdot w_3^1 &= (\frac{1}{\sqrt{2}}, \frac{-1}{\sqrt{2}}, 0, \ldots, 0) \cdot (0, 0, 0, 0, \frac{1}{\sqrt{2}}, \frac{-1}{\sqrt{2}}, 0, \ldots, 0) = \frac{1}{\sqrt{2}} \cdot 0 + \frac{-1}{\sqrt{2}} \cdot 0 + \cdots + 0 = 0 \\ &\vdots \\ w_{N/2-1}^1 \cdot w_{N/2}^1 &= (0, \ldots, 0, \frac{1}{\sqrt{2}}, \frac{-1}{\sqrt{2}}, 0, 0) \cdot (0, \ldots, 0, \frac{1}{\sqrt{2}}, \frac{-1}{\sqrt{2}}) = 0 + \cdots + \frac{1}{\sqrt{2}} \cdot 0 + \frac{-1}{\sqrt{2}} \cdot 0 = 0. \end{aligned}$$

Y ahora veamos que la norma de cada vector es 1:

$$\| w_j^1 \| = \sqrt{(\frac{1}{\sqrt{2}})^2 + (\frac{-1}{\sqrt{2}})^2 + 0^2 + \cdots + 0^2} = \sqrt{\frac{1}{2} + \frac{1}{2}} = \sqrt{1} = 1$$

para $1 \leq j \leq N/2$. Además, el valor medio de cada wavelet es 0. En primer lugar, el cardinal del conjunto soporte de las wavelets de nivel 1 es 2, es decir, $| \, sop(w_j^1) \, | = 2$,
$1 \leq j \leq N/2$. Y, por tanto, el valor medio es:

$$\frac{\frac{1}{\sqrt{2}} + \frac{-1}{\sqrt{2}} + 0 + \cdots + 0}{| \, sop(w_j^1) \, |} = \frac{0}{2} = 0$$

para $1 \leq j \leq N/2$.

Definición 17. Sea $f \in \mathbb{R}^N$ una señal. Entonces d^1 es una subseñal asociada de **primera fluctuación** de Haar (coeficientes de detalle) que se expresa como:

$$d^1 := (d_1, \ldots, d_{N/2}) = (f \cdot w_1^1, \ldots, f \cdot w_{N/2}^1).$$

Es una subseñal con $N/2$ elementos que contiene todos los productos escalares de la señal f con las wavelets del primer nivel. Entonces, para $j = 1$:

$$d_1 = f \cdot w_1^1 = (f_1, f_2, \ldots, f_N) \cdot (\frac{1}{\sqrt{2}}, \frac{-1}{\sqrt{2}}, 0, \ldots, 0) = \frac{f_1 - f_2}{\sqrt{2}}.$$

Y, en general, para cualquier j tal que $1 \leq j \leq N/2$, $d_j = \frac{f_{2j-1} - f_{2j}}{\sqrt{2}}$.

Definición 18. Sea $f \in \mathbb{R}^N$ una señal. Entonces, definimos los **coeficientes de primera tendencia** de f como:

$$a_1 = \frac{f_1 + f_2}{\sqrt{2}},\ a_2 = \frac{f_3 + f_4}{\sqrt{2}},\ \ldots,\ a_{N/2} = \frac{f_{N-1} + f_N}{\sqrt{2}}.$$

Definición 19. Definimos las **scaling de Haar** o funciones escala de nivel 1 a unas funciones simples dadas por:

$$\begin{aligned} v_1^1 &:= (\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, 0, \ldots, 0) \\ v_2^1 &:= (0, 0, \frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, 0, \ldots, 0) \\ &\vdots \\ v_{N/2}^1 &:= (0, \ldots, 0, \frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}). \end{aligned}$$

Con esta definición, podemos expresar los coeficientes de primera tendencia de f como $a_m = f \cdot v_m^1$, $m = 1, \ldots, N/2$.

Definición 20. Se define la **subseñal de primera tendencia** de una señal como $a^1 := (a_1, a_2, \ldots, a_{N/2})$.

Comparando las scaling de Haar con las wavelets observamos las siguientes similitudes:

1. También forman un sistema ortonormal.

   Para ello, primero comprobemos que es ortogonal:

$$\begin{aligned} v_1^1 \cdot v_2^1 &= (\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, 0, \ldots, 0) \cdot (0, 0, \frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, 0, \ldots, 0) = 0 \\ v_1^1 \cdot v_3^1 &= (\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, 0, \ldots, 0) \cdot (0, 0, 0, 0, \frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, 0, \ldots, 0) = 0 \\ &\vdots \\ v_{N/2-1}^1 \cdot v_{N/2}^1 &= (0, \ldots, 0, \frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, 0, 0) \cdot (0, \ldots, 0, \frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}) = 0 \end{aligned}$$

   Además, la norma de cada vector es 1:

$$\| v_m^1 \| = \sqrt{(\frac{1}{\sqrt{2}})^2 + (\frac{1}{\sqrt{2}})^2 + 0^2 + \cdots + 0^2} = \sqrt{\frac{1}{2} + \frac{1}{2}} = \sqrt{1} = 1$$

   para $1 \leq m \leq N/2$. Por tanto, vemos que es ortonormal.

2. Los soportes son iguales, $sop(v_m^1) = sop(w_m^1)$, para cada $m = 1, \ldots, N/2$.

3. Si desplazamos $2m$ posiciones el par de entradas no nulas de la scaling v_1^1 obtenemos los distintos v_m^1.

Sin embargo, se diferencian en el valor medio. El valor medio de las scaling es $1/\sqrt{2}$:

$$\frac{\frac{1}{\sqrt{2}} + \frac{1}{\sqrt{2}} + 0 + \cdots + 0}{\mid sop(v_m^1) \mid} = \frac{2}{2\sqrt{2}} = \frac{1}{\sqrt{2}}$$

para $1 \leq m \leq N/2$.

Observamos que la familia de vectores $\{v_1^1, \ldots, v_{N/2}^1, w_1^1, \ldots, w_{N/2}^1\}$ conforman una *base ortonormal* de $\mathbb{R}^N$ pues $v_m^1 \cdot w_j^1 = 0$ para todo $j, m = 1, \ldots, N/2$.

En el nivel 2, definimos las wavelets de Haar como:

$$\begin{aligned} w_1^2 &:= (\frac{1}{2}, \frac{1}{2}, \frac{-1}{2}, \frac{-1}{2}, 0, \ldots, 0) \\ w_2^2 &:= (0, 0, 0, 0, \frac{1}{2}, \frac{1}{2}, \frac{-1}{2}, \frac{-1}{2}, 0, \ldots, 0) \\ &\vdots \\ w_{N/4}^2 &:= (0, \ldots, 0, \frac{1}{2}, \frac{1}{2}, \frac{-1}{2}, \frac{-1}{2}). \end{aligned}$$

Y las scaling de segundo nivel:

$$\begin{aligned} v_1^2 &:= (\frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, 0, \ldots, 0) \\ v_2^2 &:= (0, 0, 0, 0, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, 0, \ldots, 0) \\ &\vdots \\ v_{N/4}^2 &:= (0, \ldots, 0, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}). \end{aligned}$$

La subseñal de fluctuación a nivel 2 será: $d^2 := (f \cdot w_1^2, \ldots, f \cdot w_{N/4}^2)$.

Y la subseñal de tendencia a segundo nivel: $a^2 := (f \cdot v_1^2, \ldots, f \cdot v_{N/4}^2)$.

En primer lugar, podemos observar que $\mid sop(w_j^2) \mid = \mid sop(v_m^2) \mid = 4$. Además, forman un sistema ortonormal. La demostración es análoga a las anteriores por lo que se obvia.

Sin embargo, notamos que aunque el valor medio de las wavelets de segundo nivel sigue siendo 0, el de las scaling cambia y es 1/2.

También es fácil ver que tanto las w_j^2 como las v_m^2 se obtienen desplazando $4j$ (o $4m$ respectivamente) entradas no nulas hacia la derecha de w_1^2 (o v_1^2 respectivamente), para $j, m > 1$.

Además, observamos que podemos obtener tanto las wavelets como las scaling de nivel 2 a partir del nivel anterior a través de las siguientes fórmulas:

$$\begin{aligned} v_1^2 &= \frac{v_1^1 + v_2^1}{\sqrt{2}} & \qquad w_1^2 &= \frac{v_1^1 - v_2^1}{\sqrt{2}} \\ &\vdots & &\vdots \\ v_{N/4}^2 &= \frac{v_{N/2-1}^1 + v_{N/2}^1}{\sqrt{2}} & w_{N/4}^2 &= \frac{v_{N/2-1}^1 - v_{N/2}^1}{\sqrt{2}} \end{aligned}$$

Otra forma de ver la situación anterior es que las wavelets y scaling a nivel 2 se deducen de las wavelets y scaling a nivel 1 "dilatando" el soporte (que ahora es el doble) y "contrayendo" sus valores al multiplicarlos por un factor $1/\sqrt{2}$ (para preservar la energía unitaria).

De estas igualdades deducimos que $\{v_1^2, \ldots, v_{N/4}^2, w_1^2, \ldots, w_{N/4}^2\}$ es una base ortonormal del subespacio $V^1 := lin\{v_1^1, \ldots, v_{N/2}^1\}$.

2.2.2. Análisis de multirresolución (MRA)

El *análisis de multirresolución* (MRA) es el eje central en el tratamiento de señales mediante wavelets. Dada una señal, esta se descompone en dos: una señal de promedio y otra de detalles. A continuación, la señal promedio obtenida la volvemos a descomponer en dos, y así sucesivamente. Para el proceso inverso, de síntesis, partimos de una señal de baja resolución (la última promedio obtenida) a la que se van agregando las señales detalle sucesivamente, obteniendo señales de mayor resolución.

En este apartado, describiremos este proceso con mayor precisión.

Sea $\{v_1^1, \ldots, v_{N/2}^1, w_1^1, \ldots, w_{N/2}^1\}$ una base ortonormal de $\mathbb{R}^N$. Entonces:

$$\begin{aligned} f &= (f \cdot v_1^1)v_1^1 + \cdots + (f \cdot v_{N/2}^1)v_{N/2}^1 + (f \cdot w_1^1)w_1^1 + \cdots + (f \cdot w_{N/2}^1)w_{N/2}^1 \\ &= a_1 v_1^1 + \cdots a_{N/2} v_{N/2}^1 + d_1 w_1^1 + \cdots d_{N/2}^1 w_{N/2}^1 \\ &= (\frac{a_1}{\sqrt{2}}, \frac{a_1}{\sqrt{2}}, \cdots, \frac{a_{N/2}}{\sqrt{2}}, \frac{a_{N/2}}{\sqrt{2}}) + (\frac{d_1}{\sqrt{2}}, \frac{-d_1}{\sqrt{2}}, \cdots, \frac{d_{N/2}}{\sqrt{2}}, \frac{-d_{N/2}}{\sqrt{2}}) \end{aligned}$$

Esto es: $f = A^1 + D^1$ que llamaremos **primer nivel de MRA de Haar**, donde A^1 es la señal del primer promedio:

$$A^1 = (\frac{a_1}{\sqrt{2}}, \frac{a_1}{\sqrt{2}}, \cdots, \frac{a_{N/2}}{\sqrt{2}}, \frac{a_{N/2}}{\sqrt{2}}) = \sum_{j=1}^{N/2} (f \cdot v_j^1) v_j^1,$$

y D^1 es la señal de primer detalle:

$$D^1 = (\frac{d_1}{\sqrt{2}}, \frac{-d_1}{\sqrt{2}}, \cdots, \frac{d_{N/2}}{\sqrt{2}}, \frac{-d_{N/2}}{\sqrt{2}}) = \sum_{j=1}^{N/2} (f \cdot w_j^1) w_j^1.$$

Observamos que la señal de promedio es una combinación lineal de las scaling, tomando como coeficientes los valores de la subseñal de primera tendencia, mientras que la señal de detalle es una combinación lineal de las wavelets, cuyos coeficientes son las fluctuaciones de primer nivel.

Podemos ir más allá y notar que A^1 es la proyección de f sobre el espacio V^1, y D^1 lo es sobre el espacio $W^1 := lin\{w_1^1, \ldots, w_{N/2}^1\}$ pues, como se ha visto antes, $V^1 \oplus^{\perp} W^1$.

Si expresamos A^1 y D^1 en función de f veremos que esta señal es la suma de una señal con menor resolución y otra de detalle:

$$A^1 = (\frac{f_1 + f_2}{2}, \frac{f_1 + f_2}{2}, \frac{f_3 + f_4}{2}, \frac{f_3 + f_4}{2}, \cdots, \frac{f_{N-1} + f_N}{2}, \frac{f_{N-1} + f_N}{2}),$$
$$D^1 = (\frac{f_1 - f_2}{2}, \frac{f_2 - f_1}{2}, \frac{f_3 - f_4}{2}, \frac{f_4 - f_3}{2}, \cdots, \frac{f_{N-1} - f_N}{2}, \frac{f_N - f_{N-1}}{2}).$$

En el segundo nivel del MRA de f tenemos que $f = A^2 + D^2 + D^1$ pues descomponemos A^1 en dos subseñales, con $A^2 = \sum_{j=1}^{N/4} (f \cdot v_j^2) v_j^2$ y $D^2 = \sum_{j=1}^{N/4} (f \cdot w_j^2) w_j^2$, pues $\{v_1^2, \ldots, v_{N/4}^2, w_1^2, \ldots, w_{N/4}^2, w_1^1, \ldots, w_{N/2}^1\}$ es una base ortonormal de $\mathbb{R}^N$.

Igual que en el nivel anterior, A^2 es la señal del segundo promedio de f, es decir, se proyecta esta señal (o A^1) sobre $V^2 := lin\{v_1^2, \ldots, v_{N/4}^2\}$ y D^2 es la señal de detalle a nivel 2 de f, es decir, la proyección de f (o A^1) sobre $W^2 := lin\{w_1^2, \ldots, w_{N/4}^2\}$.

Expresados en función de f:

$$A^2 = (\frac{f_1+f_2+f_3+f_4}{4}, \frac{f_1+f_2+f_3+f_4}{4}, \frac{f_1+f_2+f_3+f_4}{4}, \frac{f_1+f_2+f_3+f_4}{4}, \cdots, \frac{f_{N-3}+f_{N-2}+f_{N-1}+f_N}{4}),$$

$$D^2 = (\frac{f_1+f_2-f_3-f_4}{4}, \frac{f_1+f_2-f_3-f_4}{4}, \frac{f_3+f_4-f_1-f_2}{4}, \frac{f_3+f_4-f_1-f_2}{4}, \cdots, \frac{f_{N-1}+f_N-f_{N-3}-f_{N-2}}{4}).$$

Se puede continuar de forma inductiva y llegar a un MRA a nivel m de f tal que $f = A^m + D^m + \cdots + D^2 + D^1$, donde $m \leq k$ y $N = 2^k$. Esto nos permite reconstruir la señal original, f, a partir de una señal de baja resolución añadiendo los detalles.

Donde $A^m = \sum_{j=1}^{N/2^m} (f \cdot v_j^m) v_j^m$ y $D^m = \sum_{j=1}^{N/2^m} (f \cdot w_j^m) w_j^m$.

Las wavelets a nivel m se definen de la siguiente forma:

$$\begin{aligned}
w_1^m &:= (\underbrace{\frac{1}{2^{m/2}}, \ldots, \frac{1}{2^{m/2}}}_{2^{m-1}}, \underbrace{\frac{-1}{2^{m/2}}, \ldots, \frac{-1}{2^{m/2}}}_{2^{m-1}}, 0, \ldots, 0) \\
w_2^m &:= (\underbrace{0, \ldots, 0}_{2^m}, \underbrace{\frac{1}{2^{m/2}}, \ldots, \frac{1}{2^{m/2}}}_{2^{m-1}}, \underbrace{\frac{-1}{2^{m/2}}, \ldots, \frac{-1}{2^{m/2}}}_{2^{m-1}}, 0, \ldots, 0) \\
&\vdots \\
w_{N/2^m}^m &:= (0, \ldots, 0, \underbrace{\frac{1}{2^{m/2}}, \ldots, \frac{1}{2^{m/2}}}_{2^{m-1}}, \underbrace{\frac{-1}{2^{m/2}}, \ldots, \frac{-1}{2^{m/2}}}_{2^{m-1}}).
\end{aligned}$$

Y las scaling a nivel m así:

$$\begin{aligned}
v_1^m &:= (\underbrace{\frac{1}{2^{m/2}}, \ldots, \frac{1}{2^{m/2}}}_{2^m}, 0, \ldots, 0) \\
v_2^m &:= (\underbrace{0, \ldots, 0}_{2^m}, \underbrace{\frac{1}{2^{m/2}}, \ldots, \frac{1}{2^{m/2}}}_{2^m}, 0, \ldots, 0) \\
&\vdots \\
v_{N/2^m}^m &:= (0, \ldots, 0, \underbrace{\frac{1}{2^{m/2}}, \ldots, \frac{1}{2^{m/2}}}_{2^m}).
\end{aligned}$$

De la misma forma que las wavelets y scaling de nivel 2, las de nivel m se pueden expresar como combinación lineal de las scaling de nivel $m-1$:

$$\begin{aligned} v_1^m &= \frac{v_1^{m-1}+v_2^{m-1}}{\sqrt{2}} & w_1^m &= \frac{v_1^{m-1}-v_2^{m-1}}{\sqrt{2}} \\ &\vdots & &\vdots \\ v_{N/2^m}^m &= \frac{v_{N/2^{m-1}-1}^{m-1}+v_{N/2^{m-1}}^{m-1}}{\sqrt{2}} & w_{N/2^m}^m &= \frac{v_{N/2^{m-1}-1}^{m-1}-v_{N/2^{m-1}}^{m-1}}{\sqrt{2}} \end{aligned}$$

Por tanto, la señal del m-ésimo promedio, A^m, es la proyección ortogonal de f sobre el subespacio $V^m := lin\{v_1^m, \ldots, v_{N/2^m}^m\}$, mientras que la señal D^m de detalle a nivel m es la proyección de f sobre $W^m := lin\{w_1^m, \ldots, w_{N/2^m}^m\}$.

De aquí, podemos deducir que los subespacios $V^m, W^m \subseteq V^{m-1}$ y que $V^m \oplus^\perp W^m = V^{m-1}$.

Descomponiendo la señal promedio, tenemos que:

$$\mathbb{R}^N = V^1 \oplus^\perp W^1 = V^2 \oplus^\perp W^2 \oplus^\perp W^1 = \cdots = V^m \oplus^\perp W^m \oplus^\perp \cdots \oplus^\perp W^1.$$

Como $m \leq k (N = 2^k)$, el último nivel k tendrá una única wavelet y scaling:

$$\begin{aligned} w_1^k &= (\underbrace{\frac{1}{\sqrt{N}}, \ldots, \frac{1}{\sqrt{N}}}_{N/2}, \underbrace{\frac{-1}{\sqrt{N}}, \ldots, \frac{-1}{\sqrt{N}}}_{N/2}) \\ v_1^k &= (\underbrace{\frac{1}{\sqrt{N}}, \ldots, \frac{1}{\sqrt{N}}}_{N}) \end{aligned}$$

Esto hace que, dada una señal f, A^k es la resolución más baja posible, una señal constante:

$$A^k = (\frac{f_1+f_2+\cdots+f_N}{N}, \ldots, \frac{f_1+f_2+\cdots+f_N}{N}),$$

es decir, se toma el valor promedio de toda la señal.

Si queremos recuperar la señal original, solo hemos de agregarle sucesivamente a la señal promedio más baja, A^k, los detalles $D^k, D^{k-1}, \ldots, D^1$.

Podemos considerar la señal promedio como una señal "borrosa" de la señal original a la que agregamos los detalles para que vuelva a ser la original. En la descomposición del MRA para cada f solo hemos de calcular las tendencias, a_j, y las fluctuaciones, d_j, para $1 \leq j \leq 2^{N/2^m}$. Esto nos lleva, de forma natural, a la transformada de Haar.

2.2.3. Transformada de Haar

Básicamente, lo que hemos descrito anteriormente describe cómo funciona la *transformada de Haar*. La de nivel m es una transformación donde a cada señal f le hace corresponder una señal resultante de los coeficientes de las tendencias de nivel m y las fluctuaciones de este nivel, e inferior, ordenadas, es decir, $f \xmapsto{H_m} (a^m \mid d^m \mid d^{m-1} \mid \cdots \mid d^1)$.

Se trata de una transformación lineal y también ortogonal (por la forma en cómo se obtienen los coeficientes, explicados en apartados anteriores). Matricialmente, para el nivel 1 tenemos:

$$\left[\frac{a^1}{d^1}\right] = \left[\begin{array}{c} v_1^1 \\ \vdots \\ v_{N/2}^1 \\ \hline w_1^1 \\ \vdots \\ w_{N/2}^1 \end{array}\right] \left[\begin{array}{c} f_1 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ f_N \end{array}\right]$$

Se observa que las filas de la matriz forman una base ortonormal. Luego H_1 es ortogonal. Esto permite definir la transformada inversa (a nivel 1): $(a^1 \mid d^1) \xmapsto{H_1^{-1}} f$ que se calcula, transponiendo la matriz:

$$\left[\begin{array}{c} v_1^1 \\ \vdots \\ v_{N/2}^1 \\ \hline w_1^1 \\ \vdots \\ w_{N/2}^1 \end{array}\right]^T \left[\frac{a^1}{d^1}\right] = \left[\begin{array}{c} f_1 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ f_N \end{array}\right]$$

Y, de forma análoga, la transformación H_m a nivel m es también ortogonal.

$$\begin{bmatrix} \frac{a^m}{d^m} \\ \hline \vdots \\ \hline d^1 \end{bmatrix} = \begin{bmatrix} v_1^m \\ \vdots \\ \frac{v_{N/2^m}^m}{w_1^m} \\ \vdots \\ \frac{w_{N/2^m}^m}{\vdots} \\ \hline w_1^1 \\ \vdots \\ w_{N/2}^1 \\ \\ \end{bmatrix} \begin{bmatrix} f_1 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ f_N \end{bmatrix}$$

Y análogamente podemos invertir el proceso de forma sencilla.

Volvamos al caso $m = 1$, entonces la transformada de Haar se define como:

$$H_1(f) = (\underbrace{\frac{f_1 + f_2}{\sqrt{2}}, \ldots, \frac{f_{N-1} + f_N}{\sqrt{2}}}_{N/2}, \underbrace{\frac{f_1 - f_2}{\sqrt{2}}, \ldots, \frac{f_{N-1} - f_N}{\sqrt{2}}}_{N/2}).$$

Tal y como vimos anteriormente, se trata de una "compresión" de la señal original en la primera mitad multiplicada por $1/\sqrt{2}$ y de las fluctuaciones en la segunda mitad. Las fluctuaciones serán "pequeñas" si la señal original no tiene cambios "bruscos".

A niveles superiores, $m > 1$, comprimimos la mitad de la subseñal de tendencia del paso $m - 1$ y la colocamos al principio de la señal, a continuación colocamos las fluctuaciones de dicha subseñal y, por último, las fluctuaciones que ya teníamos en el nivel anterior.

Además, la transformada de Haar presenta dos propiedades fundamentales. Por una parte, por ser una transformación ortogonal, conserva la energía. Y, por otra parte se cumple la característica de las fluctuaciones pequeñas. Esto es, si una señal es constante, al multiplicarla escalarmente por las wavelets, su resultado será cero, es decir, la fluctuación es nula. Si la señal no "cambia" mucho entonces la fluctuación será "pequeña".

La conjugación de las dos observaciones anteriores lleva a la *compactación de la energía*. Concretamente, la energía de la señal original es la suma de las energías de las subseñales de tendencia y fluctuación: $E(f) = E(a^m) + E(d^m) + \cdots + E(d^1)$.

Sin embargo, por la característica de las fluctuaciones pequeñas, va a ser la tendencia quien concentre, generalmente, la mayor parte de la energía de la transformada. El *perfil de energía acumulada* es una forma de cuantificarla:

$$\left(\frac{f_1^2}{E(f)}, \frac{f_1^2 + f_2^2}{E(f)}, \ldots, \frac{f_1^2 + \cdots + f_{N-1}^2}{E(f)}, 1 \right).$$

El número de operaciones para la transformada de Haar es $O(N)$.

2.3. Familia de wavelets ortogonales

En esta sección se describen las wavelets ortogonales descubiertas por Daubechies cuya diferencia respecto a las de Haar es la forma en cómo se obtienen las scaling y wavelets. Aunque mantienen las propiedades del MRA y la transformada, tratan mejor las señales más "suaves".

2.3.1. Wavelets de Daubechies

Las wavelets de Daubechies fueron introducidas en el año 1990 por Ingrid Daubechies. Como se ha mencionado anteriormente, se trata de una familia de wavelets ortogonales que definen una transformación discreta de wavelets [16].

Definición 21. Definimos las scaling y wavelets **Daubechies2** (*db2*) a nivel 1 de la siguiente forma, respectivamente:

$$\begin{aligned} v_1^1 &:= (\alpha_1, \alpha_2, \alpha_3, \alpha_4, 0, \ldots, 0) \\ v_2^1 &:= (0, 0, \alpha_1, \alpha_2, \alpha_3, \alpha_4, 0, \ldots, 0) \\ &\vdots \\ v_{N/2-1}^1 &:= (0, \ldots, 0, \alpha_1, \alpha_2, \alpha_3, \alpha_4) \\ v_{N/2}^1 &:= (\alpha_3, \alpha_4, 0, \ldots, 0, \alpha_1, \alpha_2) \end{aligned} \qquad \begin{aligned} w_1^1 &:= (\beta_1, \beta_2, \beta_3, \beta_4, 0, \ldots, 0) \\ w_2^1 &:= (0, 0, \beta_1, \beta_2, \beta_3, \beta_4, 0, \ldots, 0) \\ &\vdots \\ w_{N/2-1}^1 &:= (0, \ldots, 0, \beta_1, \beta_2, \beta_3, \beta_4) \\ w_{N/2}^1 &:= (\beta_3, \beta_4, 0, \ldots, 0, \beta_1, \beta_2) \end{aligned}$$

Donde las entradas de las scaling son:

$$\alpha_1 := \frac{1+\sqrt{3}}{4\sqrt{2}},\ \alpha_2 := \frac{3+\sqrt{3}}{4\sqrt{2}},\ \alpha_3 := \frac{3-\sqrt{3}}{4\sqrt{2}},\ \alpha_4 := \frac{1-\sqrt{3}}{4\sqrt{2}}.$$

y las entradas en las wavelets son:

$$\beta_1 := \alpha_4,\ \beta_2 := -\alpha_3,\ \beta_3 := \alpha_2,\ \beta_4 := -\alpha_1.$$

Estos valores cumplen las siguientes ecuaciones:

$$\begin{cases} \alpha_1^2 + \alpha_2^2 + \alpha_3^2 + \alpha_4^2 = 1, \\ \alpha_1\alpha_3 + \alpha_2\alpha_4 = 0 \end{cases}$$

Estas relaciones implican que $\{v_1^1, \ldots, v_{N/2}^1, w_1^1, \ldots, w_{N/2}^1\}$ es una base ortonormal de $\mathbb{R}^N$. Además:

$$\beta_1 + \beta_2 + \beta_3 + \beta_4 = 0$$

que conlleva a que el valor medio de las wavelets sea 0.

Definiendo las subseñales de primera tendencia (a^1) y fluctuaciones (d^1) igual que las de Haar:

$$\begin{aligned} a^1 &:= (f \cdot v_1^1, f \cdot v_2^1, \ldots, f \cdot v_{N/2}^1), \\ d^1 &:= (f \cdot w_1^1, f \cdot w_2^1, \ldots, f \cdot w_{N/2}^1), \end{aligned}$$

se observa que los a_i^1, son una media ponderada de los 4 valores consecutivos de una señal f, mientras que las d_i^1 son una diferencia ponderada, para $i = 1, 2, \ldots, N/2$.

Para niveles superiores, se definen a partir del nivel anterior de forma recurrente:

$$\begin{aligned} v_1^m &= \alpha_1 v_1^{m-1} + \alpha_2 v_2^{m-1} + \alpha_3 v_3^{m-1} + \alpha_4 v_4^{m-1} \\ &\vdots \\ v_{N/2^m}^m &= \alpha_1 v_{N/2^{m-1}-1}^{m-1} + \alpha_2 v_{N/2^{m-1}}^{m-1} + \alpha_3 v_1^{m-1} + \alpha_4 v_2^{m-1}; \end{aligned}$$

$$\begin{aligned} w_1^m &= \beta_1 v_1^{m-1} + \beta_2 v_2^{m-1} + \beta_3 v_3^{m-1} + \beta_4 v_4^{m-1} \\ &\vdots \\ w_{N/2^m}^m &= \beta_1 v_{N/2^{m-1}-1}^{m-1} + \beta_2 v_{N/2^{m-1}}^{m-1} + \beta_3 v_1^{m-1} + \beta_4 v_2^{m-1}. \end{aligned}$$

De nuevo, como ocurre con la wavelet de Haar, el MRA a nivel m de una señal f se define: $f = A^m + D^m + \cdots + D^2 + D^1$, donde $m \leq k-1$ ($N = 2^k$) con la señal promedio y detalle, respectivamente:

$$\begin{aligned} A^m &= (f \cdot v_1^m)v_1^m + \cdots + (f \cdot v_{N/2^m}^m)v_{N/2^m}^m, \\ D^m &= (f \cdot w_1^m)w_1^m + \cdots + (f \cdot w_{N/2^m}^m)w_{N/2^m}^m. \end{aligned}$$

Por lo tanto, la m-ésima señal del promedio, A^m, es también la proyección ortogonal de f sobre el subespacio $V^m := lin\{v_1^m, \ldots, v_{N/2^m}^m\}$; mientras que la señal de detalle a nivel m, D^m, es la proyección de f sobre $W^m := lin\{w_1^m, \ldots, w_{N/2^m}^m\}$.

Es decir, obtenemos, de nuevo, que $\mathbb{R}^N = V^m \oplus^\perp W^m \oplus^\perp \cdots \oplus^\perp W^1$, con $m \leq k-1$ ($N = 2^k$). En el último nivel solo tendríamos un par de wavelets y de scaling.

La transformada *db2* a nivel m también es una transformación tal que a cada señal f le hace corresponder la señal en la tendencia de nivel m junto con las fluctuaciones de ese nivel e inferiores ordenadas, es decir, $f \xmapsto{db2_m} (a^m \mid d^m \mid d^{m-1} \mid \cdots \mid d^1)$.

Esta transformación también es ortogonal y se puede escribir en forma matricial:

$$\begin{bmatrix} a^m \\ \hline d^m \\ \hline \vdots \\ \hline d^1 \end{bmatrix} = \begin{bmatrix} v_1^m \\ \vdots \\ \hline v_{N/2^m}^m \\ \hline w_1^m \\ \vdots \\ \hline w_{N/2^m}^m \\ \hline \vdots \\ \hline w_1^1 \\ \vdots \\ w_{N/2}^1 \\ \\ \\ \end{bmatrix} \begin{bmatrix} f_1 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ f_N \end{bmatrix}$$

Por ser una transformación ortogonal, la energía se conserva ($E(db2_m(f)) = E(f)$) y se da la propiedad de las "fluctuaciones pequeñas". Además, la transformación se puede invertir trasponiendo la matriz. Es más, podemos obtener las subseñales de tendencia y fluctuación aplicando la transformación $db2_1$ al nivel anterior.

Característica de las fluctuaciones pequeñas

Si una señal en el soporte de una wavelet *db2* es lineal, el producto escalar de la señal por dicha wavelet es 0, es decir, su fluctuación es nula. Si la señal es "aproximadamente" lineal a lo largo del soporte, entonces la fluctuación asociada es "pequeña".

Este tipo de wavelets obtienen una mejora con respecto a las de Haar porque cumplen que:

$$1\beta_1 + 2\beta_2 + 3\beta_3 + 4\beta_4 = 0.$$

Veamos que la afirmación es cierta. Para el nivel 1. Sea $\{i, i+1, i+2, i+3\}$ el soporte de cierta wavelet w_j^1. Suponemos que f es lineal a lo largo de dicho soporte, es decir, existen $a, b \in \mathbb{R}$ tales que:

$$f_l = a + b(l - i + 1),\ l = i, \ldots, i+3.$$

Entonces:

$$d_j^1 = f \cdot w_j^1 = \beta_1(a+b) + \beta_2(a+2b) + \beta_3(a+3b) + \beta_4(a+4b) =$$
$$a(\beta_1 + \beta_2 + \beta_3 + \beta_4) + b(1\beta_1 + 2\beta_2 + 3\beta_3 + 4\beta_4) = b(1\beta_1 + 2\beta_2 + 3\beta_3 + 4\beta_4) = 0.$$

Los valores de los coeficientes de $db2$ se obtienen a partir del sistema de ecuaciones que hemos obtenido y descrito anteriormente:

$$\begin{cases} \beta_1^2 + \beta_2^2 + \beta_3^2 + \beta_4^2 = 1, \\ \beta_1 + \beta_2 + \beta_3 + \beta_4 = 0, \\ \beta_1\beta_3 + \beta_2\beta_4 = 0, \\ 1\beta_1 + 2\beta_2 + 3\beta_3 + 4\beta_4 = 0. \end{cases}$$

Por último mencionar que el número de operaciones necesarias para realizar la transformada $db2$ es $O(N)$.

Si aumentamos el número de condiciones a las *db2* y, por tanto, también aumentamos el número de scaling o wavelets, entonces, se puede mejorar su comportamiento para señales "suaves".

Definición 22. Definimos las scaling y wavelets **Daubechies3** (*db3*) a nivel 1 de la siguiente forma, respectivamente:

$$\begin{array}{rcl@{\qquad}rcl} v_1^1 & := & (\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5, \alpha_6, 0, \ldots, 0) & w_1^1 & := & (\beta_1, \beta_2, \beta_3, \beta_4, \beta_5, \beta_6, 0, \ldots, 0) \\ v_2^1 & := & (0, 0, \alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5, \alpha_6, 0, \ldots, 0) & w_2^1 & := & (0, 0, \beta_1, \beta_2, \beta_3, \beta_4, \beta_5, \beta_6, 0, \ldots, 0) \\ \vdots & & & \vdots & & \\ v_{N/2-1}^1 & := & (\alpha_5, \alpha_6, 0, \ldots, 0, \alpha_1, \alpha_2, \alpha_3, \alpha_4) & w_{N/2-1}^1 & := & (\beta_5, \beta_6, 0, \ldots, 0, \beta_1, \beta_2, \beta_3, \beta_4) \\ v_{N/2}^1 & := & (\alpha_3, \alpha_4, \alpha_5, \alpha_6, 0, \ldots, 0, \alpha_1, \alpha_2) & w_{N/2}^1 & := & (\beta_3, \beta_4, \beta_5, \beta_6, 0, \ldots, 0, \beta_1, \beta_2) \end{array}$$

Donde los valores que definen las scaling *db3* son:

$$\begin{aligned}
\alpha_1 &:= 0.332670552950083 & \alpha_4 &:= -0.135011020010255\\
\alpha_2 &:= 0.806891509311092 & \alpha_5 &:= -0.0854412738820267\\
\alpha_3 &:= 0.459877502118491 & \alpha_6 &:= 0.0352262918857095
\end{aligned}$$

y las correspondientes para las wavelets son:

$$\beta_1 := \alpha_6,\ \beta_2 := -\alpha_5,\ \beta_3 := \alpha_4,\ \beta_4 := -\alpha_3,\ \beta_5 := \alpha_2,\ \beta_6 := -\alpha_1.$$

Igual que antes, se pueden definir las scaling y wavelets de niveles superiores de forma recursiva:

$$\begin{aligned}
v_1^m &= \alpha_1 v_1^{m-1} + \alpha_2 v_2^{m-1} + \alpha_3 v_3^{m-1} + \alpha_4 v_4^{m-1} + \alpha_5 v_5^{m-1} + \alpha_6 v_6^{m-1}\\
&\vdots\\
v_{N/2^m}^m &= \alpha_1 v_{N/2^{m-1}-1}^{m-1} + \alpha_2 v_{N/2^{m-1}}^{m-1} + \alpha_3 v_1^{m-1} + \alpha_4 v_2^{m-1} + \alpha_5 v_3^{m-1} + \alpha_6 v_4^{m-1};
\end{aligned}$$

$$\begin{aligned}
w_1^m &= \beta_1 v_1^{m-1} + \beta_2 v_2^{m-1} + \beta_3 v_3^{m-1} + \beta_4 v_4^{m-1} + \beta_5 v_5^{m-1} + \beta_6 v_6^{m-1}\\
&\vdots\\
w_{N/2^m}^m &= \beta_1 v_{N/2^{m-1}-1}^{m-1} + \beta_2 v_{N/2^{m-1}}^{m-1} + \beta_3 v_1^{m-1} + \beta_4 v_2^{m-1} + \beta_5 v_3^{m-1} + \beta_6 v_4^{m-1}.
\end{aligned}$$

Las subseñales de tendencia (A), fluctuaciones (D), la descomposición ortogonal del MRA y la transformada se definen de forma análoga al caso de Haar y *db2*.

De la misma manera que en las Daubechies2 se añadió una segunda condición para mejorar la señal respecto a la de Haar, ahora se agrega una tercera. Concretamente, se ha de cumplir el siguiente sistema de ecuaciones lineales:

$$\begin{cases}
\beta_1 + \beta_2 + \beta_3 + \beta_4 + \beta_5 + \beta_6 = 0,\\
1\beta_1 + 2\beta_2 + 3\beta_3 + 4\beta_4 + 5\beta_5 + 6\beta_6 = 0,\\
1^2\beta_1 + 2^2\beta_2 + 3^2\beta_3 + 4^2\beta_4 + 5^2\beta_5 + 6^2\beta_6 = 0.
\end{cases}$$

Esta última ecuación es la que permite mejorar el tratamiento de señales bajo condiciones de diferenciabilidad buenas.

La característica de las fluctuaciones pequeñas se continúa cumpliendo de forma muy similar a como ocurría antes.

Si una señal en el soporte de una wavelet *db3* es cuadrática, el producto escalar de la señal por dicha wavelet es 0, es decir, su fluctuación es nula. Si la señal es "aproximadamente" cuadrática a lo largo del soporte, entonces la fluctuación asociada es "pequeña".

Esto implica que la compresión y eliminación del ruido de señales de estas características será mejor con la transformada *db3* que con la *db2* o de Haar. En cambio, si lo que pretendemos es buscar cambios bruscos en la señal (es decir, donde existan "picos"), entonces, es mejor utilizar la transformada *db2* que no esta.

Podemos, ahora, generalizar las transformadas de wavelets de Daubechies de la siguiente forma:

Definición 23. Se definen las transformadas de wavelets de Daubechies, *dbJ* con $J = 4, 5, \ldots, 10$ de la misma forma que antes.

Las scaling $\alpha_1, \ldots, \alpha_{2J}$ son tales que:

$$\begin{cases} \alpha_1^2 + \alpha_2^2 + \cdots + \alpha_{2J}^2 = 1, \\ \alpha_1 + \alpha_2 + \cdots + \alpha_{2J} = \sqrt{2}. \end{cases}$$

Y los números wavelets $\beta_1, \ldots, \beta_{2J}$ se toman como:

$$\beta_1 := \alpha_{2J},\ \beta_2 := -\alpha_{2J-1},\ \ldots,\ \beta_{2J-1} := \alpha_2,\ \beta_{2J} := -\alpha_1.$$

Además, han de cumplir la ortogonalidad junto con las ecuaciones de anulación de momentos:

$$\beta_1 + 2^n\beta_2 + 3^n\beta_3 + \cdots + (2J)^n\beta_{2J} = 0,\ n = 0, 1, \ldots, J-1.$$

Es, precisamente, la anulación de momentos lo que permite obtener fluctuaciones pequeñas.

2.4. Compresión de señales y reducción del ruido

Hasta ahora se ha podido ver cómo compactar la energía a través de la transformada wavelet pues se encuentra concentrada en la tendencia. Para comprimir señales y reducir ruido hace falta llevar a cabo los tres procesos que se detallan a continuación. Utilizaremos el término "señal contaminada" como aquella que contenga ruido, es decir, una alteración no deseada de los valores de la señal original.

En primer lugar, aplicamos la **transformada wavelet** a la señal (esté o no contaminada) a un nivel determinado. Seguidamente, "**cuantizamos**"los valores obtenidos de la transformación con el fin de localizarlos en un conjunto finito. Entre estos dos pasos, podemos dar un valor umbral (necesario para reducir el ruido y opcional para comprimir) para anular valores que se encuentren cerca del 0. Tanto fijar un umbral como cuantizar los valores implica perder información. Finalmente, si queremos comprimir será necesario **codificar** en modo binario la señal obtenida con el fin de que ocupe lo mínimo posible.

Cuanto más suave queremos la señal transformada, en el primer paso, elegiremos una señal con mayor cantidad de momentos nulos. A continuación, se detallan los procesos de cuantización, umbralizado y codificación pues la transformada ya se detalló anteriormente.

2.4.1. Cuantización y umbralizado

Para el proceso de cuantización supondremos que los valores de la señal transformada se encuentran en el intervalo $[-\alpha, \alpha]$. También se puede restar su valor medio para centrarla en el 0.

Definición 24. Sea q un entero positivo par. Si se quiere cuantizar la señal a q niveles, tomamos $\delta := 2\alpha/q$. Se define la función de **cuantización escalar uniforme**:

$$Q(x) := \begin{cases} 0 & \text{si } |x| \leq \delta \\ sig(x)(k+1/2)\delta & \text{si } k\delta < |x| \leq (k+1)\delta \end{cases}.$$

Es decir, los valores que caen en cada subintervalo de la partición son redondeados al punto medio. Al aplicarle a la señal la función de cuantización $Q(x)$, los valores los tomará ahora en el conjunto finitio:

$$\left\{-\frac{q-1}{q}\alpha, -\frac{q-3}{q}\alpha, \ldots, 0, \ldots, \frac{q-3}{q}\alpha, \frac{q-1}{q}\alpha\right\}.$$

Si lo que buscamos es comprimir la señal, antes de hacer una cuantización, realizamos una *umbralización* de los valores. Un criterio a tener en cuenta se basa en la energía. Para ello, primero ordenamos de forma decreciente los valores absolutos de la transformada de una señal f:

$$y_1 \geq y_2 \geq \cdots \geq y_N.$$

Y calculamos los cocientes:

$$\frac{y_1^2}{E(f)}, \frac{y_1^2 + y_2^2}{E(f)}, \ldots$$

hasta llegar a un valor cercano a 1, es decir, hasta conseguir un nivel alto de energía acumulada.

Si nuestro umbral está en 0.9999 significaría que la energía acumulada de nuestro umbral es del 99.99 %. Sea j el primer índice tal que:

$$\frac{y_1^2 + y_2^2 + \cdots + y_j^2}{E(f)} \geq 0.9999.$$

Entonces, el valor umbral (*threshold*) a tomar es $T := y_j$.

A continuación, se anularían todos los coeficientes que no superen este umbral T. Existen varias formas de hacerlo, definimos dos. El caso más simple consiste en anular los coeficientes por debajo del umbral y dejar como están los restantes, lo que se concreta en la llamada función de umbralización fuerte.

Definición 25. Se define la función de **umbralización fuerte**:

$$D_f(x) := \begin{cases} 0 & \text{si } |x| \leq T \\ x & \text{si } |x| > T \end{cases}.$$

Otro procedimiento alternativo, que tan solo difiere en cómo afecta a los coeficientes que superan el umbral, es mediante la umbralización suave.

Definición 26. Se define la función de **umbralización suave**:

$$D_s(x) := \begin{cases} 0 & \text{si } |x| \leq T \\ x - T & \text{si } x > T \\ x + T & \text{si } x < -T \end{cases} .$$

En ambos casos se anulan los coeficientes debajo del umbral pero difiere en cómo se trata a los coeficientes que lo superan. La umbralización suave, también denominada *Shrinkage Denoising*, se considera una extensión de la fuerte. El resultado de ambas se puede ver en la Figura 2.1.

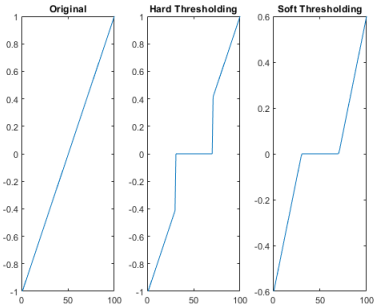

Figura 2.1: Señal original y resultado tras aplicarle un umbral fuerte y suave respectivamente.

Se observa que en el umbral fuerte se crean distorsiones en $D_f(x) = \pm T$ mientras que no ocurre eso al aplicar la suave [1].

Si nuestro objetivo es reducir el ruido de una señal, es necesario umbralizar la señal. Supongamos que dicho ruido es aditivo, es decir, dada una señal f tenemos una señal contaminada g tal que: $g = f + x$. También se supone que el ruido, x, es aleatorio (llamado ruido blanco) y de media 0.

Al realizar la transformada wavelet de la señal g, los coeficientes del ruido x de altas frecuencias por su componente aleatoria, se concentrarán en las fluctuaciones.

Un método para reducir el ruido consistiría en establecer el umbral de corte examinando visualmente la porción de fluctuaciones en la transformada. Pero otros métodos como los basados en estadísticos son más sofisticados.

Si el ruido es gaussiano, es decir, que su intensidad es equivalente a su función de probabilidad (Figura 2.2), se puede escoger el valor T automáticamente sin conocer la señal. Para reducirlo adecuadamente es necesario que el umbral T sea menor que el que se precisa para capturar la energía original (no contaminada) eficientemente. De lo contrario, se podrían eliminar coeficientes significativos de la señal.

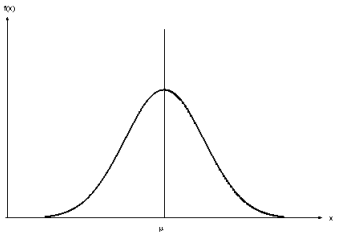

Figura 2.2: Curva de función de densidad de probabilidad de Gauss.

Un resultado de Donoho y Johnstone cuantifica el valor óptimo del umbral mediante la siguiente fórmula:

$$T = \sigma\sqrt{2 \cdot log(N)}$$

donde N la longitud de la señal con ruido y σ es la desviación típica del ruido. Se puede estimar este valor a partir de la primera fluctuación de la señal transformada.

Finalmente, añadimos un último concepto relacionado con la cuatización y umbralización que nos servirá para introducir la codificación:

Definición 27. Definimos el **mapa de significancia** como el conjunto de ceros y unos que indican si un determinado valor (su correspondiente índice) sobrevivió al umbral (corresponde un 1) o no (se denota por 0).

2.4.2. Codificación

Con los dos pasos anteriores, gran parte de los coeficientes pasan a ser nulos. Esta información se almacena en el mapa de significancia. La codificación es necesaria si se desea comprimir la información. Así, si el concepto previamente introducido, mapa de significancia, contiene muchos valores nulos significa que la señal transformada se comprime eficientemente con los métodos usuales, sin pérdida de información.

Si se tiene en cuenta la *entropía* (cantidad de ruido e incertidumbre) alcanzaremos un buen ratio de compresión. Y es que no todos los valores de la señal transformada y cuantizada se repiten con la misma frecuencia. Esto se puede verificar a través de un histograma de frecuencias. Por ello, se debe asignar menos bits a aquellos valores con mayor frecuencia de aparición y más a los que menos aparecen; de lo contrario se estaría realizando un gasto innecesario y obteniendo una mala compresión. Este proceso se puede precisar de forma matemática utilizando resultados fundamentales en el campos de la *teoría de la información.*

Supongamos que queremos codificar la siguiente secuencia de 8 números, del 1 al 3: 21122123. Esto lo podríamos hacer utilizando, por ejemplo, 2 bits por número, de la siguiente forma:

$$1 \mapsto 00,\ 2 \mapsto 01,\ 3 \mapsto 11,$$

entonces, nuestra nueva secuencia codificada sería la siguiente: 0100000101000111, es decir, nos encontramos con una secuencia de 16 bits.

Ahora bien, se observa que cada número tiene una frecuencia distinta de aparición. Una mejor codificación sería aquella en la que se asigna menos bits a los valores más frecuentes y un mayor número de bits a los que menos veces aparecen. En este caso, observamos que el 2 es el que más se repite y 3 el que menos y decidimos asignar estos valores:

$$1 \mapsto 10,\ 2 \mapsto 0,\ 3 \mapsto 110$$

codificado queda: 0101000100110 en 13 bits. Comparándola con la codificación anterior observamos que hemos ahorrado un 18.75 %.

Es importante mencionar que en este último caso donde el sistema de codificación tiene una asignación de longitud variable de bits, no debe existir ambigüedad en la decodificación. Es por ello, que debe cumplirse la condición del prefijo, es decir, que ningún símbolo tenga asignada una secuencia de bits que sea prefijo de otro signo. A esto se le denomina *código prefijo.*

En particular, esta última codificación hecha se conoce como *código Huffman* descubierto por David A. Huffman en 1951; se trata de un código prefijo. Se sigue utilizando hoy en día para transmitir datos comprimidos por internet, módem o fax.

Para explicar más este último concepto, tomamos el siguiente ejemplo. Sea $A = \{a_1, a_2, a_3, a_4\}$ los valores que puede tomar una señal. Entonces, se podría codificar de las siguientes formas:

c_1: no decodificable	c_2: decodificable	c_3: código prefijo	c_4: código prefijo
$a_1 \mapsto 0$	$a_1 \mapsto 0$	$a_1 \mapsto 00$	$a_1 \mapsto 0$
$a_2 \mapsto 1$	$a_2 \mapsto 01$	$a_2 \mapsto 01$	$a_2 \mapsto 10$
$a_3 \mapsto 00$	$a_3 \mapsto 011$	$a_3 \mapsto 10$	$a_3 \mapsto 110$
$a_4 \mapsto 11$	$a_4 \mapsto 0111$	$a_4 \mapsto 11$	$a_4 \mapsto 111$

Se observa que si codificamos una señal con c_1 no podríamos decodificarla. En cambio, las otras tres opciones sí se podría aunque la señal c_2, a diferencia de la c_3 y c_4 no es un código prefijo como se explicó antes. Cualquier longitud fija y decodificable de forma única es también un prefijo. Estos códigos prefijos se pueden representar mediante árboles como se muestra en la Figura 2.3. El representado a la derecha de la figura es el que daría lugar a la codificación de Huffman.

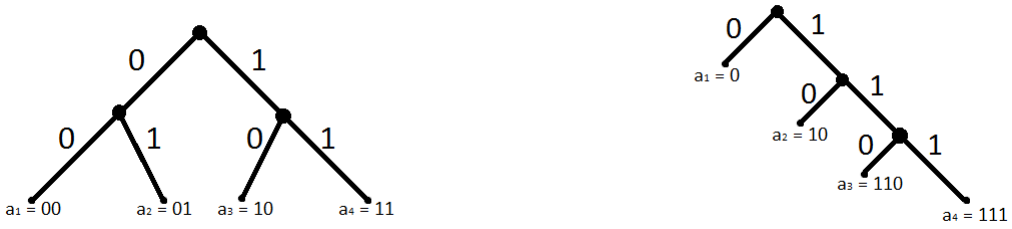

Figura 2.3: Representación de los códigos prefijos c_3 y c_4 en forma de árbol binario.

2.4.3. Medidas de aproximación

Una vez completados los pasos anteriores, se puede cuantificar el grado de aproximación de la señal obtenida respecto a la original. Esto se puede hacer de diferentes maneras. Una de ellas es a través del error cuadrático medio, muy útil para el tratamiento de señales.

Definición 28. Sea f y g dos señales distintas, el **error cuadrático medio** entre f y g se define:

$$RMS(f,g) := \sqrt{\frac{(f_1-g_1)^2 + (f_2-g_2)^2 + \cdots + (f_N-g_N)^2}{N}} = \frac{\sqrt{E(f-g)}}{\sqrt{N}}.$$

2.5. Aplicaciones en Matlab

En esta sección expondremos algunos ejemplos de las explicaciones teóricas anteriores. Para ello, se utilizará Matlab, una herramienta de software matemático con un entorno de programación y lenguaje propio [9]. Este programa utiliza una aritmética de doble precisión por lo que los datos son almacenados internamente con 8 bytes (es decir, 64 bits, en lugar de 32 bits). Los algoritmos utilizados han sido adaptados a partir de las explicaciones y programas encontrados en [20].

Dado que la señal es continua, discretizamos la señal tomando valores de esta en un intervalo definido. Estos valores se representarán en un vector columna. Aunque todos los programas se harán a través de la línea de comandos, Matlab también dispone de una herramienta gráfica para nuestro caso: *Wavelet Toolbox*. A pesar de que esta segunda opción resulta más fácil de utilizar, ofrece un menor control y tampoco es posible visualizar las subseñales de tendencia y fluctuación con ella.

2.5.1. Wavelets de Haar

A continuación, vamos a analizar la siguiente señal

$$g_1(x) := 20x^2(1-x)^4 cos(12\pi x)$$

evaluada en el intervalo $[0, 1]$ sobre 2048 puntos, es decir, $N = 2048 = 2^{11}$ donde $k = 11$.

Para ello, se ha hecho un programa principal (B.3) en el que se introducen los límites del intervalo, la cantidad de puntos y el número de niveles de las wavelets y scalings a calcular. En este caso, se ha escogido utilizar una wavelet de $Haar$ de 3 niveles.

Para obtener los valores de la señal, se utiliza la función *obtenerValoresFuncionG* B.2 que obtiene valores de acuerdo a la función G (B.1) escogida.

En el programa principal, una vez se ha obtenido el vector columna con todos los N valores de la señal en el intervalo, se obtiene en un vector C los valores de tendencia y fluctuación de la señal. El producto escalar entre la señal "s" y las wavelets y las scalings de los distintos niveles los hace la función *wavedec*.

Con la función *appcoef* extraemos los coeficientes de tendencia del nivel indicado y con *detcoef* los de fluctuación. A partir de estos podemos calcular, con la función *wrcoef*, las señales promedio y detalle del análisis de multirresolución.

Finalmente, podemos representar todos estos valores en una ventana con múltiples gráficos como se ve en la Figura 2.4.

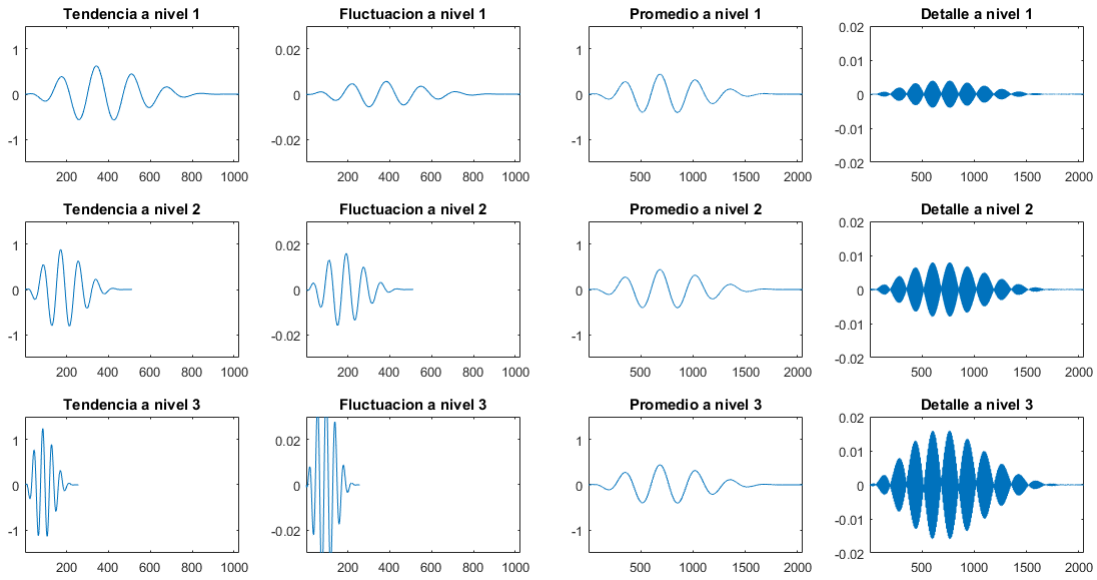

Figura 2.4: Análisis de tres niveles de la señal $g_1(x)$. Las columnas representan, por orden, las subseñales de tendencia, fluctuación, promedio y detalle. Cada fila representa un nivel.

Tal y como se explicó en capítulos anteriores, las subseñales de tendencia y fluctuaciones van haciéndose más pequeñas en cuanto a número de valores. Concretamente, se reducen a la mitad; pasando de los 2048 puntos iniciales a 1024 en el primer paso. Así mismo, sus valores se van incrementando con cada nivel. Podemos observar que en el tercer nivel, la tendencia supera el valor 1 y las fluctuaciones el 0.2 cuando en niveles anteriores no lo hace.

Para estudiar bien el caso de las otras dos subseñales, nos hemos enfocado en unos valores concretos ampliando la imagen. Podemos ver este resultado en la Figura 2.5. En ella se puede apreciar, tal y como se explicó en el capítulo de MRA, que las subseñales pierden "nitidez" en cada nivel del promedio. Y esto es precisamente lo que se observa en esta figura. Para ir de un punto a otro, los saltos son cada vez más bruscos. Esto se debe a que el número de valores va reduciéndose en cada nivel.

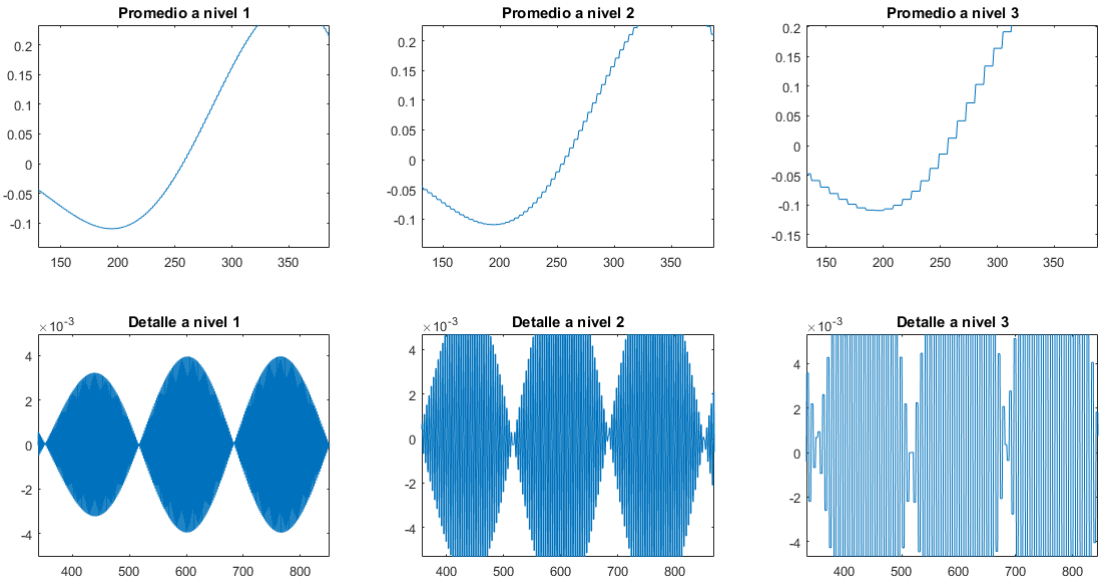

Figura 2.5: Zoom de las subseñales promedio y de detalle a nivel 1, 2 y 3 de la señal $g_1(x)$.

Si quisieramos calcular el soporte de la fluctuación de nivel 1 de la señal $g_1(x)$ podemos usar el programa que se encuentra en el Anexo B.4. Y observaríamos que su orden (sumamos los elementos del vector) es de 1024, precisamente la cantidad de elementos de la subseñal. En cambio, con la función

$$g_2(x) := sign(sin(12 \cdot \pi \cdot x)),$$

que es más sencilla, solo obtendríamos, en el mismo intervalo, 7 elementos no nulos.

Esto último es entendible si vemos el gráfico de esa función en la Figura 2.6 pues se trata de una señal discreta con muy pocos cambios.

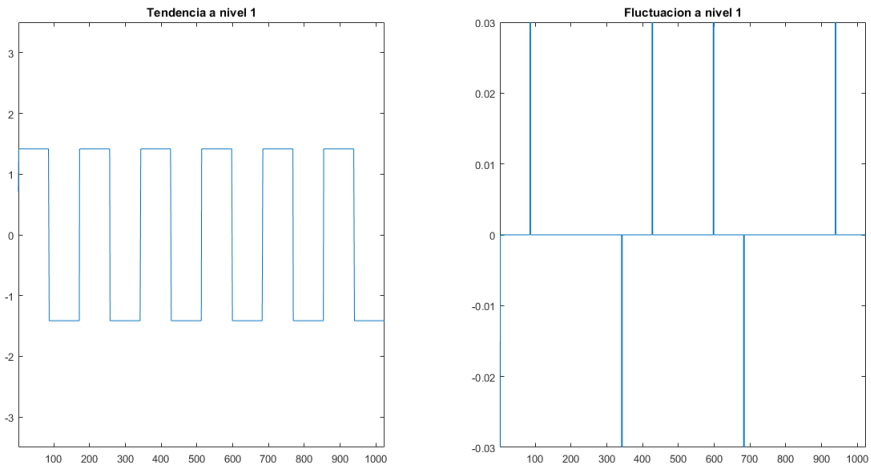

Figura 2.6: Subseñales de tendencia y fluctuación a nivel 1 de la señal $g_2(x)$.

Si queremos conocer la energía que almacena una señal a partir de una transformación, podemos utilizar la función creada en el Anexo B.6. Veamos cuál es la energía de la señal $g_1(x)$ y su transformación en una wavelet de Haar a nivel 1, 2 y 3. Estas se representan en la Figura 2.7.

Notemos que cuanto más alto es el nivel de la señal transformada, el perfil de energía acumulado es mayor, es decir, la energía se acumula más en los valores iniciales. Y, por tanto, la señal se encuentra más compacta.

Si hiciésemos el perfil de energía acumulada con Haar de la señal $g_2(x)$ obtendríamos algo muy similar, con la diferencia de que el perfil de la energía acumulada de la señal original (las primeras dos ventanas) sería una recta diagonal constante que va de 0 a 1, es decir, en ningún momento la energía acumulada llega al máximo sino que todos los puntos aportan energía. Esto se puede ver en la Figura 2.8.

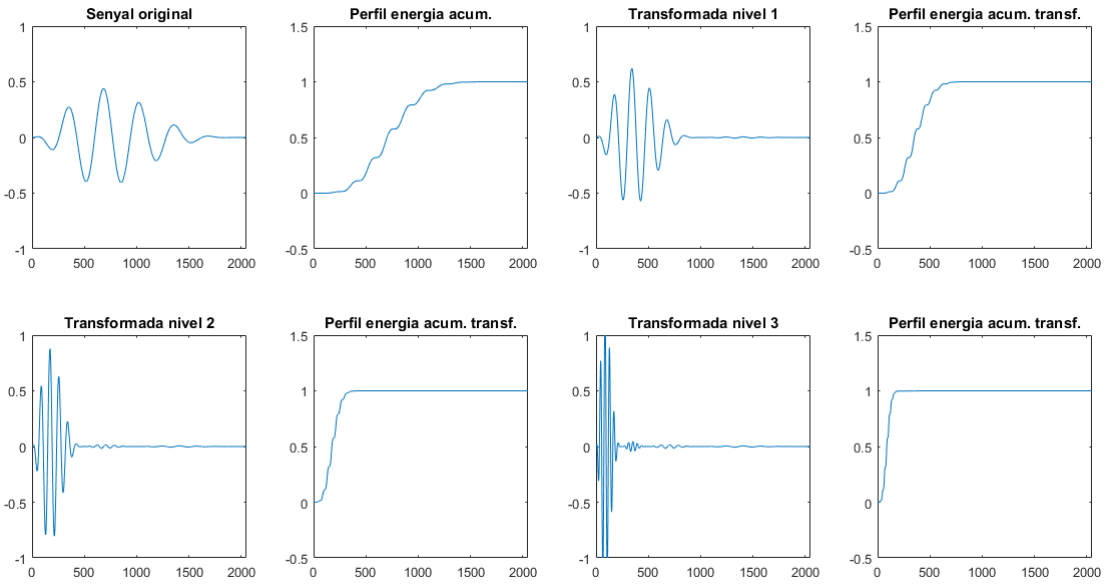

Figura 2.7: Perfil de energía acumulada de la señal $g_1(x)$ y sus respectivas transformaciones de Haar a nivel 1, 2 y 3.

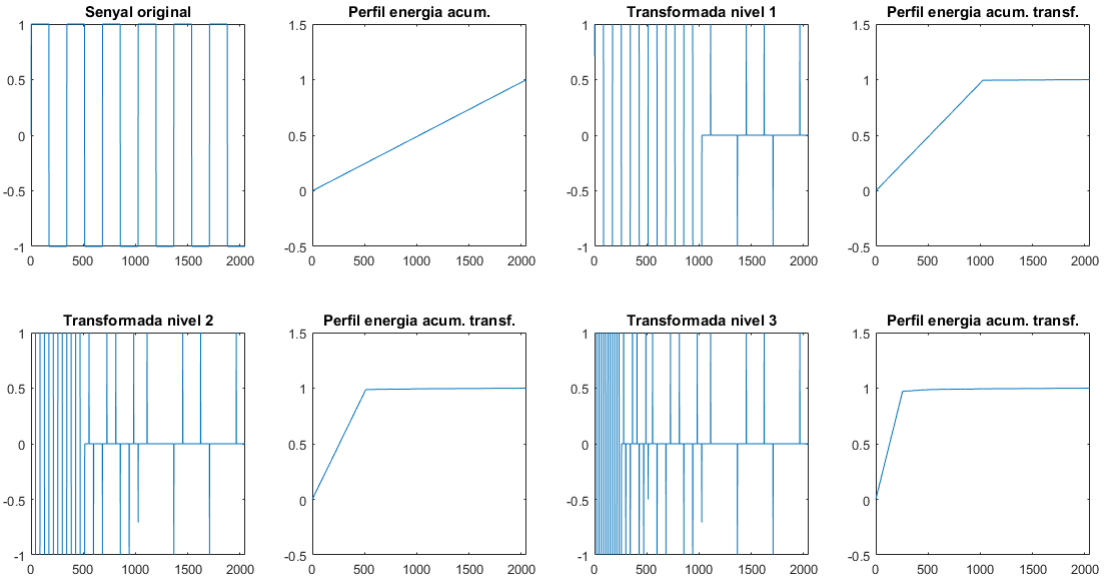

Figura 2.8: Perfil de energía acumulada de la señal $g_2(x)$ y sus respectivas transformaciones de Haar a nivel 1, 2 y 3.

2.5.2. Compresión de señales

A continuación, veamos cómo quedan nuestras dos señales anteriores si las comprimimos al 99.99 %. Para ello, utilizamos el programa que se encuentra en el Anexo B.8 en el que calculamos una transformación de Haar a nivel 3. Obtenemos el gráfico de la Figura 2.9 y los siguientes datos: de una longitud inicial de 2048 valores, solo 586 son significativos. Esto hace que el factor de compresión sea aproximadamente $3:1$. En cambio, si escogemos una Haar de nivel 10, los valores significativos no son mucho menos, tan solo 573 pero es lo suficiente para que el factor de compresión pase a ser en torno al $4:1$, es decir, cuatro veces más pequeño.

El error cuadrático medio (RMS) que se comete en este último caso es de $3.899099 \cdot 10^{-5}$.

A efectos visuales, se puede decir que la señal es prácticamente igual. En cambio, si hacemos exactamente el mismo proceso pero con la señal $g_2(x)$, obtenemos el resultado de la Figura 2.10.

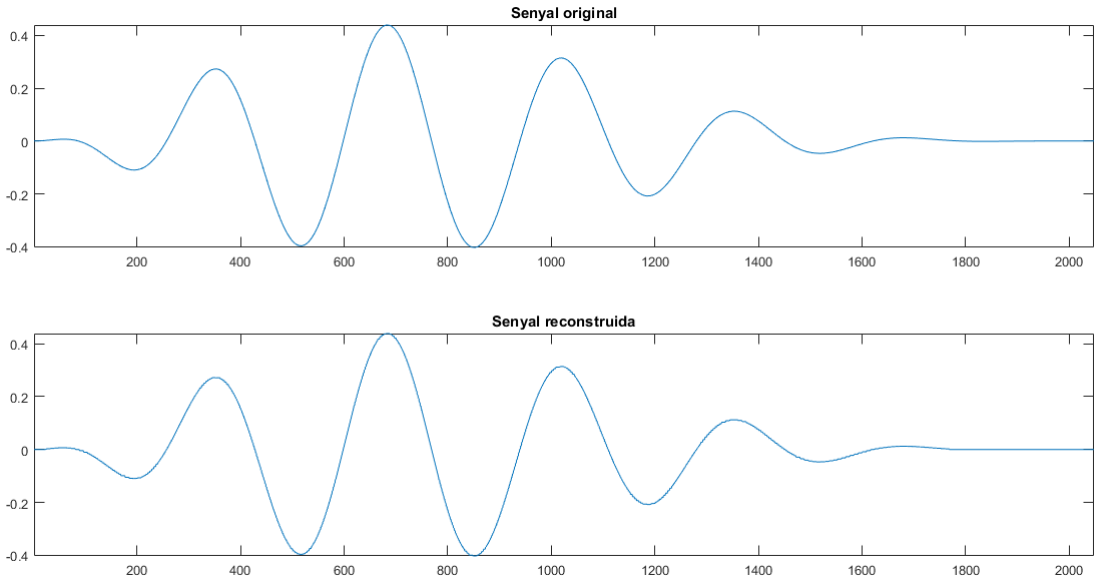

Figura 2.9: Perfil de la señal $g_1(x)$ y su reconstrucción al 99.99 % utilizando una transformación de Haar a nivel 3.

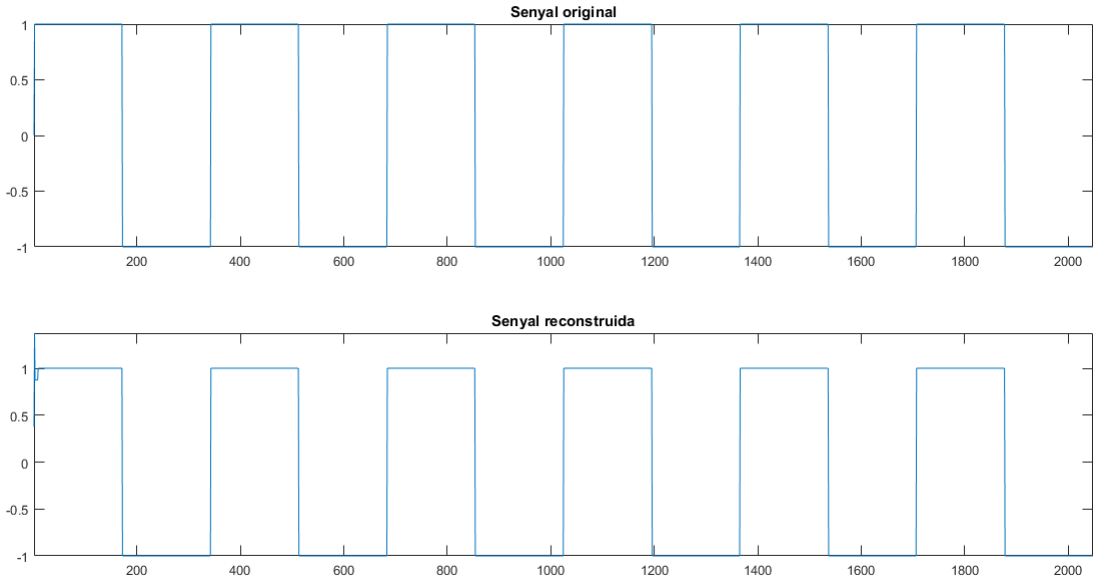

Figura 2.10: Perfil de la señal $g_2(x)$ y su reconstrucción al 99.99 % utilizando una transformación de Haar a nivel 3.

En la segunda imagen se puede apreciar que es una reconstrucción casi idéntica de la señal $g_2(x)$ excepto en los primeros 10 puntos (aproximadamente) donde la señal difiere de la original hasta que se estabiliza y es, entonces, cuando se parece a ella. En este caso, se han reducido 2048 valores significativos a 279, lo que implica un fator de compresión aproximado de $7:1$ con 3 niveles Haar. El error cuadrático medio cometido en este caso ha sido de $2.9901 \cdot 10^{-4}$.

En cambio, si hubiésemos usado un nivel de Haar 10, los valores significativos hubiesen sido tan solo 66 y su factor de compresión $31:1$ con un error de $2.511914 \cdot 10^{-4}$ que no es muy distinto del anterior.

La diferencia entre ambos se puede apreciar en la Figura 2.11. Y es que, mientras la anterior reconstrucción el error se acumulaba al principio, en esta es menor al principio pero hay un ligero salto en otro valor (cerca del 130).

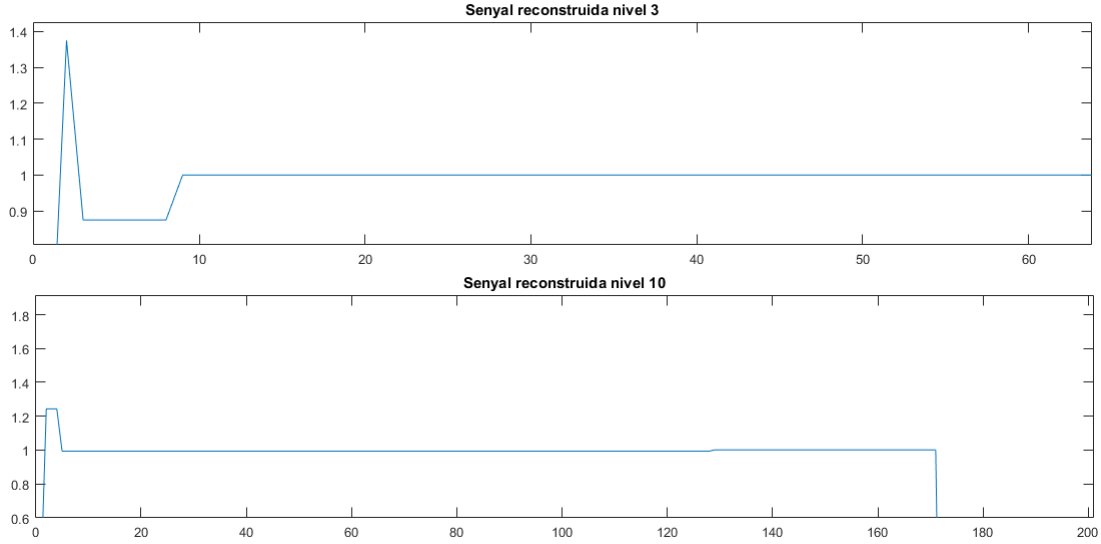

Figura 2.11: Zoom en el perfil de la señal $g_2(x)$ reconstruída al 99.99 % utilizando una transformación de Haar a nivel 3 (arriba) y nivel 10 (abajo).

2.5.3. Wavelets de Daubechies

En la sección anterior, hemos tratado una señal usando la wavelet de Haar. A continuación, analizaremos la siguiente señal usando la wavelet de Daubechies *db2*.

En el programa del Anexo B.9 se toma la siguiente señal

$$g_3(x) = 20x^2(1-x)^2 cos(64\pi x) + 30x^2(1-x)^4 sin(30\pi x)$$

definida en el intervalo $[0, 1]$ con $2^{14} = 16384$ datos.

En primer lugar, extraemos la señal de tendencia de nivel 3 para la wavelet $db2$ y se hace lo mismo con las wavelets $db3$ y $db10$. A continuación, se representa la señal $2\sqrt{2}g_3(8x)$ en el intervalo $[0, 1]/8$ con $2^{14}/2^3 = 2^{11}$ datos. Y, finalmente, se calcula el error máximo y el error cuadrático medio entre las wavelets y esta señal modificada. En la Figura 2.12 se pueden ver los gráficos de este programa.

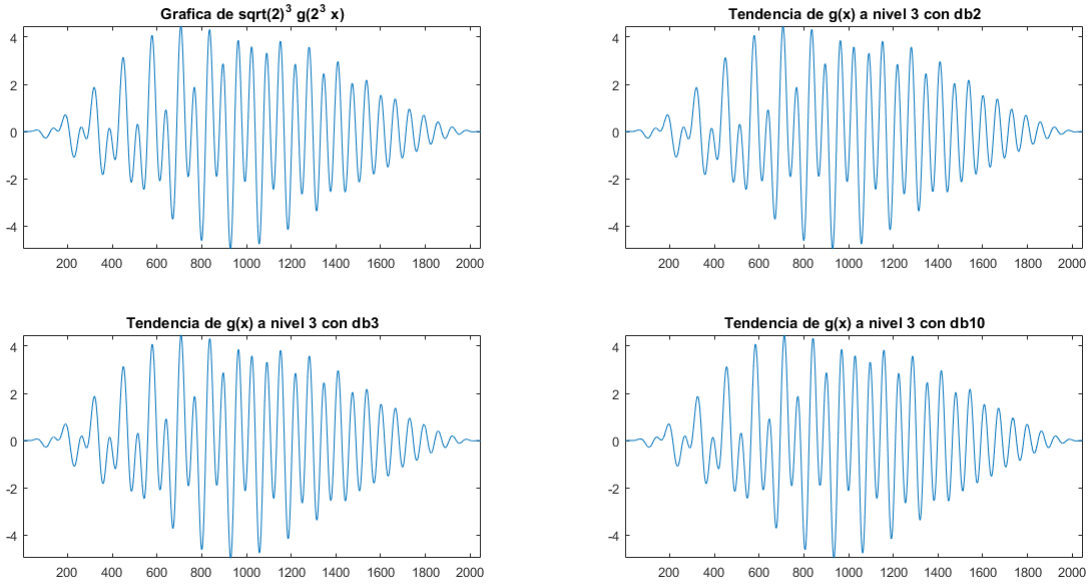

Figura 2.12: Representación de la señal señal $2\sqrt{2}g_3(8x)$ y las wavelets $db2$, $db3$ y $db10$ a nivel 3 de $g_3(x)$.

Aunque visualmente parecen iguales, en realidad el error que se comete al aproximar la señal es mayor cuanto más restricciones tiene la wavelet que utilizamos.

En la Tabla 2.1 se visualiza este hecho. De ella, se extrae la conclusión que la tendencia de las wavelets de Daubechies reproducen peor la señal cuanto mayor es el número de elementos con los que trabajan.

Tipo de error	Valor
Error máximo para la db2	$1.294098 \cdot 10^{-1}$
Error RMS para la db2	$1.133251 \cdot 10^{-3}$
Error máximo para la db3	$4.184300 \cdot 10^{-1}$
Error RMS para la db3	$3.661706 \cdot 10^{-3}$
Error máximo para la db10	2.398573
Error RMS para la db10	$2.098299 \cdot 10^{-2}$

Tabla 2.1: Errores de las wavelets de Daubechies 2, 3 y 10 de $g_3(x)$ a nivel 3 comparada con $2\sqrt{2}g_3(8x)$.

2.5.4. Reducción de ruido

Vamos ahora a reducir el ruido de una señal. Para ello, generaremos, en primer lugar, ruido blanco con el programa que se encuentra en el Anexo B.10. Y, a continuación, calculamos la señal transformada utilizando una wavelet de $Haar$, $db2$ y $db10$ de nivel 3 y 7. Obtendremos la desviación típica, σ, de una porción, fijaremos el umbral y eliminaremos aquellos valores que no lo superen. Finalmente, reconstruiremos la señal y la compararemos con la original. Todo esto se encuentra en el programa del Anexo B.11.

La función a la que vamos a agregar ruido y posteriormente reconstruir es

$$g_4(x) = 4x^2(1-x)^3(2-x)^2 cos(18x(1+x))$$

en el intervalo $[0, 2]$. Se añadirá un ruido blanco gaussiano de 25 dBW de potencia. Y se puede apreciar en la Figura 2.13.

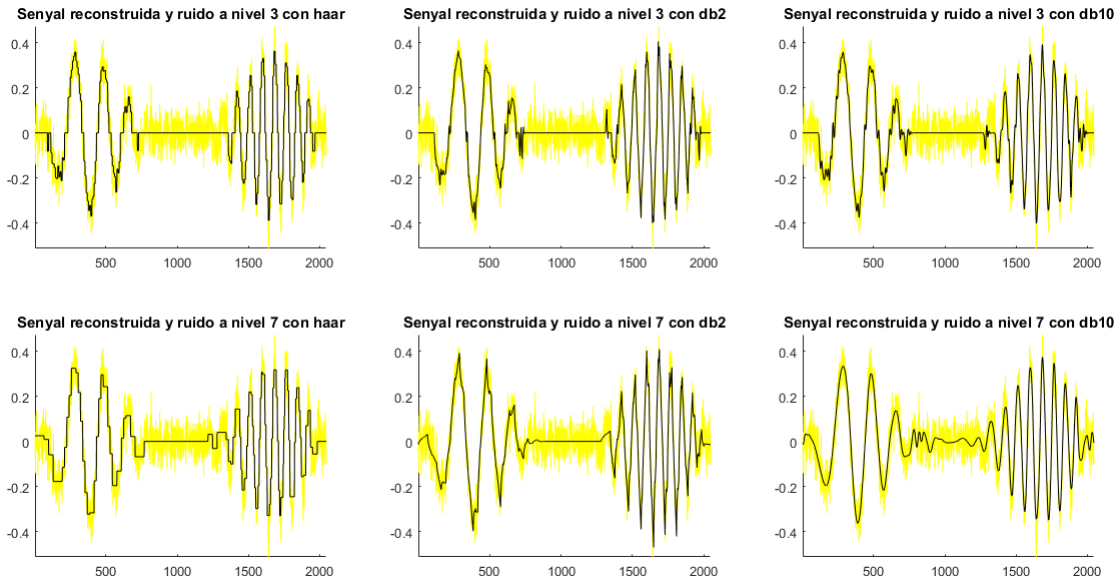

Figura 2.13: Señal $g_4(x)$ contaminada (amarillo). Superpuesta la señal reconstruida después de reducir el ruido (negro) utilizando wavelets de $Haar$, $db2$ y $db10$ a nivel 3 y 10. Se utiliza una umbralización fuerte.

De las figuras se puede deducir que la wavelet de $Haar$ reconstruye peor la señal que una Daubechies. Hemos comprobado el resultado teórico que se explicó cuando se hablaba de estas wavelets. Pero de entre $db2$ y $db10$, es esta última la que mejor aproxima pues lo hace más suave que la primera. Nuevamente, hemos comprobado que una Daubechies con mayor número de restricciones es la que mejor elimina el ruido. En la Tabla 2.2 se puede ver el porcentaje de energía acumulada al reconstruir la señal y el error RMS cometido respecto a la original.

Wavelet	Nivel	Error o energía	Valor
Haar	3	Error RMS	$1.342367 \cdot 10^{-3}$
	3	Energía acumulada	85.628 %
	7	Error RMS	$1.395766 \cdot 10^{-3}$
	7	Energía acumulada	84.461 %
db2	3	Error RMS	$1.275872 \cdot 10^{-3}$
	3	Energía acumulada	87.016 %
	7	Error RMS	$1.309344 \cdot 10^{-3}$
	7	Energía acumulada	86.326 %
db10	3	Error RMS	$1.297435 \cdot 10^{-3}$
	3	Energía acumulada	86.574 %
	7	Error RMS	$1.254629 \cdot 10^{-3}$
	7	Energía acumulada	87.445 %

Tabla 2.2: Error RMS y energía al reconstruir la señal $g_4(x)$ con wavelets de $Haar$, $db2$ y $db10$ a nivel 3 y 7 utilizando una umbralización fuerte.

A pesar de que visualmente sí encontramos diferencias notables entre el uso de una wavelet u otra, cuando miramos la energía que consumen y el error cometido de una wavelet a otra, no observamos grandes diferencias. Es más, cuanto más nivel tiene la wavelet, peor es el error cometido y menor es, también, la energía acumulada; excepto en el caso de la $db10$ que mejora. En este último caso hemos utilizado una umbralización fuerte.

Capítulo 3

Conclusiones

Finalmente, en este último capítulo se realizará un breve resumen a través de una valoración personal tanto de mi estancia en prácticas como del trabajo de fin de grado.

3.1. Estancia en prácticas

Durante mi estancia en Soluciones Cuatroochenta he puesto en práctica muchos de los conocimientos adquiridos durante mis estudios en el grado de Matemática Computacional. Sin embargo, también he podido continuar formándome pues en la carrera no se enseñan bases de datos no relacionales que suponían una herramienta fundamental para llevar a cabo mi estancia; por lo que invertí una buena parte de mis prácticas en aprender y entender cómo funcionaban este tipo de programas con sus respectivos lenguajes.

Además, también aprendí nuevos lenguajes de programación que no solo me fueron útiles durante mi estancia sino también de cara a un futuro profesional con el que continuar creciendo en el sector de la programación. Es más, no solo aprendí lenguajes de programación sino herramientas necesarias para el controlar versiones de un proyecto y comprobar el buen funcionamiento de las aplicaciones programadas.

Logré cumplir los objetivos propuestos antes de finalizar mi estancia, por lo que no solo tuve tiempo de aprender la parte *back-end* del proyecto sino también la parte *front-end* ampliando mi formación en la programación.

A pesar de que al principio no tenía compañero para el desarrollo del proyecto más que a mi supervisor, con el tiempo se incorporó una nueva miembro al equipo. Por tanto, no solo pude desarrollar mis funciones trabajando en solitario sino también en equipo. Por supuesto, el ambiente en la empresa ha sido excelente para poder lograr este estado ya que cualquier duda que tenía podía consultársela a los demás compañeros de la empresa.

Puedo concluir que mi estancia en prácticas ha sido satisfactoria no solo por haber logrado cumplir los objetivos propuestos antes del tiempo estimado sino por haber podido aprender nuevos lenguajes y herramientas de programación que pueden servirme de ayuda en mi futuro profesional.

3.2. Memoria TFG

El propósito de este trabajo era detallar conceptos básicos sobre la teoría de wavelets, concretamente las de Haar y Daubechies. Durante la memoria, he aplicado conceptos aprendidos en las asignaturas de álgebra lineal y análisis durante el grado en Matemática Computacional.

Una parte fundamental del trabajo ha consistido en el desarrollo de la construcción de las wavelets de Haar ya que las de Daubechies se construyen de forma similar. Además, se han analizado las principales propiedades de dichas wavelets. Mientras que las wavelets de Haar son útiles para encontrar "picos" en señales, las de Daubechies son las más adecuadas para tratar señales "suaves".

Además, se vio que el análisis de multirresolución es la principal aplicación de las wavelets y que este permite descomponer una señal en subseñales (de promedio y detalle) por estar estas constituidas por bases ortonormales. Esta misma propiedad permite recuperar la señal original aunque con menor "nitidez".

También se estudió como ambas transformadas, tanto la de Haar como la de Daubechies, permiten comprimir una señal conservando su energía y cumpliendo la propiedad de fluctuaciones pequeñas. Asimismo se ha estudiado la capacidad de las wavelets para comprimir señales y reducir el ruido en las mismas.

Finalmente, haciendo uso del software Matlab se han puesto en práctica todos los conceptos aprendidos. Más concretamente, se han tomado señales concretas y se ha llevado a cabo su descomposición en subseñales, utilizando tanto la wavelet de Haar como la de Debauchies, y se ha eliminado el ruido de señales haciendo uso del Matlab.

Este trabajo no solo me ha ayudado a entender los conceptos más elementales en un campo tan interesante como es el tratamiento de señales sino que también he aprendido a redactar documentos matemáticos y a ampliar mis conocimientos de *LaTeX* [8].

Bibliografía

[1] Algoritmos para reducción de ruido en señales. http://catarina.udlap.mx/u_dl_a/tales/documentos/lem/hernandez_d_m/capitulo3.pdf.

[2] Apache cassandra. https://cassandra.apache.org.

[3] Apache spark. https://spark.apache.org.

[4] D3.js. https://d3js.org.

[5] Datastax academy. https://academy.datastax.com.

[6] Git. https://git-scm.com.

[7] Jetbrains: Webstorm. https://www.jetbrains.com/webstorm.

[8] Latex. https://www.latex-project.org/.

[9] Matlab. https://es.mathworks.com.

[10] Mockaroo. https://www.mockaroo.com.

[11] Node.js. https://nodejs.org/en.

[12] Postman. https://www.getpostman.com.

[13] Soluciones Cuatroochenta S.L. http://www.cuatroochenta.com.

[14] Sourcetree. https://www.sourcetreeapp.com.

[15] Udemy academy. https://www.udemy.com.

[16] Wikipedia: Daubechies wavelet. https://en.wikipedia.org/wiki/Daubechies_wavelet.

[17] Wikipedia: Isometry. https://en.wikipedia.org/wiki/Isometry.

[18] Wikipedia: Wavelet. https://en.wikipedia.org/wiki/Wavelet.

[19] Elena Rupérez Cerezo. Wavelets. http://eprints.ucm.es/16705/1/SI_Elena_Rup%C3%A9rez_Cerezo_Jun12.pdf. [Universidad Complutense (Madrid)].

[20] Félix Martínez Giménez, Alfredo Peris Maguillot & Francisco Rodenas Escribá. *Tratamiento de Señales Digitales Mediante Wavelets y su Uso con MATLAB*. [Valencia (España) 2004].

[21] Thiago L. T. Da Silveira & Alice J. Kozakevicius. Transformada wavelet de haar. conceitos, formulações e aplicações. https://www.sbm.org.br/coloquio-sul-4/wp-content/uploads/sites/4/2016/04/Minicurso_Transformada_wavelet.pdf.

Anexo A

Algunos programas de la Estancia en Prácticas

A.1. generateData.py

```
#encoding utf-8
import csv, random, uuid

def readFileToMatrix(fileName):
    ListRowData = []
    with open(fileName, newline='', encoding='utf-8') as csvfile:
        file = csv.reader(csvfile, delimiter=',')
        for row in file:
            ListRowData.append([elem for elem in row])
    return ListRowData

def getColumnFromMatrix(column, matrixData):
    listColumn = []
    for i in range(len(matrixData)):
        if(len(matrixData[i]) > 0):
            listColumn.append(matrixData[i][column])
    return listColumn

def notRepeatedElements(listData, capitalist=True):
    if capitalist:
        capitalList = [e.capitalize() for e in listData]
        return list(set(capitalList))
    else:
        return set(listData)

def saveAColumnToFile(listData, fileName):
```

```
27      with open(fileName, 'w', newline='', encoding='utf-8') as csvfile:
28          writer = csv.writer(csvfile, lineterminator='\n', delimiter='\n')
29          writer.writerow(listData)
30
31  def appendRawNumbers(PercentMaxRaw, NumberRawTupleList, matrixData, header=False)
        :
32      if(header):
33          matrixData[0].append('RANDOM_NUMBER')
34          row = 1
35      else:
36          row = 0
37      nRows = len(matrixData)
38      while(row<nRows):
39          for i in range(len(PercentMaxRaw)):
40              minN, maxN = NumberRawTupleList[i]
41              numberInserts = round(PercentMaxRaw[i]/100 * nRows)
42              for _ in range(numberInserts):
43
44                  if(row<nRows):
45                      matrixData[row].append(random.randrange(minN, maxN))
46                      row += 1
47      return matrixData
48
49  def saveMatrixToFile(matrixData, fileName):
50      with open(fileName, 'w', newline='', encoding='utf-8') as csvfile:
51          writer = csv.writer(csvfile, lineterminator='\n', delimiter=',')
52          for elem in matrixData:
53              writer.writerow(elem)
54
55  def changeIdToUUID(matriz, col, header=False):
56      prevElem = matriz[1][col]
57      prevUUID = uuid.uuid4()
58      if(header):
59          row = 1
60      else:
61          row = 0
62      for i in range(row, len(matriz)):
63          if matriz[i][col] == prevElem:
64              matriz[i][col] = prevUUID
65          else:
66              prevElem = matriz[i][col]
67              matriz[i][col] = uuid.uuid4()
68              prevUUID = matriz[i][col]
69      return matriz
70
71  def cartesianProductTicketLine(matrixToChange, matrixFrom, header=False):
72      j = 0
73      initial = 0
74      prevTicketUUID = matrixToChange[initial][0]
75      prevTime = matrixFrom[j][1]
76      prevUser = matrixFrom[j][2]
77      prevSuper = matrixFrom[j][3]
```

```
78      if(header):
79          matrixToChange[initial].append(prevTime)
80          matrixToChange[initial].append(prevUser)
81          matrixToChange[initial].append(prevSuper)
82          initial = 1
83          j = 1
84      for i in range(initial, len(matrixToChange)):
85          if matrixToChange[i][0] != prevTicketUUID:
86              prevTicketUUID = matrixToChange[i][0]
87              prevTime = matrixFrom[j][1]
88              prevUser = matrixFrom[j][2]
89              prevSuper = matrixFrom[j][3]
90              j += 1
91          matrixToChange[i].append(prevTime)
92          matrixToChange[i].append(prevUser)
93          matrixToChange[i].append(prevSuper)
94      return matrixToChange
95
96  def transformTicketLineIntoAnotherMatrix(matrix, header=False):
97      m = []
98      list = [0, 0, 0, 0, 0, 0]
99      initial = 0
100     if(header):
101         m.append(['TICKET_ID', 'TIME', 'USER_ID', 'SUPERMARKET_ID', '
        DISCOUNT_TICKET', 'TOTAL_TICKET'])
102         initial = 1
103     list[0] = matrix[initial][0]  # ticket_id
104     list[1] = matrix[initial][2]  # time
105     list[2] = matrix[initial][3]  # user_id
106     list[3] = matrix[initial][4]  # supermarket_id
107     list[4] = 0                   # dicount_ticket
108     list[5] = float(matrix[initial][8])  # total
109     for i in range(initial+1, len(matrix)):
110         if matrix[i][0] == list[0]:
111             list[5] += float(matrix[i][8])
112         else:
113             m.append(list)
114             list = [0]*6
115             list[0] = matrix[i][0]  # ticket_id
116             list[1] = matrix[i][2]  # time
117             list[2] = matrix[i][3]  # user_id
118             list[3] = matrix[i][4]  # supermarket_id
119             list[4] = 0             # dicount_ticket
120             list[5] = float(matrix[i][8]) # total
121     m.append(list)
122     return m
123
124 def addTotalCol(matrix, header=False):
125     initial = 0
126     if(header):
127         matrix[initial].append('TOTAL_LINE')
128         initial = 1
```

```
    for i in range(initial, len(matrix)-1):
        total = int(matrix[i][5])*float(matrix[i][6])*(100-int(matrix[i][7]))/100
        matrix[i].append(total)
    return matrix

########################
######## MAIN #########
########################
# Comentar las lineas de codigo que no se vayan a ejecutar. Ejecutar por bloques. Seguir instrucciones.

if __name__ == "__main__":
  # Creo columna random en TICKET_LINE y generar un UUID.
  # Despues extraere esa columna y la adjuntare a TICKET manualmente.
  # Finalmente hare el producto entre ambas para guardar los datos de ticket en ticket_line

    ticket_line = readFileToMatrix('TICKET_LINE.csv')
    NumeroFilas = [len(ticket_line)]
    MinMaxNumeroUUIDs = [(1,2000)]
    ticket_line_withRandomCol = appendRawNumbers(NumeroFilas, MinMaxNumeroUUIDs, ticket_line, True)
    saveMatrixToFile(ticket_line_withRandomCol, 'ticket_line_withRandomCol.csv')

    # Hay que ordenar manualmente el archivo por UUID de mayor a menor

  ticket_line_withRandomCol = readFileToMatrix('TICKET_LINE.csv')
  ticket_line_withUUID = changeIdToUUID(ticket_line_withRandomCol, 0, True)
  saveMatrixToFile(ticket_line_withUUID, 'ticket_line_withUUID.csv')
  ticket_Line_UUIDCol = getColumnFromMatrix(0, ticket_line_withUUID)
  UUIDColSet = notRepeatedElements(ticket_Line_UUIDCol, False)
  saveAColumnToFile(UUIDColSet, 'UUID_tickets.csv')

  # Ahora, manualmente, se junta la columna de UUID a la matriz TICKET
  # Actualizamos ambos archivos: TICKET y TICKET_LINE y ambos ordenamos por UUID
  # A continuacion volcamos la info de TICKET a TICKET_LINE

  ticketLine = readFileToMatrix('TICKET_LINE.csv')
  ticket = readFileToMatrix('TICKET.csv')
  newTicketLine = cartesianProductTicketLine(ticketLine, ticket, True)
  saveMatrixToFile(newTicketLine, 'TICKET_LINE_completed.csv')

  # Ahora volvemos a ordenar las columnas de la tabla TICKET_LINE
  # Calculamos bien la columna TOTAL_LINE por si hubiese algun fallo previo

  ticketLine = readFileToMatrix('TICKET_LINE.csv')
  ticketLine_withNewTotal = addTotalCol(ticketLine, True)
  saveMatrixToFile(ticketLine_withNewTotal, 'TICKETLINE_completedNewTotal.csv')

  # Arreglo el archivo TICKET_LINE y calculo a mano el ultimo valor porque daba error

```

```
  ticketLine = readFileToMatrix('TICKET_LINE.csv')
  ticket = transformTicketLineIntoAnotherMatrix(ticketLine, True)
  saveMatrixToFile(ticket, 'TICKET_completed.csv')

  # Finalmente obtenemos ahora el nuevo CSV TICKET.
```

A.2. paths.js

```
var path = require('path'),
  csv = require('csvtojson'),
  moment = require('moment'),
  saveJSON = require('./saveJSON'),
  jsonToCsv = require('./JSONToCSVConvertor.js');
  // Main table CSVs
  // Not showed
  // JSON files
  // Not showed

module.exports = {
  csv,
  moment,
  saveJSON,
  jsonToCsv
  // Main table CSVs
  // Not showed
  // JSON files
  // Not showed
}
```

A.3. JSONToCSVConvertor.js

```
var fs = require('fs');

function JSONToCSVConvertor_separateKeyVector(JSONData, fileName,
    vectorMaxKeyLength) { // { [item1, ..., itemN ?]: data or [data]
    var dicData = typeof JSONData != 'object' ? JSON.parse(JSONData) : JSONData;

    var CSV = '';

    for (var key in dicData) {
        let row = `${key.substring(0, vectorMaxKeyLength[0])},`;
        let values = [];
        for (let i=1; i<vectorMaxKeyLength.length; i++) {
            row += `${key.substring(vectorMaxKeyLength[i-1],vectorMaxKeyLength[i
    ])},`;
        }
        if(Array.isArray(dicData[key])){
            for (var value in dicData[key]) {
```

```
                row += '"' + dicData[key][value] + '",';
            }
        } else {
            row += '"' + dicData[key] + '",';
        }
        row = row.slice(0,-1);
        CSV += row + '\n';
    }

    if (CSV == '') {
        alert("Invalid data");
        return;
    }

    fs.writeFileSync(`./../filesCSV/${fileName}.csv`, CSV, 'utf8', function(err)
    {
        if (err) {
            console.log('Unable to write in file');
        }
    })
};

function JSONToCSVConvertor_separateKeyVectorWithoutFinal(JSONData, fileName,
    vectorMaxKeyLength) { // { [item1, ..., itemN]: data or  [data]
    var dicData = typeof JSONData != 'object' ? JSON.parse(JSONData) : JSONData;

    var CSV = '';

    for (var key in dicData) {
        let row = `${key.substring(0, vectorMaxKeyLength[0])},`;
        let values = [];
        vectorMaxKeyLength[vectorMaxKeyLength.length-1] = key.length;
        for (let i=1; i<vectorMaxKeyLength.length; i++) {
            row += `${key.substring(vectorMaxKeyLength[i-1],vectorMaxKeyLength[i
    ])},`;
        }
        if(Array.isArray(dicData[key])){
            for (var value in dicData[key]) {
                row += '"' + dicData[key][value] + '",';
            }
        } else {
            row += '"' + dicData[key] + '",';
        }
        row = row.slice(0,-1);
        CSV += row + '\n';
    }

    if (CSV == '') {
        alert("Invalid data");
        return;
    }

```

```
    fs.writeFileSync(`./../filesCSV/${fileName}.csv`, CSV, 'utf8', function(err)
    {
        if (err) {
            console.log('Unable to write in file');
        }
    })
};

module.exports = {
    JSONToCSVConvertor_separateKeyVector,
    JSONToCSVConvertor_separateKeyVectorWithoutFinal
}
```

A.4. saveJSON.js

```
var fs = require('fs');

function saveJSON(JSONData, fileName) {
    var dicData = typeof JSONData != 'object' ? JSON.parse(JSONData) : JSONData;

    fs.writeFileSync(`./../filesJSON/${fileName}.json`, JSON.stringify(dicData),
    'utf8', function(err) {
        if (err) {
            console.log('Unable to write in file');
        }
    })
};

module.exports = {
    saveJSON
}
```

Anexo B

Programas de Matlab sobre la Memoria del TFG

B.1. funcionG.m

```
function [y] = funcionG(x)
    y = 20*x^2*(1-x)^4*cos(12*pi*x);
end
```

B.2. obtenerValoresFuncionG.m

```
function matrix = obtenerValoresFuncionG(a,b,n)
    matrix = zeros(n,1);
    puntos = linspace(a,b,n); %n puntos en [a,b]
    for i=1:n
        matrix(i,1) = funcionG(puntos(i));
    end
end
```

B.3. analisisSenyal.m

```
clear;
s = obtenerValoresFuncionG(0,1,2048); %Senyal
w = 'haar'; %Wavelet
n = 3; %Niveles
[C,L] = wavedec(s,n,w); %Descomposicion de la senyal

```

```
%C : tendencia & fluctuacion
cA1 = appcoef(C,L,w,1); cA2 = appcoef(C,L,w,2); cA3 = appcoef(C,L,w,3);
cD1 = detcoef(C,L,1);   cD2 = detcoef(C,L,2);   cD3 = detcoef(C,L,3);

%Senyales promedio y detalles
A1 = wrcoef('a',C,L,w,1); A2 = wrcoef('a',C,L,w,2); A3 = wrcoef('a',C,L,w,3);
D1 = wrcoef('d',C,L,w,1); D2 = wrcoef('d',C,L,w,2); D3 = wrcoef('d',C,L,w,3);

%Representacion grafica
figure;
subplot(3,4,1); plot(cA1); title('Tendencia a nivel 1');
axis([1 length(s)/2 -1.5 1.5]);
subplot(3,4,2); plot(cD1); title('Fluctuacion a nivel 1');
axis([1 length(s)/2 -0.03 0.03]);
subplot(3,4,5); plot(cA2); title('Tendencia a nivel 2');
axis([1 length(s)/2 -1.5 1.5]);
subplot(3,4,6); plot(cD2); title('Fluctuacion a nivel 2');
axis([1 length(s)/2 -0.03 0.03]);
subplot(3,4,9); plot(cA3); title('Tendencia a nivel 3');
axis([1 length(s)/2 -1.5 1.5]);
subplot(3,4,10); plot(cD3); title('Fluctuacion a nivel 3');
axis([1 length(s)/2 -0.03 0.03]);
subplot(3,4,3); plot(A1); title('Promedio a nivel 1');
axis([1 length(s) -1.5 1.5]);
subplot(3,4,4); plot(D1); title('Detalle a nivel 1');
axis([1 length(s) -0.02 0.02]);
subplot(3,4,7); plot(A2); title('Promedio a nivel 2');
axis([1 length(s) -1.5 1.5]);
subplot(3,4,8); plot(D2); title('Detalle a nivel 2');
axis([1 length(s) -0.02 0.02]);
subplot(3,4,11); plot(A3); title('Promedio a nivel 3');
axis([1 length(s) -1.5 1.5]);
subplot(3,4,12); plot(D3); title('Detalle a nivel 3');
axis([1 length(s) -0.02 0.02]);

%Analisis de energia
C1 = wavedec(s,1,w);
C2 = wavedec(s,2,w);

subplot(2,4,1);
plot(s);
axis([0 length(s) -1 1]);
title('Senyal original');
subplot(2,4,2);
plot(energiaAcumulada(s));
axis([0 length(s) -0.5 1.5]);
title('Perfil energia acum.');
subplot(2,4,3);
plot(C1);
axis([0 length(s) -1 1]);
title('Transformada nivel 1');
subplot(2,4,4);
```

```
plot(energiaAcumulada(C1));
axis([0 length(s) -0.5 1.5]);
title('Perfil energia acum. transf.');
subplot(2,4,5);
plot(C2);
axis([0 length(s) -1 1]);
title('Transformada nivel 2');
subplot(2,4,6);
plot(energiaAcumulada(C2));
axis([0 length(s) -0.5 1.5]);
title('Perfil energia acum. transf.');
subplot(2,4,7);
plot(C);
axis([0 length(s) -1 1]);
title('Transformada nivel 3');
subplot(2,4,8);
plot(energiaAcumulada(C));
axis([0 length(s) -0.5 1.5]);
title('Perfil energia acum. transf.');
```

B.4. mapaSignificancia.m

```
function [v] = mapaSignificancia(x)
    v = x ~= 0;
end
```

B.5. energia.m

```
function [y] = energia(x)
   y = norm(x)^2;
end
```

B.6. energiaAcumulada.m

```
function [y] = energiaAcumulada(x)
   y = cumsum(x.^2)/(norm(x)^2);
end
```

B.7. rms.m

```
function [e] = rms(x, y)
   e = sqrt(norm(x-y)^2)/length(x);
end
```

B.8. comprimir.m

```
clear;
s = obtenerValoresFuncionG(0,1,2048); % Senyal
ls = length(s);
w = 'haar'; % Wavelet
n = 3; % Niveles
thr_met = 'h'; % Metodo threslholding
en_cons = 0.9999; % Energia a conservar
[C,L] = wavedec(s,n,w); % Descomposicion

C_dec = abs(sort(-abs(C))); % Reordenamos en val. abs.
ind_sobran = find(energiaAcumulada(C_dec)>=en_cons);
umbral = C_dec(ind_sobran(1)); % Valor umbral (threshold)

% Anulamos valores de la transformada que no pasan el umbral
C_sob = wthresh(C,thr_met,umbral);
s_rec = waverec(C_sob,L,w); % Recomponemos la senyal

% Grafico de las dos senyales
figure;
subplot(2,1,1); plot(s);
axis([1 ls min(s) max(s)]); title('Senyal original');
subplot(2,1,2); plot(s_rec);
axis([1 ls min(s_rec) max(s_rec)]); title('Senyal reconstruida');

% Resumen de resultados
error = rms(s,s_rec);
map = mapaSignificancia(C_sob);
val_sig = sum(map);
[comp,err] = sprintf('%d:%d',ls,val_sig);
[comp_ap,err] = sprintf('%d:%d',round(ls/val_sig),1);

[l_long_orig,err] = sprintf('Longitud original: %d \n',ls);
[l_val_sig,err] = sprintf('Valores significativos: %d \n',val_sig);
[l_comp,err] = sprintf('Factor compresion de %s \n',comp);
[l_comp_ap,err] = sprintf('\t (aproximadamente de %s) \n',comp_ap);
[l_rms,err] = sprintf('Error RMS: %d \n',error);
sprintf('%s%s%s%s%s',...
l_long_orig,l_val_sig,l_comp,l_comp_ap,l_rms)
```

B.9. analisisSenyalDb.m

```
clear; dwtmode('per');
% Senyal(t)
t1 = 0:(1-0)/(2^14-1):1;
s1 = 20*(t1.^2).*((1-t1).^2).*cos(64*pi*t1)+...
    30*(t1.^2).*((1-t1).^4).*sin(30*pi*t1);
n = 3; % Niveles
w1 = 'db2'; % Wavelet
```

```
[C1,L1] = wavedec(s1,n,w1); %Descomposicion de la senyal
tend_w1 = appcoef(C1,L1,w1,n); %Tendencia

%Otras dos wavelets
w2 = 'db3'; %Wavelet
[C2,L2] = wavedec(s1,n,w2); %Descomposicion de la senyal
tend_w2 = appcoef(C2,L2,w2,n); %Tendencia
w3 = 'db10'; %Wavelet
[C3,L3] = wavedec(s1,n,w3); %Descomposicion de la senyal
tend_w3 = appcoef(C3,L3,w3,n); %Tendencia

%(sqrt(2)^n)*Senyal((2^n)*t) en 1/(2^n)*dominio
t2 = t1(1:(2^n):length(t1))/(2^n);
s2 = (sqrt(2)^n)*(20*((2^n*t2).^2)...
.*((1-(2^n*t2)).^2).*cos(64*pi*(2^n*t2))+...
30*((2^n*t2).^2).*((1-(2^n*t2)).^4).*sin(30*pi*(2^n*t2)));

%Grafico
figure;
subplot(2,2,1); plot(s2);
axis([1 length(s2) min(s2) max(s2)]);
[tit,err] = sprintf('Grafica de sqrt(2)^%d g(2^%d x)',n,n);
title(tit);
subplot(2,2,2); plot(tend_w1);
axis([1 length(tend_w1) min(tend_w1) max(tend_w1)]);
[tit_w1,err] = sprintf('Tendencia de g(x) a nivel %d con %s',n,w1);
title(tit_w1);
max_error_w1 = max(abs(tend_w1-s2));
rms_error_w1 = rms(tend_w1,s2);
subplot(2,2,3); plot(tend_w2);
axis([1 length(tend_w2) min(tend_w2) max(tend_w2)]);
[tit_w2,err] = sprintf('Tendencia de g(x) a nivel %d con %s',n,w2);
title(tit_w2);
max_error_w2 = max(abs(tend_w2-s2));
rms_error_w2 = rms(tend_w2,s2);
subplot(2,2,4); plot(tend_w3);
axis([1 length(tend_w3) min(tend_w3) max(tend_w3)]);
[tit_w3,err] = sprintf('Tendencia de g(x) a nivel %d con %s',n,w3);
title(tit_w3);
max_error_w3 = max(abs(tend_w3-s2));
rms_error_w3 = rms(tend_w3,s2);

%Resultados
[l_max_err_w1,err] = sprintf('Error maximo para la %s %d \n',w1,max_error_w1);
[l_rms_err_w1,err] = sprintf('Error rms para la %s %d \n',w1,rms_error_w1);
[l_max_err_w2,err] = sprintf('Error maximo para la %s %d \n',w2,max_error_w2);
[l_rms_err_w2,err] = sprintf('Error rms para la %s %d \n',w2,rms_error_w2);
[l_max_err_w3,err] = sprintf('Error maximo para la %s %d \n',w3,max_error_w3);
[l_rms_err_w3,err] = sprintf('Error rms para la %s %d \n',w3,rms_error_w3);
sprintf('%s%s%s%s%s%s',l_max_err_w1,l_rms_err_w1,...
    l_max_err_w2,l_rms_err_w2,l_max_err_w3,l_rms_err_w3)
```

B.10. hacerRuido.m

```
function [s] = hacerRuido
    a = 0; b = 2; % Intervalo
    N = 2^11; % Cantidad de valores
    x = a:(b-a)/(N-1):b;
    s = (4*(x.^2)).*((1-x).^3).*((2-x).^2).*cos((18*x).*(1+x));
    s = awgn(s,25); % Anyadimos ruido blanco de 25 dB
end
```

B.11. eliminarRuido.m

```
clear;
s = hacerRuido(); % Senyal
w1 = 'haar'; % Wavelet a utilizar
w2 = 'db2';
w3 = 'db10';
n1 = 3; % Niveles de transformada
n2 = 7;
thr_met = 'h'; % Metodo thresholding: Hard

% Numero de veces la desviacion tipica en umbral
coef = sqrt(2*log(length(s)));
[C11,L11] = wavedec(s,n1,w1); % Transformada wavelet
[C12,L12] = wavedec(s,n1,w2);
[C13,L13] = wavedec(s,n1,w3);
[C21,L21] = wavedec(s,n2,w1);
[C22,L22] = wavedec(s,n2,w2);
[C23,L23] = wavedec(s,n2,w3);

% Calculo del umbral (threshold)
cD1_11 = detcoef(C11,L11,1); % Extraemos la primera fluctuacion
cD1_12 = detcoef(C12,L12,1);
cD1_13 = detcoef(C13,L13,1);
cD1_21 = detcoef(C21,L21,1);
cD1_22 = detcoef(C22,L22,1);
cD1_23 = detcoef(C23,L23,1);
des_tip_11 = std(cD1_11); % Desviacion tipica
des_tip_12 = std(cD1_12);
des_tip_13 = std(cD1_13);
des_tip_21 = std(cD1_21);
des_tip_22 = std(cD1_22);
des_tip_23 = std(cD1_23);
thr_11 = coef*des_tip_11; % Umbral (threshold)
thr_12 = coef*des_tip_12;
thr_13 = coef*des_tip_13;
thr_21 = coef*des_tip_21;
thr_22 = coef*des_tip_22;
thr_23 = coef*des_tip_23;
Cthr_11 = wthresh(C11,thr_met,thr_11); % Umbralizamos la transformada
```

```
Cthr_12 = wthresh(C12,thr_met,thr_12);
Cthr_13 = wthresh(C13,thr_met,thr_13);
Cthr_21 = wthresh(C21,thr_met,thr_21);
Cthr_22 = wthresh(C22,thr_met,thr_22);
Cthr_23 = wthresh(C23,thr_met,thr_23);
s_den_11 = waverec(Cthr_11,L11,w1); % Reconstruimos
s_den_12 = waverec(Cthr_12,L12,w2);
s_den_13 = waverec(Cthr_13,L13,w3);
s_den_21 = waverec(Cthr_21,L21,w1);
s_den_22 = waverec(Cthr_22,L22,w2);
s_den_23 = waverec(Cthr_23,L23,w3);

% Graficos
figure; subplot(2,3,1); hold on; axis([1 length(s) min(s) max(s)]);
[tit_11,err] = sprintf('Senyal reconstruida y ruido a nivel %d con %s',n1,w1);
title(tit_11); plot(s','y'); plot(s_den_11','k'); hold off;
subplot(2,3,2); hold on; axis([1 length(s) min(s) max(s)]); plot(s','y');
[tit_12,err] = sprintf('Senyal reconstruida y ruido a nivel %d con %s',n1,w2);
title(tit_12); plot(s_den_12','k'); hold off;
subplot(2,3,3); hold on; axis([1 length(s) min(s) max(s)]); plot(s','y');
[tit_13,err] = sprintf('Senyal reconstruida y ruido a nivel %d con %s',n1,w3);
title(tit_13); plot(s_den_13','k'); hold off;
subplot(2,3,4); hold on; axis([1 length(s) min(s) max(s)]); plot(s','y');
[tit_21,err] = sprintf('Senyal reconstruida y ruido a nivel %d con %s',n2,w1);
title(tit_21); plot(s_den_21','k'); hold off;
subplot(2,3,5); hold on; axis([1 length(s) min(s) max(s)]); plot(s','y');
[tit_22,err] = sprintf('Senyal reconstruida y ruido a nivel %d con %s',n2,w2);
title(tit_22); plot(s_den_22','k'); hold off;
subplot(2,3,6); hold on; axis([1 length(s) min(s) max(s)]); plot(s','y');
[tit_23,err] = sprintf('Senyal reconstruida y ruido a nivel %d con %s',n2,w3);
title(tit_23); plot(s_den_23','k'); hold off;

% Errores y energia
error_11 = rms(s,s_den_11); % Error
error_12 = rms(s,s_den_12);
error_13 = rms(s,s_den_13);
error_21 = rms(s,s_den_21);
error_22 = rms(s,s_den_22);
error_23 = rms(s,s_den_23);
[error_men_11,err] = sprintf('Error: %d \n',error_11);
[error_men_12,err] = sprintf('Error: %d \n',error_12);
[error_men_13,err] = sprintf('Error: %d \n',error_13);
[error_men_21,err] = sprintf('Error: %d \n',error_21);
[error_men_22,err] = sprintf('Error: %d \n',error_22);
[error_men_23,err] = sprintf('Error: %d \n',error_23);
ene_ret_11 = 100*energia(s_den_11)/energia(s); % Energia acumulada
ene_ret_12 = 100*energia(s_den_12)/energia(s);
ene_ret_13 = 100*energia(s_den_13)/energia(s);
ene_ret_21 = 100*energia(s_den_21)/energia(s);
ene_ret_22 = 100*energia(s_den_22)/energia(s);
ene_ret_23 = 100*energia(s_den_23)/energia(s);
[ene_ret_men_11,err] = sprintf('Energia retenida: %2.3f %% \n',ene_ret_11);
```

```
[ene_ret_men_12,err] = sprintf('Energia retenida: %2.3f %% \n',ene_ret_12);
[ene_ret_men_13,err] = sprintf('Energia retenida: %2.3f %% \n',ene_ret_13);
[ene_ret_men_21,err] = sprintf('Energia retenida: %2.3f %% \n',ene_ret_21);
[ene_ret_men_22,err] = sprintf('Energia retenida: %2.3f %% \n',ene_ret_22);
[ene_ret_men_23,err] = sprintf('Energia retenida: %2.3f %% \n',ene_ret_23);
sprintf('%s %s %s %s %s %s %s %s %s %s %s %s',error_men_11,ene_ret_men_11,...
    error_men_12,ene_ret_men_12,error_men_13,ene_ret_men_13,...
    error_men_21,ene_ret_men_21,error_men_22,ene_ret_men_22,...
    error_men_23,ene_ret_men_23)
```