Eliminación de Ruido en Imágenes Digitales usando Algoritmos de Multirresolución Lineales y No Lineales

Autora: Maria Domínguez González
Directores:
Sergio Amat Plata
Sonia Busquier Sáez
Juan Carlos Trillo Moya

Índice de Tablas

Índice de figuras

Índice

1. Introducción

Hoy en día vivimos en un entorno rodeados de nuevas tecnologías y muchas de ellas ya se consideran como parte de nuestra vida diaria. Entre todas las numerables hacemos referencia a la fotografía digital. Aunque en algunos aspectos siguen conviviendo los dos tipos de fotografía, analógica y digital, la fotografía digital cubre la mayor parte de los campos para el uso diario de la misma tanto a nivel de usuario como profesional gracias a las prestaciones que nos ofrece en comparación con la analógica. A lo largo de este proyecto se desarrolla un estudio para lograr la eliminación, total o parcial, de ruido en las fotografías digitales. Se van a utilizar técnicas de multirresolución, que consisten en una descomposición de una señal en escalas. Se usarán diferentes algoritmos de multirresolución, tanto lineales como no lineales ([11][12][4][3]).

En primer lugar vamos a proceder a explicar de manera gráfica para que sea más fácilmente comprensible como se realiza el proceso de multirresolución que se aplica a la matriz (fotografía) original dependiendo del número de niveles aplicados. Supongamos que ya disponemos de un algoritmo en una dimensión, el cual dado un vector discreto lo descompone en una parte correspondiente a valores significativos y otra a detalles. Esta descomposición contiene la misma información que el vector original. Aplicando el algoritmo 1-dimensional primero a cada una de las filas y después a cada una de las columnas tenemos una descomposición 2-dimensional ([1][2]).
Para la matriz original, el primer nivel de multirresolución se realiza de la siguiente manera:

La matriz en la esquina superior izquierda corresponde con la matriz original, la segunda matriz corresponde con la aplicación del algoritmo 1-dimensional a cada una de las filas. Hemos representado mediante puntos negros los valores significativos y con círculos verdes los detalles horizontales.
La tercera matriz, situada debajo, corresponde con el resultado de aplicar el algoritmo 1-dimensional a cada una de las columnas del resultado de la paso anterior. Quedan representados con puntos negros los valores significativos, con círculos verdes los detalles horizontales, con estrellas rosas los detalles verticales y con triángulos azules los detalles mixtos. Por tanto la última matriz está formada por los valores significativos, los detalles verticales, detalles horizontales y detalles mixtos aunque se encuentren todos entremezclados (ver Figura 1).

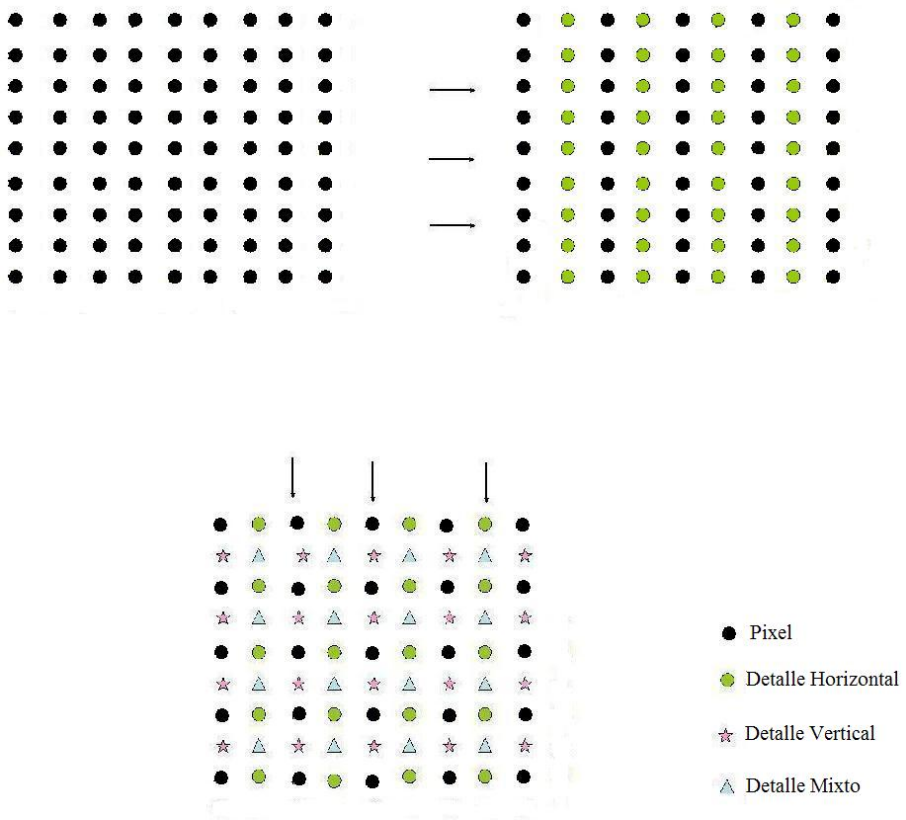

Figura 1: Pasos del algoritmo de multirresolución: imagen original, algoritmo 1D aplicado a las filas y algoritmo 1D aplicado a las columnas

El siguiente paso sería recolocar la matriz agrupando por tipos cada uno de los valores y detalles. El resultado de la matriz reordenada puede verse en la Figura 2.

Para realizar el siguiente nivel de multirresolución procederíamos a realizar los mismos pasos tomando como matriz inicial la submatriz superior izquierda, es decir, la que está formada por los valores significativos obtenidos. Volveríamos a localizar los detalles horizontales (de esa submatriz), a continuación los verticales y por tanto los mixtos y reordenaríamos de nuevo la nueva matriz obtenida.

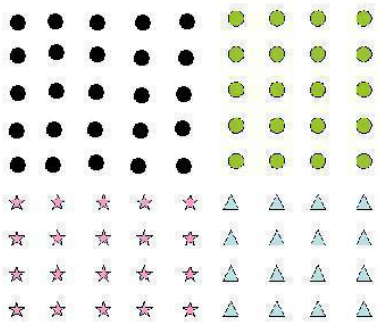

Figura 2: Versión de multirresolución de la imagen una vez reorganizados los coeficientes

Este proceso se repite tantas veces como niveles de multirresolución hayamos pedido que realice.

Como ejemplo de 3 niveles de multirresolución se muestra la Figura 3, donde se podría observar como se va subdividiendo primero la imagen inicial en cuatro, luego la superior izquierda (que sería la que contiene los valores significativos) en otras cuatro y así hasta tres veces, obteniendo como resultado final una versión a menor resolución de la imagen original, en la esquina superior izquierda, más los diferentes detalles asociados a cada una de las escalas.

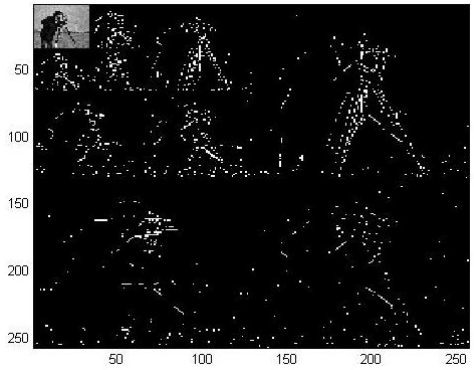

Figura 3: Versión de multirresolución con tres escalas de la imagen del cámara

Vamos a exponer brevemente la manera en que se realiza el truncamiento de las imágenes digitales durante el proceso de eliminación de ruido. Vamos a trabajar con dos métodos para realizar el truncamiento: truncamiento fuerte y truncamiento suave.

El truncamiento fuerte se realiza de la siguiente manera: Se considera un valor del parámetro de truncamiento ϵ, y se compara con el valor absoluto del detalle, si dicho valor es menor o igual que ϵ, el detalle pasa directamente a tener un valor 0, si el detalle es mayor que ϵ mantiene su valor, es decir para un detalle Δ_i^k :

$$\hat{\Delta}_i^k = \begin{cases} 0 & |\Delta_i^k| \leq \epsilon \\ \Delta_i^k & \text{en otro caso} \end{cases}$$

En el truncamiento suave se toma el valor del parámetro de truncamiento ϵ y se compara con el valor del detalle. Si el valor del detalle es menor que ϵ, directamente se le asigna al detalle el valor 0 y si el detalle es mayor que el valor de ϵ, se asigna al detalle el valor resultante de restar el valor absoluto del detalle menos ϵ. Es decir, para un detalle Δ_i^k:

$$\hat{\Delta}_i^k = \eta_\epsilon\left(\Delta_i^k\right) = sgn\left(\Delta_i^k\right) * \text{máx}\left(abs(\Delta_i^k) - \epsilon, 0\right).$$

De los dos métodos explicados, el más utilizado y que ofrece buenos resultados en la práctica es el truncamiento suave ([7][11][9]).

Además de tener en cuenta la elección del tipo filtro para el truncamiento hay que seleccionar el método para calcular el valor del parámetro de truncamiento, ϵ. La elección más extendida es la que se conoce como universal thresholding y combina el filtro ddo por el truncamiento suave con la elección de parámetro de truncamiento ϵ de acuerdo con la siguiente fórmula:

$$\begin{aligned} \epsilon_1^k &= \sigma\sqrt{2\ln(M_1^k)} \\ \epsilon_2^k &= \sigma\sqrt{2\ln(M_2^k)} \\ \epsilon_3^k &= \sigma\sqrt{2\ln(M_3^k)} \end{aligned}$$

donde M_1^k, M_2^k, M_3^k se corresponde con la dimensión de cada una de las matrices de detalles. Esta manera de proceder fue introducida por Donoho y Johnstone en [9].

A continuación se muestra una gráfica donde se puede observar la diferencia entre los dos filtros explicados anteriormente (ver Figura 4). Para ello vamos a expresar los filtros como $F(\Delta_i^k) = \lambda \Delta_i^k$, donde Δ_i^k representa a los diferentes detalles y λ es un parámetro que depende del filtro utilizado. Para el truncamiento fuerte, representado en la gráfica mediante la linea de color verde, λ coincide con:

$$\lambda = \begin{cases} 0 & |d| \leq \epsilon \\ 1 & |d| > \epsilon \end{cases}$$

Para el truncamiento suave, representado en la gráfica mediante la linea azul, el valor de λ coincide con:

$$\lambda = \begin{cases} 0 & |d| \leq \epsilon \\ \frac{1-sgn(d)\epsilon}{d} & |d| > \epsilon \end{cases}$$

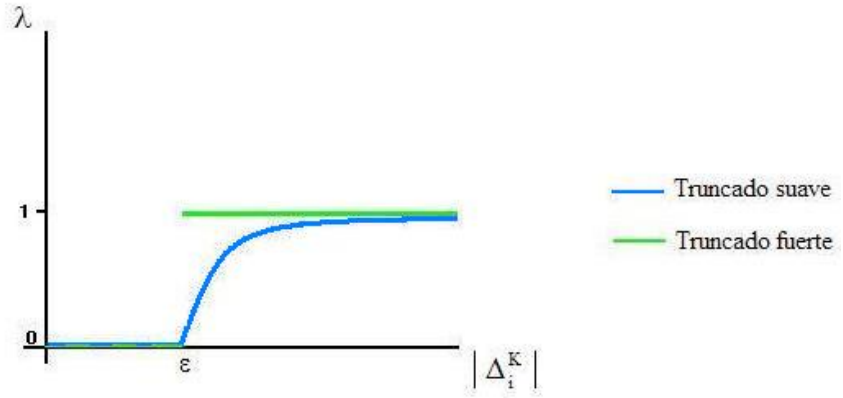

Figura 4: Comportamiento de λ para los filtros de truncación fuerte y suave dependiendo del tamaño de los detalles

La memoria está organizada de la siguiente forma: En la sección 2 se presenta con cierto detalle la teoría empleada para la eliminación de ruido

en imágenes digitales, incluyendo algunos experimentos numéricos. En la sección 3 se explica el desarrollo de los programas Matlab utilizados en el proyecto, los cuales son añadidos al final de la sección. Y por último en la sección 4 se presentan las conclusiones extraídas del proyecto.

2. Eliminación de ruido en imágenes digitales usando multirresoluciones lineales y no lineales

Dados unos datos f^L donde L representa un nivel de resolución, una representación de multirresolución de f^L es cualquier secuencia del tipo $\{f^0, d^1, ..., d^L\}$ donde f^k es una aproximación de f^L en la resolución $k < L$ y d^{k+1} representa los detalles requeridos para conseguir f^{k+1} a partir de f^k. Las representaciones de multirresolución lineales de los datos, como las descomposiciones 'wavelet', son multirresoluciones que implican operadores lineales de inter-resolución.

La eficiencia de las descomposiciones de multirresolución lineal está limitada por la presencia de bordes o ejes en las imágenes. Los coeficientes numéricamente significativos d_j^k son principalmente aquellos para los cuales el soporte de la función wavelet asociada cruza las discontinuidades.

El entorno de resolución de Harten ha sido desarrollado para incorporar un tratamiento específico adaptado a las singularidades. La ventaja de este marco general está en su flexibilidad, donde el operador de reconstrucción juega un papel principal. Los niveles de resolución discretos son conectados por operadores de inter-resolución, llamados decimación(desde un nivel de resolución(k) a un nivel de resolución menor($k-1$)) y predicción(de menor resolución a mayor resolución). Estos operadores entre escalas están directamente relacionados con los operadores de discretización y de reconstrucción, los cuales actúan entre un espacio funcional adecuado (el cual depende de los datos discretos con los que se trabaja) y cada uno de los niveles discretos (donde quedan representadas las secuencias f^k). Este entorno de Harten hace posible considerar técnicas de reconstrucción dependientes de los datos, las cuales son interesantes para obtener un índice óptimo de reducción de ruido. Se pueden considerar diferentes tipos de algoritmos de multirresolución dependiendo del operador de discretización lineal que produce los datos. En este proyecto se considera la multirresolución por valores puntuales porque es en este entorno donde se obtienen reconstrucciones de manera más fácil. Usando funciones primitivas podemos obtener las reconstrucciones asociadas al caso de discretización por medias en celda.
En el caso de valores puntuales el operador de reconstrucción equivale a hacer interpolación. Normalmente se consideran interpolaciones independientes de los datos. Una opción para obtener una adaptación cerca de las singularidades es considerar interpolación no lineal(dependiente de los datos). Teóricamente usando los esquemas ENO (esencialmente no oscilatorios) podemos obtener una alta exactitud en todos los intervalos sin ninguna singularidad.

El método de interpolación ENO utiliza reconstrucciones polinómicas a trozos basadas en un procedimiento de selección del conjunto de puntos para interpolar (stencil) que se mueve lejos de las singularidades. Los resultados numéricos, en la compresión de imágenes, revelan que estas reconstrucciones no lineales superan ampliamente a las reconstrucciones lineales clásicas en el caso de imágenes geométricas, pero no proporcionan mejoras sustanciales para las imágenes reales que contienen texturas adicionales. A lo largo de este estudio se mostrarán conclusiones similares que han sido obtenidas en la eliminación del ruido en las imágenes digitales.
Se considera también la transformación de multirresolución que resulta de las técnicas de reconstrucción PPH (*piecewise polynomial harmonic*) . Se cuenta con el hecho de que su utilización conduzca a una mejora, especialmente cuando la textura está presente, igual que sucede en el caso de la compresión. La meta del proyecto es proporcionar diferentes comparaciones entre interpolaciones lineales y no lineales para poder observar los resultados que se obtienen en la eliminación del ruido de las imágenes. Se realizan pruebas con cuatro métodos diferentes: primeramente con la interpolación clásica de Lagrange ([8]), en segundo lugar con una interpolación ENO dependiente de los datos ([4][14][15]), en tercer lugar se utiliza ENO jerárquico (otra versión de la interpolación ENO que utiliza una selección jerárquica del stencil), y finalmente con una reconstrucción no lineal PPH ([3]).

2.1. El entorno de multirresolución por valores puntuales

Se considera el conjunto de redes anidadas en el intervalo [0,1] dado por:

$$X^k = \{x_j^k\}_{j=0}^{J_k}, \quad x_j^k = jh_k, \quad h_k = 2^{-k}/J_0, \quad J_k = 2^k J_0,$$

donde J_0 es un entero fijo. La discretización por valores puntuales viene dada por

$$D_k : \begin{cases} C([0,1]) & \to V^k \\ f & \mapsto f^k = (f_j^k)_{j=0}^{J_k} = (f(x_j^k))_{j=0}^{J_k} \end{cases} \tag{1}$$

donde V^k es el espacio de las secuencias reales de dimensión $J^k + 1$. Un operador de reconstrucción para esta discretización es cualquier operador R_k tal que

$$R_k : V^k \to C([0,1]); \quad \text{y satisface} \quad D_k R_k f^k = f^k, \tag{2}$$

lo cual significa que

$$(R_k f^k)(x_j^k) = f_j^k = f(x_j^k). \tag{3}$$

En otras palabras $(R_k f^k)(x)$ es una función continua que interpola los datos f^k en X^k.

Si se escribe $(R_k f^k)(x) = I_k(x; f^k)$, entonces uno puede definir las transformadas directa (4) e inversa (5) de la multirresolución como

$$f^L \to Mf^L \begin{cases} \text{Do} \quad k = L, \ldots, 1 & \\ \quad f_j^{k-1} = f_{2j}^k & 0 \le j \le J_{k-1}, \\ d_j^k \quad = f_{2j-1}^k - I_{k-1}(x_{2j-1}^k; f^{k-1}) & 1 \le j \le J_{k-1}. \end{cases} \tag{4}$$

y

$$Mf^L \to M^{-1}Mf^L \begin{cases} \text{Do} \quad k = 1, \ldots, L & \\ f_{2j-1}^k = I_{k-1}(x_{2j-1}^k; f^{k-1}) + d_j^k & 1 \le j \le J_{k-1}, \\ \quad f_{2j}^k = f_j^{k-1} & 0 \le j \le J_{k-1}. \end{cases} \tag{5}$$

Las técnicas de interpolación más usuales son los polinomios.

2.1.1. *Técnicas de reconstrucción lineal:interpolación de Lagrange*

Tomando $\mathcal{S}$ como el conjunto

$$\mathcal{S} = \mathcal{S}(r, s) = \{-s, -s+1, \ldots, -s+r\}, \quad r \ge s > 0, \ r \ge 1,$$

y $\{L_m(y)\}_{m \in S}$ como los polinomios interpoladores de Lagrange de grado r basados en los elementos del conjunto $S(r, s)$,

$$L_m(y) = \prod_{l=-s, l \ne m}^{-s+r} \left(\frac{y-l}{m-l}\right), \quad L_m(j) = \delta_j^m, \quad j \in \mathcal{S}.$$

La interpolación de Lagrange para

$$\{x_{j+m}^k\}_{m \in \mathcal{S}}, \text{ se escribe}$$

$$\mathcal{I}_k(x, f^k) = \sum_{m=-s}^{-s+r} f_{j+m}^k L_m\left(\frac{x - x_j^k}{h_k}\right), \quad x \in [x_{j-1}^k, x_j^k], \quad 1 \le j \le J_k.$$

Es importante observar que si $f(x) = P(x)$, donde P(x) es un polinomio de grado menor o igual que r, entonces $\mathcal{I}_k(x, f^k) = f(x)$ para $x \in [x_{j-1}^k, x_j^k]$. Lo cual significa que, para funciones suaves

$$\mathcal{I}_k(x, f^k) = f(x) + O(h_k)^{r+1}.$$

Por lo tanto, el orden del procedimiento de reconstrucción,que caracteriza su precisión, será $p = r + 1$. La situación particular para el valor $r = 2s - 1$ se corresponde con un stencil de interpolación simétrico respecto del intervalo $[x_{j-1}^k, x_j^k]$. Por ejemplo, para $s = 2(r = 3)$ se obtiene la siguiente transformada de multirresolución

$$\begin{cases} \text{Do} \quad k = L, \ldots, 1 & \\ \quad f_j^{k-1} = f_{2j}^k & 0 \le j \le J_{k-1}, \\ d_j^k = f_{2j-1}^k - (\frac{-f_{j-2}^{k-1} + 9f_{j-1}^{k-1} + 9f_j^{k-1} - f_{j+1}^{k-1}}{16}), & 1 \le j \le J_{k-1}. \end{cases} \tag{6}$$

$$\begin{cases} \text{Do} \quad k = 1, \ldots, L & \\ f_{2j-1}^k = d_j^k + (\frac{-f_{j-2}^{k-1} + 9f_{j-1}^{k-1} + 9f_j^{k-1} - f_{j+1}^{k-1}}{16}), & 1 \le j \le J_{k-1}, \\ f_{2j}^k = f_j^{k-1} & 0 \le j \le J_{k-1}. \end{cases} \tag{7}$$

Las técnicas de interpolación de Lagrange pierden mucha de su precisión en presencia de singularidades, por ejemplo en presencia de saltos en la función el error se comporta como

$$f(x) = \mathcal{I}_l(x) + O([f]).$$

2.1.2. *Técnicas de reconstrucción no lineal:interpolación ENO*

La idea de las reconstrucciones ENO es construir trozos o partes polinomiales usando datos pertenecientes a las regiones suaves de la función en la medida de lo posible. El ingrediente principal es, por lo tanto, el procedimiento de selección del stencil S (conjunto de datos usados para construir el polinomio interpolador), el cual se intenta elegir dentro de una región suave de la función $f(x)$. De forma más precisa, este proceso de selección trabaja de la siguiente manera: para cada intervalo

$$\left[x_{j-1}^k, x_j^k\right],$$

se consideran todos los posibles conjuntos S con $r \geq 2$ puntos incluyendo los puntos x^k_{j-1} y x^k_j.

Denominando el stencil ENO para el jth intervalo:

$$S_j^{ENO} = \left\{ x^k_{s_{j-1}}, x^k_{s_j}, ..., x^k_{s_j+r-1} \right\},$$

la selección de S_j^{ENO} se realiza como sigue:

(1) Selección jerárquica del stencil

```
Set s_0 = j
for  l = 0, ..., r − 2
  if  |f[x^k_{s_l−2}, ..., x^k_{s_l+l}]| < |f[x^k_{s_l−1}, ..., x^k_{s_l+l+1}]|
    s_{l+1} = s_l − 1
  else
    s_{l+1} = s_l
  end
end
s_j = s_{r−1}
```

y,

(2) Selección no jerárquica del stencil

Se elige un s_j tal que

```
Choose s_j such that
|f[x^k_{s_j−1}, ..., x^k_{s_j+r−1}])| = mín_{j−r+1≤l≤j}{|f[x^k_{l−1}, ..., x^k_{l+r−1}]|}
```

Ambos algoritmos,(1) y (2) conducen asintóticamente a conjuntos de puntos de interpolación que se mueven lejos de la discontinuidad. Consecuentemente,el orden de aproximación del operador de predicción ENO sigue siendo $r + 1$ siempre que sea posible evitar la discontinuidad.

2.1.3. *Técnicas de reconstrucción no lineal: La reconstrucción PPH*

En esta sección se describe un esquema de interpolación no lineal de cuarto orden dependiente de los datos, el cual está basado en una interpolación a trozos polinomial denominada PPH. Esta técnica de interpolación no lineal conduce a un operador de reconstrucción con varias características deseables. Primero, cada parte está construida con un stencil fijo centrado de cuatro puntos. Segundo, la reconstrucción es tan exacta como su equivalente lineal en las regiones suaves. En tercer lugar, la exactitud se reduce cerca de las singularidades, pero no se pierde completamente como ocurre en su contraparte lineal.
A continuación se describe el operador de reconstrucción PPH, el cual denotamos como R_k^P. Al igual que con todas las otras técnicas de interpolación, dado $x \in \mathbb{R}$, tomemos j tal que

$$x \in [x_{j-1}^k, x_j^k]$$

Entonces, $(R_k^P f^k)(x) = \tilde{P}_j(x, f^k)$, donde $\tilde{P}_j(x, f^k)$ es un polinomio construido a partir de los datos centrados $f_{j-2}^k, f_{j-1}^k, f_j^k, f_{j+1}^k$ y tal que $\tilde{P}_j(x_{j-1}^k, f^k) = f_{j-1}^k$ y $\tilde{P}_j(x_j^k, f^k) = f_j^k$. En lo que sigue suprimiremos el superíndice k por claridad. Se considera el conjunto de puntos $f_{j-2}, f_{j-1}, f_j, f_{j+1}$. Y vamos a describir la predicción para el punto medio $f_{j-\frac{1}{2}}$. Según lo expuesto arriba,si la función no tiene ninguna singularidad de salto en el intervalo $[x_{j-2}, x_{j+1}]$ una interpolación centrada proporciona una buena aproximación. No obstante, cuando la señal muestra singularidades la aproximación pierde su exactitud. En lo siguiente se discute la modificación propuesta cuando se detecta una singularidad en $[x_j, x_{j+1}]$. Supongamos que la diferencia dividida $f[x_{j-1}, x_j, x_{j+1}]$ es mayor o igual que $f[x_{j-2}, x_{j-1}, x_j]$ en valor absoluto. Esto indica la posible presencia de una singularidad en un punto $x_d \in [x_j, x_{j+1}]$. Se considera el trozo polinomial para $[x_{j-1}, x_j]$ escrito como,

$$P_j(x) = a_0 + a_1(x - x_{j-\frac{1}{2}}) + a_2(x - x_{j-\frac{1}{2}})^2 + a_3(x - x_{j-\frac{1}{2}})^3. \tag{8}$$

Para un esquema lineal centrado las cuatro condiciones de interpolación en los puntos x_{j-2}, x_{j-1}, x_j y x_{j+1} son

$$\begin{cases} a_0 - a_1\frac{3}{2}h + a_2(\frac{3}{2}h)^2 - a_3(\frac{3}{2}h)^3 &= f_{j-2}, \\ a_0 - a_1\frac{1}{2}h + a_2(\frac{1}{2}h)^2 - a_3(\frac{1}{2}h)^3 &= f_{j-1}, \\ a_0 + a_1\frac{1}{2}h + a_2(\frac{1}{2}h)^2 + a_3(\frac{1}{2}h)^3 &= f_j, \\ a_0 + a_1\frac{3}{2}h + a_2(\frac{3}{2}h)^2 + a_3(\frac{3}{2}h)^3 &= f_{j+1}. \end{cases} \tag{9}$$

Es fácil comprobar que

$$a_1 = \frac{f_{j-2} - 27f_{j-1} + 27f_j - f_{j+1}}{24h}.$$

De ese modo,el sistema de ecuaciones anterior es equivalente a

$$\begin{cases} a_0 - a_1\frac{3}{2}h + a_2(\frac{3}{2}h)^2 - a_3(\frac{3}{2}h)^3 & = f_{j-2}, \\ a_0 - a_1\frac{1}{2}h + a_2(\frac{1}{2}h)^2 - a_3(\frac{1}{2}h)^3 & = f_{j-1}, \\ a_0 + a_1\frac{1}{2}h + a_2(\frac{1}{2}h)^2 + a_3(\frac{1}{2}h)^3 & = f_j, \\ a_1 & = \frac{f_j - 27f_{j-1} + 27f_j - f_{j+1}}{24h}. \end{cases}$$

Se introducen las diferencias divididas definidas por $e_{j-\frac{3}{2}} = f[x_{j-2}, x_{j-1}]$, $e_{j-\frac{1}{2}} = f[x_{j-1}, x_j]$, $e_{j+\frac{1}{2}} = f[x_j, x_{j+1}]$, $D_{j-1} = f[x_{j-2}, x_{j-1}, x_j]$ y $D_j = f[x_{j-1}, x_j, x_{j+1}]$. Después de algunas manipulaciones algebráicas se llega fácilmente a

$$a_1 = \frac{-e_{j-\frac{3}{2}} + 13e_{j-\frac{1}{2}}}{12} - \frac{1}{12}\frac{D_{j-1} + D_j}{2}h.$$

En particular, se observa que en presencia de una discontinuidad de salto en $[x_j, x_{j+1}]$, $a_1 = O(\frac{1}{h})$, ya que $D_j = O(\frac{1}{h^2})$. Este comportamiento es debido a la mala aproximación de la reconstrucción en presencia de discontinuidades. Destacando que D_{j-1} sigue siendo de orden $O(1)$, se sustituye la media aritmética

$$\frac{D_{j-1}+D_j}{2}$$

por la media armónica

$$\frac{2D_{j-1}D_j}{D_{j-1}+D_j}$$

Siempre que $D_{j-1}D_j > 0$. Esta nueva cantidad está también acotada, y se obtiene así la siguiente expresión modificada para a_1,

$$\tilde{a}_1 := \frac{-e_{j-\frac{3}{2}} + 13e_{j-\frac{1}{2}}}{12} - \frac{1}{12}\frac{2D_{j-1}D_j}{D_{j-1} + D_j}h.$$

Por un lado, debido al hecho que

$$|2\frac{D_{j-1}D_j}{D_{j-1} + D_j}| \leq 2\,\text{mín}(|D_{j-1}|, |D_j|) = O(1), \tag{10}$$

asumiendo $D_{j-1}D_j > 0$, la media armónica está bien adaptada a la presencia de singularidades porque, cuando $|D_{j-1}|$ es $O(1)$ y $|D_j|$ es $O(\frac{1}{h^2})$, la media armónica permanece siendo $O(1)$, y en consecuencia, $\tilde{a}_1 = O(1)$. Por otro lado, en las regiones suaves $a_1 - \tilde{a}_1 = O(h^3)$, ya que la diferencia entre la media armónica y la aritmética original es $O(h^2)$. Por consiguiente, la reconstrucción es de cuarto orden, y en particular $(f_{j-\frac{1}{2}} - \tilde{P}_j(x_{j-\frac{1}{2}}) = O(h^4))$. Si $D_{j-1}D_j \leq 0$ la media armónica no está bajo control, ya que en algunos casos $D_{j-1} + D_j \approx 0$. Se considera por lo tanto en esta situación

$$\tilde{\tilde{a}}_1 := \frac{-e_{j-\frac{3}{2}} + 13e_{j-\frac{1}{2}}}{12}. \tag{11}$$

Entonces se tiene $a_1 - \tilde{\tilde{a}}_1 = O(h)$. La reconstrucción está adaptada en este caso a la presencia de singularidades aunque la exactitud se reduce hasta grado dos. El operador de reconstrucción PPH estará dado por la expresión polinomial de la ecuación (8) con los nuevos coeficientes $\tilde{a_0}$, $\tilde{a}_1$, $\tilde{a}_2$ y $\tilde{a}_3$ si $D_{j-1}D_j > 0$, o con $\tilde{\tilde{a}}_0$, $\tilde{\tilde{a}}_1$, $\tilde{\tilde{a}}_2$ y $\tilde{\tilde{a}}_3$ si $D_{j-1}D_j \leq 0$. Es fácil comprobar que la predicción se convierte en

$$\begin{aligned} f_{j-\frac{1}{2}} &\approx \frac{-f_{j-2} + 18f_{j-1} - 9f_j}{8} - \frac{1}{8}12\tilde{a}_1 h \\ &= \frac{f_j + f_{j-1}}{2} - \frac{1}{4}\frac{(f_j - 2f_{j-1} + f_{j-2})(f_{j+1} - 2f_j + f_{j-1})}{(f_{j+1} - f_j - f_{j-1} + f_{j-2})}, \end{aligned} \tag{12}$$

ó

$$\begin{aligned} f_{j-\frac{1}{2}} &\approx \frac{-f_{j-2} + 18f_{j-1} - 9f_j}{8} - \frac{1}{8}12\tilde{\tilde{a}}_1 h \\ &= \frac{f_j + f_{j-1}}{2}, \end{aligned} \tag{13}$$

respectivamente. Por razones de simetría la modificación es la misma cuando la singularidad pertenece a $[x_{j-2}, x_{j-1}]$. En este método, la no linealidad aparece en el proceso de selección entre una interpolación de Lagrange y una interpolación no lineal así como en la propia interpolación. Al contrario que en la interpolación ENO, la reconstrucción PPH siempre usa un stencil centrado. La meta de los operadores de reconstrucción no lineales es mejorar la exactitud de la predicción en los alrededores de las singularidades aisladas. Así se espera un mejor tratamiento de las singularidades correspondientes a los ejes de las imágenes y, por lo tanto, una representación de multirresolución más dispersa.

2.1.4. *Producto Tensor*

Para generalizar los algoritmos anteriormente definidos para secuencias de datos dos dimensionales imágenes se ha usado la estrategia del producto tensor, la cual presentamos brevemente a continuación. Representando el array bidimensional $f = (f^L_{i,j})^{J_L}_{(i,j)=0}$ por la matriz $A = A^L$, la representación de multirresolución de A^L es

$$Mf = \{A^0, (\{\Delta^1_i\}^3_{i=1}, \ldots, \{\Delta^L_i\}^3_{i=1})\}$$

con

$$\begin{aligned}
A^{k-1}_{i,j} &= A^k_{2i,2j}, \quad 0 \le i,j \le J_{k-1} \\
(\Delta^k_1) &= E^k_{2i-1,2j-1}, \quad 1 \le i,j \le J_{k-1} \\
(\Delta^k_2) &= E^k_{2i-1,2j}, \quad 1 \le i \le J_{k-1}, 0 \le j \le J_{k-1} \\
(\Delta^k_3) &= E^k_{2i,2j-1}, \quad 0 \le i \le J_{k-1}, 1 \le i \le J_{k-1}
\end{aligned}$$

donde E^k son errores de interpolación:

$$\begin{aligned}
E^k_{2i-1,2j-1} &= A^k_{2i-1,2j-1} - (D_k R_{k-1} A^{k-1})_{2i-1,2j-1}, \\
E^k_{2i-1,2j} &= A^k_{2i-1,2j} - (D_k R_{k-1} A^{k-1})_{2i-1,2j}, \\
E^k_{2i,2j-1} &= A^k_{2i,2j-1} - (D_k R_{k-1} A^{k-1})_{2i,2j-1}.
\end{aligned}$$

Clasicamente, un paso de la descomposición de multirresolución es presentada de la siguiente forma matricial

$$A^k \leftrightarrow \left(\begin{array}{c|c} A^{k-1} & \Delta^k_2 \\ \hline \Delta^k_3 & \Delta^k_1 \end{array} \right)$$

2.2. Comparación entre las reconstrucciones lineal, ENO y PPH en algoritmos separables de multirresolución aplicados a la eliminación de ruido en imágenes digitales

El objetivo de esta sección es comparar los algoritmos lineales y no lineales para la eliminación de ruido. La comparación está basada en el *PSNR*, indicador comúnmente utilizado para medir la calidad de una imagen reconstruida mediante comparación con la original.

A partir de una imagen original, se le añadirá ruido y después se estudiará qué método da un PSNR más alto y por tanto elimina mayor cantidad de dicho ruido. Se consideran reconstrucciones de valores puntuales usando un stencil de cuatro puntos. Para el caso ENO ésto significa que hay que considerar seis puntos en el procedimiento de selección.
En la frontera hemos implementado el mismo tratamiento para los diferentes algoritmos de resolución. En la frontera izquierda, se considera la predicción

$$(\frac{5}{16}f_0^{k-1} + \frac{15}{16}f_1^{k-1} - \frac{5}{16}f_2^{k-1} + \frac{1}{16}f_3^{k-1})$$

y la expresión simétrica para la frontera derecha.

2.2.1. *El truncamiento*

El proceso tiene como objetivo quitar el ruido mediante el truncamiento de los coeficientes correspondientes a los detalles mientras que se guarda los coeficientes de baja resolución sin alterarlos. Hay dos métodos de truncamiento que se usan frecuentemente. El truncamiento fuerte es utilizado de la siguiente manera:

$$\mathbf{tr}^{\epsilon}(\Delta) = \hat{\Delta}$$

con

$$\hat{\Delta}_i^k = \begin{cases} 0 & |\Delta_i^k| \leq \epsilon \\ \Delta_i^k & \text{en otro caso} \end{cases}$$

asumiendo el mismo valor de ϵ para todos los niveles de resolución utilizados. Y el truncamiento suave

$$\hat{\Delta}_i^k = \eta_\epsilon\left(\Delta_i^k\right) = sgn\left(\Delta_i^k\right) * \text{máx}\left(abs(\Delta_i^k) - \epsilon, 0\right).$$

Para obtener mejores resultados se ha utilizado el truncamiento suave variando el valor de los parámetros de truncamiento en cada banda y en cada escala:

$$\begin{aligned} \epsilon_1^k &= \sigma\sqrt{2\ln(M_1^k)} \\ \epsilon_2^k &= \sigma\sqrt{2\ln(M_2^k)} \\ \epsilon_3^k &= \sigma\sqrt{2\ln(M_3^k)} \end{aligned}$$

donde M_1^k, M_2^k, M_3^k son, respectivamente, los tamaños de las matrices Δ_1^k , Δ_2^k y Δ_3^k, $k = 1, 2, \ldots, L$, y σ^2 denota la varianza del ruido de la imagen.

La elección de ϵ_1^k está justificada por el hecho de que la banda Δ_1^k contiene la mayor parte de los detalles no malos debidos al ruido.

2.2.2. *Desarrollo de los experimentos*

Consideremos la secuencia de datos $\bar{f}^L$ conseguida a partir de la imagen original de f^L agregándole ruido blanco.
Aplicando la transformación de multirresolución inversa a la versión truncada de la multirresolución directa de $\bar{f}^L$ llegamos a

$$\hat{f}^L = M^{-1}\mathbf{tr}^{\epsilon}(Mf^L)$$

Después comparamos f^L y $\hat{f}^L$ usando la norma l_2 dada por

$$||f^L - \hat{f}^L||_{l_2}^2 \quad = \quad \frac{1}{(J_L+1)^2}\sum_i |f_i^L - \hat{f}_i^L|^2.$$

Por último se computa el *PSNR*, el cual como ya mencionamos es una medida ampliamente usada para la estimación de la calidad de la imagen reconstruida. Para una imagen de 8 bits (0-255),

$$PSNR = 20\log_{10}\left(\frac{255}{||f^L - \hat{f}^L||_{l_2}}\right)$$

Un valor de *PSNR* más alto implica una mejor calidad de la imagen.

2.2.3. *Experimentos numéricos vía producto tensor*

Se define r_{scheme} como el valor del *PSNR* entre la imagen recuperada y la original, y se calculan los coeficientes

$$R_{PPH/ENO} = \frac{r_{PPH}}{r_{ENO}} \ ,$$

$$R_{PPH/Lin} = \frac{r_{PPH}}{r_{Lin}}$$

y

$$R_{ENO/Lin} = \frac{r_{ENO}}{r_{Lin}}.$$

A continuación se muestran una serie de tablas que contienen los valores obtenidos para 4 fotografías digitales denominadas Lena (Figura 5), cameraman (Figura 6), geométrica (Figura 7) y peppers (Figura 8), cada una de ellas con unos contrastes o motivos distintos para poder observar resultados en diferentes situaciones. En las pruebas realizadas hemos tomado niveles de resolución diferentes para cada fotografía (3,4 y 5) y para cada uno de esos niveles hemos tomado tres valores de varianza de ruido.

En primer lugar se muestra la Tabla 1 para la primera fotografía Lena5. Como podemos observar los mejores resultados han sido obtenidos con el método no lineal PPH ya que las dos últimas columnas contienen valores siempre mayores que 1. Para observar las diferencias visuales hemos incluido las Figuras 9, 10, 11 y 12.

Figura 5: Lena: imagen original

A continuación se presenta la Tabla 2 para la fotografía cameraman. Las observaciones hechas anteriormente son también válidas en este caso. Volvemos a observar en las Figuras 13,14, 15 y 16 la difusión propia de los métodos de reducción de ruido.

Figura 6: Foto del cámara: imagen original

Figura 7: Foto geométrica: imagen original

Ahora observamos en la Tabla 3 los datos para la fotografía de figuras geométricas. Similares comentarios son válidos también en este caso. En la Figura 17 vemos el resultado visual de aplicar el método PPH, que vuelve a ofrecer mejores resultados.

Figura 8: Foto peppers: Imagen original

L	Ruido	$R_{ENO/LIN}$	$R_{PPH/LIN}$	$R_{PPH/ENO}$
3	3	0,988	1,011	1,023
	5	0,993	1,009	1,016
	10	0,999	1,010	1,010
4	3	0,979	1,015	1,037
	5	0,991	1,013	1,022
	10	0,998	1,007	1,009
5	3	0,963	1,009	1,048
	5	0,980	1,012	1,033
	10	0,986	1,017	1,031

Tabla 1: **Imagen original Lena5.pgm**, L niveles de multirresolución, $R_{PPH/ENO} = \frac{r_{PPH}}{r_{ENO}}$, $R_{PPH/Lin} = \frac{r_{PPH}}{r_{Lin}}$,y $R_{ENO/Lin} = \frac{r_{ENO}}{r_{Lin}}$, donde r_{scheme} es el valor del $PSNR$ entre la imagen recuperada y la original

Y por último en la Tabla 4 se muestran los resultados obtenidos para la imagen peppers

L	Ruido	$R_{ENO/LIN}$	$R_{PPH/LIN}$	$R_{PPH/ENO}$
3	3	0,972	1,032	1,061
	5	0,991	1,023	1,023
	10	1,000	1,019	1,019
4	3	0,951	1,034	1,034
	5	0,971	1,027	1,058
	10	1,010	1,027	1,016
5	3	0,932	1,041	1,116
	5	0,972	1,036	1,065
	10	0,969	1,026	1,058

Tabla 2: **Imagen original camera2.pgm**, L niveles de multirresolución, $R_{PPH/ENO} = \frac{r_{PPH}}{r_{ENO}}$, $R_{PPH/Lin} = \frac{r_{PPH}}{r_{Lin}}$,y $R_{ENO/Lin} = \frac{r_{ENO}}{r_{Lin}}$, donde r_{scheme} es el valor del $PSNR$ entre la imagen recuperada y la original

L	Ruido	$R_{ENO/LIN}$	$R_{PPH/LIN}$	$R_{PPH/ENO}$
3	3	0,979	1,043	1,065
	5	0,998	1,027	1,028
	10	1,012	1,018	1,005
4	3	0,971	1,046	1,077
	5	0,992	1,030	1,038
	10	1,014	1,017	1,003
5	3	0,0957	1,050	1,096
	5	0,994	1,024	1,030
	10	1,003	1,017	1,014

Tabla 3: **Imagen original seis5.pgm**, L niveles de multirresolución, $R_{PPH/ENO} = \frac{r_{PPH}}{r_{ENO}}$, $R_{PPH/Lin} = \frac{r_{PPH}}{r_{Lin}}$,y $R_{ENO/Lin} = \frac{r_{ENO}}{r_{Lin}}$, donde r_{scheme} es el valor del $PSNR$ entre la imagen recuperada y la original

De las pruebas realizadas con cada una de las fotografías hemos seleccionando aquellas pruebas dónde se puede visualizar de mejor manera los resultados obtenidos con cada uno de los métodos.

L	Ruido	$R_{ENO/LIN}$	$R_{PPH/LIN}$	$R_{PPH/ENO}$
3	3	0,994	1,014	1,021
	5	0,999	1,010	1,011
	10	1,002	1,011	1,009
4	3	0,982	1,016	1,034
	5	0,986	1,008	1,022
	10	0,995	1,013	1,018
5	3	0,0983	1,024	1,040
	5	1,000	1,028	1,027
	10	1,012	1,021	1,009

Tabla 4: **Imagen original peppers.pgm**, L niveles de multirresolución, $R_{PPH/ENO} = \frac{r_{PPH}}{r_{ENO}}$, $R_{PPH/Lin} = \frac{r_{PPH}}{r_{Lin}}$,y $R_{ENO/Lin} = \frac{r_{ENO}}{r_{Lin}}$, donde r_{scheme} es el valor del $PSNR$ entre la imagen recuperada y la original

Figura 9: Prueba realizada con la imagen de Lena con 3 niveles de resolución y 10 de varianza de ruido aplicando el método lineal: imagen original, imagen con ruido e imagen mejorada

Figura 10: Prueba realizada con la imagen de Lena con 3 niveles de resolución y 10 de varianza de ruido aplicando el método ENO: imagen original, imagen con ruido e imagen mejorada

Figura 11: Prueba realizada con la imagen de Lena con 3 niveles de resolución y 10 de varianza de ruido aplicando el método ENO jerárquico: imagen original, imagen con ruido e imagen mejorada

Figura 12: Prueba realizada con la imagen de Lena con 3 niveles de resolución y 10 de varianza de ruido aplicando el método PPH: imagen original, imagen con ruido e imagen mejorada

Figura 13: Prueba realizada con la imagen cameraman con 3 niveles de resolución y 10 de varianza de ruido aplicando el método lineal: imagen original, imagen con ruido e imagen mejorada

Figura 14: Prueba realizada con la imagen cameraman con 3 niveles de resolución y 10 de varianza de ruido aplicando el método ENO: imagen original, imagen con ruido e imagen mejorada

IMAGEN ORIGINAL

IMAGEN CON RUIDO

IMAGEN MEJORADA

Figura 15: Prueba realizada con la imagen cameraman con 3 niveles de resolución y 10 de varianza de ruido aplicando el método ENO jerárquico: imagen original, imagen con ruido e imagen mejorada

Figura 16: Prueba realizada con la imagen cameraman con 3 niveles de resolución y 10 de varianza de ruido aplicando el método PPH: imagen original, imagen con ruido e imagen mejorada

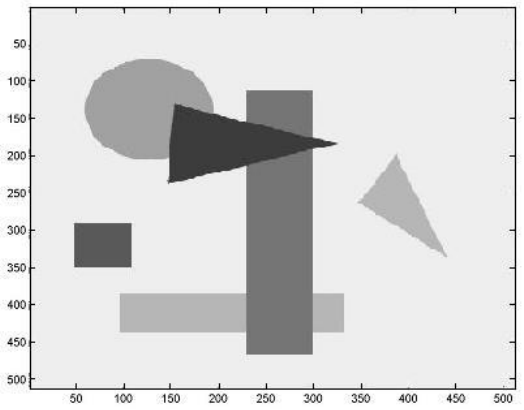

IMAGEN ORIGINAL

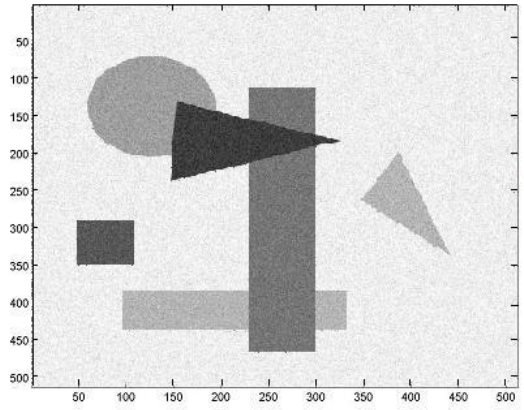

IMAGEN CON RUIDO

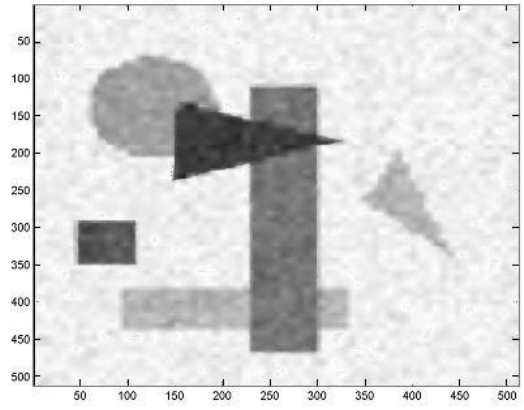

IMAGEN MEJORADA

Figura 17: Prueba realizada con la imagen geométrica con 3 niveles de resolución y 10 de varianza de ruido aplicando el método PPH: imagen original, imagen con ruido e imagen mejorada

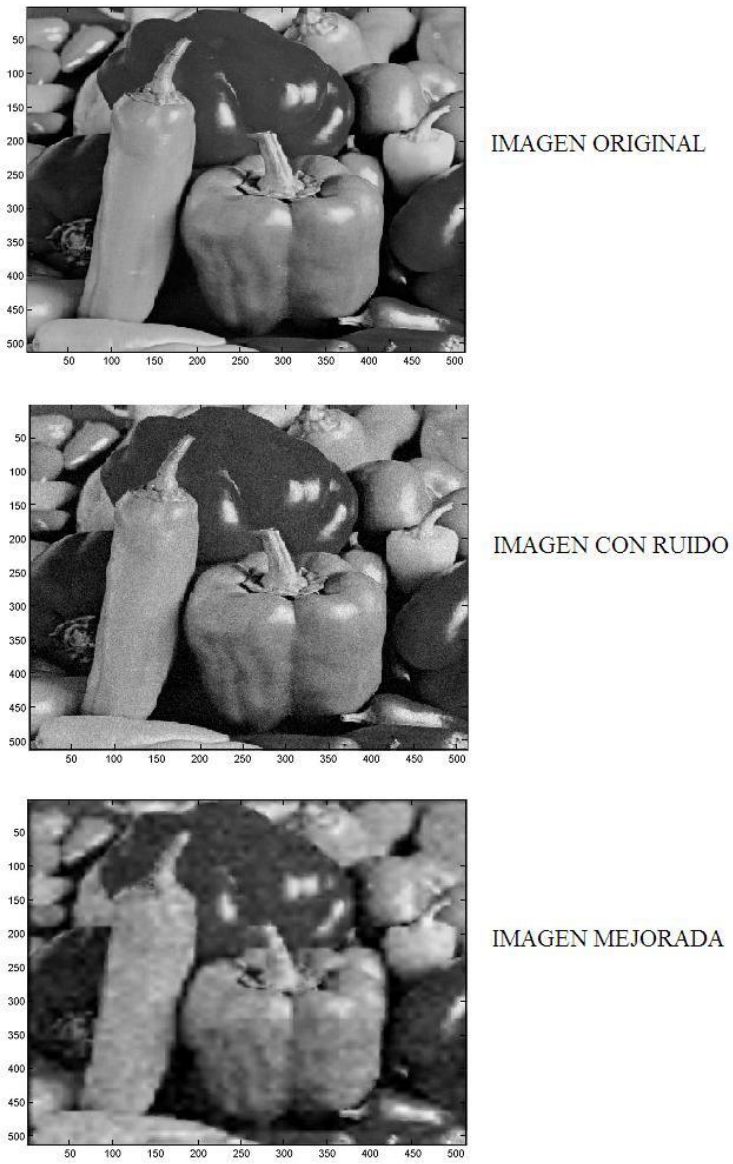

Figura 18: Prueba realizada con la imagen peppers con 3 niveles de resolución y 10 de varianza de ruido aplicando el método PPH:: imagen original, imagen con ruido e imagen mejorada

3. Desarrollo de los programas en Matlab

El desarrollo del proyecto ha sido realizado mediante programación en Matlab de las diferentes funciones necesarias para llevar a cabo cada uno de los pasos hasta la obtención de los resultados.

3.1. Explicación de las funciones y de la interfaz gráfica

Se ha desarrollado una interfaz que haga más sencillo y accesible la utilización de los mismos a cualquier usuario sin necesidad de conocer el lenguaje.

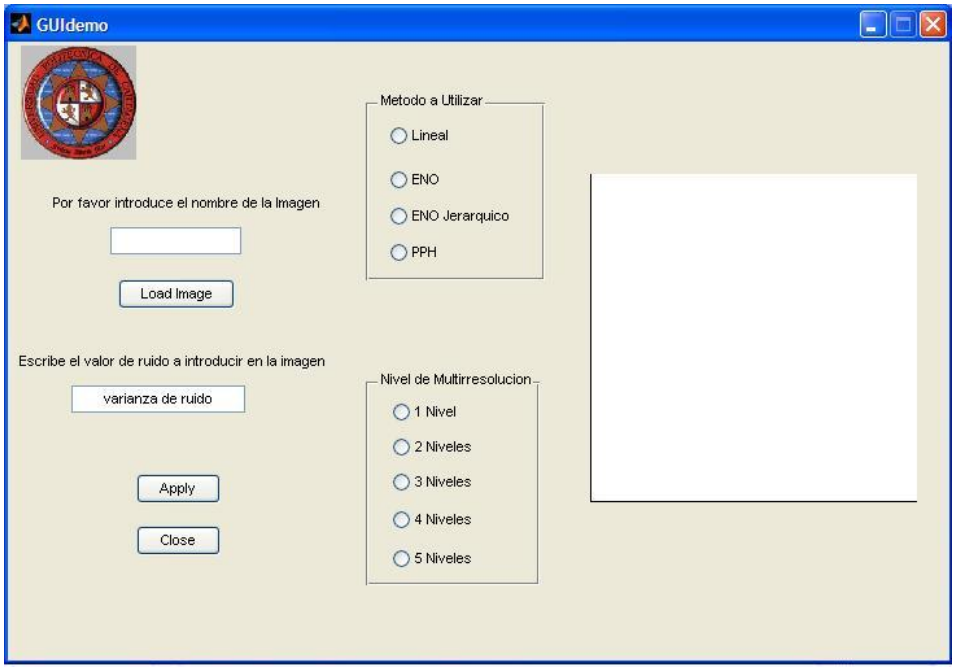

Figura 19: Entorno gráfico para ejecutar el programa

Este entorno gráfico está diseñado de la siguiente forma: Contiene una casilla donde el usuario debe seleccionar la imagen con la cual queremos trabajar y que una vez seleccionada se visualizará en la parte derecha de la interfaz. Dispone de otra casilla donde se introduce de manera manual la cantidad de ruido que queremos aplicar en la imagen original. Además

también cuenta con radio botones para seleccionar tanto los niveles de multirresolución que queremos aplicar como el método para llevar a cabo el proceso de denoising. Una vez que se han seleccionado todos los parámetros correctamente se pulsa el botón Apply mediante el cual se procede a ejecutar los programas y se nos muestra la imagen con ruido y la imagen reconstruida en dos ventanas que se abren automáticamente.

Vamos a realizar un ejemplo de utilización de la interfaz con la fotografía camera2.pgm, un nivel de ruido introducido de 5, mediante el método ENO y con 4 niveles de multirresolución. Como primer paso debemos introducir la imagen que queremos tratar en el casillero de la esquina superior izquierda, presionar el botón que encontramos debajo del mismo, Load Image, y la visualizamos a la derecha.

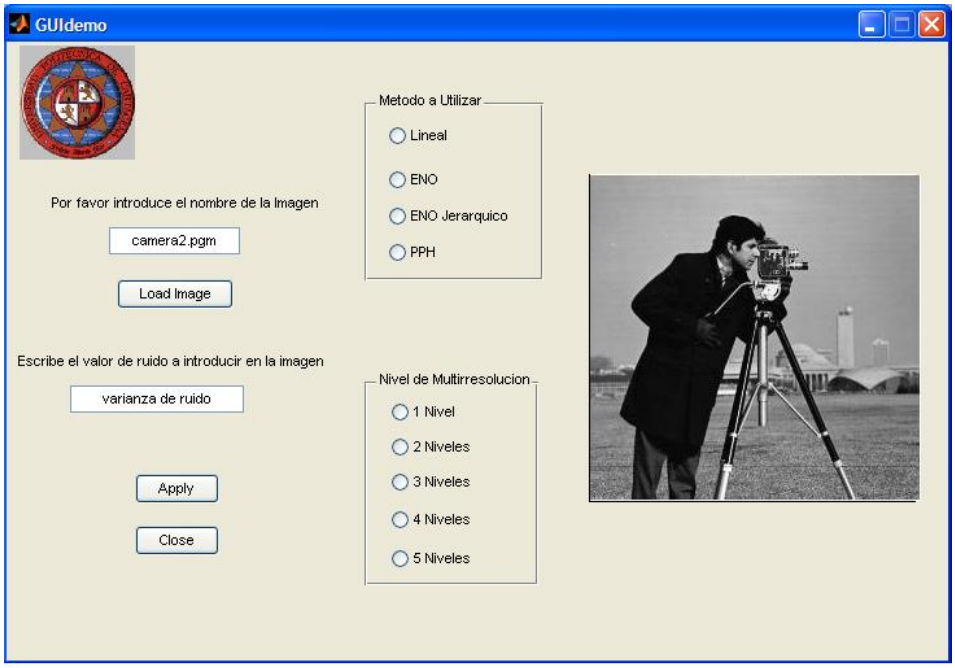

Figura 20: Entorno gráfico una vez seleccionada la imagen a tratar

Una vez que ya tenemos la imagen con la que queremos trabajar, vamos a introducir en el segundo casillero la cantidad de ruido que vamos aplicar a la imagen, en nuestro ejemplo 5, y a seleccionar mediante los radio botones los niveles de multirresolución y el método que vamos a utilizar para proceder al denoising.

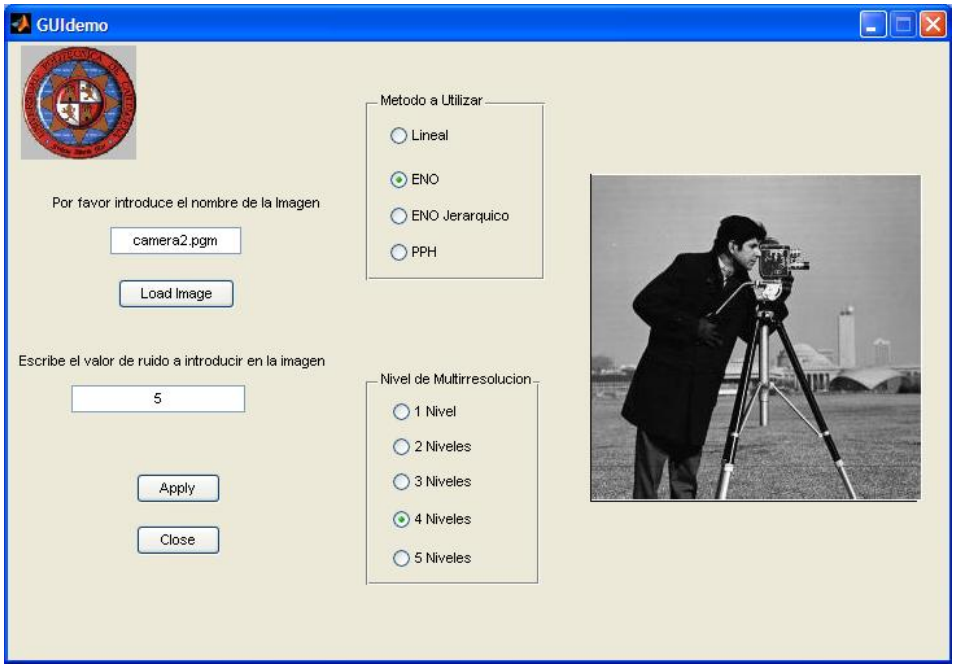

Figura 21: Entorno gráfico una vez introducidos los parámetros : ruido, número de niveles de resolución y método a utilizar

Y por último, una vez que tenemos todos los valores seleccionados pulsamos el botón Apply y el programa nos devolverá como resultado de la aplicación dos ventanas, una de ellas mostrándonos la imagen una vez introducido el ruido (ver Figura 3.1.

Y la otra de ellas mostrando la imagen una vez reconstruida mediante los parámetros que le hemos indicado (ver Figura 3.1.

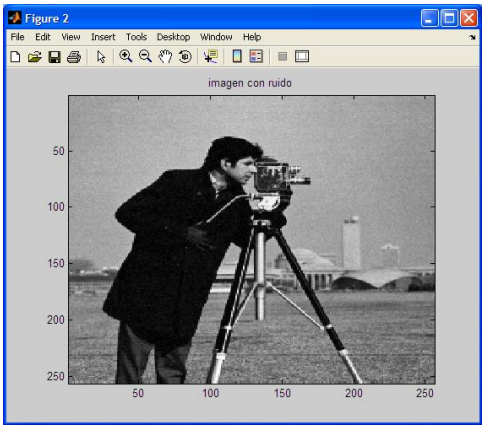

Figura 22: Ventana que muestra la imagen con ruido

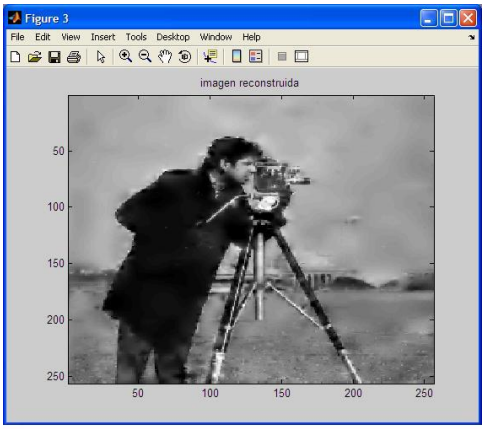

Figura 23: Ventana que muestra la imagen reconstruida

Cabe puntualizar que también es posible la ejecución del programa directamente desde la ventana de comandos del programa Matlab. Para la realización del ejemplo anterior sin el uso de la interfaz gráfica bastaría con introducir la sentencia en la ventana de comandos de Matlab (ver Figura 24).

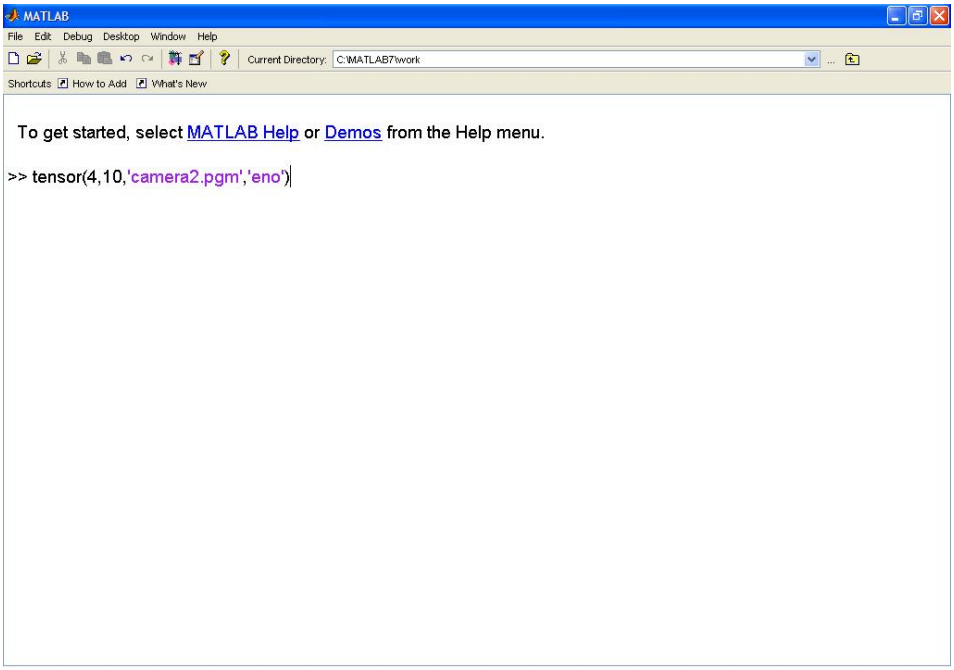

Figura 24: Ventana de comandos del programa Matlab

Describamos de forma esquemática y breve el orden en que se van ejecutando cada una de las funciones para conseguir una visión global del procedimiento seguido.

La función principal se denomina Tensor y es la que se encarga de realizar, a partir de los parámetros de entrada introducidos, los pasos necesarios para calcular y mostrar los resultados obtenidos comparando la matriz original, la matriz con ruido y la matriz mejorada. En los parámetros de entrada debemos indicar: el número de niveles de multirresolución, la varianza de ruido, la fotografía que deseamos estudiar y el método mediante el cual queremos llevar a cabo nuestro trabajo. En la primera parte de la función se realiza el descenso en la pirámide de multirresolución mediante los pasos de la Figura 25.

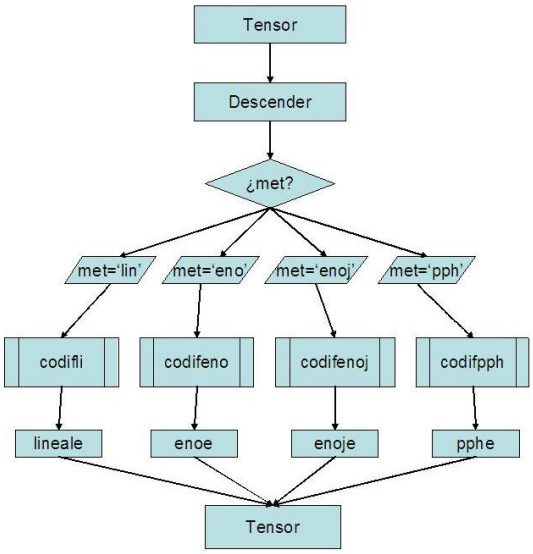

Figura 25: Diagrama de flujo de los pasos realizados para descender en la pirámide de multirresolución

En la segunda parte se realiza el proceso inverso, es decir el ascenso en la pirámide de multirresolución (ver Figura 26).

Como se puede observar en ambas gráficas se dispone de una función diferente para cada método de los cuatro que se utilizan y por tanto dependiendo del valor que introduzcamos en la variable 'met', que es la que almacena el método que queremos utilizar, se llamará a una función u otra. Una vez realizados ambos procedimientos la función Tensor se encarga, con los datos obtenidos, de calcular valores que nos son ser útiles para comparar y obtener conclusiones. Así se obtienen y dibujan por pantalla las 3 matrices (fotografías) obtenidas, la original que le pasamos como parámetro a la función tensor, la matriz con ruido y por último la matriz mejorada.
Los valores que se calculan en la ejecución del programa quedan almacenados en un fichero denominado comparación. El contenido de éste fichero por ejemplo para el caso en que quisiéramos analizar la imagen lena con 4 niveles de multirresolución y un ruido introducido de 10 y mediante el método pph, se visualiza en la Figura 27.

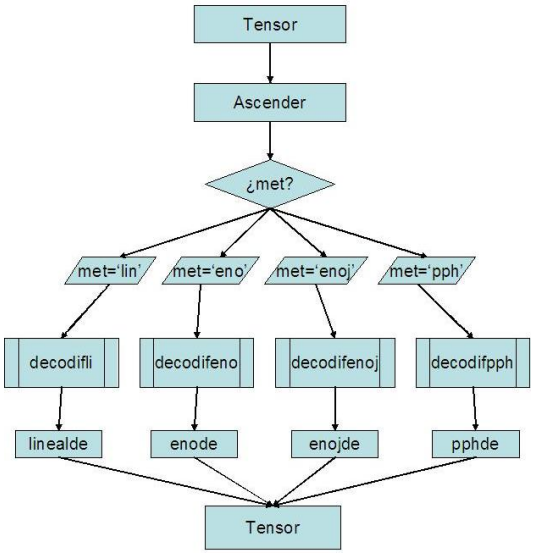

Figura 26: Diagrama de flujo para ascender en la pirámide de multirresolución

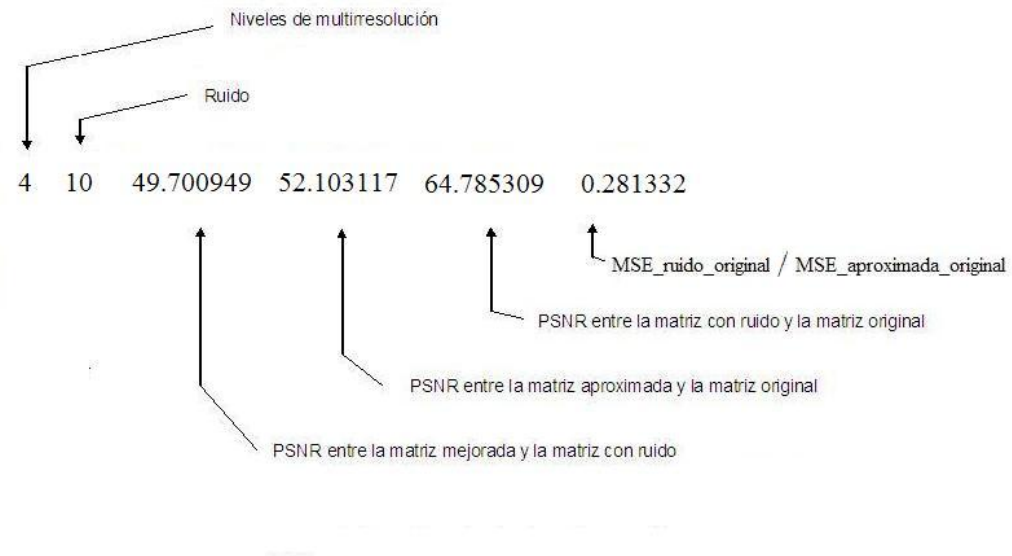

Figura 27: Fichero creado por el programa para almacenar los diferentes datos obtenidos para cada figura en cada uno de los niveles de multirresolución y para cada uno de los valores de ruido introducidos

También existe la posibilidad de que nos cree éste fichero calculando datos para distintos niveles de resolución y distintos valores de tolerancia y ruido ejecutando el programa guion.m, es decir nos devolvería un fichero de las mismas características que el de la figura anterior pero con datos para 3,4 y 5 niveles de resolución y 3 valores de varianza de ruido para cada uno de los niveles de resolución enumerados.

En este punto vamos a explicar el procedimiento que siguen cada una de las funciones programadas en Matlab enumeradas en los diagramas de flujo del apartado anterior.

Comenzamos con la función *tensor.m*. Como se dijo con anterioridad es la función principal de nuestro programa, es la encargada de recibir las características que nosotros imponemos al ejecutar el programa y devolvernos los resultados. La cabecera de la función es de la siguiente manera:

```
function [a,a1,mra]=tensor(l,ruido,im,met)
```

Como se observa en la cabecera, la función tensor recibe como entrada 4 parámetros (l,ruido,im,met) correspondientes a los niveles de resolución, ruido, imagen y método respectivamente y nos devuelve como salida otros 3 parámetros (a,a1 y mra) que corresponden a la matriz mejorada, matriz con ruido y matriz original. A partir de los éstos parámetros tensor delega en la función descender y ascender para bajar y subir en la pirámide de resolución y llevar a cabo los pasos necesarios para poder calcular las diferentes matrices y por tanto los resultados que el usuario desea. Una vez que la función ascender devuelve el control a la función tensor ésta se encarga de analizar los datos recibidos, calcular los valores que más tarde almacenará en el fichero comparación y mostrarnos por pantalla las 3 matrices obtenidas (matriz original, matriz con ruido y matriz reconstruida).

A continuación nos encontramos con la función *descender.m*. Ésta función tiene la siguiente cabecera:

```
function [ru,c]=descender(a,l,ruido,met)
```

Como parámetros de entrada recibe los mismos que tensor pero recibiendo la imagen ya en forma de matriz a través del parámetro a. Produce como variables de salida la matriz con ruido, ru, y la versión de multirresolución, ya truncada, de la imagen con ruido que viene almacenada en c. De lo que se encarga descender es en primer lugar de introducirle una cantidad de ruido a la imagen original mediante las siguientes lineas:

```
c=c+ruido*randn(n);
c=255*(c>255)+c.*(0<c & c<=255);
```

Una vez que ya tenemos creada una matriz con ruido se procede, dependiendo del método por el cual hayamos especificado en el parámetro de entrada met, a delegar en uno de los 4 ficheros de codificación de los métodos,es decir, se delega en *codifli.m, codifeno.m, codifenoj.m o codifpph.m.*

La función *codifli.m* tiene la siguiente cabecera:

```
function [d]=codifli(a,n,l,ruido)
```

Nos devuelve como variable de salida la matriz con las 4 submatrices ordenadas después de haber llevado a cabo la multirresolución y recibe como entrada la matriz a la cual se le aplica la multirresolución, el número de filas y columnas de dicha matriz, los niveles de multirresolución y la varianza del ruido. Se encarga de realizar la multirresolución por filas y por columnas mediante bucles for y utilizando la función *lineale.m* para efectuar un paso del algoritmo 1D de dicha multirresolución. Una vez que ya tenemos localizados los 4 tipos de datos de la matriz(valores significativos, detalles horizontales, detalles verticales y detalles mixtos) se procede a ordenar dicha matriz de la siguiente manera:

```
b1=d(1:2:n,1:2:n);
b2=d(1:2:n,2:2:n-1);
b3=d(2:2:n-1,1:2:n);
b4=d(2:2:n-1,2:2:n-1);
```

Así ya tendríamos la matriz con los tipos de datos ordenados, correspondiéndose b1 con la submatriz de valores significativos, b2 con los detalles horizontales, b3 detalles verticales y b4 detalles mixtos (ver Figura 28).

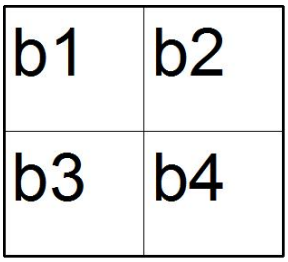

Figura 28: División y disposición de las submatrices de valores significativos y detalles

Una vez obtenida dicha matriz se procede a realizar el truncamiento suave de los 3 tipos de detalles originados con la multirresolución de la forma:

```
[b2n,b2m]=size(b2);
[b3n,b3m]=size(b3);
[b4n,b4m]=size(b4);

tol2=ruido*(nombre)*sqrt(2*(log(b2m)+log(b2n)));
tol3=ruido*(nombre)*sqrt(2*(log(b3n)+log(b3m)));
tol4=ruido*(nombre)*sqrt(2*(log(b4m)+log(b4n)));

% formulas para hacer el soft thresholding
b2=(b2-sign(b2)*tol2).*(abs(b2)>tol2);
b4=(b4-sign(b4)*tol4).*(abs(b4)>tol4);
b3=(b3-sign(b3)*tol3).*(abs(b3)>tol3);
```

Y una vez repetido el bucle tantas veces como niveles de resolución hayamos indicado se produce el return. La función que acabamos de exponer se realiza cuando el método indicado sea el lineal, si el método indicado en los parámetros de entrada de tensor fuese ENO se procedería a ejecutar en lugar de *codifli.m*, la función *codifeno.m*.

La cabecera de *codifeno.m* es la siguiente:

```
function [d]=codifeno(a,n,l,ruido)
```

Recibe los mismos parámetros de entrada que la función anterior y produce la misma variable de salida ya que queremos que a partir de los mismos datos nos produzca el mismo dato de salida pero obtenido a través de un método diferente. Por lo tanto comienza realizando la multirresolución por filas y por columnas mediante el bucle for pero utilizando una función propia que se denomina *enoe.m* y una vez obtenidos los 4 tipos de datos procede a ordenar la matriz para que quede dispuesta, al igual que en la función anterior, en b1, b2, b3 y b4. Aplica las fórmulas de truncamiento a las submatrices de detalles, repite el bucle de multirresolución tantas veces como niveles de multirresolución y se produce el return.

Si el método indicado en tensor para llevar a cabo el programa hubiese sido el ENO jerárquico, descender delegaría en *codifenoj.m* para llevar a cabo la multirresolución. La cabecera de *codifenoj.m* es:

```
function [d]=codifenoj(a,n,l,ruido)
```

Las variables de entrada y salida siguen siendo los mismos que en las dos funciones anteriores como era de esperar. La diferencia reside en la manera de realizar la multirresolución ya que el método ENO jerárquico también tiene una función propia para llevar a cabo dicha opción. Por lo tanto a la hora de ejecutar la multirresolución la función codifenoj.m accede a la función *enoje.m* que al igual que las demás está explicada más adelante. Una vez realizada la multirresolución seguimos el mismo procedimiento que en los dos casos anteriores y pasamos a ordenar la matriz según los tipos de datos que contiene, se realiza la truncación de los detalles y se repite el bucle de multirresolución las veces necesarias para completar los niveles exigidos a la hora de ejecutar tensor y se produce el return.

Y como última opción para este paso del programa nos quedaría la función *codifpph.m* que sería la que se ejecutaría en el caso de que hayamos escogido como método para llevar a cabo la multirresolución el PPH. La función tiene la siguiente cabecera:

```
function [d]=codifpph(a,n,l,ruido)
```

Y la diferencia con las 3 anteriores sigue siendo la misma que se ha dado en los otros casos, la manera de realizar la multirresolución. La función a la que acude *codifpph.m* es *pphe.m* y ésta se encargará tanto de la multirresolución por filas como por columnas y una vez realizado se ordena la matriz y realiza el truncamiento.

Ahora vamos a proceder a explicar el último paso o la última función que interviene en el proceso de en la pirámide de multirresolución. Dependiendo del método elegido la función que se encarga de éste último paso será una de las nombradas anteriormente: *lineale.m, enoe.m, enoj.m* o *pphe.m* .

Las 4 funciones tienen el mismo cometido aunque cada una lo alcance aplicando un método diferente, por lo tanto los parámetros de entrada y variables de salida de las 4 son los mismos.

Las cabeceras de las funciones son:

```
function [f1,f2]=lineale(f,n)
function [f1,f2]=enoe(f,n)
function [f1,f2]=enoje(f,n)
function [f1,f2]=pphe(f,n)
```

Los parámetros de entrada se corresponden con el vector original y con la dimensión de dicho vector respectivamente y los de salida con los detalles significativos del vector de la escala inferior y con los detalles entre escalas. A lo largo de cada una de las funciones se procede al cálculo de los datos que corresponden a valores significativos y los datos que corresponden a detalles realizando cada uno de ellos mediante el método lineal, ENO, ENO jerárquico y PPH respectivamente.
Una vez que ya se ha cubierto la tarea de descender en la pirámide de multirresolución, la función tensor llama a la función ascender para llevar a cabo el proceso inverso, es decir para ascender de nuevo en la pirámide de multirresolución y conseguir la imagen reconstruida.

La función *ascender.m* tiene la siguiente cabecera:

```
function c=ascender(a,l,met)
```

La función recibe como parámetros de entrada la matriz a la que le aplicamos el algoritmo ascendente de multirresolución, el número de los niveles de multirresolución que vamos aplicar y el método mediante el cual queremos realizar la reconstrucción respectivamente y nos devuelve como variable de salida la matriz reconstruida. Esta función se encarga de delegar en una de las siguientes 4 funciones el trabajo a realizar para reconstruir la matriz dependiendo del valor del parámetro de entrada met. Según este parámetro se accede a *decodifli.m, decodifeno.m, decodifenoj.m* o *decodifpph.m*.

El primero de todos ellos y al que se accedería si el valor de met fuese 'lineal', *decodifli.m*, tiene la siguiente cabecera:

```
function c=decodifli(a,n,l)
```

recibe como parámetros de entrada la matriz a la que vamos a aplicar el algoritmo, el número de filas y columnas de dicha matriz y el número de niveles de multirresolución a aplicar y nos produce como variable de salida la matriz reconstruida. El procedimiento que sigue para cada nivel de multirresolución es reordenar la matriz de la manera inversa a como se hacía en las funciones correspondientes al descenso en la pirámide de la manera que sigue:

```
nb=2*nl+1;
b=zeros(nb);
b(1:2:nb,1:2:nb)=c(1:nl+1,1:nl+1);
b(2:2:nb-1,2:2:nb-1)=c(nl+2:nb,nl+2:nb);
b(1:2:nb,2:2:nb-1)=c(1:nl+1,nl+2:nb);
b(2:2:nb-1,1:2:nb)=c(2+nl:nb,1:nl+1);
```

```
c(1:nb,1:nb)=b;
clear b;
```

y después se aplica el algoritmo inverso también para la multirresolución por columnas y por filas mediante el método *linealde.m*, que será explicado más adelante, y se actualiza el nivel para subir al siguiente en la escala de multirresolución.

El segundo método al que se accedería si el valor de met fuese 'ENO' tiene la siguiente cabecera:

```
function c=decodifeno(a,n,l)
```

Como ocurría durante el proceso de descenso en la pirámide de multirresolución los parámetros de entrada y variables de salida de esta función, tanto con respecto a la función anterior como a las dos siguientes, son los mismos, ya que aunque cambie la manera de calcular o de obtener los resultados dependiendo del procedimiento de cada método lo que me interesa es que me devuelvan la matriz reconstruida a partir de unos parámetros que son comunes para todos los métodos. Por lo tanto, *decodifeno.m* recibe también como entrada la matriz, el número de filas y de columnas de la matriz y los niveles de multirresolución y a partir de éstos se encarga primero de reordenar la matriz y después de realizar el algoritmo inverso de multirresolución por columnas y por filas mediante la función *enode.m* y después actualiza el nivel de multirresolución.

La función que se tiene para realizar este paso en el que nos encontramos del programa si se tratase del método ENO jerárquico sería *decodifenoj.m*, y tiene la siguiente cabecera:

```
function c=decodifenoj(a,n,l)
```

Y realiza mediante los mismos pasos que los anteriores la reordenación de la matriz, la multirresolución, por columnas y por filas con el algoritmo inverso al descenso accediendo, en éste caso, a la función *enojde.m*, y después actualiza el nivel de multirresolución.

Y por último, nos queda la opción de que la variable met tenga el valor 'PPH' y por tanto que ascender delegue el funcionamiento en la función *decodifpph.m* para llevar a cabo el ascenso en la pirámide de multirresolución, y tiene como cabecera:

```
function c=decodifpph(a,n,l)
```

Éste se encarga al igual que los tres anteriores de reordenar la matriz que recibe como parámetro de entrada, realizar la multirresolución por columnas y por filas mediante la función *pphde.m* y de actualizar el nivel de multirresolución para repetir la operación tantas veces como niveles debamos ascender en la pirámide de multirresolución.

Y ya nos quedaría por enumerar el último paso que sería el realizado por una de estas cuatro funciones: *linealde.m*, *enode.m*, *enojde.m* o *pphde.m* dependiendo una vez más del método elegido para la ejecución del programa. La función linealde.m tiene la cabecera:

```
function f=linealde(v,n)
```

Recibe dos parámetros de entrada y produce una variable de salida. Los parámetros de entrada se corresponden con el vector que contiene los valores significativos de la escala anterior y los detalles para subir al siguiente nivel y la dimensión del mismo y nos produce como salida los valores significativos de la escala superior. A lo largo de su desarrollo se encarga de calcular todos los valores superiores mediante el algoritmo lineal calculando el primer y último valor por separado y el resto de los valores intermedios mediante un bucle for.

La segunda función a la que se llamaría en el caso de que el método seleccionado fuera ENO sería *enode.m* , y su cabecera es:

```
function f=enode(v,n)
```

Recibe como parámetros de entrada el vector con los valores significativos y detalles del nivel anterior y su dimensión y nos devuelve el vector con los valores significativos del siguiente nivel. Calcula los valores de los dos primeros y últimos elementos desplazándose hacia el lado donde dispone de datos. Los elementos intermedios mediante el método ENO. Los datos son almacenados en f.

La tercera opción, correspondiente con el método ENO jerárquico sería la función enojde.m con la siguiente cabecera:

```
function f=enojde(v,n)
```

A partir de los valores de entrada (vector con valores significativos y detalles del nivel superior y con su dimensión) calcula los valores significativos del

nivel superior y los devuelve almacenados en su variable de salida.

Y como última función capaz de llevar a cabo el paso en el que nos encontramos sería a la que se accede si el método elegido es PPH, en cuyo caso utilizaríamos el método pphde.m y tiene la cabecera:

```
function f=pphde(v,n)
```

Al igual que los anteriores su misión es calcular los valores significativos del nivel siguiente, el superior, en la escala de multirresolución a partir de los valores y los detalles que se indican en el parámetro de entrada v. Una vez calculados a través del método PPH nos los devuelve en la variable de salida de la función.

Para visualizar el contenido completo de las funciones programadas con Matlab, se incluye a continuación su código completo.

3.2. Código fuente de los algoritmos

tensor.m

```
function [a,a1,mra]=tensor(l,ruido,im,met)

% [a,a1,mra]=tensor(l,ruido,im,met);
% -------variables de salida-------
% a es la matriz mejorada
% a1 es la matriz con ruido
% mra es donde guardamos el valor de a antes de trabajar con ella
% ------variables de entrada-------
% l son los niveles de multirresolución
% ruido es el nivel de ruido que se suma a la imagen
% im imagen que utiliza 'lena5.pgm','seis5.pgm', 'camera2.pgm','peppers.pbm'
% met metodo que quieres utilizar 'lin', 'eno', 'enh', 'pph'

a=imread(im);
a=double(a);
qu=a;

% limpia la pantalla de salida de resultados

clc

% tamaño de la imagen original

n1=max(size(a));

% nos guardamos la matriz original en a1 para despues calcular los
errores cometidos
descendemos por la piramide de multirresolución

[a1,a]=descender(a,l,ruido,met);
mra=a;

% ascendemos por la piramide de multirresolución

a=ascender(a,l,met);

% le quitamos la ultima fila y columna a a1

b=a1(1:n1,1:n1);
clear a1;
a1=b;
clear b;
```

```
% medimos cuanto de buena es la reconstrucción usando el MSE y el
PSNR
% a aproximación
% a1 imagen con ruido
% qu imagen original

% Diferencias entre aprox-ruido

MSE_aprox_ruido=(norm(a-a1,'fro')^2)/n1/n1;
PSNR_aprox_ruido=20*log(255/(sqrt(MSE_aprox_ruido)));

% Diferencias entre aprox-original

MSE_aprox_original=(norm(a-qu,'fro')^2)/n1/n1;
PSNR_aprox_original=20*log(255/(sqrt(MSE_aprox_original)));

% Diferencias entre ruido-original

MSE_ruido_original=(norm(qu-a1,'fro')^2)/n1/n1;
PSNR_ruido_original=20*log(255/(sqrt(MSE_ruido_original)));

cociente=MSE_ruido_original/MSE_aprox_original;

% entrada a ficheros

fid = fopen('comparacion.dat','a');
fprintf(fid,'%d %d %f %f %f
 %f\n',l,ruido,PSNR_aprox_ruido,PSNR_aprox_original,PSNR_ruido_original,cociente);
fclose(fid);


figurec(1)
imagesc(qu,[0 255])
colormap gray
title('imagen original');


figurec(2)
imagesc(a1,[0 255])
colormap gray
title('imagen con ruido');
```

```
figurec(3)
imagesc(a,[0 255])
colormap gray
title('imagen reconstruida');
```

descender.m

```
function [ru,c]=descender(a,l,ruido,met)

% [ru,c]=descender(a,l,ruido,met)
% -----variables de salida-----
% ru es la matriz con ruido
% c es la que se utiliza para ir haciendo modificaciones a la matriz
% -----variables de entrada----
% a matriz a la que le aplicamos el algorimo descendente de multirresolución
% l son los niveles de multirresolución
% ruido es el nivel de ruido que se suma a la imagen
% met metodo que quieres utilizar 'lin', 'eno', 'enh' , 'pph'

% aumentamos la matriz a en una columna y en una fila

n=max(size(a));
y1=a(1:n,n);
c=[a,y1];
clear y1;
y1=c(n,1:n+1);
c=[c;y1];
n=n+1; s=(n-1)/2^l+1;

% introducimos ruido en la imagen

c=c+ruido*randn(n);
c=255*(c>255)+c.*(0<c & c<=255);
ru=c;

% llamamos al método

if(met=='lin')
     [c]=codifli(c,n,l,ruido);
end

if(met=='eno')
     [c]=codifeno(c,n,l,ruido);
end

if(met=='enh')
    [c]=codifenoj(c,n,l,ruido);
```

```
end

if(met=='pph')
     [c]=codifpph(c,n,l,ruido);
end
```

codifli.m

```
function [d]=codifli(a,n,l,ruido)

% [d]=codifli(a,n,l,ruido)
% ----variables de salida----
% d contiene las 4 submatrices ordenadas (valores, detalles: verticales,horizontales
y mixtos)
% ----variables de entrada----
% a matriz a la que se le aplica la multirresolución
% n número de filas y columnas de a
% l son los niveles de multirresolución
% ruido es el nivel de ruido que se suma a la imagen

% inicialización de las variables

d=a;

% bucle para los niveles de multirresolución

for k=1:l

% se hace la multirresolución por filas

for i=1:n
[f1,f2]=lineale(d(i,1:n),n);
d(i,1:2:n)=f1;
d(i,2:2:n-1)=f2;
end
clear f1; clear f2;

% se hace la multirresolución por columnas

for j=1:n
[f1,f2]=lineale(d(1:n,j),n);
d(1:2:n,j)=f1;
d(2:2:n-1,j)=f2';
end

clear f1; clear f2;

% se ordena la matriz
```

```
b1=d(1:2:n,1:2:n);
b2=d(1:2:n,2:2:n-1);
b3=d(2:2:n-1,1:2:n);
b4=d(2:2:n-1,2:2:n-1);

[b2n,b2m]=size(b2);
[b3n,b3m]=size(b3);
[b4n,b4m]=size(b4);

% UNIVERSAL GLOBAL THRESHOLDING para b2, b3 y b4

tol2=ruido*(nombre)*sqrt(2*(log(b2m)+log(b2n)));
tol3=ruido*(nombre)*sqrt(2*(log(b3n)+log(b3m)));
tol4=ruido*(nombre)*sqrt(2*(log(b4m)+log(b4n)));

% formulas para hacer el soft thresholding

b2=(b2-sign(b2)*tol2).*(abs(b2)>tol2);
b4=(b4-sign(b4)*tol4).*(abs(b4)>tol4);
b3=(b3-sign(b3)*tol3).*(abs(b3)>tol3);

% se actualiza para hacer el siguiente paso del bucle

d(1:n,1:n)=[b1,b2;b3,b4];
clear b1; clear b2; clear b3; clear b4;

% se modifica el valor de n

n=(n-1)/2+1;

% se cierra el bucle de los niveles de multirresolución

end

return
```

codifeno.m

```
function [d]=codifeno(a,n,l,ruido)

% [d]=codifeno(a,n,l,ruido)
% ----variables de salida----
% d es la matriz que contiene las 4 submatrices ordenadas
% ----variables de entrada----
% a matriz a la que se le aplica la multirresolución
% n número de filas y columnas de a
% l son los niveles de multirresolución
% ruido es el nivel de ruido que se suma a la imagen

% inicialización de las variables

d=a;

% bucle para los niveles de multirresolución

for k=1:l

% se hace la multirresolución por filas

for i=1:n
[f1,f2]=enoe(d(i,1:n),n);
d(i,1:2:n)=f1;
d(i,2:2:n-1)=f2;
end
clear f1; clear f2;


% se hace la multirresolución por columnas

for j=1:n
[f1,f2]=enoe(d(1:n,j),n);
d(1:2:n,j)=f1;
d(2:2:n-1,j)=f2';
end

clear f1; clear f2;

% se ordena la matriz
```

```
b1=d(1:2:n,1:2:n);
b2=d(1:2:n,2:2:n-1);
b3=d(2:2:n-1,1:2:n);
b4=d(2:2:n-1,2:2:n-1);

[b2n,b2m]=size(b2);
[b3n,b3m]=size(b3);
[b4n,b4m]=size(b4);

% UNIVERSAL GLOBAL THRESHOLDING para b2, b3 y b4

tol2=ruido*(nombre)*sqrt(2*(log(b2m)+log(b2n)))/;
tol3=ruido*(nombre)*sqrt(2*(log(b3n)+log(b3m)))/;
tol4=ruido*(nombre)*sqrt(2*(log(b4m)+log(b4n)));


% fórmulas para hacer el soft thresholding

b2=(b2-sign(b2)*tol2).*(abs(b2)>tol2);
b4=(b4-sign(b4)*tol4).*(abs(b4)>tol4);
b3=(b3-sign(b3)*tol3).*(abs(b3)>tol3);

d(1:n,1:n)=[b1,b2;b3,b4];
clear b1; clear b2; clear b3; clear b4;


% se modifica el valor de n

n=(n-1)/2+1;

% se cierra el bucle de los niveles de multirresolución

end

return
```

codifenoj.m

```
function [d]=codifenoj(a,n,l,ruido)

% [d]=codifenoj(a,n,l,ruido)
% ----variables de salida----
% d es la matriz que contiene las 4 submatrices ordenadas
% ----variables de entrada----
% a matriz a la que se le aplica la multirresolución
% n número de filas y columnas de a
% l son los niveles de multirresolución
% ruido es el nivel de ruido que se suma a la imagen

% inicialización de las variables

d=a;

% bucle para los niveles de multirresolución

for k=1:l

% se hace la multirresolución por filas

for i=1:n
[f1,f2]=enoje(d(i,1:n),n);
d(i,1:2:n)=f1;
d(i,2:2:n-1)=f2;
end
clear f1; clear f2;


% se hace la multirresolución por columnas

for j=1:n
[f1,f2]=enoje(d(1:n,j),n);
d(1:2:n,j)=f1;
d(2:2:n-1,j)=f2';
end
clear f1; clear f2;

% se ordena la matriz
```

```
b1=d(1:2:n,1:2:n);
b2=d(1:2:n,2:2:n-1);
b3=d(2:2:n-1,1:2:n);
b4=d(2:2:n-1,2:2:n-1);

[b2n,b2m]=size(b2);
[b3n,b3m]=size(b3);
[b4n,b4m]=size(b4);

% UNIVERSAL GLOBAL THRESHOLDING para b2, b3 y b4

tol2=ruido*(nombre)*sqrt(2*(log(b2m)+log(b2n)));
tol3=ruido*(nombre)*sqrt(2*(log(b3n)+log(b3m)));
tol4=ruido*(nombre)*sqrt(2*(log(b4m)+log(b4n)));

% fórmulas para hacer el soft thresholding

b2=(b2-sign(b2)*tol2).*(abs(b2)>tol2);
b4=(b4-sign(b4)*tol4).*(abs(b4)>tol4);
b3=(b3-sign(b3)*tol3).*(abs(b3)>tol3);


d(1:n,1:n)=[b1,b2;b3,b4];
clear b1; clear b2; clear b3; clear b4;


% se modifica el valor de n

n=(n-1)/2+1;

% se cierra el bucle de los niveles de multirresolución

end

return
```

codifpph.m

```
function [d]=codifpph(a,n,l,ruido)

% [d]=codifpph(a,n,l,ruido);
% ----variables de salida----
% d es la matriz que contiene las 4 submatrices ordenadas
% ----variables de entrada----
% a matriz a la que se le aplica la multirresolución
% n número de filas y columnas de a
% l son los niveles de multirresolución
% ruido es el nivel de ruido que se suma a la imagen


   % inicialización de las variables

d=a;

% bucle para los niveles de multirresolución

for k=1:l

% se hace la multirresolución por filas

for i=1:n
[f1,f2]=pphe(d(i,1:n),n);
d(i,1:2:n)=f1;
d(i,2:2:n-1)=f2;
end
clear f1; clear f2;

% se hace la multirresolución por columnas

for j=1:n
[f1,f2]=pphe(d(1:n,j),n);
d(1:2:n,j)=f1;
d(2:2:n-1,j)=f2';
end
clear f1; clear f2;


% se ordena la matriz
```

```
b1=d(1:2:n,1:2:n);
b2=d(1:2:n,2:2:n-1);
b3=d(2:2:n-1,1:2:n);
b4=d(2:2:n-1,2:2:n-1);


[b2n,b2m]=size(b2);
[b3n,b3m]=size(b3);
[b4n,b4m]=size(b4);


% UNIVERSAL GLOBAL THRESHOLDING para b2, b3 y b4

tol2=ruido*(nombre)*sqrt(2*(log(b2m)+log(b2n)));
tol3=ruido*(nombre)*sqrt(2*(log(b3n)+log(b3m)));
tol4=ruido*(nombre)*sqrt(2*(log(b4m)+log(b4n)));


% fórmulas para hacer el soft thresholding

b2=(b2-sign(b2)*tol2).*(abs(b2)>tol2);
b4=(b4-sign(b4)*tol4).*(abs(b4)>tol4);
b3=(b3-sign(b3)*tol3).*(abs(b3)>tol3);


% se actualiza para hacer el siguiente paso del bucle

d(1:n,1:n)=[b1,b2;b3,b4];
clear b1; clear b2; clear b3; clear b4;


% se modifica el valor de n

n=(n-1)/2+1;

% se cierra el bucle de los niveles de multirresolución

end

return
```

lineale.m

```
function [f1,f2]=lineale(f,n)

% [f1,f2]=lineale(f,n);
% ---variables de salida----
% f1 valores significativos de la escala inferior
% f2 detalles entre las escalas
% ---variables de entrada----
% f vector original
% n dimensión de f

% valores significativos de la escala inferior

f1=f(1:2:n);
nk1=(n-1)/2;

% cálculo del primer detalle

f2(1)=f(2)-((15/48)*f1(1)+(45/48)*f1(2)-(15/48)*f1(3)+(3/48)*f1(4));

% cálculo de los detalles intermedios

for j=2:nk1-1
q=(-f1(j-1)+9*f1(j)+9*f1(j+1)-f1(j+2))/16;
f2(j)=f(2*j)-q;
end

% cálculo del último detalle

f2(nk1)=f(n-1)-((15/48)*f1(nk1+1)+(45/48)*f1(nk1)-(15/48)*f1(nk1-1)+(3/48)*f1(nk1-2))
```

enoe.m

```
function [f1,f2]=enoe(f,n)

% [f1,f2]=enoe(f,n);
% ---variables de salida----
% f1 valores significativos de la escala inferior
% f2 detalles entre las escalas
% ---variables de entrada----
% f vector original
% n dimensión de f

% valores significativos de la escala inferior

f1=f(1:2:n);
nk1=(n-1)/2;

% cálculo del primer detalle

f2(1)=f(2)-((15./48)*f1(1)+(45./48)*f1(2)-(15./48)*f1(3)+(3./48)*f1(4));

% cálculo del segundo detalle

at2=abs(-f1(1)+3*f1(2)-3*f1(3)+f1(4));
at3=abs(-f1(2)+3*f1(3)-3*f1(4)+f1(5));
if(at2<=at3)
q=(-f1(1)+9*f1(2)+9*f1(3)-f1(4))/16;
else
q=(15/48)*f1(2)+(45/48)*f1(3)-(15/48)*f1(4)+(3/48)*f1(5);
end

f2(2)=f(4)-q;

% cálculo de los detalles intermedios

for j=3:nk1-2
at1=abs(-f1(j-2)+3*f1(j-1)-3*f1(j)+f1(j+1));
at2=abs(-f1(j-1)+3*f1(j)-3*f1(j+1)+f1(j+2));
at3=abs(-f1(j)+3*f1(j+1)-3*f1(j+2)+f1(j+3));

if(at2<=at1 & at2<=at3)
      q=(-f1(j-1)+9*f1(j)+9*f1(j+1)-f1(j+2))/16;
else
```

```
     if(at1<=at3)
       q=(3/48)*f1(j-2)-(15/48)*f1(j-1)+(45/48)*f1(j)+(15/48)*f1(j+1);
     else
       q=(15/48)*f1(j)+(45/48)*f1(j+1)-(15/48)*f1(j+2)+(3/48)*f1(j+3);
     end
end

f2(j)=f(2*j)-q;
end

% cálculo del penúltimo detalle

at1=abs(-f1(nk1-3)+3*f1(nk1-2)-3*f1(nk1-1)+f1(nk1));
at2=abs(-f1(nk1-2)+3*f1(nk1-1)-3*f1(nk1)+f1(nk1+1));

if(at2<=at1)
     q=(-f1(nk1-2)+9*f1(nk1-1)+9*f1(nk1)-f1(nk1+1))/16;
else
     q=(3/48)*f1(nk1-3)-(15/48)*f1(nk1-2)+(45/48)*f1(nk1-1)+(15/48)*f1(nk1);
end

f2(nk1-1)=f(2*(nk1-1))-q;

% cálculo del último detalle

f2(nk1)=f(n-1)-((3/48)*f1(nk1-2)-(15/48)*f1(nk1-1)+(45/48)*f1(nk1)+(15/48)*f1(nk1+1))
```

enoje.m

```
function [f1,f2]=enoje(f,n)

% [f1,f2]=enoje(f,n);
% ---variables de salida----
% f1 valores significativos de la escala inferior
% f2 detalles entre las escalas
% ---variables de entrada----
% f vector original
% n dimension de f

% valores significativos de la escala inferior

f1=f(1:2:n);
nk1=(n-1)/2;

% cálculo del primer detalle

f2(1)=f(2)-((15./48)*f1(1)+(45./48)*f1(2)-(15./48)*f1(3)+(3./48)*f1(4));

% cálculo del segundo detalle

p1=(f1(2)-f1(1));
p2=(f1(3)-f1(2));
p3=(f1(4)-f1(3));
p4=(f1(5)-f1(4));
s1=(p2-p1); as1=abs(s1);
s2=(p3-p2); as2=abs(s2);

if(as1<=as2)
     q=(-f1(1)+9*f1(2)+9*f1(3)-f1(4))/16;
else
     s3=(p4-p3);
     t1=(s2-s1); at1=abs(t1);
     t2=(s3-s2); at2=abs(t2);
     if(at1<=at2)
         q=(-f1(1)+9*f1(2)+9*f1(3)-f1(4))/16;
     else
         q=(15/48)*f1(2)+(45/48)*f1(3)-(15/48)*f1(4)+(3/48)*f1(5);
     end
end
f2(2)=f(4)-q;
```

```
% cálculo de los detalles intermedios

for j=3:nk1-2
p1=(f1(j-1)-f1(j-2));
p2=(f1(j)-f1(j-1));
p3=(f1(j+1)-f1(j));
p4=(f1(j+2)-f1(j+1));
p5=(f1(j+3)-f1(j+2));
s1=(p2-p1);
s2=(p3-p2); as2=abs(s2);
s3=(p4-p3); as3=abs(s3);
s4=(p5-p4);

if(as2<=as3)
    t1=(s2-s1); at1=abs(t1);
    t2=(s3-s2); at2=abs(t2);
    if(at2<=at1)
        q=(-f1(j-1)+9*f1(j)+9*f1(j+1)-f1(j+2))/16;
    else
       q=(3/48)*f1(j-2)-(15/48)*f1(j-1)+(45/48)*f1(j)+(15/48)*f1(j+1);
    end
else
   t2=(s3-s2); at2=abs(t2);
   t3=(s4-s3); at3=abs(t3);
   if(at2<=at3)
       q=(-f1(j-1)+9*f1(j)+9*f1(j+1)-f1(j+2))/16;
   else
       q=(15/48)*f1(j)+(45/48)*f1(j+1)-(15/48)*f1(j+2)+(3/48)*f1(j+3);
   end
end

f2(j)=f(2*j)-q;
end

% cálculo del penúltimo detalle

p1=(f1(nk1-2)-f1(nk1-3));
p2=(f1(nk1-1)-f1(nk1-2));
p3=(f1(nk1)-f1(nk1-1));
p4=(f1(nk1+1)-f1(nk1));
s1=(p2-p1);
```

```
s2=(p3-p2); as2=abs(s2);
s3=(p4-p3); as3=abs(s3);

if(as3<=as2)
    q=(-f1(nk1-2)+9*f1(nk1-1)+9*f1(nk1)-f1(nk1+1))/16;
else
    t1=(s2-s1); at1=abs(t1);
    t2=(s3-s2); at2=abs(t2);
    if(at2<=at1)
        q=(-f1(nk1-2)+9*f1(nk1-1)+9*f1(nk1)-f1(nk1+1))/16;
    else
        q=(3/48)*f1(nk1-3)-(15/48)*f1(nk1-2)+(45/48)*f1(nk1-1)+(15/48)*f1(nk1);
    end
end
f2(nk1-1)=f(2*(nk1-1))-q;

% calculo del último detalle

f2(nk1)=f(n-1)-((3/48)*f1(nk1-2)-(15/48)*f1(nk1-1)+(45/48)*f1(nk1)+(15/48)*f1(nk1+1))
```

pphe.m

```
function [f1,f2]=pphe(f,n)

% [f1,f2]=pphe(f,n);
% ---variables de salida----
% f1 valores significativos de la escala inferior
% f2 detalles entre las escalas
% ---variables de entrada----
% f vector original
% n dimensión de f

% valores significativos de la escala inferior

f1=f(1:2:n);
nk1=(n-1)/2;

% cálculo del primer detalle

f2(1)=f(2)-((15./48)*f1(1)+(45./48)*f1(2)-(15./48)*f1(3)+(3./48)*f1(4));

% cálculo de los detalles intermedios

for j=2:nk1-1

d1=f1(j-1)-2*f1(j)+f1(j+1);
d2=f1(j)-2*f1(j+1)+f1(j+2);
d=d1*d2;
s=(f1(j)+f1(j+1))/2;

     if (d>0)
       q=s-d/(4*(d1+d2));
     else
        q=s;
     end

f2(j)=f(2*j)-q;
end

% cálculo del último detalle

f2(nk1)=f(n-1)-((15/48)*f1(nk1+1)+(45/48)*f1(nk1)-(15/48)*f1(nk1-1)+(3/48)*f1(nk1-2))
```

ascender.m

```
function c=ascender(a,l,met)

% c=ascender(a,l,met)
% ---variables de salida---
% c imagen reconstruida a partir de la versión de multirresolucion
% ---variables de entrada---
% a matriz a la que le aplicamos el algorimo ascendente de multirresolucion
% l son los niveles de multirresolucion
% met metodo que quieres utilizar 'lin', 'eno', 'enh', 'pph'

% llamamos al método

n=max(size(a));

if(met=='lin')
     c=decodifli(a,n,l);
end

if(met=='eno')
     c=decodifeno(a,n,l);
end

if(met=='enh')
     c=decodifenoj(a,n,l);
end

if(met=='pph')
    c=decodifpph(a,n,l);
end

% le quitamos la última fila y columna a la imagen


b=c(1:n-1,1:n-1);
clear c;
c=b;
clear b;
```

decodifli.m

```
function c=decodifli(a,n,l)

% c=decodifli(a,n,l);
% ---variables de salida---
% c imagen reconstruida a partir de la versión de multirresolución
% ---variables de entrada---
% a matriz a la que se le aplica la multirresolución
% n número de filas y columnas de a
% l son los niveles de multirresolución

% inicialización de las variables

c=a;
nl=round((n-1)/(2^l));

% se trata del bucle para los niveles de multirresolución

for k=l:-1:1

% reordenamos la matriz haciendo el proceso inverso que en la codificación

nb=2*nl+1;
b=zeros(nb);
b(1:2:nb,1:2:nb)=c(1:nl+1,1:nl+1);
b(2:2:nb-1,2:2:nb-1)=c(nl+2:nb,nl+2:nb);
b(1:2:nb,2:2:nb-1)=c(1:nl+1,nl+2:nb);
b(2:2:nb-1,1:2:nb)=c(2+nl:nb,1:nl+1);

c(1:nb,1:nb)=b;
clear b;

% se hace el algoritmo de multirresolución inverso para las columnas

for j=1:nb
c(1:nb,j)=linealde(c(1:nb,j),nb);
end

% se hace el algoritmo de multirresolución inverso para las filas

for i=1:nb
c(i,1:nb)=linealde(c(i,1:nb),nb);
end
```

```
% se calcula el nl para subir al siguiente nivel

nl=2*nl;
end
```

decodifeno.m

```
function c=decodifeno(a,n,l)

% c=decodifeno(a,n,l)
% ---variables de salida---
% c imagen reconstruida a partir de la versión de multirresolución
% ---variables de entrada---
% a matriz a la que se le aplica la multirresolución
% n número de filas y columnas de a
% l son los niveles de multirresolución

% inicialización de las variables

c=a;
nl=(n-1)/(2^l);

% se trata del bucle para los niveles de multirresolución

for k=l:-1:1

% reordenamos la matriz haciendo el proceso inverso que en la codificación

nb=2*nl+1;
b=zeros(nb);
b(1:2:nb,1:2:nb)=c(1:nl+1,1:nl+1);
b(2:2:nb-1,2:2:nb-1)=c(nl+2:nb,nl+2:nb);
b(1:2:nb,2:2:nb-1)=c(1:nl+1,nl+2:nb);
b(2:2:nb-1,1:2:nb)=c(2+nl:nb,1:nl+1);

c(1:nb,1:nb)=b;
clear b;


% se hace el algoritmo de multirresolución inverso para las columnas

for j=1:nb
c(1:nb,j)=enode(c(1:nb,j),nb);
end

% se hace el algoritmo de multirresolución inverso para las filas

for i=1:nb
c(i,1:nb)=enode(c(i,1:nb),nb);
end
```

```
% se calcula el nl para subir al siguiente nivel
nl=2*nl;
end
```

decodifenoj.m

```
function c=decodifenoj(a,n,l)

% c=decodifenoj(a,n,l)
% ---variables de salida---
% c imagen reconstruida a partir de la versión de multirresolución
% ---variables de entrada---
% a matriz a la que se le aplica la multirresolución
% n número de filas y columnas de a
% l son los niveles de multirresolución

% inicialización de las variables

c=a;
nl=(n-1)/(2^l);

% se trata del bucle para los niveles de multirresolución

for k=l:-1:1


% reordenamos la matriz haciendo el proceso inverso que en la codificación

nb=2*nl+1;
b=zeros(nb);
b(1:2:nb,1:2:nb)=c(1:nl+1,1:nl+1);
b(2:2:nb-1,2:2:nb-1)=c(nl+2:nb,nl+2:nb);
b(1:2:nb,2:2:nb-1)=c(1:nl+1,nl+2:nb);
b(2:2:nb-1,1:2:nb)=c(2+nl:nb,1:nl+1);

c(1:nb,1:nb)=b;
clear b;


% se hace el algoritmo de multirresolución inverso para las columnas

for j=1:nb
c(1:nb,j)=enojde(c(1:nb,j),nb);
end

% se hace el algoritmo de multirresolución inverso para las filas
```

```
for i=1:nb
c(i,1:nb)=enojde(c(i,1:nb),nb);
end


% se calcula el nl para subir al siguiente nivel

nl=2*nl;

end
```

decodifpph.m

```
function c=decodifpph(a,n,l)

% c=decodifpph(a,n,l);
% ---variables de salida---
% c imagen reconstruida a partir de la versión de multirresolución
% ---variables de entrada---
% a matriz a la que se le aplica la multirresolución
% n número de filas y columnas de a
% l son los niveles de multirresolución

% inicialización de las variables

c=a;
nl=(n-1)/(2^l);

% se trata del bucle para los niveles de multirresolución

for k=l:-1:1


% reordenamos la matriz haciendo el proceso inverso que en la codificación

nb=2*nl+1;
b=zeros(nb);
b(1:2:nb,1:2:nb)=c(1:nl+1,1:nl+1);
b(2:2:nb-1,2:2:nb-1)=c(nl+2:nb,nl+2:nb);
b(1:2:nb,2:2:nb-1)=c(1:nl+1,nl+2:nb);
b(2:2:nb-1,1:2:nb)=c(2+nl:nb,1:nl+1);

c(1:nb,1:nb)=b;
clear b;

% se hace el algoritmo de multirresolución inverso para las columnas

for j=1:nb
c(1:nb,j)=pphde(c(1:nb,j),nb);

% se hace el algoritmo de multirresolución inverso para las filas

for i=1:nb
c(i,1:nb)=pphde(c(i,1:nb),nb);
end
```

```
% se calcula el nl para subir al siguiente nivel
nl=2*nl;
end
```

linealde.m

```
function f=linealde(v,n)

% f=linealde(v,n);
% ---variables de salida---
% f valores significativos de la escala superior
% ---variables de entrada---
% v vector que contiene los valores significativos de la escala
inferior
% y los detalles para subir
% n dimensión de v


f=zeros(size(v));
f(1:2:n)=v(1:2:n);

% predicción del primer elemento de f

f(2)=v(2)+((15/48)*v(1)+(45/48)*v(3)-(15/48)*v(5)+(3/48)*v(7));


% predicción de los valores intermedios de f

for j=4:2:n-3

q=(-v(j-3)+9*v(j-1)+9*v(j+1)-v(j+3))/16;

f(j)=v(j)+q;

end

% predicción del último elemento

f(n-1)=v(n-1)+((15/48)*v(n)+(45/48)*v(n-2)-(15/48)*v(n-4)+(3/48)*v(n-6));
```

enode.m

```
function f=enode(v,n)


% f=enode(v,n)
% ---variables de salida---
% f valores significativos de la escala superior
% ---variables de entrada---
% v vector que contiene los valores significativos de la escala
inferior
% y los detalles para subir
% n dimensión de v

f=zeros(size(v));
f(1:2:n)=v(1:2:n);

% predicción del primer elemento de f

f(2)=v(2)+((15/48)*v(1)+(45/48)*v(3)-(15/48)*v(5)+(3/48)*v(7));

% predicción del segundo elemento

at2=abs(-v(1)+3*v(3)-3*v(5)+v(7));
at3=abs(-v(3)+3*v(5)-3*v(7)+v(9));

if(at2<=at3)
   q=(-v(1)+9*v(3)+9*v(5)-v(7))/16;
else
   q=(15/48)*v(3)+(45/48)*v(5)-(15/48)*v(7)+(3/48)*v(9);
end

f(4)=v(4)+q;


% predicción de los valores intermedios de f

for j=6:2:n-5
at1=abs(-v(j-5)+3*v(j-3)-3*v(j-1)+v(j+1));
at2=abs(-v(j-3)+3*v(j-1)-3*v(j+1)+v(j+3));
at3=abs(-v(j-1)+3*v(j+1)-3*v(j+3)+v(j+5));
```

```
if(at2<=at1 & at2<=at3)
     q=(-v(j-3)+9*v(j-1)+9*v(j+1)-v(j+3))/16;
else
     if(at1<=at3)
     q=(3/48)*v(j-5)-(15/48)*v(j-3)+(45/48)*v(j-1)+(15/48)*v(j+1);
     else
     q=(15/48)*v(j-1)+(45/48)*v(j+1)-(15/48)*v(j+3)+(3/48)*v(j+5);
     end
end

f(j)=v(j)+q;
end

% predicción del penúltimo elemento

at1=abs(-v(n-8)+3*v(n-6)-3*v(n-4)+v(n-2));
at2=abs(-v(n-6)+3*v(n-4)-3*v(n-2)+v(n));

if(at2<=at1)
   q=(-v(n-6)+9*v(n-4)+9*v(n-2)-v(n))/16;
else
   q=(3/48)*v(n-8)-(15/48)*v(n-6)+(45/48)*v(n-4)+(15/48)*v(n-2);
end

f(n-3)=v(n-3)+q;


% predicción del último elemento

f(n-1)=v(n-1)+((3/48)*v(n-6)-(15/48)*v(n-4)+(45/48)*v(n-2)+(15/48)*v(n));
```

enojde.m

```
function f=enojde(v,n)

% f=enojde(v,n);
% ---variables de salida---
% f valores significativos de la escala superior
% ---variables de entrada---
% v vector que contiene los valores significativos de la escala
inferior
% y los detalles para subir
% n dimensión de v

f=zeros(size(v));
f(1:2:n)=v(1:2:n);


% predicción del primer elemento de f

f(2)=v(2)+((15/48)*v(1)+(45/48)*v(3)-(15/48)*v(5)+(3/48)*v(7));


% predicción del segundo elemento

p1=(v(3)-v(1));
p2=(v(5)-v(3));
p3=(v(7)-v(5));
p4=(v(9)-v(7));
s1=(p2-p1); as1=abs(s1);
s2=(p3-p2); as2=abs(s2);

if(as1<=as2)
   q=(-v(1)+9*v(3)+9*v(5)-v(7))/16;
else
   s3 =(p4-p3);
   t1=(s2-s1); at1=abs(t1);
   t2=(s3-s2); at2=abs(t2);
   if(at1<=at2)
      q=(-v(1)+9*v(3)+9*v(5)-v(7))/16;
   else
      q=(15/48)*v(3)+(45/48)*v(5)-(15/48)*v(7)+(3/48)*v(9);
   end
```

```
end

f(4)=v(4)+q;


% predicción de los valores intermedios de f

for j=6:2:n-5
p1=(v(j-3)-v(j-5));
p2=(v(j-1)-v(j-3));
p3=(v(j+1)-v(j-1));
p4=(v(j+3)-v(j+1));
p5=(v(j+5)-v(j+3));
s1=(p2-p1);
s2=(p3-p2); as2=abs(s2);
s3=(p4-p3); as3=abs(s3);
s4=(p5-p4);

if(as2<=as3)
       t1=(s2-s1); at1=abs(t1);
       t2=(s3-s2); at2=abs(t2);
       if(at2<=at1)
           q=(-v(j-3)+9*v(j-1)+9*v(j+1)-v(j+3))/16;
       else
           q=(3/48)*v(j-5)-(15/48)*v(j-3)+(45/48)*v(j-1)+(15/48)*v(j+1);
       end
else
       t2=(s3-s2); at2=abs(t2);
       t3=(s4-s3); at3=abs(t3);
       if(at2<=at3)
           q=(-v(j-3)+9*v(j-1)+9*v(j+1)-v(j+3))/16;
       else
           q=(15/48)*v(j-1)+(45/48)*v(j+1)-(15/48)*v(j+3)+(3/48)*v(j+5);
       end
end

f(j)=v(j)+q;

end

% predicción del penúltimo elemento
```

```
p1=(v(n-6)-v(n-8));
p2=(v(n-4)-v(n-6));
p3=(v(n-2)-v(n-4));
p4=(v(n)-v(n-2));
s1=(p2-p1);
s2=(p3-p2); as2=abs(s2);
s3=(p4-p3); as3=abs(s3);

if(as3<=as2)
   q=(-v(n-6)+9*v(n-4)+9*v(n-2)-v(n))/16;
else
   t1=(s2-s1); at1=abs(t1);
   t2=(s3-s2); at2=abs(t2);
   if(at2<=at1)
        q=(-v(n-6)+9*v(n-4)+9*v(n-2)-v(n))/16;
   else
        q=(3/48)*v(n-8)-(15/48)*v(n-6)+(45/48)*v(n-4)+(15/48)*v(n-2);
   end
end
f(n-3)=v(n-3)+q;

% predicción del último elemento

f(n-1)=v(n-1)+((3/48)*v(n-6)-(15/48)*v(n-4)+(45/48)*v(n-2)+(15/48)*v(n));
```

pphde.m

```
function f=pphde(v,n)

% f=pphde(v,n);
% ---variables de salida---
% f valores significativos de la escala superior
% ---variables de entrada---
% v vector que contiene los valores significativos de la escala
inferior
% y los detalles para subir
% n dimensión de v

f=zeros(size(v));
f(1:2:n)=v(1:2:n);

% predicción del primer elemento de f

f(2)=v(2)+((15/48)*v(1)+(45/48)*v(3)-(15/48)*v(5)+(3/48)*v(7));


% predicción de los valores intermedios de f

for j=4:2:n-3
d1=v(j-3)-2*v(j-1)+v(j+1);
d2=v(j-1)-2*v(j+1)+v(j+3);
d=d1*d2;
s=(v(j-1)+v(j+1))/2;

 if (d>0)
       q=s-d/(4*(d1+d2));
 else
       q=s;
 end

f(j)=v(j)+q;

end

% predicción del último elemento

f(n-1)=v(n-1)+((15/48)*v(n)+(45/48)*v(n-2)-(15/48)*v(n-4)+(3/48)*v(n-6));
```

4. Conclusiones

Se ha realizado un estudio para llevar a cabo el denoising mediante algoritmos de multirresolución tanto lineales como no lineales, en particular se ha experimentado con los métodos: lineal, ENO, ENO jerárquico y PPH.

El entorno de multirresolución en el que hemos trabajado es el correspondiente a una discretización por valores puntuales. Es este entorno el que produce de manera más directa una descomposición en escalas de una imagen. Sin embargo llevamos en mente completar el estudio realizado con la implementación de los algoritmos de multirresolución por medias en celdas.

El proceso de truncación ha sido llevado a cabo a través del universal thresholding de Donoho y Johnstone que consiste en lo que es llamado truncación suave de los detalles wavelet mediante el parámetro de truncación $\epsilon_i^k = \sigma\sqrt{2\ln(M_i^k)}$, donde M_i^k es el tamaño de la matriz de detalles.

Hemos realizado diversos experimentos numéricos con diferentes imágenes reales y los resultados obtenidos nos muestran que la imagen recuperada aparece difuminada en todos los casos. Esta difusión en las imágenes es un efecto típico en los procesos de eliminación de ruido. Para reducir el efecto difusivo de estos métodos se propone utilizar algoritmos no lineales (tipo PPH) que se adapten mejor a la geometría de cada imagen.

El software utilizado ha sido: el lenguaje de programación Matlab para la implementación de los algoritmos y su posterior aplicación a las imágenes digitales, el editor de textos científico Latex para la realización de la memoria y en particular el paquete Beamer para la elaboración de la presentación.

Referencias

[1] Amat S., Aràndiga F., Cohen A. and Donat R., (2002). Tensor product multiresolution analysis with error control for compact image representation. *Signal Processing*, **82**(4), 587-608.

[2] Amat S., Aràndiga F., Cohen A., Donat R., García G. and von Oehsen M., (2001). Data compression with ENO schemes: A case study. *Applied and Computational Harmonic Analysis*, **11**, 273-288.

[3] Amat S., Donat R., Liandrat J. and Trillo J.C. (2006), Analysis of a new nonlinear subdivision scheme. Applications in image processing. Foundations of Computational Mathematics, **6(2)**, 193-226.

[4] Aràndiga F. and Donat R., (2000). Nonlinear Multi-scale Decomposition: The Approach of A.Harten, Numerical Algorithms, **23**, 175-216.

[5] Aràndiga F., Donat R. and Harten A., (1999) Multiresolution Based on Weighted Averages of the Hat Function I: Linear Reconstruction Operators, *SIAM J. Numer. Anal.*, **36**, 160-203.

[6] Aràndiga F., Donat R. and Harten A., (1999). Multiresolution Based on Weighted Averages of the Hat Function II: Nonlinear Reconstruction Operators, *SIAM J. Sci. Comput.*, **20**(3), 1053-1099.

[7] Bachelli Silvia, Papi Serena, (2004). Filtered wavelet thresholding mehthods. Journal of Computational and Applied Mathematics, **164-165**, 39-52.

[8] Delauries G. and Dubuc S., (1989). Symmetric Iterative Interpolation Scheme, *Constr. Approx.* **5**, 49-68.

[9] Donoho I. Johnstone, (1994). Ideal spatial adaptation by wavelet shrinkage, Briometika, **81**, 425-455.

[10] Donoho D., (1994). Interpolating wavelet transforms. Prepint Standford University.

[11] Donoho D., (1995). Denoising by soft thresholding, *IEEE Trans. on Inform. Theory*, **41**(3), 613-627.

[12] Harten A., (1993). Discrete multiresolution analysis and generalized wavelets, *J. Appl. Numer. Math.*, **12**, 153-192.

[13] Harten A., (1996). Multiresolution representation of data II, *SIAM J. Numer. Anal*, **33**(3), 1205-1256.

[14] Harten A., Osher S.J., Engquist B. and Chakravarthy S.R., (1987). Some results on uniformly high-order accurate essentially non-oscillatory schemes, *Appl. Numer. Math.*, **2**, 347-377.

[15] Harten A., Engquist B., Osher S.J. and Chakravarthy S.R., (1987) Uniformly high order accurate essentially non-oscillatory schemes III, *J. Comput. Phys*, **71**, 231-303.

[16] Micchelli C.A., (1996). Interpolatory subdivision schemes and wavelets, Journal of Approximation Theory, **86**, 41-71.

[17] Rabbani M. and Jones P.W., (1991). Digital Image Compression Techniques. Tutorial Text, *Society of Photo-Optical Instrumentation Engineers (SPIE)*, TT07.