TESIS DOCTORAL

Compresión y restauración de imágenes por técnicas no lineales

Vicente Renau Torres

Dirección: Pep Mulet Mestre y Francesc Aràndiga Llaudes

Universitat de València
València, 2011.

Compresión y restauración de imágenes por técnicas no lineales

Memoria presentada por Vicente Renau Torres, Licenciado en Matemáticas; realizada en el departamento de Matemática Aplicada de la Universidad de Valencia bajo la dirección de Pep Mulet Mestre, Profesor Titular de este departamento, y Francesc Aràndiga Llaudes, catedrático de este departamento, con el objetivo de aspirar al Grado de Doctor en Matemáticas.

Valencia, 24 de junio de 2011

Francesc Aràndiga Director	Pep Mulet Director	Vicente Renau Aspirante al Grado de Doctor

Departamento de Matemática Aplicada
Facultad de Matemáticas
Universidad de Valencia

Agradecimientos

Quiero expresar mi más sincero agradecimiento a mis directores de tesis, Pep Mulet y Paco Aràndiga, por sus ánimos constantes, su confianza y su comprensión a lo largo de estos años. Igualmente les agradezco sus recomendaciones, sus consejos y su ayuda decidida siempre que la he necesitado.

Ellos han guiado esta investigación atendiendo todas mis dudas con infinita paciencia y han hecho posible la realización de este proyecto.

Quiero dar las gracias también a mi familia por su apoyo incondicional. A ellos dedico este trabajo.

Valencia, 2011 *Vicente*

Índice general

Introducción

Esta memoria consta de dos partes: compresión de imágenes y restauración de imágenes.

Empezamos la parte dedicada a la compresión de imágenes haciendo un repaso sobre fundamentos de compresión de imágenes y explicando algunos estándares. Describimos los algoritmos EZW y SPIHT con detalle en el apéndice A.

Hay aplicaciones de compresión de datos, donde el control de calidad es de suma importancia. Ciertas características de la señal decodificada deben ser recuperadas de forma exacta, o con mucha precisión, aunque se prefiriera la eficiencia con respecto a almacenamiento y velocidad de cálculo. En esta tesis presentamos algoritmos multiescala de compresión de datos en el marco de la multiresolución de Harten que dan una determinada estimación del error entre la imagen original y la decodificada, medida en las normas discretas L_∞ y L_2.

El algoritmo bidimensional propuesto, que no se basa en una estrategia de producto tensorial, proporciona cotas a priori del error máximo (PAE), del Error Cuadrático Medio (ECM) y del Root Mean Square Error (RMSE) de la imagen decodificada que dependen de los parámetros de cuantificación. Además, después de la compresión de la imagen mediante la aplicación de esta transformación multiescala no separable, el usuario tiene *el valor exacto de la* PAE y del RMSE antes de que el proceso de decodificación se lleva a cabo. Mostramos cómo esta técnica se puede utilizar para obtener algoritmos de compresión sin pérdida y casi-sin pérdida.

También presentamos algoritmos de multiresolución bidimensionales no separables en el contexto de medias en celda que nos

permiten obtener algoritmos que garantizan *a priori* una cierta calidad en la imagen descomprimida. Por otra parte, después de aplicar el algoritmo conocemos el error exacto entre la imagen original y la imagen descomprimida, medido en la norma discreta L_2, sin que sea necesario descomprimir la imagen.

En [6] se propone la extensión tensorial 2D de interpoladores ponderados ENO (WENO) unidimensionales basados en valores puntuales. Esta interpolación puede genéricamente lograr un orden de precisión de $2r$, al utilizar conjuntos de $2r$ puntos, en regiones en las que la función que se interpola es suave.

En esta tesis se propone un algoritmo de interpolación 2D no separable basado en interpolación 2D ENO ponderada donde los pesos son calculados de manera similar a como se propone en [36]. Además, en esta tesis obtenemos los siguientes resultados para cualquier r, utilizando conjuntos de puntos de $2r \times 2r$ puntos: (1) el orden de interpolación es de $2r$ en regiones suaves; (2) el orden la interpolación es de $r+1$, como en los interpolantes ENO, cuando la función tiene una discontinuidad en el conjunto de $2r \times 2r$ puntos, pero es suave en al menos uno de los subconjuntos de $(r+1)\times(r+1)$ puntos; (3) los pesos óptimos se obtienen en forma cerrada.

Los resultados aquí obtenidos han sido sometidos a publicación en los trabajos siguientes:

- Francesc Aràndiga, Pep Mulet y Vicente Renau, *Lossless And Near-lossless Image Compression Based On Multiresolution Analysis*.

- Francesc Aràndiga, Pep Mulet y Vicente Renau, *Cell average data compression algorithms with exact error control*.

- Francesc Aràndiga, Pep Mulet y Vicente Renau, *Non-separable two-dimensional weighted ENO interpolation*.

En la parte que versa sobre la restauración de imágenes estudiamos problemas variacionales que son generalizaciones de la supresión de ruido en imágenes mediante minimización de la variación total [46]. Con estas generalizaciones se pretende suavizar el *efecto escalera* que produce la restauración por variación total

en transiciones suaves. Específicamente, proponemos un método rápido y robusto para la solución de las ecuaciones de Euler-Lagrange de los problemas generalizados. Este método generaliza el método propuesto en [15].

Los resultados aquí obtenidos han sido sometidos a publicación en el siguiente trabajo:

- Francesc Aràndiga, Pep Mulet y Vicente Renau, *A fast primal-dual method for generalized Total Variation denoising*

Esquema de la Tesis

Esta tesis está organizada de la siguiente forma:

Capítulo 1: introducimos los hechos y conceptos básicos de la compresión de imágenes.

Capítulo 2: proponemos transformaciones multiescala no separables con un mecanismo de control del error que resulta útil en muchas aplicaciones y totalmente necesario en algunas.

Capítulo 3: proponemos un mecanismo de interpolación Weighted Essentially Non-Oscillatory (WENO) bidimensional y no separable y presentamos diversas comprobaciones de sus prestaciones en esquemas de subdivisión (zoom).

Capítulo 4: exponemos el problema de la eliminación del ruido en imágenes mediante técnicas variacionales, mencionando la restauración por variación total y problemas relacionados, así como la necesidad de modificar el funcional regularizador para disminuir el efecto escalera.

Capítulo 5: planteamos las ecuaciones para resolver discretizaciones de generalizaciones del problema de la minimización de la variación total para la eliminación de ruido en imágenes, así como métodos básicos para su solución.

Capítulo 6: proponemos un método de Newton primal-dual para la solución rápida y robusta de las ecuaciones obtenidas en el capítulo anterior.

Capítulo 7: exponemos las conclusiones obtenidas a lo largo de la memoria y proponemos líneas de investigación futuras.

Parte I

Compresión de imágenes

1

Fundamentos de la compresión de imágenes

Comprimir es el proceso mediante el cual se consigue representar cierto volumen de información usando menor cantidad de datos. Este tipo de representaciones son útiles porque permiten ahorrar recursos de almacenamiento y de transmisión. Los compresores de datos son codificadores universales, capaces de comprimir cualquier secuencia de datos si en ella se detecta redundancia estadística. La compresión de imágenes es un proceso semejante, pero los datos comprimidos siempre son representaciones digitales de señales bidimensionales. Los compresores de imágenes explotan, además de la redundancia estadística, la redundancia espacial. De esta forma, los niveles de compresión que se pueden conseguir aumentan sensiblemente.

Las aplicaciones gráficas informáticas, en especial las que generan fotografías digitales pueden generar archivos muy grandes.

Debido a cuestiones de almacenamiento, y a la necesidad de transmitir rápidamente datos de imágenes a través de redes y por Internet, se han desarrollado una serie de técnicas de compresión de imágenes cuyo propósito es reducir el tamaño físico de los archivos.

La mayoría de las técnicas de compresión son independientes de formatos de archivos específicos; de hecho, muchos formatos soportan varios tipos diferentes de compresión. Forman una parte esencial de la creación, el uso y el almacenamiento de imágenes digitales. La mayoría de los algoritmos están especialmente adaptados a circunstancias específicas, que deben ser comprendidas para su uso efectivo. Por ejemplo, unos son más eficaces comprimiendo imágenes en monocromo (en blanco y negro), mientras que otros dan mejores resultados con imágenes complejas en color. Algunos algoritmos de compresión están patentados y sólo pueden utilizarse con licencia, otros han sido desarrollados como estándares abiertos. Esto puede ser importante en términos tanto de costes de creación como de sostenibilidad a largo plazo.

Las teorías de la información y de la codificación tienen dos objetivos principales: La transmisión de la máxima cantidad de información posible a través de un canal y la detección y corrección de errores de transmisión debido a ruidos en dicho canal. Claramente ambos objetivos persiguen a su vez el abaratamiento de los costes y la seguridad en la transmisión de la información.

El interés por la teoría de la información, desde el punto de vista de la compresión de datos, se centra principalmente en la codificación eficiente de la información en ausencia de ruido. Así, la meta es representar la información de la forma más compacta posible.

Shannon definió en [49] numerosos conceptos y medidas que se utilizan en todos los compresores de datos actuales, y de los que vamos a comentar en este capítulo.

1.1 Teoría de la información

La teoría de la información es una de las teorías más complejas de definir y delimitar, debido a que es muy complicado definir qué es la información, como se explica en [25]. Nosotros vamos a separar la información de la transmisión, con la idea de estructurar mejor este documento.

La información puede decirse que es todo aquello que tiene asociado algún significado. A parte de esta definición podemos añadir que es necesario representar la información de alguna forma para poder almacenarla o transmitirla. Decimos entonces que la información se almacena o transmite mediante mensajes.

En un proceso de transmisión de información existen tres elementos básicos:

- Emisor de información
- Canal de comunicación
- Receptor de información

1.1.1 Información

La información que fluye desde el emisor hasta el receptor, pasando por el canal, puede verse alterada en el caso de que el canal de comunicación no esté exento de ruido. En el contexto de la compresión de datos, como ya hemos indicado, el canal se supone sin ruido y por tanto, definiremos la información $I(a)$ asociada al suceso a como el logaritmo del inverso de la probabilidad de que el suceso a ocurra. Por tanto,

$$I(a) = \log \frac{1}{p(a)} = -\log p(a). \tag{1.1}$$

La definición de información dada es coherente si tenemos en cuenta que:

- Si $p(a) = 1$ (el suceso a siempre ocurre), entonces $I(a) = 0$ (la ocurrencia del suceso no proporciona al sujeto ninguna información).
- Si $p(a) \to 0$ (el suceso a casi nunca ocurre), entonces $I(a) \to +\infty$ (la ocurrencia del suceso proporciona mucha información).

Cuando la base del logaritmo es 2 en la expresión (1.1), la información se mide en "bits de información". Por ejemplo, si ha ocurrido un suceso a cuya probabilidad de ocurrir era 0,5, esto nos dará $I(a) = -\log_2(0,5) = 1$ bit de información.

1.1.2 Fuentes de información

Llamamos fuente de información discreta a aquella que produce mensajes como sucesiones de símbolos-fuente extraídos de entre una colección finita de símbolos-fuente que conforman un alfabeto-fuente. La notación

$$A = \{a_1, \ldots, a_r\}$$

indica un alfabeto-fuente A compuesto de r símbolos-fuente a_i. A r se le llama base de la fuente de información discreta. Nótese que hablar de bits de información no significa tomar $r = 2$. la fuente de información produce los símbolos-fuente de acuerdo con unas ciertas probabilidades

$$P = \{p(a_1), \ldots, p(a_r)\}.$$

La probabilidad asociada a un símbolo-fuente debe ser mayor que cero, para que el símbolo-fuente tenga razón de existir como tal en el alfabeto-fuente (o de lo contrario nunca formaría parte de ninguna secuencia-fuente) e inferior a uno, para que existan al menos dos símbolos-fuente diferentes en el alfabeto-fuente. Cuando, la probabilidad de un símbolo-fuente no depende del previamente emitido, hablamos de fuentes sin memoria, Cuando ocurre lo contrario, hablamos de fuentes con memoria.

1.1.3 Redundancia de información

La entropía $H(A)$ de una fuente de información discreta sin memoria A es la cantidad de información producida por la fuente en promedio. Puede calcularse a partir de las probabilidades de los símbolos-fuente mediante la expresión

$$H(A) = \sum_{i=1}^{r} p(a_i) \times I(a_i) = -\sum_{i=1}^{r} p(a_i) \times \log_2 p(a_i).$$

Debido a la base tomada en el logaritmo (dos), la entropía se calcula en bits de información/símbolos-fuente. La entropía $H(A)$ de una fuente A con r símbolos-fuente se maximiza cuando

$$p(a_i) = \frac{1}{r} \quad \forall\, a_i \in A.$$

En este caso

$$H(A) = \log_2 r.$$

Se define la entropía relativa o eficiencia de la fuente como la razón entre la entropía real y la máxima entropía de la fuente

$$E(A) = \frac{H(A)}{\log_2 r}.$$

Nótese que $0 < E(A) \leqslant 1$.

Se define además la redundancia de una fuente de información como

$$R(A) = 1 - E(A).$$

El concepto de redundancia de la fuente está ligado al exceso de terminología por parte de una fuente para producir mensajes que contienen una cierta cantidad de información. Una fuente más redundante que otra necesitaría producir mensajes más largos para transmitir la misma cantidad de información.

1.2 Teoría de la codificación

Ya comentamos que la información es imposible de manipular en la práctica si no está representada de alguna forma y en este punto es donde entra en juego el proceso de la codificación. El problema de la compresión de datos puede ser separado en dos fases fundamentales:

- La descomposición de un conjunto de datos (por ejemplo, un fichero de texto o una imagen) en una secuencia de eventos (porciones de información) cuya representación actual no es suficientemente compacta
- La codificación de estos eventos usando tan pocos elementos de representación como sea posible.

Un alfabeto-código es un conjunto B de elementos de representación utilizados en un proceso de codificación de la información. Los elementos de B denotados por b_i (a los que llamaremos símbolos-código) se usan para representar símbolos-fuente a_i. Al número s de símbolos-código distintos en B se llama base del código. Por tanto, un alfabeto-código se define como

$$B = \{b_1, \ldots, b_s\}.$$

Cuando $s = 2$, los elementos del alfabeto-código se llaman bits de código. Es importante diferenciar entre un bit de código y un bit de información. Debe tenerse en cuenta que un bit de código puede representar o no un bit de información. Esto en realidad depende del nivel de compresión conseguido. El nivel de representación más compacto es aquel en el que un bit de información está representado por un bit de código.

1.2.1 Codificación

La función codificación o código es una función que asigna a cada símbolo-fuente a_i un conjunto de uno o más símbolos-código

b_j, generando una cadena o bloque c_i. Sea C el conjunto de todos los c_i. Cuando r (la base del alfabeto-fuente) es igual a s (la base del alfabeto-código), cada símbolo-fuente se representa mediante un símbolo-código. Sin embargo, como suele ser habitual debido a requerimientos tecnológicos, $r > s$ y por tanto es necesario usar bloques de símbolos-código para designar a cada símbolo-fuente. A un bloque c_i de símbolos-código que realiza tal función lo llamaremos palabra-código. El número de símbolos-código que forman una palabra-código c_i es su longitud y la denotaremos por $l_s(c_i)$, donde s es la base del código. La secuencia de palabras-código que codifican una secuencia-fuente se llama secuencia-código (*code-stream*).

Veamos ahora una serie de definiciones que se utilizarán más adelante:

Código descodificable es aquel cuyos símbolos-fuente pueden ser descodificados. No dan pie a confusión.

Código instantáneamente descodificable Es aquel cuyos símbolos-fuente pueden ser descodificados sin necesidad de conocer la siguiente palabra-código de la secuencia-código. También se llaman *códigos prefijo* porque ninguna palabra-código es prefijo de cualquier otra.

Código de longitud fija es el que utiliza palabras-código de la misma longitud.

Código de longitud variable es aquel cuyas palabras-código utilizadas en una codificación tienen distinta longitud.

Debido a que la longitud de una palabra-código puede no ser fija, se define la longitud media de un código como

$$L_s(C) = \sum_{i=1}^{r} l_s(c_i) \times p(a_i),$$

donde s es la base del alfabeto-código utilizado y r la base del alfabeto-fuente. La longitud media de un código se expresa en símbolos-código/símbolos-fuente o símbolos-código / palabra-código.

Como es lógico cuando $s = 2$ (lo mínimo posible) estamos utilizando un alfabeto-código binario y entonces

$$L_2(C) = L(C) = \sum_{i=1}^{r} l_2(c_i) \times p(a_i),$$

donde hablaremos forzosamente de bits de código. Nótese que por definición de codificación se cumple que $p(a_i) = p(c_i)$.

Típicamente, en un proceso de compresión de datos, una secuencia-fuente está inicialmente codificada mediante una secuencia de palabras-código de longitud fija y se trata de encontrar una codificación de longitud variable que asigne a los símbolos-fuente más frecuentes nuevas palabras-código más cortas (y viceversa). Por tanto, la longitud media de un código de longitud fija coincide con la longitud de una palabra-código y es $\log_s r$ donde s es el número de símbolos-código y r es el número de símbolos-fuente a codificar.

1.2.2 Redundancia de un código

Utilizando la desigualdad de Kraft [35] es posible conocer cómo de bueno puede llegar a ser un código para la compresión de datos. Por tanto, podemos definir la eficiencia de un código como la razón existente entre la longitud óptima (determinada por la entropía) y la longitud actual conseguida por el código

$$E_s(C) = \frac{H_s(A)}{L_s(C)}.$$

La eficiencia de un código se expresa en bits de información/bits de código. Nótese que la eficiencia es un valor real comprendido entre cero y uno, y cuanto mayor sea, más eficiente sería el código. Así, podemos definir la redundancia de un código calculada como

$$R_s(C) = 1 - E_s(C).$$

La redundancia de un código se expresa en las mismas unidades que la eficiencia.

1.3 Compresión

Un sistema de compresión consiste en dos bloques estructurales distintos: un codificador y un decodificador. La imagen original entra en el codificador, el cual crea una serie de símbolos a partir de los datos de entrada. Después de la transmisión a través del canal, la representación codificada entra en el decodificador, donde se genera una reconstrucción de la imagen codificada. Si el sistema no está libre de error, se tendrá un cierto nivel de distorsión en la imagen decodificada. Tanto el codificador y el decodificador están formados por bloques independientes.

La compresión de datos se define como el proceso de reducir la cantidad de datos necesarios para representar eficazmente una información, es decir, la eliminación de datos redundantes.

1.3.1 Tipos de redundancia

En el caso de la compresión de imágenes, aparecen descritos en la bibliografía (por ejemplo [19], [25], [12]) tres tipos diferentes de redundancia, que pueden ser abordados de diferentes formas:

- **Redundancia de código**. Nos vamos a referir a una imagen como una función real f, definida en el conjunto $\Omega = (0,1)^2$, con $f(x,y)$ la intensidad (o nivel de gris) en el píxel (x,y). $f : \Omega \longrightarrow \mathbb{R}$. Una imagen en color sería una función vectorial $\vec{f} : \Omega \longrightarrow \mathbb{R}^3$ pues suponemos que tiene tres componentes de color (RGB, R = rojo, G = verde, B = azul).

  Normalmente, en una imagen suele haber un conjunto de niveles de gris que aparezcan más que el resto. La determinación de estos niveles de gris viene dada por el histograma de la imagen. Si el histograma de una imagen no es plano, la imagen presenta redundancia de código que puede expresarse utilizando la entropía. Una forma de eliminar la redundancia de codificación es asignar menos bits de código a niveles

de gris más frecuentes y más bits de código a los niveles de gris menos frecuentes. A esto se le llama código de longitud variable.

- **Redundancia entre píxeles**. La mayoría de las imágenes presentan semejanzas o correlaciones entre sus píxeles. Estas correlaciones se deben a la existencia de estructuras similares en las imágenes, puesto que no son completamente aleatorias. De esta manera, el valor de un píxel puede emplearse para predecir el de sus vecinos. Por ejemplo, una imagen con un nivel de gris constante es totalmente predecible una vez que se conozca el valor del primer píxel. Por el contrario, una imagen compuesta por un ruido blanco gaussiano, es totalmente impredecible.

- **Redundancia psico-visual**. El ojo humano responde con diferente sensibilidad a la información visual que recibe. La información a la que es menos sensible se puede descartar sin afectar la percepción de la imagen, así lo que se está eliminando es la redundancia visual.

  Dado que la eliminación de la redundancia psico-visual lleva una pérdida de información cuantitativa irreversible, se le suele denominar cuantificación. Técnicas de compresión como JPEG hacen uso de variaciones en la cuantización.

1.3.2 Medidas de compresión

Existe un gran número de métricas usadas en la medición de la eficiencia de un sistema de compresión. Esta medición siempre es relativa a los datos que están siendo comprimidos. Las medidas relativas siempre se calculan expresando una relación entre las longitudes de las secuencias-código antes y después del proceso de compresión (A y B respectivamente).

Citamos a continuación, algunas de las medidas más utilizadas descritas en [25]:

Bits por símbolo Una métrica bastante intuitiva es el número de bits de código/símbolo-fuente (BPS = Bits Por Símbolo) obtenidos tras comprimir. Este valor se calcula como

$$\text{BPS} = l \times \frac{B}{A},$$

donde l es el tamaño en bits (de código) de la representación no comprimida de los símbolos fuente.

Factor de compresión El factor de compresión es otra medida acerca del nivel de compresión conseguido que utilizaremos. Indica la proporción existente entre el tamaño del fichero sin comprimir A y el tamaño del fichero comprimido B y se expresa típicamente de la forma (X:1) donde

$$\text{X} = \frac{A}{B}.$$

Tasa de compresión Para calcular el nivel de compresión utilizaremos una medida llamada tasa de compresión (TC) definida por

$$\text{TC} = 1 - \frac{B}{A}.$$

1.3.3 Técnicas de compresión de imágenes

Las técnicas de compresión de imágenes se basan en algoritmos que pueden desarrollarse tanto en hardware como en software. El rendimiento de un algoritmo de compresión de imágenes se mide en función de sus características de compresión de datos, por la distorsión inducida en las imágenes y por la complejidad de la implantación de este algoritmo. Como se indica en [12]. Un sistema de compresión puede modelarse como una secuencia de tres operaciones:

- Transformación (mapeo): Efectúa una translación del conjunto de píxeles hacia otro dominio, donde el cuantificador y el codificador pueden utilizarse eficazmente. Es una operación reversible.

- Cuantificación: Efectúa la operación de clasificar los datos transformados en clases, para obtener un número más pequeño de valores. Es una operación irreversible.
- Codificación: La información ya comprimida se agrega a un tren binario que contiene ya sea la información que se debe transmitir o los códigos de protección y de identificación de las imágenes. Es una operación reversible.

Una manera eficiente de lograr una compresión de imagen consiste en la codificación de una transformada de la misma, en lugar de la propia imagen. El objetivo de la transformación es obtener un conjunto de coeficientes en el plano transformado que posean una correlación menor que la existente en el plano de la imagen (redundancia entre píxeles).

Clasificación de los métodos de compresión

Los algoritmos de compresión de gráficos se dividen en dos categorías:

- La compresión con pérdidas (técnicas “lossy”) consigue su propósito eliminando alguna información de la imagen, produciendo por tanto una pérdida de calidad en la misma.
- Las técnicas de compresión sin pérdida (técnicas “lossless”) reducen el tamaño conservando toda la información original de la imagen, y por tanto no hay reducción de calidad de la imagen.

Aunque las técnicas con pérdidas pueden ser muy útiles para crear versiones de imágenes para el uso cotidiano o en Internet, deben evitarse para el archivo de versiones maestras de imágenes.

1.3.4 Compresión sin pérdida

En diversas aplicaciones, como el almacenamiento de imágenes médicas o de documentos de negocios, las compresiones libres de

error son las únicas legalmente aceptadas. En el procesamiento de imágenes de satélite, el alto coste de la adquisición de datos hace que no sea deseable cualquier error o pérdida en la transmisión. En radiografía digital, una pérdida de información puede hacer que el diagnóstico no sea válido. Por tanto, la necesidad de compresión de imagen sin pérdida de información es deseable en función de la aplicación de la imagen procesada.

Hay una limitación intrínseca en cuánto se puede comprimir la imagen, si se sobrepasa ese límite se elimina información necesaria para recuperar la imagen original. La entropía es una medida de ese límite, si es elevada significa que hay mucha información aleatoria y por lo tanto los ratios de compresión sin pérdida son muy bajos, pero en la vida real las imágenes no son totalmente aleatorias entonces la entropía es baja.

Las técnicas de compresión sin error están compuestas, generalmente, de dos operaciones independientes: diseño de una representación alternativa de la imagen en la cual se disminuya la redundancia entre píxeles, y codificación de la representación para evitar redundancias de codificación. Estos pasos se corresponden con las operaciones de mapeado y codificación de símbolos llevadas a cabo en el bloque codificador del modelo de compresión de imágenes descrito en el apartado 1.3.3. Las técnicas de compresión sin pérdidas, como se explica en [19] se clasifican en:

- Técnicas de codificación de longitud variable
- Técnicas de codificación de plano de bit
- Técnicas de codificación con predicción sin pérdida

La combinación de estos grupos de técnicas da lugar a un conjunto de métodos eficientes de compresión de imagen sin pérdida de información, utilizados en diversos estándares internacionales.

Técnicas de codificación de longitud variable

Veamos las técnicas más importantes utilizadas para reducir la redundancia de codificación:

Codificación Huffman. La técnica más popular para eliminar la redundancia de codificación es la de Huffman. Desarrollada por David Huffman en 1952 [32], es uno de los algoritmos de compresión más antiguos y mejor establecidos. Es un algoritmo sin pérdidas y se utiliza para proporcionar una etapa final de compresión en varios sistemas de compresión muy utilizados, tales como JPEG y Deflate. Para una imagen concreta, esta codificación proporciona el menor número de símbolos de código por símbolo de fuente. La principal desventaja es que los símbolos de fuente han de ser codificados una vez para cada imagen. Este método trabaja asignando los códigos de longitud más corta a los símbolos fuente con mayor probabilidad de aparición en la imagen. Después de que el código ha sido creado, la codificación y la decodificación se realizan por el método simple de consulta de tabla.

La tasa de compresión del método Huffman varía con el tipo de imagen, pero pocas veces supera niveles de compresión de 8:1. Este método tiende a comportarse peor con ficheros que contienen cadenas largas de píxeles idénticos, que suelen comprimirse mejor con otros métodos. Huffman es un proceso lento y con mucha sensibilidad a la pérdida o adición de bits.

Codificación aritmética. El método de codificación aritmética no genera bloques de código, es decir, no existe una correspondencia biunívoca entre símbolos fuente y símbolos de código, como es el caso de la codificación Huffman y sus versiones modificadas. En lugar de ello, se asigna una sola palabra de código aritmético a una cadena completa de símbolos fuente (o mensaje). La propia palabra de código define un intervalo de números reales entre 0 y 1. A medida que crece el número de símbolos del mensaje, el intervalo utilizado para representarlo se hace menor y el número de unidades de información (bits) necesarios para representar el intervalo se hace mayor como se explica en [19]. Cada símbolo del mensaje reduce el tamaño del intervalo de acuerdo con su probabilidad de ocurrencia.

Los métodos más importantes dentro de la codificación arit-

mética son los algoritmos de diccionario entre los que se puede destacar los algoritmos:

- **LZ77** (Lempel y Ziv, 1977). El algoritmo LZ77 posee una tasa de compresión muy buena para muchos tipos de datos, pero la velocidad de codificación es lenta. Sin embargo, la decodificación es muy rápida.
- **LZSS** (Storer y Szymanski, 1982) es una mejora del algoritmo LZ77 que proporciona generalmente una tasa de compresión mayor que éste. Algunas de las implementaciones software más importantes en las que se incluye el algoritmo LZSS son los compresores de archivos clásicos PKZIP, ARJ, LHARC , ZOO o RAR.
- **LZ78** (Lempel y Ziv, 1978). Es más rápido que LZ77 pero con una tasa de compresión similar.
- **LZW** (Welch, 1984). Es una mejora del algoritmo LZW. La compresión LZW se encuentra en una serie de formatos de archivos gráficos comunes, incluyendo TIFF, GIF y *compress* de Unix.

Técnicas de codificación de plano de bit

Los métodos anteriores reducen la redundancia de codificación. Las técnicas de codificación de plano de bit afrontan además las redundancias entre píxeles de una imagen. Se basan en descomponer una imagen multinivel en una serie de imágenes binarias y comprimir cada una de ellas mediante alguno de los métodos de compresión binarios existentes.

Los niveles de gris de una imagen en escala de grises de m bits pueden representarse en la forma del polinomio de base 2

$$a_{m-1}2^{m-1} + a_{m-2}2^{m-2} + \ldots + a_1 2^1 + a_0 2^0.$$

Una de las técnicas más utilizadas y quizá la técnica de compresión más sencilla que se utiliza comúnmente es la RLE (Run Length Encoding). Los algoritmos RLE funcionan buscando secuencias de bits, bytes o píxeles del mismo valor, y codificando la

longitud y el valor de la secuencia. Como tal, RLE logra sus mejores resultados con imágenes que contienen grandes zonas de color contiguo, y especialmente imágenes en monocromo. Las imágenes complejas en color, tales como fotografías, no se comprimen bien pues en algunos casos, RLE puede incluso aumentar el tamaño del archivo.

Hay cierto número de variantes de RLE de uso común, que se encuentran en los formatos gráficos TIFF, PCX y BMP.

1.3.5 Compresión con pérdida

La codificación con pérdidas se basa en el compromiso existente entre la exactitud de la imagen reconstruida y la tasa de compresión. Si, en función de la aplicación, se puede tolerar la distorsión resultante (sea esta visible o no), puede lograrse un aumento significativo de la compresión. De hecho, algunas técnicas de compresión de imágenes monocromas con pérdidas logran reproducir versiones aceptables mediante una compresión de 30:1. Por su parte, las técnicas de compresión sin pérdidas rara vez logran una tasa de compresión mayor que 3:1. Como se comentó, la principal diferencia entre ambas aproximaciones consiste en la existencia o ausencia del bloque de cuantificación en el modelo del sistema de compresión.

Las técnicas de codificación de imágenes digitales con pérdida de información se clasifican en función de la aproximación que realizan al problema de la reducción de la redundancia psicovisual, en tres grupos:

- Técnicas de codificación con predicción
- Técnicas de codificación de transformada
- Técnicas de codificación fractal

Técnicas de codificación con predicción

Las técnicas de codificación con predicción operan directamente sobre los píxeles de una imagen y se llaman, por tanto, métodos

del dominio espacial. Entre las técnicas de codificación con predicción destacan la Modulación delta, los predictores óptimos y la cuantificación de Lloyd-Max [38]. Podemos encontrar más información de cada uno de las técnicas de codificación en [19].

Técnicas de codificación de transformada

En este apartado se consideran las técnicas de compresión basadas en la modificación de la transformada de una imagen. En las técnicas de codificación de transformadas, se usa una transformación reversible (Wavelets o Fourier, por ejemplo), para mapear la imagen en un conjunto de coeficientes transformados, los cuales son cuantificados y codificados. Para la mayoría de las imágenes reales, la mayor parte de los coeficientes obtenidos poseen magnitudes muy pequeñas, que pueden ser cuantificadas de manera gruesa o descartados directamente con una pequeña distorsión final de la imagen. El decodificador de un sistema de codificación de transformada realiza la secuencia de pasos inversa a la realizada por el codificador, a excepción de la función de cuantificación.

La elección de una transformada particular depende de la cantidad de error de reconstrucción permisible para una aplicación concreta y de los recursos computacionales disponibles. La compresión se realiza durante el paso de cuantificación, no durante la transformación. Las transformaciones más comúnmente utilizadas para la compresión de imágenes digitales son las de Fourier (DFT), Hartley (DHT), Karhunen-Loève (KLT), coseno discreto (DCT), Walsh-Hadamard (WHT), Haar, Gabor (DGT) y Wavelet (DWT), entre otras.

Técnicas de codificación fractal

La compresión fractal utiliza los principios matemáticos de la geometría fractal para identificar patrones redundantes y repetidos dentro de las imágenes. Estos patrones pueden ser identificados mediante el uso de transformaciones geométricas, tales como escalado y rotación, sobre elementos de la imagen. Una vez identificado, un patrón repetido sólo necesita almacenarse una vez, junto con la información sobre su ubicación en la imagen y las

transformaciones necesarias en cada caso. La compresión fractal hace un uso extremadamente intensivo del ordenador, aunque la descompresión es mucho más rápida. Es una técnica con pérdidas que puede lograr grandes tasas de compresión. A diferencia de otros métodos con pérdidas, una compresión más alta no produce la pixelación de la imagen, y aunque todavía se pierde información, esto tiende a ser menos evidente. La compresión fractal funciona mejor con imágenes complejas y altas profundidades de color.

Medidas de calidad

Las medidas señal-a-ruido (SNR, *Signal to Noise Ratio*) son estimaciones de la calidad de una imagen reconstruida comparada con una imagen original. La idea básica es reflejar en un número la calidad de la imagen reconstruida. Las imágenes reconstruidas con valores más altos se juzgan como mejores. Estas medidas son fáciles de computar y no se comparan con la opinión subjetiva humana.

Dada una imagen fuente $f(i,j)$ que contenga $N \times M$ píxeles y una imagen reconstruida $\hat{f}(i,j)$ donde $\hat{f}$ es la imagen reconstruida decodificando la versión codificada del f suponiendo que los valores de los píxeles $f(i,j)$ tienen un rango entre el negro (0) y el blanco (255). Calculamos SNR como:

$$SNR = \left(\frac{\sum_{i=1}^{N}\sum_{j=1}^{M}\left(f(i,j)-\bar{f}\right)^2}{\sum_{i=1}^{N}\sum_{j=1}^{M}\left(f(i,j)-\hat{f}(i,j)\right)^2} \right)^{\frac{1}{2}}, \quad \bar{f} = \frac{1}{NM}\sum_{i=1}^{N}\sum_{j=1}^{M} f(i,j).$$

Una medición real que computaremos es el valor relación señal a ruido pico de la imagen reconstruida que se llama PSNR (del inglés *Peak Signal to Noise Ratio*). Primero obtenemos el valor Error Cuadrático Medio (en inglés *mean squared error*, MSE) de la imagen reconstruida como sigue:

$$MSE = \frac{1}{NM}\sum_{i=1}^{N}\sum_{j=1}^{M}\left(f(i,j)-\hat{f}(i,j)\right)^2.$$

El sumatorio se realiza sobre todos los píxeles. Llamamos RMSE a la raíz cuadrada del MSE.

$$RMSE = \sqrt{\frac{1}{NM}\sum_{i=1}^{N}\sum_{j=1}^{M}\left(f(i,j) - \hat{f}(i,j)\right)^2}.$$

LA PSNR en decibelios (dB) se obtiene usando:

$$PSNR = 20\log_{10}\left(\frac{2^{bpp}-1}{RMSE}\right).$$

El rango de los valores PSNR típicos son entre 20 y 40. Se utilizan generalmente dos decimales (por ejemplo $23,58$). El valor real no es significativo, pero la comparación entre dos valores para diversas imágenes reconstruidas da una medida de calidad.

1.4 Estándares de compresión de imágenes

Existe una gran variedad de sistemas de compresión de imagen. El uso de uno u otro dependerá de la naturaleza de la imagen (imágenes monocromáticas, de escala de grises, imágenes en color, imágenes médicas, etc.) así como de la aplicación particular (edición, representación, almacenamiento, distribución, etc).

Los estándares de compresión aplican diferentes técnicas de manera conjunta para reducir los tres tipos de redundancia definidos en la sección 1.3.1. Esta sección describe los algoritmos de compresión de datos de imágenes más comúnmente utilizados.

1.4.1 JPEG

El algoritmo de compresión JPEG tiene sus orígenes en intentos de desarrollar técnicas de compresión para la transmisión de imágenes en color y en escala de grises. Fue desarrollado en 1990 por

el *Joint Photographic Experts Group* de la *International Standards Organization* (ISO) y CCITT. JPEG es una técnica con pérdidas que proporciona muy buenas tasas de compresión con imágenes complejas de 24 bits (color verdadero). Logra su efecto eliminando datos de imagen imperceptibles al ojo humano, utilizando la transformada discreta del coseno (DCT), seguida de la codificación Huffman para lograr una compresión aún mayor.

Figura 1.1: *Resultados de la compresión de la imagen Lena con el algoritmo JPEG. (a) Imagen original* 512×512 *con 8 bpp (b) factor de compresión aproximado 16:1 (c) factor de compresión aproximado 32:1 (d) factor de compresión aproximado 64:1.*

La especificación JPEG permite a los usuarios establecer el nivel de compresión, usando un Ajuste de calidad abstracto. Esto proporciona una compensación entre tasa de compresión y calidad de la imagen: cuanto mayor es el ajuste, mejor será la calidad de la imagen, pero a costa de un mayor tamaño de archivo.

La compresión se realiza en tres pasos secuenciales: cálculo de la DCT, cuantificación, y asignación de código de longitud variable de Huffman. La imagen se subdivide primero en bloques de 8 x 8 píxeles ($n, m = 8$ en las fórmulas), que se procesan de izquierda a derecha, y de arriba abajo. El proceso de transformación y normalización produce una gran cantidad de coeficientes nulos. Estos coeficientes se ordenan en forma de zigzag, para dar una secuencia 1D de coeficientes no nulos.

El estándar JPEG está diseñado para la compresión de imágenes digitales en color o escala de grises de imágenes reales, donde puede lograr tasas de compresión superiores a 25:1. No trabaja tan bien sobre imágenes sintéticas, como dibujos o texto.

En la figura 1.1 podemos observar las imágenes obtenidas al utilizar el algoritmo JPEG con la imagen "Lena" (512×512 en escala de grises) con diferentes ajustes de calidad. En el cuadro 1.1 podemos encontrar más resultados de la compresión de Lena con JPEG.

bpp	factor	MSE	PSNR	$\|\text{error}\|_1$	$\|\text{error}\|_2$	$\|\text{error}\|_\infty$
4	2:1	0,62	50,21	0,5288	0,7869	3
2	4:1	4,65	41,45	1,6432	2,1570	13
1	8:1	10,96	37,73	2,4415	3,3109	33
0,5	16:1	22,62	34,59	3,4021	4,7555	58
0,25	32:1	59,46	30,39	5,5020	7,7110	90
0,125	64:1	246,79	24,21	11,8970	15,6937	140

Tabla 1.1: *Resultados de la compresión de la imagen Lena con el algoritmo JPEG utilizando diferentes ajustes de calidad. El factor de compresión es aproximado porque se utiliza un ajuste de calidad abstracto.*

1.4.2 JPEG 2000

JPEG 2000 sustituye al algoritmo JPEG, desarrollado por el grupo ISO JPEG en 2000. Permite la compresión con y sin pérdidas, y utiliza la transformada wavelet discreta (DWT) para conseguir tasas de compresión más altas con una reducción menor de la calidad de la imagen. El estándar JPEG 2000 define un formato de archivo (JP2).

JPEG 2000 puede trabajar con niveles de compresión mayores que JPEG sin recurrir a los principales defectos del formato anterior con altas tasas de compresión: Generación de bloques uniformes y aspecto borroso. También se adapta mejor a la carga progresiva de las imágenes. Su principales desventajas son que tiende a emborronar más la imagen que JPEG incluso para un mismo tamaño de archivo, y que elimina algunos detalles pequeños y texturas que el formato JPEG normal sí llega a representar.

Una ventaja de JPEG2000 es la posibilidad de poder seleccionar un "área de interés". Esto quiere decir que el usuario encuadra la zona que desea visualizar con más detalle, con el consecuente ahorro en el ancho de banda de transmisión, dejando con menos detalles la zona que no interesa.

Otra de las ventajas del estándar JPEG2000 es que posee una modalidad de resolución progresiva. En dicha modalidad se aumenta la resolución o tamaño de la imagen a medida que se va decodificando la información. Existe también otro modo denominado “SNR progresivo” en donde lo que va aumentando es la calidad de la imagen a medida que va decodificando más información, como se explica en [21].

Inicialmente, las imágenes tienen que ser transformadas a partir del espacio de color RGB a otro espacio de color, dando lugar a tres componentes que se manejan por separado. Hay dos opciones posibles:

- Transformación reversible del espacio de color (RCT). Utiliza una versión modificada del espacio de color YUV que no introduce errores de cuantificación, por lo que es totalmente reversible.

- Transformación irreversible del espacio de color (ICT). Se llama "irreversible" porque tiene que ser aplicado sobre valores en coma flotante y provoca errores de redondeo.

Después de la transformación del color, la imagen se divide en las llamadas baldosas, regiones rectangulares de la imagen que se transforman y se codifican por separado. También es posible considerar la imagen completa como una sola baldosa. Una vez que se elige un tamaño, todos las baldosas en que se divide la imagen deben tener el mismo tamaño (salvo, opcionalmente, los de las fronteras derecha e inferior). Dividir la imagen en bloques tiene la ventaja de que el decodificador necesitará menos memoria para decodificar la imagen. La desventaja de este baldoseo es que la calidad de la imagen disminuye [51].

Luego se utiliza la transformada wavelet para descomponer cada baldosa en diferentes niveles. Pero son dos transformadas wavelet las que se pueden emplear:

- La transformada wavelet Cohen-Daubechies-Feauveau (CDF) 9/7 con la que se consigue una compresión con pérdidas.

- La transformada wavelet LeGall 5/3 que utiliza solamente coeficientes enteros con lo que se consigue una compresión sin pérdidas.

Después de la transformada wavelet, los coeficientes son escalares cuantificados para reducir la cantidad de bits para que los represente, a expensas de una pérdida de calidad. El resultado es un conjunto de números enteros que tienen que ser codificados bit a bit. El parámetro que se puede cambiar para establecer la calidad final es el paso de cuantificación: cuanto mayor es el paso, mayor es la compresión y la pérdida de calidad.

Para codificar los bits de todos los coeficientes cuantizados se utiliza el algoritmo EBCOT (*Embedded Block Coding with Optimal Truncation*) que fue introducido en 1998 por David Taubman en [53]. Se basa en dividir la imagen wavelet en bloques cuadrados (típicamente de 32×32 o 64×64 coeficientes) llamados bloques de código y comprimir progresivamente (*embedded*) cada uno de forma independiente. Una vez que los bloques de código se han comprimido, se seccionan y se reordenan los bit-streams (en capas)

para que se minimice la distorsión en el receptor. Así, el bit-stream se convierte en un pack-stream donde cada paquete contiene una cabecera que indica la contribución de ese bloque de código a esa capa de calidad seguido de los datos del correspondiente bloque de código truncados en un punto óptimo.

Usando EBCOT no se explota la redundancia que existe entre los distintos bloques de código. Además, cada *bit-stream* no es totalmente progresivo, aunque posee una gran cantidad de puntos de truncado óptimos, donde este efecto no se nota.

En la figura 1.2 podemos observar las imágenes obtenidas al utilizar el algoritmo JPEG2000 con la imagen "Lena" (512×512 en escala de grises) con diferentes ajustes de calidad. En el cuadro 1.2 podemos encontrar más resultados de la compresión de Lena con JPEG2000.

bpp	factor	MSE	PSNR	$\|\text{error}\|_1$	$\|\text{error}\|_2$	$\|\text{error}\|_\infty$
4	2:1	0,45	51,58	0,4026	0,6721	3
2	4:1	3,21	43,07	1,3922	1,7908	8
1	8:1	8,01	39,06	2,2101	2,8428	19
0,5	16:1	16,70	35,90	3,0796	4,0872	35
0,25	32:1	36,07	32,56	4,2970	6,0060	50
0,125	64:1	81,38	29,03	6,1648	9,0212	87
0,0625	128:1	198,55	25,15	9,5379	14,0909	119

Tabla 1.2: *Resultados de la compresión de la imagen Lena con el algoritmo JPEG2000 utilizando diferentes ajustes de calidad.*

1.4.3 JPEG-LS

JPEG-LS (JPEG LosslesS) [58] es un estándar de compresión de imágenes diseñado para conseguir una compresión efectiva de imágenes sin pérdidas o casi sin pérdidas.

El algoritmo utilizado en el estándar JPEG-LS es LOCO-I (*LOw COmplexity, context-based, lossless image compression algorithm*) propuesto por Hewlett-Packard en [57]. LOCO-I se construye sobre

Figura 1.2: *Resultados de la compresión de la imagen Lena con el algoritmo JPEG2000. (a) factor de compresión 16:1 (b) factor de compresión 32:1 (c) factor de compresión 64:1 (d) factor de compresión 128:1.*

un concepto conocido como modelado de contexto. La idea en la que se basa es utilizar la estructura de la imagen, modelada en términos de probabilidades condicionales de cuál es el valor del píxel que sigue a cualquier otro píxel de la imagen.

Los autores del algoritmo LOCO-I se percataron de que había mucha redundancia que podía ser explotada teniendo en cuenta el contexto espacial, a un coste muy bajo en comparación con el uso de codificadores más complejos como es la codificación aritmética. Por este motivo, decidieron usar la codificación de Rice y diseñar así un compresor y descompresor muy rápidos. Todo esto ha provocado que LOCO-I haya sido adoptado casi sin modificación alguna para el estándar JPEG-LS.

1.4.4 PNG

El formato PNG (*Portable Network Graphics*). Soporta imágenes de color indexado, en escala de grises y de color de alta calidad (16 bits por píxel), además de un canal opcional alfa (corrección de color). Este formato se ha diseñado para aplicaciones como Internet, de manera que la tasa de compresión utilizada es bastante alta (superior a GIF). Además, proporciona (como GIF y JPEG) una visualización progresiva. La característica más destacable de este formato es su robustez ante errores de transmisión. El algoritmo de compresión sin pérdidas es el algoritmo de codificación aritmética LZ77. Aunque el GIF soporta animación, el PNG se desarrolló como un formato de imagen estático y se creó el formato MNG como su variante animada.

1.4.5 BZIP2

Bzip2 es un algoritmo de compresión de datos sin pérdida desarrollado por Julian Seward en 1996. Comprime más que los tradicionales compresores gzip o ZIP aunque es más lento. En la mayoría de casos, bzip es superado por los algoritmos PPM en términos

de compresión pero es el doble de rápido al comprimir y seis veces más rápido al descomprimir. Bzip2 utiliza la transformada de Burrows-Wheeler y la codificación Huffman. El antecesor de bzip2 (bzip) utilizaba codificación aritmética.

1.4.6 RAR

RAR es un formato de archivo, con un algoritmo de compresión sin pérdida utilizado para la compresión de datos y archivado, desarrollado por el ingeniero de software ruso, Eugene Roshal. RAR utiliza un algoritmo de compresión basado en el LZSS que, a su vez, se basaba en el LZ77. Este formato utiliza lo que se conoce como compresión sólida que permite comprimir varios ficheros juntos, de forma que una misma ventana de búsqueda se aplica a todo el conjunto de archivos, con lo que el nivel de compresión es mayor.

1.4.7 PPMZ

PPMZ es un algoritmo de codificación de datos sin pérdida desarrollado por Charles Bloom en 1997 [11] basado en PPM (*Prediction by Partial Matching*). Los modelos PPM usan un conjunto de símbolos previos en el flujo de símbolos no comprimidos para predecir el siguiente símbolo en dicho flujo.

El algoritmo PPMZ es muy eficiente para comprimir imágenes del tipo mapa de bits pero no es el mejor para codificar imágenes transformadas pues no tiene en cuenta la estructura de árbol que sí que se aprovecha con los algoritmos EZW o EBCOT, por ejemplo.

1.4.8 EZW

El método de compresión EZW (*Embedded Zerotree Wavelet*) fue propuesto por Shapiro en 1993 en [50]. Este método explota las propiedades aportadas por la DWT para obtener resultados satisfactorios en la compresión: un gran porcentaje de coeficientes wavelets próximos a cero y la agrupación de la energía de la imagen. El EZW es sensible al grupo de bits transmitidos por orden de significancia, lo que le permite una compresión progresiva (embedded coding) de la imagen. Cuantos más bits se añadan al resultado de la compresión, más detalles se estarán transmitiendo.

EZW usa un código aritmético binario y un alfabeto de 4 símbolos para indicar de la forma más compacta posible al descodificador la situación de los zerotrees. EZW es considerado un punto de inflexión en la curva creciente que relaciona la tasa de compresión con el coste computacional de los algoritmos. Desde entonces se ha producido un gran esfuerzo por diseñar algoritmos más eficaces y estandarizaciones como la de JPEG 2000 o CREW (*Compression with Reversible Embedded Wavelets*) de la compañía RICOH. Todos ellos poseen el denominador común de usar un código de longitud variable (casi siempre codificación aritmética) para comprimir entrópicamente aprovechando la redundancia remanente en la DWT-2D.

1.4.9 SPIHT

El algoritmo SPIHT (Set Partitioning in Hierarchical Trees) fue ideado por Said y Pearlman en 1996 [48] y tiene sus raíces en el algoritmo EZW. Sin embargo SPIHT es conceptualmente diferente porque usa un código de longitud fija. La longitud de cada palabra de código es de un bit y esta codificación es eficiente porque la probabilidad de cada símbolo (que coincide con la de cada bit o palabra de código) es 0.5. Su existencia demuestra que es posible diseñar codecs eficientes sin la necesidad de diferenciar entre

un modelo probabilístico y un codificador de longitud variable (Al igual que ocurre con la familia de compresores LZ).

SPIHT se basa en la clasificación de árboles jerárquicos de coeficientes de la transformada wavelet. Al igual que el EZW permite la transmisión progresiva de la información por orden de bits más significativos y también logra imágenes con una gran calidad y altas tasas de compresión.

En el apéndice A se analizarán con más profundidad los algoritmos EZW y SPIHT.

1.4.10 Otros

JBIG. El algoritmo JBIG (*Joint Bi-level Image Experts Group*) trata de mejorar las deficiencias del estándar JPEG en el manejo de imágenes binarias. La aportación más importante de este estándar sobre el JPEG es la definición de un algoritmo para la compresión sin pérdidas de imágenes binarias.

Deflate. Deflate es un algoritmo sin pérdidas basado en la compresión LZ77 y la codificación Huffman. Fue desarrollado por Phil Katz en 1996 para su uso en el formato de archivo comprimido PKZIP, y forma la base de la compresión PNG [13].

GIF. El formato GIF (*Graphic Interchange Format*) es un formato de archivos de gráficos de mapa de bits desarrollado por *Compuserve*. Existen dos versiones de este formato de archivos desarrolladas en 1987 y 1989 respectivamente:

- El GIF87a, que es compatible con la compresión LZW, puede entrelazar, (permitir la visualización progresiva) una paleta de 256 colores y tiene la posibilidad de crear imágenes animadas (llamadas GIF animados) almacenando varias imágenes en el mismo archivo.
- El GIF89a, que tiene como agregado la posibilidad de designar un color transparente para la paleta y especificar el tiempo de las animaciones.

GIF es un formato sin pérdida de calidad para imágenes que consigue tasas de compresión de aproximadamente 2:1 y es ampliamente utilizado en Internet.

PGM. El formato PNM incluye los formatos de imagen PPM, PGM y PBM que se utilizan para almacenar imágenes de color RGB, imágenes en escala de gris e imágenes en blanco y negro respectivamente. Para cada uno de los tres formatos hay una versión binaria y otra ASCII. En el primer caso la imagen se codifica como una secuencia de bits y en el segundo se codifica en un archivo de texto donde se listan los valores numéricos de cada píxel. Las imágenes representadas mediante este tipo de formatos tienen la ventaja de que son muy sencillas de manejar y manipular, pero también tienen el inconveniente de que ocupan mucho espacio de memoria. El formato PGM (*Portable GrayMap*), que soporta imágenes en escala de grises, cada píxel se representa con un byte o con su valor en ASCII. Es un formato ampliamente utilizado por investigadores en el tema del procesamiento de imágenes, por su simplicidad y eficiencia.

TIFF. TIFF (*Tagged Image File Format*, formato de archivo de imágenes con etiquetas) debe su nombre a que contiene, además de los datos de la imagen propiamente dicha, "etiquetas" en las que se guarda información sobre las características de la imagen, que sirve para su tratamiento posterior. Como se explica en [19], estas etiquetas describen el formato de las imágenes almacenadas o el tipo de compresión aplicado a cada imagen. A pesar de incluir una técnica de recuperación perfecta, la tasa de compresión alcanzada con esta técnica es mucho menor que los estándares PNG o JPEG.

CALIC. CALIC (*Context-Based, Adaptive, Lossless Image Coding*) [59] es muy similar a LOCO-I en todos los aspectos excepto en que la complejidad del codec entrópico utilizado es superior. Se trata de otro compresor de imágenes secuencial que realiza una predicción espacial basada en el contexto. Explota tanto la redundancia local como la global. El predictor es adaptativo pues aprende de los errores cometidos en predic-

ciones anteriores. Además, CALIC igual que FELICS y LOCO-I, está también preparado para codificar imágenes de hasta 16 bpp. CALIC alcanza muy buenas tasas de compresión.

FELICS. El compresor de imágenes FELICS [30] es uno de los compresores de imágenes sin pérdida de información más sencillos, rápidos y ciertamente eficientes que se han diseñado (de hecho, su nombre es el acrónimo de *Fast, Efficient, Lossless Image Compression System*). La imagen se comprime por filas y, conceptualmente, FELICS es idéntico a JPEG en el sentido de que existe un predictor muy simple y un codificador entrópico muy rápido. El uso de un código de Rice o de un código binario ajustado hace posible que FELICS pueda codificar imágenes de cualquier profundidad de color sin que los algoritmos de codificación y descodificación sean modificados. En el caso de usar la codificación de Huffman o incluso la aritmética, esto hubiera sido mucho más complejo.

FIF. El formato FIF (*Fractal Image Format*) lleva a cabo una codificación que realiza una aproximación a cada imagen particular mediante la búsqueda de coeficientes que permitan la definición de un sistema fractal. La ventaja principal de este tipo de formato, además de su compresión (150:1), que es bastante mayor que la obtenida por GIF o JPEG, es que permite una visualización a varias escalas, aprovechando la naturaleza fractal de la transformación, sin una pérdida aparente de la calidad incluso a escalas muy altas de ampliación.

Este nuevo método de compresión de imágenes fue propuesto por Michael Barnsley utilizando el Sistema de Funciones Iteradas (*Iterated Functions System*, IFS) propuestos por John Hutchinson en 1980. Arnaud Jacquin en 1989, desarrolló la codificación fractal de imágenes para imágenes en escala de grises, empleando el método IFS pero particionando la imagen (*Partitioned IFS*, PIFS)[44]. Esta forma de comprensión es un método de codificación con pérdidas y su principal desventaja es el tiempo de ejecución para hallar el PIFS adecuado.

La compresión fractal es una técnica en desarrollo. Los nue-

vos avances se centran en las diferentes formas de implementar un algoritmo de compresión, que permita obtener altas tasas de compresión con una calidad de imagen determinada y un tiempo de procesado aceptable.

Esta técnica de compresión no es muy apropiada para aplicarla sobre imágenes médicas. Se logran altas tasas de compresión pero la calidad de las imágenes no logra alcanzar los niveles aceptables de fiabilidad. Un problema típico que presentan estas imágenes es el de *blocking*. Se pueden encontrar más detalles en [20], [52].

Nombre de la imagen	compresores			formatos de imagen		
	BZIP2	PPMZ	RAR	PNG	JP2	JLS
Lena	4,828	4,864	4,685	4,649	4,401	4,274
peppers	4,740	4,688	4,545	4,035	3,641	3,483
barbara	6,171	6,411	6,346	5,667	4,836	4,862
cameraman	5,581	5,573	5,849	5,261	5,133	4,740
fingerprint	5,122	5,316	5,794	5,632	5,429	5,389
sintética	0,079	0,099	0,082	0,145	0,679	0,125

Tabla 1.3: *Resultados, dados en bits por píxel (bpp), obtenidos al comprimir las diferentes imágenes test con algunos de los métodos explicados anteriormente. Los formatos de imagen utilizados son del tipo lossless (JP2=JPEG-2000, JLS=JPEG-LS).*

2

Transformaciones multiescala con Error Control

2.1 Introducción

Controlar la calidad de una imagen codificada en lugar de la tasa de compresión es conveniente para las aplicaciones en las que el control de la calidad es de la mayor importancia, sin embargo nos gustaría ser lo más económicos posible respecto al almacenamiento y a la velocidad de computación. Veremos un algoritmo de compresión basado en el marco interpolatorio de Harten para la multirresolución que garantiza una estimación específica del error

entre la imagen original y la descodificada medido en las normas máximo y L_2.

Tradicionalmente las técnicas de compresión de imágenes se han clasificado en dos categorías: "lossless schemes" (métodos sin pérdida) y "lossy schemes" (métodos con pérdida). Los métodos sin pérdida se utilizan en aplicaciones donde los pequeños detalles de una imagen pueden ser muy importantes (imágenes médicas, espaciales, etc.). Los métodos con pérdida son necesarios en situaciones donde se necesita una buena tasa de compresión (por ejemplo en fotografía digital, en la que generalmente, perder algunos detalles de la imagen no es problemático). Recientemente se está estudiando una tercera clase de técnica: "near-lossless image coding" (métodos casi sin pérdida): producen buenas tasas de compresión y garantizan al mismo tiempo el tipo y la cantidad de distorsión producida. Vamos a centrarnos en el estudio de las técnicas lossless y near-lossless basadas en unas ciertas transformaciones de multirresolución.

2.2 Transformaciones multiescala

Las transformaciones multiescala se usan actualmente como primer paso de los algoritmos para la compresión de imágenes. Un esquema de compresión basado en la multirresolución puede ser descrito esquemáticamente de la forma siguiente: Sea $\bar{f}^L$ una secuencia de datos (que representan una imagen) en una determinada resolución L, de ella obtenemos su representación en forma de multirresolución $M\bar{f}^L = (\bar{f}^0, d^1, \ldots, d^L)$ que está compuesta por:

- $\bar{f}^0$ es un "sampleado" de los datos con una resolución mucho más grosera.
- d^k es la secuencia de detalles. Representa la información intermedia necesaria para reconstruir $\bar{f}^k$ a partir de $\bar{f}^{k-1}$. Siendo k el nivel de resolución de la imagen.

Se supone que $M\bar{f}^L$ es una representación de la imagen mucho más "eficiente".

Pero ¿cómo se realiza una transformación multiescala?. Partimos de una imagen f que consideramos como una función que pertenece a un cierto espacio de funciones $\mathcal{F}$:

$$f : [0,1] \times [0,1] \longrightarrow \mathbb{R}$$

de la que tenemos cierta información $\bar{f}^k = \left\{f^k_{i,j}\right\}^{N_k}_{i,j=0}$ en el espacio V^k, que representa, de alguna manera, los diferentes niveles de resolución (a mayor k mayor resolución).

Definimos el operador *decimación*

$$D^{k-1}_k : V^k \longrightarrow V^{k-1}$$

que es un operador lineal y sobreyectivo y nos da información discreta de la señal con resolución $k-1$ a partir de la de nivel k ($D^{k-1}_k \bar{f}^k = \bar{f}^{k-1}$) . Definimos también el operador *predicción*

$$P^k_{k-1} : V^{k-1} \longrightarrow V^k$$

que nos da una aproximación a la información discreta en el nivel k a partir de la del nivel $k-1$ ($P^k_{k-1}\bar{f}^{k-1} \approx \bar{f}^k$). La propiedad básica de consistencia que estos dos operadores deben satisfacer es $D^{k-1}_k P^k_{k-1} = I_{V^{k-1}}$ donde $I_{V^{k-1}}$ es el operador identidad de V^{k-1}.

Definimos el error de predicción e^k

$$e^k := \bar{f}^k - P^k_{k-1}\bar{f}^{k-1} = \bar{f}^k - P^k_{k-1}D^{k-1}_k\bar{f}^k = \left(I_{V^k} - P^k_{k-1}D^{k-1}_k\right)\bar{f}^k \in V^k.$$

Obviamente los vectores $\{\bar{f}^k\}$ y $\{\bar{f}^{k-1}, e^k\}$ tienen la misma información aunque el número de elementos de $\{\bar{f}^k\}$ no es igual al número de elementos de $\{\bar{f}^{k-1}, e^k\}$. Pero esto podemos solucionarlo puesto que e^k contiene información redundante pues observamos que

$$\begin{aligned} D^{k-1}_k e^k = D^{k-1}_k\left(\bar{f}^k - P^k_{k-1}\bar{f}^{k-1}\right) = D^{k-1}_k\bar{f}^k - D^{k-1}_k P^k_{k-1}\bar{f}^{k-1} = \\ = \bar{f}^{k-1} - I_{V^k}\bar{f}^k = \bar{f}^{k-1} - \bar{f}^{k-1} = 0 \end{aligned}$$

por tanto $e^k \in \mathcal{N}(D^{k-1}_k) = \{\bar{f}^k \in V^k; D^{k-1}_k\bar{f}^k = 0\}$ y como $D^{k-1}_k : V^k \to V^{k-1}$ entonces

$$\text{dim}\mathcal{N}(D^{k-1}_k) = \text{dim}V^k - \text{dim}V^{k-1} =: m$$

por lo que podemos considerar que $\mathcal{N}(D_k^{k-1})$ está generado por la base $\{\mu_j^k\}_{j=1}^m$. Así, el error de la predicción que se describe en términos de dimV^k componentes puede representarse por sus m coordenadas d^k

$$e^k = \sum_{j=1}^{m} d_j^k \mu_j^k.$$

Desde un punto de vista algebraico, necesitamos un operador que descarte la información redundante de e^k asignándole sus coordenadas d^k en la base $\{\mu_j^k\}$. Vamos a llamar G_k a este operador

$$\begin{aligned} G_k : \mathcal{N}(D_k^{k-1}) &\to \mathcal{G}^k \\ e^k &\mapsto d^k \end{aligned}$$

Necesitamos también otro operador que reconstruya la información redundante original de e^k a partir de d^k. Nos referiremos a él como E_k

$$\begin{aligned} E_k : \mathcal{G}^k &\to \mathcal{N}(D_k^{k-1}) \\ d^k &\mapsto e^k. \end{aligned}$$

Como $e^k = E_k d^k$ y $d^k = G_k e^k$ tenemos que

$$E_k G_k e_k = I_{\mathcal{N}(D_k^{k-1})} e_k \forall e^k \in \mathcal{N}(D_k^{k-1}) \Rightarrow E_k G_k = I_{\mathcal{N}(D_k^{k-1})}.$$

En este punto podemos mostrar que hay una correspondencia biunívoca entre $\{\bar{f}^k\}$ y $\{\bar{f}^{k-1}, d^k\}$: dada $\bar{f}^k$ evaluamos

$$\begin{cases} \bar{f}^{k-1} = D_k^{k-1} \bar{f}^k \\ d^k = G_k \left(I_{V^k} - P_{k-1}^k D_k^{k-1} \right) \bar{f}^k \end{cases}$$

y dados $\bar{f}^{k-1}$ y d^k reconstruimos $\bar{f}^k$ con

$$\begin{aligned} P_{k-1}^k \bar{f}^{k-1} + E_k d^k &= P_{k-1}^k D_k^{k-1} \bar{f}^k + E_k G_k \left(I_{V^k} - P_{k-1}^k D_k^{k-1} \right) \bar{f}^k = \\ &= P_{k-1}^k D_k^{k-1} \bar{f}^k + \left(I_{V^k} - P_{k-1}^k D_k^{k-1} \right) \bar{f}^k = \\ &= \bar{f}^k. \end{aligned}$$

La descomposición multiescala se realiza repitiendo los dos procesos en cada nivel de resolución y así conseguimos

$$\left\{ \bar{f}^L \right\} \overset{1:1}{\longleftrightarrow} M\bar{f}^L = \left\{ \bar{f}^0, d^1, \ldots, d^L \right\}$$

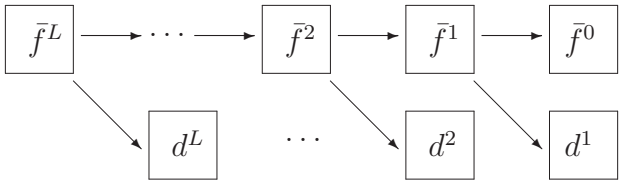


Figura 2.1: *Transformada 2D directa.*

donde la transformada de multirresolución directa viene dada por el algoritmo

$$\begin{cases} DO\ k = 1, \ldots, L \\ \bar{f}^{k-1} = D_k^{k-1} \bar{f}^k \\ d^k = G_k \left(I_{V^k} - P_{k-1}^k D_k^{k-1} \right) \bar{f}^k \end{cases} \tag{2.1}$$

y la transformada de multirresolución inversa viene dada por el algoritmo

$$\begin{cases} DO\ k = L, \ldots, 1 \\ \bar{f}^k = P_{k-1}^k \bar{f}^{k-1} + E_k d^k. \end{cases} \tag{2.2}$$

Podemos remarcar que en el ámbito del procesamiento de señales D_k^{k-1} jugaría el papel de "filtro de paso bajo" mientras que G_k sería el "filtro de paso alto" como se explica en [28].

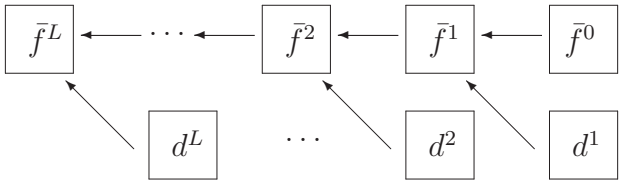


Figura 2.2: *Transformada 2D inversa.*

Por supuesto, el propósito de una descomposición multiescala no es sólo descomponer y reconstruir sino realizar algún proceso por el medio. $M\bar{f}^L$ es luego procesada usando un cuantizador (no reversible) $Q_{\bar{\varepsilon}} M \bar{f}^L = (\hat{f}^0, \hat{d}^1, \ldots, \hat{d}^L)$ y luego pasada al "encoder".

$$\bar{f}^L \xrightarrow[\text{multirresolución}]{M} M\bar{f}^L \underbrace{\xrightarrow[\text{cuantización}]{Q_{\bar{\varepsilon}}} Q_{\bar{\varepsilon}} M \bar{f}^L \xrightarrow[\text{encoder}]{} \boxed{1101000\cdots}}_{\text{COMPRESIÓN}}$$

Normalmente se diseña el cuantizador para que $M\bar{f}^L$ y $Q_{\bar{\varepsilon}}M\bar{f}^L$ sean "parecidas" en el sentido que para alguna norma discreta se tenga

$$\left\|\bar{f}^0 - \hat{f}^0\right\| \leqslant \varepsilon_0 \quad \text{y} \quad \left\|\hat{d}^k - d^k\right\| \leqslant \varepsilon_k,\ \ 1 \leqslant k \leqslant L,\ \ \bar{\varepsilon} = \{\varepsilon_0, \ldots, \varepsilon_L\}\,.$$

Descodificar la información comprimida requiere aplicar transformación de multirresolución inversa M^{-1} a la información modificada $M\bar{f}^L$. Este paso produce $\hat{f}^L = M^{-1}Q_{\bar{\varepsilon}}M\bar{f}^L$ un conjunto de datos que se esperan "parecidos" al conjunto original $\bar{f}^L$. Para que eso sea cierto, se requiere que exista un $C > 0$ tal que

$$\left\|\bar{f}^L - \hat{f}^L\right\| \leqslant C\varepsilon \tag{2.3}$$

en una norma apropiada. Es decir, es importante controlar el error obtenido. Hay algunos algoritmos lossy que controlan la norma L_2 del error pero en general no es fácil encontrar algoritmos que permitan controlar la norma L_∞. Más adelante veremos una transformación multiescala que permite una evaluación exacta del error de compresión.

Sería obvio decir que un marco general como (2.1) y (2.2) tiene muchas ventajas, probablemente la mayor de ellas es la libertad que le da al usuario para ajustar el esquema de multirresolución a sus necesidades. Se requiere que el operador decimación sea siempre lineal, no así el operador predicción. Un operador predicción lineal produce transformaciones lineales mientras que una predicción no lineal produce transformaciones de multirresolución no lineales.

El análisis de estabilidad para procesos de predicción lineales se pueden llevar a cabo usando herramientas provenientes de la teoría de wavelets, esquemas de subdivisión y análisis funcional (ver [27], [9]). Con estas técnicas se prueba la existencia de C que satisface (2.3) pero en la mayoría de los casos no es fácil encontrar esa C.

En el caso no lineal la estabilidad se asegura modificando el algoritmo de transformación directa. La idea del algoritmo modificado se debe a Ami Harten (ver [26],[8]). El propósito de este algoritmo (al que llamaremos *errorcontrol*) es seguir el paso de la acumulación del error al procesar los valores de la representación multiescala.

2.3 Obtención de los datos discretos

Los procesos de discretización son herramientas para ir de una función en algún espacio de funciones a un conjunto de datos discretos los cuales contienen información sobre la función original.

En Análisis numérico, la generación de datos discretos se hace usualmente a partir de la aplicación de un *procedimiento de discretización* particular. Vamos a ver como a partir de operadores discretización podemos definir un conjunto multirresolutivo que será el más apropiado para un conjunto de datos generados por ese proceso de discretización.

Definimos el operador discretización

$$\mathcal{D}_k : \mathcal{F} \longrightarrow V^k = \mathcal{D}_k(\mathcal{F})$$

que es lineal y sobreyectivo. Si tenemos un conjunto de operadores discretización $\{\mathcal{D}_k\}$ diremos que este conjunto es anidado si

$$\mathcal{D}_k f = 0 \Rightarrow \mathcal{D}_{k-1} f = 0.$$

A partir de una sucesión de discretizaciones anidadas podemos definir un operador decimación D_k^{k-1} de la siguiente manera: Sea $\bar{f}^k \in V^k$ y sea $f \in \mathcal{F}; \mathcal{D}_k f = \bar{f}^k$ (que existe porque $\mathcal{D}_k$ es sobreyectivo). Entonces definimos

$$D_k^{k-1} \bar{f}^k := \mathcal{D}_{k-1} f$$

y como $\{\mathcal{D}_k\}$ es anidadas, D_k^{k-1} está bien definida, es decir, su definición es independiente de f. Para verlo tomemos $f_1, f_2 \in \mathcal{F}$ tales que $\mathcal{D}_k f_1 = \bar{f}^k = \mathcal{D}_k f_2$, entonces por

$$0 = \mathcal{D}_k f_1 - \mathcal{D}_k f_2 = \mathcal{D}_k(f_1 - f_2) \Rightarrow 0 = \mathcal{D}_{k-1}(f_1 - f_2) = \mathcal{D}_{k-1} f_1 - \mathcal{D}_{k-1} f_2$$

se prueba que D_k^{k-1} está bien definida. Y como $D_k^{k-1}\mathcal{D}_k = \mathcal{D}_{k-1}$ tenemos que D_k^{k-1} es una aplicación lineal y sobreyectiva. Habría que recalcar también que la decimación se ha construido sin el conocimiento explícito de f.

Como $V^k = \mathcal{D}_k(\mathcal{F})$, se sigue que $\mathcal{D}_k$ tiene un operador inverso (al menos uno) al que denotamos por $\mathcal{R}_k$ y lo llamaremos reconstrucción de $\mathcal{D}_k$

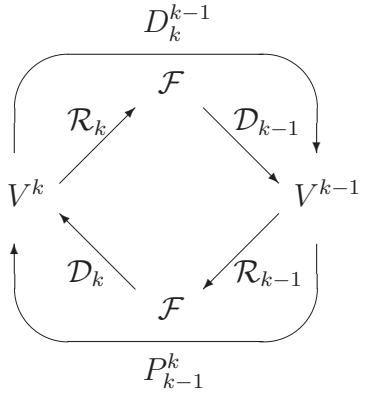
$$\mathcal{R}_k : V^k \longrightarrow \mathcal{F}.$$

Sea $\{\mathcal{D}_k\}$ anidada y $\{\mathcal{R}_k\}$ tal que $\mathcal{D}_k\mathcal{R}_k = I_{V^k}$, podemos definir el operador predicción

$$P_{k-1}^k := \mathcal{D}_k\mathcal{R}_{k-1}.$$

Se puede ver claramente que P_{k-1}^k es el operador inverso de la predicción D_k^{k-1}

$$D_k^{k-1}P_{k-1}^k = (D_k^{k-1}\mathcal{D}_k)\mathcal{R}_{k-1} = \mathcal{D}_{k-1}\mathcal{R}_{k-1} = I_{V^{k-1}}$$



2.4 Multirresolución basada en valores puntuales

Los algoritmos multiescala de Ami Harten han sido ampliamente analizados en numerosos artículos y están basados en dos operadores (*decimación* y *predicción*) que conectan dos niveles de resolución adyacentes. Aquí nos interesa el marco interpolatorio, que se puede describir de la siguiente manera: Consideramos los píxeles de una imagen dada $\bar{f}^L = \left\{\bar{f}_{i,j}^L\right\}_{i,j=0}^{N_L}$ como los valores puntuales de una función f en los puntos de la malla $X^L = \left\{\left(x_i^L, y_j^L\right)\right\}, x_i^L =$

$ih_L, y_j^L = jh_L,\ i,j = 0,\ldots,N_L,\ N_L h_L = 1$ con $N_L = 2^L N_0$ y $N_0 \in \mathbb{N}$. Se definen las correspondientes particiones para los niveles de resolución más bajos como: $X^k = \left\{\left(x_i^k, y_j^k\right)\right\}, x_i^k = ih_k, y_i^k = jh_k,\ i,j = 0,\ldots,N_k,\ N_k h_k = 1$ con $N_k = 2^k N_0$ y $N_0 \in \mathbb{N}$. Notar que $\left(x_i^{k-1}, y_j^{k-1}\right) = \left(x_{2i}^k, y_{2j}^k\right)$. Es decir, utilizamos un conjunto anidado de mallas en el intervalo $[0,1] \times [0,1]$. Si consideramos que $\bar{f}^k = \left\{\bar{f}_{i,j}^k\right\}_{i,j=0}^{N_k}$ representa los valores puntuales de la misma función en los puntos $X^k = \left\{\left(x_i^k, y_j^k\right)\right\}$, podemos reducir la resolución de la información con:

$$\bar{f}_{i,j}^{k-1} = \bar{f}_{2i,2j}^k, \quad 0 \leqslant i,j \leqslant N_k.$$

$\bar{f}^p$ $\bar{f}^{p-1}$ $\bar{f}^{p-2}$ $\bar{f}^{p-3}$

Figura 2.3: *Ejemplo de la reducción de la resolución de una imagen (Lena con $N_k = 128$) a partir de un cierto nivel de resolución p.*

A partir de la resolución al nivel $k-1$, $\bar{f}^{k-1} = \{\bar{f}_{i,j}^{k-1}\}_{i,j=0}^{N_{k-1}}$, podemos obtener la resolución al nivel k usando técnicas interpolatorias 2D estándares:

$$\left(P_{k-1}^k \bar{f}^{k-1}\right)_{2i-1,2j} = \sum_{l=1}^{s} \beta_l \left(\bar{f}_{i+l-1,j}^{k-1} + \bar{f}_{i-l,j}^{k-1}\right) \tag{2.4}$$

$$\left(P_{k-1}^k \bar{f}^{k-1}\right)_{2i,2j-1} = \sum_{l=1}^{s} \beta_l \left(\bar{f}_{i,j+l-1}^{k-1} + \bar{f}_{i,j-l}^{k-1}\right) \tag{2.5}$$

$$\begin{aligned}\left(P_{k-1}^k \bar{f}^{k-1}\right)_{2i-1,2j-1} = \sum_{l=1}^{s} \beta_l \sum_{m=1}^{s} \beta_m \left(\bar{f}_{i+l-1,j+m-1}^{k-1}\right. \\ \left. + \bar{f}_{i-l,j+m-1}^{k-1} + \bar{f}_{i+l-1,j-m}^{k-1} + \bar{f}_{i-l,j-m}^{k-1}\right)\end{aligned} \tag{2.6}$$

donde los coeficientes β son:

$$\begin{cases} s=1 \Rightarrow \beta_1 = \frac{1}{2} \\ s=2 \Rightarrow \beta_1 = \frac{9}{16}, \beta_2 = -\frac{1}{16} \\ s=3 \Rightarrow \beta_1 = \frac{150}{256}, \beta_2 = -\frac{25}{256}, \beta_3 = \frac{3}{256} \end{cases} \tag{2.7}$$

Si definimos los errores de predicción $e^k_{i,j} = \bar{f}^k_{i,j} - \left(P^k_{k-1}\bar{f}^{k-1}\right)_{i,j}$, considerando sólo

$$d^k_{i,j}(1) = e^k_{2i-1,2j-1},\ d^k_{i,j}(2) = e^k_{2i-1,2j}\ d^k_{i,j}(3) = e^k_{2i,2j-1}.$$

(Notar que $e^k_{2i,2j} = \bar{f}^k_{2i,2j} - \left(P^k_{k-1}\bar{f}^{k-1}\right)_{2i,2j} = \bar{f}^{k-1}_{i,j} - \bar{f}^{k-1}_{i,j} = 0$), se obtiene inmediatamente una correspondencia entre:

$$\left\{\bar{f}^k_{i,j}\right\} \leftrightarrow \left\{\bar{f}^{k-1}_{i,j}, d^k_{i,j}(1), d^k_{i,j}(2), d^k_{i,j}(3)\right\}$$

En la figura 2.4 se puede ver un ejemplo de cómo quedaría una imagen después de aplicarle este esquema de multirresolución y cómo quedan colocadas las diferencias.

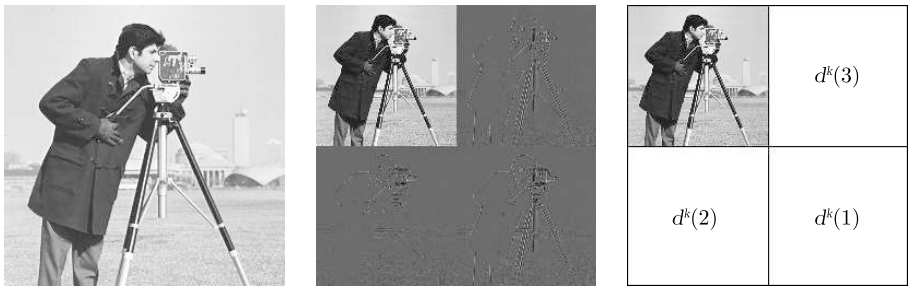


imagen original multirresolución esquema

Figura 2.4: *Izquierda: imagen original 'cameraman' (257×257), centro: resultado de realizar una transformación multiescala basada en valores puntuales, derecha: esquema que muestra la colocación de las distintas diferencias.*

Podemos reconstruir $\left\{\bar{f}^k_{i,j}\right\}$ a partir de la parte derecha de la

expresión anterior con

$$\begin{cases} \bar{f}^k_{2i,2j} = \bar{f}^{k-1}_{i,j} \\ \bar{f}^k_{2i-1,2j-1} = d^k_{i,j}(1) + \left(P^k_{k-1}\bar{f}^{k-1}\right)_{2i-1,2j-1} \\ \bar{f}^k_{2i-1,2j} = d^k_{i,j}(2) + \left(P^k_{k-1}\bar{f}^{k-1}\right)_{2i-1,2j} \\ \bar{f}^k_{2i,2j-1} = d^k_{i,j}(3) + \left(P^k_{k-1}\bar{f}^{k-1}\right)_{2i,2j-1} \end{cases}$$

Repitiendo el proceso para $k = L, \ldots, 1$ obtenemos la transformada 2D directa. Y si reconstruimos para $k = 1, \ldots, L$ tenemos la transformada 2D inversa. En la figura 2.5 podemos ver un ejemplo de una descomposición multiescala de la imagen cameraman con tres niveles de transformación.

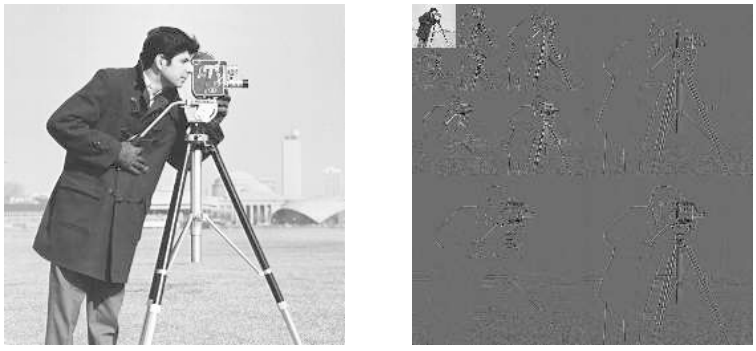


imagen original multirresolución con $L = 3$

Figura 2.5: *La imagen de la derecha muestra la transformada 2D directa de 'cameraman' con* $L = 3$.

La transformación multiescala 2D y su inversa pueden ser escritas de las siguiente manera:

ALGORITMO 2.1.
Transformación directa 2D $\bar{f}^L \longrightarrow M\bar{f}^L = (\bar{f}^0, d^1, \ldots, d^L)$

for $k = L, ..., 1$
  for $i, j = 0, ..., N_{k-1}$
    $\bar{f}^{k-1}_{i,j} = \bar{f}^k_{2i,2j}$
  end

for $i, j = 1, ..., N_{k-1}$

$d^k_{i,j}(2) = \bar{f}^k_{2i-1,2j} - \left(P^k_{k-1}\bar{f}^{k-1}\right)_{2i-1,2j}$

$d^k_{i,j}(3) = \bar{f}^k_{2i,2j-1} - \left(P^k_{k-1}\bar{f}^{k-1}\right)_{2i,2j-1}$

$d^k_{i,j}(1) = \bar{f}^k_{2i-1,2j-1} - \left(P^k_{k-1}\bar{f}^{k-1}\right)_{2i-1,2j-1}$

end

for $l = 1, ..., N_{k-1}$

$d^k_{l,0}(2) = \bar{f}^k_{2l-1,0} - \left(P^k_{k-1}\bar{f}^{k-1}\right)_{2l-1,0}$

$d^k_{0,l}(3) = \bar{f}^k_{0,2l-1} - \left(P^k_{k-1}\bar{f}^{k-1}\right)_{0,2l-1}$

end

end

ALGORITMO 2.2.
Transformación inversa 2D $M\bar{f}^L = (\bar{f}^0, d^1, \ldots, d^L) \longrightarrow M^{-1}M\bar{f}^L$

for $k = 1, ..., L$

for $i, j = 0, ..., N_{k-1}$

$\bar{f}^k_{2i,2j} = \bar{f}^{k-1}_{i,j}$

end

for $i, j = 1, ..., N_{k-1}$

$\bar{f}^k_{2i-1,2j} = d^k_{i,j}(2) + \left(P^k_{k-1}\bar{f}^{k-1}\right)_{2i-1,2j}$

$\bar{f}^k_{2i,2j-1} = d^k_{i,j}(3) + \left(P^k_{k-1}\bar{f}^{k-1}\right)_{2i,2j-1}$

$\bar{f}^k_{2i-1,2j-1} = d^k_{i,j}(1) - \left(P^k_{k-1}\bar{f}^{k-1}\right)_{2i-1,2j-1}$

end

for $l = 1, ..., N_{k-1}$

$\bar{f}^k_{2l-1,0} = d^k_{l,0}(2) + \left(P^k_{k-1}\bar{f}^{k-1}\right)_{2l-1,0}$

$\bar{f}^k_{0,2l-1} = d^k_{0,l}(3) + \left(P^k_{k-1}\bar{f}^{k-1}\right)_{0,2l-1}$

end

end

2.4.1
Algoritmo error control para valores puntuales

Dada una sucesión discreta $\bar{f}^L$ y un vector de tolerancias $\bar{\varepsilon} = \{\varepsilon_0, \ldots, \varepsilon_L\}$, nos gustaría conseguir una representación comprimida $\left\{\hat{f}^0, \hat{d}^1, \ldots, \hat{d}^L\right\}$ que esté cerca de la original, en el sentido que para alguna norma discreta $\|\cdot\|$, tengamos $\left\|\bar{f}^0 - \hat{f}^0\right\| \leqslant \varepsilon_0$ y $\left\|\bar{d}(l)^k - \hat{d}(l)^k\right\| \leqslant \varepsilon_k,\ 1 \leqslant k \leqslant L,\ \ l = 1, 2, 3$. Al mismo tiempo esperamos que la representación sea mucho más eficiente, es decir, mucho más barata de codificar y/o almacenar. Esperamos que el contenido de la información de $\hat{f}^L = M^{-1}\left\{\hat{f}^0, \hat{d}^1, \ldots, \hat{d}^L\right\}$ esté razonablemente cerca de la señal original , y para que esto sea cierto, la estabilidad del esquema de multirresolución con respecto a las perturbaciones es esencial, i.e., se pide que

$$\exists C > 0\,;\ \left\|\bar{f}^L - \hat{f}^L\right\| \leqslant C\varepsilon.$$

Motivado por el estudio de la estabilidad de las transformaciones de multirresolución no lineales, Ami Harten propone una vía sistemática que asegura una condición como esta última. Su aproximación implica una modificación de la transformación directa que incorpora el cuantizador no reversible $M_{\bar{\varepsilon}}^M$. Esta transformación modificada se diseña de tal modo que, si $\bar{\varepsilon} = \{0, \ldots, 0\}$ entonces $M_{\bar{0}}^M = Q_{\bar{0}}M = M$ y que la modificación nos permita seguir la pista del error acumulado. La ventaja adicional de esta modificación es que permite un seguimiento detallado de los errores acumulados locales y, por tanto, produce cotas de error *a priori* y *a posteriori* específicas (y computables). Probaremos que este algoritmo asegura evaluaciones "exactas" de error en las normas L_1, L_2, L_∞.

ALGORITMO 2.3. $\bar{f}^L \longrightarrow M_{\bar{\varepsilon}}^M \bar{f}^L = (\hat{f}^0, \hat{d}^1, \ldots, \hat{d}^L)$
Transformación directa 2D modificada

for $k = L, ..., 1$

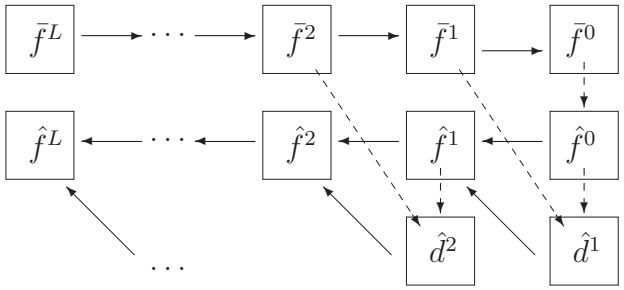


Figura 2.6: *Transformación directa 2D modificada.*

for $i, j = 0, ..., N_k$

$\bar{f}^{k-1}_{i,j} = \bar{f}^k_{2i,2j}$

end

end

$\hat{f}^0 = \text{proc}(\bar{f}^0, \varepsilon_0)$

for $k = 1, ..., L$

for $i = 1, ..., N_{k-1}, j = 0, ..., N_{k-1}$

$\tilde{d}^k_{i,j}(2) = \bar{f}^k_{2i-1,2j} - \left(P^k_{k-1}\hat{f}^{-1}\right)_{2i-1,2j}$

end

for $i = 0, ..., N_{k-1}, j = 1, ..., N_{k-1}$

$\tilde{d}^k_{i,j}(3) = \bar{f}^k_{2i,2j-1} - \left(P^k_{k-1}\hat{f}^{k-1}\right)_{2i,2j-1}$

end

for $i = 1, ..., N_{k-1}, j = 1, ..., N_{k-1}$

$\tilde{d}^k_{i,j}(1) = \bar{f}^k_{2i-1,2j-1} - \left(P^k_{k-1}\hat{f}^{k-1}\right)_{2i-1,2j-1}$

end

$\hat{d}^k(2) = \textbf{proc}(\tilde{d}^k(2), \varepsilon_k)$

$\hat{d}^k(3) = \textbf{proc}(\tilde{d}^k(3), \varepsilon_k)$

$\hat{d}^k(1) = \textbf{proc}(\tilde{d}^k(1), \varepsilon_k)$

for $i = 1, ..., N_{k-1}, j = 0, ..., N_{k-1}$

$\hat{f}^k_{2i-1,2j} = \hat{d}^k_{i,j}(2) + \left(P^k_{k-1}\bar{f}^{k-1}\right)_{2i-1,2j}$

end
for $i = 0, ..., N_{k-1}, j = 1, ..., N_{k-1}$
$\hat{f}^k_{2i,2j-1} = \hat{d}^k_{i,j}(3) + \left(P^k_{k-1}\bar{f}^{k-1}\right)_{2i,2j-1}$
end
for $i = 1, ..., N_{k-1}, j = 1, ..., N_{k-1}$
$\hat{f}^k_{2i-1,2j-1} = \hat{d}^k_{i,j}(1) + \left(P^k_{k-1}\bar{f}^{k-1}\right)_{2i-1,2j-1}$
end
for $i, j = 0, ..., N_{k-1}$
$\hat{f}^k_{2i,2j} = \bar{f}^{k-1}_{i,j}$
end
end

donde **proc** representa cualquier estrategia de procesamiento. Por ejemplo, podemos truncar o cuantizar los detalles. Esto es:

$$\textbf{proc\_tr}(d^k_{i,j};\varepsilon_k) = \begin{cases} 0 & \left|d^k_{i,j}\right| \leqslant \varepsilon_k \\ d^k_{i,j} & \left|d^k_{i,j}\right| > \varepsilon_k \end{cases} \tag{2.8}$$

$$\text{o} \quad \textbf{proc\_qu}(d^k_{i,j};\varepsilon_k) = 2\varepsilon \cdot \left[\frac{d^k_{i,j}}{2\varepsilon}\right] \tag{2.9}$$

Veremos a continuación una evaluación del error *a posteriori* que nos permite calcular el error de compresión exacto medido en las normas discretas L_1, L_2, L_∞. La definición de estas normas tiene en cuenta la dimensionalidad de las sucesiones finitas para las que se aplican. Así, dada una sucesión de dimensión finita $s = (s_\lambda)_{\lambda\in\Lambda}$, donde Λ es un conjunto (finito) de índices, consideramos las siguientes definiciones:

$$\|s\|_\infty = \max_{\lambda\in\Lambda}|s_\lambda| \qquad \|s\|_1 = \frac{1}{|\Lambda|}\sum_{\lambda\in\Lambda}|s_\lambda| \qquad \|s\|_2^2 = \frac{1}{|\Lambda|}\sum_{\lambda\in\Lambda}|s_\lambda|^2$$

Ahora, veamos unas proposiciones:

Proposición 2.1. *Dada una sucesión discreta $\bar{f}^L$, sea $\hat{f}^L = M^{-1}M^M_{\bar{\varepsilon}}\bar{f}^L$ con $M^M_{\bar{\varepsilon}}$ dada por el algoritmo 2.3 y M^{-1} dada por el algoritmo 2.2 (y siendo $p = 1$ o 2). Entonces:*

$$\left\|\bar{f}^L - \hat{f}^L\right\|_\infty = \max\left\{\left\|\bar{f}^0 - \hat{f}^0\right\|_\infty, \max_k\left\{\max_{l=1,2,3}\left\{\left\|\tilde{d}^k(l) - \hat{d}^k(l)\right\|_\infty\right\}\right\}\right\} \tag{2.10}$$

$$\begin{aligned}\left\|\bar{f}^L - \hat{f}^L\right\|_p^p = \frac{(N_0+1)^2}{(N_L+1)^2}\left\|\bar{f}^0 - \hat{f}^0\right\|_p^p + \sum_{k=1}^{L}\frac{N_{k-1}^2}{(N_L+1)^2}\left\|\tilde{d}^k(1) - \hat{d}^k(1)\right\|_p^p + \\ + \sum_{k=1}^{L}\frac{N_{k-1}^2 + N_{k-1}}{(N_L+1)^2}\left\|\tilde{d}^k(2) - \hat{d}^k(2)\right\|_p^p + \sum_{k=1}^{L}\frac{N_{k-1}^2 + N_{k-1}}{(N_L+1)^2}\left\|\tilde{d}^k(3) - \hat{d}^k(3)\right\|_p^p.\end{aligned} \tag{2.11}$$

Demostración. Del algoritmo 2.3 deducimos:

$$\begin{aligned}\bar{f}^k_{2i-1,2j} - \hat{f}^k_{2i-1,2j} &= \tilde{d}^k_{i,j}(2) - \hat{d}^k_{i,j}(2),\\ \bar{f}^k_{2i,2j-1} - \hat{f}^k_{2i,2j-1} &= \tilde{d}^k_{i,j}(3) - \hat{d}^k_{i,j}(3),\\ \bar{f}^k_{2i-1,2j-1} - \hat{f}^k_{2i-1,2j-1} &= \tilde{d}^k_{i,j}(1) - \hat{d}^k_{i,j}(1),\\ \bar{f}^k_{2i,2j} - \hat{f}^k_{2i,2j} &= \bar{f}^{k-1}_{i,j} - \hat{f}^{k-1}_{i,j},\end{aligned} \tag{2.12}$$

Entonces:

$$\begin{aligned}\left\|\bar{f}^L - \hat{f}^L\right\|_\infty &= \max_{i,j}\left|\bar{f}^L_{i,j} - \hat{f}^L_{i,j}\right| =\\ &= \max_{i,j,l}\left\{\left|\bar{f}^{L-1}_{i,j} - \hat{f}^{L-1}_{i,j}\right|, \left|\tilde{d}^k(l) - \hat{d}^k(l)\right|\right\} =\\ &= \max_{i,j,l}\left\{\left\|\bar{f}^{L-1} - \hat{f}^{L-1}\right\|_\infty, \max_{l=1,2,3}\left\{\left\|\tilde{d}^L(l) - \hat{d}^L(l)\right\|_\infty\right\}\right\}\end{aligned}$$

y obtenemos (2.10). Ahora para $p = 1$ (o 2),

$$\begin{aligned}
\left\|\bar{f}^L - \hat{f}^L\right\|_p^p &= \frac{1}{(N_L+1)^2}\sum_{i=0,j=0}^{N_L}\left|\bar{f}^L_{i,j} - \hat{f}^L_{i,j}\right|^p = \\
&= \frac{1}{(N_L+1)^2}\sum_{i=0,j=0}^{N_{L-1}}\left|\bar{f}^{L-1}_{i,j} - \hat{f}^{L-1}_{i,j}\right|^p + \\
&\quad + \frac{1}{(N_L+1)^2}\sum_{i=1,j=1}^{N_{L-1}}\left|\tilde{d}^{L-1}_{i,j}(1) - \hat{d}^{L-1}_{i,j}(1)\right|^p + \\
&\quad + \frac{1}{(N_L+1)^2}\sum_{i=1,j=0}^{N_{L-1}}\left|\tilde{d}^{L-1}_{i,j}(2) - \hat{d}^{L-1}_{i,j}(2)\right|^p + \\
&\quad + \frac{1}{(N_L+1)^2}\sum_{i=0,j=1}^{N_{L-1}}\left|\tilde{d}^{L-1}_{i,j}(3) - \hat{d}^{L-1}_{i,j}(3)\right|^p = \\
&= \frac{(N_{L-1}+1)^2}{(N_L+1)^2}\left\|\bar{f}^{L-1} - \hat{f}^{L-1}\right\|_p^p + \\
&\quad + \frac{(N_{L-1})^2}{(N_L+1)^2}\left\|\tilde{d}^L(1) - \hat{d}^L(1)\right\|_p^p + \\
&\quad + \frac{N_{L-1}^2 + N_{L-1}}{(N_L+1)^2}\left\|\tilde{d}^L(2) - \hat{d}^L(2)\right\|_p^p + \\
&\quad + \frac{N_{L-1}^2 + N_{L-1}}{(N_L+1)^2}\left\|\tilde{d}^L(3) - \hat{d}^L(3)\right\|_p^p,
\end{aligned}$$

lo que prueba (2.11). □

Esta proposición da la diferencia exacta entre $\bar{f}^L$ y $\hat{f}^L$ medida en las normas L_1, L_2, L_∞. Este valor puede ser incorporado a la representación comprimida de los datos, para que después de aplicar $M^M_{\tilde{\varepsilon}}$, uno sepa exactamente cuánta información se ha perdido respecto a estas normas. En particular, como el PSNR se define utilizando el error cuadrático medio que es $\left\|\bar{f}^L - \hat{f}^L\right\|_2$, no es necesario aplicar la transformada inversa (algoritmo 2.2) para conocer el PSNR. Lo mismo pasa con el PAE que se mide con la norma infinito.

Estos resultados se pueden utilizar para dar cotas *a priori*, i.e. pueden ayudar a ajustar el vector de parámetros $\bar{\varepsilon} = \{\varepsilon_0, \ldots, \varepsilon_L\}$ para poder garantizar una precisión deseada al descomprimir.

Corolario 2.2. *Considerar el esquema de multirresolución con error control descrito en la proposición 2.1, y cualquier estrategia de procesamiento para reducir los coeficientes tal que*

$$\left\|\bar{f}^0 - \hat{f}^0\right\|_p \leqslant \varepsilon_0 \text{ y } \left\|\tilde{d}^k(l) - \hat{d}^k(l)\right\|_p \leqslant \varepsilon_k \quad \forall l = 1,2,3, \quad \forall k = 1, ..., L$$

donde $p = 1$ *o* 2*. Entonces, tenemos:*

$$\left\|\bar{f}^L - \hat{f}^L\right\|_p \leqslant \varepsilon = \max\{\varepsilon_k\}.$$

Más aún, $PSNR \geqslant 20\log_{10}(255/\varepsilon)$*.*

Demostración. Como $N_k = 2^k N_0$ tenemos

$$\begin{aligned}\left\|\bar{f}^L - \hat{f}^L\right\|_p^p &\leqslant \varepsilon^p \frac{(N_0+1)^2}{(N_L+1)^2} + \frac{\varepsilon^p}{(N_L+1)^2}\sum_{k=1}^{L}(3(N_{k-1})^2 + 2N_{k-1}) = \\ &= \varepsilon^p \frac{(N_0+1)^2}{(N_L+1)^2} + \frac{\varepsilon^p\, 3N_0^2}{(N_L+1)^2}\sum_{k=1}^{L}4^{k-1} + \frac{\varepsilon^p\, 2N_0^2}{(N_L+1)^2}\sum_{k=1}^{L}2^{k-1} = \\ &= \frac{\varepsilon^p}{(N_L+1)^2}\left((N_0+1)^2 + N_0^2(4^L-1) + 2N_0(2^L-1)\right) = \varepsilon^p\end{aligned}$$

□

Corolario 2.3. *(estrategia casi sin pérdida) Considerar el esquema de multirresolución con error control descrito en la proposición 1 y la cuantización* ***proc_qu*** *como estrategia de procesamiento del algoritmo 2.3. Entonces, tenemos:*

$$\left\|\bar{f}^0 - \hat{f}^0\right\|_\infty \leqslant \varepsilon_0 \text{ y } \max_{l=1,2,3}\left\{\left\|\tilde{d}^k(l) - \hat{d}^k(l)\right\|_p\right\} \leqslant \varepsilon_k.$$

Por tanto:

$$PAE = \left\|\bar{f}^L - \hat{f}^L\right\|_\infty \leqslant \varepsilon = \max\{\varepsilon_k\}.$$

Para obtener una transformación sin pérdidas necesitamos introducir dos variantes del procedimiento de cuantización **p**roc_qu:

$$\textbf{proc\_up}(d_{i,j}^k;\varepsilon_k) = \begin{cases} d_{i,j}^k + \varepsilon_k & \text{si } \left| \left[\frac{d_{i,j}^k}{2\varepsilon_k}\right] - \frac{d_{i,j}^k}{2\varepsilon_k} \right| = \frac{1}{2} \\ 2\varepsilon_k \cdot \left[\frac{d_{i,j}^k}{2\varepsilon_k}\right] & \text{en otro caso} \end{cases}$$

$$\textbf{proc\_do}(d_{i,j}^k;\varepsilon_k) = \begin{cases} d_{i,j}^k - \varepsilon_k & \text{si } \left| \left[\frac{d_{i,j}^k}{2\varepsilon_k}\right] - \frac{d_{i,j}^k}{2\varepsilon_k} \right| = \frac{1}{2} \\ 2\varepsilon_k \cdot \left[\frac{d_{i,j}^k}{2\varepsilon_k}\right] & \text{en otro caso} \end{cases}$$

Corolario 2.4. *(Transformación sin pérdida) Consideramos el esquema de multirresolución con error control descrito en la proposición 1 y la cuantización* ***proc_up*** *con* $\varepsilon_k = 0{,}5$*. Entonces:*

$$\bar{f}^L = \textbf{\textit{proc\_do}}(\hat{f}^L; 0{,}5).$$

Demostración. Si aplicamos la estrategia **proc_up** en el algoritmo 2.3 obtenemos:

$$-0{,}5 < \hat{f}_{i,j}^0 - \bar{f}_{i,j}^0 \leqslant 0{,}5 \quad \text{y} \quad -0{,}5 < \hat{d}_{i,j}^0 - \tilde{d}_{i,j}^0 \leqslant 0{,}5$$

y, de (2.12):

$$-0{,}5 < \hat{f}_{i,j}^L - \bar{f}_{i,j}^L \leqslant 0{,}5$$

Ahora, si definimos $\breve{f} := \textbf{proc\_do}(\hat{f}^L; 0{,}5)$, tenemos

$$-0{,}5 \leqslant \breve{f}_{i,j}^L - \hat{f}_{i,j}^L < 0{,}5 \quad \text{y} \quad -1 < \breve{f}_{i,j}^L - \bar{f}_{i,j}^L < 1.$$

Como $\breve{f}_{i,j}^L, \bar{f}_{i,j}^L \in \mathbb{N}$. Tenemos $\bar{f}^L = \breve{f}^L = \textbf{proc\_do}((\hat{f}^L; 0{,}5))$. □

2.4.2 Experimentos numéricos con valores puntuales

En primer lugar compararemos los errores de compresión que se producen al descomponer y reconstruir la imagen Lena con diferentes tolerancias ε y con dos estrategias diferentes:

1. Aplicamos la estrategia estándar, a la que llamamos **ST**, que consiste simplemente en computar $\hat{f}^L = M^{-1}Q_\varepsilon M\bar{f}^L$ donde M y M^{-1} representan los algoritmos 2.1 y 2.2, respectivamente, y Q_ε es el operador cuantización.

2. Aplicamos la transformada discreta modificada 2D (algoritmo 2.3) y la transformada inversa 2D (algoritmo 2.2). Denotamos esta estrategia como **EC** (error control).

Usamos las mismas técnicas interpolatorias polinomiales a trozos descritas en (2.4), (2.5) y (2.6). Comparamos el rendimiento de compresión de cada estrategia en términos de la entropía.

En la tabla 2.1 y en la figura 2.7 podemos observar los efectos de las diferentes estrategias con varias tolerancias utilizando $s = 1$ (2.7). En la tabla 2.2 se muestran los resultados para $s = 2$. En ambos casos se ha empleado $L = 4$.

Aunque si controlamos el error no podemos controlar la compresión, podemos observar que dado un ε, la entropía, y por tanto la compresión, obtenida por **EC** es comparable a la entropía obtenida con la estrategia **ST**.

Con el algoritmo **EC** siempre obtenemos exactamente la cota teórica, ε, para la norma máximo. En cambio, la estrategia **ST** siempre nos da errores con la norma máximo mucho mayores que el parámetro de tolerancia.

En la tabla 2.3 mostramos la tasa de compresión en bits por píxel y el PSNR de la imagen Lena al comprimirla utilizando error control (EC) (y el algoritmo de compresión SPIHT) con varios valores de ε. También se muestran los valores obtenidos con JPEG-LS que tiene un modo casi sin pérdida donde se puede determinar el error máximo permitido antes de que se realice la compresión, como con el algoritmo que proponemos.

Se puede observar que los valores del PSNR son mayores con el algoritmo EC y que cuando aumenta la tolerancia, las tasas de compresión del método EC son mejores. En la figura 2.8 se muestra un detalle de la reconstrucción de Lena y se observa en la que corresponde al método JPEG-LS unas líneas horizontales poco naturales.

Ahora procesamos la imagen de la mamografía (Figura 2.10 arriba, izda.) usando la transformación $M_{\bar{\varepsilon}}^M$ descrita en el Algo-

ε	algoritmo	$\|\cdot\|_\infty$	$\|\cdot\|_1$	$\|\cdot\|_2$	entropía	PSNR
4	**ST**	11,94	2,46	3,04	1,79	38,46
	EC	4	1,93	2,25	1,83	41,07
8	**ST**	23,64	4,02	5,13	0,95	33,93
	EC	8	3,39	4,06	1,07	35,95
12	**ST**	35,78	5,07	6,68	0,63	31,64
	EC	12	4,76	5,75	0,73	32,94
16	**ST**	42,13	6,09	8,22	0,46	29,84
	EC	16	6,02	7,32	0,55	30,84
20	**ST**	54,38	7,06	9,65	0,36	28,44
	EC	20	7,60	9,14	0,43	28,91

Tabla 2.1: *Errores de compresión de la imagen Lena utilizando los algoritmos **ST** y **EC** con valores puntuales ($s = 1$) y con diferentes valores del parámetro ε.*

ε	algoritmo	$\|\cdot\|_\infty$	$\|\cdot\|_1$	$\|\cdot\|_2$	entropía	PSNR
4	**ST**	13,18	2,67	3,32	1,71	37,70
	EC	4	1,95	2,27	1,81	41,02
8	**ST**	26,02	4,43	5,66	0,90	33,07
	EC	8	3,53	4,19	1,08	35,68
12	**ST**	39,52	5,65	7,43	0,59	30,71
	EC	12	5,03	6,01	0,75	32,56
16	**ST**	49,35	6,84	9,17	0,43	28,89
	EC	16	6,42	7,73	0,58	30,36
20	**ST**	67,33	7,79	10,60	0,33	27,62
	EC	20	8,06	9,64	0,47	28,45

Tabla 2.2: *Errores de compresión de la imagen Lena utilizando los algoritmos **ST** y **EC** con valores puntuales ($s = 2$) y con diferentes valores del parámetro ε.*

Figura 2.7: *Imágenes reconstruidas (a)* ***ST*** *con* $\varepsilon = 12$*, entropía 0,6286 (b)* ***EC*** *con* $\varepsilon = 12$*, entropía 0,7284 (c)* ***ST*** *con* $\varepsilon = 16$*, entropía 0,4626 (d)* ***EC*** *con* $\varepsilon = 16$*, entropía 0,5498.*

	EC		JPEG-LS	
tol	**bpp**	**psnr**	**bpp**	**psnr**
1	3,12	52,90	2,75	49,89
2	2,60	46,90	2,12	45,14
3	2,12	43,43	1,78	42,20
4	1,77	41,08	1,55	40,10
5	1,49	39,39	1,38	38,49
6	1,29	38,10	1,25	37,14
7	1,12	37,02	1,15	36,00
8	0,95	36,30	1,07	35,03
9	0,87	35,77	1,00	34,13
10	0,75	35,30	0,94	33,31

Tabla 2.3: *Tasa de compresión (bpp) y PSNR obtenido después de aplicar EC con valores puntuales y $s = 1$ y JPEG-LS a la imagen Lena con diferentes valores del parámetro ε.*

(a) (b)

Figura 2.8: *Detalle de la reconstrucción de la imagen Lena utilizando (a) Error control (b) JPEG-LS.*

ritmo 2.3 con $s = 1$ en (2.7). Notar que, según el corolario 2.3, el parámetro de cuantización $\bar{\varepsilon}$ especifica la máxima desviación permitida entre los valores de los píxeles. Aplicamos a $M_{\bar{\varepsilon}}^{M} \bar{f}^{L}$ un código entrópico *lossless* similar al usado en el algoritmo SPIHT [48] que también explota la estructura de multirresolución. En la Tabla 2.4, mostramos el ratio *bit-per-pixel* (*bpp*) y el PSNR de la imagen reconstruida para varios valores de $\bar{\varepsilon}$, junto con los valores obtenidos después de utilizar el compresor estándar lossless **JPEG-LS** (ver [57, 58]).

También hemos comprimido la imagen con el estándar **JPEG-2000** [54] utilizando el software Jasper [2] con el mismo ratio de compresión que el obtenido con el **EC** (columna 7 de la Tabla 2.4). Observar que con el **JPEG-2000** no podemos controlar automáticamente el Peak Absolute Error (PAE) (columna 9 de la Tabla 2.4). Resolvemos este problema evaluando los píxeles que tienen el error más grande que la tolerancia ε y los codificamos con un código aritmético [39, 40]. Obviamente en este caso tenemos más bytes que almacenar (columna 10 de la tabla) y de aquí, el bpp es mayor (columna 11).

		EC		**JPEG-LS**		**JPEG-2000**					
ε	LPB	bpp	PSNR	bpp	PSNR	bpp	PSNR	PAE	extra bytes	bpp	PSNR
2	42.11	1.03	49.23	0.94	45.78	1.03	47.39	7	12951	1.41	50.31
4	36.09	0.67	43.44	0.67	39.21	0.67	44.28	10	7013	0.88	45.73
6	32.57	0.44	40.06	0.42	36.57	0.44	42.45	16	4017	0.56	43.30
8	30.07	0.30	38.88	0.34	35.25	0.30	41.57	19	1742	0.35	41.94
10	28.13	0.17	38.10	0.26	32.67	0.17	40.69	24	658	0.19	40.81
12	26.55	0.09	37.82	0.22	31.70	0.09	39.99	27	272	0.10	40.03

Tabla 2.4: *Ratios de compresión (bbp) y PSNR obtenido después de aplicar los algoritmos de compresión* **EC**, **JPEG-LS** *y* **JPEG-2000**, *con diferentes parámetros de truncamiento ε, a la imagen medica (Figura 2.10, Arriba Izda.). LPB es la Cota Inferior de PSNR (Lower PSNR Bound) para el algoritmo* **EC** *para cada ε (Corolario 2.2).*

En la Figura 2.9 mostramos una comparación de los resultados obtenidos para comprimir casi-lossless con las estrategias

EC, **JPEG-LS** y **JPEG-2000**. A pesar de que nuestra técnica no

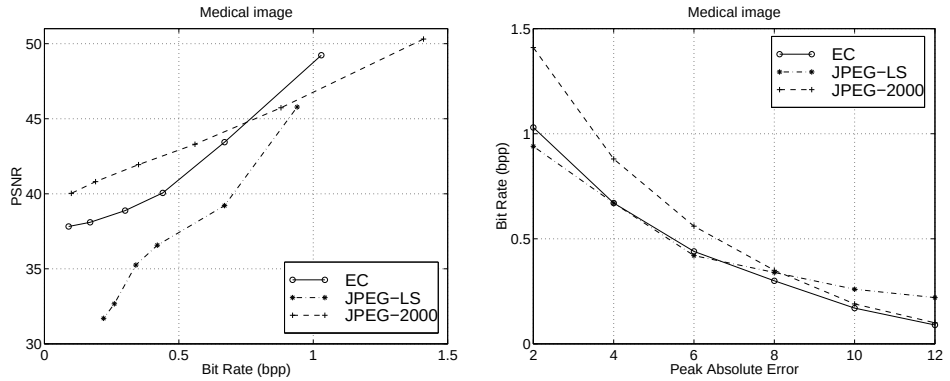


Figura 2.9: *Comparación de los resultados obtenidos con el casi-lossless en la imagen test medica. Izda., PSNR versus bit rate; dcha, bit rate versus Peak Absolute Error.*

es muy sofisticada, los ratios de compresión que obtenemos con los algoritmos **EC** y **JPEG-LS** son muy similares. Podemos ver también que para un ε grande nuestro algoritmo mejora al casi-lossless **JPEG-LS**. Hay, sin embargo, una diferencia significativa en el PSNR, y también en la calidad de la imagen reconstruida (ver la Figura 2.10).

En la imagen original un pequeño punto blanco puede ser claramente observado. Si esto indica un tumor o una anomalía, debería de ser claramente observado después de que la imagen es reconstruida, como ocurre en todos los casos. La máxima diferencia se ve en la textura de la mama, la cual se ve mejor cuando se utiliza el algoritmo **EC**.

Si comparamos nuestro algoritmo con los resultados obtenidos con el **JPEG-2000** observamos que con nuestros algoritmos, dado un PAE, comprimen más (el bpp es menor).

También aplicamos los tres algoritmos al conjunto de 12 imágenes DEM de 1200×1200 píxeles con 8 bits por píxel con un error máximo de $\varepsilon = 5$ (Tabla 2.5). Puede ser observado que el PSNR obtenido con los algoritmos **EC** y **JPEG-2000** son siempre mayores que el que obtenemos con el **JPEG-LS**. La calidad global de la imagen reconstruida siempre es mejor.

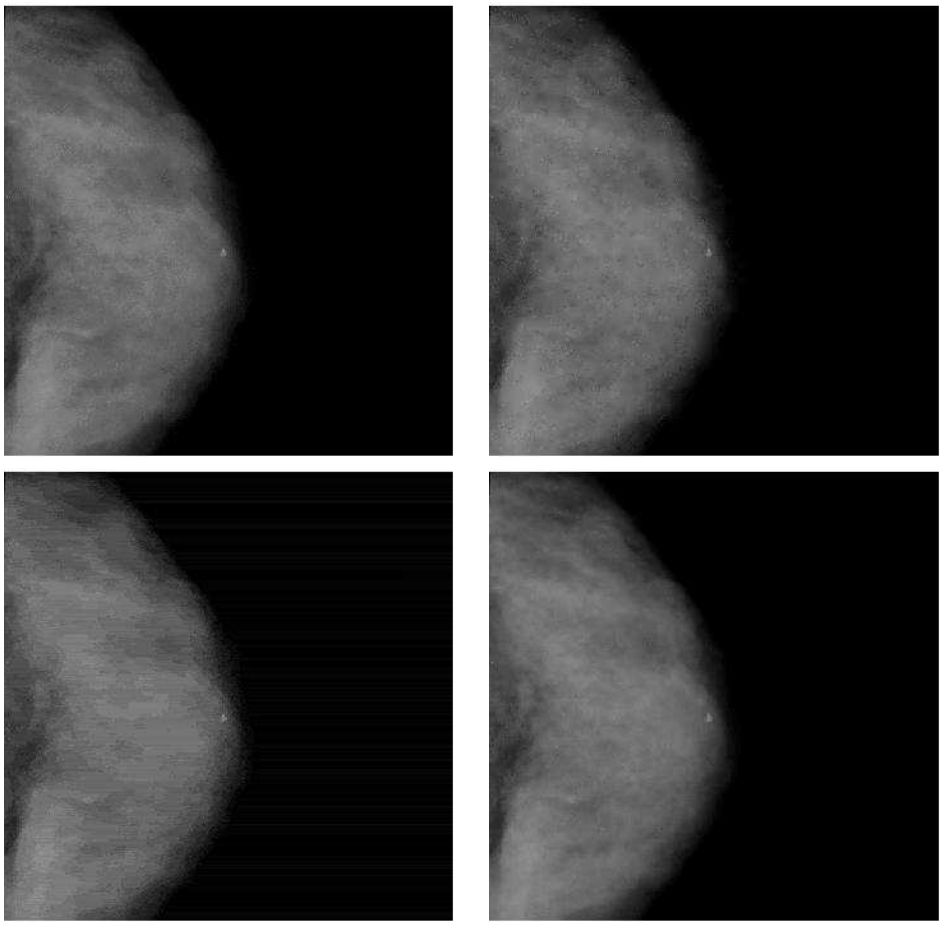


Figura 2.10: *Arriba Izda.: Imagen original médica (512×512 píxeles). Arriba derecha la misma después de comprimir con* **EC** *y* $\varepsilon = 10$ *(bpp=.17 y PSNR=38.10). Bajo izda.: con* **JPEG-LS** *y* $\varepsilon = 8$ *(bpp=.34 y PSNR=35.25). Bajo dcha: con* **JPEG-2000** *y* $\varepsilon = 10$ *(bpp=.19 y PSNR=40.81).*

	EC		**JPEG-LS**		**JPEG-2000**					
Imágenes	bpp	PSNR	bpp	PSNR	bpp	PSNR	PAE	extra bytes	bpp	PSNR
binghamton-e	.15	42.73	.63	38.77	.15	44.90	17	5284	.17	45.13
binghamton-w	.30	41.89	.81	38.81	.30	44.44	16	7814	.35	44.74
delta-e	.11	42.12	.37	39.16	.11	46.35	16	4109	.13	46.59
delta-w	.09	43.01	.37	38.88	.09	46.57	15	3646	.11	46.81
lake_champlain-e	.06	43.41	.34	38.91	.06	47.08	13	2012	.07	47.21
ogden-e	.10	43.10	.37	38.89	.10	46.54	13	5597	.13	46.90
ogden-w	.14	42.85	.61	38.79	.14	45.05	13	4787	.17	45.26
shiprock-w	.13	42.45	.38	38.99	.13	46.58	15	5925	.17	46.96
toole-e	.10	43.77	.40	39.12	.10	46.52	11	3806	.12	46.76
utica-e	.14	42.76	.55	38.90	.14	44.70	15	6984	.17	44.96
utica-w	.10	43.50	.37	38.60	.10	46.56	23	5284	.13	46.91
wells-e	.09	43.38	.42	38.81	.09	45.94	13	3907	.11	46.16

Tabla 2.5: *Ratios de compresión (bbp) y PSNR obtenido después de aplicar los algoritmos de compresión* **EC**, **JPEG-LS** *y* **JPEG-2000**, *con* $\varepsilon = 5$, *a los datos. La cota inferior del PSNR (LPB) para el* **EC** *es* $20\ log_{10}(255/\varepsilon) = 34{,}15$ *(Corolario 2.2).*

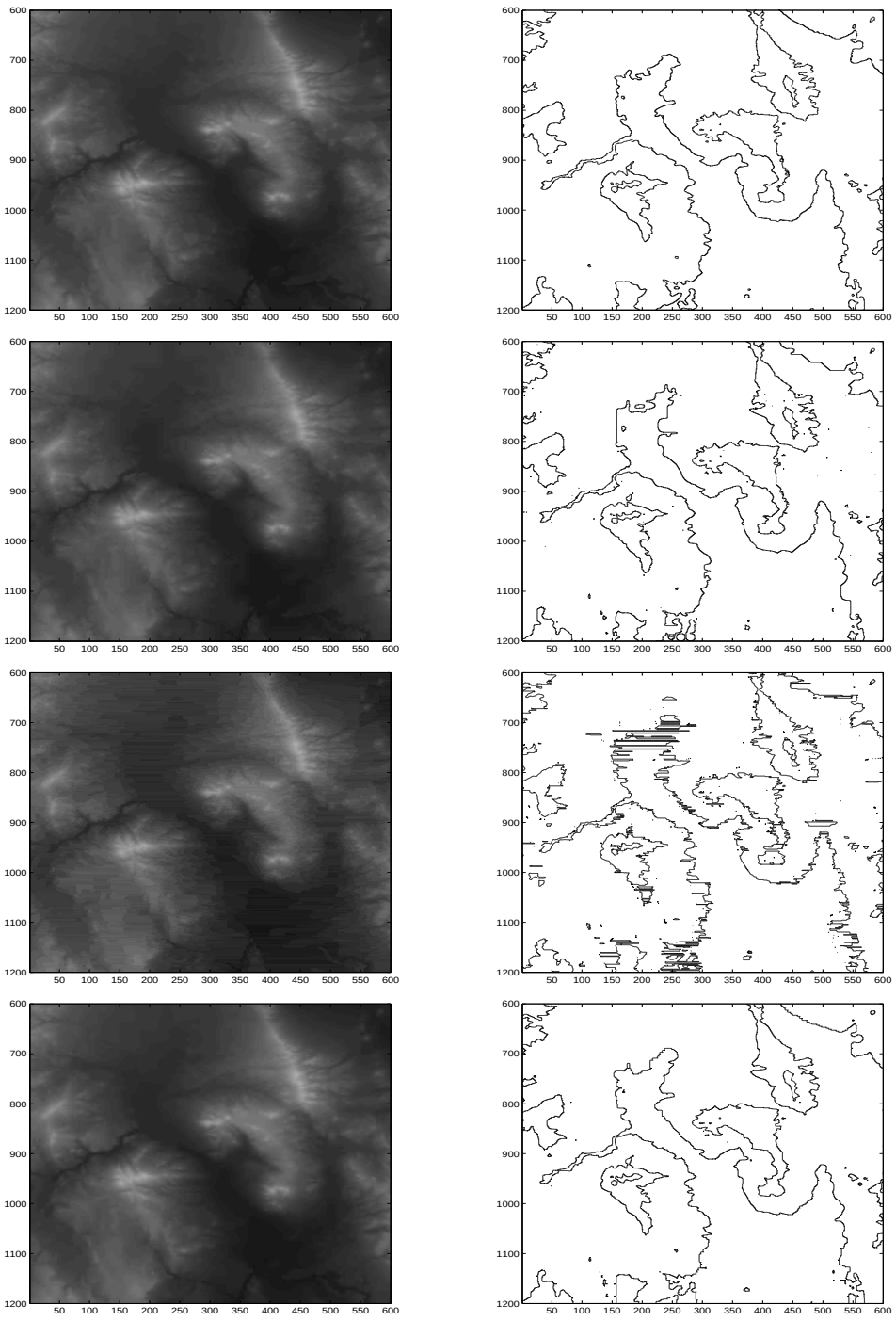


Figura 2.11: *De arriba a abajo (izda.): Región seleccionada de 600×600 píxeles de la imagen original wells-e2 (1200×1200 píxeles), la misma región después de ser comprimida con un error máximo de 5, con* **EC** *(bpp=.09 y PSNR=43.38), con* **JPEG-LS** *(bpp=.42 y PSNR=38.81) y con* **JPEG-2000** *(bpp=.11 y PSNR=46.16). Derecha: las líneas de nivel para cada imagen.*

Mostramos en la figura 2.11 la imagen original y la reconstruida después de ser comprimida con las estrategias **EC**, **JPEG-LS** y **JPEG-2000**. La figura muestra la imagen en escala de grises y sus curvas de nivel. Notar que la estructura topográfica está mejor preservada en el caso de la compresión con **EC** y **JPEG-2000** mientras que es distorsionada en el caso de la compresión **JPEG-LS**.

También hemos comparado la estrategia lossless **EC** del corolario 2.4 con el **JPEG-LS**. En la Tabla 2.6 podemos ver que con nuestra técnica obtenemos unos ratios de compresión un poco peores que con el **JPEG-LS**.

	EC	**JPEG-LS**		**EC**	**JPEG-LS**
binghamton-e	1.76	1.55	ogden-w	1.71	1.52
binghamton-w	2.30	1.98	ogden-e	1.35	1.16
delta-e	1.59	1.33	toole-e	1.50	1.29
delta-w	1.52	1.30	utica-e	1.73	1.50
lake_champlain-e	1.06	0.99	utica-w	1.38	1.20
shiprock-w	1.47	1.26	wells-e	1.44	1.27

Tabla 2.6: *Compresión para las estrategias lossless* **EC** *y* **JPEG-LS**.

Observamos que con el algoritmo **EC** siempre obtenemos la cota teórica *a priori*, ε, para la L_∞-norma. Esto es debido a el hecho de que si en un caso tenemos $|\tilde{d}^k_{i,j}(l) - \hat{d}^L_{i,j}(l)| = \varepsilon$ entonces, por (2.10), $\| \bar{f}^L - \hat{f}^L \|_\infty = \varepsilon$. El PSNR continua, en todos los casos, significativamente mayor que las cotas teóricas de error. Esto es esperable, ya que las cotas de error son calculadas en el Corolario 2.2 bajo el peor escenario. En una figura real, muchos errores de interpolación (y por tanto muchos detalles) son menores que la tolerancia, y su contribución al total del error L_2 (o L_1) es mucho menor de lo que sería en el peor escenario asumido en las pruebas.

El hecho de que las normas del error L_2 y L_1 (más apropiadas para la visualización de la imagen que la L_∞) son siempre mucho menores que sus cotas de error teóricas, sugiere que otras estrategias de procesamiento de los detalles pueden ser utilizadas dentro del contexto de error-control que aseguren que las normas

del error L_2 o L_1 estén por debajo de la tolerancia prescrita pero no demasiado lejana a ella.

En términos generales, necesitamos calcular los coeficientes de escala $\hat{d}^k$, utilizando algún procesamiento de $\tilde{d}^k$ de forma que

$$||\hat{d}^k - \tilde{d}^k||_p \leq \varepsilon_k$$

para $p = 2$ o $p = 1$.

Substituimos el cálculo de $\hat{d}^k_{i,j}(l)$, $l = 1, 2, 3$ obtenido por cuantización por la estrategia de procesamiento (2.13):

Corolario 2.5. *Consideramos el esquema de multirresolución de error control descrito en la Proposición 2.1 con la estrategia de procesamiento:*

$$\hat{d}^k(2) = \textbf{proc\_qu2A}(\tilde{d}^k(2), \varepsilon_2, \varepsilon_\infty) = \begin{cases} n = 0;\ \rho = 0 \\ \textbf{while}\ (\rho \leq \varepsilon_2\ \textbf{and}\ 2^n\varepsilon_2 \leq \varepsilon_\infty) \\ \quad \hat{d}^k(l) = \textbf{proc\_qu}(\tilde{d}^k(l), 2^n\varepsilon_2), \\ \quad \rho = ||\hat{d}^k(l) - \tilde{d}^k(l)||_2 \\ \quad n = n + 1 \\ \textbf{end} \end{cases} \tag{2.13}$$

Entonces, tenemos $||\bar{f}^L - \hat{f}^L||_2 \leq \varepsilon_2$ *y* $||\bar{f}^L - \hat{f}^L||_\infty \leq 2^n\varepsilon_2 \leq \varepsilon_\infty$.

Otra posible estrategia es la siguiente:

Corolario 2.6. *Consideremos el esquema de multirresolución de error control descrita en la Proposición: 2.1 y la estrategia de procesamiento:*

$$\hat{d}^k(l) = \textbf{proc\_qu2B}(\tilde{d}^k(l), \varepsilon_2, \varepsilon_\infty) = \begin{cases} \bar{d}^k(l) = \textbf{proc\_qu}(\tilde{d}^k(l), \varepsilon_2) \\ n = 0;\ \rho = 0 \\ \textbf{while}\ (\rho \leq \varepsilon_2\ \textbf{and}\ (\varepsilon_2 + 2n\varepsilon_2) \leq \varepsilon_\infty) \\ \quad n = n + 1 \\ \quad \hat{d}^k(l) = \textbf{proc\_tr}(\bar{d}^k(l), 2n\varepsilon_2), \\ \quad \rho = ||\hat{d}^k(l) - \tilde{d}^k(l)||_2 \\ \textbf{end} \end{cases} \tag{2.14}$$

Entonces, tenemos $||\bar{f}^L - \hat{f}^L||_2 \leq \varepsilon_2$ *y* $||\bar{f}^L - \hat{f}^L||_\infty \leq (\varepsilon_2 + 2n\varepsilon_2) \leq \varepsilon_\infty$.

Notar que estas estrategias, a les que llamaremos **EC2A** y **EC2B**, aumentan el parámetro utilizado para cuantizar o truncar en cada nivel de resolución hasta que $||\hat{d}^k(l) - \tilde{d}^k(l)||_\infty \leq \varepsilon_\infty$, pero que aseguran $||\hat{d}^k(l) - \tilde{d}^k(l)||_2 \leq \varepsilon_2$.

En la Tabla 2.7 observamos los efectos de los diferentes estrategias de procesamiento (dentro del algoritmo **EC**) para diferentes parámetros de tolerancia. En ambos casos vemos los errores actuales están más cercanos a las cotas teóricas de error que con la cuantización estándar **proc_qu** y, al mismo tiempo, obtenemos mejores ratios de compresión.

ε_∞	ε_2	strategy	LPB	PAE	PSNR	bpp
16	16	**EC**	24.09	16	36.95	0.035
16	16	**JPEG-LS**		16	29.24	0.161
16	4	**EC2A**	36.09	16	36.89	0.032
12	12	**EC**	26.55	12	37.82	0.09
12	12	**JPEG-LS**		12	31.70	0.22
12	4	**EC2B**	36.09	12	37.71	0.074

Tabla 2.7: *Ratios de compresión (bbp), PSNR y PAE obtenidos después de aplicar el* **JPEG-LS** *y el* **EC**, *con las estrategias* **EC2A** *y* **EC2B**, *y diferentes parámetros de truncamiento ε a la imagen médica. LPB es la cota inferior de PSNR para el algoritmo* **EC** *para cada ε (Corolario 2.2).*

2.5

Multirresolución basada en medias en celda

En el marco de las medias en celda, los datos discretos son interpretados como la media de una función en unas celdas de una malla, la cual determina el nivel de resolución de la información dada (ver [8]). Supongamos que disponemos de una sucesión de mallas $\left\{X^k\right\}_{k=0}^{L}$, donde $X^k = \left\{\left(x_i^k, y_j^k\right)\right\}, x_i^k = ih_k,\ y_j^k = jh_k,\ i, j =$

$1, \ldots, N_k,\ N_k h_k = 1$. Con $h_k = 2^{-k} h_0$, la sucesión de datos discretos $\bar{f}^k = \left\{ f^k_{i,j} \right\}$ en el k-ésimo nivel de resolución se obtienen a partir de

$$f^k_{i,j} = \tfrac{1}{h_k^2} \int_{y^k_{j-1}}^{y^k_j} \int_{x^k_{j-1}}^{x^k_j} f(x,y) dx dy.$$

Podemos reducir la resolución de la información con:

$$\bar{f}^{k-1}_{i,j} = \frac{1}{4} \left(\bar{f}^k_{2i,2j} + \bar{f}^k_{2i-1,2j} + \bar{f}^k_{2i,2j-1} + \bar{f}^k_{2i-1,2j-1} \right), \quad 1 \leqslant i,j \leqslant N_{k-1}.$$

La consistencia para el operador predicción, $D^{k-1}_k P^k_{k-1} = I_{V^{k-1}}$, se formula como:

$$\frac{1}{4} \Big(\left(P^k_{k-1} \bar{f}^{k-1} \right)_{2i,2j} + \left(P^k_{k-1} \bar{f}^{k-1} \right)_{2i-1,2j} + \\ + \left(P^k_{k-1} \bar{f}^{k-1} \right)_{2i,2j-1} + \left(P^k_{k-1} \bar{f}^{k-1} \right)_{2i-1,2j-1} \Big) = \bar{f}^{k-1}_{i,j}.$$

Entonces, si definimos $e^k_{i,j} = \bar{f}^k_{i,j} - \left(P^k_{k-1} \bar{f}^{k-1} \right)_{i,j}$

$$\frac{1}{4} \left(e^k_{2i,2j} + e^k_{2i-1,2j} + e^k_{2i,2j-1} + e^k_{2i-1,2j-1} \right) = 0$$

y reducimos la información redundante definiendo

$$d^k_{i,j}(2) = \frac{1}{4} \left(e^k_{2i,2j} - e^k_{2i,2j-1} + e^k_{2i-1,2j} - e^k_{2i-1,2j-1} \right)$$

$$d^k_{i,j}(1) = \frac{1}{4} \left(e^k_{2i,2j} - e^k_{2i,2j-1} - e^k_{2i-1,2j} + e^k_{2i-1,2j-1} \right)$$

$$d^k_{i,j}(3) = \frac{1}{4} \left(e^k_{2i,2j} + e^k_{2i,2j-1} - e^k_{2i-1,2j} - e^k_{2i-1,2j-1} \right),$$

se puede observar que existe una correspondencia biunívoca entre los conjuntos

$$\left\{ \bar{f}^k_{i,j} \right\} \leftrightarrow \left\{ \bar{f}^{k-1}_{i,j}, d^k_{i,j}(1), d^k_{i,j}(2), d^k_{i,j}(3) \right\}$$

con $\bar{f}^{k-1} = D^{k-1}_k \bar{f}^k$.

La transformación multiescala en dos dimensiones y su inversa puede escribirse de la manera siguiente:

ALGORITMO 2.4.
Transformación directa 2D $\bar{f}^L \longrightarrow M\bar{f}^L = (\bar{f}^0, d^1, \ldots, d^L)$

for $k = L, ..., 1$
for $i, j = 1, ..., N_{k-1}$

$$\bar{f}^{k-1}_{i,j} = \tfrac{1}{4}\left(\bar{f}^k_{2i,2j} + \bar{f}^k_{2i-1,2j} + \bar{f}^k_{2i,2j-1} + \bar{f}^k_{2i-1,2j-1}\right)$$
$$e^k_{2i,2j} = \bar{f}^k_{2i,2j} - \left(P^k_{k-1}\bar{f}^{k-1}\right)_{2i,2j}$$
$$e^k_{2i-1,2j} = \bar{f}^k_{2i-1,2j} - \left(P^k_{k-1}\bar{f}^{k-1}\right)_{2i-1,2j}$$
$$e^k_{2i,2j-1} = \bar{f}^k_{2i,2j-1} - \left(P^k_{k-1}\bar{f}^{k-1}\right)_{2i,2j-1}$$
$$e^k_{2i-1,2j-1} = \bar{f}^k_{2i-1,2j-1} - \left(P^k_{k-1}\bar{f}^{k-1}\right)_{2i-1,2j-1}$$
$$d^k_{i,j}(2) = \tfrac{1}{4}\left(e^k_{2i,2j} - e^k_{2i,2j-1} + e^k_{2i-1,2j} - e^k_{2i-1,2j-1}\right)$$
$$d^k_{i,j}(1) = \tfrac{1}{4}\left(e^k_{2i,2j} - e^k_{2i,2j-1} - e^k_{2i-1,2j} + e^k_{2i-1,2j-1}\right)$$
$$d^k_{i,j}(3) = \tfrac{1}{4}\left(e^k_{2i,2j} + e^k_{2i,2j-1} - e^k_{2i-1,2j} - e^k_{2i-1,2j-1}\right)$$

end
end

ALGORITMO 2.5.
Transformación inversa 2D $(M\bar{f}^L \longrightarrow M^{-1}M\bar{f}^L)$

for $k = 1, ..., L$
for $i, j = 1, ..., N_{k-1}$

$$\bar{f}^k_{2i-1,2j-1} = \left(P^k_{k-1}\bar{f}^{k-1}\right)_{2i-1,2j-1} + d^k_{i,j}(1) - d^k_{i,j}(2) - d^k_{i,j}(3)$$
$$\bar{f}^k_{2i-1,2j} = \left(P^k_{k-1}\bar{f}^{k-1}\right)_{2i-1,2j} - d^k_{i,j}(1) + d^k_{i,j}(2) - d^k_{i,j}(3)$$
$$\bar{f}^k_{2i,2j-1} = \left(P^k_{k-1}\bar{f}^{k-1}\right)_{2i,2j-1} - d^k_{i,j}(1) - d^k_{i,j}(2) + d^k_{i,j}(3)$$
$$\bar{f}^k_{2i,2j} = \left(P^k_{k-1}\bar{f}^{k-1}\right)_{2i,2j} + d^k_{i,j}(1) + d^k_{i,j}(2) + d^k_{i,j}(3)$$

end
end

Ahora vamos a describir un tipo particular de operadores predicción lineales que se construyen usando polinomios interpoladores 2D estándares. Consideremos los polinomios 2D

$$p^s_{i,j}(x, y) = \sum_{0 \leqslant l,m \leqslant 2s} c_{l,m} x^l y^m.$$

Determinamos sus $(2s+1)^2$ coeficientes imponiendo

$$\int_{x_{i+l-1}^{k-1}}^{x_{i+l}^{k-1}} \int_{y_{j+m-1}^{k-1}}^{y_{j+m}^{k-1}} p_{i,j}^s(x,y)dydx = \bar{f}_{i+l,j+m}^{k-1}, \quad l,m = -s,\ldots,s.$$

Entonces definimos

$$\left(P_{k-1}^k \bar{f}^{k-1}\right)_{2i,2j-1} = \int_{x_{2i-1}^k}^{x_{2i}^k} \int_{y_{2j-2}^k}^{y_{2j-1}^k} p_{i,j}^s(x,y)dydx$$

$$\left(P_{k-1}^k \bar{f}^{k-1}\right)_{2i-1,2j-1} = \int_{x_{2i-2}^k}^{x_{2i-1}^k} \int_{y_{2j-2}^k}^{y_{2j-1}^k} p_{i,j}^s(x,y)dydx$$

$$\left(P_{k-1}^k \bar{f}^{k-1}\right)_{2i-1,2j} = \int_{x_{2i-2}^k}^{x_{2i-1}^k} \int_{y_{2j-1}^k}^{y_{2j}^k} p_{i,j}^s(x,y)dydx$$

$$\left(P_{k-1}^k \bar{f}^{k-1}\right)_{2i,2j} = \int_{x_{2i-1}^k}^{x_{2i}^k} \int_{y_{2j-1}^k}^{y_{2j}^k} p_{i,j}^s(x,y)dydx.$$

A partir de los resultados de interpolación 2D (ver[10]) obtenemos:

$$\left(P_{k-1}^k \bar{f}^{k-1}\right)_{2i-1,2j-1} = \bar{f}^{k-1} + Q_x^s\left(i,j;\bar{f}^{k-1}\right) + Q_y^s\left(i,j;\bar{f}^{k-1}\right) + Q_{xy}^s\left(i,j;\bar{f}^{k-1}\right)$$
$$\left(P_{k-1}^k \bar{f}^{k-1}\right)_{2i-1,2j} = \bar{f}^{k-1} + Q_x^s\left(i,j;\bar{f}^{k-1}\right) - Q_y^s\left(i,j;\bar{f}^{k-1}\right) - Q_{xy}^s\left(i,j;\bar{f}^{k-1}\right)$$
$$\left(P_{k-1}^k \bar{f}^{k-1}\right)_{2i,2j-1} = \bar{f}^{k-1} - Q_x^s\left(i,j;\bar{f}^{k-1}\right) + Q_y^s\left(i,j;\bar{f}^{k-1}\right) - Q_{xy}^s\left(i,j;\bar{f}^{k-1}\right)$$
$$\left(P_{k-1}^k \bar{f}^{k-1}\right)_{2i,2j} = \bar{f}^{k-1} - Q_x^s\left(i,j;\bar{f}^{k-1}\right) - Q_y^s\left(i,j;\bar{f}^{k-1}\right) + Q_{xy}^s\left(i,j;\bar{f}^{k-1}\right)$$

donde

$$Q_x\left(i,j;\bar{f}^{k-1}\right) = \sum_{l=1}^{s} \beta_l \left(\bar{f}_{i+l,j}^{k-1} - \bar{f}_{i-l,j}^{k-1}\right)$$

$$Q_y\left(i,j;\bar{f}^{k-1}\right) = \sum_{m=1}^{s} \beta_m \left(\bar{f}_{i,j+m}^{k-1} - \bar{f}_{i,j-m}^{k-1}\right)$$

$$Q_{xy}\left(i,j;\bar{f}^{k-1}\right) = \sum_{l=1}^{s} \beta_l \sum_{m=1}^{s} \beta_m \left(\bar{f}_{i+l,j+m}^{k-1} + \bar{f}_{i-l,j+m}^{k-1} + \bar{f}_{i+l,j-m}^{k-1} + \bar{f}_{i-l,j-m}^{k-1}\right)$$

y los coeficientes β son

$$\begin{cases} s = 1 \Rightarrow \beta_1 = -\frac{1}{8} \\ s = 2 \Rightarrow \beta_1 = -\frac{22}{128}, \beta_2 = -\frac{3}{128} \end{cases}$$

2.5.1

Algoritmo error control con medias en celda

Al igual que en el marco de los valores puntuales se propone también ahora una modificación de la transformación directa que incorpora el cuantizador no reversible $M_{\bar{\varepsilon}}^{M}$ y que nos permite seguir la pista del error acumulado. Las transformaciones multiescala 2D modificadas y sus inversas pueden ser escritas de las siguiente manera:

ALGORITMO 2.6.
Transformación directa 2D modificada (versión 1)

for $k = L, ..., 1$
  for $i, j = 1, ..., N_{k-1}$
    $\bar{f}_{i,j}^{k-1} = \frac{1}{4}\left(\bar{f}_{2i,2j}^{k} + \bar{f}_{2i-1,2j}^{k} + \bar{f}_{2i,2j-1}^{k} + \bar{f}_{2i-1,2j-1}^{k}\right)$
  end
end
$\hat{f}^0 = \text{proc}(\bar{f}^0, \varepsilon_0)$
for $k = L, ..., 1$
  for $i, j = 1, ..., N_{k-1}$
    $e_{2i,2j}^{k} = \bar{f}_{2i,2j}^{k} - \left(P_{k-1}^{k}\hat{f}^{k-1}\right)_{2i,2j}$
    $e_{2i-1,2j}^{k} = \bar{f}_{2i-1,2j}^{k} - \left(P_{k-1}^{k}\hat{f}^{k-1}\right)_{2i-1,2j}$
    $e_{2i,2j-1}^{k} = \bar{f}_{2i,2j-1}^{k} - \left(P_{k-1}^{k}\hat{f}^{k-1}\right)_{2i,2j-1}$
    $e_{2i-1,2j-1}^{k} = \bar{f}_{2i-1,2j-1}^{k} - \left(P_{k-1}^{k}\hat{f}^{k-1}\right)_{2i-1,2j-1}$
    $\tilde{d}_{i,j}^{k}(2) = \frac{1}{4}\left(e_{2i,2j}^{k} - e_{2i,2j-1}^{k} + e_{2i-1,2j}^{k} - e_{2i-1,2j-1}^{k}\right)$
    $\tilde{d}_{i,j}^{k}(1) = \frac{1}{4}\left(e_{2i,2j}^{k} - e_{2i,2j-1}^{k} - e_{2i-1,2j}^{k} + e_{2i-1,2j-1}^{k}\right)$
    $\tilde{d}_{i,j}^{k}(3) = \frac{1}{4}\left(e_{2i,2j}^{k} + e_{2i,2j-1}^{k} - e_{2i-1,2j}^{k} - e_{2i-1,2j-1}^{k}\right)$
  end

$\hat{d}^k(2) = \text{proc}(\tilde{d}^k(2), \varepsilon_k)$
$\hat{d}^k(1) = \text{proc}(\tilde{d}^k(1), \varepsilon_k)$
$\hat{d}^k(3) = \text{proc}(\tilde{d}^k(3), \varepsilon_k)$
for $i, j = 1, ..., N_{k-1}$
$\hat{f}^k_{2i-1,2j-1} = \left(P^k_{k-1}\hat{f}^{k-1}\right)_{2i-1,2j-1} + \hat{d}^k_{i,j}(1) - \hat{d}^k_{i,j}(2) - \hat{d}^k_{i,j}(3)$
$\hat{f}^k_{2i-1,2j} = \left(P^k_{k-1}\hat{f}^{k-1}\right)_{2i-1,2j} - \hat{d}^k_{i,j}(1) + \hat{d}^k_{i,j}(2) - \hat{d}^k_{i,j}(3)$
$\hat{f}^k_{2i,2j-1} = \left(P^k_{k-1}\hat{f}^{k-1}\right)_{2i,2j-1} - \hat{d}^k_{i,j}(1) - \hat{d}^k_{i,j}(2) + \hat{d}^k_{i,j}(3)$
$\hat{f}^k_{2i,2j} = \left(P^k_{k-1}\hat{f}^{k-1}\right)_{2i,2j} + \hat{d}^k_{i,j}(1) + \hat{d}^k_{i,j}(2) + \hat{d}^k_{i,j}(3)$
end
end

La transformación inversa de este algoritmo seria el Algoritmo 2.5

Otra posibilidad de modificar el algoritmos seria la siguiente:

ALGORITMO 2.7.
Transformación directa 2D modificada (versión 2)

for $k = L, ..., 1$
for $i, j = 1, ..., N_{k-1}$
$\bar{f}^{k-1}_{i,j} = \frac{1}{4}\left(\bar{f}^k_{2i,2j} + \bar{f}^k_{2i-1,2j} + \bar{f}^k_{2i,2j-1} + \bar{f}^k_{2i-1,2j-1}\right)$
end
end
$\hat{f}^0 = \text{proc}(\bar{f}^0, \varepsilon_0)$
for $k = L, ..., 1$
for $i, j = 1, ..., N_{k-1}$
$e^k_{2i,2j} = \bar{f}^k_{2i,2j} - \left(P^k_{k-1}\hat{f}^{k-1}\right)_{2i,2j}$
$e^k_{2i-1,2j} = \bar{f}^k_{2i-1,2j} - \left(P^k_{k-1}\hat{f}^{k-1}\right)_{2i-1,2j}$
$e^k_{2i,2j-1} = \bar{f}^k_{2i,2j-1} - \left(P^k_{k-1}\hat{f}^{k-1}\right)_{2i,2j-1}$
$e^k_{2i-1,2j-1} = \bar{f}^k_{2i-1,2j-1} - \left(P^k_{k-1}\hat{f}^{k-1}\right)_{2i-1,2j-1}$

$\tilde{d}^k_{i,j}(2) = \frac{1}{4}\left(e^k_{2i-1,2j-1} + e^k_{2i-1,2j} - e^k_{2i,2j-1} - e^k_{2i,2j}\right)$

$\tilde{d}^k_{i,j}(1) = \frac{1}{2}\left(e^k_{2i-1,2j-1} - e^k_{2i-1,2j}\right)$

$\tilde{d}^k_{i,j}(3) = \frac{1}{2}\left(e^k_{2i,2j-1} - e^k_{2i,2j}\right)$

end

$\hat{d}^k(2) = \text{proc}(\tilde{d}^k(2), \varepsilon_k)$

$\hat{d}^k(1) = \text{proc}(\tilde{d}^k(1), \varepsilon_k)$

$\hat{d}^k(3) = \text{proc}(\tilde{d}^k(3), \varepsilon_k)$

for $i, j = 1, ..., N_{k-1}$

$\hat{f}^k_{2i-1,2j-1} = \left(P^k_{k-1}\hat{f}^{k-1}\right)_{2i-1,2j-1} + \hat{d}^k_{i,j}(1) + \hat{d}^k_{i,j}(2)$

$\hat{f}^k_{2i-1,2j} = \left(P^k_{k-1}\hat{f}^{k-1}\right)_{2i-1,2j} - \hat{d}^k_{i,j}(1) + \hat{d}^k_{i,j}(2)$

$\hat{f}^k_{2i,2j-1} = \left(P^k_{k-1}\hat{f}^{k-1}\right)_{2i,2j-1} - \hat{d}^k_{i,j}(2) + \hat{d}^k_{i,j}(3)$

$\hat{f}^k_{2i,2j} = \left(P^k_{k-1}\hat{f}^{k-1}\right)_{2i,2j} - \hat{d}^k_{i,j}(2) - \hat{d}^k_{i,j}(3)$

end

end

ALGORITMO 2.8.
Transformación inversa 2D (versión 2)

for $k = 1, ..., L$

for $i, j = 1, ..., N_{k-1}$

$\hat{f}^k_{2i-1,2j-1} = \left(P^k_{k-1}\hat{f}^{k-1}\right)_{2i-1,2j-1} + \tilde{d}^k_{i,j}(1) + \tilde{d}^k_{i,j}(2)$

$\hat{f}^k_{2i-1,2j} = \left(P^k_{k-1}\hat{f}^{k-1}\right)_{2i-1,2j} - \tilde{d}^k_{i,j}(1) + \tilde{d}^k_{i,j}(2)$

$\hat{f}^k_{2i,2j-1} = \left(P^k_{k-1}\hat{f}^{k-1}\right)_{2i,2j-1} - \tilde{d}^k_{i,j}(2) + \tilde{d}^k_{i,j}(3)$

$\hat{f}^k_{2i,2j} = \left(P^k_{k-1}\hat{f}^{k-1}\right)_{2i,2j} - \tilde{d}^k_{i,j}(2) - \tilde{d}^k_{i,j}(3)$

end

end

Para tales algoritmos se pueden probar los siguientes resultados:

Proposición 2.7. *Dada una sucesión discreta $\bar{f}^L$, sea $\hat{f}^L = M^{-1} M_{\bar{\varepsilon}}^M \bar{f}^L$ con $M_{\bar{\varepsilon}}^M$ dada por el algoritmo 2.6 y M^{-1} dada por el algoritmo 2.5. Entonces:*

$$\left\| \bar{f}^L - \hat{f}^L \right\|_\infty \leqslant \left\| \bar{f}^0 - \hat{f}^0 \right\|_\infty + \sum_{l=1}^{3} \sum_{k=1}^{L} \left\| \tilde{d}^k(l) - \hat{d}^k(l) \right\|_\infty \tag{2.15}$$

$$\left\| \bar{f}^L - \hat{f}^L \right\|_1 \leqslant \left\| \bar{f}^0 - \hat{f}^0 \right\|_1 + \sum_{l=1}^{3} \sum_{k=1}^{L} \left\| \tilde{d}^k(l) - \hat{d}^k(l) \right\|_1 \tag{2.16}$$

$$\left\| \bar{f}^L - \hat{f}^L \right\|_2^2 = \left\| \bar{f}^0 - \hat{f}^0 \right\|_2^2 + \sum_{l=1}^{3} \sum_{k=1}^{L} \left\| \tilde{d}^k(l) - \hat{d}^k(l) \right\|_2^2 \tag{2.17}$$

Demostración. Como consecuencia de la relación de consistencia

$$\hat{f}_{i,j}^{k-1} = \frac{1}{4} \left(\left(P_{k-1}^k \hat{f}^{k-1} \right)_{2i-1,2j-1} + \left(P_{k-1}^k \hat{f}^{k-1} \right)_{2i-1,2j} + \right.$$
$$\left. + \left(P_{k-1}^k \hat{f}^{k-1} \right)_{2i,2j-1} + \left(P_{k-1}^k \hat{f}^{k-1} \right)_{2i,2j} \right)$$

y de la definición de los coeficientes de detalle $\tilde{d}_{i,j}^k(1)$, $\tilde{d}_{i,j}^k(2)$ y $\tilde{d}_{i,j}^k(3)$ tenemos:

$$\begin{aligned}
\bar{f}_{2i-1,2j}^k - \hat{f}_{2i-1,2j}^k &= \bar{f}_{i,j}^{k-1} - \hat{f}_{i,j}^{k-1} + \\
&+ \left(\tilde{d}_{i,j}^k(1) - \hat{d}_{i,j}^k(1) \right) + \left(\tilde{d}_{i,j}^k(2) - \hat{d}_{i,j}^k(2) \right) - \left(\tilde{d}_{i,j}^k(3) - \hat{d}_{i,j}^k(3) \right) \\
\bar{f}_{2i,2j-1}^k - \hat{f}_{2i,2j-1}^k &= \bar{f}_{i,j}^{k-1} - \hat{f}_{i,j}^{k-1} - \\
&- \left(\tilde{d}_{i,j}^k(1) - \hat{d}_{i,j}^k(1) \right) - \left(\tilde{d}_{i,j}^k(2) - \hat{d}_{i,j}^k(2) \right) + \left(\tilde{d}_{i,j}^k(3) - \hat{d}_{i,j}^k(3) \right) \\
\bar{f}_{2i-1,2j-1}^k - \hat{f}_{2i-1,2j-1}^k &= \bar{f}_{i,j}^{k-1} - \hat{f}_{i,j}^{k-1} + \\
&+ \left(\tilde{d}_{i,j}^k(1) - \hat{d}_{i,j}^k(1) \right) - \left(\tilde{d}_{i,j}^k(2) - \hat{d}_{i,j}^k(2) \right) - \left(\tilde{d}_{i,j}^k(3) - \hat{d}_{i,j}^k(3) \right) \\
\bar{f}_{2i,2j}^k - \hat{f}_{2i,2j}^k &= \bar{f}_{i,j}^{k-1} - \hat{f}_{i,j}^{k-1} + \\
&+ \left(\tilde{d}_{i,j}^k(1) - \hat{d}_{i,j}^k(1) \right) + \left(\tilde{d}_{i,j}^k(2) - \hat{d}_{i,j}^k(2) \right) + \left(\tilde{d}_{i,j}^k(3) - \hat{d}_{i,j}^k(3) \right)
\end{aligned} \tag{2.18}$$

Como $\max(|a|, |b|) = \frac{1}{2}(|a+b| + |a-b|)$ tenemos

$$\max(|a|, |b|, |c|, |d|) \leqslant \frac{1}{4}(|a+b+c+d| + |a+b-c-d| + |a-b+c-d| + |a-b-c+d|).$$

Entonces

$$\begin{aligned}\text{máx}\left(\left|\bar{f}^k_{2i-1,2j}-\hat{f}^k_{2i-1,2j}\right|,\left|\bar{f}^k_{2i,2j-1}-\hat{f}^k_{2i,2j-1}\right|,\right.\\ \left.,\left|\bar{f}^k_{2i-1,2j-1}-\hat{f}^k_{2i-1,2j-1}\right|,\left|\bar{f}^k_{2i,2j}-\hat{f}^k_{2i,2j}\right|\right)=\\ =\left|\bar{f}^{k-1}_{i,j}-\hat{f}^{k-1}_{i,j}\right|+\left|\tilde{d}^k_{i,j}(1)-\hat{d}^k_{i,j}(1)\right|+\\ +\left|\tilde{d}^k_{i,j}(2)-\hat{d}^k_{i,j}(2)\right|+\left|\tilde{d}^k_{i,j}(3)-\hat{d}^k_{i,j}(3)\right|,\end{aligned}$$

de este modo

$$\left\|\hat{f}^k-\bar{f}^k\right\|_\infty\leqslant\left\|\hat{f}^{k-1}-\bar{f}^{k-1}\right\|_\infty+\sum_{l=1,2,3}\left\|\tilde{d}^k(l)-\hat{d}^k(l)\right\|_\infty$$

y deducimos (2.15).

Además de (2.18), deducimos

$$\begin{aligned}&\left|\bar{f}^k_{2i-1,2j}-\hat{f}^k_{2i-1,2j}\right|+\left|\bar{f}^k_{2i,2j-1}-\hat{f}^k_{2i,2j-1}\right|+\\ &+\left|\bar{f}^k_{2i-1,2j-1}-\hat{f}^k_{2i-1,2j-1}\right|+\left|\bar{f}^k_{2i,2j}-\hat{f}^k_{2i,2j}\right|\leqslant\\ &\leqslant 4\left|\bar{f}^{k-1}_{i,j}-\hat{f}^{k-1}_{i,j}\right|+4\left|\tilde{d}^k_{i,j}(1)-\hat{d}^k_{i,j}(1)\right|+\\ &+4\left|\tilde{d}^k_{i,j}(2)-\hat{d}^k_{i,j}(2)\right|+4\left|\tilde{d}^k_{i,j}(3)-\hat{d}^k_{i,j}(3)\right|\end{aligned}$$

y también

$$\left\|\hat{f}^k-\bar{f}^k\right\|_1\leqslant\left\|\hat{f}^{k-1}-\bar{f}^{k-1}\right\|_1+\sum_{l=1,2,3}\left\|\tilde{d}^k(l)-\hat{d}^k(l)\right\|_1$$

así hemos probado (2.16).

Para probar (2.17), obtenemos de (2.18)

$$\begin{aligned}&\left(\bar{f}^k_{2i-1,2j}-\hat{f}^k_{2i-1,2j}\right)^2+\left(\bar{f}^k_{2i,2j-1}-\hat{f}^k_{2i,2j-1}\right)^2+\left(\bar{f}^k_{2i-1,2j-1}-\hat{f}^k_{2i-1,2j-1}\right)^2+\\ &+\left(\bar{f}^k_{2i,2j}-\hat{f}^k_{2i,2j}\right)^2=4\left(\bar{f}^{k-1}_{i,j}-\hat{f}^{k-1}_{i,j}\right)^2+4\left(\tilde{d}^k_{i,j}(1)-\hat{d}^k_{i,j}(1)\right)^2+\\ &+4\left(\tilde{d}^k_{i,j}(2)-\hat{d}^k_{i,j}(2)\right)^2+4\left(\tilde{d}^k_{i,j}(3)-\hat{d}^k_{i,j}(3)\right)^2\end{aligned}$$

así

$$\left\|\hat{f}^k - \bar{f}^k\right\|_2^2 = \left\|\hat{f}^{k-1} - \bar{f}^{k-1}\right\|_2^2 + \left\|\tilde{d}^k(1) - \hat{d}^k(1)\right\|_2^2 + \\ + \left\|\tilde{d}^k(2) - \hat{d}^k(2)\right\|_2^2 + \left\|\tilde{d}^k(3) - \hat{d}^k(3)\right\|_2^2$$

y la proposición queda probada. □

Proposición 2.8. *Dada una sucesión discreta $\bar{f}^L$, sea $\hat{f}^L = M^{-1} M_{\bar{\varepsilon}}^M \bar{f}^L$ con $M_{\bar{\varepsilon}}^M$ dada por el algoritmo 2.7 y M^{-1} dada por el algoritmo 2.8. Entonces:*

$$\left\|\bar{f}^L - \hat{f}^L\right\|_\infty \leqslant \left\|\bar{f}^0 - \hat{f}^0\right\|_\infty + \sum_{k=1}^{L} \left\|\tilde{d}^k(2) - \hat{d}^k(2)\right\|_\infty + \\ + \sum_{k=1}^{L} \max\left\{\left\|\tilde{d}^k(1) - \hat{d}^k(1)\right\|_\infty, \left\|\tilde{d}^k(3) - \hat{d}^k(3)\right\|_\infty\right\} \tag{2.19}$$

$$\left\|\bar{f}^L - \hat{f}^L\right\|_1 \leqslant \left\|\bar{f}^0 - \hat{f}^0\right\|_1 + \sum_{k=1}^{L} \left\|\tilde{d}^k(2) - \hat{d}^k(2)\right\|_1 + \\ + \frac{1}{2}\sum_{k=1}^{L} \left\|\tilde{d}^k(1) - \hat{d}^k(1)\right\|_1 + \frac{1}{2}\sum_{k=1}^{L} \left\|\tilde{d}^k(3) - \hat{d}^k(3)\right\|_1 \tag{2.20}$$

$$\left\|\bar{f}^L - \hat{f}^L\right\|_2^2 \leqslant \left\|\bar{f}^0 - \hat{f}^0\right\|_2^2 + \sum_{k=1}^{L} \left\|\tilde{d}^k(2) - \hat{d}^k(2)\right\|_2^2 + \\ + \frac{1}{2}\sum_{k=1}^{L} \left\|\tilde{d}^k(1) - \hat{d}^k(1)\right\|_2^2 + \frac{1}{2}\sum_{k=1}^{L} \left\|\tilde{d}^k(3) - \hat{d}^k(3)\right\|_2^2 \tag{2.21}$$

Demostración. Como consecuencia de la relación de consistencia

$$\hat{f}_{i,j}^{k-1} = \frac{1}{4}\left(\left(P_{k-1}^k \hat{f}^{k-1}\right)_{2i-1,2j-1} + \left(P_{k-1}^k \hat{f}^{k-1}\right)_{2i-1,2j} + \\ + \left(P_{k-1}^k \hat{f}^{k-1}\right)_{2i,2j-1} + \left(P_{k-1}^k \hat{f}^{k-1}\right)_{2i,2j}\right)$$

y de la definición de los coeficientes de detalle $\tilde{d}_{i,j}^k(1)$, $\tilde{d}_{i,j}^k(2)$ y $\tilde{d}_{i,j}^k(3)$

tenemos:

$$\begin{aligned}
\bar{f}^k_{2i-1,2j} - \hat{f}^k_{2i-1,2j} &= \bar{f}^{k-1}_{i,j} - \hat{f}^{k-1}_{i,j} - \left(\tilde{d}^k_{i,j}(1) - \hat{d}^k_{i,j}(1)\right) + \left(\tilde{d}^k_{i,j}(2) - \hat{d}^k_{i,j}(2)\right)\\
\bar{f}^k_{2i,2j-1} - \hat{f}^k_{2i,2j-1} &= \bar{f}^{k-1}_{i,j} - \hat{f}^{k-1}_{i,j} - \left(\tilde{d}^k_{i,j}(2) - \hat{d}^k_{i,j}(2)\right) + \left(\tilde{d}^k_{i,j}(3) - \hat{d}^k_{i,j}(3)\right)\\
\bar{f}^k_{2i-1,2j-1} - \hat{f}^k_{2i-1,2j-1} &= \bar{f}^{k-1}_{i,j} - \hat{f}^{k-1}_{i,j} + \left(\tilde{d}^k_{i,j}(1) - \hat{d}^k_{i,j}(1)\right) + \left(\tilde{d}^k_{i,j}(2) - \hat{d}^k_{i,j}(2)\right)\\
\bar{f}^k_{2i,2j} - \hat{f}^k_{2i,2j} &= \bar{f}^{k-1}_{i,j} - \hat{f}^{k-1}_{i,j} - \left(\tilde{d}^k_{i,j}(2) - \hat{d}^k_{i,j}(2)\right) - \left(\tilde{d}^k_{i,j}(3) - \hat{d}^k_{i,j}(3)\right)
\end{aligned} \tag{2.22}$$

Como $\max(|a|,|b|) = \frac{1}{2}(|a+b| + |a-b|)$ tenemos

$$\max(|a|,|b|,|c|,|d|) \leqslant \frac{1}{4}(|a+b+c+d|+|a+b-c-d|+|a-b+c-d|+|a-b-c+d|).$$

Entonces

$$\begin{aligned}
&\max\left(\left|\bar{f}^k_{2i-1,2j} - \hat{f}^k_{2i-1,2j}\right|, \left|\bar{f}^k_{2i,2j-1} - \hat{f}^k_{2i,2j-1}\right|, \left|\bar{f}^k_{2i-1,2j-1} - \hat{f}^k_{2i-1,2j-1}\right|,\right.\\
&\left.\left|\bar{f}^k_{2i,2j} - \hat{f}^k_{2i,2j}\right|\right) \leqslant \left|\bar{f}^{k-1}_{i,j} - \hat{f}^{k-1}_{i,j}\right| + \left|\tilde{d}^k_{i,j}(2) - \hat{d}^k_{i,j}(2)\right| +\\
&+ \max\left\{\left|\tilde{d}^k_{i,j}(1) - \hat{d}^k_{i,j}(1)\right|, \left|\tilde{d}^k_{i,j}(3) - \hat{d}^k_{i,j}(3)\right|\right\}
\end{aligned}$$

de este modo

$$\begin{aligned}
\left\|\hat{f}^k - \bar{f}^k\right\|_\infty \leqslant \left\|\hat{f}^{k-1} - \bar{f}^{k-1}\right\|_\infty + \left\|\tilde{d}^k(2) - \hat{d}^k(2)\right\|_\infty +\\
+ \max\left\{\left\|\tilde{d}^k(1) - \hat{d}^k(1)\right\|_\infty, \left\|\tilde{d}^k(3) - \hat{d}^k(3)\right\|_\infty\right\}
\end{aligned}$$

y deducimos (2.19).

Además de (2.22), deducimos

$$\begin{aligned}
&\left|\bar{f}^k_{2i-1,2j} - \hat{f}^k_{2i-1,2j}\right| + \left|\bar{f}^k_{2i,2j-1} - \hat{f}^k_{2i,2j-1}\right| + \left|\bar{f}^k_{2i-1,2j-1} - \hat{f}^k_{2i-1,2j-1}\right| +\\
&+ \left|\bar{f}^k_{2i,2j} - \hat{f}^k_{2i,2j}\right| \leqslant 4\left|\bar{f}^{k-1}_{i,j} - \hat{f}^{k-1}_{i,j}\right| + 2\left|\tilde{d}^k_{i,j}(1) - \hat{d}^k_{i,j}(1)\right| +\\
&+4\left|\tilde{d}^k_{i,j}(2) - \hat{d}^k_{i,j}(2)\right| + 2\left|\tilde{d}^k_{i,j}(3) - \hat{d}^k_{i,j}(3)\right|,
\end{aligned}$$

y

$$\begin{aligned}
\left\|\hat{f}^k - \bar{f}^k\right\|_1 \leqslant \left\|\hat{f}^{k-1} - \bar{f}^{k-1}\right\|_1 + \frac{1}{2}\left\|\tilde{d}^k(1) - \hat{d}^k(1)\right\|_1 +\\
+ \left\|\tilde{d}^k(2) - \hat{d}^k(2)\right\|_1 + \frac{1}{2}\left\|\tilde{d}^k(3) - \hat{d}^k(3)\right\|_1,
\end{aligned}$$

así hemos probado (2.20).

Para probar (2.21), obtenemos de (2.22)

$$\left(\bar{f}^k_{2i-1,2j} - \hat{f}^k_{2i-1,2j}\right)^2 + \left(\bar{f}^k_{2i,2j-1} - \hat{f}^k_{2i,2j-1}\right)^2 + \left(\bar{f}^k_{2i-1,2j-1} - \hat{f}^k_{2i-1,2j-1}\right)^2 + \\ + \left(\bar{f}^k_{2i,2j} - \hat{f}^k_{2i,2j}\right)^2 \leqslant 4\left(\bar{f}^{k-1}_{i,j} - \hat{f}^{k-1}_{i,j}\right)^2 + 2\left(\tilde{d}^k_{i,j}(1) - \hat{d}^k_{i,j}(1)\right)^2 + \\ +4\left(\tilde{d}^k_{i,j}(2) - \hat{d}^k_{i,j}(2)\right)^2 + 2\left(\tilde{d}^k_{i,j}(3) - \hat{d}^k_{i,j}(3)\right)^2$$

por tanto

$$\left\|\hat{f}^k - \bar{f}^k\right\|_2^2 = \left\|\hat{f}^{k-1} - \bar{f}^{k-1}\right\|_2^2 + \frac{1}{2}\left\|\tilde{d}^k(1) - \hat{d}^k(1)\right\|_2^2 + \\ + \left\|\tilde{d}^k(2) - \hat{d}^k(2)\right\|_2^2 + \frac{1}{2}\left\|\tilde{d}^k(3) - \hat{d}^k(3)\right\|_2^2$$

y la proposición queda demostrada. □

Corolario 2.9. *Considerar el esquema de multirresolución con error control descrito en la Proposición 2.7, y cualquier estrategia de procesamiento para reducir los coeficientes tal que*

$$\left\|\bar{f}^0 - \hat{f}^0\right\|_p \leqslant \varepsilon_0 \text{ y } \left\|\tilde{d}^k(l) - \hat{d}^k(l)\right\|_p \leqslant \varepsilon_k \quad \forall l = 1,2,3, \quad \forall k = 1,...,L \tag{2.23}$$

donde $p = 1$ o 2. Entonces, tenemos:

$$\left\|\bar{f}^L - \hat{f}^L\right\|_\infty \leqslant \varepsilon_0 + 3\sum_{k=1}^{L}\varepsilon_k$$

$$\left\|\bar{f}^L - \hat{f}^L\right\|_1 \leqslant \varepsilon_0 + 3\sum_{k=1}^{L}\varepsilon_k$$

$$\left\|\bar{f}^L - \hat{f}^L\right\|_2^2 \leqslant \varepsilon_0^2 + 3\sum_{k=1}^{L}\varepsilon_k^2$$

En particular, si consideramos

$$\varepsilon_0 = \frac{\varepsilon}{2^L}, \quad \varepsilon_k = \frac{\varepsilon}{2^{L-k+1}}, \quad k = 1,\ldots,L \tag{2.24}$$

obtenemos

$$\left\|\bar{f}^L - \hat{f}^L\right\|_\infty \leqslant \left(3 - \frac{1}{2^{L-1}}\right)\varepsilon \tag{2.25}$$

$$\left\|\bar{f}^L - \hat{f}^L\right\|_1 \leqslant \frac{\varepsilon}{2^L} + \left(3 - \frac{1}{2^{L-1}}\right)\varepsilon \tag{2.26}$$

$$\left\|\bar{f}^L - \hat{f}^L\right\|_2^2 \leqslant \varepsilon^2 \tag{2.27}$$

Corolario 2.10. *Considerar el esquema de multirresolución con error control descrito en la Proposición 2.8, y cualquier estrategia de procesamiento para reducir los coeficientes tal que*

$$\left\|\bar{f}^0 - \hat{f}^0\right\|_p \leqslant \varepsilon_0 \text{ y } \left\|\tilde{d}^k(l) - \hat{d}^k(l)\right\|_p \leqslant \varepsilon_k \quad \forall l = 1, 2, 3, \quad \forall k = 1, ..., L$$

donde $p = 1$ *o* 2*. Entonces, tenemos:*

$$\left\|\bar{f}^L - \hat{f}^L\right\|_\infty \leqslant \varepsilon_0 + 2\sum_{k=1}^{L}\varepsilon_k$$

$$\left\|\bar{f}^L - \hat{f}^L\right\|_1 \leqslant \varepsilon_0 + 2\sum_{k=1}^{L}\varepsilon_k$$

$$\left\|\bar{f}^L - \hat{f}^L\right\|_2^2 \leqslant \varepsilon_0^2 + 2\sum_{k=1}^{L}\varepsilon_k^2$$

En particular, si consideramos

$$\varepsilon_0 = \frac{\varepsilon}{2^L}, \varepsilon_k = \frac{\varepsilon}{2^{L-k+1}} \quad k = 1, \ldots, L$$

obtenemos

$$\left\|\bar{f}^L - \hat{f}^L\right\|_\infty \leqslant \left(2 - \frac{1}{2^L}\right)\varepsilon$$

$$\left\|\bar{f}^L - \hat{f}^L\right\|_1 \leqslant \left(2 - \frac{1}{2^L}\right)\varepsilon$$

$$\left\|\bar{f}^L - \hat{f}^L\right\|_2^2 \leqslant \left(2 + \frac{1}{4^L}\right)\frac{\varepsilon^2}{3}$$

2.5.2 Experimentos numéricos con medias en celda

En los experimentos numéricos aplicamos a la imagen de Lena 512×512 las siguientes estrategias de compresión:

1. La transformada $M_{\bar{\varepsilon}}^{M} \bar{f}^{L}$ del Algoritmo 2.6 a la que llamaremos **EC1** (Error Control Versión 1). Usamos cuantización (2.9) como estrategia de procesamiento. Observamos, que

$$\hat{d}_{i,j}^{k}(l) = \textbf{proc\_qu}(\tilde{d}_{i,j}^{k}(l); \varepsilon_k) = 2\varepsilon_k \cdot \left[\frac{\tilde{d}_{i,j}^{k}(l)}{2\varepsilon_k} \right] \Rightarrow |\hat{d}_{i,j}^{k}(l) - \tilde{d}_{i,j}^{k}(l)| \leq \varepsilon_k.$$

   Entonces (2.23) es cierto y podemos aplicar la Proposición 2.7 y el Corolario 2.9.

2. La transformada $M_{\bar{0}}^{M} \bar{f}^{L}$ del Algoritmo 2.7, con $\bar{0} = \{0, \ldots, 0\}$. Entonces, cuantizamos $M_{\bar{0}}^{M} \bar{f}^{L} = (\bar{f}^0, \tilde{d}^1, \tilde{d}^2, \ldots, \tilde{d}^L)$, esto es, $Q_{\bar{\varepsilon}} M_{\bar{0}}^{M} \bar{f}^{L} = (\hat{f}^0, \hat{d}^1, \hat{d}^2, \ldots, \hat{d}^L)$, donde $\hat{d}^k(l) = \textbf{proc\_qu}(\tilde{d}^k(l); \varepsilon_k)$ y $\hat{f}^0 = \textbf{proc\_qu}(\bar{f}^0; \varepsilon_0)$. Lo llamaremos **ST1** (Standard Thresholding Versión 1)

En todos los casos aplicamos los algoritmos con $L = 4$ y utilizamos el operador predicción descrito en la Sección 2.5 con $s = 2$.

Aplicamos a $M_{\bar{\varepsilon}}^{M} \bar{f}^{L}$ y a $Q_{\bar{\varepsilon}} M_{\bar{0}}^{M} \bar{f}^{L}$ el algoritmo PPMZ (lossless) [11], el cual está basado en la predicción por el *partial matching PPM*.

En la Tabla 2.8, mostramos el ratio de compresión *bit-per-pixel* (*bpp*) para varios valores de ε, donde $\bar{\varepsilon} = \{\varepsilon_0, \ldots, \varepsilon_L\}$ está definido como (2.24) junto con los errores $\| \cdot \|_\infty$ y $\| \cdot \|_2$ de la imagen reconstruida. En la tabla podemos ver:

- El error real (REAL), esto es la evaluación exacta de $||\bar{f}^L - \hat{f}^L||_p$
- La cota de error *a posteriori* (POST), esto es la evaluación de la parte derecha de (2.15), (2.16) y (2.17) (Proposición 2.7).

ε	estra-tegia	$\|\|\cdot\|\|_\infty$ error			$\|\|\cdot\|\|_2$ error			bpp
		PRIO	POST	REAL	PRIO	POST	REAL	
8	**EC1**	24	23	16.39	8	3.38	3.38	0.828
	ST1			15.78			3.40	0.822
	ECA1		9.34	56.85	8	5.63	5.63	0.357
	ECB1		118	55.39	8	5.97	5.97	0.294
16	**EC1**	48	45.99	29.67	16	4.93	4.93	0.430
	ST1			31.62			4.96	0.427
	ECA1		192.1	82.98	16	8.06	8.06	0.167
	ECB1		212.9	87.24	16	8.45	8.45	0.147
32	**EC1**	96	91.85	57.76	32	7.09	7.09	0.217
	ST1			56.50			7.14	0.216
	ECA1		296	115.7	32	11.49	11.49	0.078
	ECB1		333	117.9	32	12.19	12.19	0.064

Tabla 2.8: *Ratios de compresión (bpp) y errores después de aplicar los Algoritmos 2.6 y 2.5 a la imagen Lena con diferentes estrategias de procesamiento (***EC1***, ***ST1***, ***ECA1*** y ***ECB1****) y diferentes valores* ε

- La cota *a priori* (PRIO), esto es la cota obtenida en el corolario 2.9 (2.25), (2.26) y (2.27).

Notemos que cuando utilizamos las transformaciones modificadas tenemos control completo del error de compresión pero, al mismo tiempo, hemos perdido el ratio de compresión. Sin embargo, observamos en los experimentos numéricos que para el mismo ε, el ratio de compresión de la representación comprimida obtenida haciendo $Q_{\bar{\varepsilon}} M_0^M \bar{f}^L$ está muy cerca de la obtenida aplicando $M_{\bar{\varepsilon}}^M \bar{f}^L$.

En el caso de la estrategia **ST1** no tenemos la cota *a priori* y la *a posteriori*.

Observemos que las cotas *a priori* y *a posteriori* están muy cerca en la norma $\|\cdot\|_\infty$. Esto es debido a que si en algún caso $|\bar{d}_{i,j}^k(l) - \tilde{d}_{i,j}^k(l)| \approx \varepsilon_k$ entonces $\| \bar{d}^k(l) - \tilde{d}^k(l) \|_\infty \approx \varepsilon_k$.

Gracias a la igualdad (2.17) tenemos que, en la norma $\|\cdot\|_2$, la cota *a posteriori* es exactamente el error real.

Como era esperable, el error en la norma $||\cdot||_2$ está lejos de de las cotas *a priori* (ε para el **EC1** (Corolario 2.9)).

Proponemos dos nuevas estrategias de procesamiento que nos permiten obtener mejores ratios de compresión y que a la vez la norma $||\cdot||_2$ está todavía bajo la cota *a priori*.

Una posible estrategia de procesamiento es substituir el cálculo de $\hat{d}^k_{i,j}(l)$, $l = 1, 2, 3$ utilizando (2.9) por:

$\hat{d}^k(1)$=**proc_ECA(**$\tilde{d}^k(1), \varepsilon$**)**
$\delta = \varepsilon$
Do
for $i, j = 1, \ldots, N_{k-1}$
$\hat{d}^k_{i,j}(1) =$ **proc_qu**$(\tilde{d}^k_{i,j}(1), \delta)$,
end
$\rho^2 = \frac{1}{(N_{k-1})^2} \sum_{i,j} |\hat{d}^k_{i,j}(1) - \tilde{d}^k_{i,j}(1)|^2 = ||\hat{d}^k(1) - \tilde{d}^k(1)||_2^2$
$\delta = 2 * \delta$
while $\rho \leq \epsilon$
end

El mismo procedimiento se repetiría para calcular $\hat{d}^k(1)$ y $\hat{d}^k(3)$.

La estrategia anterior incrementa el parámetro utilizado para cuantizar en cada nivel pero asegura, al mismo tiempo, $||\hat{d}^k(l) - \tilde{d}^k(l)||_2 \leq \varepsilon$, y es adecuado cuando queremos controlar la norma ρ como

$$\rho = \frac{1}{(N_{k-1})^2} \sum_{i,j} |\hat{d}^k_{i,j}(l) - \tilde{d}^k_{i,j}(l)| \tag{2.28}$$

tenemos un control directo en la norma L^1.

Otra posible estrategia es:

$\hat{d}^k(1)$=**proc_ECB(**$\tilde{d}^k(1), \varepsilon$**)**
$\delta = \varepsilon$
for $i, j = 1, \ldots, N_{k-1}$
$\hat{d}^k_{i,j}(1) =$ **proc_qu**$(\tilde{d}^k_{i,j}(1), \varepsilon)$,
end

Do
for $i, j = 1, \ldots, N_{k-1}$
$\hat{d}^k_{i,j}(1) = \textbf{proc\_tr}(\tilde{d}^k_{i,j}(1), \delta)$,
end
$\rho^2 = \frac{1}{(N_{k-1})^2} \sum_{i,j} |\hat{d}^k_{i,j}(1) - \tilde{d}^k_{i,j}(1)|^2 = ||\hat{d}^k(1) - \tilde{d}^k(1)||_2^2$
$\delta = \delta + \varepsilon$
while $\rho \leq \varepsilon_k$
end

Notar que estas estrategias, a las que llamaremos **ECA** y **ECB**, no tienen sentido cuando consideramos el algoritmo de la transformada directa, ya que no hay cotas *a posteriori* como las obtenidas en la Proposición 2.7.

ε	estrategia	$\|\|\cdot\|\|_\infty$ error			$\|\|\cdot\|\|_2$ error			bpp
		PRIO	POST	REAL	PRIO	POST	REAL	
8	**EC2**	16	15.50	12.61	6.56	3.02	3.02	1.055
	ST2			14.32			3.04	1.046
	ECA2		75.39	54.65	6.56	5.35	5.35	0.438
	ECB2		90.5	57.3	6.56	5.68	5.68	0.346
16	**EC2**	32	30.99	25.65	13.12	4.53	4.53	0.538
	ST2			28.14			4.57	0.532
	ECA2		127.8	71.57	13.12	7.91	7.91	0.201
	ECB2		135	75.23	13.12	7.73	7.73	0.182
32	**EC2**	64	61.98	48.93	26.24	6.62	6.62	0.265
	ST2			53.66			6.67	0.264
	ECA2		231	97.8	26.24	10.38	10.38	0.099
	ECB2		253	105	26.24	10.99	10.99	0.084

Tabla 2.9: *Ratios de compresión (bpp) y errores obtenidos despues de aplicar los Algoritmos 2.7 y 2.8 a la imagen de Lena con diferentes estrategias de procesamiento (***EC2***,* **ST2***,* **ECA2** *y* **ECB2***) y diferentes valores* ε

En la Tabla 2.9 presentamos los resultados que obtenemos cuando repetimos todos los experimentos con la Versión 2 de el Error Control. Esto es, Algoritmos 2.7 y 2.8, Proposición 2.8 y Corolario 2.10.

Como conclusión podemos decir que hemos presentado dos algoritmos no separables 2D de error control, los cuales nos permiten tener un control del error total. Aunque la idea original de Harten era usarlos cuando se toman operadores de predicción no lineales, ver [5, 7], es importante notar también que puede utilizarse con operadores de predicción lineales.

Con estos algoritmos podemos garantizar *a priori* una cierta calidad de la imagen reconstruida. Además, después de aplicar los algoritmos conocemos de forma exacta el error L_2 entre las imágenes originales y las reconstruidas sin ser necesario decodificar la imagen.

2.6 Comparación de métodos de compresión

Cuando se evalúan varios algoritmos de codificación de imágenes, hay diversos factores que determinan la elección de un determinado algoritmo para alguna aplicación. Uno de los factores más importantes en la mayoría de los casos es la eficiencia en la compresión, la cual evaluaremos en esta sección. Sin embargo, hay otros factores, como la complejidad, que pueden ser tan determinantes como la eficiencia de compresión.

Nuestro objetivo es estudiar cuál es el algoritmo que consigue comprimir más la transformada que hemos obtenido utilizando los algoritmos de error control. Haremos experimentos para la compresión con y sin pérdida. Para ésta, se mide la tasa de compresión lograda por el algoritmo. Para la compresión con pérdida se medirá la compresión conseguida de una transformada con error control con valores puntuales para diferentes tolerancias.

La evaluación se ha realizado con seis imágenes de tamaño 512 $\times$ 512 píxeles con una profundidad de 8 bits por píxel: “lena”, “barbara”, “peppers”, “fingerprint”, “goldhill” y “sintética”, mostradas en la figura 2.12. Los algoritmos o formatos de imagen que se han comparado son:

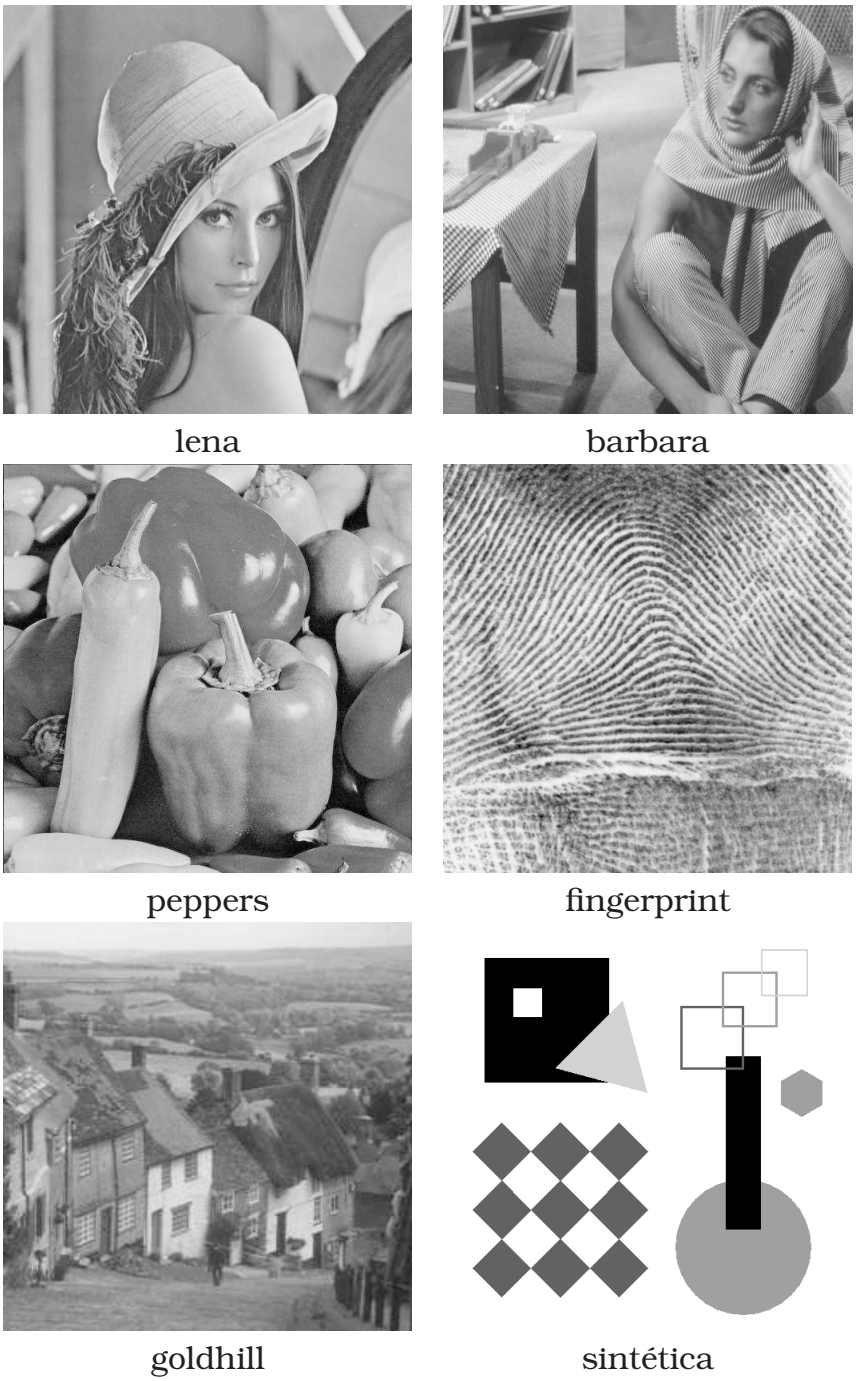


Figura 2.12: *Imágenes utilizadas para el estudio de la compresión.*

RAR es un algoritmo de compresión de datos sin pérdida desarrollado en 1993 por Eugene Roshal.

BZIP2 es un algoritmo de compresión sin pérdida, de código abierto, desarrollado en 1996 por Julian Seward.

PPMZ es un algoritmo de codificación de imágenes sin pérdida desarrollado en 1996 por Charles Bloom.

PNG es un formato de imágenes desarrollado en 1996, basado en un algoritmo de compresión sin pérdida.

JPEG-2000 es un formato de imagen basado en wavelets desarrollado en el año 2000. Incluye un modo *lossless* basado en el filtro wavelet 3/5.

JPEG-LS es un formato de imagen sin pérdida de información, creado en 1999. Incluye un modo near-lossless.

SPIHT es un algoritmo ideado por Amir Said y William A. Pearlman en 1996 basado en la codificación de árboles jerárquicos de la transformada wavelet.

Para el algoritmo SPIHT no se ha utilizado ningún tipo de codificación aritmética.

Estos algoritmos que se comparan son explicados con más detalle en la sección 1.4. Además los algoritmos EZW y SPIHT se describen más detalladamente en el anexo A.

2.6.1 Compresión sin pérdida

Vamos a comparar las tasas de compresión obtenidas por cada uno de los algoritmos (RAR, BZIP2, PPMZ, PNG, JPEG-2000, JPEG-LS, SPIHT).

En primer lugar, realizamos la descomposición multiescala de cada una de las imágenes de la figura 2.12 según el algoritmo 2.3 de multirresolución directa con control del error visto en la sección 2.4.1, utilizando 6 niveles de descomposición wavelet, $s = 1$ y la compresión sin pérdida conseguida utilizando el corolario 2.4.

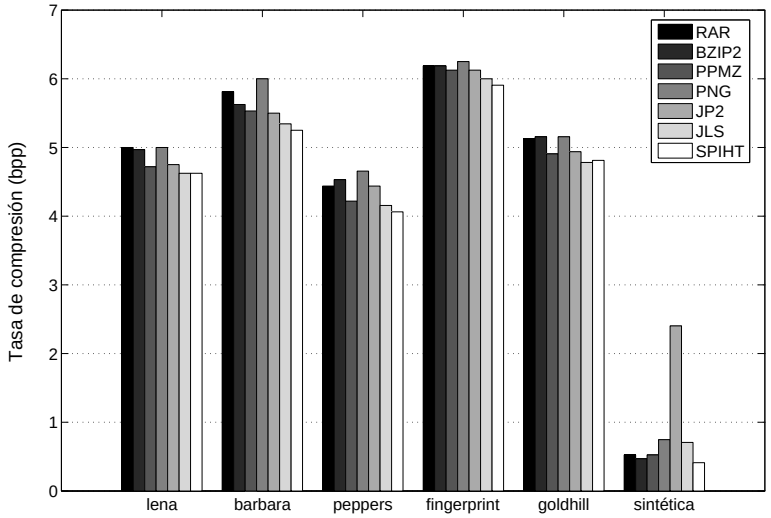


Figura 2.13: *Tasas de compresión sin pérdida obtenidas para cada una de las imágenes.*

En segundo lugar, guardamos con un formato de imagen (RAW) cada una de las matrices transformadas mediante el algoritmo 2.3 reescalando sus valores para que estén en el intervalo $[0, 255]$.

Por último, comprimimos cada una de las imágenes obtenidas con cada uno de los algoritmos de compresión y calculamos las tasas de compresión.

En la figura 2.13 se muestra la eficiencia de compresión de los diferentes métodos estudiados. Se puede observar que en la mayoría de los casos, los mejores resultados se obtienen con el algoritmo SPIHT. Con JPEG-LS (JLS) también se obtienen muy buenas tasas de compresión así como con el compresor PPMZ. Los resultados obtenidos con JPEG-2000 (JP2) son comparables a los del compresor PPMZ excepto con la imagen sintética donde JPEG-2000 consigue peores tasas de compresión con diferencia que el resto de métodos.

2.6.2 Compresión con error control

Vamos a comparar ahora las tasas de compresión obtenidas por los algoritmos RAR, BZIP2, PPMZ, PNG, JPEG-2000, JPEG-LS y SPIHT.

Primero, realizamos la descomposición multiescala de cada una de las imágenes de la figura 2.12 según el algoritmo 2.3 de multirresolución directa con control del error visto en la sección 2.4.1, utilizando 6 niveles de descomposición wavelet y $s = 1$. Aplicamos para ello, diferentes tolerancias.

A continuación, guardamos en formato RAW (como una imagen) cada una de las matrices transformadas mediante el algoritmo 2.3 reescalando sus valores para que estén en el intervalo $[0, 255]$.

Para finalizar, comprimimos cada una de las imágenes obtenidas con cada uno de los algoritmos de compresión y calculamos las tasas de compresión.

Las figuras 2.14, 2.15, 2.16 y 2.17 muestran las tasas de compresión conseguidas con los diferentes algoritmos utilizando las distintas tolerancias. Las figuras se separan en dos grupos: para las primeras utilizamos tolerancias que van de 1 a 10 y para las siguientes utilizamos tolerancias mayores (12, 16, 20, 24, 28, 32).

A la vista de las figuras se observa que SPIHT es el algoritmo que mejores tasas de compresión consigue, seguido de JPEG-LS y el compresor PPMZ. El formato de imagen JPEG-2000 consigue tasas similares a los compresores RAR y BZIP2 (siendo este último el mejor de los dos) y el formato de imagen PNG es el que peores resultados obtiene.

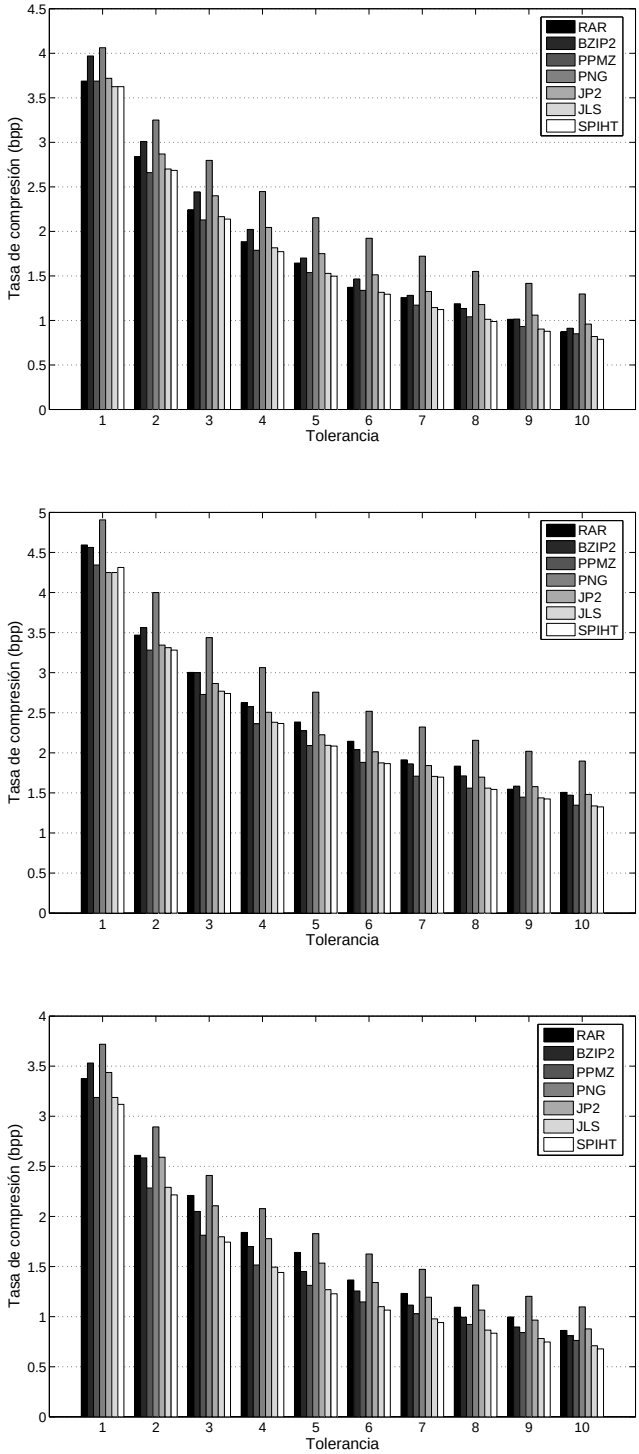


Figura 2.14: *De arriba a abajo: tasas de compresión de las imágenes lena, barbara y peppers transformadas con error control con diferentes valores de tolerancias (de 1 a 10).*

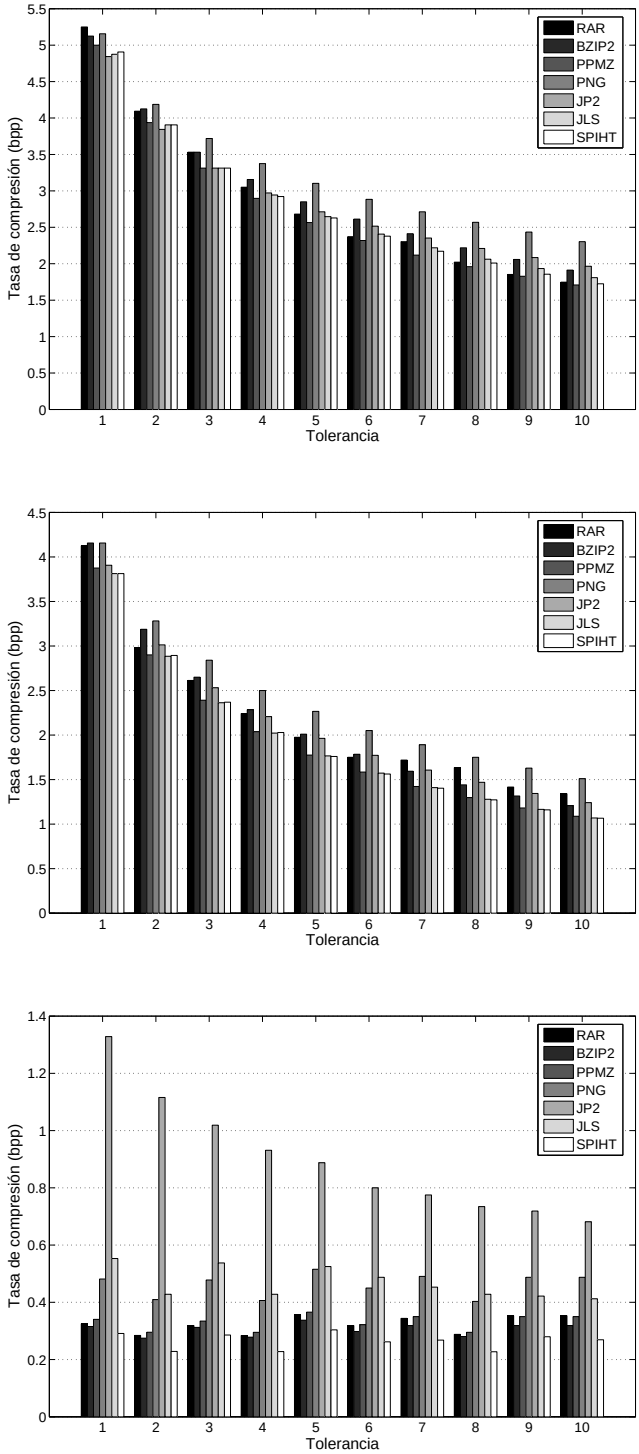


Figura 2.15: *De arriba a abajo: tasas de compresión de las imágenes fingerprint, goldhill y sintética transformadas con error control con diferentes valores de tolerancias (de 1 a 10).*

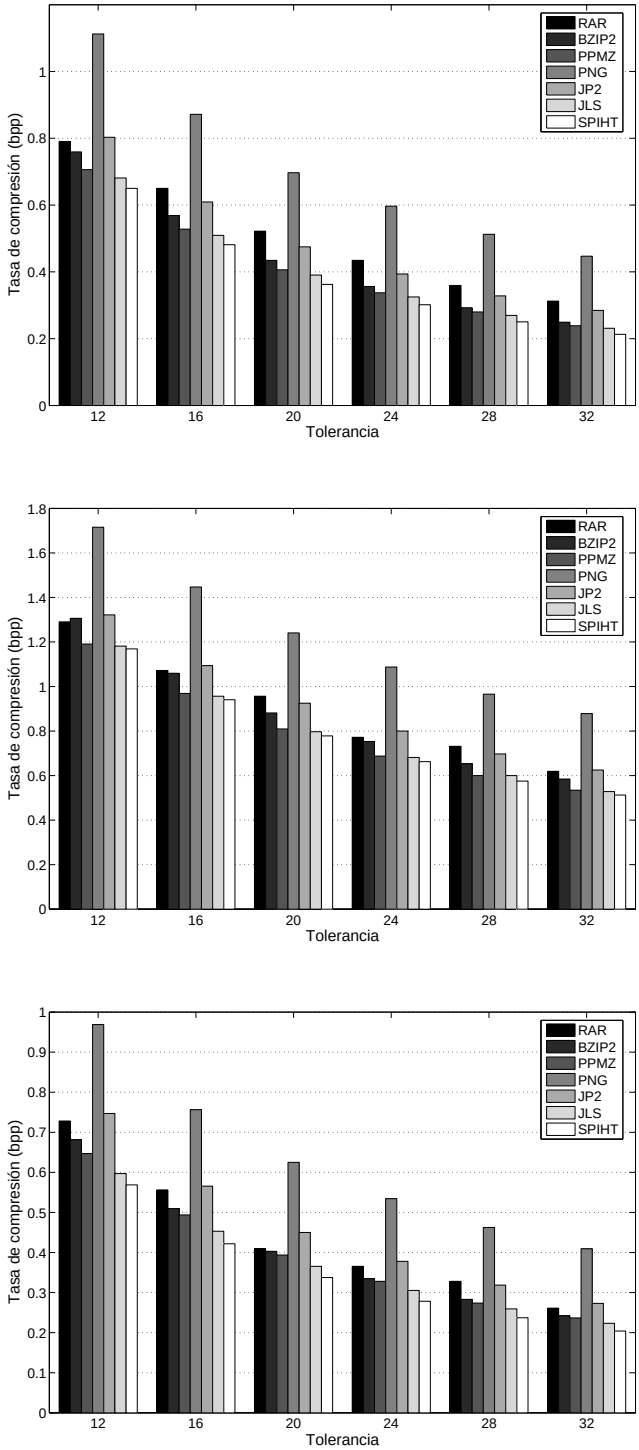


Figura 2.16: *De arriba a abajo: tasas de compresión de las imágenes lena, barbara y peppers transformadas con error control con diferentes valores de tolerancias (12, 16, 20, 24, 28, 32).*

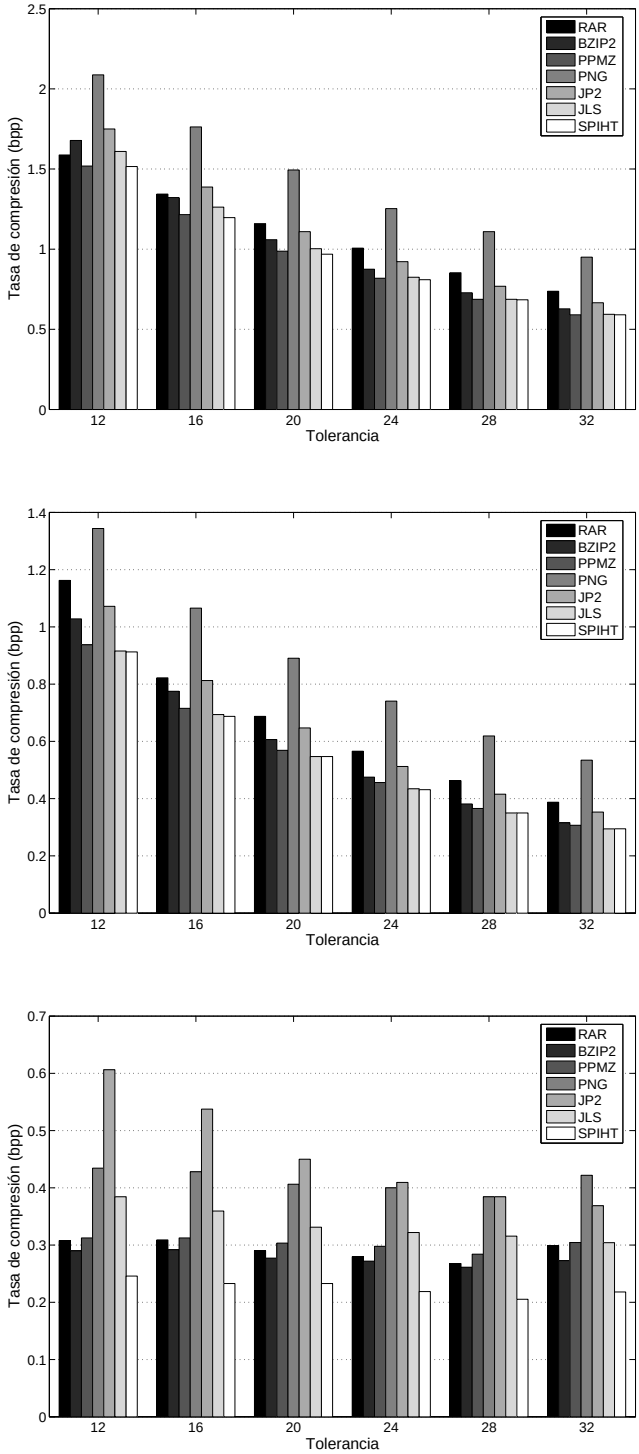


Figura 2.17: *De arriba a abajo: tasas de compresión de las imágenes fingerprint, goldhill y sintética transformadas con error control con diferentes valores de tolerancias (12, 16, 20, 24, 28, 32).*

2.7 Regiones de interés

En la compresión de imágenes muchas veces surge la necesidad de priorizar una zona de interés sobre el resto de la imagen. Algunos compresores como JPEG2000, EZW o SPIHT permiten darle un tratamiento diferencial a algunas zonas llamadas regiones de interés (en inglés, ROI, *Region of Interest*). Este tratamiento preferencial consiste en poder obtener más calidad o resolución de esa región en la imagen reconstruida, o en darle prioridad en la transmisión progresiva.

2.7.1 Método de escalado

Para codificar una ROI se utiliza el método de escalado que consiste en el desplazamiento de los coeficientes de tal manera que los bits asociados a la ROI sean colocados en planos de bits más altos que los bits asociados al fondo de la imagen como muestra la figura 2.18 (ver [2, 51]) . Entonces, durante el proceso de incrustación, los planos de bits más significativos de la ROI se sitúan en la secuencia de código antes que cualquier plano de bit del fondo de la imagen.

Dependiendo del valor de desplazamiento, algunos bits de los coeficientes de la ROI se pueden codificar junto con los coeficientes del fondo. Así, si la secuencia de código se trunca o el proceso de codificación termina antes de que la imagen sea completamente codificada, la ROI tendrá más calidad que el resto de la imagen. En [43] se incorpora una codificación de ROI en el SPIHT sin comprometer otras características deseables como el rendimiento en cuanto a tasa de compresión o el tiempo de computación.

Cuando el valor de desplazamiento es tal que los planos de bits codificados en primer lugar sólo contienen información de la ROI mientras que los siguientes sólo contienen la información del fondo de la imagen, el método de escalado recibe el nombre de método

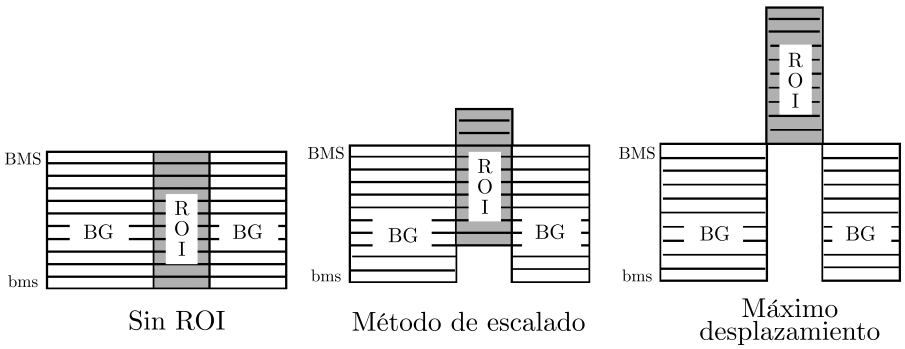


Figura 2.18: *Escalado de los coeficientes ROI. BG (background) es el fondo de la imagen, BMS son los bits más significativos y bms son los bits menos significativos*

de máximo desplazamiento (Maxshift).

El procedimiento que seguimos para comprimir una imagen $\bar{f}$ con una región de interés, ROI $\subset \bar{f}$, es el siguiente:

1. Hallamos la transformada de multirresolución de $\bar{f}$ a la que llamamos $\hat{f}$ utilizando para ello el algoritmo **2.3** con $s = 1$ y seis niveles de descomposición wavelet.

2. Calculamos la transformada $\hat{R}$ de la imagen $R = A \cdot \mathbf{1}_{\text{ROI}}$ (hacemos cero todos los píxeles de $\bar{f}$ que no pertenezcan a la ROI).

3. Convertimos los elementos distintos de cero de $\hat{R}$ en unos.

4. Codificamos con el algoritmo SPIHT, la matriz: $\hat{f} \cdot (\mathit{factor} \cdot \mathbf{1}_{\hat{R}} + \mathbf{1}_{1-\hat{R}})$.

5. Decodificamos y reconstruimos con el algoritmo **2.2**.

En las figuras **2.21** se puede apreciar el efecto visual producido al comprimir la imagen Lena 512×512 con una ROI centrada en el rostro, utilizando el método de escalado con distintos valores y con diferentes tasas de compresión. A la vista de las imágenes se puede apreciar que, en la mayoría de los casos, cuando se utiliza la ROI se obtienen visualmente mejores resultados sin sacrificar el PSNR.

2.7.2
Método de las tolerancias

Asimismo proponemos la utilización de las regiones de interés para comprimir una imagen con zonas con distinta calidad. Por ejemplo, podemos guardar la ROI sin pérdida y el fondo de la imagen (BG) almacenarlo con pérdida.

Primero calculamos la transformada de R con el algoritmo 2.3 y por tanto tenemos $(\hat{R}^0, \hat{d}_R^1, \ldots, \hat{d}_R^L)$. Luego calculamos

$$\bar{R}'^0_{i,j} = \begin{cases} 1 & \textbf{si} \quad \bar{R}^0_{i,j} \neq 0 \\ 0 & \textbf{si} \quad \bar{R}^0_{i,j} = 0 \end{cases}$$

y

$$\hat{d}'^k_{Ri,j} = \begin{cases} 1 & \textbf{si} \quad \hat{d}^k_{Ri,j} \neq 0 \\ 0 & \textbf{si} \quad \hat{d}^k_{Ri,j} = 0 \end{cases}$$

con $k = 0, ..., L$ y $i, j = 0, ..., N_k$, que utilizaremos en el siguiente algoritmo para codificar la imagen $\bar{f}$ con unas tolerancias ($\bar{\varepsilon}^{ROI}$) para la ROI y otras tolerancias ($\bar{\varepsilon}^{BG}$) para el resto de la imagen.

ALGORITMO 2.9. $\bar{f}^L \longrightarrow M^R_{\bar{\varepsilon}^{ROI},\bar{\varepsilon}^{BG}} \bar{f}^L = (\hat{f}^0, \hat{d}^1, \ldots, \hat{d}^L)$

for $k = L, ..., 1$
  for $i, j = 0, ..., N_k$
    $\bar{f}^{k-1}_{i,j} = \bar{f}^k_{2i,2j}$
  end
end
$\hat{f}^0 = \textbf{proc}(\bar{f}^0, \varepsilon_0^{ROI}) \cdot \mathbf{1}_{\bar{R}'^0} + \textbf{proc}(\bar{f}^0, \varepsilon_0^{BG}) \cdot \mathbf{1}_{1-\bar{R}'^0}$
for $k = 1, ..., L$
  for $i = 1, ..., N_{k-1}, j = 0, ..., N_{k-1}$
    $\tilde{d}^k_{i,j}(2) = \bar{f}^k_{2i-1,2j} - \left(P^k_{k-1}\hat{f}^{-1}\right)_{2i-1,2j}$
  end
  for $i = 0, ..., N_{k-1}, j = 1, ..., N_{k-1}$
    $\tilde{d}^k_{i,j}(3) = \bar{f}^k_{2i,2j-1} - \left(P^k_{k-1}\hat{f}^{k-1}\right)_{2i,2j-1}$

end
for $i = 1, ..., N_{k-1}, j = 1, ..., N_{k-1}$
$\tilde{d}^k_{i,j}(1) = \bar{f}^k_{2i-1,2j-1} - \left(P^k_{k-1}\hat{f}^{k-1}\right)_{2i-1,2j-1}$
end
$\hat{d}^k(2) = \mathbf{proc}(\tilde{d}^k(2), \varepsilon^{ROI}_k) \cdot \mathbf{1}_{\hat{d}'^k_{Ri,j}(2)} + \mathbf{proc}(\tilde{d}^k(2), \varepsilon^{BG}_k) \cdot \mathbf{1}_{1-\hat{d}'^k_{Ri,j}(2)}$
$\hat{d}^k(3) = \mathbf{proc}(\tilde{d}^k(3), \varepsilon^{ROI}_k) \cdot \mathbf{1}_{\hat{d}'^k_{Ri,j}(3)} + \mathbf{proc}(\tilde{d}^k(3), \varepsilon^{BG}_k) \cdot \mathbf{1}_{1-\hat{d}'^k_{Ri,j}(3)}$
$\hat{d}^k(1) = \mathbf{proc}(\tilde{d}^k(1), \varepsilon^{ROI}_k) \cdot \mathbf{1}_{\hat{d}'^k_{Ri,j}(1)} + \mathbf{proc}(\tilde{d}^k(1), \varepsilon^{BG}_k) \cdot \mathbf{1}_{1-\hat{d}'^k_{Ri,j}(1)}$
for $i = 1, ..., N_{k-1}, j = 0, ..., N_{k-1}$
$\hat{f}^k_{2i-1,2j} = \hat{d}^k_{i,j}(2) + \left(P^k_{k-1}\bar{f}^{k-1}\right)_{2i-1,2j}$
end
for $i = 0, ..., N_{k-1}, j = 1, ..., N_{k-1}$
$\hat{f}^k_{2i,2j-1} = \hat{d}^k_{i,j}(3) + \left(P^k_{k-1}\bar{f}^{k-1}\right)_{2i,2j-1}$
end
for $i = 1, ..., N_{k-1}, j = 1, ..., N_{k-1}$
$\hat{f}^k_{2i-1,2j-1} = \hat{d}^k_{i,j}(1) + \left(P^k_{k-1}\bar{f}^{k-1}\right)_{2i-1,2j-1}$
end
for $i, j = 0, ..., N_{k-1}$
$\hat{f}^k_{2i,2j} = \bar{f}^{k-1}_{i,j}$
end
end

La tabla 2.10 muestra los resultados obtenidos al utilizar el algoritmo 2.9 con la imagen Lena de tamaño 512×512 y 6 niveles de multirresolución. También se han utilizado diferentes valores de tolerancia para la zona ROI y para el resto de la imagen, BG. La región de interés utilizada se puede observar en la figura 2.7.2 y representa sólo el $2,13\,\%$ de los píxeles de la imagen completa.

En la figura 2.22 se puede apreciar el efecto visual producido al comprimir la imagen Lena 512×512 con la región de interés centrada en el rostro, utilizando un valor de tolerancia para la ROI y otro para el resto de la imagen. La columna de la izquierda muestra tres imágenes a las que se les ha aplicado una tolerancia menor

en la región de interés que en el fondo de la imagen mientras que la columna de la derecha muestra el resultado de codificar la imagen con la misma tolerancia para la ROI y el BG. A la vista de las imágenes y de los datos numéricos podemos decir que, cuando se utiliza una tolerancia menor para la ROI, obviamente la calidad global de la imagen aumenta un poco (dependiendo del tamaño de la ROI. En el ejemplo, el PSNR sube alrededor de 0,15 dB) por lo que la tasa de compresión disminuye a su vez, también en poca cantidad. Pero, por otro lado, el efecto visual producido que se obtiene es muy positivo pues se mantiene con mayor calidad las zonas más importantes de la imagen.

ε^{ROI}	ε^{BG}	**PSNR**	**MSE**	$\|\cdot\|_\infty$	$\|\cdot\|_\infty^{ROI}$	**bpp**
0,5	0,5	58,92	0,08	0,5	0,5	4,62
	4	41,19	4,94	4	0,5	1,99
	8	36,06	16,13	8	0,5	1,19
	16	30,93	52,52	16	0,5	0,87
	32	25,36	189	32	0,5	0,37
4	4	41,08	5,07	4	4	1,77
	8	36,02	16,26	8	4	1,05
	16	30,92	52,64	16	4	0,56
	32	25,36	189	32	4	0,29
8	8	35,96	16,62	8	8	0,99
	16	30,88	53,03	16	8	0,52
	32	25,36	189	32	8	0,25
16	16	30,77	54,51	16	16	0,48
	32	25,31	192	32	16	0,23
32	32	25,19	196	32	32	0,22

Tabla 2.10: *Resultados obtenidos al comprimir la imagen* 512×512 *Lena con una región de interés, utilizando Error control con* 6 *niveles de transformación y diferentes valores de tolerancia, uno para la ROI y otro para el resto de la imagen, BG.*

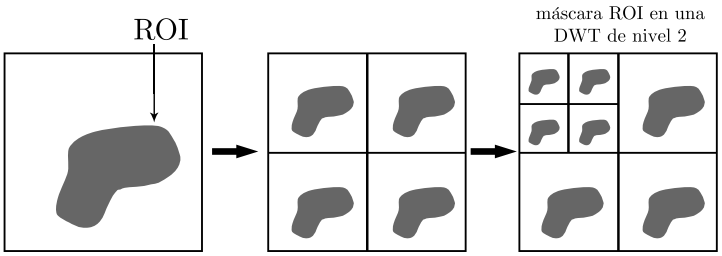
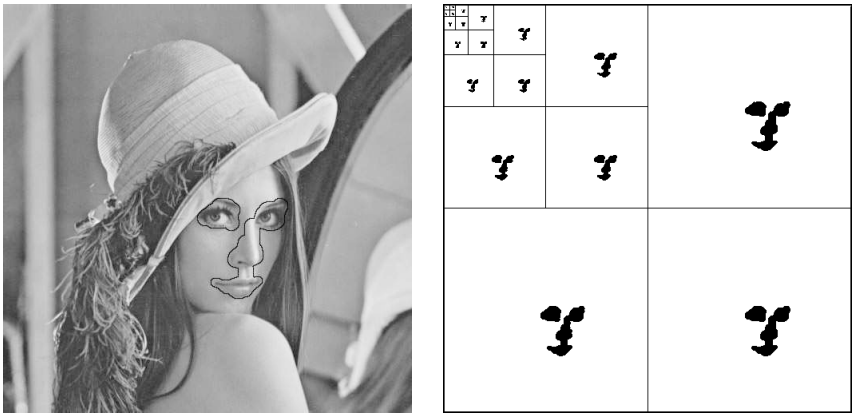


Figura 2.19: *Generación de una máscara ROI a partir de una ROI.*



(a) (b)

Figura 2.20: *(a) Silueta de la región de interés utilizada en las figuras 2.21 y 2.22 (b) máscara ROI producida.*

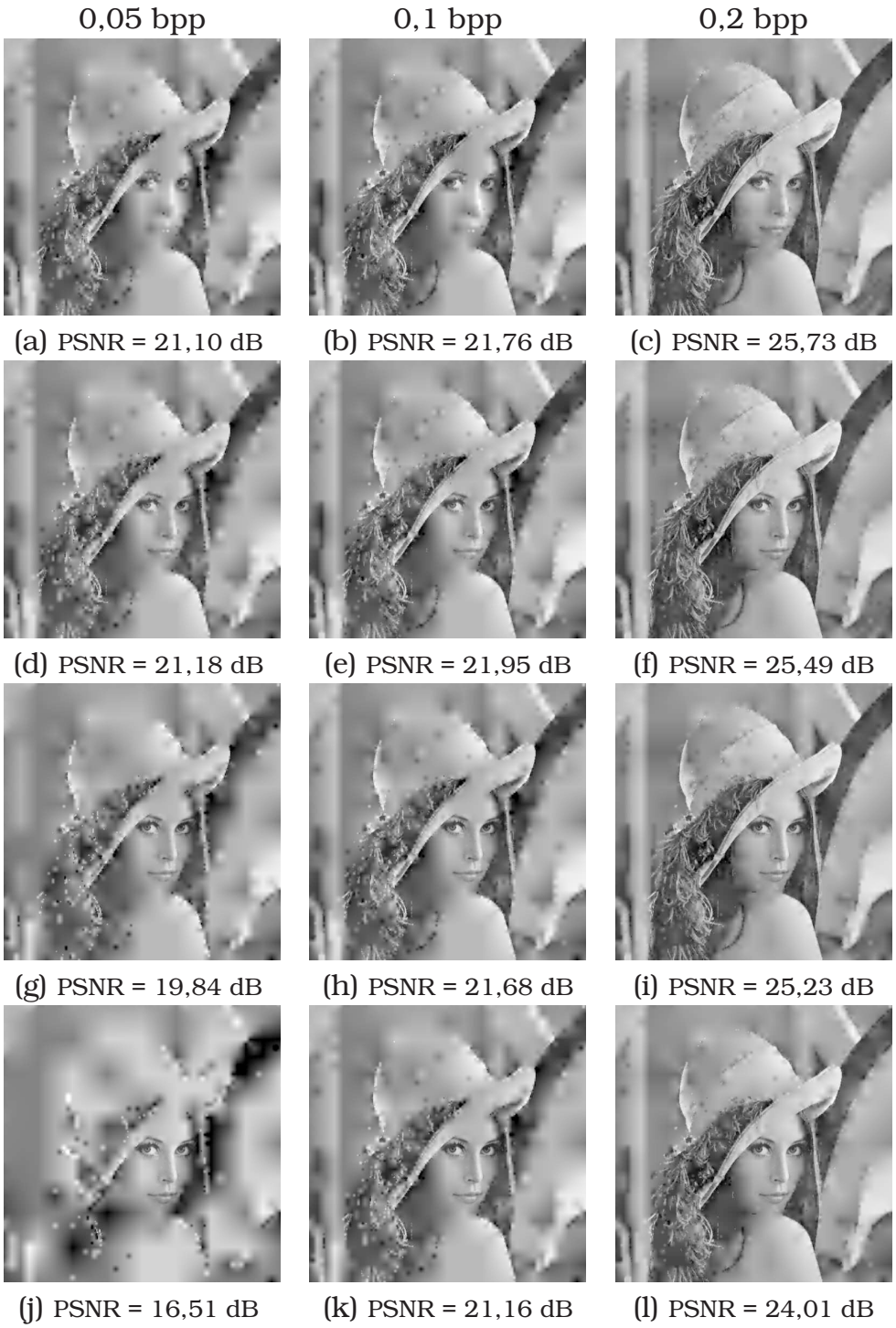


Figura 2.21: *Efecto visual producido al comprimir la imagen Lena con una región de interés centrada en el rostro. En la 1^a fila no se ha utilizado ninguna ROI, en la 2^a, 3^a y 4^a fila se han desplazado los coeficientes de la ROI uno, dos y tres planos de bits respectivamente. Las tasas de compresión son: 1^a columna 0,05 bpp, 2^a columna 0,1 bpp y 3^a columna 0,2 bpp.*

(a) $\varepsilon^{ROI} = 0,5$, $\varepsilon^{BG} = 8$ (b) $\varepsilon^{ROI} = 8$, $\varepsilon^{BG} = 8$

(c) $\varepsilon^{ROI} = 4$, $\varepsilon^{BG} = 16$ (d) $\varepsilon^{ROI} = 16$, $\varepsilon^{BG} = 16$

(e) $\varepsilon^{ROI} = 8$, $\varepsilon^{BG} = 32$ (f) $\varepsilon^{ROI} = 32$, $\varepsilon^{BG} = 32$

Figura 2.22: *Resultados de la codificación de la imagen Lena con una región de interés utilizando una tolerancia para la ROI y otra para el resto de la imagen, BG: (a) PSNR = 36,05 dB, 1,19 bpp, (b) PSNR = 35,92 dB, 0,99 bpp, (c) PSNR = 30,92 dB, 0,56 bpp, (d) PSNR = 30,77 dB, 0,48 bpp, (e) PSNR = 25,36 dB, 0,25 bpp, (f) PSNR = 25,19 dB, 0,22 bpp*

3

Interpolación WENO 2D para valores puntuales

3.1 Introducción

La información discreta para reconstruir una función normalmente viene como valores puntuales o medias en celda de la función en un conjunto finito de puntos o celdas, respectivamente. Un interpolador, esto es, una función simple cuyos valores o medias en celda en el conjunto de puntos o, respectivamente, celdas, coincide con los de la original, puede ser construido en cada punto una vez que se ha elegido un *stencil* a su alrededor. Si el *stencil* es constante, entonces la interpolación es un procedimiento lineal con respecto a los valores en el conjunto dado de puntos. En este

caso, el orden de la singularidad limita el orden de precisión de la aproximación, de manera que cualquier *stencil* que cruce la singularidad dará aproximaciones poco satisfactorias. Esto significa que habrá regiones mayores con poca precisión en la aproximación cerca de las singularidades cuando se incremente el grado del polinomio interpolador. La idea fundamental de la técnica ENO (Essentially Non-Oscillatory), introducida por Harten et al. [29] en el contexto de esquemas de captura de ondas de choque de alta resolución para leyes de conservación, es la elección de *stencils* que evitan cruzar singularidades, siempre que sea posible. Con el interpolador ENO la región afectada por cada singularidad se reduce al intervalo que la contiene, siempre que las singularidades estén suficientemente bien separadas.

En [36] Liu et al. introdujeron las reconstrucciones *weighted ENO* (WENO) como mejora de la técnica ENO. La idea consiste en asignar a cada subintervalo todos los *stencils* de una determinada longitud que lo contengan y construir el polinomio interpolador como una combinación convexa de los polinomios correspondientes. De esta manera usamos toda la información aportada por los nodos contenidos entre los *stencils* candidatos en el proceso de selección ENO, con la esperanza de conseguir mayor orden de precisión en puntos especificados en regiones donde la función es suave. La clave está en una asignación inteligente de los pesos de la combinación convexa. Dado que los pesos varían según la suavidad de la función en los *stencils*, la medida de la suavidad será crucial para la definición de los pesos. Jiang y Shu presentaron en [33] una medida de suavidad de la función interpolada más eficiente que la propuesta por Liu et al. en [36].

En [6] se propuso un interpolador WENO unidimensional y se aplicó de manera tensorial a multirresolución para aplicaciones de compresión de imágenes. En este capítulo adaptamos las técnicas de [36, 33, 6] para obtener una interpolación WENO totalmente bidimensional, la cual se aplicará en un contexto de interpolación de valores puntuales para su uso en representaciones de datos con multirresolución, con el objetivo de reducir el *efecto escalera* que se produce al interpolar tensorialmente cerca de perfiles oblicuos. En la sección 3.2 mencionamos los resultados básicos sobre interpolación tensorial lineal. En la sección 3.3 revisamos brevemente

las ideas básicas de la técnica WENO y proponemos interpoladores WENO totalmente bidimensionales, basados en indicadores de suavidad que son extensión de los introducidos en [6], los cuales, a su vez, están inspirados por los indicadores de suavidad de Jiang y Shu, pero a través de los correspondientes interpoladores de Lagrange, en vez de los interpoladores en el sentido de medias en celda. Específicamente, conseguimos los siguientes resultados para cualquier r, cuando usamos *stencils* de $2r \times 2r$ puntos, únicamente mediante el uso de propiedades de la interpolación de Lagrange: (1) el orden de la interpolación es $2r$ en regiones donde la función es suave, incluso cerca de extremos; (2) el orden de la interpolación es $r+1$, como el de los interpoladores ENO, cuando la función tiene una discontinuidad en el *stencil* de $2r \times 2r$ puntos, pero es suave en al menos uno de los *sub-stencils* de $(r+1) \times (r+1)$ puntos; (3) los pesos óptimos se obtienen en forma explícita.

3.2 Interpolación bidimensional

En esta sección mencionamos los resultados básicos sobre interpolación tensorial que serán necesarios como base de la interpolación WENO bidimensional que propondremos.

El problema de la interpolación en el conjunto $\{x_0, \ldots, x_r\} \times \{y_0, \ldots, y_r\}$ consiste en encontrar un polinomio

$$p(x,y) = \sum_{i,j=0}^{r} a_{i,j} x^i y^j,$$

tal que $p(x_i, y_j) = f(x_i, y_j)$ $i, j = 0, \ldots, r$.

Usamos la notación para los interpoladores unidimensionales:

$$P[x_0, \ldots, x_r; f](x)$$

y para los interpoladores bidimensionales:

$$P[\{x_0, \ldots, x_r\} \times \{y_0, \ldots, y_r\}; f](x,y).$$

Las diferencias divididas bidimensionales

$$f[x_0,\ldots,x_i][y_0,\ldots,y_j]$$

se definen recursivamente, dimensión por dimensión, de la manera habitual. Se puede probar que, si f es suficientemente diferenciable, entonces:

$$f[x_0,\ldots,x_i][y_0,\ldots,y_j]=\frac{1}{i!j!}f^{(i,j)}(\hat{x},\hat{y}) \tag{3.1}$$

donde a partir de ahora usaremos la notación:

$$f^{(m,n)}:=\frac{\partial^{m+n}f}{\partial x^m\partial y^n}.$$

Se puede deducir que:

$$\begin{aligned}P[\{x_0,\ldots,x_r\}\times\{y_0,\ldots,y_r\};f](x_*,y_*)=\\ P\Big[y_0,\ldots,y_r;P[x_0,\ldots,x_r;f(\cdot,y)](x_*)\Big](y_*)=\\ \sum_{i,j=0}^{k}f[x_0,\ldots,x_i][y_0,\ldots,y_j]\prod_{l=0}^{i-1}(x_*-x_l)\prod_{l=0}^{j-1}(y_*-y_l)\end{aligned}$$

(es interpolador simplemente a partir de la definición, tiene la expresión polinómica correcta y es único porque sus coeficientes son la solución de un sistema lineal cuya matriz es un producto de Kronecker de matrices de Vandermonde no singulares) y que el error de interpolación:

$$E(x_*,y_*)=f(x_*,y_*)-P[\{x_0,\ldots,x_r\}\times\{y_0,\ldots,y_r\};f](x_*,y_*)$$

viene dado por:

$$
\begin{gathered}
E(x_*, y_*) = f(x_*, y_*) - P[y_0, \ldots, y_k; P[x_0, \ldots, x_k; f(\cdot, y)](x_*)](y_*) = \\
f(x_*, y_*) - P[y_0, \ldots, y_k; f(x_*, y)](y_*) + \\
P[y_0, \ldots, y_k; f(x_*, y)](y_*) - P[y_0, \ldots, y_k; P[x_0, \ldots, x_k; f(\cdot, y)](x_*)](y_*) = \\
f[x_*][y_0, \ldots, y_r, y_*](y_* - y_0) \ldots (y_* - y_r) + \\
P[y_0, \ldots, y_k; f(x_*, y) - P[x_0, \ldots, x_k; f(\cdot, y)](x_*)](y_*) = \\
f[x_*][y_0, \ldots, y_r, y_*](y_* - y_0) \ldots (y_* - y_r) + \\
P[y_0, \ldots, y_k; f[x_0, \ldots, x_r, x_*][y](x_* - x_0) \ldots (x_* - x_r)](y_*) = \\
f[x_*][y_0, \ldots, y_r, y_*](y_* - y_0) \ldots (y_* - y_r) + (x_* - x_0) \ldots (x_* - x_r) \big(\\
f[x_0, \ldots, x_r, x_*][y_0] + f[x_0, \ldots, x_r, x_*][y_0, y_1](y_* - y_0) + \cdots + \\
f[x_0, \ldots, x_r, x_*][y_0, y_1, \ldots, y_r](y_* - y_0) \ldots (y_* - y_{r-1}) \big)
\end{gathered}
$$

o, simplemente suprimiendo $*$:

$$
\begin{aligned}
E(x, y) &= d(x, y)a(y) + b(x)c(x, y) \\
a(y) &= (y - y_0) \ldots (y - y_r) \\
b(x) &= (x - x_0) \ldots (x - x_r) \\
c(x, y) &= \sum_{l=0}^{k} f[x_0, \ldots, x_r, x][y_0, y_1, \ldots, y_l](y - y_0) \ldots (y - y_{l-1}) \\
d(x, y) &= f[x][y_0, \ldots, y_r, y]
\end{aligned}
$$

Para la definición de los indicadores de suavidad necesitamos estimar $E^{(m,n)}$, para lo cual tenemos en cuenta que

$$
f[x_0, \ldots, x_k, x]^{(n)} = n! f[x_0, \ldots, x_k, \overbrace{x, \ldots, x}^{n+1}]
$$

para calcular:

$$c^{(l,0)}(x,y) = \sum_{j=0}^{r} l! f[x_0,\dots,x_r,\overbrace{x,\dots,x}^{l+1}][y_0,\dots,y_j]\prod_{i=0}^{j-1}(y-y_i)$$

$$d^{(l,0)}(x,y) = l! f[\overbrace{x,\dots,x}^{l+1}][y_0,\dots,y_r,y]$$

$$c^{(l,s)}(x,y) = \sum_{j=0}^{r} l! f[x_0,\dots,x_r,\overbrace{x,\dots,x}^{l+1}][y_0,\dots,y_j]\frac{\partial^s \prod_{i=0}^{j-1}(y-y_i)}{\partial y^s}$$

$$d^{(l,s)}(x,y) = l!s! f[\overbrace{x,\dots,x}^{l+1}][y_0,\dots,y_r,\overbrace{y,\dots,y}^{s+1}].$$

Entonces, por la fórmula de Leibniz para la derivadas de cualquier orden de un producto, tenemos:

$$E^{(m,0)}(x,y) = d^{(m,0)}(x,y)a(y) + \sum_{q=0}^{m}\binom{m}{q} b^{(q)}(x)c^{(m-q,0)}(x,y)$$

$$E^{(m,n)}(x,y) = \sum_{q=0}^{n}\binom{n}{q} d^{(m,n-q)}(x,y)a^{(q)}(y) + \sum_{q=0}^{m}\binom{m}{q} b^{(q)}(x)c^{(m-q,n)}(x,y)$$

Para $(x_i,y_j) = (x,y) + \mathcal{O}(h)$, $a^{(q)}(y) = \mathcal{O}(h^{r+1-q})$, $b^{(q)}(x) = \mathcal{O}(h^{r+1-q})$, entonces, si f es suficientemente derivable:

$$E^{(m,n)}(x,y) = \mathcal{O}(h^{r+1-\max(m,n)}).$$

3.3 Interpolación WENO bidimensional

El objetivo de esta sección es definir un interpolador WENO bidimensional para su uso en un contexto de multirresolución para calcular

$$\mathcal{I}(x^{l-1}_{i,j-\frac{1}{2}}; f^{l-1}), \quad \mathcal{I}(x^{l-1}_{i-\frac{1}{2},j}; f^{l-1}), \quad \mathcal{I}(x^{l-1}_{i-\frac{1}{2},j-\frac{1}{2}}; f^{l-1}).$$

Podemos suponer un nivel de resolución fijo y un desplazamiento de los datos, de manera que podemos suprimir el superíndice $l-1$ y

suponer $i = j = 0$, de manera que sólo necesitamos dar los valores del interpolador en

$$(x, y) \in \{(-\frac{h}{2}, 0), (0, -\frac{h}{2}), (-\frac{h}{2}, -\frac{h}{2})\}.$$

La técnica interpolatoria WENO apareció como una mejora de las reconstrucciones ENO para obtener valores puntuales a partir de medias en celda y fue propuesta inicialmente para obtener esquemas de captura de ondas de choque de alta resolución para leyes de conservación. Cuando se usan *stencils* de $r+1$ nodos, las reconstrucciones ENO proporcionan un orden de precisión de $r+1$, excepto en aquellos subintervalos que contienen singularidades. Esto se consigue siguiendo un proceso de selección del stencil, en el que se elige aquel stencil en el que la función interpolada es más suave en algún sentido. Este proceso de selección es sensible a errores de redondeo, lo cual puede implicar que el *stencil* elegido no sea el más conveniente.

Por otra parte, en vez de considerar sólo un *stencil* que contiene $r+1$ nodos, podríamos tomar la información proporcionada por los $2r$ nodos del *stencil* que toma parte en el proceso de selección e intentar conseguir un orden de precisión $2r$ en las regiones donde la función sea suave. A continuación describimos la extensión bidimensional de esta idea.

Dado p, $0 \leq p < r$, denotamos la base de Lagrange unidimensional para la interpolación en el *stencil* $\{p-r, \ldots, p\}$ como $L^r_{p,m}$:

$$L^r_{p,m}(x_i) = \delta_{i,m}, p - r \leq i, m \leq p, \tag{3.2}$$

entonces el interpolador bidimensional en el *stencil* $\mathcal{S}_{p,q} = \{p - r, \ldots, p\} \times \{q - r, \ldots, q\}$ se denota por $P^r_{p,q}$ y viene dado por:

$$P^r_{p,q}(x, y) = \sum_{m=p-r}^{p} \sum_{n=q-r}^{q} f_{m,n} L^r_{p,m}(x) L^r_{q,n}(y). \tag{3.3}$$

Siguiendo las ideas de [36, 33], nuestro objetivo es encontrar *pesos óptimos* bidimensionales $C^r_{p,q}$ tal que

$$\sum_{p,q=0}^{r-1} C^r_{p,q} P^r_{p,q} = P^{2r-1}_{r-1,r-1},$$

y a continuación definir *pesos no lineales* $\omega_{p,q}^r$, que verifican

$$\omega_{p,q}^r = C_{p,q}^r + \mathcal{O}(h^{r-1}), \tag{3.4}$$

de manera que se puede definir el interpolador WENO:

$$Q(x,y) = \sum_{p,q=0}^{r-1} w_{p,q}^r P_{p,q}^r(x,y)$$

que satisface $f(x,y) - Q(x,y) = \mathcal{O}(h^{2r})$.

Proposición 3.1. *Los pesos óptimos bidimensionales vienen dados por* $C_{p,q}^r = C_p^r C_q^r$, *donde* $C_p^r = \frac{1}{2^{2r-1}}\binom{2r}{2p+1}$, *y verifican:*

$$\sum_{p,q=0}^{r-1} C_{p,q}^r P_{p,q}^r(x,y) = P_{r-1,r-1}^{2r-1}(x,y),$$

para $(x,y) \in \{(-\frac{h}{2},0),(0,-\frac{h}{2}),(-\frac{h}{2},-\frac{h}{2})\}$.

Demostración. En el contexto de la interpolación WENO unidimensional [6] se prueba que los pesos óptimos vienen dados por:

$$C_p^r = \frac{1}{2^{2r-1}}\binom{2r}{2p+1}, \quad p = 0,\dots,r-1.$$

y son aquellos que verifican:

$$\sum_{p=\max(m,0)}^{\min(m+r,r-1)} C_p^r L_{p,m}^r(-\frac{h}{2}) = L_{r-1,m}^{2r-1}(-\frac{h}{2}), \quad m = -r,\dots,r-1.$$

Teniendo en cuenta que

$$P_{p,q}^r(x,y) = \sum_{m=p-r}^{p}\sum_{n=q-r}^{q} f_{m,n} L_{p,m}^r(x) L_{q,n}^r(y)$$

$$P_{r-1,r-1}^{2r-1}(x,y) = \sum_{m=-r}^{r-1}\sum_{n=-r}^{r-1} f_{m,n} L_{r,m}^{2r-1}(x) L_{r,n}^{2r-1}(y)$$

entonces

$$\begin{aligned}\sum_{p,q=0}^{r-1} C_{p,q}^r P_{p,q}^r(-\frac{h}{2},-\frac{h}{2}) &= \sum_{p,q=0}^{r-1} C_p^r C_q^r \sum_{m=p-r}^{p}\sum_{n=q-r}^{q} f_{m,n} L_{p,m}^r(-\frac{h}{2}) L_{q,n}^r(-\frac{h}{2}) \\ &= \sum_{n,m=-r}^{r-1} f_{m,n} \sum_{p=\text{máx}(m,0)}^{\text{mín}(m+r,r-1)} C_p^r L_{p,m}^r(-\frac{h}{2}) \sum_{q=\text{máx}(n,0)}^{\text{mín}(n+r,r-1)} C_q^r L_{q,n}^r(-\frac{h}{2}) \\ &= \sum_{n,m=-r}^{r-1} f_{m,n} L_{r-1,m}^{2r-1}(-\frac{h}{2}) L_{r-1,n}^{2r-1}(-\frac{h}{2}) = P_{r-1,r-1}^{2r-1}(-\frac{h}{2},-\frac{h}{2})\end{aligned}$$

$$\begin{aligned}\sum_{p,q=0}^{r-1} C_{p,q}^r P_{p,q}^r(-\frac{h}{2},0) &= \sum_{p,q=0}^{r-1} C_p^r C_q^r \sum_{m=p-r}^{p}\sum_{n=q-r}^{q} f_{m,n} L_{p,m}^r(-\frac{h}{2}) L_{q,n}^r(0) \\ &= \sum_{p,q=0}^{r-1} C_p^r C_q^r \sum_{m=p-r}^{p}\sum_{n=q-r}^{q} f_{m,n} L_{p,m}^r(-\frac{h}{2}) \delta_{n,0} \\ &= \sum_{p,q=0}^{r-1} C_p^r C_q^r \sum_{m=p-r}^{p} f_{m,0} L_{p,m}^r(-\frac{h}{2}) \\ &= \sum_{q=0}^{r} C_q^r \sum_{m=-r}^{r-1} f_{m,0} \sum_{p=\text{máx}(m,0)}^{\text{mín}(m+r,r-1)} C_p^r L_{p,m}^r(-\frac{h}{2}) \\ &= \sum_{m=-r}^{r-1} f_{m,0} L_{r-1,m}^{2r-1}(-\frac{h}{2}) \\ &= \sum_{m,n=-r}^{r-1} f_{m,n} L_{r-1,m}^{2r-1}(-\frac{h}{2}) L_{r-1,n}^{2r-1}(0) = P_{r-1,r-1}^{2r-1}(-\frac{h}{2},0)\end{aligned}$$

De la misma forma:

$$\sum_{p,q=0}^{r-1} C_{p,q}^r P_{p,q}^r(0,-\frac{h}{2}) = P_{r-1,r-1}^{2r-1}(0,-\frac{h}{2}).$$

□

3.3.1 Indicadores de suavidad

Podemos definir indicadores de suavidad similares a los de [6] e inspirados en los que se introdujeron en [33] basados en el funcional

$$I_h(f) = \sum_{(m,n)\in\mathcal{I}} h^{2(m+n-1)} \int_{-h}^{0}\int_{-h}^{0} f^{(m,n)}(x,y)^2 dxdy, \tag{3.5}$$

donde $\mathcal{I} = \{0,\dots,r\}^2 \setminus \{(0,0)\}$. Si f es suave en $[-h,0]^2$, entonces $\int_{[-h,0]^2} f^{(m,n)2} = \mathcal{O}(h^2)$ y

$$I_h(f) = \mathcal{O}(h^2). \tag{3.6}$$

Si p es un interpolador como en la sección 3.2 entonces

$$I_h(f) - I_h(p) = \sum_{(m,n)\in\mathcal{I}} h^{2(m+n-1)} \int_{-h}^{0}\int_{-h}^{0} \left(f^{(m,n)}(x,y)^2 - p^{(m,n)}(x,y)^2\right) dxdy =$$

$$\sum_{(m,n)\in\mathcal{I}} h^{2(m+n-1)} \int_{-h}^{0}\int_{-h}^{0} E^{(m,n)}(x,y)(2f^{(m,n)}(x,y) - E^{(m,n)}(x,y))dxdy =$$

$$\sum_{(m,n)\in\mathcal{I}} h^{2(m+n-1)} h^2 \mathcal{O}(h^{r+1-\text{máx}(m,n)}) = \sum_{(m,n)\in\mathcal{I}} \mathcal{O}(h^{r+1+m+n}) = \mathcal{O}(h^{r+2}).$$

Por tanto

$$I_h(P^r_{p,q}) - I_h(P^r_{i,j}) = I_h(P^r_{p,q}) - I_h(f) + I_h(f) - I_h(P^r_{i,j}) = \mathcal{O}(h^{r+2}). \tag{3.7}$$

Por otra parte, si el *stencil* $\mathcal{S}_{p,q}$ contiene una discontinuidad, entonces

$$I_h(P^r_{p,q}) \not\to 0. \tag{3.8}$$

Teorema 3.2. *El interpolador WENO bidimensional*

$$Q(x,y) = \sum_{p,q=0}^{r-1} w^r_{p,q} P^r_{p,q}(x,y)$$

con

$$\omega^r_{p,q} = \frac{\alpha^r_{p,q}}{\sum_{i,j=0}^{r-1} \alpha^r_{i,j}}$$

$$\alpha^r_{i,j} = \frac{C^r_{i,j}}{(Kh^2 + I_h(P^r_{i,j}))^d}, \quad d \geq \frac{r+1}{2},$$

para $K > 0$ fijo, satisface:

$$\begin{aligned} f(x,y) - Q(x,y) &= \mathcal{O}(h^{2r}) && \textit{en regiones suaves} \\ f(x,y) - Q(x,y) &= \mathcal{O}(h^{r+1}) && \textit{si } \exists \textit{ stencil en región suave.} \end{aligned}$$

para $(x,y) \in \{(-\frac{h}{2},0),(0,-\frac{h}{2}),(-\frac{h}{2},-\frac{h}{2})\}$.

Demostración. Suponemos primero que todos los *stencils* involucrados en el cálculo están en una región donde f es suave. Dado que, por la Proposición 3.1,

$$\hat{Q}(x,y) = \sum_{p,q=0}^{r-1} C_{p,q}^r P_{p,q}^r(x,y)$$

verifica

$$\hat{Q}(x,y) = P_{r-1,r-1}^{2r-1}(x,y), (x,y) \in \{(-\frac{h}{2},0),(0,-\frac{h}{2}),(-\frac{h}{2},-\frac{h}{2})\},$$

a partir del argumento usual del análisis del WENO, es suficiente demostrar que $\omega_{p,q}^r - C_{p,q}^r = \mathcal{O}(h^{r-1})$ para conseguir la primera estimación del error de interpolación.

Denotemos $\varepsilon_h = Kh^2$, $I_{i,j} = I_h(P_{i,j}^r)$ y estimemos la expresión siguiente, usando (3.6), (3.7):

$$\begin{aligned} \frac{\frac{1}{(\varepsilon_h+I_{i,j})^d} - \frac{1}{(\varepsilon_h+I_{p,q})^d}}{\frac{1}{(\varepsilon_h+I_{p,q})^d}} &= \left(\frac{\varepsilon_h + I_{p,q}}{\varepsilon_h + I_{i,j}}\right)^d - 1 \\ &= \left(\frac{\varepsilon_h + I_{p,q}}{\varepsilon_h + I_{i,j}} - 1\right) \sum_{l=0}^{d-1} \left(\frac{\varepsilon_h + I_{p,q}}{\varepsilon_h + I_{i,j}}\right)^l \\ &= \frac{I_{p,q} - I_{i,j}}{Kh^2 + \mathcal{O}(h^2)} \sum_{l=0}^{d-1} \left(\frac{K + \mathcal{O}(h^2)/h^2}{K + \mathcal{O}(h^2)/h^2}\right)^l \\ &= \frac{\mathcal{O}(h^{r+2})}{Kh^2 + \mathcal{O}(h^2)} \sum_{l=0}^{d-1} \mathcal{O}(1)^l = \mathcal{O}(h^r) \end{aligned}$$

Por tanto:

$$\frac{1}{(\varepsilon_h + I_{i,j})^d} = \frac{1}{(\varepsilon_h + I_{p,q})^d}(1 + \mathcal{O}(h^r)), \quad \forall (i,j),(p,q).$$

Entonces para p, q fijos y cualquier $0 \leq i, j < r$,

$$\alpha_{i,j} = \frac{C^r_{i,j}(1 + \mathcal{O}(h^r))}{(\varepsilon_h + I_{p,q})^d},$$

por lo que,

$$\sum_{i,j=0}^{r-1} \alpha_{i,j} = \frac{1}{(\varepsilon_h + I_{p,q})^d} \sum_{i,j=0}^{r-1} C^r_{i,j}(1 + \mathcal{O}(h^r)) = \frac{1 + \mathcal{O}(h^r)}{(\varepsilon_h + I_{p,q})^d},$$

$$\omega^r_{p,q} = \frac{C^r_{p,q}/(\varepsilon_h + I_{p,q})^d}{(1 + \mathcal{O}(h^r))/(\varepsilon_h + I_{p,q})^d} = C^r_{p,q}(1 + \mathcal{O}(h^r)). \tag{3.9}$$

Ahora, si f tiene una singularidad en algún substencil, pero no en todos los $S_{p,q}$, $p, q = 0, \ldots, r-1$, entonces $I_{p,q}/h^2 \not\to 0$, mientras que $I_{p,q} = O(h^2)$ en caso contrario, entonces

$$\alpha_{p,q} = \begin{cases} \mathcal{O}(1) & f \text{ no es suave en } S_{p,q} \\ \mathcal{O}(h^{-2d}) & f \text{ es suave en } S_{p,q} \end{cases},$$

por lo que $\sum_{p,q=0}^{r-1} \alpha_{p,q} = \mathcal{O}(h^{-2d})$ y $\omega^r_{p,q} = \mathcal{O}(h^{2d})$ si f no es suave en $S_{p,q}$. Si denotamos $\mathcal{K} = \{(p,q)/f \text{ no es suave en } S_{p,q}\}$, entonces

$$\begin{aligned} f(x,y) - Q(x,y) &= \sum_{p,q=0}^{r-1} \omega^r_{p,q}\big(f(x,y) - P^r_{p,q}(x,y)\big) \\ &= \sum_{k \in \mathcal{K}} \mathcal{O}(h^{2d})\mathcal{O}(1) + \sum_{k \notin \mathcal{K}} \mathcal{O}(1)\mathcal{O}(h^r) = \mathcal{O}(h^{\min(r,2d)}). \end{aligned}$$

□

La interpolación bidimensional propuesta en el resultado previo generaliza la introducida en [6], en el sentido que si los datos $f_{m,n}$ verifican $f_{m,n} = \tilde{f}_m, \forall n$, entonces $Q(-h/2, -h/2) = \tilde{Q}(-h/2)$, donde $\tilde{Q}$ es el interpolador unidimensional propuesto en [6].

Proposición 3.3. *El indicador de suavidad definido en (3.5) puede ser calculado como:*

$$I_h(P^r_{p,q}) \;=\; \sum_{m_1,m_2=0}^{r} \sum_{n_1,n_2=0}^{r} f_{p-r+m_1,q-r+n_1} f_{p-r+m_2,q-r+n_2} \mu^r_{p,q,m_1,m_2,n_1,n_2}$$

siendo

$$\mu^r_{p,q,m_1,m_2,n_1,n_2} = \sum_{(i,j)\in\mathcal{I}} \lambda^r_{p,m_1,m_2,i}\lambda^r_{q,n_1,n_2,j} \tag{3.10}$$

y

$$\lambda^r_{p,m_1,m_2,l} = (l!)^2 \sum_{j_1,j_2=l}^{r} \binom{j_1}{l}\binom{j_2}{l} a^r_{p,m_1,j_1} a^r_{p,m_2,j_2} \frac{(-1)^{j_1+j_2}}{j_1+j_2-2l+1}, \tag{3.11}$$

con $a^r_{p,m,j} = (B^r_p)^{-1}_{m,j}$ *y*

$$(B^r_p)_{j,i} = (p-r+i)^j, i, j = 0, \ldots, r.$$

Demostración. Si denotamos por $P^r_{p,q}[f,h]$ el interpolador de f en el stencil bidimensional $\mathcal{S}_{p,q}$, entonces tenemos:

$$P^r_{p,q}[f,h](xh,yh) = P^r_{p,q}[S_h f, 1](x,y),$$

con $S_h f(x,y) = f(hx,hy)$, por lo que

$$h^{i+j} P^r_{p,q}[f,h]^{(i,j)}(xh,yh) = P^r_{p,q}[S_h f,1]^{(i,j)}(x,y).$$

Con el cambio de variables $xh = \bar{x}$, $yh = \bar{y}$:

$$\begin{aligned}
&\int_{-1}^{0}\int_{-1}^{0} \left(P^r_{p,q}[S_h f,1]^{(i,j)}(x,y)\right)^2 dxdy \\
&\qquad = h^{-2}\int_{-h}^{0}\int_{-h}^{0} \left(P^r_{p,q}[S_h f,1]^{(i,j)}(\bar{x}/h,\bar{y}/h)\right)^2 d\bar{x}d\bar{y} \\
&\qquad\quad = h^{-2}\int_{-h}^{0}\int_{-h}^{0} \left(h^{i+j} P^r_{p,q}[f,h]^{(i,j)}(\bar{x},\bar{y})\right)^2 d\bar{x}d\bar{y} \\
&\qquad\qquad = h^{2(i+j-1)}\int_{-h}^{0}\int_{-h}^{0} \left(P^r_{p,q}[f,h]^{(i,j)}(\bar{x},\bar{y})\right)^2 d\bar{x}d\bar{y}.
\end{aligned}$$

Por tanto el indicador de suavidad, $I_h(P^r_{p,q})$, se puede escribir como:

$$\sum_{(i,j)\in\mathcal{I}} \int_{-1}^{0}\int_{-1}^{0} \left((P^r_{p,q}[S_h f,1])^{(i,j)}(x,y)\right)^2 dxdy$$

A partir de (3.3)

$$(P^r_{p,q}[S_h f,1])^{(i,j)}(x,y) = \sum_{m=0}^{r}\sum_{n=0}^{r} f_{p-r+m,q-r+n} (L^r_{p,p-r+m})^{(i)}(x)(L^r_{q,q-r+n})^{(j)}(y)$$

deducimos

$$I_h(P^r_{p,q}) = \sum_{(i,j)\in\mathcal{I}} \int_{-1}^{0}\int_{-1}^{0} (P^r_{p,q}[S_h f,1])^{(i,j)}(x,y)^2 dxdy \tag{3.12}$$

$$= \sum_{m_1,m_2=0}^{r}\sum_{n_1,n_2=0}^{r} f_{p-r+m_1,q-r+n_1} f_{p-r+m_2,q-r+n_2} \mu^r_{p,q,m_1,m_2,n_1,n_2} \tag{3.13}$$

donde,

$$\mu^r_{p,q,m_1,m_2,n_1,n_2} = \sum_{(i,j)\in\mathcal{I}} \int_{-1}^{0}\int_{-1}^{0} \psi^r_{p,m_1,m_2,i}(x)\psi^r_{q,n_1,n_2,j}(y)dxdy \tag{3.14}$$

$$\mu^r_{p,q,m_1,m_2,n_1,n_2} = \sum_{(i,j)\in\mathcal{I}} \int_{-1}^{0} \psi^r_{p,m_1,m_2,i}(x)dx \int_{-1}^{0} \psi^r_{q,n_1,n_2,j}(y)dy \tag{3.15}$$

$$\mu^r_{p,q,m_1,m_2,n_1,n_2} = \sum_{(i,j)\in\mathcal{I}} \lambda^r_{p,m_1,m_2,i}\lambda^r_{q,n_1,n_2,j}, \tag{3.16}$$

con

$$\psi^r_{p,m_1,m_2,i}(x) = (L^r_{p,p-r+m_1})^{(i)}(x)(L^r_{p,p-r+m_2})^{(i)}(x),$$
$$\lambda^r_{p,m_1,m_2,i} = \int_{-1}^{0} \psi^r_{p,m_1,m_2,i}(x)dx.$$

Ahora, si

$$L^r_{p,p-r+m}(x) = \sum_{j=0}^{r} a^r_{p,m,j} x^j$$

entonces (3.2) da

$$\sum_{j=0}^{r} a^r_{p,m,j}(p-r+i)^j = \delta_{i,m}, \quad m,i = 0,\ldots,r.$$

Esto es, si definimos $(B^r_p)_{j,i} = (p-r+i)^j, i,j = 0,\ldots,r$, entonces $a^r_{p,m,j} = (B^r_p)^{-1}_{m,j}$.

Teniendo esto en cuenta y también (3.14):

$$(L^r_{p,p-r+m})^{(l)} = l! \sum_{j=l}^{r} \binom{j}{l} a^r_{p,m,j} x^{j-l}$$

$$(L^r_{p,p-r+m_1})^{(l)} (L^r_{p,p-r+m_2})^{(l)} = (l!)^2 \sum_{j_1,j_2=l}^{r} \binom{j_1}{l}\binom{j_2}{l} a^r_{p,m_1,j_1} a^r_{p,m_2,j_2} x^{j_1+j_2-2l}$$

$$\lambda^r_{p,m_1,m_2,l} = (l!)^2 \sum_{j_1,j_2=l}^{r} \binom{j_1}{l}\binom{j_2}{l} a^r_{p,m_1,j_1} a^r_{p,m_2,j_2} \int_{-1}^{0} x^{j_1+j_2-2l} dx$$

$$\lambda^r_{p,m_1,m_2,l} = (l!)^2 \sum_{j_1,j_2=l}^{r} \binom{j_1}{l}\binom{j_2}{l} a^r_{p,m_1,j_1} a^r_{p,m_2,j_2} \frac{(-1)^{j_1+j_2}}{j_1+j_2-2l+1}$$

□

3.4 Experimentos numéricos

El código para la interpolación WENO bidimensional para $r = 2, \ldots, 5$ se puede generar a través de la página web `http://gata.uv.es/weno2d`. Se trata de una implementación en aritmética de múltiple precisión (usando la librería GMP [23]) de las fórmulas expuestas en este capítulo.

En el primer experimento comprobamos numéricamente que el orden del error de la interpolación propuesta en este capítulo es $2r$, cuando se usan *stencils* $2r \times 2r$ en zonas donde la función es suave. Concretamente, consideramos la función

$$f(x,y) = \frac{1}{x^2+y^2+1},$$

y calculamos los errores $e_{1,0}, e_{0,1}, e_{1,1}$ de la interpolación WENO bidimensional con $r = 2, 3$ en $(-\frac{h}{2}, 0), (0, -\frac{h}{2}), (-\frac{h}{2}, -\frac{h}{2})$, respectivamente, para $h = 2^{-i}$, $i = 1, \ldots, 10$. En la tabla 3.1 exponemos los errores obtenidos con $r = 2$. Observamos la convergencia de los órdenes deducidos de los errores hacia 4, lo cual confirma numéricamente el Teorema 3.2. En la tabla 3.2 exponemos los errores obtenidos

h	$e_{1,0}(h)$	$o_{1,0}(h)$	$e_{0,1}(h)$	$o_{0,1}(h)$	$e_{1,1}(h)$	$o_{1,1}(h)$
$5{,}00e-01$	$1{,}66e-02$	3,27	$1{,}66e-02$	3,27	$2{,}86e-02$	3,13
$2{,}50e-01$	$1{,}71e-03$	3,76	$1{,}71e-03$	3,76	$3{,}28e-03$	3,71
$1{,}25e-01$	$1{,}26e-04$	3,91	$1{,}26e-04$	3,91	$2{,}50e-04$	3,90
$6{,}25e-02$	$8{,}39e-06$	3,98	$8{,}39e-06$	3,98	$1{,}67e-05$	3,97
$3{,}12e-02$	$5{,}33e-07$	3,99	$5{,}33e-07$	3,99	$1{,}07e-06$	3,99
$1{,}56e-02$	$3{,}35e-08$	4,00	$3{,}35e-08$	4,00	$6{,}69e-08$	4,00
$7{,}81e-03$	$2{,}09e-09$	4,00	$2{,}09e-09$	4,00	$4{,}19e-09$	4,00
$3{,}91e-03$	$1{,}31e-10$	4,00	$1{,}31e-10$	4,00	$2{,}62e-10$	4,00
$1{,}95e-03$	$8{,}19e-12$	4,00	$8{,}19e-12$	4,00	$1{,}64e-11$	4,00
$1{,}95e-03$	$8{,}19e-12$		$8{,}19e-12$		$1{,}64e-11$	

Tabla 3.1: *Tabla correspondiente al primer experimento con $r = 2$. Se ha usado la notación $o_*(h) = \log_2 \frac{e_*(h)}{e_*(h/2)}$ para los órdenes deducidos de los errores.*

con $r = 3$. Observamos la convergencia de los órdenes deducidos de los errores hacia 6, excepto en aquellos casos en los que el error está cercano a la unidad de redondeo de la doble precisión usada para estos cálculos. Esto también confirma numéricamente el Teorema 3.2 en este caso.

En el segundo experimento comprobamos numéricamente que el orden del error de la interpolación propuesta en este capítulo es $r+1$, cuando hay algún *substencil* $(r+1)\times(r+1)$ donde la función es suave, pero no todos los *substencils* verifican esto. Concretamente, consideramos la función

$$f(x,y) = \begin{cases} \exp(x+y)\cos(x-y) & x+y \leq 0 \\ 1+\exp(x+y)\cos(x-y) & x+y > 0 \end{cases},$$

la cual presenta una discontinuidad a lo largo de la línea $x+y=0$, con lo cual los *stencils* $\{p-r,\dots,p\}\times\{q-r,\dots,q\}$, $p,q=0,\dots,r-1$, cruzan la discontinuidad, excepto cuando $p=q=0$. En este caso calculamos los errores $e_{1,0}, e_{0,1}, e_{1,1}$ de la interpolación WENO bidimensional con $r=2,3$ en $(-\frac{h}{2},0),(0,-\frac{h}{2}),(-\frac{h}{2},-\frac{h}{2})$, respectivamente, para $h=2^{-i}$, $i=1,\dots,10$. En la tabla 3.3 exponemos los errores obtenidos con $r=2$. Observamos la convergencia de los órdenes

h	$e_{1,0}(h)$	$o_{1,0}(h)$	$e_{0,1}(h)$	$o_{0,1}(h)$	$e_{1,1}(h)$	$o_{1,1}(h)$
$5{,}00e-01$	$8{,}70e-03$	4,14	$8{,}70e-03$	4,14	$1{,}49e-02$	3,99
$2{,}50e-01$	$4{,}94e-04$	5,36	$4{,}94e-04$	5,36	$9{,}40e-04$	5,31
$1{,}25e-01$	$1{,}20e-05$	5,88	$1{,}20e-05$	5,88	$2{,}37e-05$	5,86
$6{,}25e-02$	$2{,}05e-07$	5,98	$2{,}05e-07$	5,98	$4{,}08e-07$	5,97
$3{,}12e-02$	$3{,}26e-09$	5,99	$3{,}26e-09$	5,99	$6{,}51e-09$	5,99
$1{,}56e-02$	$5{,}11e-11$	6,00	$5{,}11e-11$	6,00	$1{,}02e-10$	6,00
$7{,}81e-03$	$7{,}99e-13$	5,99	$7{,}99e-13$	6,01	$1{,}60e-12$	6,00
$3{,}91e-03$	$1{,}25e-14$	5,82	$1{,}24e-14$	5,22	$2{,}50e-14$	5,49
$1{,}95e-03$	$2{,}22e-16$	0,00	$3{,}33e-16$	0,58	$5{,}55e-16$	2,32
$1{,}95e-03$	$2{,}22e-16$		$3{,}33e-16$		$5{,}55e-16$	

Tabla 3.2: *Tabla correspondiente al primer experimento con $r = 3$. Se ha usado la notación $o_*(h) = \log_2 \frac{e_*(h)}{e_*(h/2)}$ para los órdenes deducidos de los errores.*

deducidos de los errores hacia 3, lo cual confirma numéricamente en este caso el Teorema 3.2. En la tabla 3.4 exponemos los errores obtenidos con $r = 3$. Observamos la convergencia de los órdenes deducidos de los errores hacia 4. Esto también confirma numéricamente en este caso el Teorema 3.2.

En el último experimento comparamos diversos métodos de interpolación para obtener esquemas de subdivisión (zoom) según el Algoritmo 3.1: interpolación lineal de cuarto orden, interpolación WENO 2D, interpolación WENO tensorial e interpolación WENO híbrida, para la cual la interpolación en $(2i-1, 2j)$ y $(2i, 2j-1)$ se realiza por interpolación WENO tensorial y en $(2i-1, 2j-1)$ por interpolación WENO 2D. Los resultados expuestos en la figura 3.1 revelan que el método lineal, tal como cabía esperar, presenta oscilaciones en los bordes de la imagen (en color blanco). El resultado obtenido con la interpolación WENO2D mejora el anterior en cuanto a disminución de oscilaciones y efecto escalera. El resultado correspondiente a la interpolación WENO tensorial reduce las oscilaciones pero muestra un efecto escalera más acusado. En cambio, el resultado obtenido con el método híbrido muestra un efecto escalera muy reducido sin oscilaciones aparentes.

h	$e_{1,0}(h)$	$o_{1,0}(h)$	$e_{0,1}(h)$	$o_{0,1}(h)$	$e_{1,1}(h)$	$o_{1,1}(h)$
$5{,}00e-01$	$8{,}18e-03$	2,37	$8{,}18e-03$	2,37	$1{,}45e-02$	2,37
$2{,}50e-01$	$1{,}59e-03$	2,81	$1{,}59e-03$	2,81	$2{,}80e-03$	2,75
$1{,}25e-01$	$2{,}26e-04$	2,93	$2{,}26e-04$	2,93	$4{,}17e-04$	2,88
$6{,}25e-02$	$2{,}96e-05$	2,98	$2{,}96e-05$	2,98	$5{,}65e-05$	2,94
$3{,}12e-02$	$3{,}77e-06$	2,99	$3{,}77e-06$	2,99	$7{,}34e-06$	2,97
$1{,}56e-02$	$4{,}74e-07$	3,00	$4{,}74e-07$	3,00	$9{,}36e-07$	2,99
$7{,}81e-03$	$5{,}94e-08$	3,00	$5{,}94e-08$	3,00	$1{,}18e-07$	2,99
$3{,}91e-03$	$7{,}44e-09$	3,00	$7{,}44e-09$	3,00	$1{,}48e-08$	3,00
$1{,}95e-03$	$9{,}31e-10$	3,00	$9{,}31e-10$	3,00	$1{,}86e-09$	3,00
$1{,}95e-03$	$9{,}31e-10$		$9{,}31e-10$		$1{,}86e-09$	

Tabla 3.3: *Tabla correspondiente al segundo experimento con $r = 2$. Se ha usado la notación $o_*(h) = \log_2 \frac{e_*(h)}{e_*(h/2)}$ para los órdenes deducidos de los errores.*

h	$e_{1,0}(h)$	$o_{1,0}(h)$	$e_{0,1}(h)$	$o_{0,1}(h)$	$e_{1,1}(h)$	$o_{1,1}(h)$
$5{,}00e-01$	$5{,}57e-03$	3,37	$5{,}57e-03$	3,37	$8{,}15e-03$	3,27
$2{,}50e-01$	$5{,}38e-04$	3,71	$5{,}38e-04$	3,71	$8{,}41e-04$	3,66
$1{,}25e-01$	$4{,}11e-05$	3,86	$4{,}11e-05$	3,86	$6{,}67e-05$	3,83
$6{,}25e-02$	$2{,}84e-06$	3,93	$2{,}84e-06$	3,93	$4{,}69e-06$	3,92
$3{,}12e-02$	$1{,}87e-07$	3,96	$1{,}87e-07$	3,96	$3{,}11e-07$	3,96
$1{,}56e-02$	$1{,}20e-08$	3,98	$1{,}20e-08$	3,98	$2{,}00e-08$	3,98
$7{,}81e-03$	$7{,}57e-10$	3,99	$7{,}57e-10$	3,99	$1{,}27e-09$	3,99
$3{,}91e-03$	$4{,}76e-11$	4,00	$4{,}76e-11$	4,00	$7{,}98e-11$	3,99
$1{,}95e-03$	$2{,}99e-12$	4,00	$2{,}99e-12$	4,00	$5{,}01e-12$	4,00
$1{,}95e-03$	$2{,}99e-12$		$2{,}99e-12$		$5{,}01e-12$	

Tabla 3.4: *Tabla correspondiente al segundo experimento con $r = 3$. Se ha usado la notación $o_*(h) = \log_2 \frac{e_*(h)}{e_*(h/2)}$ para los órdenes deducidos de los errores.*

ALGORITMO 3.1. Algoritmo de subdivisión asociado a operadores de interpolación P_{k-1}^k.

for $k = 1, ..., L$
  for $i, j = 1, ..., N_{k-1}$
    $\hat{f}^k_{2i-1,2j-1} = \left(P^k_{k-1}\hat{f}^{k-1}\right)_{2i-1,2j-1}$
    $\hat{f}^k_{2i-1,2j} = \left(P^k_{k-1}\hat{f}^{k-1}\right)_{2i-1,2j}$
    $\hat{f}^k_{2i,2j-1} = \left(P^k_{k-1}\hat{f}^{k-1}\right)_{2i,2j-1}$
    $\hat{f}^k_{2i,2j} = \left(P^k_{k-1}\hat{f}^{k-1}\right)_{2i,2j}$
  end
end

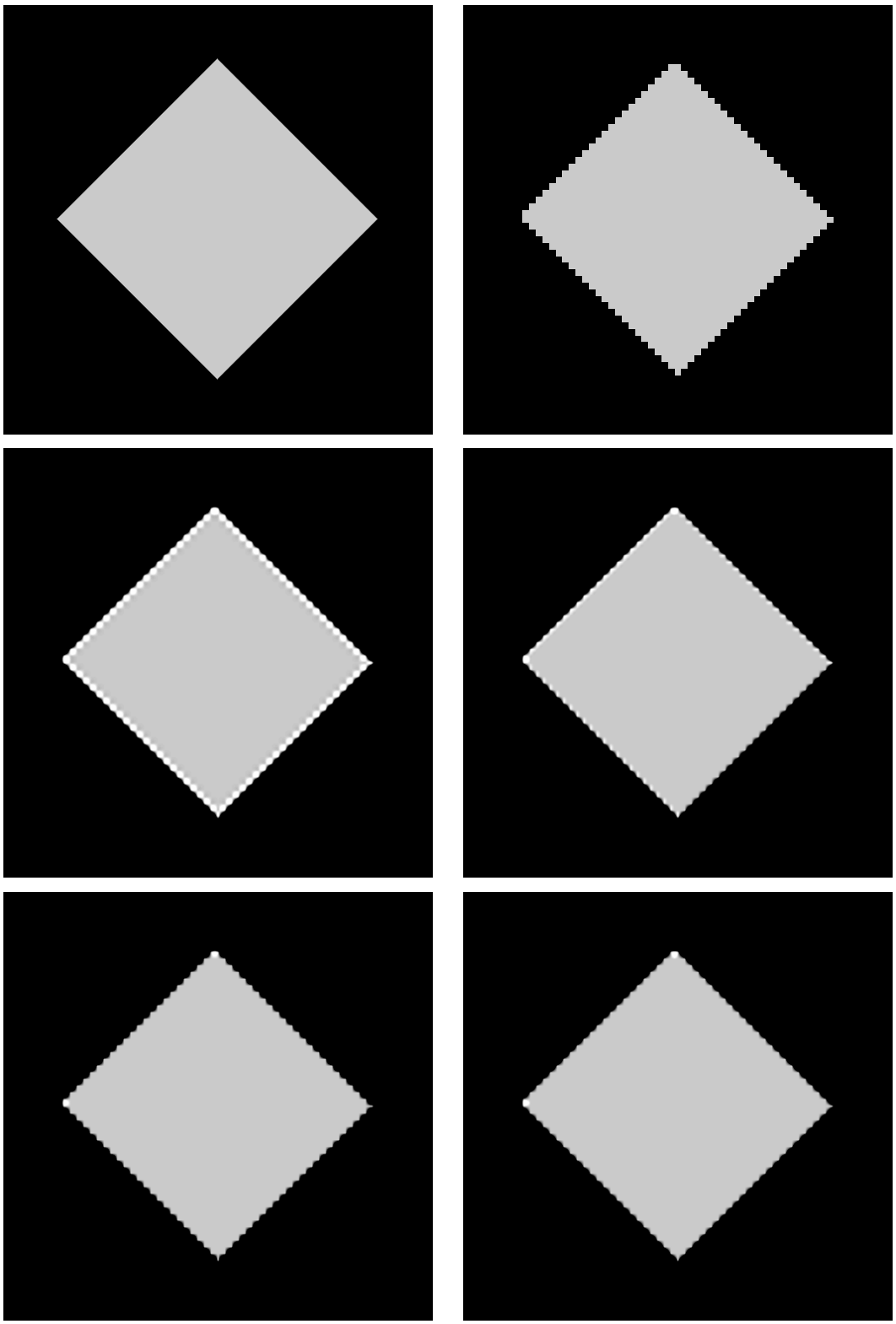


Figura 3.1: *De arriba a abajo y de izquierda a derecha: imagen original, imagen decimada a 3 niveles de resolución, zoom por interpolación lineal de cuarto orden, zoom por interpolación WENO 2D, zoom por interpolación WENO tensorial y zoom por interpolación WENO híbrida.*

Parte II

Restauración de imágenes

4

Eliminación de ruido en imágenes

4.1 Introducción

El procesamiento de imágenes abarca una serie de tareas cuyo objetivo final es la extracción de la información contenida en ellas. Algunas de estas tareas son la restauración, la segmentación o la compresión. En esta parte de la memoria nos dedicamos a la *restauración de imágenes*, es decir, el proceso de suprimir en la medida de lo posible aquellas distorsiones que se hayan podido producir en el proceso de captación de la imagen. Esta tarea se puede considerar una tarea de bajo nivel dentro del proceso de imágenes y es conveniente, si no necesaria, para las tareas de nivel

más alto, como puede ser el reconocimiento de formas, necesario en muchas áreas, como, por ejemplo, control de calidad o robótica.

Figura 4.1: *eliminación del ruido de una imagen utilizando Variación Total.*

La restauración de imágenes implica muchas tareas computacionalmente intensivas, puesto que el número de píxeles, incluso para una imagen de dos dimensiones con una resolución modesta, a menudo sobrepasa los varios cientos de miles (por no hablar de las figuras en 3D o las animaciones).

Los métodos estándar implican la computación en el campo de las frecuencias, facilitada por la transformada de Fourier (algoritmos FFT) y más recientemente por los wavelets. En los últimos años los pasos se han encaminado hacia las EDP, lo que nos lleva a usar técnicas algebraicas, es decir, manipulación de grandes sistemas de ecuaciones. Los nuevos modelos están motivados por una aproximación más sistemática para restaurar imágenes con grandes perfiles. Los métodos basados en la Variación Total pertenecen a esta nueva clase de modelos.

Desde el punto de vista computacional, la formulación basada en EDPs nos conduce a problemas que requieren unas técnicas que puedan explotar mejor la naturaleza fundamental de las EDPs no lineales. Hasta ahora, los modelos de eliminación de ruido no lineales eran considerados algo "caros" comparados con los métodos tradicionales y naturalmente una parte de las investigaciones en este área van dirigidas a hacerlos más eficientes al tiempo que conserven los rasgos principales de la imagen.

El registro de una imagen supone normalmente un proceso de degradación: un *emborronamiento*, debido a turbulencias atmosféricas, desenfoque de la cámara o movimiento relativo, seguido

por un *ruido* aleatorio debido a errores de los sensores físicos. El modelo actual de degradación depende de muchos factores, pero se usa comúnmente un modelo con un operador de difusión lineal y un ruido gaussiano blanco aditivo. Otros modelos utilizan operadores de difusión no lineales, ruido multiplicativo o ruido con distribuciones más complicadas y con posible correlación con la imagen.

El objetivo de la restauración de imágenes es la estimación de la imagen real ideal partiendo de la imagen registrada. Vamos a intentar resolverlo matemáticamente con una formulación variacional, esto significa que adoptamos el punto de vista continuo en escala de grises y por tanto nos vamos a referir a una imagen como una función real f, definida en $\Omega = (0,1)^2$, con $f(x,y)$ la intensidad (o nivel de gris) en el píxel (x,y):

$$f : \Omega \longrightarrow \mathbb{R}.$$

Una imagen en color sería una función vectorial $\vec{f} : \Omega \longrightarrow \mathbb{R}^3$. En este trabajo nos restringiremos al caso de las imágenes en escala de grises.

Llamaremos u a la imagen real que queremos estimar y z a la imagen observada. Modelamos el sistema de captura de imágenes como:

$$z = Ku + N,$$

donde K es un operador lineal (el operador de difusión) y N es el ruido, del cual podemos conocer, a lo sumo, algunos de sus parámetros estadísticos.

Normalmente K es una convolución por una PSF (función de extensión de punto) g conocida, que se obtiene de una distorsión de una imagen puntual

$$Ku(x,y) = \int_\Omega g(x-\xi, y-\eta)\, u(\xi,\eta)\, d\xi d\eta$$

que corresponde a un proceso de emborronamiento *invariante por translación*. Esto supone una buena situación computacional, ya que se puede aplicar FFT para calcular K.

Numerosas degradaciones o emborronamientos habituales han sido modelados ya, y podemos usar esa g para restaurar la imagen.

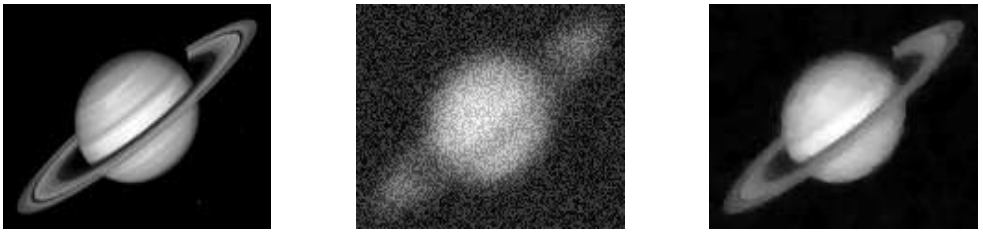


Figura 4.2: *a) imagen real, b) imagen con ruido y difusión, c) imagen restaurada.*

Algunas degradaciones sencillas vienen dadas por las transformadas de Fourier siguientes:

- Movimiento relativo de cámara y objeto

$$\hat{g}(\xi,\eta) = \frac{\sin(\pi\xi)}{\pi\xi}.$$

- Turbulencia atmosférica

$$\hat{g}(\xi,\eta) = e^{-c(\xi^2+\eta^2)^{\frac{5}{6}}}.$$

Como ilustración de las dificultades que se pueden encontrar al aplicar técnicas de restauración lineales, consideramos la restauración por deconvolución. Como nuestro objetivo es obtener u de $z = Ku + N$ y sabemos que $Ku = g * u$ entonces $z = g * u + N$. Si usamos transformadas de Fourier tenemos:

$$\hat{z} = \hat{g}\hat{u} + \hat{N}$$

de donde podemos despejar $\hat{u}$.

$$\hat{u} = \frac{\hat{z}}{\hat{g}} - \frac{\hat{N}}{\hat{g}}$$

y así, si

$$\frac{\hat{N}}{\hat{g}} \approx 0$$

entonces la solución será

$$u \approx \left(\frac{\hat{z}}{\hat{g}}\right)^{\vee}.$$

Pero el problema es que la suposición que hemos hecho no es válida pues $\hat{g}$ es pequeño para altas frecuencias i.e. $\hat{g}(k) \approx 0$ para $k \gg 0$ y además $\hat{N}(k)$ es una variable aleatoria, por tanto

$$\frac{\hat{N}(k)}{\hat{g}(k)} \gg 0 \quad \text{para} \quad k \gg 0.$$

Esta solución, que consiste en multiplicar $\hat{u}$ por $h := \frac{1}{\hat{g}}$, "excita" las altas frecuencias y los resultados no son demasiado buenos. Otra posibilidad es una h de la forma:

$$h(k) = \begin{cases} \hat{g}(k)^{-1} & k \text{ pequeña}, \\ \text{"pequeño"} & k \text{ grande}. \end{cases} \tag{4.1}$$

Sólo nos queda definir $u = (\hat{z}h)^{\vee}$ y esperar que $h\hat{N}$ sea pequeño.

Este pequeño análisis ilustra que K es usualmente compacto y, en este caso, el problema $Ku = z$ está mal puesto (no existe K^{-1} o K^{-1} es discontinua), por tanto no existe solución o ésta no depende de z de manera continua. En consecuencia, hay que reformular la restauración de imágenes como un problema bien puesto, es decir, regularizar el problema.

El caso que se trata en los capítulos siguientes es cuando K es la identidad, es decir, trataremos el problema de la supresión de ruido.

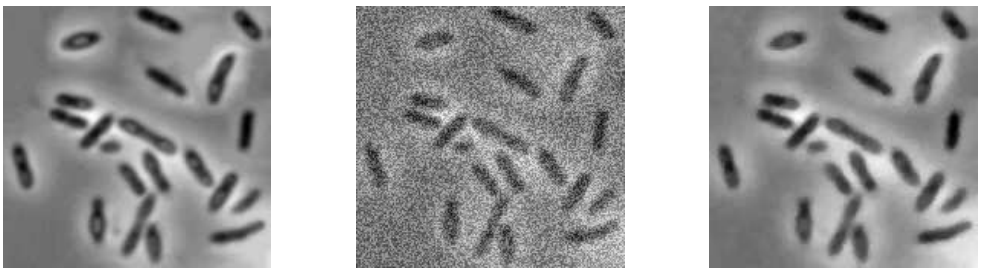


Figura 4.3: *(bacterias) a) imagen real, b) imagen con ruido, c) imagen restaurada.*

4.2 Descripción del problema

Vamos a considerar una imagen en escala de grises $u : \Omega \longrightarrow \mathbb{R}$, definida en el cuadrado $\Omega = (0,1)^2$. Denotaremos por u a la imagen real que queremos observar y $z : \Omega \longrightarrow \mathbb{R}$ a la imagen observada. Modelamos el sistema de adquisición de imágenes como:

$$z = u + N$$

donde $N : \Omega \longrightarrow \mathbb{R}$ representa un ruido aditivo. Suponemos que N es un ruido gaussiano blanco, lo que significa para cada píxel (x, y), $N(x, y)$ es una variable aleatoria con distribución gaussiana de media cero y varianza σ^2 y que la correlación de estas variable en distintos píxeles es nula. Debido a esto, se verifican las dos propiedades siguientes:

1. La media de la función ruido es cero en Ω:

$$\bar{N} := |\Omega|^{-1} \int_\Omega N \, dxdy = \int_\Omega N \, d\mathbf{x} = 0.$$

   Esto implica que los niveles de gris medios $\bar{z}$ y $\bar{u}$ de z y de u son iguales en Ω:

$$z = u + N \quad \rightarrow \quad z - u = N$$

$$\int_\Omega (z - u) \, d\mathbf{x} = \int_\Omega N \, d\mathbf{x} = 0$$

$$\int_\Omega z \, d\mathbf{x} = \int_\Omega u \, d\mathbf{x} \quad \rightarrow \quad \bar{z} = \bar{u}$$

2. Suponiendo conocida la varianza σ^2 de N, tenemos:

$$\sigma^2 = \int_\Omega N^2 \, d\mathbf{x} - \left(\int_\Omega N \, d\mathbf{x} \right)^2 = \int_\Omega N^2 \, d\mathbf{x} = \int_\Omega (z - u)^2 \, d\mathbf{x}. \quad (4.2)$$

Está claro que la aleatoriedad del ruido no nos permitirá obtener exactamente la imagen real u a partir de la imagen observada

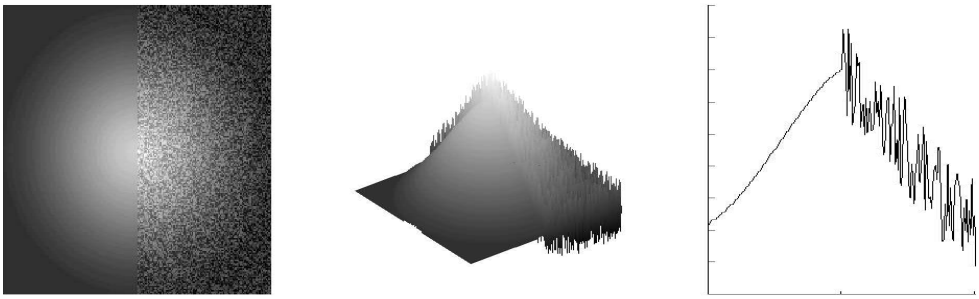


Figura 4.4: *a) imagen con ruido del 30 % en su parte derecha, b) la misma imagen pero vista en 3D, c) corte horizontal por la mitad de la imagen.*

z. Sin embargo, debemos intentar encontrar una solución con menos ruido que la imagen observada y que conserve los contornos más importantes de u.

Una imagen con ruido es una imagen muy irregular desde el punto de vista funcional, tiene por tanto, mucha variación. Sea $\mathcal{R} : X \to \mathbb{R}$ un funcional que mide la irregularidad de la función $u \in X \subseteq L^2$, al que vamos a exigir que $\mathcal{R}(u)$ esté en función de $|\nabla u|$, $\mathcal{R}(u) = \widetilde{\mathcal{R}}(|\nabla u|)$, donde $\widetilde{\mathcal{R}}$ es una función positiva.

Si buscamos una imagen u con menos ruido que z, debemos encontrar una función u que sea más suave que z, o lo que es lo mismo, que sea menos irregular, es decir:

$$\mathcal{R}(u) \leqslant \mathcal{R}(z).$$

Si suponemos conocida la varianza del ruido, la expresión (4.2) nos permite formular el problema de la supresión del ruido como:

$$\begin{cases} \min\limits_{u \in X} \quad \mathcal{R}(u) \\ \text{sujeto a} \quad \displaystyle\int_\Omega (u - z)^2 \, d\mathbf{x} = \sigma^2 \end{cases} \tag{4.3}$$

Para resolverlo, podemos usar la técnica de los multiplicadores de Lagrange:

- Creamos el lagrangiano:

$$L(u, \lambda) = \mathcal{R}(u) + \lambda \left(\|u - z\|_{L^2}^2 - \sigma^2 \right).$$

- Igualamos a cero sus derivadas parciales:

$$\begin{cases} \dfrac{\partial L}{\partial u}(u,\lambda) = 0 \\ \dfrac{\partial L}{\partial \lambda}(u,\lambda) = \|u-z\|_{L^2}^2 - \sigma^2 = 0. \end{cases}$$

Nuestro problema se reduce a encontrar puntos críticos de $L(u,\lambda)$, pero antes vamos a hacer un pequeño cambio:

$$L(u,\lambda) = 2\lambda\left(\frac{1}{2\lambda}\mathcal{R}(u) + \frac{1}{2}\|u-z\|_{L^2}^2\right) - \lambda\sigma^2.$$

Definimos:

$$\alpha := \frac{1}{2\lambda}$$

y como $\lambda\sigma^2$ es constante y 2λ es positivo, calcular el mínimo de $L(u,\lambda)$ es equivalente a calcular

$$\min_{u\in X} \alpha\mathcal{R}(u) + \frac{1}{2}\|u-z\|_{L^2}^2$$

que es la regularización que propuso Tikhonov en [55], donde α nos sirve como control del equilibrio entre la irregularidad de la solución y el ajuste de los datos. Si aumenta α, $\mathcal{R}$ deberá disminuir, con lo que la imagen será más suave; por el contrario, si α disminuye, $\mathcal{R}$ aumentará y entonces la solución será más irregular.

Consideramos, por tanto, el siguiente funcional

$$T(u) \;=\; T_\alpha(u) \;=\; \alpha\mathcal{R}(u) + \frac{1}{2}\|u-z\|_{L^2}^2$$

y el problema de encontrar una $u^* \in X$ tal que

$$T(u^*) = \min_{u\in X} T(u).$$

4.3 Funcionales cuadráticos

La elección usual del funcional $\mathcal{R}$ que nos mide la irregularidad de la imagen u es un funcional cuadrático de la forma $\mathcal{R}(u) =$

$\|Qu\|_{L^2}^2$, siendo Q un funcional lineal. Entonces podemos escribir:

$$T(u) = \alpha \|Qu\|_{L^2}^2 + \|u - z\|_{L^2}^2 = \left\| \begin{bmatrix} I \\ \sqrt{\alpha} Q \end{bmatrix} u - \begin{bmatrix} z \\ 0 \end{bmatrix} \right\|_{L^2}^2 = \|Au - f\|_{L^2}^2 \tag{4.4}$$

como se expone en [16]. Tenemos pues, un problema común de mínimos cuadrados, con el inconveniente, como veremos más adelante, de que suaviza demasiado los grandes perfiles. Vamos a elegir $Qu = \nabla u$ y veremos con algunos ejemplos, a qué soluciones podemos llegar resolviendo el problema (4.3). (Para resolver el problema hemos utilizado métodos numéricos que serán explicados en la sección 5.4.

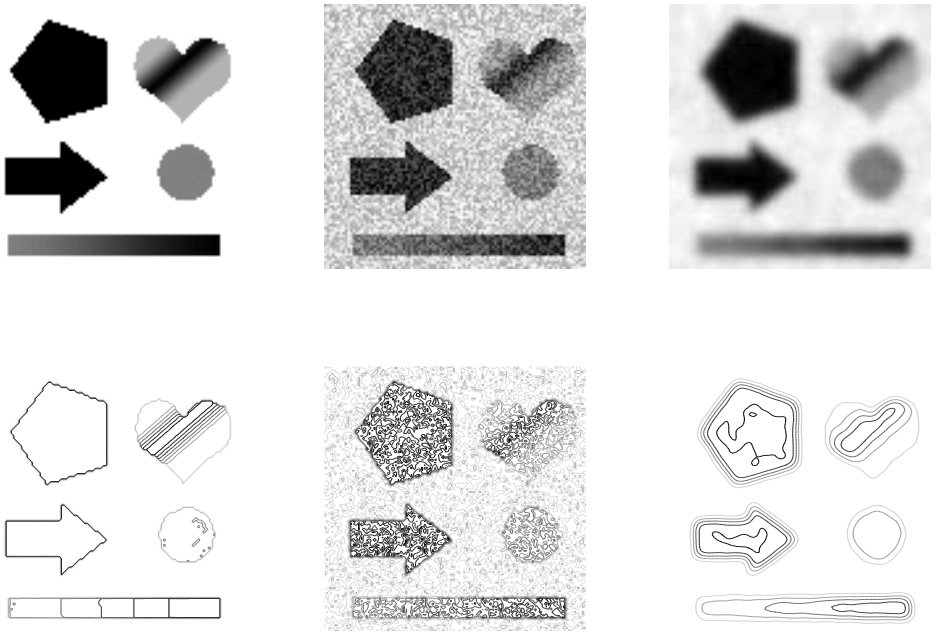


Figura 4.5: *De izquierda a derecha y de arriba a abajo: imagen original, imagen con un ruido del 30%, imagen restaurada, contorno de la imagen original, contorno de la imagen con ruido y contorno de la restaurada.*

Con estos ejemplos se ve claramente que el funcional cuadrático $\mathcal{R}(u) = \|\nabla u\|_{L^2(\Omega)}^2$ suaviza demasiado las imágenes y no conserva los perfiles más grandes de las mismas. Por tanto, no es una

Figura 4.6: *imagen original, imagen con un ruido del 20%, imagen restaurada, contorno de la imagen original, contorno de la imagen con ruido y contorno de la restaurada.*

buena elección si estamos interesados en restaurar imágenes con grandes perfiles.

4.4 El funcional Variación Total

El inconveniente principal de usar funcionales de regularización cuadráticos como la seminorma H^1 de u, $\|\nabla u\|^2_{L^2(\Omega)}$, es la incapacidad de recuperar grandes discontinuidades. Matemáticamen-

te está claro, porque las funciones discontinuas no tienen seminormas de H^1 acotadas. Por ejemplo:

$$\textbf{si} \quad u_\alpha(x) := \frac{\arctan(\alpha x)}{\pi} + \frac{1}{2} \Rightarrow \lim_{\alpha\to\infty} u_\alpha(x) = \begin{cases} 0 & x \leqslant 0 \\ 1 & x > 0 \end{cases}, \quad \alpha > 0.$$

Entonces

$$|u_\alpha|_{H^1} = \int_{-\infty}^{+\infty} (u'_\alpha(x))^2 dx = \frac{\alpha}{2\pi},$$

lo que da:

$$\lim_{\alpha\to\infty} \frac{\alpha}{2\pi} = \infty.$$

Para evitar esto, Rudin, Osher y Fatemi propusieron en [46] la utilización de la *Variación Total*

$$TV(u) = \int_\Omega |\nabla u| d\mathbf{x} = \int_\Omega \sqrt{u_x^2 + u_y^2}\, dxdy$$

como funcional de regularización. La razón para usar este funcional, es que mide los *saltos* de u, incluso si es discontinua. Veamos cual es la variación total de la función del ejemplo anterior:

$$\left|u'_\alpha(x)\right| = \frac{\alpha}{\pi(\alpha^2x^2+1)}, \quad \int_{-\infty}^{+\infty} \left|u'_\alpha(x)\right| dx = \left.\frac{\arctan(\alpha x)}{\pi}\right|_{-\infty}^{\infty} = 1$$

Hay una definición alternativa del funcional variación total para cuando u no es diferenciable (ver [22]):

$$TV(u) := \sup\left\{ \int_\Omega u \,\mathrm{div}\,\mathbf{v}\, d\mathbf{x}\,;\, \mathbf{v} \in C_0^\infty(\Omega;\mathbb{R}^2) : |\mathbf{v}(\mathbf{x})| \leqslant 1 \,\forall \mathbf{x} \in \Omega \right\}$$

El siguiente resultado visto en [16] muestra, para un problema de 1D simple, la ventaja de usar Variación Total en vez del funcional cuadrático dado por la seminorma de H^1

Lema 4.1. *Sea* $S = \{\, f \mid f(0) = a, f(1) = b\}$*. Entonces:*

1. *Para la norma de variación total, se cumple que* $\min_{f\in S} TV(u) = b - a$ *se consigue con cualquier* $f \in S$ *monótona, no necesariamente continua.*

2. *Para la seminorma* H^1*, se cumple que* $\min_{f \in S} \int_\Omega \left(f'(x)\right)^2 d\mathbf{x}$ *tiene una única solución* $f(x) = a(1-x) + bx$*.*

Así, la norma variación total admite más funciones como solución, incluyendo las discontinuas, para restaurar la función con ruido, mientras que la seminorma H^1 "prefiere" las funciones lineales. Por tanto $TV(u)$ permite recubrir los perfiles mayores de la imagen.

El funcional que hay que minimizar ahora es

$$T(u) = \int_\Omega \alpha \left|\nabla u\right| + \frac{1}{2}(u-z)^2 dxdy.$$

Veamos también, con algunos ejemplos, que resultados se obtienen minimizando el funcional anterior, con los métodos numéricos del capítulo 5. Compararemos además, estos resultados con los obtenidos con el funcional cuadrático.

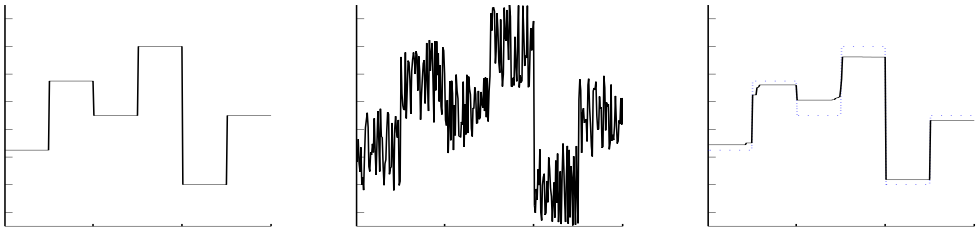


Figura 4.7: *Función original, con un ruido del 30 % y función restaurada con variación total.*

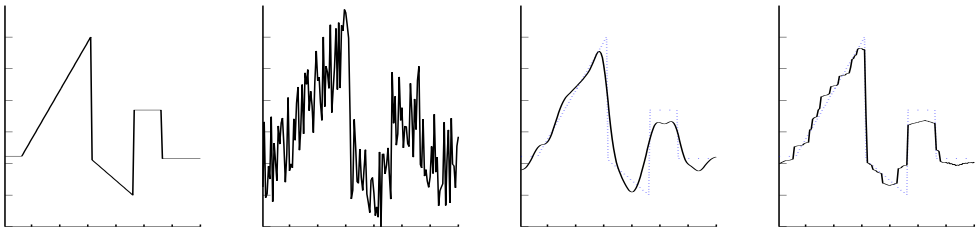


Figura 4.8: *Función original, con un ruido del 20 %, función restaurada con funcional cuadrático y función restaurada con variación total.*

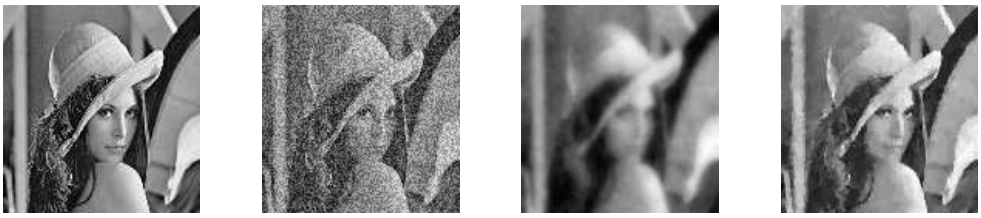


Figura 4.9: *Imagen de Lena (128×128), imagen con un ruido del 30%, imagen restaurada con funcional cuadrático e imagen restaurada con variación total.*

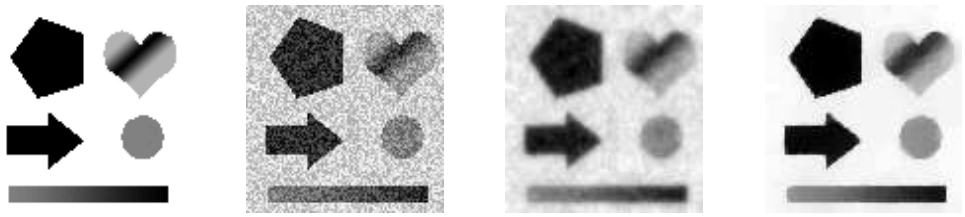


Figura 4.10: *Imagen original (100×100), con un ruido del 30%, imagen restaurada con funcional cuadrático e imagen restaurada con variación total.*

Se ve claramente en las figuras 4.7, 4.8, 4.9 y 4.10 que el funcional TV conserva mejor los grandes perfiles que el funcional cuadrático.

Con estas imágenes se puede observar que la variación total funciona mejor que el funcional cuadrático aunque para imágenes reales como la de Lena, que tienen muchos perfiles, el resultado es mejorable.

4.5 Efecto escalera. Generalización de $\mathcal{R}(u)$

Como se ve en la figura 4.8, en una dimensión, el funcional Variación Total aplicado a una función afín degradada con ruido, produce una solución monótona constante a trozos. Esto es debido

en parte al hecho que la norma TV no tiene preferencias entre funciones continuas y discontinuas, como se explica en el lema 4.1. En cambio el funcional

$$H^1(u) = \int_\Omega |\nabla u|^2 \, d\mathbf{x}$$

"prefiere" las funciones continuas frente a las que no lo son.

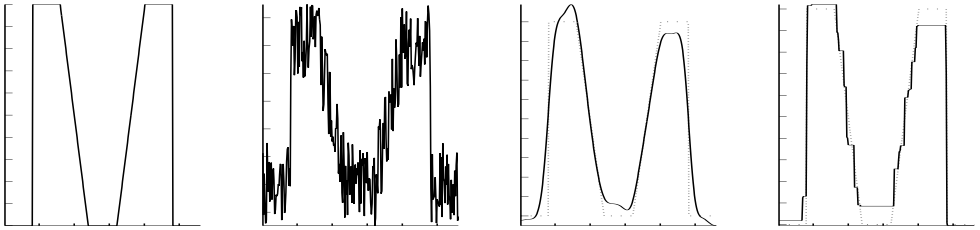


Figura 4.11: *Esta figura muestra los resultados que produce la norma H^1 y la norma TV en las partes lineales y en los saltos más grandes, al restaurar una función con un ruido del 30% de la señal.*

Para resolver este problema, la propuesta de Blomgren, Chan, Mulet y Wong en [41] es interpolar entre el espacio de funciones de TV, $W_1^1(\Omega)$, y el espacio de la seminorma H^1, $W_2^1(\Omega)$, para coger lo mejor de cada espacio, en relación a mantener grandes perfiles y suavizar las regiones de transición sin saltos, considerando

$$\mathcal{R}(u) = \int_\Omega \phi(|\nabla u|) \, d\mathbf{x} \tag{4.5}$$

para funciones reales ϕ adecuadas. Por ejemplo, la norma TV se obtiene con $\phi(s) = s$ y el funcional H^1 se obtiene con $\phi(s) = s^2$. Para interpolar entre las dos, se puede considerar $\phi(s) = s^p$ con $p \in [1, 2]$

$$\mathcal{R}(u) = \int_\Omega |\nabla u|^p \, d\mathbf{x}. \tag{4.6}$$

Pero el único caso en que se admiten soluciones discontinuas es cuando $p = 1$, que es precisamente la norma TV. Otro hecho importante es que la penalización de gradientes grandes aumenta con p. Esto trataremos de mejorarlo en el capítulo 6, buscando un exponente p cercano a 1 cuando estemos cerca de un gran salto

de la función y un exponente p cercano a 2 en las regiones más suaves.

Para ilustrar estos hechos, mostramos en la figura 4.12 algunos resultados para comprobar qué soluciones se obtiene con el funcional (4.6) para algunos valores del exponente p.

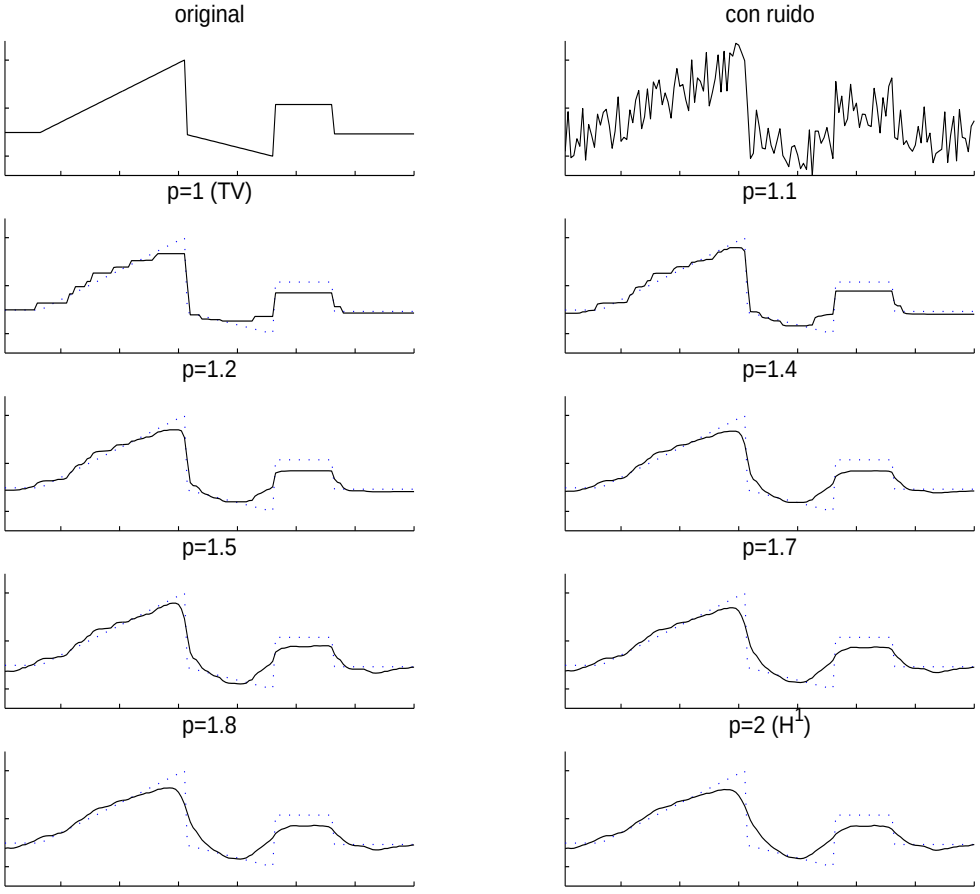


Figura 4.12: *función de 512 puntos. Soluciones para p=1, 1.1, 1.2, 1.4, 1.5, 1.7,1.8, 2, del problema con el funcional (4.6).*

5

Discretización y solución del problema

Consideraremos en adelante la resolución de problemas

$$T(u) = \frac{1}{2}\int_{\Omega}(u-z)^2\,d\mathbf{x} + \alpha\int_{\Omega}\phi(|\nabla u|)\,d\mathbf{x} \tag{5.1}$$

con ϕ no negativa, no decreciente y convexa, u perteneciente a algún espacio funcional adecuado y con el abuso de notación $|\nabla u| = \sqrt{|\nabla u|^2 + \beta}$, con $\beta > 0$. El análisis teórico de la solución de este problema requiere conceptos de análisis funcional no contemplados en esta memoria. Por este motivo, nos planteamos discretizar el funcional T para analizar la existencia y unicidad de la solución, así como su estabilidad frente a cambios de β.

Para discretizar el funcional T usamos la regla del rectángulo para la aproximación de las integrales y diferencias hacia delante para las derivadas: tomamos $u \in \mathbb{R}^{n\times n}$ y, obviando un factor h^2,

$h = \frac{1}{n}$, que no necesitamos para calcular el mínimo, obtenemos:

$$T_h(u) = \alpha \sum_{i,j=1}^{n} \phi\left(|\nabla_{h,i,j} u|\right) + \frac{1}{2} \sum_{i,j=1}^{n} \left(u_{i,j} - z_{i,j}\right)^2$$

con $u_{i,j} \approx u\,(ih, jh)$, $i, j = 0, \ldots, n$ y

$$\nabla_{h,i,j} u = \begin{pmatrix} \frac{u_{i+1,j} - u_{i,j}}{h} \\ \frac{u_{i,j+1} - u_{i,j}}{h} \end{pmatrix}.$$

Debido a que no habrá posibilidad de confusión en el resto de la memoria, suprimiremos en adelante el subíndice h.

5.1 Existencia y unicidad

Una vez discretizado, la existencia de mínimos del problema (5.1) es consecuencia de resultados usuales de análisis convexo (ver [18, 45]). Por completitud esbozamos aquí su demostración.

Dado que el funcional T es no negativo, podemos tomar una sucesión minimizadora del problema, que es una sucesión en $\mathbb{R}^{n\times n}$ $(u_k)_{k\in\mathbb{N}}$ tal que

$$\lim_k T(u_k) = \inf_{u\in\mathbb{R}^{n\times n}} T(u) \geq 0.$$

Dado que $T(u) \geq ||u - z||_2^2$ (es *coercivo*), tenemos que la sucesión $(u_k - z)$ es acotada y, por tanto, (u_k) también lo es. Si tomamos $(u_{k_j})_j$ una subsucesión convergente a u_*, la continuidad de ϕ implica la de T, por lo que tenemos

$$T(u_*) = \lim_j T(u_{k_j}) = \inf_{u\in\mathbb{R}^{n\times n}} T(u).$$

La *convexidad* es un ingrediente esencial para probar la unicidad de las soluciones. Veamos en qué consiste.

Definición 5.1. *Un funcional $F : X \to \mathbb{R}$, donde X es un espacio lineal, se dice que es convexo si para $x, y \in X$ y para cualquier $\lambda \in (0, 1)$*

$$F(\lambda x + (1 - \lambda)y) \leqslant \lambda F(x) + (1 - \lambda)F(y).$$

F se dice estrictamente convexo si la desigualdad anterior es estricta siempre que $x \neq y$. Notar que la convexidad (estricta) se preserva por sumas y productos por escalares positivos.

Un conjunto $M \subset X$ se dice convexo si $\forall x, y \in M$ y para cualquier $\lambda \in (0,1)$

$$\lambda x + (1-\lambda)y \in M.$$

Teorema 5.2. *Sea $M \subset X$ un conjunto convexo de un espacio lineal X y sea $F : M \to \mathbb{R}$ un funcional convexo. Entonces, cualquier minimizador local $x \in M$ es también un minimizador global (en M). Si F es estrictamente convexo entonces existe como mucho un minimizador global.*

Demostración. Supongamos que en $\tilde{x} \in X$ hay un mínimo local. Sea $U \subset M$ un entorno de $\tilde{x}$ tal que $F(\tilde{x}) \leqslant F(y) \, \forall \, y \in U$. Entonces par un $x \in M$ arbitrario, existe un número real $\lambda \in (0,1)$, tal que $\lambda x + (1-\lambda)\tilde{x} \in U$ y, por tanto,

$$F(\tilde{x}) \leqslant F(\lambda x + (1-\lambda)\tilde{x}) \leqslant \lambda F(x) + (1-\lambda)F(\tilde{x}),$$

lo que implica inmediatamente que $F(\tilde{x}) \leqslant F(x)$. Así que $\tilde{x}$ es un minimizador global. Para la segunda parte supongamos que hay dos minimizadores globales diferentes x, $\tilde{x} \in M$. Tenemos obviamente que $F(x) = F(\tilde{x})$ y por la convexidad estricta podemos concluir

$$F(\lambda x + (1-\lambda)\tilde{x}) < \lambda F(x) + (1-\lambda)F(\tilde{x}) = F(x), \forall \lambda \in (0,1)$$

lo cual está en contradicción con el hecho de que x es un minimizador global de F. □

Volviendo al problema de eliminación de ruido con el funcional objetivo

$$T(u) = \frac{1}{2}\|u - z\|_2^2 + \alpha \underbrace{\sum_{i,j=1}^{n} \phi\left(|\nabla_{i,j} u|\right)}_{=\mathcal{R}(u)} \tag{5.2}$$

la cuestión es conocer bajo qué condiciones de ϕ, el funcional T es convexo o estrictamente convexo. Notar que el *término distancia* $\|u - z\|_2^2$ es estrictamente convexo. Si además el *término de suavidad* $\mathcal{R}$ es convexo, entonces todo el funcional T es estrictamente

convexo. Por tanto, nos queda la cuestión de saber si para una cierta función ϕ, el término de suavidad es convexo o no.

Teorema 5.3. *Sea $\phi : \mathbb{R}^+ \to \mathbb{R}^+$ una función continua y no decreciente. Entonces el funcional*

$$\mathcal{R}(u) = \sum_{i,j=1}^{n} \phi\left(|\nabla_{i,j} u|\right)$$

es convexo si y sólo si ϕ es convexa en $\mathbb{R}$.

Demostración. Sea $\phi : \mathbb{R}^+ \to \mathbb{R}^+$ convexa, como $\nabla_{i,j}(\cdot)$ es lineal y $|\cdot|$ es convexa, la suma de convexas es convexas y la composición de funciones convexas es otra función convexa, entonces $\mathcal{R}(u), u \in \mathbb{R}^{n\times n}$ es un funcional convexo.

Supongamos ahora que $\mathcal{R}$ es un funcional convexo. Sea $\mathbf{e}$ la matriz dada por $\mathbf{e}_{i,j} = ih$, que verifica $|\nabla_{i,j} e| = 1$, si $i < n$ o $j < n$, o $|\nabla_{i,j} e| = 0$ en caso contrario. Entonces se cumple:

$$\mathcal{R}(s\mathbf{e}) = \sum_{i,j=1}^{n} \phi(s\,|\nabla_{i,j} e|) = (n-1)^2\phi(s) + (2n+1)\phi(0)$$

y despejando

$$\phi(s) = \frac{1}{(n-1)^2}\left(\mathcal{R}(s\mathbf{e}) - (2n+1)\phi(0)\right).$$

Deducimos de esta expresión y de la convexidad de $\mathcal{R}$:

$$\begin{aligned}
\lambda\phi(s) + (1-\lambda)\phi(t) &= \frac{1}{(n-1)^2}\left(\lambda\mathcal{R}(s\mathbf{e}) + (1-\lambda)\mathcal{R}(t\mathbf{e}) - (2n+1)\phi(0)\right) \\
&\geqslant \frac{1}{(n-1)^2}(\mathcal{R}((\lambda s + (1-\lambda)t)\mathbf{e}) - (2n+1)\phi(0)) \\
&= \phi(\lambda s + (1-\lambda)t),
\end{aligned}$$

lo que prueba la convexidad de ϕ. □

A partir de los resultados anteriores la existencia y unicidad de la solución de nuestro problema para $\phi(x) = x^p$, $p \geqslant 1$ queda establecida

Nota 5.4. Si $p > 2$ el problema está bien puesto pero los resultados no son buenos porque produce soluciones demasiado suaves, como muestran los siguientes resultados:

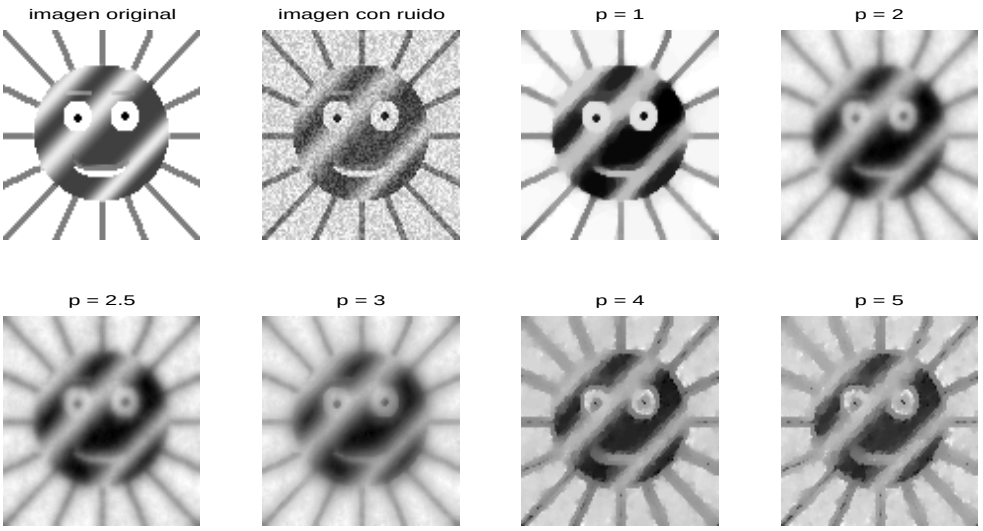


Figura 5.1: *Restauración de una imagen (100 × 100) con ruido aumentando p para la función $\phi(s) = s^p$.*

5.2 Estabilidad

En este apartado veremos que las soluciones de los problemas modificados con $\sqrt{|\nabla u|^2 + \beta}$ convergen, cuando $\beta \to 0$, a la solución del problema con $\beta = 0$. Este resultado generaliza el obtenido en [1] para la variación total.

Proposición 5.5. *Sean u_β y u_0 los minimizadores de $T_\beta(u)$ y $T_0(u) := T(u)$ respectivamente, con $\beta > 0$. Entonces*

$$\lim_{\beta \to 0} u_\beta = u_o.$$

Demostración. Como T_β es un funcional continuo por ser composición de funciones continuas, y las funciones ϕ y $\sqrt{x}$ son monótonas crecientes, es obvio que

$$\text{si } \beta_1 \leqslant \beta_2 \Rightarrow T_{\beta_1}(u) \leqslant T_{\beta_2}(u). \tag{5.3}$$

Pero además,

$$\text{si } \beta_1 \leqslant \beta_2 \Rightarrow T_{\beta_1}(u_{\beta_1}) \leqslant T_{\beta_1}(u_{\beta_2}) \leqslant T_{\beta_2}(u_{\beta_2}).$$

La primera desigualdad es cierta porque $u_{\beta_1} = \arg\min T_{\beta_1}$ y la segunda por la desigualdad (5.3). Por tanto

$$\text{si } \beta_1 \leqslant \beta_2 \Rightarrow T_{\beta_1}(u_{\beta_1}) \leqslant T_{\beta_2}(u_{\beta_2}). \tag{5.4}$$

Por otra parte, dado que $T_0(u) \geqslant \frac{1}{2}\|u - z\|_2^2$, entonces

$$(u_\beta) \to \infty\,(\beta \to 0) \Rightarrow T_0(u_\beta) \to \infty\,(\beta \to 0)$$

y, en consecuencia, $T_\beta(u_\beta) \to \infty\,(\beta \to 0)$ por (5.3). Lo cual está en contradicción con el hecho de que $T_\beta(u_\beta) \leqslant T_1(u_1), \forall \beta \leqslant 1$. Entonces $\{u_\beta/\beta \in [0,1]\}$ está acotada. Sea (β_i) una sucesión decreciente ($(\beta_i) \to 0$) tal que $u_{\beta_i} \to v$. Por tanto,

$$T_0(v) = \lim_{i\to\infty} T_0(u_{\beta_i}) \leqslant \lim_{i\to\infty} T_{\beta_i}(u_{\beta_i}) \leqslant \lim_{i\to\infty} T_{\beta_i}(u_0) = T_0(u_0),$$

pero como

$$u_0 = \arg\min(T_0)$$

y el minimizador es único, entonces:

$$\lim_{\beta\to 0} u_\beta = u_0,$$

puesto que $\{u_\beta/\beta \in [0,1]\}$ es acotado y cualquier subsucesión convergente tiene el mismo límite. □

5.3 Solución del problema discreto

Hemos visto que el problema de minimizar (5.1) está bien puesto. Vamos pues, a intentar encontrar su solución.

La primera variación del funcional T de (5.2) es $[T'(u)](v) = F'(0)$, donde $F(\varepsilon) := T(u + \varepsilon v)$:

$$F(\varepsilon) = \alpha \sum_{i,j=1}^{n} \phi\left(|\nabla_{i,j}(u + \varepsilon v)|\right) + \frac{1}{2}\sum_{i,j=1}^{n} \left(u_{i,j} + \varepsilon v_{i,j} - z_{i,j}\right)^2$$

$$F(\varepsilon) = \alpha \sum_{i,j=1}^{n} \phi\left(|\nabla_{i,j} u + \varepsilon \nabla_{i,j} v|\right) + \frac{1}{2}\sum_{i,j=1}^{n} \left(u_{i,j} - z_{i,j} + \varepsilon v_{i,j}\right)^2$$

Teniendo en cuenta que

$$\frac{d}{d\varepsilon}\sqrt{(\nabla_{i,j}u+\varepsilon\nabla_{i,j}v)\cdot(\nabla_{i,j}u+\varepsilon\nabla_{i,j}v)+\beta}=\frac{\nabla_{i,j}u\cdot\nabla_{i,j}v+\varepsilon\nabla_{i,j}v\cdot\nabla_{i,j}v}{|\nabla_{i,j}(u+\varepsilon v)|}$$

obtenemos:

$$F'(\varepsilon)=\alpha\sum_{i,j=1}^{n}\phi'\left(|\nabla_{i,j}(u+\varepsilon v)|\right)\frac{\nabla_{i,j}u\cdot\nabla_{i,j}v+\varepsilon\nabla_{i,j}v\cdot\nabla_{i,j}v}{|\nabla_{i,j}(u+\varepsilon v)|}+$$
$$+\sum_{i,j=1}^{n}\left[(u_{i,j}-z_{i,j})v_{i,j}+\varepsilon v_{i,j}^2\right]$$

$$F'(0)=\alpha\sum_{i,j=1}^{n}\phi'\left(|\nabla_{i,j}u|\right)\frac{\nabla_{i,j}u\cdot\nabla_{i,j}v}{|\nabla_{i,j}u|}+\sum_{i,j=1}^{n}(u_{i,j}-z_{i,j})v_{i,j}$$

Nos queda:

$$\left[T'(u)\right](v)=\sum_{i,j=1}^{n}\alpha\frac{\phi'\left(|\nabla_{i,j}u|\right)\nabla_{i,j}u}{|\nabla_{i,j}u|}\cdot\nabla_{i,j}v+(u_{i,j}-z_{i,j})v_{i,j}$$

$$v^TT'(u)=v^T\Big(\big(\alpha\sum_{i,j=1}^{n}\frac{\phi'\left(|\nabla_{i,j}u|\right)}{|\nabla_{i,j}u|}\nabla_{i,j}^T\nabla_{i,j}\big)u+u-z\Big)$$

$$T'(u)=\left(\alpha\sum_{i,j=1}^{n}\frac{\phi'\left(|\nabla_{i,j}u|\right)}{|\nabla_{i,j}u|}\nabla_{i,j}^T\nabla_{i,j}+I\right)u-z,$$

donde hemos usado

$$\nabla_{i,j}u\cdot\nabla_{i,j}v=(\nabla_{i,j}v)^T\nabla_{i,j}u=v^T\nabla_{i,j}^T\nabla_{i,j}u.$$

Dado que $\nabla_{i,j}^T$ es una discretización del operador $-\mathrm{div}$ evaluado en (ih,jh), la primera variación de T es una discretización de la primera variación del funcional continuo, por lo que obtenemos las ecuaciones de Euler-Lagrange, tanto del problema discreto como del problema continuo:

$$0=-\alpha\mathbf{div}\left(\frac{\phi'(|\nabla u|)}{|\nabla u|}\nabla u\right)+u-z. \tag{5.5}$$

Para obtener la segunda variación de T planteamos $T''(u)(v,v) = v^T T''(u)v = F''(0)$:

$$F''(\varepsilon) = \alpha\Big\{ \sum_{i,j=1}^{n} \phi''(|\nabla_{i,j}(u+\varepsilon v)|) \left[\frac{\nabla_{i,j}u\cdot\nabla_{i,j}v + \varepsilon\nabla_{i,j}v\cdot\nabla_{i,j}v}{|\nabla_{i,j}(u+\varepsilon v)|}\right]^2 +$$

$$+ \phi'(|\nabla_{i,j}(u+\varepsilon v)|)\frac{\nabla_{i,j}v\cdot\nabla_{i,j}v|\nabla_{i,j}(u+\varepsilon v)| - \frac{\left(\nabla_{i,j}u\cdot\nabla_{i,j}v+\varepsilon\nabla_{i,j}v\cdot\nabla_{i,j}v\right)^2}{|\nabla_{i,j}(u+\varepsilon v)|}}{|\nabla_{i,j}(u+\varepsilon v)|^2}\Big\} +$$

$$+ \sum_{i,j=1}^{n} v_{i,j}^2$$

$$F''(0) = \alpha \sum_{i,j=1}^{n} \Big\{ \phi''(|\nabla_{i,j}u|) \left[\frac{\nabla_{i,j}u\cdot\nabla_{i,j}v}{|\nabla_{i,j}u|}\right]^2 +$$

$$+ \phi'(|\nabla_{i,j}u|)\frac{\nabla_{i,j}v\cdot\nabla_{i,j}v|\nabla_{i,j}u|^2 - \left(\nabla_{i,j}u\cdot\nabla_{i,j}v\right)^2}{|\nabla_{i,j}u|^3}\Big\} + \sum_{i,j=1}^{n} v_{i,j}^2$$

$$F''(0) = \alpha \sum_{i,j=1}^{n} \Big\{ \left(\frac{\phi''(|\nabla_{i,j}u|)}{|\nabla_{i,j}u|^2} - \frac{\phi'(|\nabla_{i,j}u|)}{|\nabla_{i,j}u|^3}\right)\left(\nabla_{i,j}u\cdot\nabla_{i,j}v\right)^2 +$$

$$+ \phi'(|\nabla_{i,j}u|)\frac{\nabla_{i,j}v\cdot\nabla_{i,j}v}{|\nabla_{i,j}u|}\Big\} + \sum_{i,j=1}^{n} v_{i,j}^2$$

Usamos que $(\nabla_{i,j}u\cdot\nabla_{i,j}v)^2 = v^T(\nabla_{i,j}^T(\nabla_{i,j}u(\nabla_{i,j}u)^T)\nabla_{i,j}v$ y que $\nabla_{i,j}v\cdot\nabla_{i,j}v = v^T\nabla_{i,j}^T\nabla_{i,j}v$ para obtener:

$$F''(0) = v^T\Big[\alpha \sum_{i,j=1}^{n} \nabla_{i,j}^T\Big\{ \left(\frac{\phi''(|\nabla_{i,j}u|)}{|\nabla_{i,j}u|^2} - \frac{\phi'(|\nabla_{i,j}u|)}{|\nabla_{i,j}u|^3}\right)\nabla_{i,j}u(\nabla_{i,j}u)^T +$$

$$+ \frac{\phi'(|\nabla_{i,j}u|)}{|\nabla_{i,j}u|}I_2\Big\}\nabla_{i,j} + I\Big]v,$$

Dado que $F''(0) = v^T T''(u)v$, deducimos:

$$T''(u) = \alpha \sum_{i,j=1}^{n} \nabla_{i,j}^T \Big\{ \left(\frac{\phi''(|\nabla_{i,j}u|)}{|\nabla_{i,j}u|^2} - \frac{\phi'(|\nabla_{i,j}u|)}{|\nabla_{i,j}u|^3} \right) \nabla_{i,j}u(\nabla_{i,j}u)^T + \\ + \frac{\phi'(|\nabla_{i,j}u|)}{|\nabla_{i,j}u|} I_2 \Big\} \nabla_{i,j} + I = \alpha \sum_{i,j=1}^{n} \nabla_{i,j}^T \Big\{ \frac{\phi''(|\nabla_{i,j}u|)}{|\nabla_{i,j}u|^2} \nabla_{i,j}u(\nabla_{i,j}u)^T + \\ + \frac{\phi'(|\nabla_{i,j}u|)}{|\nabla_{i,j}u|} (I_2 - \frac{\nabla_{i,j}u(\nabla_{i,j}u)^T}{|\nabla_{i,j}u|^2}) \Big\} \nabla_{i,j} + I.$$

Como cabía esperar de la convexidad estricta de T, deducimos que $T''(u)$ es una matriz definida positiva, ya que I lo es, $\alpha > 0$, $\phi', \phi'' \geq 0$ y las matrices $\nabla_{i,j}u(\nabla_{i,j}u)^T$ y $I_2 - \frac{\nabla_{i,j}u(\nabla_{i,j}u)^T}{|\nabla_{i,j}u|^2}$ son semidefinidas positivas.

5.4

Algoritmos básicos para la supresión de ruido por variación total generalizada

En esta sección repasamos brevemente algunos algoritmos para la solución de (5.5). Son generalizaciones directas de los algoritmos correspondientes para la restauración por variación total.

5.4.1

Time Marching

Dado que (5.5) es una ecuación en derivadas parciales de tipo elíptico, su solución se puede obtener mediante la resolución del estado estacionario de la ecuación parabólica asociada:

$$u_t = \alpha \text{div} \left(\frac{\phi'(|\nabla u|_\beta)}{|\nabla u|_\beta} \nabla u \right) - (u - z), \quad \frac{\partial u}{\partial \overrightarrow{n}} = 0, u(x,0) = z(x) \qquad (5.6)$$

para la función incógnita $u = u(x,t)$. En principio, se puede aplicar un esquema de Euler explícito que nos da la iteración de tipo *time marching*:

$$u^{n+1} = u^n + \Delta t \left(\alpha \text{div} \left(\frac{\phi'(|\nabla u^n|_\beta)}{|\nabla u^n|_\beta} \nabla u^n \right) - (u^n - z) \right). \tag{5.7}$$

Este esquema simple, propuesto y usado en el trabajo original sobre la restauración de imágenes por minimización de la variación total [46], tiene la desventaja de tener severas restricciones de tipo CFL a la estabilidad, debido no sólo a su naturaleza parabólica, sinó también al hecho de que los coeficientes de difusión $\frac{\phi'(|\nabla u^n|_\beta)}{|\nabla u^n|_\beta}$ pueden ser enormes para pequeños valores del parámetro β. Se podría utilizar esquemas implícitos, pero su dificultad sería comparable a la de las alternativas que citamos a continuación y no estaría claro que fueran eficientes en la comparación.

5.4.2 Método de punto fijo

Una linealización natural de (5.5) consiste en la siguiente iteración de punto fijo:

$$-\alpha \text{div} \left(\frac{\phi'(|\nabla u^n|_\beta)}{|\nabla u^n|_\beta} \nabla u^{n+1} \right) + (u^{n+1} - z) = 0, n = 0, \ldots, \quad u^0 = z, \tag{5.8}$$

donde a cada paso se resuelve una ecuación lineal de reacción-difusión, con el coeficiente de difusión calculado en el paso previo. Este esquema generaliza el esquema de *punto fijo con difusión retrasada* presentado por Vogel y Oman en [56] para la restauración por variación total.

La discretización natural de este esquema converge a la solución de (5.5) con velocidad lineal que se deteriora hacia 1 cuando $\beta \to 0$ (ver [17] para la demostración).

5.4.3
Método de Newton primal

En principio, una mayor velocidad de convergencia se puede obtener con el método de Newton. Este método consiste en la iteración:

$$T''(u^n)(\delta u^n) = -T'(u^n), \quad u^{n+1} = u^n + \delta u^n.$$

Es un algoritmo con convergencia cuadrática (dado que podemos asegurar que $T''(u)$ es siempre invertible), pero su convergencia no está asegurada, a menos que se usen costosos procedimientos de *line search* ([34]). Pero, típicamente, estos procedimientos restringen el paso del método de forma drástica cuando β es pequeño. Se pueden utilizar *estrategias de continuación* sobre el parámetro β para tratar esta dificultad (ver [14, 15]).

6

Método de Newton primal-dual

6.1

Derivación del método

La dificultad de controlar la convergencia del método de Newton primal para el problema de la restauración de imágenes por variación total llevó a los autores de [15] a plantear un método de Newton primal-dual que resultó ser extremadamente robusto y rápido en la solución del problema. A partir de las ideas de [15], las ecuaciones de Euler-Lagrange de (5.5) se pueden reescribir con la introducción de $\frac{\nabla u}{|\nabla u|}$ como nueva variable w, de manera que nos

planteamos el sistema de ecuaciones con las incógnitas u, w:

$$-\mathbf{div}\,(\phi'(|\nabla u|)w) + \lambda(u - z) = 0, \tag{6.1}$$

$$|\nabla u|w - \nabla u = 0, \tag{6.2}$$

al cual se le aplica el método de Newton:

$$-\,\mathbf{div}\left(\phi'(|\nabla u|)\delta w + \phi''(|\nabla u|)(\frac{\nabla u}{|\nabla u|}\cdot\nabla\delta u)w\right) + \lambda(u-z) = \\ = -(-\mathbf{div}\,(\phi'(|\nabla u|w)) + \lambda(u-z)),$$

$$\frac{\nabla u\cdot\nabla\delta u}{|\nabla u|}w - \nabla\delta u + |\nabla u|\delta w = -(|\nabla u|w - \nabla u).$$

De esta última ecuación δw se puede expresar en términos de δu:

$$\delta w = -(1 + \frac{\nabla u\cdot\nabla\delta u}{|\nabla u|^2})w + \frac{\nabla u}{|\nabla u|} + \frac{\nabla\delta u}{|\nabla u|}, \tag{6.3}$$

el cual, cuando se sustituye en la ecuación anterior, da:

$$-\mathbf{div}\,(K(u,w)\nabla\delta u) + \lambda\delta u = \mathbf{div}\left(\frac{\phi'(|\nabla u|)}{|\nabla u|}\nabla u\right) - \lambda(u-z),$$

$$K(u,w)_{i,j} = \frac{\phi'(|\nabla u_{i,j}|)}{|\nabla u_{i,j}|}I_2 + \\ + \left(\phi''(|\nabla u_{i,j}|) - \frac{\phi'(|\nabla u_{i,j}|)}{|\nabla u_{i,j}|}\right)w_{i,j}\left(\frac{\nabla_{i,j}u}{|\nabla_{i,j}u|_\beta}\right)^T.$$

Debido a la presencia del término $w_{i,j}\left(\frac{\nabla u_{i,j}}{|\nabla_{i,j}u|_\beta}\right)^T$, las matrices 2×2 $K(u,w)_{i,j}$ son simétricas sólo cuando $w_{i,j} = \frac{\nabla_{i,j}u}{|\nabla_{i,j}u|}$. Desde luego, el método de Newton primal se recupera con esta elección. El punto clave consiste en sustituir $K(u,w)$ por su simetrización (omitimos los índices i, j):

$$\widetilde{K}(u,w) = \frac{1}{2}(K(u,w) + K(u,w)^T) \\ = \frac{\phi'(|\nabla u|)}{|\nabla u|}I_2 + \frac{1}{2}\left(\phi''(|\nabla u|) - \frac{\phi'(|\nabla u|)}{|\nabla u|}\right)\left(w\left(\frac{\nabla u}{|\nabla u|}\right)^T + \frac{\nabla u}{|\nabla u|}w^T\right).$$

El método de Newton *inexacto* resultante puede usar métodos numéricos para la solución de los sistemas lineales que aparecen en cada iteración, sin sacrificar la convergencia cuadrática del método de Newton original (ver [34, chap. 5 and 6]), ya que la matriz

$$-\operatorname{div}\left(\widetilde{K}(u^*, w^*)\nabla\right) + \lambda I \tag{6.4}$$

coincide con $F''(u^*)$ en la solución (u^*, w^*), para la cual $w^* = \frac{\nabla u^*}{|\nabla u^*|}$. La ventaja de esta transformación es que se pueden utilizar métodos iterativos muy eficientes (gradiente conjugado o symmlq [42]) para las matrices dispersas y simétricas que aparecen a lo largo de la iteración.

El método de Newton primal-dual que proponemos como extensión del propuesto en [15] queda:

$$-\operatorname{div}\left(\widetilde{K}(u^n, w^n)\nabla\delta u^n\right) + \lambda\delta u^n = \operatorname{div}\left(\frac{\phi'(|\nabla u^n|)}{|\nabla u^n|}\nabla u^n\right) - \lambda(u^n - z) \tag{6.5}$$

$$\delta w^n = -(1 + \frac{\nabla u^n \cdot \nabla\delta u^n}{|\nabla u^n|^2})w^n + \frac{\nabla u^n}{|\nabla u|^n} + \frac{\nabla\delta u^n}{|\nabla u^n|}, \tag{6.6}$$

$$u^{n+1} = u^n + \delta u^n, \quad w^{n+1} = w^n + s\delta w^n, \tag{6.7}$$

donde $u^0 = z$, $w^0 = 0$ y s se calcula para asegurar $|w^{n+1}|_\infty \leq 1$ de manera eficiente, calculando:

$$s = \min(0{,}95 \min\{s_i, \delta w_i \neq 0\}, 1),$$
$$s_i = \frac{-w_i \cdot \delta w_i + \sqrt{(w_i \cdot \delta w_i)^2 + (1 - w_i \cdot w_i)(\delta w_i \cdot \delta w_i)}}{\delta w_i \cdot \delta w_i + 10^{-16}}.$$

6.2 Experimentos numéricos

En esta sección comparamos la velocidad de convergencia del método de punto fijo y el método de Newton primal-dual para algunas funciones $\phi(x)$.

Dado que estamos comprobando la convergencia total, es decir, que el llamado *residual no lineal* (el término de la derecha de

(5.5) o (6.5)) esté cercano a anularse, el criterio de parada para la iteración (exterior) de Newton es que se produzca un decrecimiento relativo de este residual no lineal por un factor de 10^{-8}. Usamos un método de Newton *truncado*, basado en el método del gradiente conjugado con factorización de Cholesky incompleta con un nivel de llenado 5 ([24]). El criterio de parada para la n-ésima solución iterativa es que se produzca una disminución relativa del residual lineal por un factor de η_n, donde, siguiendo la sugerencia de [34, Eq. 6.18] para conseguir convergencia cuadrática de forma eficiente, asignamos:

$$\eta_n = \begin{cases} 0{,}01 \text{ if } n = 0, \\ \text{mín}(0{,}01, 0{,}6||g_n||^2/||g_{n-1}||^2) \end{cases} \tag{6.8}$$

donde g_n denota el residual no lineal. Mencionamos aquí que, aunque sólo podemos asegurar la definición positiva de la matriz de (6.5) para $n = 0$ (para el cual $w = 0$ y la matriz es ciertamente definida positiva) o cerca del mínimo u^* (en el cual coincide con $F''(u^*)$, la cual es definida positiva), la convergencia del método del gradiente conjugado ha sido muy satisfactoria en todos los casos, requiriendo muy pocas iteraciones al principio de la iteración exterior y algunos más cuando el parámetro η_n decrece cuando la convergencia total está a punto de obtenerse.

Para la iteración de punto fijo, usamos el mismo resolvedor lineal con una tolerancia relativa sobre el residual fijada a $0{,}01$, ya que una tolerancia mayor no variaría el resultado, pero sí incrementaría el coste computacional.

Para la comparación de ambos métodos usamos las imágenes Lena, Peppers y Barbara de la figura 2.12, a las que se les ha añadido ruido blanco gaussiano, resultando en SNR respectivas de 2, 2 y 5. Realizamos los experimentos con la función ϕ de Huber, que se usa en estimación robusta ([31]):

$$\phi(x) = \begin{cases} x^2 & x < \xi \\ 2\xi x - \xi^2 & x \geq \xi. \end{cases} \tag{6.9}$$

Esta función es convexa y continuamente diferenciable, pero la segunda derivada es discontinua en ξ. En este punto especificamos

$\phi''(\xi) = \phi''(\xi^+) = 0$. Esta función está diseñada para penalizar pequeños saltos más que el funcional de la variación total, como se prescribe en [41] para tratar el efecto escalera.

Figura 6.1: *Ampliaciones de parte de la imagen original (izquierda) y de la imagen restaurada (derecha) con la función $\phi(x) = x$ (restauración por variación total) con $\beta = 1$*

La imagen de la izquierda de las figuras 6.1 (Lena), 6.5 (Peppers), 6.9 (Barbara) muestra una zona de la imagen original. Esta zona contiene partes suaves (aunque en la imagen original ya contienen algo de ruido) y perfiles. Nos proponemos ejecutar el método que proponemos y el método del punto fijo para la función de Huber ϕ con varios parámetros umbral ξ y comparar su resultado (común) con el de la supresión de ruido por minimización de la variación total, el cual se muestra a la derecha en las figuras 6.1 (Lena), 6.5 (Peppers), 6.9 (Barbara). También comparamos la historia de la convergencia de ambos métodos en términos del residual no lineal con respecto a iteraciones y tiempo de CPU. El parámetro λ se ha calculado para asegurar, en cada caso, la restricción:

$$\int_\Omega (u(x) - z(x))^2 \, dx = \sigma^2.$$

Los parámetros umbral que hemos elegido son $\xi = 1000$ (resultados en las Figuras 6.2 para el test Lena, 6.6 para el test Peppers,

Figura 6.2: *Imagen restaurada con la función de Huber ϕ con $\xi = 1000$ y ampliación de parte de la imagen restaurada.*

6.10 para el test Barbara), $\xi = 500$ (resultados en las Figuras 6.3 para el test Lena, 6.7 para el test Peppers, 6.11 para el test Barbara) y $\xi = 100$ (resultados en las Figuras 6.4 para el test Lena, 6.8 para el test Peppers, 6.12 para el test Barbara). Estos resultados muestran perfiles tan bien definidos como los obtenidos con el funcional variación total. Como era de esperar, a mayor umbral ξ, menor efecto escalera en las imágenes restauradas.

En los experimentos analizamos la influencia del parámetro β en la velocidad de convergencia del algoritmo. Ejecutamos ambos algoritmos para las funciones $\phi(x)$ mencionadas anteriormente y $\beta = 1, 10^{-5}, 10^{-10}$ y mostramos en Fig. 6.13-6.21 (Lena), Fig. 6.22-6.30 (Peppers), Fig. 6.31-6.39 (Barbara), la historia de la convergencia en términos del residual no lineal con respecto a iteraciones (externas) y tiempo de CPU, con escala vertical logarítmica. En estas gráficas los resultados para el método primal-dual se representan con $\circ$ y los del método de punto fijo con $+$.

De estas gráficas deducimos la convergencia cuadrática del método de Newton primal-dual (la gráfica del logaritmo del residual no lineal con respecto a iteraciones adquiere forma de exponencial invertida a partir de una cierta iteración) y la convergencia lineal del método del punto fijo (la gráfica del logaritmo del residual no li-

Figura 6.3: *Imagen restaurada con la función de Huber ϕ con $\xi = 500$ y ampliación de parte de la imagen restaurada.*

Figura 6.4: *Imagen restaurada con la función de Huber ϕ con $\xi = 100$ y ampliación de parte de la imagen restaurada.*

neal con respecto a iteraciones adquiere forma de recta a partir de una cierta iteración) y que los gradientes de la función objetivo en cada iteración del método primal-dual son inferiores (mejor resultado) a los del método de punto fijo, es decir, que a cada iteración el método de Newton primal-dual obtiene mejor aproximación a la

Figura 6.5: *Ampliaciones de parte de la imagen original (izquierda) y de la imagen restaurada (derecha) con la función* $\phi(x) = x$ *(restauración por variación total) con* $\beta = 1$

Figura 6.6: *Imagen restaurada con la función de Huber* ϕ *con* $\xi = 1000$ *y ampliación de parte de la imagen restaurada.*

solución (el residual no lineal es menor) que el método de punto fijo. Con respecto a la eficiencia, las figuras que muestran el residual no lineal con respecto al tiempo de CPU muestran que el método del punto fijo es ligeramente más eficiente que el método

Figura 6.7: *Imagen restaurada con la función de Huber ϕ con $\xi = 500$ y ampliación de parte de la imagen restaurada.*

Figura 6.8: *Imagen restaurada con la función de Huber ϕ con $\xi = 100$ y ampliación de parte de la imagen restaurada.*

primal-dual en las primeros iteraciones, debido al hecho de que el coste por iteración del resolvedor lineal es aproximadamente 30 % mayor para éste último.

Finalmente, en las tablas 6.1-6.9, en las que se muestra el número final de iteraciones y el tiempo de CPU en cada caso, la dife-

Figura 6.9: *Ampliaciones de parte de la imagen original (izquierda) y de la imagen restaurada (derecha) con la función $\phi(x) = x$ (restauración por variación total) con $\beta = 1$*

Figura 6.10: *Imagen restaurada con la función de Huber ϕ con $\xi = 1000$ y ampliación de parte de la imagen restaurada.*

rencia entre el coste computacional del método de punto fijo con respecto al primal dual crece cuando se reduce β.

Figura 6.11: *Imagen restaurada con la función de Huber ϕ con $\xi = 500$ y ampliación de parte de la imagen restaurada.*

Figura 6.12: *Imagen restaurada con la función de Huber ϕ con $\xi = 100$ y ampliación de parte de la imagen restaurada.*

	primal-dual		punto fijo	
β	iteraciones	tiempo de CPU	iteraciones	tiempo de CPU
1	28	63,2	1000	1149,7
10^{-5}	33	72,4	1000	1153,3
10^{-10}	33	71,9	1000	1151,3

Tabla 6.1: *Número de iteraciones y tiempo de CPU para el test con la imagen Lena y la función de Huber para* $\xi = 100$ *y* $\beta = 1, 10^{-5}, 10^{-10}$*. El método de punto fijo no ha convergido en* 1000 *iteraciones.*

	primal-dual		punto fijo	
β	iteraciones	tiempo de CPU	iteraciones	tiempo de CPU
1	24	51,2	572	655,7
10^{-5}	26	55,9	572	656,5
10^{-10}	26	55,5	572	656,8

Tabla 6.2: *Número de iteraciones y tiempo de CPU para el test con la imagen Lena y la función de Huber para* $\xi = 500$ *y* $\beta = 1, 10^{-5}, 10^{-10}$*.*

	primal-dual		punto fijo	
β	iteraciones	tiempo de CPU	iteraciones	tiempo de CPU
1	22	47,3	372	424,2
10^{-5}	23	49,1	373	427,8
10^{-10}	23	49,0	373	432,5

Tabla 6.3: *Número de iteraciones y tiempo de CPU para el test con la imagen Lena y la función de Huber para* $\xi = 1000$ *y* $\beta = 1, 10^{-5}, 10^{-10}$*.*

β	primal-dual		punto fijo	
	iteraciones	tiempo de CPU	iteraciones	tiempo de CPU
1	30	72,4	1000	1306,3
10^{-5}	33	80,8	1000	1315,8
10^{-10}	33	80,5	1000	1206,8

Tabla 6.4: *Número de iteraciones y tiempo de CPU para el test con la imagen Peppers y la función de Huber para* $\xi = 100$ *y* $\beta = 1, 10^{-5}, 10^{-10}$*. El método de punto fijo no ha convergido en* 1000 *iteraciones.*

β	primal-dual		punto fijo	
	iteraciones	tiempo de CPU	iteraciones	tiempo de CPU
1	23	54,0	683	781,7
10^{-5}	26	60,3	683	781,6
10^{-10}	26	56,5	683	775,2

Tabla 6.5: *Número de iteraciones y tiempo de CPU para el test con la imagen Peppers y la función de Huber para* $\xi = 500$ *y* $\beta = 1, 10^{-5}, 10^{-10}$*.*

β	primal-dual		punto fijo	
	iteraciones	tiempo de CPU	iteraciones	tiempo de CPU
1	21	45,6	435	491,1
10^{-5}	22	46,8	435	492,3
10^{-10}	22	47,5	435	492,7

Tabla 6.6: *Número de iteraciones y tiempo de CPU para el test con la imagen Peppers y la función de Huber para* $\xi = 1000$ *y* $\beta = 1, 10^{-5}, 10^{-10}$*.*

β	primal-dual		punto fijo	
	iteraciones	tiempo de CPU	iteraciones	tiempo de CPU
1	25	37,5	864	729,7
10^{-5}	28	43,9	813	695,6
10^{-10}	29	43,3	813	694,7

Tabla 6.7: *Número de iteraciones y tiempo de CPU para el test con la imagen Barbara y la función de Huber para* $\xi = 100$ *y* $\beta = 1, 10^{-5}, 10^{-10}$.

β	primal-dual		punto fijo	
	iteraciones	tiempo de CPU	iteraciones	tiempo de CPU
1	20	29,8	233	200,7
10^{-5}	23	34,3	233	198,2
10^{-10}	23	34,1	233	195,2

Tabla 6.8: *Número de iteraciones y tiempo de CPU para el test con la imagen Barbara y la función de Huber para* $\xi = 500$ *y* $\beta = 1, 10^{-5}, 10^{-10}$.

β	primal-dual		punto fijo	
	iteraciones	tiempo de CPU	iteraciones	tiempo de CPU
1	19	27,9	146	121,2
10^{-5}	20	29,6	146	118,7
10^{-10}	20	29,8	146	118,2

Tabla 6.9: *Número de iteraciones y tiempo de CPU para el test con la imagen Barbara y la función de Huber para* $\xi = 1000$ *y* $\beta = 1, 10^{-5}, 10^{-10}$.

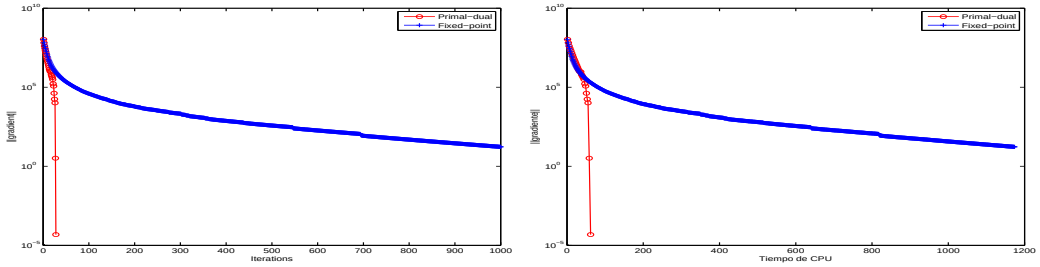
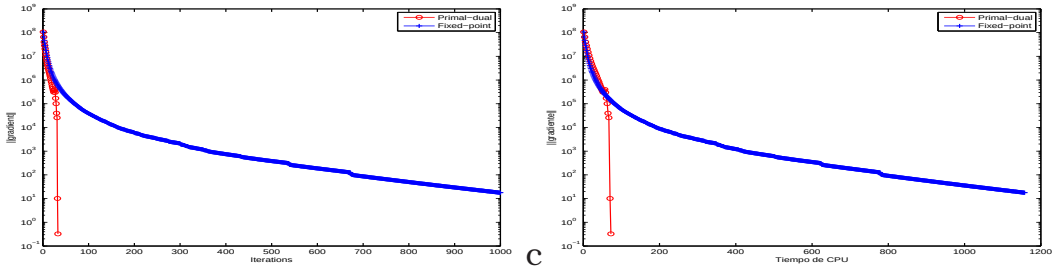


Figura 6.13: *Historia de la convergencia para la imagen Lena función de Huber ϕ con $\xi = 100$ y $\beta = 1{,}000000e+00$ y $\lambda = 3{,}286280e+03$: izquierda, residual no lineal con respecto a iteraciones; derecha, residual no lineal con respecto a tiempo de CPU. Los resultados para el método primal-dual se representan con $\circ$ y los del método de punto fijo con $+$.*



c

Figura 6.14: *Historia de la convergencia para la imagen Lena función de Huber ϕ con $\xi = 100$ y $\beta = 1{,}000000e-05$ y $\lambda = 3{,}286280e+03$: izquierda, residual no lineal con respecto a iteraciones; derecha, residual no lineal con respecto a tiempo de CPU.*

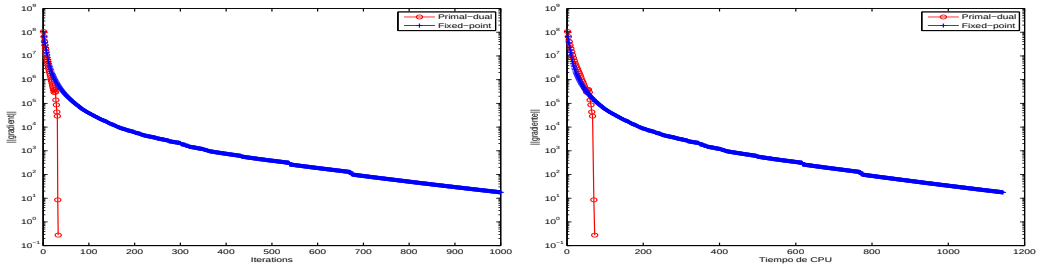


Figura 6.15: *Historia de la convergencia para la imagen Lena función de Huber ϕ con $\xi = 100$ y $\beta = 1{,}000000e-10$ y $\lambda = 3{,}286280e+03$: izquierda, residual no lineal con respecto a iteraciones; derecha, residual no lineal con respecto a tiempo de CPU.*

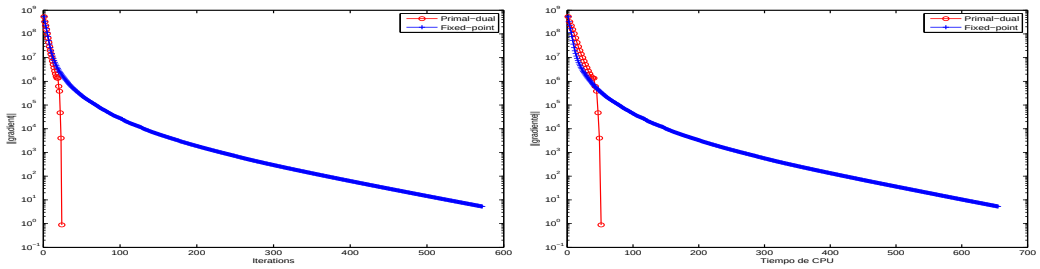


Figura 6.16: *Historia de la convergencia para la imagen Lena función de Huber ϕ con $\xi = 500$ y $\beta = 1{,}000000e+00$ y $\lambda = 1{,}590630e+04$: izquierda, residual no lineal con respecto a iteraciones; derecha, residual no lineal con respecto a tiempo de CPU.*

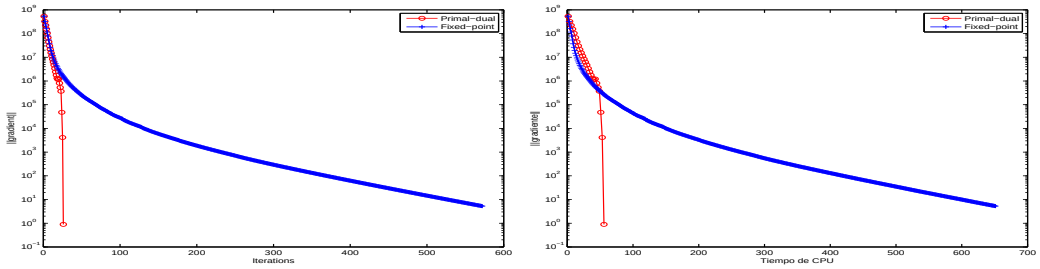


Figura 6.17: *Historia de la convergencia para la imagen Lena función de Huber ϕ con $\xi = 500$ y $\beta = 1{,}000000e-05$ y $\lambda = 1{,}590630e+04$: izquierda, residual no lineal con respecto a iteraciones; derecha, residual no lineal con respecto a tiempo de CPU.*

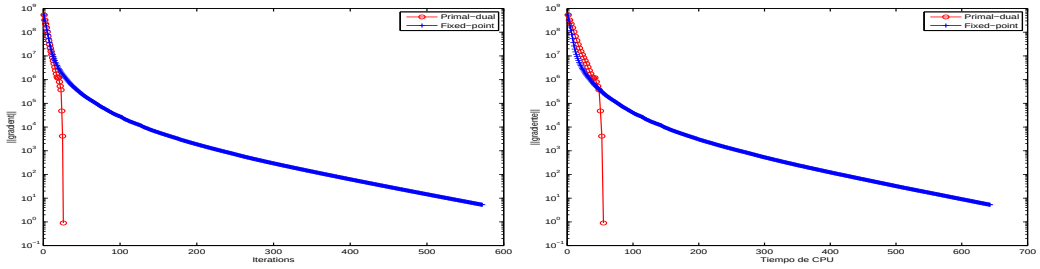


Figura 6.18: *Historia de la convergencia para la imagen Lena función de Huber ϕ con $\xi = 500$ y $\beta = 1{,}000000e-10$ y $\lambda = 1{,}590630e+04$: izquierda, residual no lineal con respecto a iteraciones; derecha, residual no lineal con respecto a tiempo de CPU.*

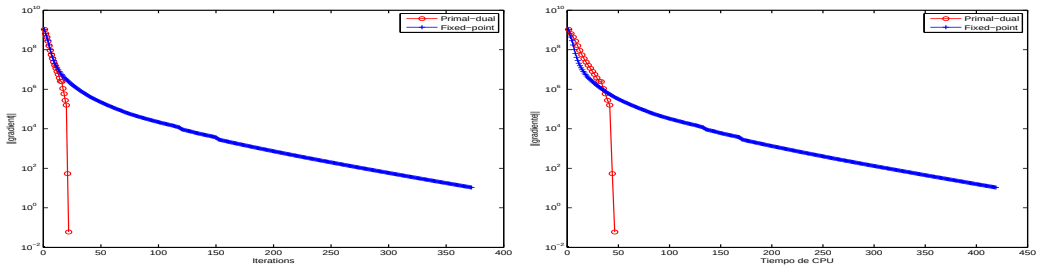


Figura 6.19: *Historia de la convergencia para la imagen Lena función de Huber ϕ con $\xi = 1000$ y $\beta = 1{,}000000e+00$ y $\lambda = 3{,}004967e+04$: izquierda, residual no lineal con respecto a iteraciones; derecha, residual no lineal con respecto a tiempo de CPU.*

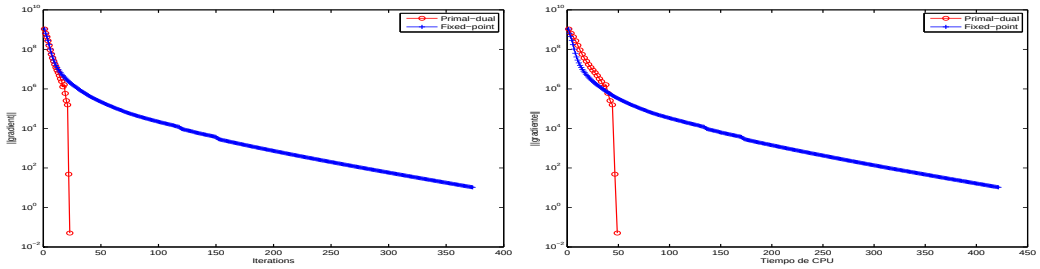


Figura 6.20: *Historia de la convergencia para la imagen Lena función de Huber ϕ con $\xi = 1000$ y $\beta = 1{,}000000e-05$ y $\lambda = 3{,}004967e+04$: izquierda, residual no lineal con respecto a iteraciones; derecha, residual no lineal con respecto a tiempo de CPU.*

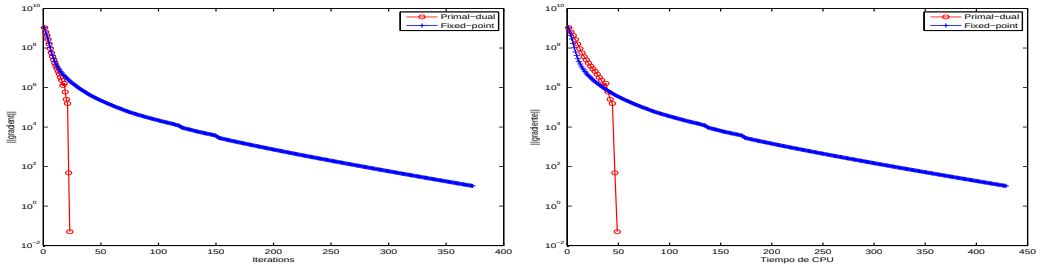


Figura 6.21: *Historia de la convergencia para la imagen Lena función de Huber ϕ con $\xi = 1000$ y $\beta = 1{,}000000e-10$ y $\lambda = 3{,}004967e+04$: izquierda, residual no lineal con respecto a iteraciones; derecha, residual no lineal con respecto a tiempo de CPU.*

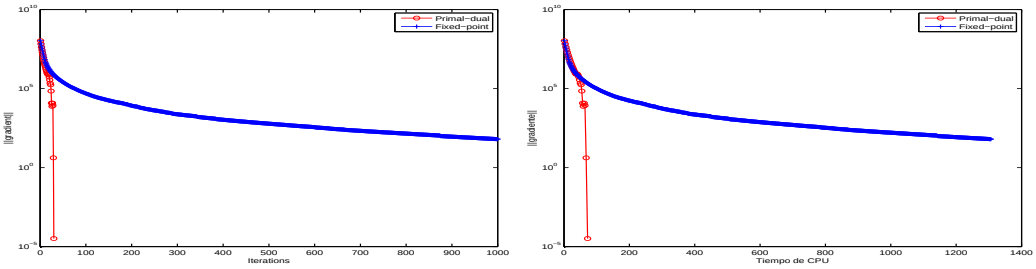


Figura 6.22: *Historia de la convergencia para la imagen Peppers función de Huber ϕ con $\xi = 100$ y $\beta = 1{,}000000e+00$ y $\lambda = 2{,}349032e+03$: izquierda, residual no lineal con respecto a iteraciones; derecha, residual no lineal con respecto a tiempo de CPU.*

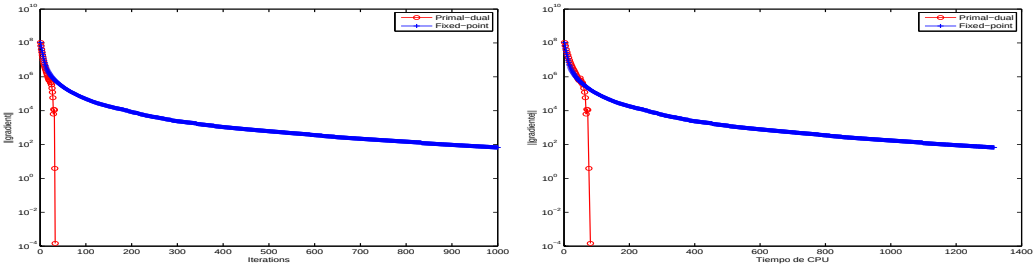


Figura 6.23: *Historia de la convergencia para la imagen Peppers función de Huber ϕ con $\xi = 100$ y $\beta = 1{,}000000e-05$ y $\lambda = 2{,}349032e+03$: izquierda, residual no lineal con respecto a iteraciones; derecha, residual no lineal con respecto a tiempo de CPU.*

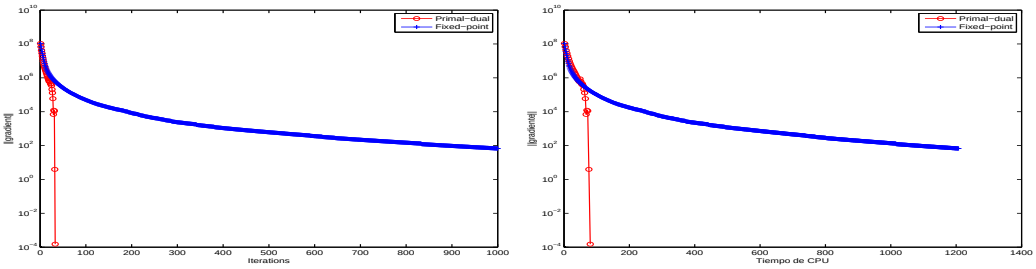


Figura 6.24: *Historia de la convergencia para la imagen Peppers función de Huber ϕ con $\xi = 100$ y $\beta = 1{,}000000e-10$ y $\lambda = 2{,}349032e+03$: izquierda, residual no lineal con respecto a iteraciones; derecha, residual no lineal con respecto a tiempo de CPU.*

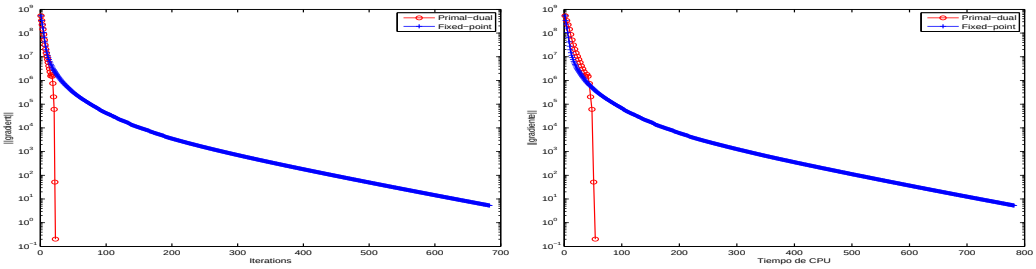


Figura 6.25: *Historia de la convergencia para la imagen Peppers función de Huber ϕ con $\xi = 500$ y $\beta = 1{,}000000e+00$ y $\lambda = 1{,}152935e+04$: izquierda, residual no lineal con respecto a iteraciones; derecha, residual no lineal con respecto a tiempo de CPU.*

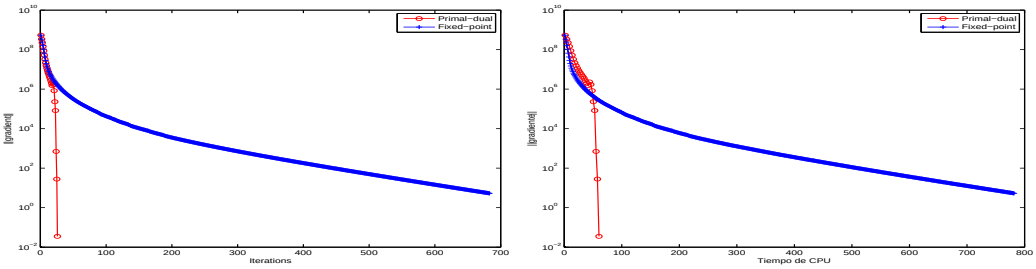


Figura 6.26: *Historia de la convergencia para la imagen Peppers función de Huber ϕ con $\xi = 500$ y $\beta = 1{,}000000e-05$ y $\lambda = 1{,}152935e+04$: izquierda, residual no lineal con respecto a iteraciones; derecha, residual no lineal con respecto a tiempo de CPU.*

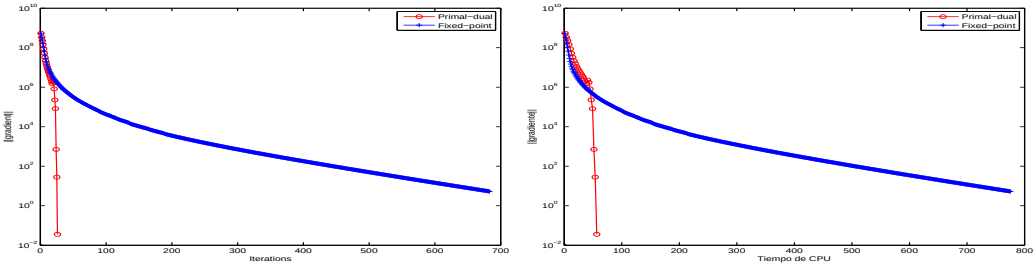


Figura 6.27: *Historia de la convergencia para la imagen Peppers función de Huber ϕ con $\xi = 500$ y $\beta = 1{,}000000e-10$ y $\lambda = 1{,}152935e+04$: izquierda, residual no lineal con respecto a iteraciones; derecha, residual no lineal con respecto a tiempo de CPU.*

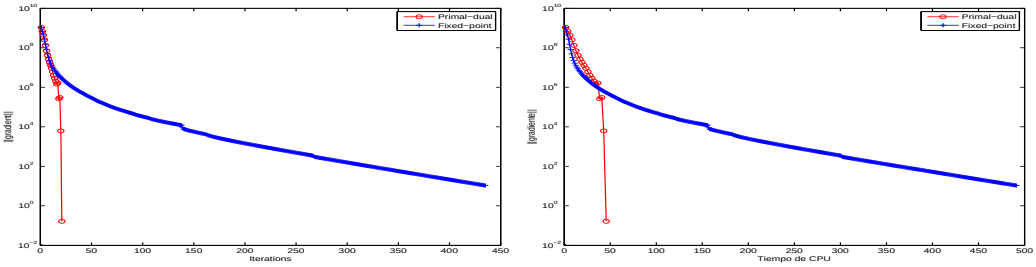


Figura 6.28: *Historia de la convergencia para la imagen Peppers función de Huber ϕ con $\xi = 1000$ y $\beta = 1{,}000000e+00$ y $\lambda = 2{,}210107e+04$: izquierda, residual no lineal con respecto a iteraciones; derecha, residual no lineal con respecto a tiempo de CPU.*

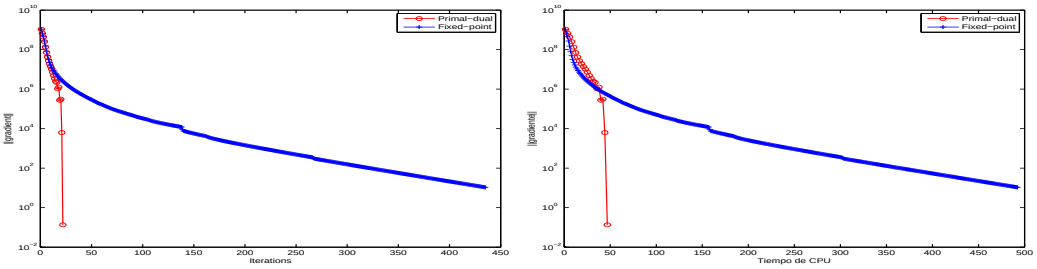


Figura 6.29: *Historia de la convergencia para la imagen Peppers función de Huber ϕ con $\xi = 1000$ y $\beta = 1{,}000000e-05$ y $\lambda = 2{,}210107e+04$: izquierda, residual no lineal con respecto a iteraciones; derecha, residual no lineal con respecto a tiempo de CPU.*

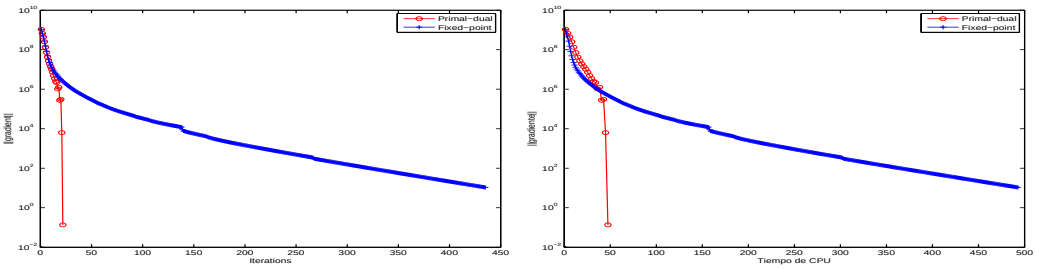


Figura 6.30: *Historia de la convergencia para la imagen Peppers función de Huber ϕ con $\xi = 1000$ y $\beta = 1{,}000000e-10$ y $\lambda = 2{,}210107e+04$: izquierda, residual no lineal con respecto a iteraciones; derecha, residual no lineal con respecto a tiempo de CPU.*

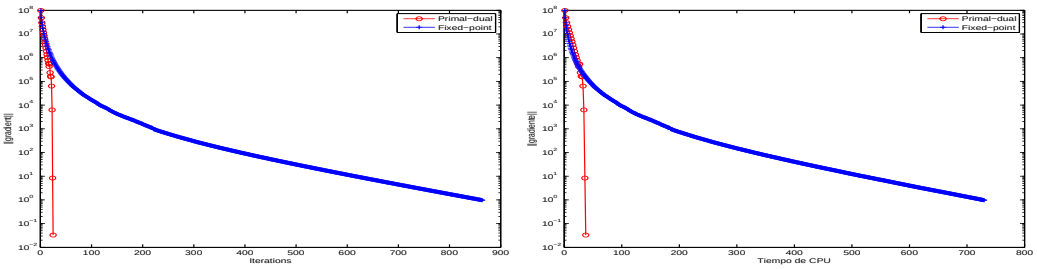


Figura 6.31: *Historia de la convergencia para la imagen Barbara función de Huber ϕ con $\xi = 100$ y $\beta = 1{,}000000e+00$ y $\lambda = 1{,}180436e+04$: izquierda, residual no lineal con respecto a iteraciones; derecha, residual no lineal con respecto a tiempo de CPU.*

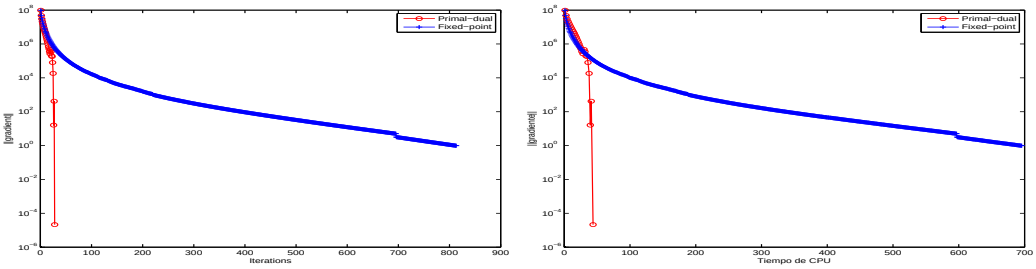


Figura 6.32: *Historia de la convergencia para la imagen Barbara función de Huber ϕ con $\xi = 100$ y $\beta = 1{,}000000e-05$ y $\lambda = 1{,}180436e+04$: izquierda, residual no lineal con respecto a iteraciones; derecha, residual no lineal con respecto a tiempo de CPU.*

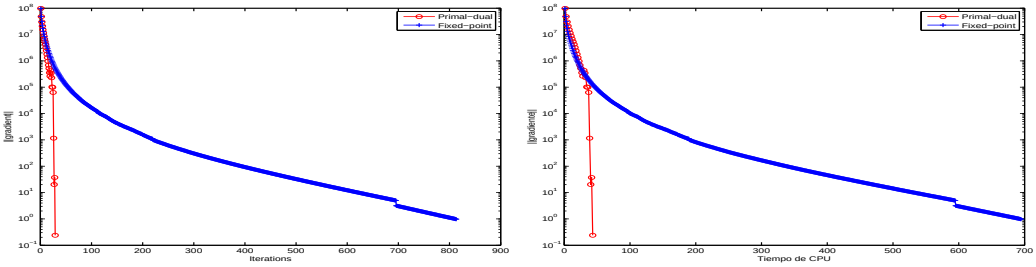


Figura 6.33: *Historia de la convergencia para la imagen Barbara función de Huber ϕ con $\xi = 100$ y $\beta = 1{,}000000e-10$ y $\lambda = 1{,}180436e+04$: izquierda, residual no lineal con respecto a iteraciones; derecha, residual no lineal con respecto a tiempo de CPU.*

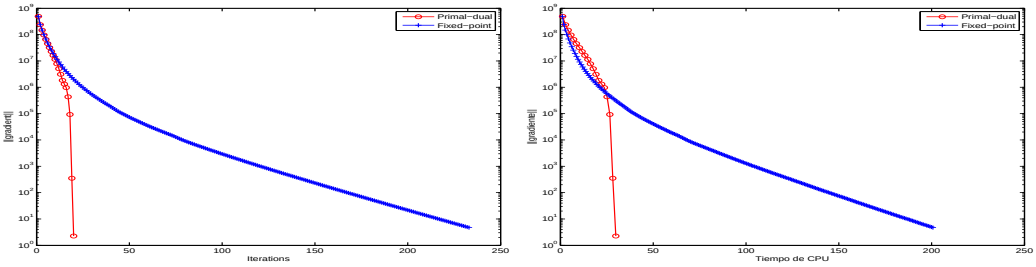


Figura 6.34: *Historia de la convergencia para la imagen Barbara función de Huber ϕ con $\xi = 500$ y $\beta = 1{,}000000e+00$ y $\lambda = 5{,}849835e+04$: izquierda, residual no lineal con respecto a iteraciones; derecha, residual no lineal con respecto a tiempo de CPU.*

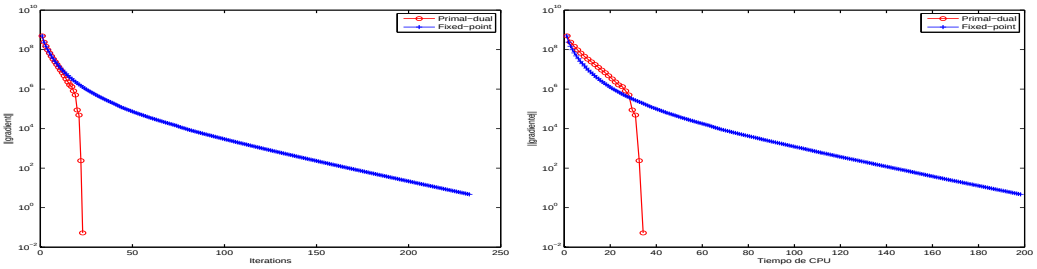


Figura 6.35: *Historia de la convergencia para la imagen Barbara función de Huber ϕ con $\xi = 500$ y $\beta = 1{,}000000e-05$ y $\lambda = 5{,}849835e+04$: izquierda, residual no lineal con respecto a iteraciones; derecha, residual no lineal con respecto a tiempo de CPU.*

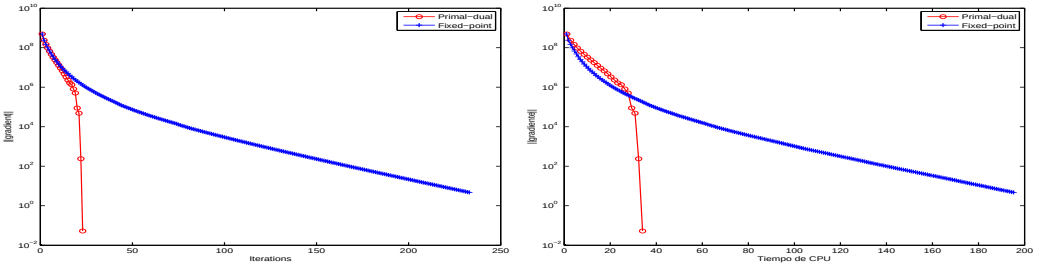


Figura 6.36: *Historia de la convergencia para la imagen Barbara función de Huber ϕ con $\xi = 500$ y $\beta = 1{,}000000e-10$ y $\lambda = 5{,}849835e+04$: izquierda, residual no lineal con respecto a iteraciones; derecha, residual no lineal con respecto a tiempo de CPU.*

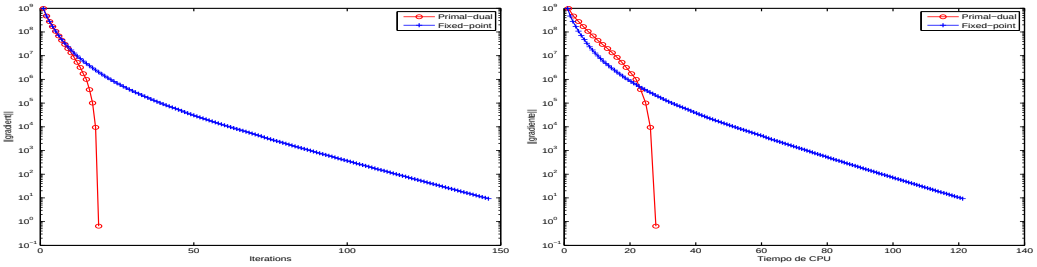


Figura 6.37: *Historia de la convergencia para la imagen Barbara función de Huber ϕ con $\xi = 1000$ y $\beta = 1{,}000000e+00$ y $\lambda = 1{,}153023e+05$: izquierda, residual no lineal con respecto a iteraciones; derecha, residual no lineal con respecto a tiempo de CPU.*

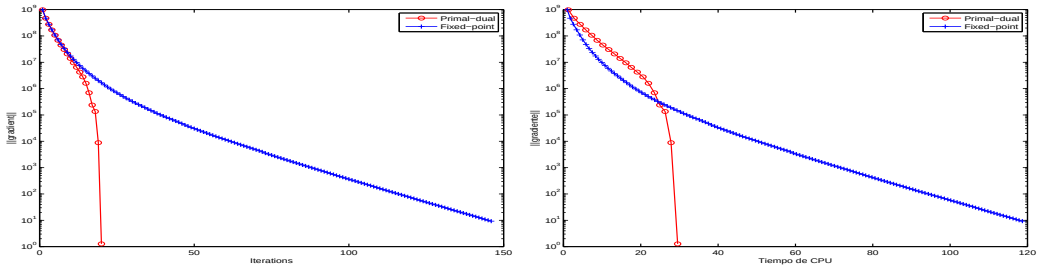


Figura 6.38: *Historia de la convergencia para la imagen Barbara función de Huber* ϕ *con* $\xi = 1000$ *y* $\beta = 1{,}000000e-05$ *y* $\lambda = 1{,}153023e+05$*: izquierda, residual no lineal con respecto a iteraciones; derecha, residual no lineal con respecto a tiempo de CPU.*

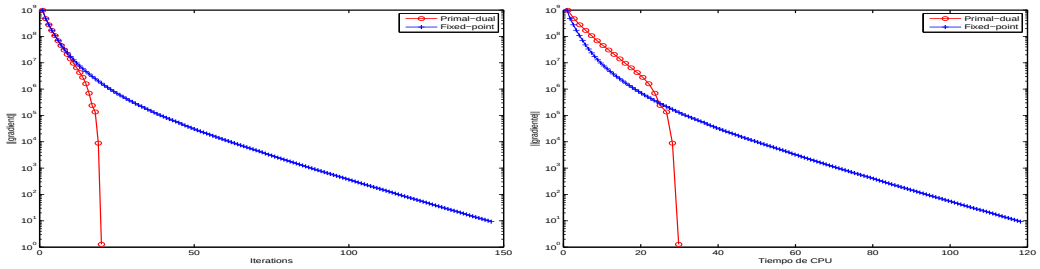


Figura 6.39: *Historia de la convergencia para la imagen Barbara función de Huber* ϕ *con* $\xi = 1000$ *y* $\beta = 1{,}000000e-10$ *y* $\lambda = 1{,}153023e+05$*: izquierda, residual no lineal con respecto a iteraciones; derecha, residual no lineal con respecto a tiempo de CPU.*

7

Conclusiones y perspectivas

En esta tesis se han presentado unos algoritmos 2D no separables en el marco de la multirresolución de Harten que nos dan una determinada estimación del error entre la imagen original y la imagen decodificada medido en las normas discretas L_∞ y L_2. Con esto, una vez que el parámetro de cuantización se fija, tenemos cotas *a priori* del error. Por otra parte, no es necesario aplicar la transformada inversa para conocer el *error exacto*.

Hemos demostrado que este algoritmo puede ser usado para obtener codificadores de imágenes sin pérdidas y casi sin pérdida.

En los experimentos numéricos hemos demostrado que fijando el parámetro ε tenemos una cota *a priori* del PSNR y que conocemos con exactitud el PAE. Hemos visto que el PSNR real es mayor que esta cota inferior PSNR. Se demuestra que, con diferentes estrategias de cuantificación la PSNR obtenida está más cerca de la

cota inferior del PSNR. Por otra parte, en todos los casos, el PSNR puede ser evaluado con el codificador.

En los experimentos numéricos hemos demostrado que, a pesar de que nuestra técnica de compresión es poco sofisticada, las tasas de compresión obtenida con los algoritmos del **EC** y el **JPEG-LS** están muy cerca. También podemos ver que para grandes ε nuestra algoritmo supera al algoritmo casi sin pérdidas **JPEG-LS**.

Si comparamos nuestro algoritmo con los resultados obtenidos con el **JPEG-2000** observamos que con nuestro algoritmo, para un determinado PAE, tenemos más compresión.

Dos formas de mejorar las tasas de compresión obtenidas con nuestro codificador serían las siguientes: en primer lugar, el uso de técnicas de interpolación no lineales con el fin de reducir la entropía de la transformada y por tanto, obtener mejores tasas de compresión; en segundo lugar, esperamos que, con un codificador entrópico adaptado a los datos que tenemos los resultados serán mejores.

Una ventaja de nuestro método es que su sencillez nos da la posibilidad de utilizar diferentes estrategias (sin pérdida, casi sin pérdida) en diferentes regiones obteniendo los algoritmos de compresión con Regiones de Interés (ROI).

Hemos desarrollado un algoritmo similar en el contexto de medias en celda que es más adecuado para trabajar con imágenes. En este caso también obtenemos unos resultados gracias a los cuales podemos conocer el error exacto entre la imagen original y la imagen reconstruida sin tener que calcular ésta (sin decodificar la imagen).

También hemos presentado un algoritmo de interpolación 2D basado en técnicas no lineales WENO. Presentamos unos resultados teóricos que luego se comprueban en los experimentos numéricos.

La restauración de imágenes por minimización de la variación total es una técnica fiable para recuperar perfiles acusados, pero tiende típicamente a agudizar excesivamente transiciones suaves entre niveles de gris, convirtiéndolas en funciones constantes a trozos (efecto escalera) En la segunda parte de esta memoria hemos propuesto un método para resolver problemas que generalizan la minimización de la variación total, propuestos para aliviar

este efecto escalera, de una forma rápida y robusta. Hemos mostrado que este método es áltamente competitivo con respecto al método del punto fijo con difusión retrasada, como demuestran los experimentos incluidos. En esta línea se podría plantear la investigación futura para funciones ϕ que no sean convexas, lo cual requerirá replantear los métodos numéricos para la solución de los sistemas lineales.

A

Descripción de los algoritmos EZW y SPIHT

A.1

Algoritmo EZW

En esta sección explicaremos la implementación del codificador EZW (Embedded Zerotree Wavelet) presentado por J.Shapiro en su trabajo [50] de 1993. EZW es un codificador especialmente creado para ser utilizado con la transformada wavelet y que, aunque fue en su origen diseñado para imágenes (señales 2D), también puede ser utilizado con señales de otras dimensiones.

EZW está basado en la *codificación progresiva* para comprimir una imagen en una secuencia de bits con precisión creciente. Esto significa que cuantos más bits son añadidos a la cadena, la imagen

descodificada tendrá más detalles. La codificación progresiva es también conocida por *codificación incrustada* (*embedded coding*) y genera los bits en la secuencia-código en orden de importancia.

Al codificar una imagen usando el esquema del EZW se consigue una muy buena compresión de imágenes con la propiedad que la secuencia de datos comprimidos puede tener cualquier tasa de compresión deseada. Esto sólo es posible si hay alguna pérdida de información por lo que el compresor será *lossy*. Sin embargo, la compresión sin pérdida (lossless) es posible también con el codificador EZW pero, por supuesto, con resultados menos espectaculares.

A.1.1 Zerotrees

El codificador EZW se basa en dos observaciones importantes: La primera es que cuando a una imagen se le aplica una transformación wavelet, la energía de las subbandas disminuye cuando decrece la escala (escalas bajas significan alta resolución), por lo que los coeficientes wavelet serán, en media, más pequeños en las subbandas con frecuencias más altas que en las subbandas con frecuencias más bajas. Esto muestra que la codificación progresiva es una elección muy adecuada para comprimir imágenes transformadas con wavelets, pues las escalas bajas sólo añaden detalles. La segunda es que los coeficientes wavelet grandes son más importantes que los coeficientes wavelet pequeños.

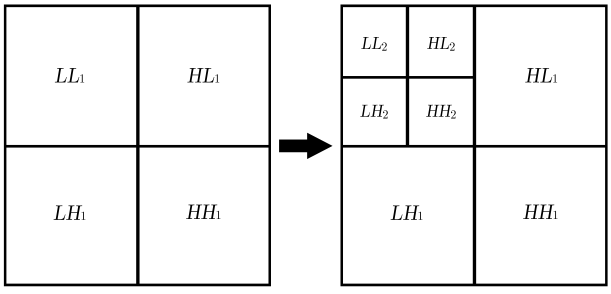


Figura A.1: *Los dos primeros pasos de una transformada wavelet.*

Estas dos características son explotadas al codificar los coeficientes wavelet en orden decreciente, en varios pasos. En cada paso se elige una tolerancia con la que se comparan todos los coeficientes wavelet. Si un coeficiente wavelet es mayor que la tolerancia se codifica y se elimina de la imagen. Si es menor, se deja para el siguientes paso. Cuando todos los coeficientes wavelet han sido revisados se disminuye la tolerancia y la imagen se vuelve a examinar para añadir más detalles a la ya codificada imagen. Este proceso se repite hasta que todos los coeficientes han sido completamente codificados o se satisface cualquier otro criterio de parada (por ejemplo, número máximo de bits).

La clave ahora es utilizar la dependencia entre los coeficientes wavelet a través de las diferentes escalas para codificar de forma eficiente grandes zonas de la imagen que están por debajo de la actual tolerancia. Aquí es donde entran los zerotrees.

Después de la transformación wavelet de una imagen podemos representarla usando árboles gracias al submuestreo que se lleva a cabo al transformar. Podemos decir que un coeficiente en una subbanda de frecuencia baja tiene cuatro descendientes en la siguiente subbanda de frecuencia más alta (ver figura A.2). Cada uno de los coeficientes tienen a su vez cuatro descendientes en la siguiente subbanda más baja de ahí que tengamos un *cuatro-árbol* (cada raíz tiene cuatro hojas).

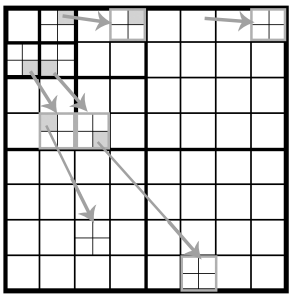


Figura A.2: *La relación entre los coeficientes wavelet en diferentes subbandas de un cuatro-árbol.*

Un zerotree, por tanto, es un cuatro-árbol en el que todos los nodos son iguales o menores que la raíz. El árbol se codifica con un sólo símbolo y se reconstruye con el decodificador como un

cuatro-árbol lleno de ceros. Sólo faltaría añadir que la raíz debe ser menor que la tolerancia con la que se están comparando los coeficientes wavelet.

El codificador EZW aprovecha los zerotrees basándose en la observación que los coeficientes wavelet decrecen con la escala. Se asume que habrá una alta probabilidad de que todos los coeficientes en un cuatro-árbol serán menores que una cierta tolerancia si la raíz es menor que esa tolerancia. Si éste es el caso entonces el árbol entero puede ser codificado con un simple símbolo de zerotree. Si la imagen se escanea en un orden predefinido, yendo desde las altas escalas a las bajas, implícitamente se codifican muchas posiciones con el uso de los símbolos de zerotrees. Por supuesto, la regla del zerotree será violada a menudo (es decir, habrá coeficientes insignificantes que tendrán algún descendiente significante), pero como demuestra la práctica, la probabilidad de que esto ocurra es muy baja en general. El precio que hay que pagar es la inclusión del símbolo zerotree en nuestro alfabeto-código.

A.1.2 Codificación

Ahora que tenemos todos los términos definidos podemos empezar a comprimir. Empezaremos con la codificación de los coeficientes de mayor a menor magnitud. Una aproximación muy directa es simplemente transmitir los valores de los coeficientes en orden decreciente, pero esto no es muy eficiente. De esta manera se utilizan muchos bits y no utilizamos el hecho de que *sabemos* que los coeficientes están prácticamente ya ordenados de forma decreciente. Una mejor aproximación es usar una tolerancia y una única señal para decodificar si los valores son mayores o menores que una tolerancia. Para llegar a la reconstrucción perfecta repetimos el proceso después de disminuir la tolerancia hasta que ésta llegue a ser menor que el coeficiente más pequeño que queremos transmitir. Podemos hacer que este proceso sea mucho más eficiente para el decodificador si utilizamos una sucesión de tolerancias predeterminada pues no tenemos que transmitirlas al decodificador y así salvar algo de ancho de banda. Si la sucesión

predeterminada es una secuencia de potencias de dos esto se convertirá en codificación por planos de bits ya que las tolerancias en este caso corresponden a los bits en la representación binaria de los coeficientes. El codificador EZW utiliza este tipo de codificación.

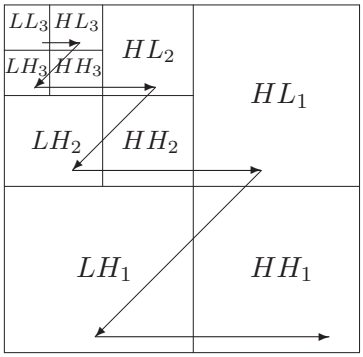


Figura A.3: *Las subbandas con frecuencias altas deben ser escaneadas siempre antes que las subbandas de frecuencias bajas.*

Como se ha comentado, EZW utiliza un orden de escaneo predefinido para codificar la transmisión de los coeficientes wavelet pero el uso de los zerotrees codifica implícitamente muchas posiciones. Varios órdenes de escaneo son posibles (ver figura A.4), mientras las escalas altas sean completamente escaneadas antes de pasar a las escalas más bajas (ver figura A.3). En [50] se utiliza un escaneo del tipo raster. El orden de escaneo tiene influencia en la compresión final. Como se estudia en [3].

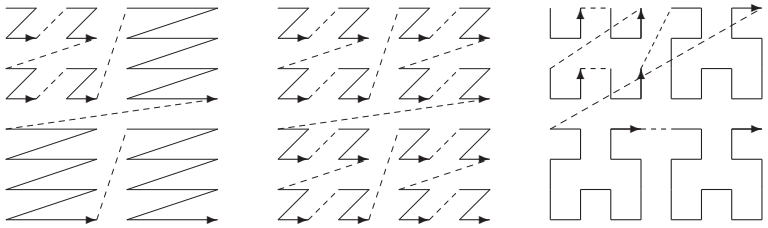


Figura A.4: *Tres ejemplos de escaneo de una imagen con dos niveles de descomposición wavelet (izquierda: Raster, centro: Morton, derecha: Peano). En el ejemplo del apartado A.1.4 se utiliza el orden Morton.*

A.1.3 El algoritmo

Ahora que sabemos cómo codifica el esquema EZW los valores y las posiciones de los coeficientes podemos continuar con el algoritmo. La secuencia de salida del EZW deberá empezar con alguna información que se le pasará al decodificador. La información requerida por el decodificador es el número de escalas utilizadas en la transformación wavelet y la tolerancia inicial, suponiendo que se utiliza siempre la misma transformada wavelet. Además podemos enviar las dimensiones y la media de la imagen. Enviar la media de la imagen es útil pues después de una reconstrucción imperfecta, el decodificador puede substituir la media imperfecta por la media original, lo que aumenta significativamente la PSNR.

El primer paso del algoritmo EZW es determinar la tolerancia inicial. Si utilizamos una codificación por planos de bit nuestra tolerancia inicial T_0 será:

$$T_0 = 2^{\lfloor \log_2(\max\{|w(x,y)|\}) \rfloor},$$

donde $w(x,y)$ denota los coeficientes de la transformada wavelet. Con esta tolerancia introducimos el bucle principal (utilizando lenguaje Matlab):

```
tolerancia = 2^(floor(log2(max(max(abs(imagen))))));
while tolerancia ~= 0.5
    paso_dominante(imagen, tolerancia);
    paso_subordinado(imagen, tolerancia);
    tolerancia = tolerancia/2;
end
```

Vemos que se utilizan dos pasos para codificar la imagen. En el primer paso, el *paso dominante*, se escanea la imagen y se saca un símbolo por cada coeficiente. Si el valor absoluto del coeficiente es mayor que la tolerancia y el coeficiente es positivo se codifica una "P", si el valor absoluto del coeficiente es menor que la tolerancia y el coeficiente es negativo se codifica una "N". Si el coeficiente es la raíz de un zerotree entonces se codifica una "T" y finalmente, si el coeficiente es menor que la tolerancia pero no es un zerotree,

entonces se codifica una "Z" (cero aislado). Esto ocurre cuando en el subárbol hay un coeficiente mayor que la tolerancia. Nótese que para determinar si un coeficiente es la raíz de un zerotree o un cero aislado, tendremos que examinar el cuatro-árbol entero y obviamente esto lleva tiempo.

Finalmente, todos los coeficientes que en valor absoluto son mayores que la actual tolerancia se extraen y se colocan sin su signo en la lista subordinada y sus posiciones en la imagen se llenan con ceros. Esto impedirá que sean codificados otra vez.

El segundo paso, el *paso subordinado*, es el paso de refinamiento. Este paso se reduce a sacar el siguiente bit más significativo de todos los coeficientes de la lista subordinada. Con otras palabras, durante el paso subordinado, la longitud del cuantizador efectivo, que define un intervalo de incertidumbre para la magnitud real del coeficiente se divide por la mitad. Para cada valor de la lista subordinada, este refinamiento se puede codificar usando un alfabeto binario: El símbolo "1" indicaría que el valor verdadero está en la mitad superior del antiguo intervalo de incertidumbre y el símbolo "0" indicaría la mitad inferior. Nótese que antes de este refinamiento, la anchura de la región de incertidumbre es exactamente igual a la actual tolerancia. Después de completar el paso subordinado los valores de la lista subordinada se ordenan de mayor a menor. El proceso continua alternando pasos dominantes y subordinados donde la tolerancia se divide por la mitad antes de cada paso dominante.

El bucle principal termina cuando la tolerancia alcanza un valor mínimo. Para coeficientes enteros, este valor mínimo es 0,5. Podemos añadir otra condición de parada por el número de bits sacados por el codificador aritmético, en ese caso podemos conseguir exactamente cualquier tasa de compresión sin realizar mucho esfuerzo.

- **Inicialización**
  $T_0 = 2^{\lfloor \log_2(\max |w(i,j)|) \rfloor}$, $k = 0$
  Lista dominante $\leftarrow$ todos los coeficientes
  Lista subordinada $\leftarrow \varnothing$

- **Paso dominante** (paso de significancia)
  Para cada entrada w de la lista dominante

Si $|w| \geq T_k$ (i.e. w es significante)

Si w es positivo $\rightarrow$ transmitir **P**

Si no (si es negativo) $\rightarrow$ transmitir **N**

End

Mover w a la lista subordinada

Si no (w insignificante)

caso 1: w es parte de un zerotree $\rightarrow$ no codificar

caso 2: w es un zerotree $\rightarrow$ transmitir **T**

caso 3: w es un cero aislado $\rightarrow$ transmitir **Z**

End

End

- **Paso subordinado** (paso de refinamiento)
  Para cada w de la lista subordinada

  Si $w \in$ mitad inferior de $[T_k, 2T_k] \rightarrow$ transmitir un 0

  Si $w \in$ mitad superior de $[T_k, 2T_k] \rightarrow$ transmitir un 1

  End

  End

- **Cuantización**
  $T_{k+1} = T_k/2$
  $k = k + 1$
  Ir al paso dominante

A.1.4 Ejemplo

Vamos a ver un ejemplo que se cita en [50] para ilustrar el orden de las operaciones realizadas por el algoritmo EZW. Veremos la secuencia de código completa pues el texto original sólo presenta el primer paso dominante. La matriz del ejemplo se muestra en la figura A.5 y se considera la transformada wavelet con 3 escalas de una imagen de 8×8 píxeles. Como el coeficiente mayor es 63,

elegimos la tolerancia inicial $T_0 = 32$. Teniendo en cuenta que se ha utilizado el orden de escaneo Morton, la secuencia de código, separada en pasos dominantes y pasos subordinados por niveles, es:

64	-34	49	10	7	13	-12	7
-31	23	14	-13	3	4	6	-1
15	14	3	12	5	-7	3	9
-9	-7	-14	8	4	-2	3	2
-5	9	-1	47	4	6	-2	2
3	0	-3	2	3	-2	0	4
2	-3	6	-4	3	6	3	6
5	11	5	6	0	3	-4	4

Figura A.5: *Matriz utilizada como ejemplo en el artículo de Shapiro*

```
D1: pnztpttttztttttttptt
S1: 1010
D2: ztnpttttttt
S2: 100110
D3: zzzzzppnppnttnnptpttnttttttttptttptttttttttptttttttttttt
S3: 10011101111011011000
D4: zzzzzzztztznzzzzpttptpptpnptntttttptpnpppptttttptptttpnp
S4: 1101111101100100000111011010000100101011100
D5: zzzzztzzzzztpzzzttptttttnptpptttptttnppntttttpnnpttptttpptttt
S5: 10111100110100010111111010110110010000000011011011001100011l
D6: zzzttztttztttttnnttt
```

Veamos cómo se obtienen algunos de los primeros símbolos en el primer paso dominante:

- El primer coeficiente tiene magnitud 63 que es mayor que la tolerancia 32, además es positivo por lo que se genera el símbolo **p**. Después de decodificar este símbolo, el decodificador sabe que el coeficiente está en el intervalo $[32, 64)$ cuyo centro es 48.

- El segundo coeficiente que se examina es el que tiene magnitud -34 que al ser mayor que 32 (en valor absoluto) y además negativo, se genera el símbolo **n**. El decodificador entenderá que que el coeficiente se encuentra en $(-64, 32]$ cuyo centro es -48.

coeficiente	base 2	símbolo	reconstrucción
63	1**1**1111	1	56
34	1**0**0010	0	40
49	1**1**0001	1	56
47	1**0**1111	0	40

Figura A.6: *Resultado del primer paso subordinado.*

- Aunque el coeficiente -31 es insignificante respecto a la tolerancia 32, tiene un descendiente significante dos generaciones más abajo con magnitud 47, por lo que se ha generado el símbolo **z** para un cero aislado.

- El coeficiente 23 es menor que 32 y todos sus descendientes también son insignificantes. Se genera el símbolo de un zerotree (**z**) y no se generará ningún símbolo para los descendientes durante el actual paso dominante.

Durante el primer paso dominante, donde se ha usado la tolerancia 32, se han encontrado cuatro coeficientes significantes. Estos coeficientes serán refinados durante el primer paso subordinado. Antes del primer paso subordinado, el intervalo de incertidumbre para las magnitudes de todos los coeficientes significantes es $[32, 64)$. El primer paso subordinado refinará estas magnitudes y las identificará como pertenecientes al intervalo $[32, 48)$, que será codificado con el símbolo '0', o al intervalo $[48, 64)$, que será codificado con el símbolo '1'. No es casualidad que estos símbolos sean exactamente el primer bit a la derecha del bit más significativo de la representación binaria de las magnitudes. El orden de las operaciones del primer paso subordinado se ilustra en la figura A.6. La primera entrada tiene magnitud 63 y se sitúa en el intervalo superior cuyo centro es 56. La siguiente entrada tiene magnitud 34, que se sitúa en el intervalo inferior cuyo centro es 40.

El proceso continua con el segundo paso dominante. La nueva tolerancia es 16. Durante este paso, sólo se escanean aquellos coeficientes que no han sido encontrados significantes. Además, los coeficientes significantes en el paso anterior son tratados como ceros.

A.1.5
Resultados numéricos

Todos los experimentos se realizan codificando y decodificando una cadena de bits para verificar el correcto funcionamiento del algoritmo. Después de una cabecera de 12 bytes, la cadena de bits entera se codifica aritméticamente usando un único codificador con un modelo adaptativo. El modelo se inicializa con una nueva tolerancia cada nuevo paso dominante y subordinado, por eso el codificador es totalmente adaptativo. Nótese que no se utilizan estadísticas de la imagen.

Después de la cabecera no hay ningún símbolo extra excepto el que señala el fin de la cadena de bits, que siempre se mantiene con probabilidad mínima. Este símbolo extra no es necesario almacenarlo en el ordenador si el fin del archivo se puede detectar.

El codificador EZW se ha aplicado a las imágenes test en escala de grises (8 bpp) y 512×512 "Lena" y "Barbara" mostradas en las figuras A.7 (a) y A.9 (a). Los resultados de la codificación de Lena se resumen en la tabla A.2 y la figura A.7. Para todos los resultados presentados usamos una descomposición wavelet con 6 niveles basada en los wavelets biortogonales 9/7 de [4]. Resultados similares se muestran para Barbara en la tabla A.3 y en la figura A.8.

El rendimiento del codificador EZW se compara además con JPEG y JPEG2000. El algoritmo JPEG no permite al usuario seleccionar el tamaño final del archivo pero, en vez de eso, permite elegir un factor de calidad. En los experimentos mostrados en al figura A.9 Barbara se codifica primero usando un una tasa de bits aproximada de 0,25 bpp. El PSNR en este caso es de 23,1107. El codificador EZW se aplica entonces a Barbara con la misma tasa de compresión 0,25 bpp. El resultado de PSNR es 26,7029. Con JPEG2000 se alcanza un PSNR = 26,56 dB. Para conseguir el mismo nivel de calidad con EZW, la tasa de compresión obtenida es de 0,23 bpp. Los valores obtenidos de los errores se pueden observar en la tabla A.1.

Como conclusión podríamos decir que el algoritmo EZW, que acabamos de comentar, es una técnica de codificación numérica

algor.	bpp	MSE	PSNR	$\|\text{error}\|_1$	$\|\text{error}\|_2$	$\|\text{error}\|_\infty$
EZW	**0,25**	138,93	26,70	7,92	11,79	74,68
JPEG	**0,25**	317,69	**23,11**	12,32	17,82	165,00
EZW	0,067	333,98	**23,11**	12,79	18,28	118,73
JP2	**0,25**	143,43	**26,56**	8,39	11,98	83,00
EZW	0,23	135,11	**26,56**	7,52	11,62	71,72

Tabla A.1: *Resultados de la compresión de la imagen Barbara para comparar los algoritmos EZW, JPEG y JPEG2000.*

bpp	factor	MSE	PSNR	$\|\text{error}\|_1$	$\|\text{error}\|_2$	$\|\text{error}\|_\infty$
1	8:1	7,36	39,46	2,12	2,71	14,45
0,5	16:1	16,00	36,09	2,99	4,00	26,43
0,25	32:1	33,76	32,85	4,12	5,81	42,69
0,125	64:1	66,21	29,92	5,46	8,14	69,64
1/16	128:1	112,51	27,62	6,97	10,61	89,76
1/32	256:1	174,78	25,40	8,86	13,22	131,88
1/64	512:1	294,65	23,44	12,02	17,17	121,47
1/128	1024:1	475,85	21,36	15,44	21,81	145,29

Tabla A.2: *Resultados de la compresión de la imagen Lena* 512×512 *con el algoritmo EZW utilizando diferentes tasas de compresión.*

bpp	factor	MSE	PSNR	$\|\text{error}\|_1$	$\|\text{error}\|_2$	$\|\text{error}\|_\infty$
1	8:1	19,59	35,21	3,31	4,43	28,56
0,5	16:1	58,97	30,42	5,45	7,68	55,00
0,25	32:1	138,93	26,70	7,92	11,79	74,68
0,125	64:1	244,33	24,25	10,27	15,63	108,10
1/16	128:1	333,88	22,89	12,77	18,27	116,45
1/32	256:1	415,60	21,94	14,37	20,38	140,94
1/64	512:1	531,41	20,88	16,99	23,05	140,69
1/128	1024:1	763,36	19,30	21,17	27,63	149,00

Tabla A.3: *Resultados de la compresión de la imagen* 512×512 *Barbara con el algoritmo EZW utilizando diferentes tasas de compresión.*

Figura A.7: *Resultados de la compresión de la imagen Lena con el algoritmo EZW. (a) Imagen original con 8 bpp (b) 1 bpp, factor de compresión 8:1, PSNR = 39,46 dB (c) 0,5 bpp, factor de compresión 16:1, PSNR = 36,09 dB (d) 0,25 bpp, factor de compresión 32:1, PSNR = 32,85 dB (e) 0,0625 bpp, factor de compresión 128:1, PSNR = 27,62 dB (f) 0,015625 bpp, factor de compresión 512:1, PSNR = 23,44 dB.*

Figura A.8: *Resultados de la compresión de la imagen Barbara con el algoritmo EZW. (a) 1 bpp, factor de compresión 8:1, PSNR = 35,21 dB (b) 0,5 bpp, factor de compresión 16:1, PSNR = 30,42 dB (c) 0,125 bpp, factor de compresión 64:1, PSNR = 24,25 dB (d) 0,0625 bpp, factor de compresión 128:1, PSNR = 22,89 dB.*

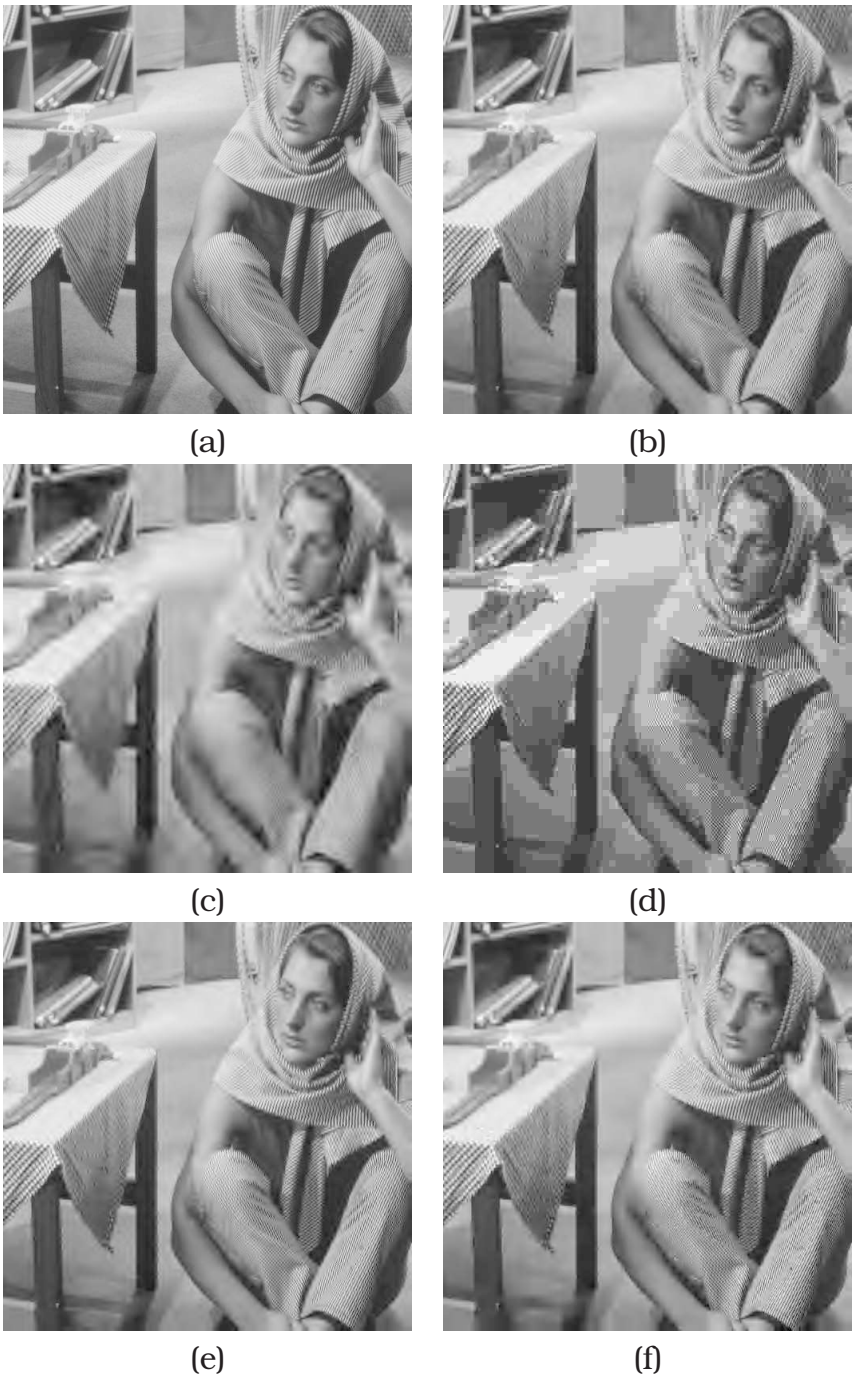


Figura A.9: *Comparación de EZW con JPEG y JPEG2000 con la imagen Barbara (a) Original (b) EZW con 0,25 bpp / PSNR = 26,70 dB (c) EZW con 0,067 bpp / PSNR = 23,11 dB (d) JPEG con 0,25 bpp / PSNR = 23,11 dB (e) EZW con 0,23 bpp / PSNR = 26,56 dB (f) EZW con 0,25 bpp / PSNR = 26,56 dB.*

que produce una cadena de bits completamente progresiva. Además, la compresión producida por este algoritmo es competitiva con la mayoría de las técnicas. Estos notables resultados se pueden atribuir a la utilización de las siguientes cuatro características:

- Una transformada wavelet discreta, que permite codificar de forma eficiente los bits más significativos de la mayoría de coeficientes, como parte de unos zerotrees que crecen de forma exponencial.
- La codificación con zerotrees, que proporciona ganancias substanciales en los mapas de significancias.
- La aproximación sucesiva, que permite la codificación de múltiples mapas de significancia usando zerotrees, y permite codificar y decodificar parando en cualquier punto.
- Codificación aritmética adaptativa, que permite al codificador incorporar conocimientos en la misma cadena de bits.

El control de la tasa de compresión exacta, que puede se elegida por el usuario, es otra de las ventajas de este algoritmo. Además, como no se necesita ningún entrenamiento o conocimiento de la imagen, el algoritmo funciona extraordinariamente bien con la mayoría de los tipos de imágenes.

A.2
Algoritmo SPIHT

Veremos ahora la implementación del algoritmo SPIHT (*Set Partitioning in Hieriarchical Trees*) que fue ideado por Amir Said y William A. Pearlman en 1996 [48] y tiene sus raíces en el algoritmo EZW, descrito en la sección anterior. SPIHT utiliza técnicas de codificación de planos de bit y es uno de los compresores progresivos más eficientes que se conocen.

La clave para determinar un método efectivo de codificación de los planos de bits está en darse cuenta de que existe una cierta

dependencia entre los coeficientes dentro del espacio wavelet pues hay bastante parecido entre dos bandas de niveles (de frecuencia) consecutivos y entre las tres bandas de un mismo nivel (misma frecuencia). Esta correlación entre coeficientes puede expresarse diciendo que si un coeficiente es significativo en un nivel superior de frecuencia, entonces los 4 coeficientes que ocupan la misma posición relativa dentro del nivel inferior tienden a ser significativos, y viceversa. Decimos por tanto que la DWT-2D posee autosimilaridad entre bandas de una misma frecuencia y entre bandas de frecuencia diferentes. La relación espacial exacta entre un coeficiente de coordenadas (i, j) y sus cuatro hijos (si es que existen), es que estos se encuentran en las coordenadas absolutas

$$\mathcal{O}(i,j) = \{(2i, 2j), (2i, 2j+1), (2i+1, 2j), (2i+1, 2j+1)\} \tag{A.1}$$

y esta relación se propaga de forma recursiva entre todos los coeficientes wavelet. Por lo tanto, un nodo (i, j) del árbol tiene 4 hijos o no tiene ninguno, lo que ocurre en el nivel más bajo de la descomposición. Para aclarar este concepto de dependencia espacial entre coeficientes, la figura A.10 muestra algunos ejemplos de árboles cuaternarios. A cada árbol diseñado usando la expresión (A.1) se le llama "árbol de orientación espacial" (AOE). A todos los descendientes de (i, j) (hijos, nietos, etc.) los denotaremos por $\mathcal{D}(i, j)$. Nótese que $\mathcal{D}(i, j) + (i, j)$ forman un AOE completo. La relación estadística entre un elemento (i, j) y sus descendientes $\mathcal{D}(i, j)$ es que si (i, j) es 1 entonces al menos uno de sus descendientes es probablemente 1. Por el contrario, si (i, j) es 0, lo más probable es que todos sus descendientes sean 0. Esta es la principal razón por la que podemos diseñar compresores a partir del espacio wavelet.

A.2.1 Codificación

SPIHT codifica eficientemente los mapas de significancia, aprovechando la similaridad o semejanza que existe entre las diferentes escalas, aunque no explota la redundancia que pueda existir entre las tres bandas de una misma escala.

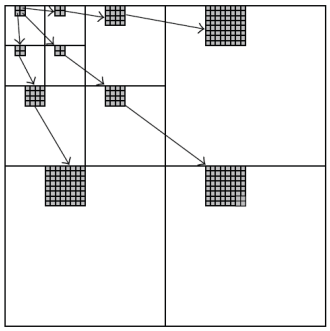


Figura A.10: *Algunos ejemplos de árboles creados con la DWT-2D.*

Lo primero que hace es transmitir el plano de bits más significativo (los coeficientes se tratan en forma signo-magnitud). Después entra en un bucle que itera tantas veces como planos de bits tienen que transmitirse. En la emisión de cada plano de bits diferencia entre los bits de significancia y los bits de refinamiento.

SPIHT trabaja particionando los AOEs de forma que tiende a mantener coeficientes no significativos en grandes conjuntos y comunica con un único bit si alguno de los elementos de un conjunto es significativo o no. Las decisiones de particionamiento son decisiones binarias que son transmitidas al descodificador e indican los planos de significancia. El particionamiento es tan eficiente que la codificación aritmética binaria de las decisiones sólo provoca una pequeña ganancia. Esto es consecuencia directa de que SPIHT efectúa un número de particionamientos mínimo de forma que la probabilidad de encontrar un coeficiente significativo en cada conjunto es aproximadamente igual a la probabilidad de no encontrarlo.

Por lo tanto, SPIHT en lugar de transmitir las coordenadas de los coeficientes significativos en el plano actual, transmite los resultados de las comparaciones que han provocado la determinación (por parte del codificador) de todos los zerotrees que forman dicho plano. El descodificador no necesita más información para saber que el resto de coeficientes que no pertenecen a ningún zerotree son iguales a 1. De hecho, el descodificador ejecuta exactamente el mismo algoritmo que el codificador y como no dispone

de los coeficientes wavelet para saber si son significativos o no, usa los resultados de las comparaciones que le llegan en la secuencia-código. De esta forma la traza de instrucciones es idéntica.

SPIHT transmite los bits de significancia y de refinamiento en dos fases independientes llamadas de ordenación y de refinamiento, respectivamente. El nombre de la fase de refinamiento es obvio. Sin embargo la fase de ordenación se llama así porque lo que el codificador hace es ordenar los coeficientes atendiendo a su valor absoluto y luego enviarlos según ha resultado dicho orden. Pero nótese que la ordenación que se produce es muy suave ya que no es necesario ordenar todos los coeficientes que son significativos en el plano que se transmite, solo debemos encontrar que coeficientes van antes que otros porque son significativos en un plano superior.

La fase de ordenación

Como ya hemos indicado, SPIHT es eficiente porque realiza un número mínimo de comparaciones equiprobables. La clave está en saber cómo formular dichas comparaciones, que se construyen usando un algoritmo de particionamiento de los AOEs.

El codificador y el decodificador manejan conceptualmente dos listas de coeficientes representados por sus coordenadas espaciales. En una se almacenan todos los coeficientes que son significativos en el plano actual de transmisión p. Todos estos coeficientes c verifican que

$$|c| \geq 2^p.$$

La otra lista almacena el resto de coeficientes que no son significativos. Inicialmente esta lista contiene todos los coeficientes (tantos como píxeles existen en la imagen) y la lista de coeficientes significativos está vacía. Denotaremos por LIP (*List of Insignificant Pixels*) a la lista de coeficientes no significativos y por LSP (*List of Significant Pixels*) a la de coeficientes significativos.

SPIHT realiza una partición inicial

$$\{(0,0),(0,1),(1,0),(1,1),\mathcal{D}(0,1),\mathcal{D}(1,0),\mathcal{D}(1,1)\}$$

donde como ya sabemos, $\mathcal{D}(i,j)$ representa a todos los coeficientes

descendientes de (i,j) que se determinan aplicando la Ecuación (A.2) recursivamente.

SPIHT averigua que elementos de esta partición son significativos. Más formalmente, evalúa la función

$$S_p(\mathcal{T}) = \begin{cases} 1 & \text{si algún coeficiente } c \in \mathcal{T} \text{ tal que } |c| \geqslant 2^p \\ 0 & \text{en caso contrario} \end{cases} \tag{A.2}$$

donde $\mathcal{T}$ puede ser un único coeficiente o un conjunto de coeficientes.

En la codificación del primer plano de bits, las 4 raíces $(0,0)$, $(0,1),(1,0),(1,1)\}$ tienen un 50 % de posibilidades de ser significativas y a sus descendientes $\{\mathcal{D}(0,1),\mathcal{D}(1,0),\mathcal{D}(1,1)\}$ ($(0,0)$ no tiene descendientes) les ocurre lo mismo, tienen probabilidad igual a 50 % (aproximadamente) de ser un zerotree. SPIHT realiza todas estas comparaciones y emite los bits de código correspondientes.

Para gestionar las particiones, SPIHT usa realmente 3 listas: LIP, LSP y LIS (*List of Insignificant Sets*) o lista de conjuntos no significativos, porque es una forma sencilla de distinguir entre coeficientes y conjuntos de coeficientes. Por lo tanto, el contenido inicial de dichas listas es:

$$\begin{aligned} LSP &\leftarrow \varnothing \\ LIP &\leftarrow \{(0,0),(0,1),(1,0),(1,1)\} \\ LIS &\leftarrow \{(0,1),(1,0),(1,1)\} \end{aligned}$$

Cuando un coeficiente de LIP no es significativo, no ocurre nada en las listas, pero si lo es, se mueve desde LIP a LSP, para posteriormente ser refinado. De forma similar, si un coeficiente de LIS no es significativo (es un zerotree) no ocurre nada en las listas, pero si es significativo, debe ser particionado en subconjuntos que tengan tantas posibilidades de ser zerotrees como de no serlo.

SPIHT particiona un $\mathcal{D}(i,j)$ en

$$\{\mathcal{O}(i,j),\mathcal{L}(i,j)\}$$

donde

$$\mathcal{L}(i,j) = \mathcal{D}(i,j) - \mathcal{O}(i,j).$$

Cada $\mathcal{D}(i,j)$ se descompone en 5 partes: los 4 nodos hijo de (i,j) y el resto de descendientes.

Los $(k,l) \in \mathcal{O}(i,j)$ se insertan en LIP o en LSP dependiendo de si son significativos o no. En el caso de viajar a LIP la inserción debe realizarse al final de la lista para que los coeficientes sean evaluados en la pasada actual. Los $\mathcal{L}(i,j)$ se insertan en LIS para ser más tarde evaluados.

En LIS podemos encontrar, por tanto, 2 tipos de conjuntos: $\mathcal{D}(i,j)$ y $\mathcal{L}(i,j)$, que tienen un numero diferente de elementos. SPIHT los diferencia diciendo que los $\mathcal{D}(i,j)$ son de tipo A mientras que los $\mathcal{L}(i,j)$ son de tipo B. El particionado de un $\mathcal{L}(i,j)$ es distinto, por tanto, al de un $\mathcal{D}(i,j)$. Si un $\mathcal{L}(i,j)$ es significativo entonces se particiona en 4 conjuntos

$$\mathcal{D}(k,l), \text{ con } (k,l) \in \mathcal{O}(i,j)$$

es decir, en los 4 árboles hijos de (i,j) que se vuelven a insertar al final de LIS. Finalmente $\mathcal{L}(i,j)$ desaparece de LIS porque este AOE ha sido particionado en sus 4 subárboles.

La fase de refinamiento

Entre cada fase de ordenación se realiza otra de refinamiento. Si p es el plano de bits a transmitir, en esta fase se emite el p-ésimo bit más significativo de cada coeficiente almacenado en LSP. Cuando un coeficiente ha sido totalmente enviado se elimina de LSP. En este punto existen dos alternativas de implementación. Si O_p representa los bits emitidos durante la fase de ordenación de la capa p y R_p a los bits de refinamiento, una posibilidad de construcción del secuencia-código es

$$O_p R_p O_{p-1} R_{p-1} O_{p-2} \ldots$$

Sin embargo, los autores del algoritmo recomiendan que los bits de ordenación del plano $p-1$ antecedan a los bits de refinamiento del plano p, es decir

$$O_p O_{p-1} R_p O_{p-2} R_{p-1} \ldots$$

Este procedimiento tiene sentido porque de esta forma primero se envían los bits que definen nuevos coeficientes distintos de 0 y a continuación se refinan los coeficientes que ya eran significativos.

Las dos alternativas han sido evaluadas y sus rendimientos son muy similares. En la exposición del pseudo-código que expone el codec, por sencillez, se usara la primera opción.

La fase de cuantificación

La fase de cuantificación se usa para decrementar el umbral de significancia que en SPIHT son siempre potencias de dos. De esta forma se seleccionan los planos de bits de los coeficientes wavelet en el orden correcto.

A.2.2 El algoritmo

Se presenta solo el algoritmo de compresión ya que el descompresor es prácticamente idéntico.

1. Fase de inicialización
   - Emitir $p = \lfloor \log_2(\max_{(i,j)}\{|w(i,j)|\}) \rfloor$ (índice del plano más significativo).
   - LSP $\leftarrow \varnothing$
   - LIP $\leftarrow \{(0,0),(0,1),(1,0),(1,1)\}$
   - LIS $\leftarrow \{(0,1),(1,0),(1,1)\}$
2. Mientras $p \geq 0$:

   Fase de ordenación

   - Para cada $(i,j) \in$ LIP:
     Emitir $S_p(i,j)$.
     Si $S_p(i,j)$=1 entonces:
     Mover (i,j) desde LIP a LSP
     Emitir el signo de (i,j).
   - Para cada $(i,j) \in$ LIS:
     - Si (i,j) es del tipo A entonces:
       Emitir $S_p(\mathcal{D}(i,j))$.

Si $S_p(\mathcal{D}(i,j)) = 1$ entonces:
Para cada $(k,l) \in \mathcal{O}(i,j)$:
Emitir $S_p(k,l)$
Si $S_p(k,l) = 1$ entonces
Añadir (k,l) a LSP
Emitir el signo de (k,l)
Si no:
Añadir (k,l) a LIP
Si $\mathcal{L}(i,j) \neq \varnothing$ (tiene al menos nietos) entonces:
Mover (i,j) al final de LIS como del tipo B
Si no:
Borrar (i,j) de LIS

- Si no (es del tipo B):
Emitir $S_p(\mathcal{L}(i,j))$ (si alguno de los nietos de (i,j) es significativo)
Si $S_p(\mathcal{L}(i,j)) = 1$ entonces:
Añadir cada hijo $(k,l) \in \mathcal{O}(i,j)$ al final de LIS como de tipo A
Borrar (i,j) de LIS

Fase de refinamiento

- Para cada $(i,j) \in$ LSP (excepto aquellos incluidos en la última fase de ordenación):
Emitir el p-ésimo bit del coeficiente (i,j).

Fase de cuantificación

$p \leftarrow p - 1$

A.2.3 Decodificación

Para encontrar el algoritmo de descodificación debe tenerse presente que todas las instrucciones de salto condicional están controladas por el valor devuelto por $S_p(\cdot)$ (ver ecuación A.2), los cuales

forman la secuencia-código generada durante la fase de ordenación y que por lo tanto son conocidos por el descodificador. Éste puede realizar exactamente la misma traza de instrucciones pues el algoritmo que ejecuta es idéntico al del codificador excepto porque donde aparece "emitir" ahora debe poner "recibir". Ya que se trata del mismo algoritmo, las tres listas LSP, LIP y LIS van a generarse exactamente de la misma forma a como se generaron en el codificador, gracias a lo cual, la reconstrucción del plano es trivial y además, las complejidades del codificador y del decodificador son idénticas.

Sin embargo, si SPIHT va a ser usado como transmisor progresivo, el decodificador debe realizar una tarea extra para ajustar los coeficientes reconstruidos a partir del intervalo de incertidumbre durante la fase de ordenación. Cuando una coordenada se mueve a LSP, se sabe que el valor del coeficiente c cumple que

$$2^p \leq |c| < 2^{p+1}$$

donde p es el plano de bit transmitido. El decodificador utiliza esta información (más el bit de signo que llega a continuación), para ajustar el valor reconstruido a la mitad del intervalo $[2^p, 2^{p+1} - 1]$ ya que de esta forma el error con respecto al valor real del coeficiente será la mitad en promedio.

De forma similar, durante la fase de refinamiento, cuando el descodificador conoce el valor real del bit $(p-1)$-ésimo, resulta otro intervalo de incertidumbre. Si el bit recibido es un 1, entonces el bit $(p-1)$-ésimo ya es correcto, pero debemos hacer el bit $(p-2)$-ésimo igual a 1 para ajustar a la mitad del nuevo intervalo de incertidumbre. Si el bit recibido es un 0, entonces el bit $(p-1)$-ésimo está equivocado y debe ser puesto a 0. El bit $(p-2)$-ésimo debe hacerse 1 para ajustar el valor reconstruido a la mitad del intervalo de incertidumbre. Por lo tanto, en cualquier caso, cuando estamos refinando, colocamos el bit recibido a su valor adecuado y el siguiente bit menos significativo se hace 1.

A.2.4 Ejemplo

Vamos a utilizar el mismo ejemplo que se utiliza en la sección A.1.4 para ilustrar el orden de las operaciones realizadas por el algoritmo SPIHT. La matriz utilizada para el ejemplo se muestra en la figura A.5 y se considera la transformada wavelet con 3 escalas de una imagen de 8×8 píxeles. La secuencia de código, separada en pasos de ordenación y de refinamiento, es:

```
p=5 FASE DE ORDENACIÓN
    paso LIP 111000
    paso LIS 11100010000001010110000
    FASE DE REFINAMIENTO
    nada

p=4 FASE DE ORDENACIÓN
    paso LIP 10110000000000
    paso LIS 00000
    FASE DE REFINAMIENTO
    1010

p=3 FASE DE ORDENACIÓN
    paso LIP 1111101111100000
    paso LIS 10101011110110010001100101100110000010110 0
    FASE DE REFINAMIENTO
    100110

p=2 FASE DE ORDENACIÓN
    paso LIP 10000010000011110111111 0000
    paso LIS 11110111111111011011111100100011101100101 11011
    FASE DE REFINAMIENTO
    10011101111011000110

p=1 FASE DE ORDENACIÓN
    paso LIP 0101111110111011011111111011101011011011 11
    paso LIS nada
    FASE DE REFINAMIENTO
    1101111101100100100010010 11

p=0 FASE DE ORDENACIÓN
    paso LIP 1001000
    paso LIS nada
    FASE DE REFINAMIENTO
```

```
1011110011010001110111111010001011000000001011011111001000111
```

Veamos cómo se obtienen algunos de los primeros símbolos observando cómo van cambiando las diferentes listas:

- **INICIALIZACIÓN**

```
 LIP             LSP             LIS
(1,1)  63       vacía           (1,2)A -34
(1,2) -34                       (2,1)A -31
(2,1) -31                       (2,2)A  23
(2,2)  23
```

- $p = \lfloor \log_2(63) \rfloor = 5$, tolerancia $= 32$
  - **Fase de ordenación**

```
paso LIP

(1,1) 63 -> 1, signo -> 1
(1,2)-34 -> 1, signo -> 0
(2,1)-31 -> 0
(2,2) 23 -> 0
```

Así es como quedan las listas:

```
 LIP             LSP             LIS
(2,1) -31       (1,1)  63       (1,2)A -34
(2,2)  23       (1,2) -34       (2,1)A -31
                                (2,2)A  23
paso LIS

max(D(1,2)A)=49 -> 1
(1,3) 49 -> 1, signo -> 1
(1,4) 10 -> 0
(2,3) 14 -> 0
(2,4)-13 -> 0

 LIP             LSP             LIS
(2,1) -31       (1,1)  63       (2,1)A -31
(2,2)  23       (1,2) -34       (2,2)A  23
(1,4)  10       (1,3)  49       (1,2)B -34
(2,3)  14
(2,4) -13

max(D(2,1)A)=47 -> 1
```

```
(3,1)15 -> 0
(3,2)14 -> 0
(4,1)-9 -> 0
(4,2)-7 -> 0

 LIP             LSP             LIS
(2,1)-31        (1,1) 63        (2,2)A  23
(2,2) 23        (1,2)-34        (1,2)B -34
(1,4) 10        (1,3) 49        (2,1)B -31
(2,3) 14
(2,4)-13
(3,1) 15
(3,2) 14
(4,1) -9
(4,2) -7

max(D(2,2)A)=14 -> 0
max(D(1,2)B)=13 -> 0
max(D(2,1)B)=47 -> 1

 LIP             LSP             LIS
(2,1)-31        (1,1) 63        (2,2)A  23
(2,2) 23        (1,2)-34        (1,2)B -34
(1,4) 10        (1,3) 49        (3,1)A  15
(2,3) 14                        (3,2)A  14
(2,4)-13                        (4,1)A  -9
(3,1) 15                        (4,2)A  -7
(3,2) 14
(4,1) -9
(4,2) -7

max(D(3,1)A)= 9 -> 0
max(D(3,2)A)=47 -> 1
(5,3)-1 -> 0
(5,4)47 -> 1, signo -> 1
(6,3)-3 -> 0
(6,4) 2 -> 0

 LIP             LSP             LIS
(2,1)-31        (1,1) 63        (2,2)A  23
(2,2) 23        (1,2)-34        (1,2)B -34
(1,4) 10        (1,3) 49        (3,1)A  15
(2,3) 14        (5,4) 47        (4,1)A  -9
(2,4)-13                        (4,2)A  -7
(3,1) 15
(3,2) 14
```

```
(4,1) -9
(4,2) -7
(5,3) -1
(6,3) -3
(6,4)  2

max(D(4,1)A)=11 -> 0
max(D(4,2)A)= 6 -> 0
```

Después de estos dos últimos ceros, las listas se quedan igual.

- **Fase de refinamiento**
  Nada

- $p = 4$, tolerancia$= 16$
  - **Fase de ordenación**

```
paso LIP

(2,1)-31 -> 1, signo -> 0
(2,2) 23 -> 1, signo -> 1
(1,4) 10 -> 0
(2,3) 14 -> 0
(2,4)-13 -> 0
(3,1) 15 -> 0
(3,2) 14 -> 0
(4,1) -9 -> 0
(4,2) -7 -> 0
(5,3) -1 -> 0
(6,3) -3 -> 0
(6,4)  2 -> 0

 LIP             LSP             LIS
(1,4) 10        (1,1) 63        (2,2)A  23
(2,3) 14        (1,2)-34        (1,2)B -34
(2,4)-13        (1,3) 49        (3,1)A  15
(3,1) 15        (5,4) 47        (4,1)A  -9
(3,2) 14        (2,1)-31        (4,2)A  -7
(4,1) -9        (2,2) 23
(4,2) -7
(5,3) -1
(6,3) -3
(6,4)  2

paso LIS
```

```
max(D(2,2)A)=14 -> 0
max(D(1,2)B)=13 -> 0
max(D(3,1)A)= 9 -> 0
max(D(4,1)A)=11 -> 0
max(D(4,2)A)= 6 -> 0
```

Las listas no cambian

- **Fase de refinamiento**

```
(1,1) 63 = 111111(base 2) -> 1 (5º bit por la dcha.)
(1,2)-34 = 100010 -> 0
(1,3) 49 = 110001 -> 1
(5,4) 47 = 101111 -> 0
```

A.2.5 Resultados numéricos

Los siguientes resultados se obtuvieron con imágenes en escala de grises (8 bpp) de tamaño 512×512 píxeles. Se han utilizado wavelets biortogonales 9.7 que son con los que se consiguen mejores resultados como se prueba en [37] y con 6 escalas pues, según se indica en [37], SPIHT es más eficiente cuando se usan 5 niveles o más de descomposición wavelet. Es importante observar que las tasas de bits no son estimaciones sino que se calcularon a partir del tamaño de los archivos comprimidos.

Los resultados de la codificación de las imágenes Lena, Barbara y Goldhill se muestran en las tablas A.4, A.5 y A.6 y en las figuras A.13, A.14 y A.15 respectivamente. Además el rendimiento del algoritmo SPIHT se compara con los algoritmos EZW y JPEG2000 en las figuras A.11 y A.12 donde se muestra la eficiencia del nuevo método.

Podemos decir que SPIHT es un método de codificación completamente progresiva que utiliza:

- Los principios del ordenamiento parcial por magnitud
- La partición de conjuntos por significancia de magnitudes con respecto a las tolerancias decrecientes
- transmisión por planos de bits

bpp	factor	MSE	PSNR	$\|\text{error}\|_1$	$\|\text{error}\|_2$	$\|\text{error}\|_\infty$
1	8:1	6,71	39,86	2,04	2,59	14,85
0,5	16:1	13,67	36,77	2,80	3,70	26,12
0,25	32:1	28,06	33,65	3,81	5,30	45,13
0,125	64:1	56,22	30,63	5,19	7,50	62,38
1/16	128:1	106,40	27,86	6,98	10,32	81,31
1/32	256:1	183,43	25,50	9,27	13,54	127,06
1/64	512:1	276,89	23,71	11,42	16,64	119,78
1/128	1024:1	421,61	21,88	14,38	20,53	135,28

Tabla A.4: *Resultados de la compresión de la imagen Lena* 512×512 *con el algoritmo SPIHT utilizando diferentes tasas de compresión y sin codificación aritmética.*

bpp	factor	MSE	PSNR	$\|\text{error}\|_1$	$\|\text{error}\|_2$	$\|\text{error}\|_\infty$
1	8:1	16,77	35,89	3,12	4,10	29,27
0,5	16:1	52,02	30,97	5,19	7,21	50,71
0,25	32:1	126,00	27,13	7,78	11,23	71,72
0,125	64:1	233,95	24,44	10,45	15,29	100,44
1/16	128:1	318,54	23,10	12,34	17,85	118,86
1/32	256:1	404,49	22,06	14,27	20,11	140,61
1/64	512:1	538,76	20,82	17,21	23,21	145,32
1/128	1024:1	702,99	19,66	20,06	26,51	146,40

Tabla A.5: *Resultados de la compresión de la imagen Barbara* 512×512 *con el algoritmo SPIHT utilizando diferentes tasas de compresión y sin codificación aritmética.*

bpp	factor	MSE	PSNR	$\|\text{error}\|_1$	$\|\text{error}\|_2$	$\|\text{error}\|_\infty$
1	8:1	12,01	37,34	2,67	3,47	20,63
0,5	16:1	30,69	33,26	4,21	5,54	40,30
0,25	32:1	58,22	30,48	5,71	7,63	49,79
0,125	64:1	94,35	28,38	7,11	9,71	77,56
1/16	128:1	143,99	26,55	8,69	12,00	110,73
1/32	256:1	198,66	25,15	10,30	14,09	128,35
1/64	512:1	267,89	23,85	12,05	16,37	155,12
1/128	1024:1	366,99	22,48	14,20	19,16	146,44

Tabla A.6: *Resultados de la compresión de la imagen Goldhill* 512×512 *con el algoritmo SPIHT utilizando diferentes tasas de compresión y sin codificación aritmética.*

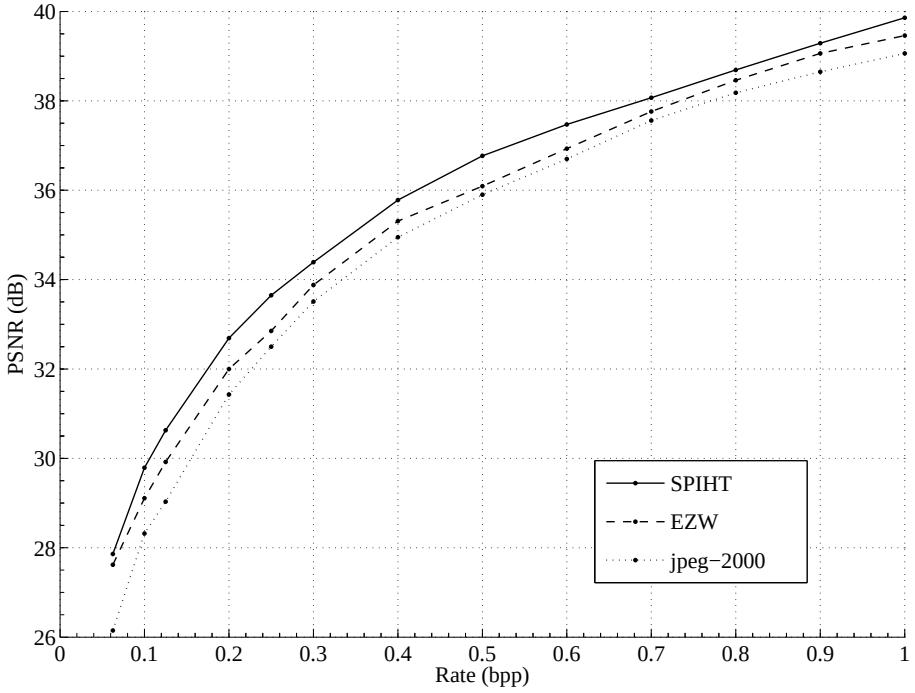


Figura A.11: *Comparación de las tasas de compresión de los algoritmos jpeg-2000, EZW y SPIHT con la imagen Lena*

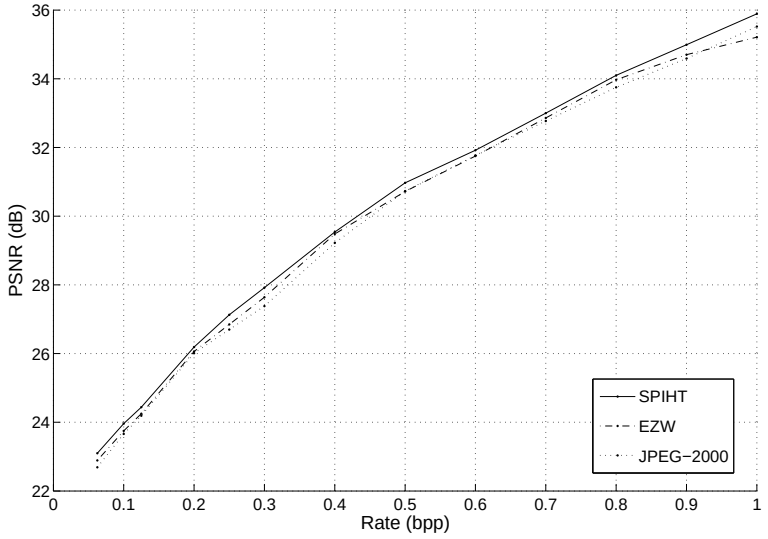


Figura A.12: *Comparación de las tasas de compresión de los algoritmos jpeg-2000, EZW y SPIHT con la imagen Barbara*

- autosimilaridad de las escalas de una imagen transformada con wavelets

La realización de estos principios con los algoritmos de codificación y decodificación se muestra más efectiva que las implementaciones del código EZW. Los resultados del algoritmo SPIHT superan en la mayoría de casos los conseguidos con los algoritmos creados anteriormente con las mismas imágenes, los cuales utilizan algoritmos mucho más complejos y no poseen la propiedad de la codificación progresiva y el control preciso de la tasa de compresión. Podemos decir que tanto la velocidad del algoritmo SPIHT como sus resultados son excelentes. Además, como se muestra en [48], se pueden mejorar las tasas de compresión utilizando codificación aritmética.

SPIHT codifica los bits de la imagen transformada con wavelets siguiendo una secuencia de planos de bits. Por tanto, es capaz de reconstruir la imagen perfectamente codificando todos los bits de la transformada. Sin embargo, la transformada wavelet consigue

una reconstrucción perfecta sólo si los números se almacenan con precisión infinita. En la práctica es muy común usar redondeo después de la transformada inversa para conseguir una reconstrucción perfecta, pero no es la manera más eficiente de hacerlo.

Para una compresión sin pérdida, los creadores del algoritmo SPIHT proponen una transformación de multirresolución con enteros en [47], [48], que ellos llaman transformada S+P (*Sequential + Prediction Transform*). Esta transformada soluciona el problema truncando los coeficientes de la transformada durante la transformación (en lugar de hacerlo después). Un resultado sorprendente obtenido con este codec es que para compresión sin pérdida es tan efectivo como los mejores codificadores sin pérdida (exceptuando JPEG-LS).

Figura A.13: *Resultados de la compresión de la imagen Lena con el algoritmo SPIHT. (a) Imagen original con 8 bpp (b) 1 bpp, factor de compresión 8:1, PSNR = 39,86 dB (c) 0,5 bpp, factor de compresión 16:1, PSNR = 36,77 dB (d) 0,25 bpp, factor de compresión 32:1, PSNR = 33,65 dB (e) 0,0625 bpp, factor de compresión 128:1, PSNR = 27,86 dB (f) 0,015625 bpp, factor de compresión 512:1, PSNR = 23,70 dB.*

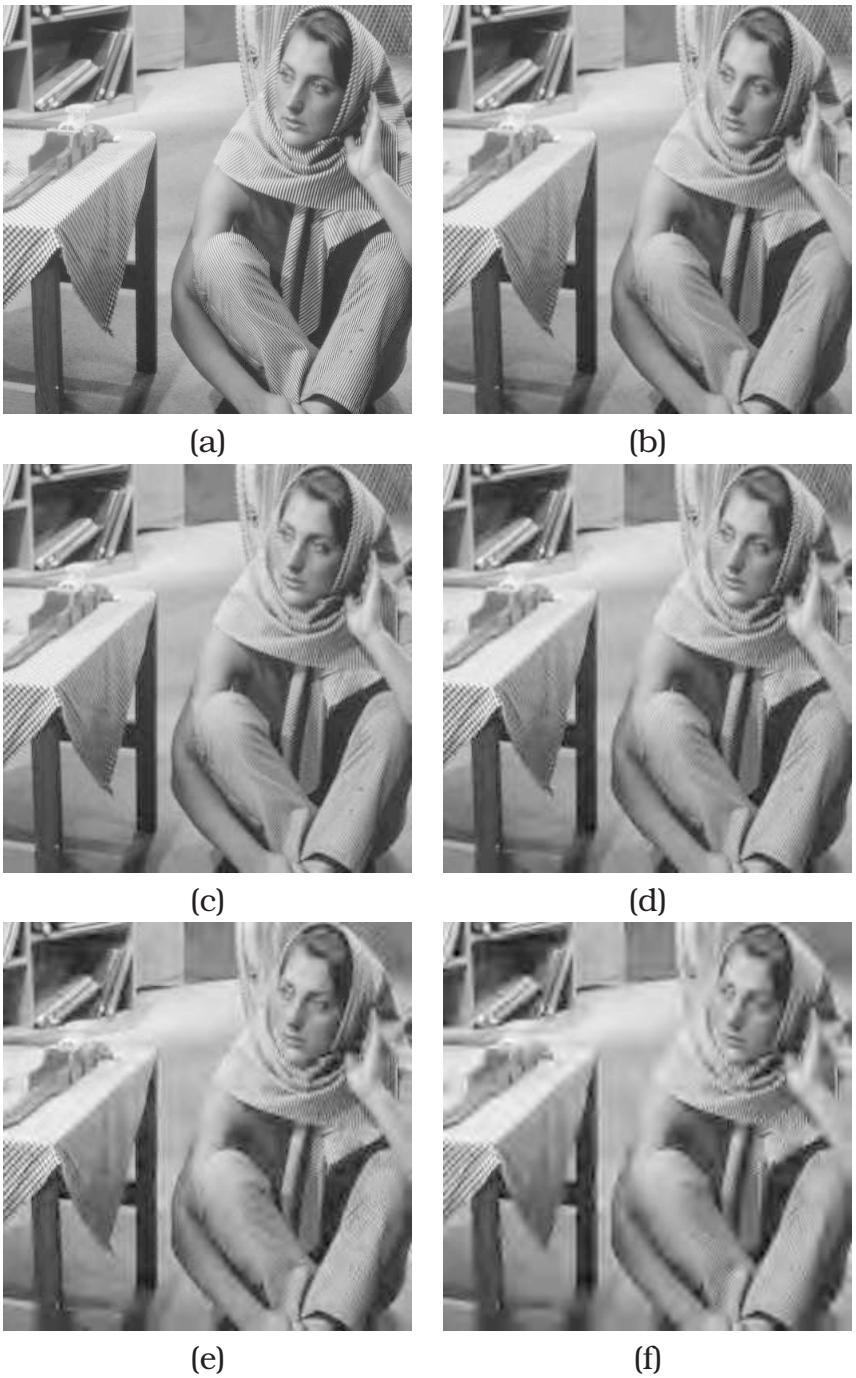


Figura A.14: *Resultados de la compresión de la imagen Barbara con el algoritmo SPIHT. (a) Imagen original con 8 bpp (b) 1 bpp, factor de compresión 8:1, PSNR = 35,89 dB (c) 0,5 bpp, factor de compresión 16:1, PSNR = 30,97 dB (d) 0,25 bpp, factor de compresión 32:1, PSNR = 27,13 dB (e) 0,125 bpp, factor de compresión 128:1, PSNR = 24,44 dB (f) 0,0625 bpp, factor de compresión 512:1, PSNR = 23,10 dB.*

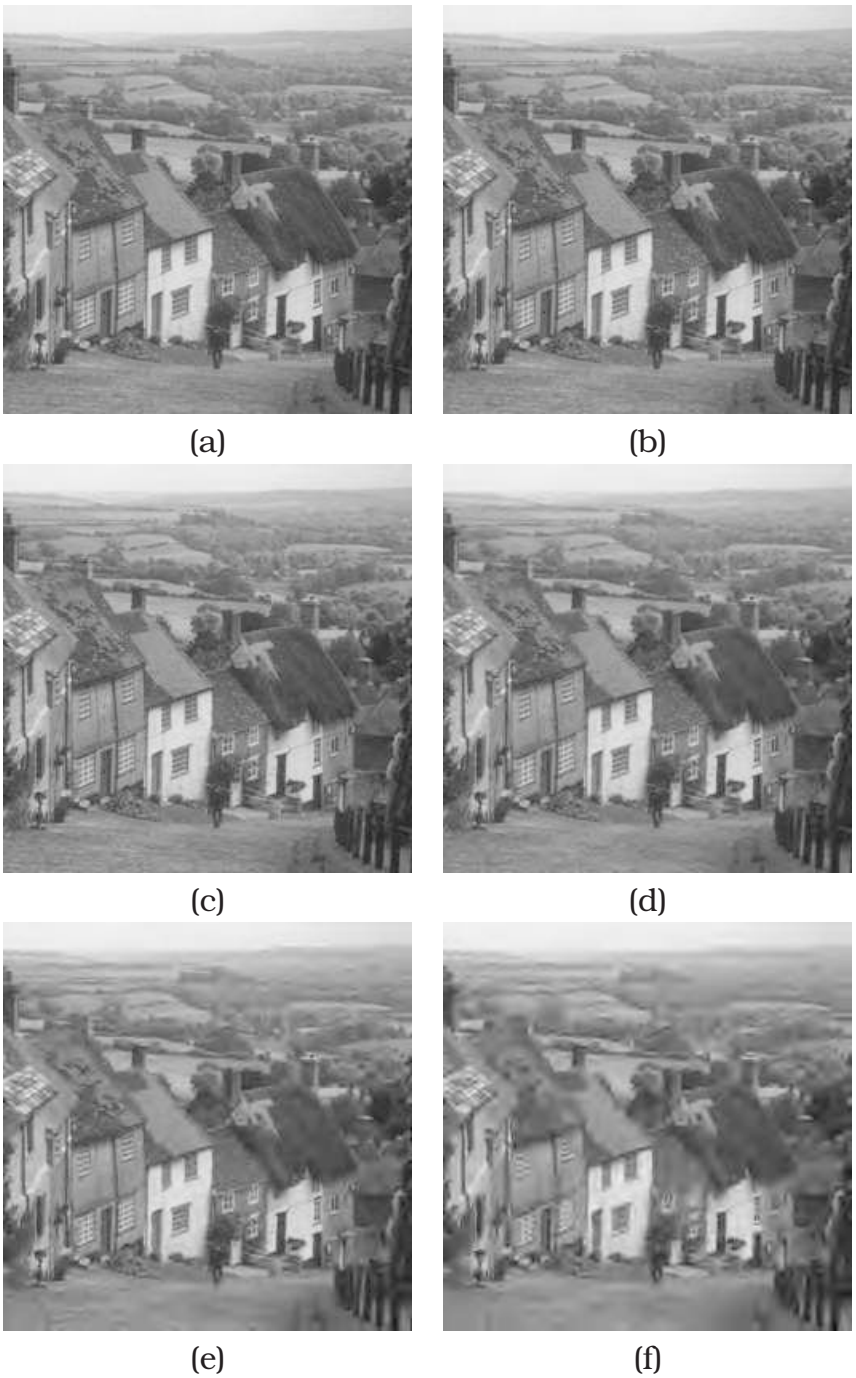


Figura A.15: *Resultados de la compresión de la imagen Goldhill con el algoritmo SPIHT. (a) Imagen original con 8 bpp (b) 1 bpp, factor de compresión 8:1, PSNR = 37,34 dB (c) 0,5 bpp, factor de compresión 16:1, PSNR = 33,26 dB (d) 0,25 bpp, factor de compresión 32:1, PSNR = 30,48 dB (e) 0,125 bpp, factor de compresión 128:1, PSNR = 28,38 dB (f) 0,0625 bpp, factor de compresión 512:1, PSNR = 26,55 dB.*

Bibliografía

[1] R. Acar and C. R. Vogel. Analysis of total variation penalty methods for ill-posed problems. *Inverse Problems*, 10:1217–1229, 1994.

[2] M. D. Adams. The JPEG-2000 still image compression standard. iso/iec jtc 1/sc 29/wg 1 n 2412. *available at http://www.ece.uvic.ca/˜mdadams*, pages 1–15, September 2001.

[3] V. R. Algazi and Jr R. R. Estes. Analysis based coding of image transform and subband coefficients. CIPIC, Center for Image Processing and Integrated Computing University of California, Davis, 1995.

[4] M. Antonini, M. Barlaud, P. Mathieu, and I. Daubechies. Image coding using wavelet transform. *IEEE Trans. Image Processing*, 1:205–220, 1992.

[5] F. Arandiga, J. Baccou, M. Doblas, and J. Liandrat. Image compression based on a multi-directional map-dependent algorithm. *Appl. Comput. Harmon. Anal.*, 23(2):181–197, 2007.

[6] F. Aràndiga, A. M. Belda, and P. Mulet. Point-value WENO multiresolution applications to stable image compression. *J. Sci. Comput.*, 43(2):158–182, 2010.

[7] F. Arandiga, A. Cohen, R. Donat, N. Dyn, and B. Matei. Approximation of piecewise smooth functions and images

by edge-adapted (ENO-EA) nonlinear multiresolution techniques. *Appl. Comput. Harmon. Anal.*, 24(2):225–250, 2008.

[8] F. Aràndiga and R. Donat. Nonlinear multiscale decompositions: the approach of A. Harten. *Numer. Algorithms*, 23:175–216, 2000.

[9] F. Aràndiga, R. Donat, and A. Harten. Multiresolution based on weighted averages of the hat function I: Linear reconstruction operators. *SIAM J. Numer. Anal.*, 36:160–203, 1999.

[10] B. L. Bihari and A. Harten. Application of generalized wavelets: an adaptive multiresolution scheme. *J. Comput. Anal. Math.*, 61:275–321, 1995.

[11] C. Bloom. Solving the problems of context modeling. http://www.cbloom.com/papers/, 1998.

[12] D. C. Botero, N. Londoño, and C. Posada. Técnicas de compresión de imágenes. Programa de Ingeniería Biomecánica EIA-CES, 2006.

[13] A. Brown. Preservación digital: compresión de imágenes. The National Archives DPGN-05, 2003.

[14] R. H. Chan, T. F. Chan, and H. M. Zhou. Continuation method for total variation denoising problems. Technical Report 95-18, University of California, Los Angeles, 1995.

[15] T. F. Chan, G. H. Golub, and P. Mulet. A nonlinear primal-dual method for total variation-based image restoration. *SIAM J. Sci. Comput.*, 20(6):1964–1977 (electronic), 1999.

[16] T. F. Chan and P. Mulet. Iterative methods for total variation image restoration. In Singapore Springer, editor, *Iterative methods in scientific computing*, 1997.

[17] T. F. Chan and P. Mulet. On the convergence of the lagged diffusivity fixed point method in total variation image restoration. *SIAM J. Numer. Anal.*, 36(2):354–367 (electronic), 1999.

[18] I. Ekeland and R. Temam. *Analyse convexe et problèmes variationnels*. Dunod, 1974. Collection Études Mathématiques.

[19] J. A. Fernández. Estudio comparativo de las técnicas de procesamiento digital de imágenes. Master's thesis, Universitat Politécnica de Madrid, 1999.

[20] Y. Fisher. *Fractal Image Compression*. The San Diego Super Computer Center University of California San Diego (1992).

[21] B. J. García, E. Mancilla, and A. R. Montes. Optimización de la transformada wavelet discreta. Universidad Complutense de Madrid, 2005.

[22] E. Giusti. *Minimal Surfaces and Functions of Bounded Variations*. Birkhäuser, 1984.

[23] gmplib.org. *The GNU Multiple Precision Arithmetic Library*. http://gmplib.org/.

[24] G. Golub and C. van Loan. *Matrix computations,* 2^{nd} *ed.* The John Hopkins University Press, 1989.

[25] V. González. *Compresión reversible y transmisión de imágenes*. PhD thesis, Universidad de Almería, 2000.

[26] A. Harten. Discrete multiresolution analysis and generalized wavelets. *J. of Applied Num. Math.*, 12:153–193, 1993.

[27] A. Harten. Multiresolution representation of data: a general framework. *SIAM J. Numer. Anal.*, 33:1205–1256, 1996.

[28] A. Harten. Multiresolution representation and numerical algorithms: a brief review. In *Parallel numerical algorithms (Hampton, VA, 1994)*, volume 4 of *ICASE/LaRC Interdiscip. Ser. Sci. Eng.*, pages 289–322. Kluwer Acad. Publ., Dordrecht, 1997.

[29] A. Harten, B. Engquist, S. Osher, and S.R. Chakravarthy. Uniformly high-order accurate essentially nonoscillatory schemes. III. *J. Comput. Phys.*, 71(2):231–303, 1987.

[30] P. G. Howard. *The Design and Analysis of Efficient Lossless Data Compression Systems*, 1993.

[31] P. J. Huber. Robust estimation of a location parameter. *Ann. Math. Statist.*, 35:73–101, 1964.

[32] D. A. Huffman. A method for the construction of minimum redundancy codes. *Proc. IRE*, 40(10):1098–1101, 1949.

[33] G.-S. Jiang and C.-W. Shu. Efficient implementation of weighted ENO schemes. *J. Comput. Phys.*, 126(1):202–228, 1996.

[34] C. T. Kelley. *Iterative Methods for Linear and Nonlinear Equations*, volume 16 of *Frontiers in Applied Mathematics*. SIAM, 1995.

[35] L. G. Kraft. A device for quantizing, grouping and coding amplitude modulated pulses. PhD thesis, MIT, Cambridge (1949).

[36] X.-D. Liu, S. Osher, and T. Chan. Weighted essentially non-oscillatory schemes. *J. Comput. Phys.*, 115(1):200–212, 1994.

[37] J. Malý and P. Rajmic. Dwt-spiht image codec implementation. Brno University of Technology, Brno, Czech Republic, 2003.

[38] J. Max. Quantizing for minimum distortion. *IRE Trans. Info. Theory*, 6:7–12, 1960.

[39] A. Moffat. An improved data structure for cummulative probability tables. *Software-Practice and Experience*, 29(7):647–659, 1999.

[40] A. Moffat, R.Ñeal, and I.H. Witten. Arithmetic coding revisted. *ACM Transactions on Information Systems*, 16(3):256–294, 1998.

[41] P. Mulet P. Blomgren, T. F. Chan and C. K. Wong. Total variation image restoration: Numerical methods and extensions. *Los Angeles*, 1997.

[42] C. C. Paige and M. A. Saunders. Solutions of sparse indefinite systems of linear equations. *SIAM J. Numer. Anal.*, 12(4):617–629, 1975.

[43] K. H. Park and H. W. Park. Region-of-interest coding based on set partitioning in hierarchical trees. *Circuits and Systems for Video Technology, IEEE Transactions on*, 12(2):106–113, 2002.

[44] J. A. Pérez. *Codificación fractal de imágenes*, 1998.

[45] T. Rockafellar. *Convex analysis*. SIAM, 1970.

[46] L. Rudin, S. Osher, and E. Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D*, 60:259–268, 1992.

[47] A. Said and W. A. Pearlman. *Reversible Image compression via multiresolution representation and predictive coding*, 1993.

[48] A. Said and W. A. Pearlman. A new fast and efficient image codec based on set partitioning in hierarchical trees. *IEEE Transactions on Circuits and Systems for Video Technology*, 6:243–250, 1996.

[49] C.E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27:379–423, 1948.

[50] J. Shapiro. Embedded image coding using zerotrees of wavelet coefficient. *IEEE Signal Processing*, 41(3445-3462), 1993.

[51] A. Skodras, C. Christopoulos, and T. Ebrahimi. The jpeg 2000 still image compression standard. IEEE SIGNAL PROCESSING MAGAZINE 1053-5888/01, 2001.

[52] H. I. Sánchez, L. A. Luján, and E. Rodríguez. *Compresión de imágenes con fractales*, 2008.

[53] D. Taubman. *High Perfomance Scalable Image Compression With Ebcot*, 1998. University of New South Wales, Sydney, Australia.

[54] D. Taubman and M. Marcellin. *JPEG-2000: Image Compression Fundamentals, Standards, and Practice*. Springer, 2001.

[55] A.N. Tikhonov and V. Y. Arsenin. *Solutions of Ill-Posed Problems*. John Wiley, New York, 1977.

[56] C. R. Vogel and M. E. Oman. Iterative methods for total variation denoising. *SIAM J. Sci. Statist. Comput.*, 17:227–238, 1996.

[57] M. J. Weinberger, G. Seroussi, and G. Sapiro. LOCO-I: A low complexity, context- based, lossless image compression algorithm. Hewlett-Packard Laboratories, Palo Alto, CA 94304, 1996.

[58] M. J. Weinberger, G. Seroussi, and G. Sapiro. The LOCO-I lossless image compression algoritm: Principles and standarization into JPEG-LS. *IEEE Transactions on Image Processing*, 9(8):1309–1324, 2000.

[59] X. Wu. Context-based, adaptive, lossless image coding. *IEEE Trans. Communications*, 45(4), 1997.