Estudio de Filtros Óptimos para la Eliminación de Ruido en Imágenes Digitales usando Algoritmos de Multirresolución No Lineales

Autor:
Jorge García Pérez

Directores:
Sergio Amat Plata
Juan Carlos Trillo Moya

Mi más sincero agradecimiento:

A toda mi familia porque sin ellos no hubiera llegado hasta aquí,
A Fran y a Mery por todas sus aportaciones

A mis directores ***Sergio*** *y* ***Juan Carlos***
por su ayuda, sus consejos y su apoyo

¡Muchas gracias!

Autor	Jorge García Pérez
E-mail del Autor	JorgeGP1982@gmail.com
Directores	Sergio Amat Plata, Juan Carlos Trillo Moya
E-mail del Director	jctrillo@upct.es
Título del PFC	Estudio de Filtros Óptimos para la Eliminación de Ruido en Imágenes Digitales usando Algoritmos de Multirresolución No Lineales
Descritores	Eliminación de ruido, imágenes digitales, multirresolución
Resúmen	Los algoritmos de multirresolución están principalmente basados en el uso de dos operadores, el operador de discretización y el operador de reconstrucción. El operador de discretización debe ser lineal, y depende de la aplicación en concreto se elige uno u otro. Usualmente en el campo de las imágenes digitales se usa el operador de discretización por medias en celda. El operador de reconstrucción por su parte puede ser no lineal, y así abre muchas posibilidades para adaptarse a la imagen en concreto con la que se está trabajando. Dentro de los algoritmos de multirresolución es muy importante el uso de un operador de reconstrucción apropiado. Los métodos de eliminación de ruido que se estudian en este proyecto son el método lineal de interpolación de Lagrange, que implica operadores lineales de interresolución, el método ENO (esencialmente no oscilatorios), que construye trozos o partes polinomiales usando datos pertenecientes a las regiones suaves de la función en la medida de lo posible, el método WENO que hace una media ponderada de los trozos polinomiales posibles en la reconstrucción, y por último el método PPH (piecewise polynomial harmonic), basado en una interpolación no lineal a trozos polinomial.
Titulación	Ingeniero Técnico de Telecomunicaciones, Especialidad Telemática
Departamento	Matemática Aplicada y Estadística
Fecha de Presentación	Julio de 2009

Índice de Tablas

Índice de figuras

Índice

1. Introducción

La imagen digital se ha convertido en algo imprescindible en la sociedad actual. Prácticamente cualquier aparato tecnológico que sale al mercado puede reproducir o capturar una imagen digital. Posteriormente las imágenes digitales pueden ser tratadas en el ordenador. Estudios como el de este proyecto sobre la imagen digital son de importancia para el desarrollo de este campo tan de actualidad.

Las imágenes se pueden representar mediante retículas de celdillas a las que vamos asignando valores. Cada una de las celdillas de dicha retícula se llama píxel. Una forma muy importante de clasificar las imágenes de mapa de bits es saber cuánta información se asigna a cada píxel. Así, en una imagen en escala de grises se utilizan 8 bits por píxel, dando 256 valores distintos. Para una imagen a color se debe utilizar como mínimo 24 bits por píxel divididos en tres grupos de ocho. Lo que se hace es superponer tres canales de luz, uno rojo, otro verde y otro azul, cada uno con 256 posibilidades de tono.

Entendemos por ruido en imágenes digitales cualquier valor de un píxel de una imagen que no se corresponde exactamente con la realidad. El ruido se debe, la mayoría de las veces, al equipo electrónico utilizado en la captación de las imágenes y al ruido añadido en los tramos de transmisión. Este ruido no se puede eliminar pero se puede corregir gracias a los algoritmos de eliminación de ruido. En este proyecto se estudian y comparan diferentes métodos. Vamos a incidir más en los métodos basados en una descomposición de multirresolución ([5], [11], [6], [7]). Otros métodos que consideraremos están basados en consideraciones estadísticas. Así por ejemplo haremos comparaciones entre los métodos de multirresolución y los filtros de vecindad, de mediana, el filtro gaussiano, y el filtro de Wiener.

Los algoritmos de multirresolución están principalmente basados en el uso de dos operadores, el operador de discretización y el operador de reconstrucción. El operador de discretización debe ser lineal, y depende de la aplicación en concreto se elige uno u otro. Usualmente en el campo de las imágenes digitales se usa el operador de discretización por medias en celda. El operador de reconstrucción por su parte puede ser no lineal, y así abre muchas posibilidades para adaptarse a la imagen en concreto con la que se está trabajando.

Dentro de los algoritmos de multirresolución es muy importante el uso de un operador de reconstrucción apropiado. Los métodos de eliminación de ruido que se estudian en este proyecto son el método lineal de interpolación

de Lagrange, que implica operadores lineales de inter-resolución, el método ENO (esencialmente no oscilatorios), que construye trozos o partes polinomiales usando datos pertenecientes a las regiones suaves de la función en la medida de lo posible, el método WENO que hace una media ponderada de los trozos polinomiales posibles en la reconstrucción, y por último el método PPH (piecewise polynomial harmonic), basado en una interpolación no lineal a trozos polinomial.

A lo largo de este proyecto, como se ha dicho, se desarrolla un estudio para lograr la eliminación, total o parcial, de ruido en las fotografías digitales, y en particular nos vamos a centrar en técnicas de multirresolución, que consisten en una descomposición de una señal en escalas, estas descomposiciones pueden ser tanto lineales como no lineales ([19],[8],[4]).
Una vez que se ha obtenido la versión de multirresolución de una imagen, el siguiente paso consiste en procesar las bandas de altas frecuencias que se corresponden con los detalles necesarios para pasar de una escala a la siguiente. Este procesado se realiza típicamente mediante unos filtros de truncamiento: nosotros estudiaremos el filtro duro, el filtro blando, y el filtro polinomial. También de pasada implementaremos el filtro exponencial. Dentro de cada uno de estos filtros encontramos también la necesidad de elegir unos parámetros adecuados de truncamiento. Se estudiaran diferentes posibilidades.

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 mediante métodos de multirresolución por medias en celda. incluyendo algunos experimentos numéricos. En la sección 3 se presentan otros métodos clásicos de eliminación de ruido, comparándolos numéricamente con los anteriores. En la sección 4 se explican las interfaces gráficas desarrolladas para permitir al usuario la fácil ejecución de los programas de eliminación de ruido presentados anteriormente. En la sección 5 aparece el código Matlab de los programas utilizados en el proyecto. Y por último en la sección 6 se presentan las conclusiones extraídas del proyecto.

2. Eliminación de ruido en imágenes digitales mediante métodos de multirresolución por medias en celda

En esta sección vamos a presentar la teoría básica necesaria para entender el proceso de eliminación de ruido en imágenes digitales usando algoritmos de multirresolución y truncamiento mediante diferentes filtros. En particular vamos a describir diversos algoritmos de multirresolución por medias en celda tanto en 1D como en 2D, vamos a estudiar diferentes tipos de ruido posibles, y analizaremos también algunas maneras de llevar a cabo la truncación de los coeficientes de detalles.

2.1. Algoritmos de multirresolución por medias en celda en 1D

Consideramos el conjunto de redes anidadas en $[0,1]$:

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

donde J_0 es un entero fijado y tomamos la discretización

$$\mathcal{D}_k : L^1[0,1] \to V^k, \qquad f_j^k = (\mathcal{D}_k f)_j = \frac{1}{h_k}\int_{x_{j-1}^k}^{x_j^k} f(x)dx, \quad 1 \le j \le J_k, \quad (1)$$

donde $L^1[0,1]$ es el espacio de funciones absolutamente integrables en $[0,1]$ y V^k es el espacio de secuencias con J_k componentes.

De las propiedades básicas de la integral obtenemos:

$$f_j^{k-1} = \frac{1}{h_{k-1}}\int_{x_{j-1}^{k-1}}^{x_j^{k-1}} f(x)dx = \frac{1}{2h_k}\int_{x_{2j-2}^k}^{x_{2j}^k} f(x)dx = \frac{1}{2}(f_{2j-1}^k + f_{2j}^k).$$

Consideremos ahora la secuencia $\{F_j^k\}$ dada por

$$F_j^k = h_k \sum_{i=1}^{j} f_i^k = \int_0^{x_j^k} f(x)dx = F(x_j^k) \quad \Rightarrow \quad f_j^k = \frac{F_j^k - F_{j-1}^k}{h_k}. \qquad (2)$$

La función $F(x)$ es una función primitiva de $f(x)$ y los valores de la secuencia $\{F_j^k\}$ corresponden a la discretización por valores puntuales de $F(x)$ en el mallado $\{X^k\}$.

Denotaremos por $\mathcal{I}_{k-1}(x; F^{k-1})$ a la reconstrucción de F en el entorno de valores puntuales que cumple

$$\mathcal{I}_{k-1}(x_j^{k-1}; F^{k-1}) = F_j^{k-1},$$

es decir, se trata de una interpolación.

Así, podemos encontrar una aproximación $\tilde{f}_p^k$, de f_p^k:

$$\tilde{f}_p^k = (\mathcal{I}_{k-1}(x_p^k, F^{k-1}) - \mathcal{I}_{k-1}(x_{p-1}^k, F^{k-1}))/h_k. \tag{3}$$

Este será el valor de nuestra predicción.

Se definen los errores ε^{k_p} como la diferencia entre el valor exacto f_p^k y el aproximado $\tilde{f}_p^k$,

$$\varepsilon^{k_p} = f_p^k - \tilde{f}_p^k$$

La predicción de errores satisface

$$0 = \frac{\varepsilon_{2j-1}^k + \varepsilon_{2j}^k}{2},$$

y en particular podemos definir los detalles como

$$d_j^k \quad = \quad \varepsilon_{2j-1}^k, \quad 1 \leq j \leq J_{k-1},$$

teniendo toda la información necesaria ya que

$$\begin{aligned} \varepsilon_{2j-1}^k &= d_j^k, \qquad 1 \leq j \leq J_{k-1}, \\ \varepsilon_{2j}^k &= -d_j^k, \quad 1 \leq j \leq J_{k-1}. \end{aligned}$$

Por construcción, denotando P_{k-1}^k el operador de predicción, tenemos

$$f_{2j-1}^k = (P_{k-1}^k f^{k-1})_{2j-1} + d_j^{k-1}. \tag{4}$$

Una reconstrucción no lineal conduce a una multirresolución no lineal. Las reconstrucciones en el entorno de medias en celda usualmente se construyen al igual que el operador de predicción, haciendo uso de una reconstrucción para la función primitiva en el entorno de valores puntuales. La expresión del operador reconstrucción es en este caso

$$R_k(x, f^k) = \frac{d}{d_x} I_k(x, F^k)$$

2.2. Operadores de predicción lineales y no lineales

Vamos a definir los operadores de reconstrucción y de predicción asociados a la multirresolución por valores puntuales, ya que es en este entorno donde la definición es más natural y más fácilmente entendible. Posteriormente se obtendrán los operadores de reconstrucción y de predicción en el entorno de medias en celda gracias a la técnica anteriormente expuesta de la función primitiva.

2.2.1. 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

$$D_k : \begin{cases} C([0,1]) & \to V^k \\ f & \mapsto f^k = (f_j^k)_{j=0}^{J_k} = (f(x_j^k))_{j=0}^{J_k} \end{cases} \tag{5}$$

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

$$R_k : V^k \to C([0,1]); \quad \text{y satisface} \quad D_k R_k f^k = f^k, \tag{6}$$

lo cual significa que

$$(R_k f^k)(x_j^k) = f_j^k = f(x_j^k). \tag{7}$$

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

$$f^L \to Mf^L \begin{cases} \text{Do} \quad k = L, \ldots, 1 & \\ \quad f_j^{k-1} = f_{2j}^k & 0 \le j \le J_{k-1}, \\ d_j^k \quad = f_{2j-1}^k - I_{k-1}(x_{2j-1}^k; f^{k-1}) & 1 \le j \le J_{k-1}. \end{cases} \tag{8}$$

y

$$Mf^L \rightarrow M^{-1}Mf^L \begin{cases} \text{Do} \quad k = 1, \ldots, L & \\ f^k_{2j-1} = I_{k-1}(x^k_{2j-1}; f^{k-1}) + d^k_j & 1 \leq j \leq J_{k-1}, \\ \quad f^k_{2j} = f^{k-1}_j & 0 \leq j \leq J_{k-1}. \end{cases} \tag{9}$$

Las técnicas de interpolación más usuales son los polinomios.

2.2.2. *Técnicas de reconstrucción lineal: La Interpolación de Lagrange*

Tomando $\mathcal{S}$ como el conjunto:

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

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

$$L_m(y) = \prod_{l=-s, l \neq m}^{-s+r} (\frac{y-l}{m-l}), \quad L_m(j) = \delta^m_j, \quad j \in \mathcal{S}$$

La interpolación de Lagrange para:

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

$$\mathcal{I}_k(x, f^k) = \sum_{m=-s}^{-s+r} f^k_{j+m} L_m(\frac{x - x^k_j}{h_k}), \quad x \in [x^k_{j-1}, x^k_j], \quad 1 \leq j \leq J_k.$$

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 $x \in [x^k_{j-1}, x^k_j]$. 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 $[x^k_{j-1}, x^k_j]$. Por ejemplo, para $s = 2(r = 3)$ se obtiene la siguiente transformada de multirresolución:

$$\begin{cases} \text{Do} \quad k = L, \ldots, 1 & \\ \quad f_j^{k-1} = f_{2j}^k & 0 \le j \le J_{k-1}, \\ d_j^k = f_{2j-1}^k - (\frac{-f_{j-2}^{k-1} + 9f_{j-1}^{k-1} + 9f_j^{k-1} - f_{j+1}^{k-1}}{16}), & 1 \le j \le J_{k-1}. \end{cases} \tag{10}$$

$$\begin{cases} \text{Do} \quad k = 1, \ldots, L & \\ f_{2j}^k = f_j^{k-1} & 0 \le j \le J_{k-1}, \\ f_{2j-1}^k = d_j^k + (\frac{-f_{j-2}^{k-1} + 9f_{j-1}^{k-1} + 9f_j^{k-1} - f_{j+1}^{k-1}}{16}), & 1 \le j \le J_{k-1}. \end{cases} \tag{11}$$

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:

$$f(x) = \mathcal{I}_k^L(x, f^k) + O([f]) \tag{12}$$

Cuando se utiliza interpolación lineal centrada, cada stencil (conjunto de puntos usados para interpolar) se elige de forma independiente de los datos, es decir, no se tiene en cuenta la suavidad de la función que se interpola. Esto significa que una singularidad aislada en un intervalo I_k producirá una pérdida de exactitud en varios subintervalos de la partición, ya que los stencils correspondientes a los subintervalos adyacentes a I_k contendrán también la singularidad, con lo que el error de interpolación vendrá dado por la expresión (12).

2.2.3. *Técnicas de reconstrucción no lineal: La 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 punto clave en la interpolación ENO es el proceso 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 $r \ge 2$ puntos incluyendo los puntos x_{j-1}^k y x_j^k.

Denominemos el stencil ENO para el jth intervalo:

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

siendo $r+1$ el orden de la interpolación.

Existen dos estrategias para realizar esta selección: la estrategia jerárquica y la no jerárquica.

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

(1) La selección jerárquica del stencil consiste en, partiendo de los extremos del intervalo $\left[x_{j-1}^k, x_j^k\right]$, ir añadiendo puntos a derecha o izquierda, comparando las diferencias divididas correspondientes a los conjuntos formados por los extremos del intervalo más los puntos añadidos, y escogiendo la de menor valor absoluto. El algoritmo para este procedimiento es el siguiente:

Set $s_0 = j$
for $l = 0, \ldots, r-2$
  if $|f[x_{s_l-2}^k, \ldots, x_{s_l+l}^k]| < |f[x_{s_l-1}^k, \ldots, x_{s_l+l+1}^k]|$
    $s_{l+1} = s_l - 1$
  else
    $s_{l+1} = s_l$
  end
end
$s_j = s_{r-1}$

y,

(2) La selección no jerárquica del stencil considera las diferencias divididas de mayor orden correspondientes a todos los stencils posibles, y calcula el mínimo de entre todos los valores absolutos de dichas diferencias. El algoritmo es el siguiente:

Se elige un s_j tal que

Choose s_j such that
$|f[x_{s_j-1}^k, \ldots, x_{s_j+r-1}^k])| = \text{mín}_{j-r+1 \le l \le j}\{|f[x_{l-1}^k, \ldots, x_{l+r-1}^k]|\}$

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.

Notar además, que no hay diferencia entre ambos algoritmos cuando $r=2$, pero los stencils obtenidos pueden variar cuando $r>2$. En cualquier caso, se observa que el stencil siempre contiene los extremos del subintervalo en el que se realiza la interpolación, evitando siempre que sea posible los intervalos que contienen singularidades.

En el caso en que $r=3$ tenemos tres posibles stencils (ver Figura 1), es decir,

$$S_j^1 = \left\{x_{j-3}^k, x_{j-2}^k, x_{j-1}^k, x_j^k\right\},$$
$$S_j^2 = \left\{x_{j-2}^k, x_{j-1}^k, x_j^k, x_{j+1}^k\right\},$$
$$S_j^3 = \left\{x_{j-1}^k, x_j^k, x_{j+1}^k, x_{j+2}^k\right\}.$$

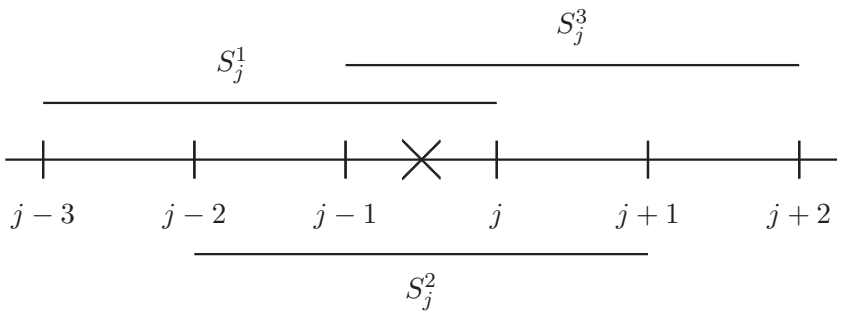


Figura 1: Stencil ENO.

El algoritmo de multirresolución queda,

$$\begin{cases} \text{Do} \quad k = L, \ldots, 1 \\ \quad f_j^{k-1} = f_{2j}^k \\ d_j^k = \begin{cases} f_{2j-1}^k - (5f_{j-3}^{k-1} + 15f_{j-2}^{k-1} - 5f_{j-1}^{k-1} + f_j^{k-1}), & si\ S_j^1 \\ f_{2j-1}^k - (\frac{-f_{j-2}^{k-1}+9f_{j-1}^{k-1}+9f_j^{k-1}-f_{j+1}^{k-1}}{16}), & si\ S_j^2 \\ f_{2j-1}^k - (5f_{j-1}^{k-1} + 15f_j^{k-1} - 5f_{j+1}^{k-1} + f_{j+2}^{k-1}), & si\ S_j^3 \end{cases} \end{cases} \tag{13}$$

$$\begin{cases} \text{Do} \quad k = 1, \ldots, L \\ \quad f_{2j}^k = f_j^{k-1} \\ f_{2j-1}^k = \begin{cases} d_j^k + (5f_{j-3}^{k-1} + 15f_{j-2}^{k-1} - 5f_{j-1}^{k-1} + f_j^{k-1}), & si\ S_j^1 \\ d_j^k + (\frac{-f_{j-2}^{k-1}+9f_{j-1}^{k-1}+9f_j^{k-1}-f_{j+1}^{k-1}}{16}), & si\ S_j^2 \\ d_j^k + (5f_{j-1}^{k-1} + 15f_j^{k-1} - 5f_{j+1}^{k-1} + f_{j+2}^{k-1}), & si\ S_j^3 \end{cases} \end{cases} \tag{14}$$

2.2.4. *Técnicas de reconstrucción no lineal: La Interpolación WENO*

Mediante la interpolación ENO se consigue una aproximación con un buen orden de exactitud en todos los intervalos excepto en aquellos que contienen singularidades. Pero existen algunas características de esta técnica de interpolación que se podrían mejorar:

En primer lugar, el proceso de selección del stencil es muy sensible a las perturbaciones: si los valores de las diferencias divididas que se comparan en el criterio de selección son muy similares, una pequeña perturbación, como un error de redondeo, por ejemplo, podría hacer que el stencil seleccionado cambiase.

Por otra parte, en las regiones suaves de la función no es necesario realizar esta selección de stencil, ya que el stencil centrado que utiliza la interpolación lineal producirá la misma aproximación.

Por último, se podría aumentar el orden de exactitud de la aproximación, ya que el método ENO consiste en seleccionar uno de entre todos los stencils posibles, para hacer la interpolación en cada subintervalo. Tomando stencils formados por r intervalos, esto significa que la interpolación ENO se

hará seleccionando un stencil entre r posibles, dando una aproximación con un orden de exactitud igual a r. Hay $2r-1$ subintervalos contenidos en los r stencils, por lo que se pierde la información proporcionada por $r-1$ de estos intervalos. Si la función es suave en estas regiones, esta información podría servir para obtener una mejor aproximación, de manera que utilizando la información dada por los $2r-1$ subintervalos contenidos en los distintos stencils, se podría obtener un orden de exactitud máximo igual a $2r$ (en el caso de valores puntuales).

Para solucionar los dos primeros problemas, se presentó en [18] y en [28] una estrategia de sesgo, que consiste en tomar como base un stencil preferido, que será el stencil centrado en el intervalo donde se realiza la interpolación, y utilizarlo para modificar el criterio de selección de stencil con un parámetro de sesgo. La idea es no alejarse de este stencil, excepto en el caso de que en el stencil alternativo la función sea mucho más suave, y este mucho viene dado por el sesgo.

Posteriormente, Liu et al. [25] introdujeron, en el contexto de las leyes de conservación, la técnica WENO, como mejora de la interpolación ENO. Una versión más eficiente de esta técnica fue propuesta por Jiang y Shu en [23]. La diferencia entre ENO y WENO radica en la forma en que se construye el interpolante. El interpolante ENO se construye seleccionando un stencil para cada subintervalo, mientras que en el método WENO, a cada subintervalo se le asignan todos los stencils posibles, y el polinomio interpolador se calcula como una combinación lineal convexa de los polinomios correspondientes a dichos stencils. En esta combinación lineal, se da más importancia a los polinomios construidos a partir de stencils en los que la función es suave, de forma que los polinomios que cruzan alguna singularidad tienen un efecto prácticamente nulo. De este modo, se conserva el efecto ENO, es decir, la interpolación en intervalos próximos a singularidades se hace tomando información sólo de regiones donde la función es suave, y además, como se utiliza una combinación convexa de polinomios interpoladores, los errores cometidos por unos se pueden cancelar con los de otros, resultando en un orden de aproximación mayor.

El algoritmo de multirresolución en este caso es

$$\begin{cases} \text{Do} \quad k = L, \ldots, 1 \\ \quad f_{j-1}^{k-1} = f_{2j}^{k} \\ d_j^k = f_{2j-1}^k - (w_{j-1}^{k-1} s_{j-1}^{k-1} + w_j^{k-1} s_j^{k-1} + w_{j+1}^{k-1} s_{j+1}^{k-1}) \end{cases} \tag{15}$$

$$\begin{cases} \text{Do} \quad k = 1, \ldots, L \\ \quad f_{2j}^{k} = f_{j-1}^{k-1} \\ f_{2j-1}^k = d_j^k + (w_{j-1}^{k-1} s_{j-1}^{k-1} + w_j^{k-1} s_j^{k-1} + w_{j+1}^{k-1} s_{j+1}^{k-1}) \end{cases} \tag{16}$$

donde

$$s_{j-1}^{k-1} = 5f_{j-3}^{k-1} + 15f_{j-2}^{k-1} - 5f_{j-1}^{k-1} + f_j^{k-1}$$

$$s_j^{k-1} = \frac{-1f_{j-2}^{k-1} + 9f_{j-1}^{k-1} + 9f_j^{k-1} - 1f_{j+1}^{k-1}}{16}$$

$$s_{j+1}^{k-1} = 5f_{j-1}^{k-1} + 15f_j^{k-1} - 5f_{j+1}^{k-1} + f_{j+2}^{k-1}$$

es decir, los valores en el punto x_{2j-1}^k de los polinomios interpoladores basados en los stencils S_j^1, S_j^2 y S_j^3 respectivamente (ver Figura 1).

Y los pesos w_{j-1}^{k-1}, w_j^{k-1} y w_{j+1}^{k-1} se calculan mediante las expresiones

$$w_{j-1}^{k-1} = \frac{\alpha_{j-1}^{k-1}}{\alpha_{j-1}^{k-1} + \alpha_j^{k-1} + \alpha_{j+1}^{k-1}}$$

$$w_j^{k-1} = \frac{\alpha_j^{k-1}}{\alpha_{j-1}^{k-1} + \alpha_j^{k-1} + \alpha_{j+1}^{k-1}}$$

$$w_{j+1}^{k-1} = \frac{\alpha_{j-1}^{k-1}}{\alpha_{j-1}^{k-1} + \alpha_j^{k-1} + \alpha_{j+1}^{k-1}}$$

Los valores de α_{j-1}^{k-1}, α_j^{k-1} y α_{j+1}^{k-1} son

$$\alpha_{j-1}^{k-1} = \frac{\frac{3}{16}}{\epsilon + IS_{j-1}^{k-1}}, \quad \alpha_j^{k-1} = \frac{\frac{10}{16}}{\epsilon + IS_j^{k-1}}, \quad \alpha_{j+1}^{k-1} = \frac{\frac{3}{16}}{\epsilon + IS_{j+1}^{k-1}}.$$

con ϵ una constante positiva que se introduce para evitar que el denominador se anule, y suele tener el valor $\epsilon = 10^{-5}$ o $\epsilon = 10^{-6}$.

Finalmente necesitamos saber cómo calcular los indicadores de suavidad IS_{j-1}^{k-1}, IS_j^{k-1} y IS_{j+1}^{k-1}. Una posibilidad es (ver Liu et al. [25]).

$$\begin{aligned} IS_{j-1}^{k-1} &= \left(f\left[x_{j-1}^{k-1}, x_j^{k-1}\right] - 2f\left[x_{j-2}^{k-1}, x_{j-1}^{k-1}\right] + f\left[x_{j-3}^{k-1}, x_{j-2}^{k-1}\right]\right)^2 \\ &+ \frac{1}{2}\left(f\left[x_{j-2}^{k-1}, x_{j-1}^{k-1}\right] - f\left[x_{j-3}^{k-1}, x_{j-2}^{k-1}\right]\right)^2 \\ &+ \frac{1}{2}\left(f\left[x_{j-1}^{k-1}, x_j^{k-1}\right] - f\left[x_{j-2}^{k-1}, x_{j-1}^{k-1}\right]\right)^2, \end{aligned}$$

$$\begin{aligned} IS_j^{k-1} &= \left(f\left[x_j^{k-1}, x_{j+1}^{k-1}\right] - 2f\left[x_{j-1}^{k-1}, x_j^{k-1}\right] + f\left[x_{j-2}^{k-1}, x_{j-1}^{k-1}\right]\right)^2 \\ &+ \frac{1}{2}\left(f\left[x_{j-1}^{k-1}, x_j^{k-1}\right] - f\left[x_{j-2}^{k-1}, x_{j-1}^{k-1}\right]\right)^2 \\ &+ \frac{1}{2}\left(f\left[x_j^{k-1}, x_{j+1}^{k-1}\right] - f\left[x_{j-1}^{k-1}, x_j^{k-1}\right]\right)^2, \end{aligned}$$

$$\begin{aligned} IS_{j+1}^{k-1} &= \left(f\left[x_{j+1}^{k-1}, x_{j+2}^{k-1}\right] - 2f\left[x_j^{k-1}, x_{j+1}^{k-1}\right] + f\left[x_{j-1}^{k-1}, x_j^{k-1}\right]\right)^2 \\ &+ \frac{1}{2}\left(f\left[x_j^{k-1}, x_{j+1}^{k-1}\right] - f\left[x_{j-1}^{k-1}, x_j^{k-1}\right]\right)^2 \\ &+ \frac{1}{2}\left(f\left[x_{j+1}^{k-1}, x_{j+2}^{k-1}\right] - f\left[x_j^{k-1}, x_{j+1}^{k-1}\right]\right)^2. \end{aligned}$$

2.2.5. *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 (ver [4]). 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 $\mathcal{I}_k^P(x, f^k)$. 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, $\mathcal{I}_k^P(x, f^k) = \tilde{P}_j(x, f^k)$, 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,

$$P_j(x) = a_0 + a_1(x - x_{j-\frac{1}{2}}) + a_2(x - x_{j-\frac{1}{2}})^2 + a_3(x - x_{j-\frac{1}{2}})^3. \tag{17}$$

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

$$\begin{cases} a_0 - a_1\frac{3}{2}h + a_2(\frac{3}{2}h)^2 - a_3(\frac{3}{2}h)^3 &= f_{j-2}, \\ a_0 - a_1\frac{1}{2}h + a_2(\frac{1}{2}h)^2 - a_3(\frac{1}{2}h)^3 &= f_{j-1}, \\ a_0 + a_1\frac{1}{2}h + a_2(\frac{1}{2}h)^2 + a_3(\frac{1}{2}h)^3 &= f_j, \\ a_0 + a_1\frac{3}{2}h + a_2(\frac{3}{2}h)^2 + a_3(\frac{3}{2}h)^3 &= f_{j+1}. \end{cases} \tag{18}$$

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

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

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

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

por la media armónica

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

siempre que $D_{j-1}D_j > 0$. 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

$$|2\frac{D_{j-1}D_j}{D_{j-1} + D_j}| \leq 2\,\text{mín}(|D_{j-1}|, |D_j|) = O(1), \tag{19}$$

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

$$\tilde{\tilde{a}}_1 := \frac{-e_{j-\frac{3}{2}} + 13e_{j-\frac{1}{2}}}{12}. \tag{20}$$

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

$$\begin{aligned} f_{j-\frac{1}{2}} &\approx \frac{-f_{j-2} + 18f_{j-1} - 9f_j}{8} - \frac{1}{8}12\tilde{a}_1 h \\ &= \frac{f_j + f_{j-1}}{2} - \frac{1}{4}\frac{(f_j - 2f_{j-1} + f_{j-2})(f_{j+1} - 2f_j + f_{j-1})}{(f_{j+1} - f_j - f_{j-1} + f_{j-2})}, \end{aligned} \tag{21}$$

ó

$$\begin{aligned} f_{j-\frac{1}{2}} &\approx \frac{-f_{j-2} + 18f_{j-1} - 9f_j}{8} - \frac{1}{8}12\tilde{\tilde{a}}_1 h \\ &= \frac{f_j + f_{j-1}}{2}, \end{aligned} \tag{22}$$

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.

El algoritmo de multirresolución queda,

$$\begin{cases} \text{Do} \quad k = L, \ldots, 1 \\ \quad f_j^{k-1} = f_{2j}^k \\ d_j^k = \begin{cases} f_{2j-1}^k - (\frac{f_{j-1}^k + f_j^k}{2} - \frac{1}{4}\frac{Df_{j-1}^k Df_j^k}{Df_{j-1}^k + Df_j^k}), & si\ Df_{j-1}^k Df_j^k > 0 \\ f_{2j-1}^k - (\frac{f_{j-1}^k + f_j^k}{2}), & en\ otro\ caso \end{cases} \end{cases} \tag{23}$$

$$\begin{cases} \text{Do} \quad k = 1, \ldots, L \\ \quad f_{2j}^k = f_j^{k-1} \\ f_{2j-1}^k = \begin{cases} d_j^k + (\frac{f_{j-1}^k + f_j^k}{2} - \frac{1}{4}\frac{Df_{j-1}^k Df_j^k}{Df_{j-1}^k + Df_j^k}), & si\ Df_{j-1}^k Df_j^k > 0 \\ d_j^k + (\frac{f_{j-1}^k + f_j^k}{2}), & en\ otro\ caso \end{cases} \end{cases} \tag{24}$$

donde $Df^k_{j-1} = f^k_{j-2} - 2f^k_{j-1} + f^k_j$ y $Df^k_j = f^k_{j-1} - 2f^k_j + f^k_{j+1}$.

2.3. Paso a 2D: algoritmos basados en producto tensor

Para generalizar los algoritmos arriba definidos para secuencias en dos dimensiones (imágenes) ha sido usada una aproximación por producto tensor que es presentada brevemente aquí.

Representando el array dos dimesional $f = \{f^L_{i,j}\}_{(i,j)=1}^{J_L}$ por la matriz $A = A^L$, la representación multirresolución de A^L es

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

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 los errores de reconstrucción

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

Tradicionalmente, un paso de la descomposición multirresolutiva se representa en forma matricial de la siguiente manera

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

2.4. Tipos de ruido en imágenes digitales.

Entendemos por ruido en imágenes digitales cualquier valor de un píxel de una imagen que no se corresponde exactamente con la realidad. El ruido

se debe, la mayoría de las veces al equipo electrónico utilizado en la captación de las imágenes y al ruido añadido en los tramos de transmisión. Tenemos que distinguir al menos tres clases diferentes de ruido:

- Ruido gaussiano: es el ruido cuya densidad de probabilidad responde a una distribución normal (o distribución de Gauss). Este tipo de ruido es el que más aparece en los equipos electrónicos.
- Ruido impulsivo: es no continuo y está constituido por picos irregulares de corta duración y amplitud relativamente grande. Estos picos se generan por diversas causas, por ejemplo, por perturbaciones electromagnéticas producidas por tormentas atmosféricas o por defectos en los sistemas de comunicación.
- Ruido sal y pimienta: Se caracteriza por la aparición de píxeles con valores arbitrarios normalmente detectables porque se diferencian mucho de sus vecinos más próximos.

En este proyecto nos vamos a centrar en el ruido gaussiano.

2.5. Elección de filtros y de tolerancias de truncación

A continuación se va a exponer brevemente la manera en que se realiza el truncamiento de los coeficientes de detalle de las versiones de multirresolución.

Los tres métodos más extendidos para realizar el truncamiento son truncamiento fuerte, truncamiento blando y truncamiento polinomial ($p = 2$). En este proyecto, aunque se van a explicar los tres, sólo se utilizarán el truncamiento blando y el truncamiento polinomial ($p = 2$) puesto que ambos ofrecen mejores resultados en la práctica (en el caso del polinomial siempre y cuando p esté entre dos y tres).

Truncamiento fuerte: Se considera un valor del parámetro de truncamiento ϵ, y se compara con el valor absoluto del detalle. Si el valor del detalle 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 d_j^k :

$$\hat{d}_j^k = \begin{cases} 0 & |d_j^k| \leq \epsilon \\ d_i^k & \text{en otro caso} \end{cases}$$

Truncamiento blando: Se toma el valor del parámetro de truncamiento ϵ y se compara con el valor del detalle. Si el valor absoluto del detalle es menor que ϵ, directamente se le asigna al detalle el valor 0 y si el valor absoluto del detalle es mayor que el valor de ϵ, se le asigna el valor resultante de restar el valor absoluto del detalle menos ϵ multiplicado por el signo del detalle. Es decir, para un detalle d_j^k:

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

Truncamiento polinomial: Queda reflejado en [11] como la mejor solución para el problema del denoising. Este truncamiento está basado en la función cuadrática

$$F_m(c,\epsilon) = \begin{cases} (1 - \frac{\epsilon^m}{|c|^m}) & \text{si} \quad |c| > \epsilon, \\ 0 & \text{en otro caso,} \end{cases} \tag{25}$$

que actua de filtro, es decir que $\hat{d}_i^k = F_m(d_i^k, \epsilon)d_i^k$. Los nuevos valores $\hat{d}_i^k$ para $m = 1$ son los que se obtienen con el filtro blando. También se obtienen buenos resultados para $m \in [2, 3]$, más concretamente con $m = 2$. Siendo éste el valor de m que tomaremos en el apartado de experimentos numéricos dentro de este proyecto.

Además de tener en cuenta la elección del tipo de 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. Esta manera de proceder fue introducida por Donoho y Johnstone en [17]. En este proyecto hemos trabajado con esta elección así como con pequeñas variaciones de la misma. La elección de Donoho consiste en

$$\varepsilon_1^k = \sigma\sqrt{2 \cdot \ln(M_1^k)} \tag{26}$$

$$\varepsilon_2^k = \sigma\sqrt{2 \cdot \ln(M_2^k)} \tag{27}$$

$$\varepsilon_3^k = \sigma\sqrt{2 \cdot \ln(M_3^k)} \tag{28}$$

donde M_1^k, M_2^k, M_3^k se corresponden con el tamaño de las matrices $\Delta_1^k, \Delta_2^k, \Delta_3^k$, $k = 1, 2, ..., L$, siendo L el números de niveles de multirresolución y σ^2 representa la varianza del ruido.

Una variación que produce también buenos resultados es la siguiente:

$$\varepsilon_1^k = \frac{\sigma\sqrt{2\cdot\ln(M_1^k)}}{(k+1)\cdot(k+1)} \tag{29}$$

$$\varepsilon_2^k = \frac{\sigma\sqrt{2\cdot\ln(M_2^k)}}{(k+1)\cdot(k+1)} \tag{30}$$

$$\varepsilon_3^k = \frac{\sigma\sqrt{2\cdot\ln(M_3^k)}}{(k+1)\cdot(k+1)}. \tag{31}$$

Truncamiento Exponencial: En los programas Matlab hemos incluido también la posibilidad de llevar a cabo el truncamiento mediante un filtro exponencial (ver [11] para más detalles), pero los resultados no parecen ser muy favorables.

2.6. Experimentos numéricos

En esta sección presentaremos los resultados, numéricos y visuales, de algunos experimentos. Vamos a trabajar con imágenes a color, y el proceso de reducción del ruido será hecho en las tres bandas.

Aplicando la transformada inversa de multirresolución a la versión truncada de la multirresolución directa de $\bar{f}^L$, calculamos

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

donde $\bar{f}^L$ es conseguida de f^L añadiéndole ruido. Nosotros vamos a considerar principalmente ruido gaussiano.

En nuestro primer experimento añadimos a la imagen 'dibujo' un ruido gaussiano con varianza 0.01 (ver Figura 2). A continuación comparamos los resultados obtenidos por los diferentes métodos de reconstrucción propuestos, es decir, método lineal basado en interpolación segmentaria de Lagrange, método ENO jerárquico, método ENO no jerárquico, método WENO y método PPH. En las Figuras 2, 3,4 vemos cómo han eliminado el ruido en la imagen los diferentes métodos. Es claro observar que se produce una difusión de la imagen al reconstruirla, siendo dicha difusión menor en los métodos no lineales. En particular el resultado obtenido con los métodos PPH y WENO superan claramente los otros métodos. Y entre ellos la calidad visual de la imagen parece un poco mejor para el método PPH.

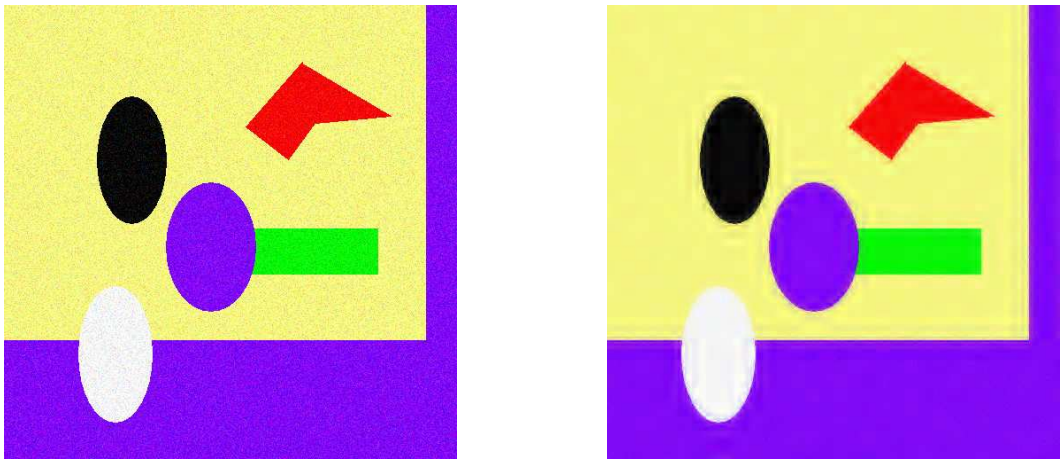


Figura 2: A la izquierda imagen con ruido Gaussiano de varianza 0.01 y a la derecha imagen reconstruida con método Lineal, $L = 4$, Umbral suave y Donoho método 2.

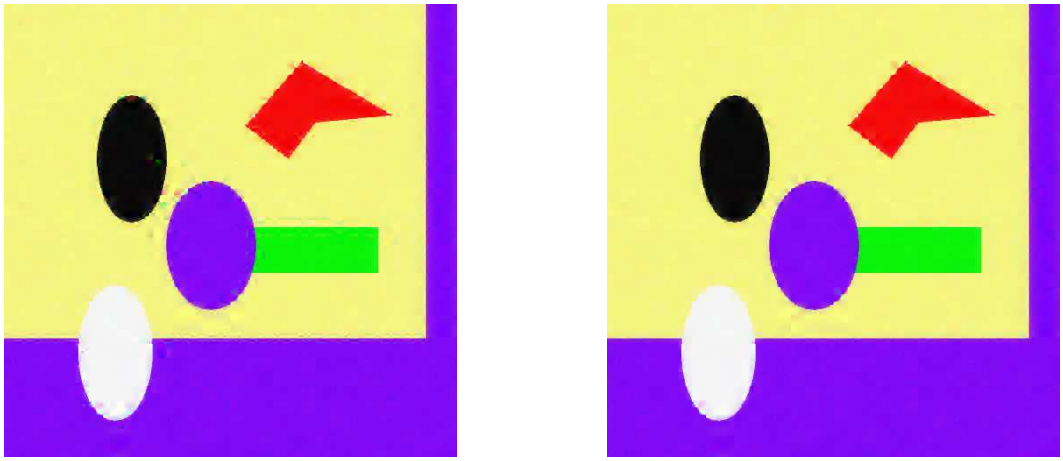


Figura 3: A la izquierda imagen reconstruida con método ENO y a la derecha imagen reconstruida con método ENO jerarquico, $L = 4$, Umbral suave y Donoho método 2.

También vamos a trabajar con la imagen 'girl' que aparece en la Figura 5, a la que le hemos introducido ruido gaussiano de varianza 0.01. Los resultados obtenidos pueden verse en las Figuras 5,6,7.

Finalmente queremos también añadir un experimento donde vamos a introducir una gran cantidad de ruido gaussiano, varianza 0.1, a la imagen

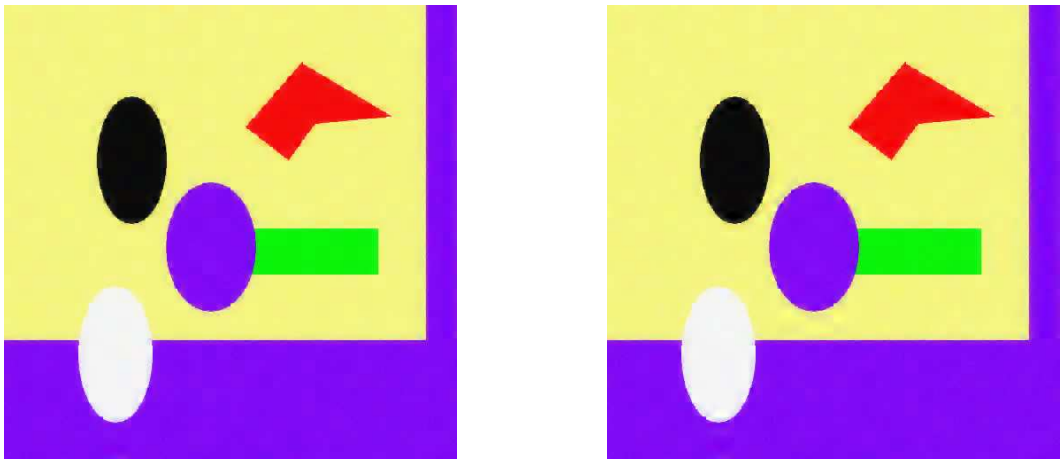


Figura 4: A la izquierda imagen reconstruida con método PPH y a la derecha imagen reconstruida con método WENO, $L = 4$, Umbral suave y Donoho método 2.

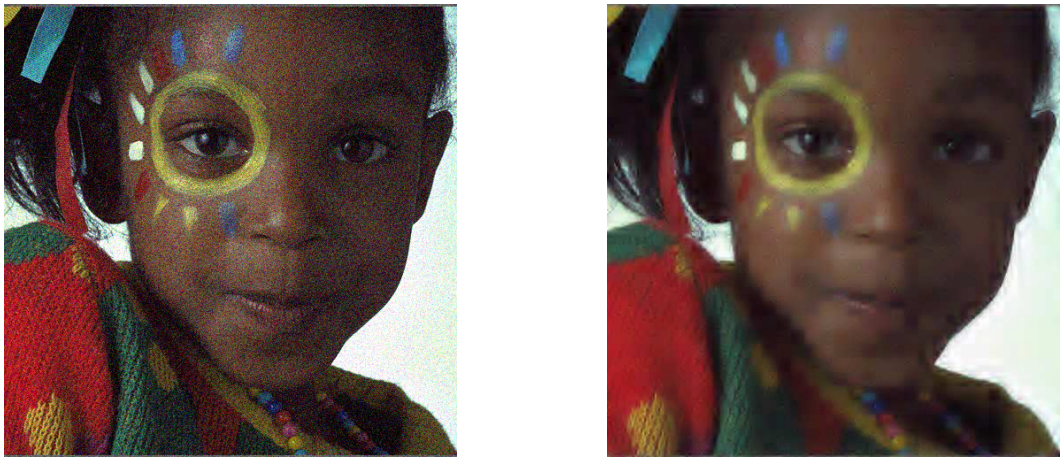


Figura 5: A la izquierda imagen con ruido Gaussiano de varianza 0.01 y a la derecha imagen reconstruida con método Lineal, $L = 4$, Umbral suave y Donoho método 2.

'girl', vease la Figura 8. Y posteriormente vamos a reconstruir la señal con el método de multirresolulión PPH pero con distintos valores estimados de la desviación típica del ruido. Esto lo hacemos porque cuando en los métodos de multirresolución se truncan demasiados detalles, el resultado obtenido es pobre debido al efecto de pixelado que se produce. Es por ello que cuando la

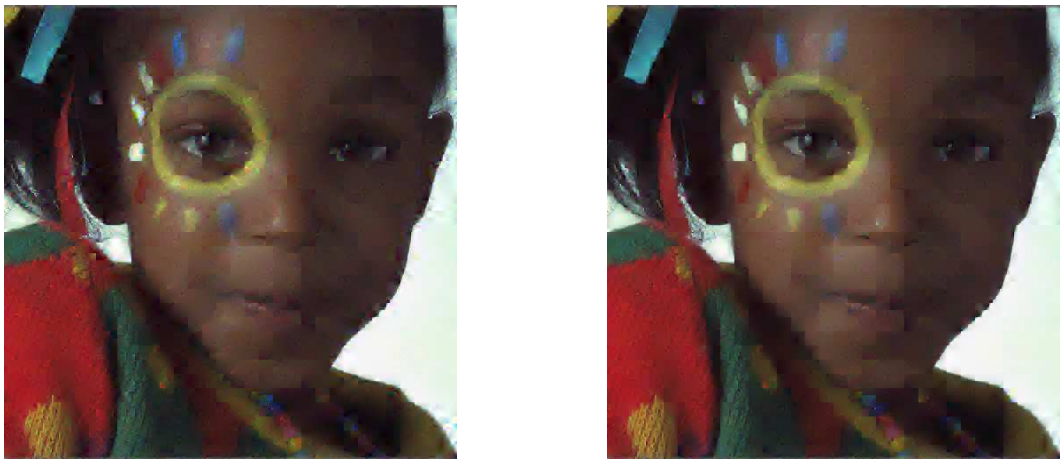


Figura 6: A la izquierda imagen reconstruida con método ENO y a la derecha imagen reconstruida con método ENO jerarquico, $L = 4$, Umbral suave y Donoho método 2.

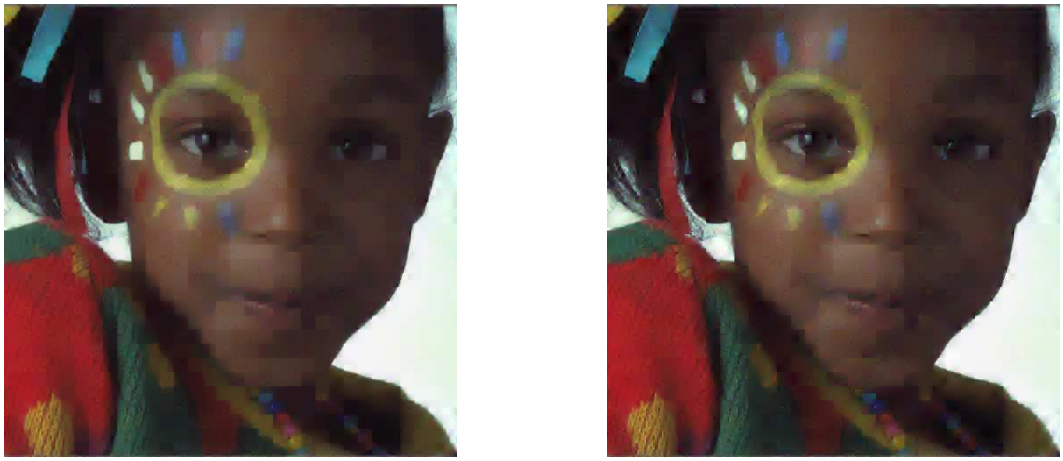


Figura 7: A la izquierda imagen reconstruida con método PPH y a la derecha imagen reconstruida con método WENO, $L = 4$, Umbral suave y Donoho método 2.

imagen es muy ruidosa, en vez de intentar eliminar 'todo el ruido', es mejor estrategia para estos métodos contentarse con eliminar parte del ruido y no llegar a producir el efecto indeseado de pixelación. Los resultados obtenidos pueden verse en la Figura 9.

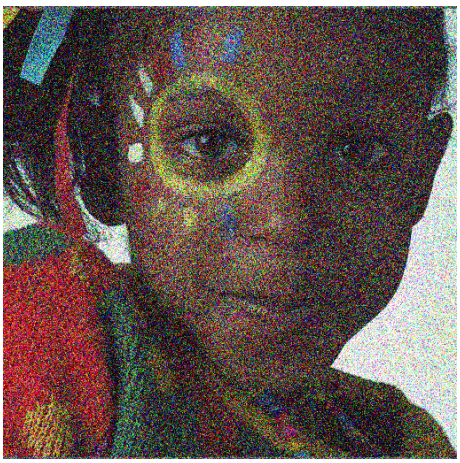
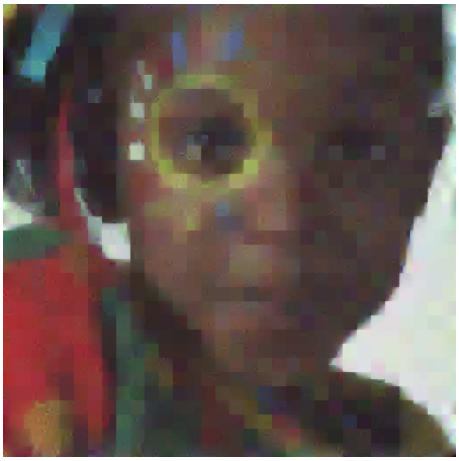


Figura 8: Imagen con ruido Gaussiano de varianza 0.1.



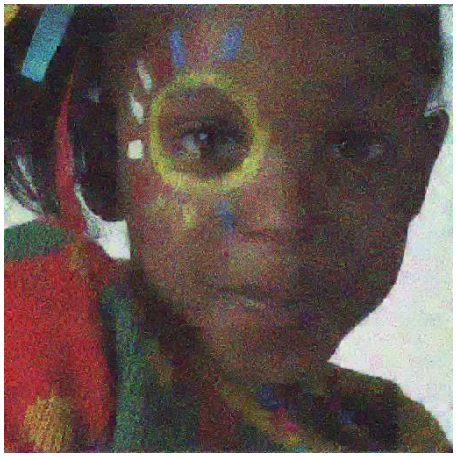


Figura 9: A la izquierda imagen reconstruida con método PPH y a la derecha imagen reconstruida con método PPH pero para una estimación de la desviación típica del ruido igual a 30, $L = 4$, Umbral suave y Donoho método 2.

3. Eliminación de ruido en imagenes digitales mediante filtros

3.1. Filtro de Gauss

La función principal del filtro Gaussiano es la eliminación de los detalles y el ruido presente en la imagen mediante un operador de convolución bidimensional. El núcleo gaussiano tiene la siguiente expresión:

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

La idea principal es que el filtro Gaussiano intenta hacer una media ponderada entre una representación discreta de la distribución bidimensional Gaussiana y los píxeles de la imagen. El valor central de la matriz de distribución discreta esta situado sobre el píxel que queremos tratar y entonces el suavizado Gaussiano puede ser realizado usando métodos estandar de convolución.

3.2. Filtro de Mediana

Los filtros de suavizado lineales o filtros paso bajo tienden a difuminar los ejes a causa de que las altas frecuencias de una imagen son atenuadas. La visión humana es muy sensible a esta información de alta frecuencia. La preservación y el posible realce de este detalle es muy importante al filtrar. Cuando el objetivo es más la reducción del ruido que el difuminado, el empleo de los filtros de mediana representan una posibilidad alternativa.

A menudo, las imágenes digitales se corrompen con ruido durante la transmisión o en otras partes del sistema. Esto se ve a menudo en las imágenes convertidas a digital de una señal de la televisión. Usando técnicas del filtrado de ruido, el ruido puede ser suprimido y la imagen corrompida se puede restaurar a un nivel aceptable. En aplicaciones de ingeniería eléctrica, el ruido se elimina comúnmente con un filtro paso bajo. El filtrado paso bajo es satisfactorio para quitar el ruido gaussiano pero no para el ruido impulsivo. Una imagen corrupta por ruido impulsivo tiene varios píxeles que tienen intensidades visiblemente incorrectas como 0 o 255. Hacer un filtrado paso bajo alterarán estas señales con los valores extremos sobre la vecindad del píxel. Un método mucho más eficaz para eliminar el ruido impulsivo es el

filtrado de mediana.

En el filtrado de mediana, el nivel de gris de cada píxel se reemplaza por la mediana de los niveles de gris en un entorno de este píxel, en lugar de por la media. Recordar que la mediana M de un conjunto de valores es tal que la mitad de los valores del conjunto son menores que M y la mitad de los valores mayores que M, es decir en un conjunto ordenado de mayor a menor o viceversa, sería el valor de la posición central.

El filtro de la mediana no puede ser calculado con una máscara de convolución, ya que es un filtro no lineal. Podemos ver como este tipo de filtro elimina totalmente el punto que tenia un valor muy diferente al resto de sus vecinos. Como se selecciona el valor de centro, el filtrado de mediana consiste en forzar que puntos con intensidades muy distintas se asemejen más a sus vecinos, por lo que observamos que el filtro de mediana es muy efectivo para eliminar píxeles cuyo valor es muy diferente del resto de sus vecinos, como por ejemplo eliminando ruido de la imagen.

Al implementar el filtro de mediana encontramos el mismo problema de bordes que teníamos en la convolución: cuándo la ventana de filtrado está centrada en el píxel (0,0), el filtrado lineal (media) da un mejor resultado a la hora de eliminar ruido gaussiano, mientras que el filtrado no lineal (mediana) es más adecuado a la hora de eliminar ruido impulsivo.

3.3. Filtro de Vecindad

Dada una imagen $f(x, y)$ de tamaño nxm, para aplicar un filtro a la imagen es necesario definir una matriz que contendrá los coeficientes del filtro, lo que a su vez define los píxeles del entorno que serán utilizados como argumento del filtro que alterará el valor del píxel. A esta matriz se le denomina máscara con dimensión [m, n]. El valor del nivel de gris de la imagen suavizada $g(x, y)$ en el punto (x, y) se obtiene promediando valores de nivel de gris de los puntos de f contenidos en una cierta vecindad de (x, y).

$$g(x, y) = \frac{1}{M} \sum_{(n,m)\in S} f(n, m)$$

donde $x, y = 0, 1, ..., N-1$. S es el conjunto de coordenadas de los puntos vecinos a (x, y), incluyendo el propio (x, y), y M es el número de puntos de

la vecindad. Por ejemplo, imaginemos la subimagen y la máscara siguientes:

$f(x-1,y-1)$	$f(x,y-1)$	$f(x+1,y-1)$
$f(x-1,y)$	$f(x,y)$	$f(x+1,y)$
$f(x-1,y+1)$	$f(x,y+1)$	$f(x+1,y+1)$

Tabla 1: Valores de los pixeles de la subimagen

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Tabla 2: Valores de la máscara del filtro

y que queremos reemplazar el valor de $f(x,y)$ por el promedio de los puntos en una región de tamaño 3x3 centrada en (x,y), es decir, queremos asignar el valor promedio a $g(x,y)$:

$$\begin{aligned} g(x,y) &= 1/9(f(x-1,y-1)+f(x-1,y)+f(x+1,y-1) \\ &+ f(x-1,y)+f(x,y)+f(x+1,y) \\ &+ f(x-1,y-1)+f(x,y+1)+f(x+1,y+1)) \end{aligned}$$

Esta operación se puede realizar de forma general centrando la máscara en (x,y) y multiplicando cada punto debajo de la máscara por el correspondiente coeficiente de la matriz y sumando el resultado.

El problema del filtro de vecindad es que aparece de manera muy notable la difuminación de bordes.

3.4. Filtro de Wiener

El filtro de wiener es un filtro de suavizado que se adapta de manera local a la imagen. Donde la varianza es mayor, wiener aplica menor suavizado. Y donde la varianza es menor, wiener aplica un suavizado mucho mayor. Esta aproximación suele producir un mejor resultado que los filtros lineales. Como consecuencia, el filtrado de wiener requiere mucho más procesamiento.

3.5. Experimentos numéricos con filtros clásicos de suavizado

Nosotros estamos trabajando en este proyecto con ruido gaussiano, pero los programas realizados permiten tanto añadir como eliminar ruido de otros tipos, tales como impulsivo o sal y pimienta. Hacemos notar que no todos los filtros son igualmente útiles para todos los tipos de ruido. Así por ejemplo el filtro de mediana sería más adecuado que el de vecindad para eliminar ruido impulsivo y viceversa.

A continuación ofrecemos una bateria de experimentos realizados con imágenes contaminadas con ruido gaussiano. En las Figuras 10,11,12,13,14 podemos ver el experimento con el programa de suavizado, en el cual introducimos una varianza igual a 0.01 y eliminamos el ruido con los distintos métodos del programa y comparamos a la vez la salida de cada método para una máscara 3x3 y para una máscara 7x7.

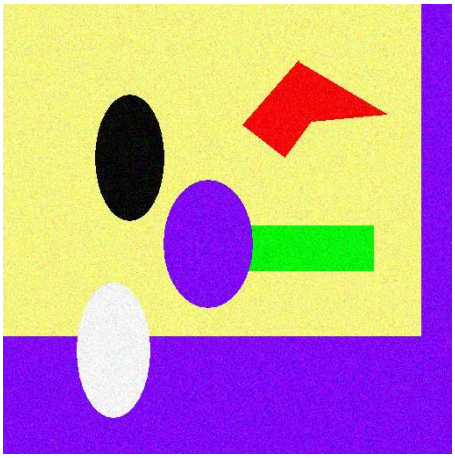


Figura 10: Imagen con ruido Gaussiano de varianza 0.01.

En las Figuras 15,16,17,18,19 podemos ver el experimento con el programa de suavizado, en el cual introducimos una varianza igual a 0.1 y eliminamos el ruido con los distintos métodos del programa y comparamos a la vez la salida de cada método para una máscara 3x3 y para una máscara 7x7.

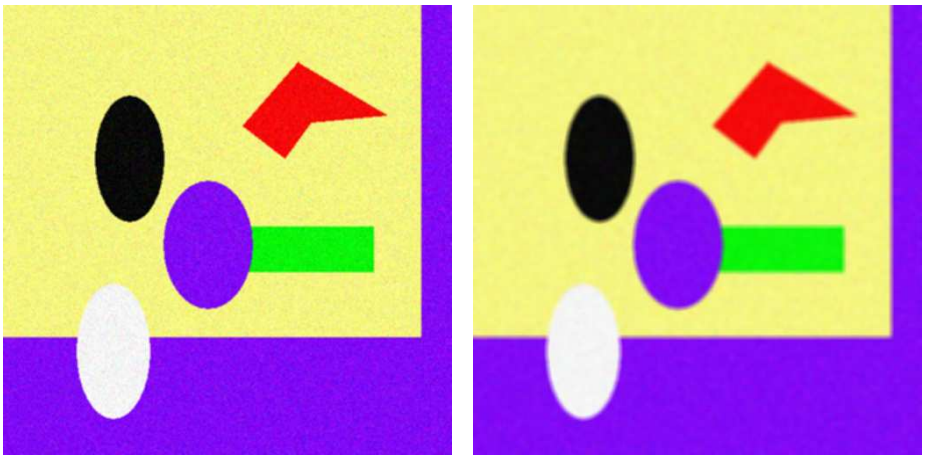


Figura 11: Ambas imágenes han sido reconstruidas con el método Gaussiano con sigma igual a 10. A la izquierda es el caso para una máscara 3x3, y a la derecha para una máscara 7x7.

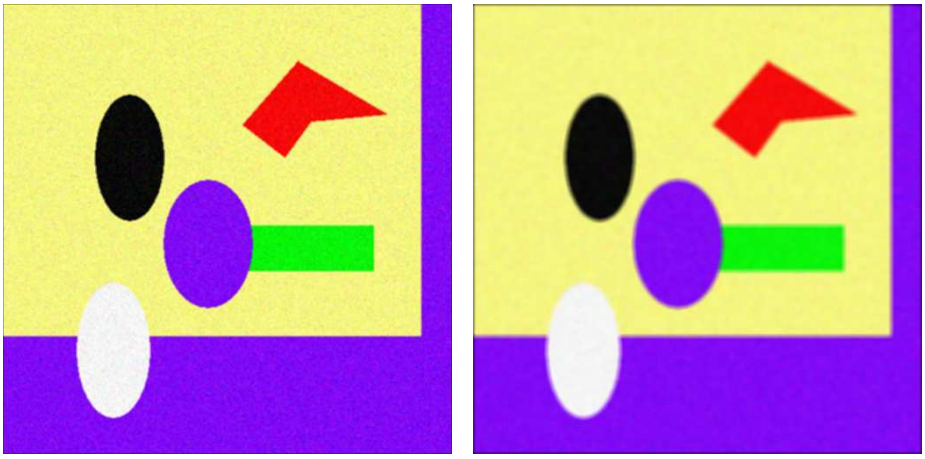


Figura 12: Ambas imágenes han sido reconstruidas con el filtro de vecindad. A la izquierda es el caso para una máscara 3x3, y a la derecha para una máscara 7x7.

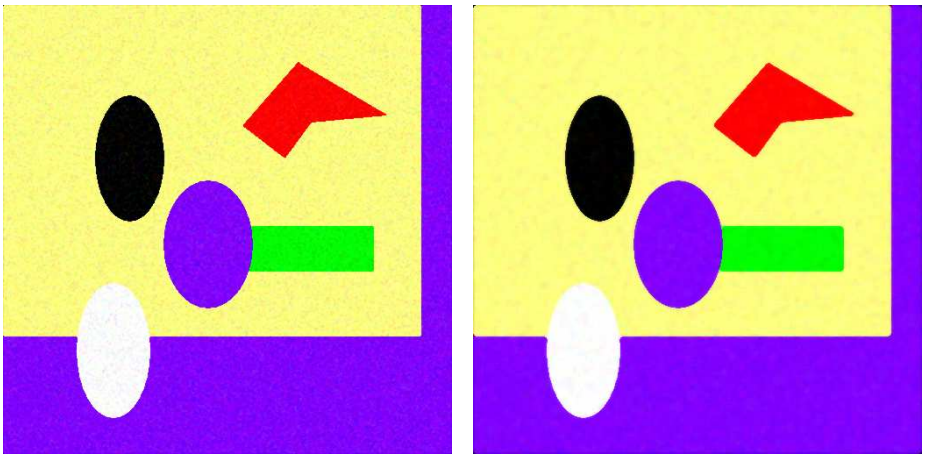


Figura 13: Ambas imágenes han sido reconstruidas con el filtro de mediana. A la izquierda es el caso para una máscara 3x3, y a la derecha para una máscara 7x7.

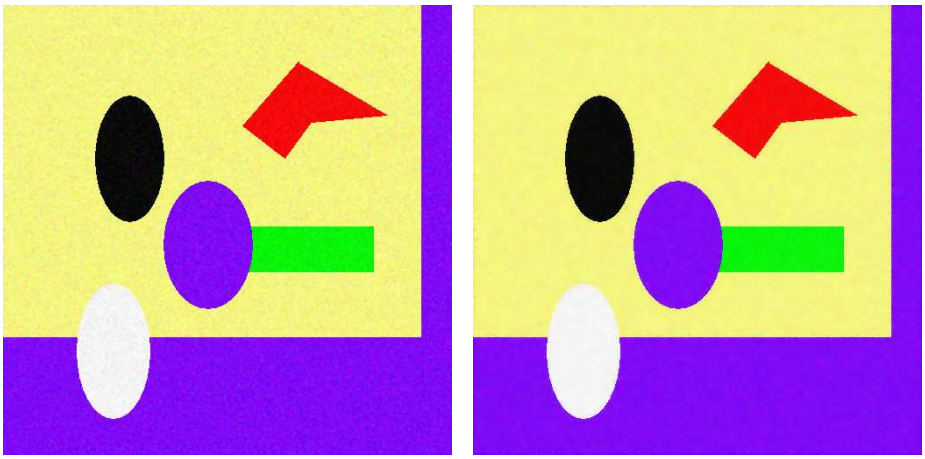


Figura 14: Ambas imágenes han sido reconstruidas con el filtro de wiener. A la izquierda es el caso para una máscara 3x3, y a la derecha para una máscara 7x7.

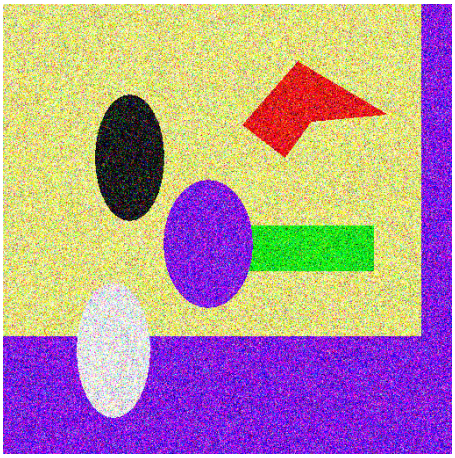


Figura 15: Imagen con ruido Gaussiano de varianza 0.1.

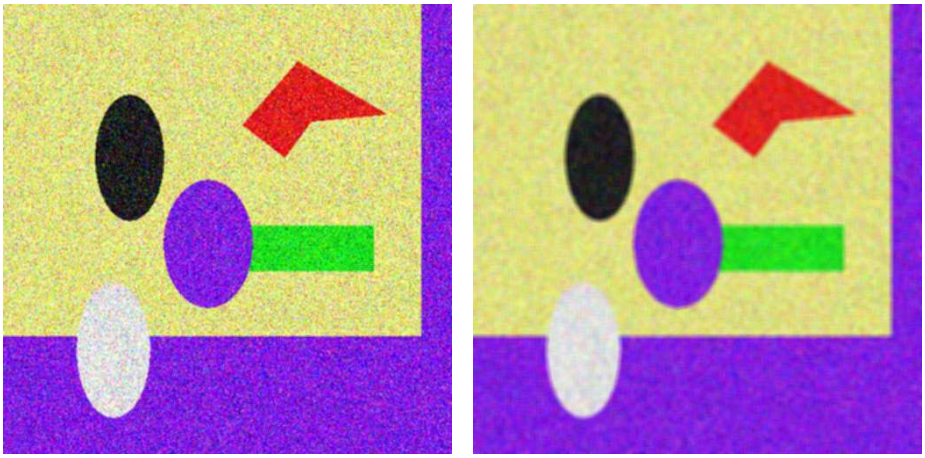


Figura 16: Ambas imágenes han sido reconstruidas con el método Gaussiano con sigma igual a 10. A la izquierda es el caso para una máscara 3x3, y a la derecha para una máscara 7x7.

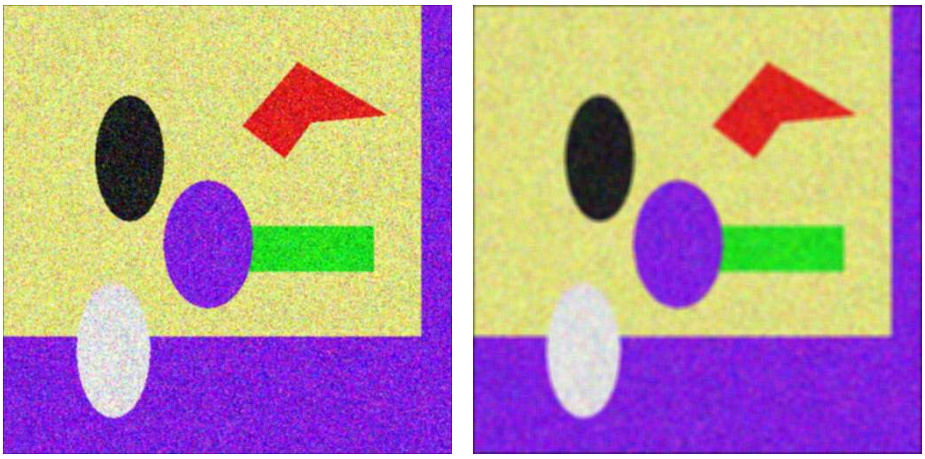


Figura 17: Ambas imágenes han sido reconstruidas con el filtro de vecindad. A la izquierda es el caso para una máscara 3x3, y a la derecha para una máscara 7x7.

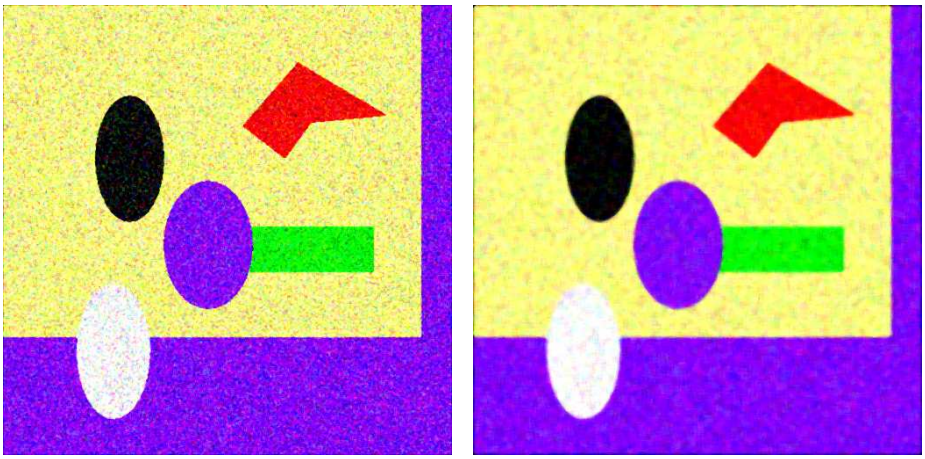


Figura 18: Ambas imágenes han sido reconstruidas con el filtro de mediana. A la izquierda es el caso para una máscara 3x3, y a la derecha para una máscara 7x7.

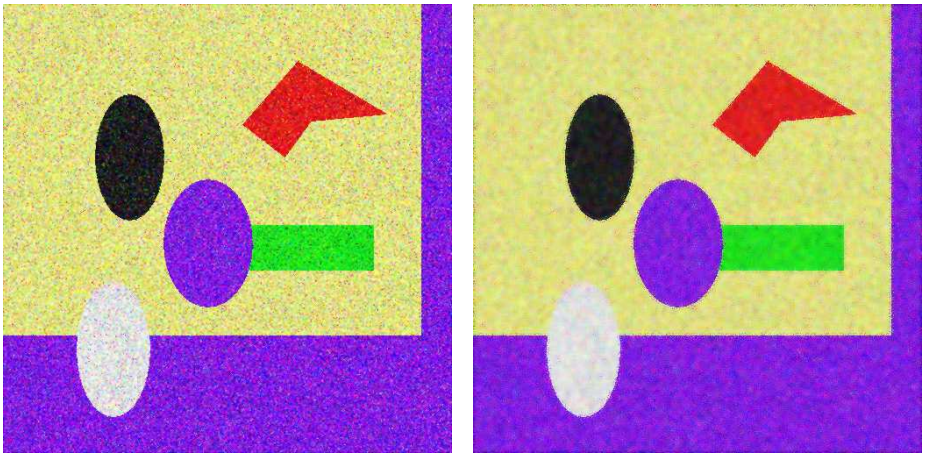


Figura 19: Ambas imágenes han sido reconstruidas con el filtro de wiener. A la izquierda es el caso para una máscara 3x3, y a la derecha para una máscara 7x7.

En las Figuras 20,21,22,23,24 podemos ver el experimento realizado anteriormente con la imagen 'dibujo', pero esta vez utilizaremos la imagen 'girl'. Recordemos que en estas primeras figuras el valor de la varianza es igual a 0.01.

Figura 20: Imagen con ruido Gaussiano de varianza 0.01.

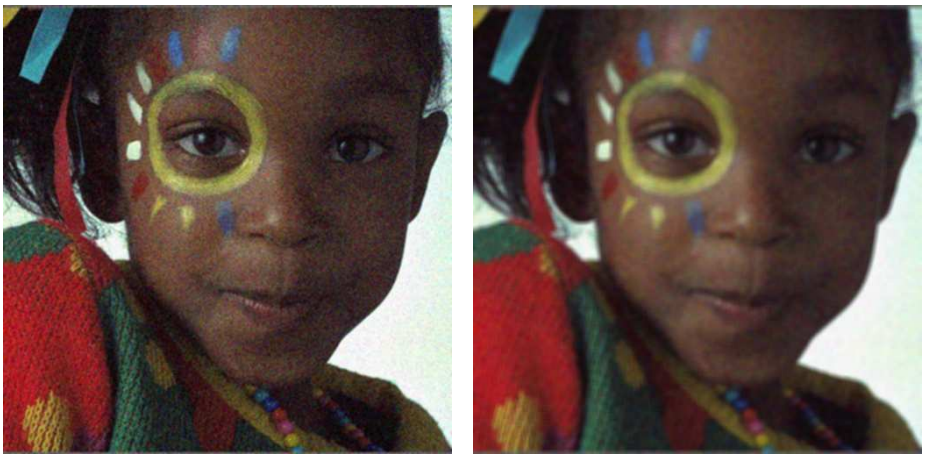


Figura 21: Ambas imágenes han sido reconstruidas con el método Gaussiano con sigma igual a 10. A la izquierda es el caso para una máscara 3x3, y a la derecha para una máscara 7x7.

Y en las Figuras 25,26,27,28,29 se realiza el experimento con una varianza igual a 0.1.

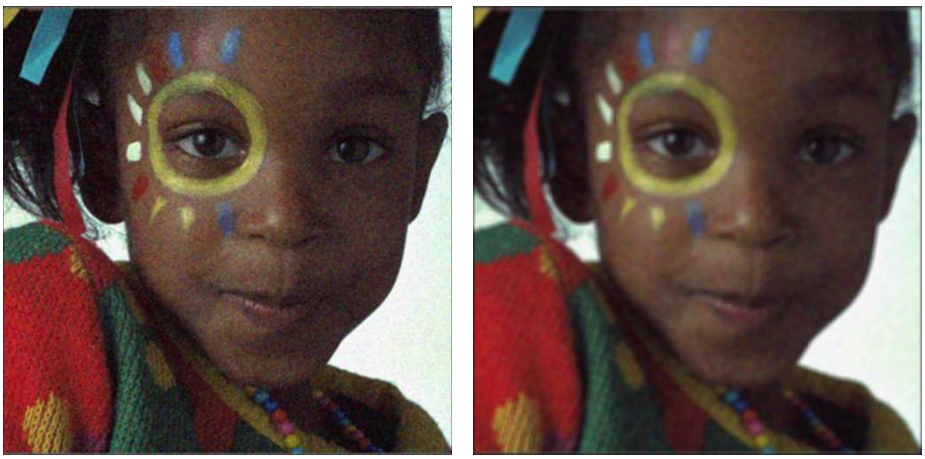


Figura 22: Ambas imágenes han sido reconstruidas con el filtro de vecindad. A la izquierda es el caso para una máscara 3x3, y a la derecha para una máscara 7x7.

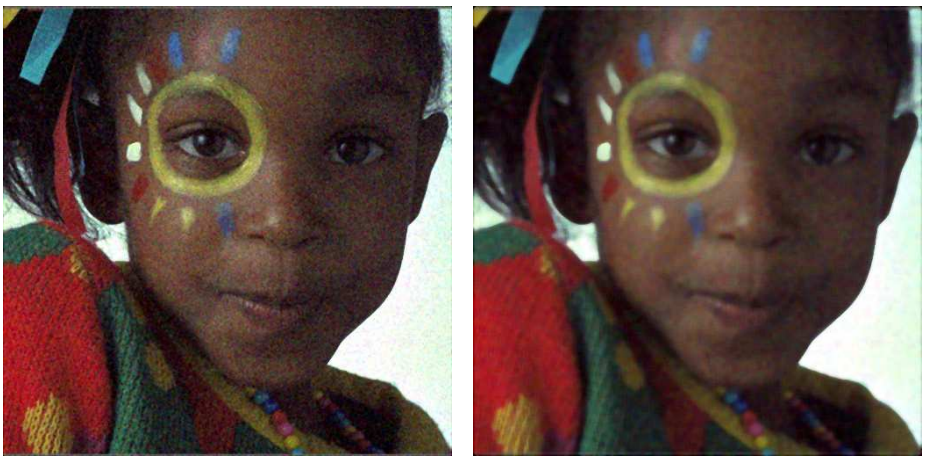


Figura 23: Ambas imágenes han sido reconstruidas con el filtro de mediana. A la izquierda es el caso para una máscara 3x3, y a la derecha para una máscara 7x7.

En general observamos que los resultados son bastante buenos, sobre todo con el filtro gaussiano y el de Wiener. Observamos que cuanto mayor es el tamaño de la máscara utilizada más ruido quitamos en la imagen aunque también se produce un mayor suavizado. En cualquier caso, en general pensamos que los algoritmos basados en multirresolución se presentan

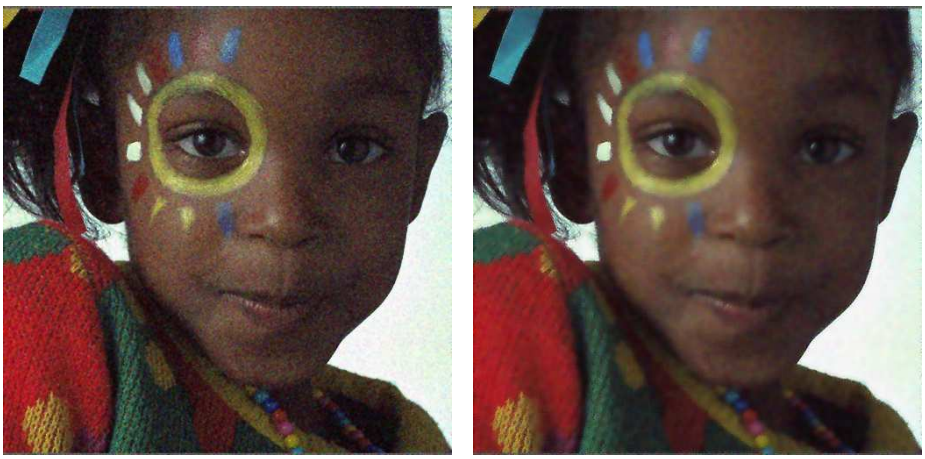


Figura 24: Ambas imágenes han sido reconstruidas con el filtro de wiener. A la izquierda es el caso para una máscara 3x3, y a la derecha para una máscara 7x7.

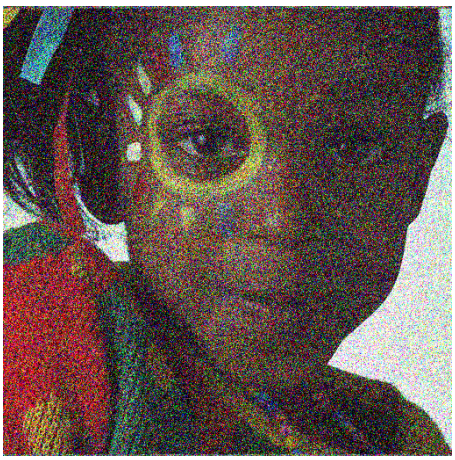


Figura 25: Imagen con ruido Gaussiano de varianza 0.1.

como una buena alternativa a estos métodos clásicos para la eliminación de ruido gaussiano en las imágenes digitales. Para otros tipos de ruido como el sal y pimienta, los métodos de multirresolución no dan tan buen resultado.

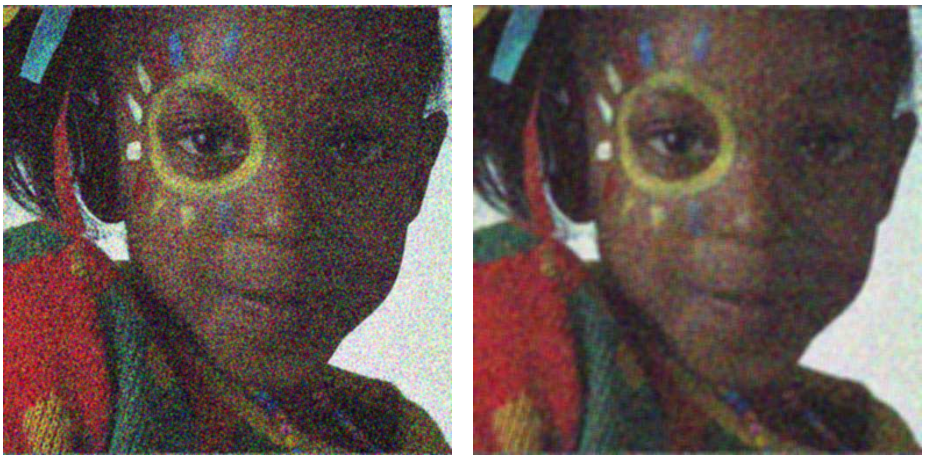


Figura 26: Ambas imágenes han sido reconstruidas con el método Gaussiano con sigma igual a 10. A la izquierda es el caso para una máscara 3x3, y a la derecha para una máscara 7x7.

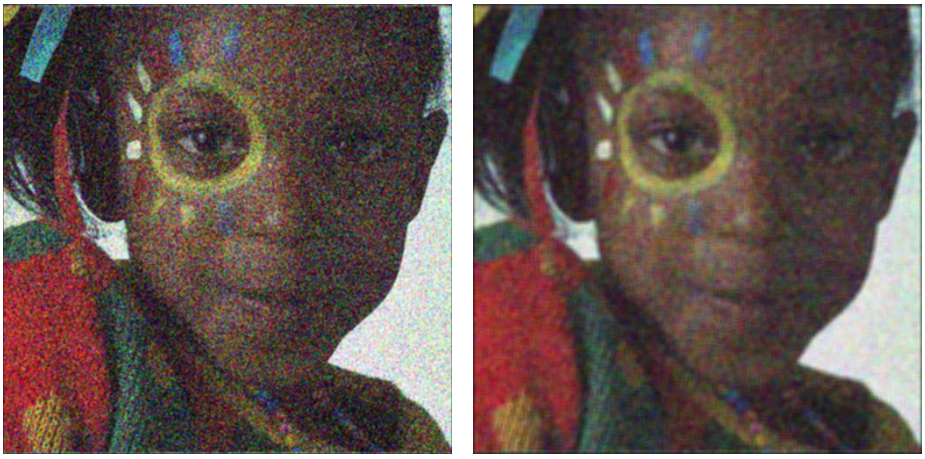


Figura 27: Ambas imágenes han sido reconstruidas con el filtro de vecindad. A la izquierda es el caso para una máscara 3x3, y a la derecha para una máscara 7x7.

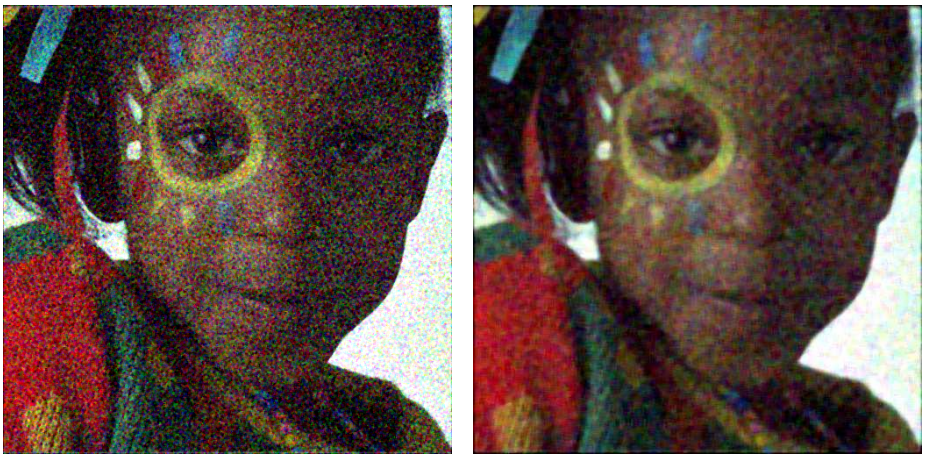


Figura 28: Ambas imágenes han sido reconstruidas con el filtro de mediana. A la izquierda es el caso para una máscara 3x3, y a la derecha para una máscara 7x7.

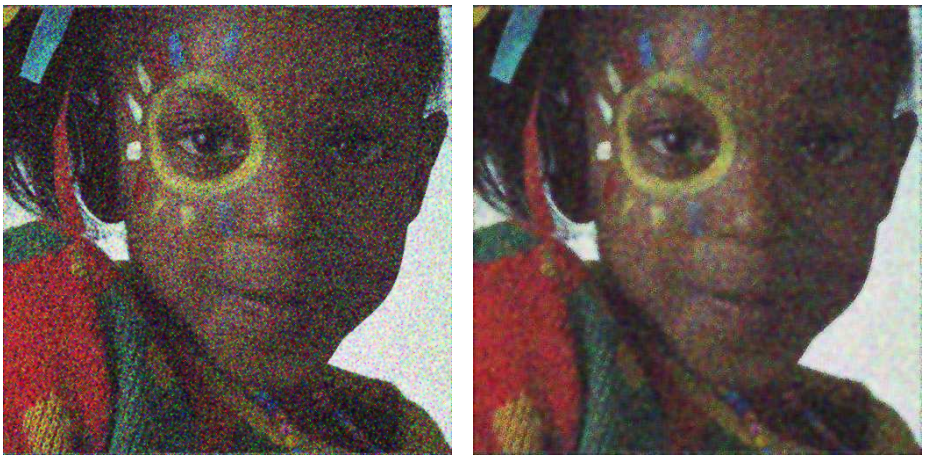


Figura 29: Ambas imágenes han sido reconstruidas con el filtro de wiener. A la izquierda es el caso para una máscara 3x3, y a la derecha para una máscara 7x7.

4. Desarrollo de los programas en Matlab

El proyecto ha sido realizado mediante la programación en Matlab del programa de multirresolución, y de cada una de las funciones de las que se compone.

4.1. Explicación de la interfaz gráfica y de la ejecución del programa de multirresolución

La creación de una interfaz gráfica es una buena manera de facilitar el uso del programa de multirresolución, tanto para usuarios expertos como para usuarios menos familiarizados con los conceptos matemáticos que se desarrollan en la multirresolución. La figura 30 muestra el entorno gráfico creado en este proyecto para facilitar el uso de los algoritmos.

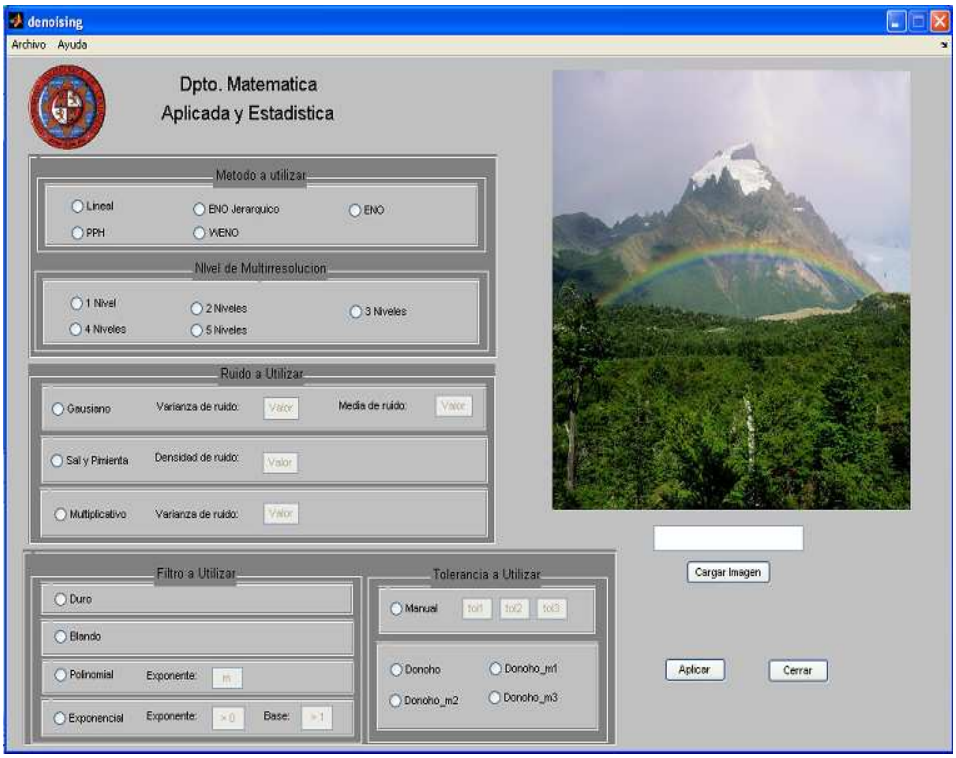


Figura 30: Entorno gráfico del programa Denoising.

El entorno gráfico está dividido en la imagen a la que se aplica la multi-

rresolución, dos menus desplegables en la parte superior, varios paneles con los distintos parámetros del método de multirresolución y los botones de 'cargar imagen' y 'aplicar'. En este entorno gráfico se carga una imagen por defecto, hay dos maneras para cambiar la imagen, bien con el boton de 'cargar imagen', bien con la opción 'cargar imagen' de la pestaña de 'archivo' del menú desplegable de la parte superior izquierda de la ventana. Este menu desplegable se compone de dos pestañas, una la de archivo donde están las opciones de 'cargar imagen' y 'salir', y la pestaña de ayuda donde están las opciones de 'acerca de' y 'ejecución', que facilitan la ejecución del programa con algunos consejos. Los distintos paneles de los que se compone la interfaz gráfica están debidamente separados y nombrados, cada uno contiene varios radio botones para elegir los parámetros a introducir, y al seleccionar ciertos radio botones se activan las casillas correspondientes para introducir los valores. En la opción 'ejecución' del menú desplegable de ayuda se ven algunos valores factibles para cada casilla. Una vez introducidos todos los parámetros correctamente y seleccionada la imagen que se desea, se debe clicar en el botón 'aplicar' para que corran los programas. Como resultado se muestran dos ventanas, una con la imagen con ruido y otra con la imagen reconstruida. Por último, para cerrar el entorno gráfico se puede clicar en el botón 'cerrar', o bien en el menu desplegable de Archivo, en la opción 'salir'.

A continuación vamos a realizar un ejemplo de ejecución de la interfaz con la imagen lena5.pgm, con el método Lineal y 4 niveles de multirresolución, con un ruido Gaussiano de media 0 y varianza 0,01. Utilizaremos el filtro Polinomial con exponente igual a 2 y como tolerancia escogeremos el método 2 de Donoho. El primer paso para introducir estos datos es cambiar la imagen, bien con el botón 'cargar imagen' bien con el menu desplegable 'archivo' y la opción 'cargar imagen'. Como segundo paso introducimos los datos mediante los radio botones, así elegimos el método Lineal, 4 niveles de multirresolución, el ruido Gaussiano, el filtro Polinomial y la tolerancia Donoho_m2. Al seleccionar el ruido Gaussiano se activan las casillas de media y varianza para insertarle los valores, y al seleccionar el filtro Polinomial se activa la casilla de Exponente para introducir el valor. Como último paso queda clicar en el botón 'aplicar' para que el programa se ejecute con los valores elegidos. Como resultado nos devolverá dos ventanas, una con la imagen con ruido y otra con la imagen recuperada mediante los valores seleccionados.

El programa también se puede ejecutar desde la ventana de comandos de Matlab, mediante la función 'tensorc'. Para familiarizarnos con esta función veremos varios ejemplos del comando necesario para la ejecución en la ventana de comandos de Matlab:

En el primer caso introduciremos los parámetros del ejemplo anterior:

tensorc(4,'gaussian',[0,0.01],'camera2.pgm','lin','polinomial', [2,0],'Donoho_m2',[0,0,0]);

En el siguiente caso trabajamos con la imagen 'casa_roja.tiff' y le introduciremos ruido sal y pimienta de densidad 0,02 y vamos a hacer 2 niveles de multirresolución. Se aplicará el método de reconstrucción pph con el filtro exponencial con exponente 1 y base 3, y el método de tolerancia de truncamiento Donoho:

tensorc(2,'salt & pepper', [0.02,0], 'casa_roja.tiff', 'pph', 'exponencial', [1,3],'Donoho',[0,0,0]);

Como último caso aplicamos a la imagen lena5.pgm el ruido multiplicativo con varianza 0,04 y un nivel de multirresolución 3, y la imagen la reconstruimos mediante el método eno, un filtro suave y el método de tolerancia de truncamiento usuario, con valores de epsilon 6, 5 y 6.

tensorc(3,'speckle', [0.02,0], 'lena5.pgm', 'eno', 'suave', [0,0],'usuario',[6,5,6]);

4.2. Explicación de la interfaz gráfica y de la ejecución del programa de suavizado

Al igual que para Denoising, para el programa Suavizado hemos desarrollado un entorno gráfico de manejo sencillo para cualquier tipo de usuario. En la figura 31 se puede ver la interfaz del programa realizado para facilitar el uso de algoritmos de eliminación de ruido basados en algoritmos de filtrado.

Figura 31: Entorno gráfico del programa Suavizado.

El entorno gráfico del programa de suavizado tiene un aspecto similar al entorno gráfico del programa de multirresolución anteriormente presentado. Dispone de tres paneles con los parámetros de ruido (gaussiano, sal y pimienta y multiplicativo), de filtro (gaussiano, de vecindad, de mediana y wiener) y el tamaño de la máscara a utilizar. Al seleccionar el tipo de ruido deseado se activaran las casillas para introducir el valor de los parámetros. En la parte superior tiene dos pestañas (archivo y ayuda) con sus respectivos menus desplegables que nos dan la posibilidad de cargar la imagen a utilizar, salir del

programa o explicarnos brevemente como funciona el programa. Además, en la parte inferior de la imagen esta el botón 'cargar imagen' con el que podemos modificar la imagen con la que queremos trabajar. Y por últimos disponemos de los botones 'aplicar' y 'salir' para ejecutar el programa y salir de él respectivamente.

5. Código fuente de los algoritmos

5.1. Código fuente del algoritmo de Multirresolución

tensor.m

```
function
[a,a1,mra]=tensorc(l,tipo\_ruido,param\_ruido,im,met,fil,param\_filtro,...
epsi,epsilon)


% [a,a1,mra]=tensorc(l,tipo_ruido,param_ruido,im,met,fil,param_filtro,...
% epsi,epsilon);
% a aproximación a la matriz original
% a1 matriz original con el ruido añadido
% mra versión de multirresolución de a1
% l son los niveles de multirresolución
% tipo_ruido ruido elegido por el usuario que debe aplicarse a la imagen
% 'gaussian', 'salt & pepper', 'speckle'
% param_ruido parámetros necesarios según el ruido escogido, media,varianza
% o densidad
% im imagen que utiliza
% met metodo que quieres utilizar 'lin', 'eno', 'enh', 'wen', 'pph'
% fil filtro elegido por el usuario 'duro', 'suave', 'polinomial',
% 'exponencial'
% param_filtro parámetros necesarios para aplicar el filtro polinomial
% o exponencial
% epsi método de elección de la tolerancia de truncamiento elegido por
% el usuario
% 'usuario', 'Donoho', 'Donoho_m1', 'Donoho_m2', 'Donoho_m3'
% epsilon vector de tolerancias de truncamiento introducido por
% el usuario
%
% Ejemplo:
% [a,a1,mra]=tensorc(4,'gaussian',[0,0.01],'camera2.pgm','lin','polinomial',...
% [2,0],
% 'Donoho_m2',[0,0,0]);
%
% definición de la matriz, la guardamos en a


a=imread(im);


% nos guardamos la matriz original en a_orig para despues calcular
% los errores cometidos
```

```
a\_orig=a;

% y en [n1,m1,ban] las dimensiones de la matriz

[n1,m1,ban]=size(a);

% añadimos ruido a la imagen original y llamamos a1 a la nueva imagen

if (strcmp(tipo\_ruido,'gaussian'))
    a1=imnoise(a\_orig, 'gaussian', param\_ruido(1), param\_ruido(2));
    ruido=255*sqrt(param\_ruido(2));
elseif (strcmp(tipo\_ruido,'salt \& pepper'))
    a1=imnoise(a\_orig, 'salt \& pepper',param\_ruido(1));
    ruido=255*sqrt(param\_ruido(1));
elseif (strcmp(tipo\_ruido,'speckle'))
    a1=imnoise(a\_orig, 'speckle',param\_ruido(1));
    ruido=255*sqrt(param\_ruido(1));
else
    display('Error al introducir el ruido')
end

% transformamos las matrices a valores double para trabajar con ellas

a=double(a); a1=double(a1); a\_orig=double(a\_orig);
mra=zeros(size(a));

for i=1:ban

% descendemos por la piramide de multirresolución y realizamos la truncación
adecuada para la eliminación de ruido

mra(:,:,i)=descenderc(a1(:,:,i),l,ruido,met,fil,param\_filtro,epsi,epsilon);

% ascendemos por la piramide de multirresolución

a(:,:,i)=ascenderc(mra(:,:,i),l,met);

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

% Diferencias entre aprox-ruido
```

```
MSE\_aprox\_ruido(i)=(norm(a(:,:,i)-a1(:,:,i),'fro')^2)/n1/m1;
PSNR\_aprox\_ruido(i)=20*log(255/(sqrt(MSE\_aprox\_ruido(i))));


% Diferencias entre aprox-original

MSE\_aprox\_original(i)=(norm(a(:,:,i)-a\_orig(:,:,i),'fro')^2)/n1/m1;
PSNR\_aprox\_original(i)=20*log(255/(sqrt(MSE\_aprox\_original(i))));


% Diferencias entre ruido-original

MSE\_ruido\_original(i)=(norm(a\_orig(:,:,i)-a1(:,:,i),'fro')^2)/n1/m1;
PSNR\_ruido\_original(i)=20*log(255/(sqrt(MSE\_ruido\_original(i))));


% Cociente entre MSE_ruido_original/MSE_aprox_original

cociente(i)=MSE\_ruido\_original(i)/MSE\_aprox\_original(i); end


% Entrada a ficheros

 fid = fopen('comparacion.dat','a'); if ban==3
   fprintf(fid,'%s %s %d %s %f %f \n %s %f %f %s %f %f %f \n %f %f %f %f \n...
   %f %f %f %f \n %f %f %f %f\n\n',im,met,l,tipo\_ruido,param\_ruido,fil,...
   param\_filtro, epsi,epsilon,PSNR\_aprox\_ruido(1),PSNR\_aprox\_original(1),...
   PSNR\_ruido\_original(1),cociente(1),PSNR\_aprox\_ruido(2),...
   PSNR\_aprox\_original(2),PSNR\_ruido\_original(2),cociente(2),...
   PSNR\_aprox\_ruido(3),PSNR\_aprox\_original(3),PSNR\_ruido\_original(3),...
   cociente(3));
elseif ban==1
  fprintf(fid,'%s %s %d %s %f %f \n %s %f %f %s %f %f %f \n...
   %f %f %f %f \n\n',im,met,l,tipo\_ruido,param\_ruido,fil,param\_filtro,...
   epsi,epsilon,PSNR\_aprox\_ruido(1),PSNR\_aprox\_original(1),...
   PSNR\_ruido\_original(1),cociente(1));
end fclose(fid);


% Salida de las imágenes por pantalla

figure('Tag','primera','Name',[blanks(6), 'IMAGEN CON RUIDO...
GAUSSIANO DE VARIANZA', blanks(2),num2str(ruido)],'Position',...
[0 280 560 420]) if(ban==1)
 imagesc(a1,[0 255])
 colormap gray
```

```
else
 a1=uint8(a1);
 imshow(a1);
end
title('imagen con ruido');

figure('Tag','segunda','Name',[blanks(6),'IMAGEN...
RECONSTRUIDA',blanks(2),'METODO',blanks(2),...
upper(met),blanks(2),num2str(l),blanks(2),...
'NIVELES'],'Position',[0 16 560 420]) if(ban==1)
 imagesc(a,[0 255])
 colormap gray
else
 a=uint8(a);
 imshow(a);
end
title('imagen reconstruida');
```

descenderc.m

```
function
[c]=descenderc(a,l,ruido,met,fil,param\_filtro,epsi,epsilon)


% [c]=descenderc(a,l,ruido,met,fil,param_filtro,epsi,epsilon)
% c version de multirresolución de la matriz
% a matriz a la que le aplicamos el algorimo descendente de multiresolucion
% l son los niveles de multiresolucion
% ruido es el nivel de ruido que se suma a la imagen
% met metodo que quieres utilizar 'lin', 'eno', 'enh', 'wen', 'pph'
% fil filtro elegido por el usuario 'duro', 'suave', 'polinomial',
% 'exponencial'
% param_filtro parámetros necesarios para aplicar el filtro polinomial
% o exponencial
% epsi método de elección de la tolerancia de truncamiento elegido
% por el usuario
% 'usuario', 'Donoho', 'Donoho_m1', 'Donoho_m2', 'Donoho_m3'
```

```
% epsilon vector de tolerancias de truncamiento introducido por el usuario


% Llamamos al método

if(met==’lin’)
     [c]=codiflic(a,l,ruido,fil,param\_filtro,epsi,epsilon);
elseif(met==’eno’)
     [c]=codifenoc(a,l,ruido,fil,param\_filtro,epsi,epsilon);
elseif(met==’enh’)
     [c]=codifenhc(a,l,ruido,fil,param\_filtro,epsi,epsilon);
elseif(met==’wen’)
     [c]=codifwenc(a,l,ruido,fil,param\_filtro,epsi,epsilon);
elseif (met==’pph’)
     [c]=codifpphc(a,l,ruido,fil,param\_filtro,epsi,epsilon);
end
```

codiflic.m

```
function [d]=codiflic(a,l,ruido,fil,param\_filtro,epsi,epsilon)


% [d]=codiflic(a,l,ruido,fil,param_filtro,epsi,epsilon);
% d matriz de multirresolución truncada
% a matriz a la que se le aplica la multirresolución
% l son los niveles de multirresolución
% ruido es el nivel de ruido introducido
% fil filtro elegido por el usuario ’duro’, ’suave’, ’polinomial’,
% ’exponencial’
% param_filtro parámetros necesarios para aplicar el filtro polinomial
% o exponencial
% epsi método de elección de la tolerancia de truncamiento elegido por
% el usuario
% ’usuario’, ’Donoho’, ’Donoho_m1’, ’Donoho_m2’, ’Donoho_m3’
% epsilon vector de tolerancias de truncamiento introducido por el usuario
```

```
[n,m]=size(a);
d=a;


% inicialización de las variables


% bucle para los niveles de multirresolución

for k=1:l


% se hace la multirresolución por filas

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


% se hace la multirresolución por columnas

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


% se ordena la matriz

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


% se pasan los diferentes filtros con la elección de tolerancia pedida

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

```
if strcmp(fil,'duro')
    fil\_ind=1;

elseif strcmp(fil,'suave')
    fil\_ind=2;

elseif strcmp(fil,'polinomial')
    fil\_ind=3;

elseif strcmp(fil,'exponencial')
    fil\_ind=4;

else
    display('Error al introducir el filtro')

end

if strcmp(epsi,'usuario')
    epsitmp=1;

elseif strcmp(epsi,'Donoho')
    epsitmp=2;

elseif strcmp(epsi,'Donoho\_m1')
    epsitmp=3;

elseif strcmp(epsi,'Donoho\_m2')
    epsitmp=4;

elseif strcmp(epsi,'Donoho\_m3')
    epsitmp=5;

else
    display('Error al introducir epsi')

end

switch(fil\_ind)

    case 1
% duro
      switch(epsitmp)

          case 1
```

```
%'usuario'

                epsilon=epsilon/2^(k-1);
                tol2=epsilon(1); tol3=epsilon(2); tol4=epsilon(3);
        case 2

%'Donoho'

                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)));
                tol4=ruido*sqrt(2*(log(b4m)+log(b4n)));
        case 3

%'Donoho_m1'

                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)));
                tol4=2*ruido*sqrt(2*(log(b4m)+log(b4n)));
        case 4

%'Donoho_m2'

                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)))/((k+1)*(k+1));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)))/((k+1)*(k+1));
                tol4=ruido*sqrt(2*(log(b4m)+log(b4n)))/((k+1)*(k+1));
         case 5

%'Donoho_m3'

                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)))/((k+1)*(k+1));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)))/((k+1)*(k+1));
                tol4=2*ruido*sqrt(2*(log(b4m)+log(b4n)))/((k+1)*(k+1));
      end


% HARD THRESHOLDING

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

case 2

   %'suave'
```

```
      switch(epsitmp)

          case 1

    %'usuario'

                epsilon=epsilon/2^(k-1);
                tol2=epsilon(1); tol3=epsilon(2); tol4=epsilon(3);
          case 2

    %'Donoho'

                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)));
                tol4=ruido*sqrt(2*(log(b4m)+log(b4n)));
          case 3

    %'Donoho_m1'

                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)));
                tol4=2*ruido*sqrt(2*(log(b4m)+log(b4n)));
          case 4

    %'Donoho_m2'

                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)))/((k+1)*(k+1));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)))/((k+1)*(k+1));
                tol4=ruido*sqrt(2*(log(b4m)+log(b4n)))/((k+1)*(k+1));
          case 5

    %'Donoho_m3'

                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)))/((k+1)*(k+1));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)))/((k+1)*(k+1));
                tol4=2*ruido*sqrt(2*(log(b4m)+log(b4n)))/((k+1)*(k+1));
      end


% SOFT THRESHOLDING

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

case 3
```

```
    %'polinomial'

       switch(epsitmp)

           case 1

    %'usuario'

                  epsilon=epsilon/2^(k-1);
                  tol2=epsilon(1); tol3=epsilon(2); tol4=epsilon(3);
          case 2

    %'Donoho'

                  tol2=ruido*sqrt(2*(log(b2m)+log(b2n)));
                  tol3=ruido*sqrt(2*(log(b3n)+log(b3m)));
                  tol4=ruido*sqrt(2*(log(b4m)+log(b4n)));
          case 3

    %'Donoho_m1'

                  tol2=ruido*sqrt(2*(log(b2m)+log(b2n)));
                  tol3=ruido*sqrt(2*(log(b3n)+log(b3m)));
                  tol4=2*ruido*sqrt(2*(log(b4m)+log(b4n)));
          case 4

    %'Donoho_m2'

                  tol2=ruido*sqrt(2*(log(b2m)+log(b2n)))/((k+1)*(k+1));
                  tol3=ruido*sqrt(2*(log(b3n)+log(b3m)))/((k+1)*(k+1));
                  tol4=ruido*sqrt(2*(log(b4m)+log(b4n)))/((k+1)*(k+1));
           case 5

    %'Donoho_m3'

                  tol2=ruido*sqrt(2*(log(b2m)+log(b2n)))/((k+1)*(k+1));
                  tol3=ruido*sqrt(2*(log(b3n)+log(b3m)))/((k+1)*(k+1));
                  tol4=2*ruido*sqrt(2*(log(b4m)+log(b4n)))/((k+1)*(k+1));
       end


% FILTRO POLINOMIAL
```

```
      b2=((1-tol2^param\_filtro(1)./(b2.^param\_filtro(1)+...
      10^(-10))).*(abs(b2)>tol2)).*b2;
      b3=((1-tol3^param_\filtro(1)./(b3.^param\_filtro(1)+...
      10^(-10))).*(abs(b3)>tol3)).*b3;
      b4=((1-tol4^param\_filtro(1)./(b4.^param\_filtro(1)+...
      10^(-10))).*(abs(b4)>tol4)).*b4;

case 4

   %'exponencial'

      switch(epsitmp)

          case 1

   %'usuario'

                epsilon=epsilon/2^(k-1);
                tol2=epsilon(1); tol3=epsilon(2); tol4=epsilon(3);
        case 2

   %'Donoho'

                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)));
                tol4=ruido*sqrt(2*(log(b4m)+log(b4n)));
        case 3

   %'Donoho_m1'

                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)));
                tol4=2*ruido*sqrt(2*(log(b4m)+log(b4n)));
        case 4

   %'Donoho_m2'

                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)))/((k+1)*(k+1));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)))/((k+1)*(k+1));
                tol4=ruido*sqrt(2*(log(b4m)+log(b4n)))/((k+1)*(k+1));
         case 5

   %'Donoho_m3'
```

```
                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)))/((k+1)*(k+1));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)))/((k+1)*(k+1));
                tol4=2*ruido*sqrt(2*(log(b4m)+log(b4n)))/((k+1)*(k+1));
      end


% FILTRO EXPONENCIAL

     b2=b2.*param\_filtro(1).^((-param\_filtro(2)*tol2^2)./(b2.^2+10^(-10)));
     b3=b3.*param\_filtro(1).^((-param\_filtro(2)*tol3^2)./(b3.^2+10^(-10)));
     b4=b4.*param\_filtro(1).^((-param\_filtro(2)*tol4^2)./(b4.^2+10^(-10)));

end

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


% se modifica el valor de n

n=n/2; m=m/2;


% se cierra el bucle de los niveles de multirresolución

end
return
```

codifenoc.m

```
function [d]=codifenoc(a,l,ruido,fil,param\_filtro,epsi,epsilon)


% [d]=codiflic(a,l,ruido,fil,param_filtro,epsi,epsilon);
% d matriz de multirresolución truncada
% a matriz a la que se le aplica la multirresolución
% l son los niveles de multirresolución
```

```
% ruido es el nivel de ruido introducido
% fil filtro elegido por el usuario 'duro', 'suave', 'polinomial',
% 'exponencial'
% param_filtro parámetros necesarios para aplicar el filtro polinomial
% o exponencial
% epsi método de elección de la tolerancia de truncamiento elegido por
% el usuario
% 'usuario', 'Donoho', 'Donoho_m1', 'Donoho_m2', 'Donoho_m3'
% epsilon vector de tolerancias de truncamiento introducido por el usuario


[n,m]=size(a);
d=a;


% inicializacion de las variables
% bucle para los niveles de multirresolución

for k=1:l


% se hace la multirresolución por filas

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


% se hace la multirresolución por columnas

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


% se ordena la matriz

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

```
% se pasan los diferentes filtros con la elección de tolerancia pedida


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

if strcmp(fil,'duro')
    fil\_ind=1;

elseif strcmp(fil,'suave')
    fil\_ind=2;

elseif strcmp(fil,'polinomial')
    fil\_ind=3;

elseif strcmp(fil,'exponencial')
    fil\_ind=4;

else
    display('Error al introducir el filtro')

end

if strcmp(epsi,'usuario')
    epsitmp=1;

elseif strcmp(epsi,'Donoho')
    epsitmp=2;

elseif strcmp(epsi,'Donoho\_m1')
    epsitmp=3;

elseif strcmp(epsi,'Donoho\_m2')
    epsitmp=4;

elseif strcmp(epsi,'Donoho\_m3')
    epsitmp=5;

else
    display('Error al introducir epsi')

end
```

```
switch(fil\_ind)

    case 1

   % 'duro'

      switch(epsitmp)

          case 1

   %'usuario'

                epsilon=epsilon/2^(k-1);
                tol2=epsilon(1); tol3=epsilon(2); tol4=epsilon(3);
         case 2

   %'Donoho'

                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)));
                tol4=ruido*sqrt(2*(log(b4m)+log(b4n)));
         case 3

   %'Donoho_m1'

                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)));
                tol4=2*ruido*sqrt(2*(log(b4m)+log(b4n)));
         case 4

   %'Donoho_m2'

                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)))/((k+1)*(k+1));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)))/((k+1)*(k+1));
                tol4=ruido*sqrt(2*(log(b4m)+log(b4n)))/((k+1)*(k+1));
          case 5

   %'Donoho_m3'

                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)))/((k+1)*(k+1));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)))/((k+1)*(k+1));
                tol4=2*ruido*sqrt(2*(log(b4m)+log(b4n)))/((k+1)*(k+1));
      end


% HARD THRESHOLDING
```

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

case 2

   %'suave'

      switch(epsitmp)

          case 1

   %'usuario'

                epsilon=epsilon/2^(k-1);
                tol2=epsilon(1); tol3=epsilon(2); tol4=epsilon(3);
          case 2

   %'Donoho'

                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)));
                tol4=ruido*sqrt(2*(log(b4m)+log(b4n)));
          case 3

   %'Donoho_m1'

                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)));
                tol4=2*ruido*sqrt(2*(log(b4m)+log(b4n)));
          case 4

   %'Donoho_m2'

                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)))/((k+1)*(k+1));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)))/((k+1)*(k+1));
                tol4=ruido*sqrt(2*(log(b4m)+log(b4n)))/((k+1)*(k+1));
          case 5

   %'Donoho_m3'

                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)))/((k+1)*(k+1));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)))/((k+1)*(k+1));
                tol4=2*ruido*sqrt(2*(log(b4m)+log(b4n)))/((k+1)*(k+1));
      end
```

```
% SOFT THRESHOLDING

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

case 3

   %'polinomial'

      switch(epsitmp)

          case 1

   %'usuario'

                epsilon=epsilon/2^(k-1);
                tol2=epsilon(1); tol3=epsilon(2); tol4=epsilon(3);
         case 2

   %'Donoho'

                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)));
                tol4=ruido*sqrt(2*(log(b4m)+log(b4n)));
         case 3

   %'Donoho_m1'

                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)));
                tol4=2*ruido*sqrt(2*(log(b4m)+log(b4n)));
         case 4

   %'Donoho_m2'

                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)))/((k+1)*(k+1));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)))/((k+1)*(k+1));
                tol4=ruido*sqrt(2*(log(b4m)+log(b4n)))/((k+1)*(k+1));
          case 5

   %'Donoho_m3'
```

```
                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)))/((k+1)*(k+1));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)))/((k+1)*(k+1));
                tol4=2*ruido*sqrt(2*(log(b4m)+log(b4n)))/((k+1)*(k+1));
        end


% FILTRO POLINOMIAL

        b2=((1-tol2^param\_filtro(1)./(b2.^param\_filtro(1)+...
        10^(-10))).*(abs(b2)>tol2)).*b2;
        b3=((1-tol3^param_\filtro(1)./(b3.^param\_filtro(1)+...
        10^(-10))).*(abs(b3)>tol3)).*b3;
        b4=((1-tol4^param\_filtro(1)./(b4.^param\_filtro(1)+...
        10^(-10))).*(abs(b4)>tol4)).*b4;

case 4

    %'exponencial'

        switch(epsitmp)

            case 1

    %'usuario'

                epsilon=epsilon/2^(k-1);
                tol2=epsilon(1); tol3=epsilon(2); tol4=epsilon(3);
        case 2

    %'Donoho'

                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)));
                tol4=ruido*sqrt(2*(log(b4m)+log(b4n)));
        case 3

    %'Donoho_m1'

                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)));
                tol4=2*ruido*sqrt(2*(log(b4m)+log(b4n)));
        case 4

    %'Donoho_m2'
```

```
                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)))/((k+1)*(k+1));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)))/((k+1)*(k+1));
                tol4=ruido*sqrt(2*(log(b4m)+log(b4n)))/((k+1)*(k+1));
          case 5

    %'Donoho_m3'

                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)))/((k+1)*(k+1));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)))/((k+1)*(k+1));
                tol4=2*ruido*sqrt(2*(log(b4m)+log(b4n)))/((k+1)*(k+1));
       end


% FILTRO EXPONENCIAL

     b2=b2.*param\_filtro(1).^((-param\_filtro(2)*tol2^2)./(b2.^2+10^(-10)));
     b3=b3.*param\_filtro(1).^((-param\_filtro(2)*tol3^2)./(b3.^2+10^(-10)));
     b4=b4.*param\_filtro(1).^((-param\_filtro(2)*tol4^2)./(b4.^2+10^(-10)));

end

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


% se modifica el valor de n

n=n/2; m=m/2;


% se cierra el bucle de los niveles de multirresolución

end
return
```

codifenhc.m

```
function [d]=codifenhc(a,l,ruido,fil,param\_filtro,epsi,epsilon)


% [d]=codifenhc(a,l,ruido,fil,param_filtro,epsi,epsilon);
% d matriz de multirresolución truncada
% a matriz a la que se le aplica la multirresolución
% l son los niveles de multirresolución
% ruido es el nivel de ruido introducido
% fil filtro elegido por el usuario 'duro', 'suave', 'polinomial',
% 'exponencial'
% param_filtro parámetros necesarios para aplicar el filtro polinomial
o
% exponencial
% epsi método de elección de la tolerancia de truncamiento elegido por
el usuario
% 'usuario', 'Donoho', 'Donoho_m1', 'Donoho_m2', 'Donoho_m3'
% epsilon vector de tolerancias de truncamiento introducido por el usuario

[n,m]=size(a);
d=a;


% inicializacion de las variables
% bucle para los niveles de multirresolución

for k=1:l


% se hace la multirresolución por filas

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


% se hace la multirresolución por columnas

for j=1:m
[f1,f2]=enhec(d(1:n,j)',n);
d(1:2:n,j)=f1';
d(2:2:n,j)=f2';
end
clear f1; clear f2;
```

```
% se ordena la matriz

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


% se pasan los diferentes filtros con la elección de tolerancia pedida

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

if strcmp(fil,'duro')
    fil\_ind=1;

elseif strcmp(fil,'suave')
    fil\_ind=2;

elseif strcmp(fil,'polinomial')
    fil\_ind=3;

elseif strcmp(fil,'exponencial')
    fil\_ind=4;

else
    display('Error al introducir el filtro')

end

if strcmp(epsi,'usuario')
    epsitmp=1;

elseif strcmp(epsi,'Donoho')
    epsitmp=2;

elseif strcmp(epsi,'Donoho\_m1')
    epsitmp=3;

elseif strcmp(epsi,'Donoho\_m2')
    epsitmp=4;

elseif strcmp(epsi,'Donoho\_m3')
```

```
    epsitmp=5;

else
    display('Error al introducir epsi')

end

switch(fil\_ind)

    case 1

    % 'duro'

      switch(epsitmp)

          case 1

    %'usuario'

                epsilon=epsilon/2^(k-1);
                tol2=epsilon(1); tol3=epsilon(2); tol4=epsilon(3);
          case 2

    %'Donoho'

                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)));
                tol4=ruido*sqrt(2*(log(b4m)+log(b4n)));
          case 3

    %'Donoho_m1'

                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)));
                tol4=2*ruido*sqrt(2*(log(b4m)+log(b4n)));
          case 4

    %'Donoho_m2'

                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)))/((k+1)*(k+1));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)))/((k+1)*(k+1));
                tol4=ruido*sqrt(2*(log(b4m)+log(b4n)))/((k+1)*(k+1));
           case 5

    %'Donoho_m3'
```

```
                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)))/((k+1)*(k+1));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)))/((k+1)*(k+1));
                tol4=2*ruido*sqrt(2*(log(b4m)+log(b4n)))/((k+1)*(k+1));
      end


% HARD THRESHOLDING

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

case 2

   %'suave'

      switch(epsitmp)

          case 1

   %'usuario'

                epsilon=epsilon/2^(k-1);
                tol2=epsilon(1); tol3=epsilon(2); tol4=epsilon(3);
         case 2

   %'Donoho'

                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)));
                tol4=ruido*sqrt(2*(log(b4m)+log(b4n)));
         case 3

   %'Donoho_m1'

                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)));
                tol4=2*ruido*sqrt(2*(log(b4m)+log(b4n)));
         case 4

   %'Donoho_m2'

                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)))/((k+1)*(k+1));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)))/((k+1)*(k+1));
                tol4=ruido*sqrt(2*(log(b4m)+log(b4n)))/((k+1)*(k+1));
          case 5
```

```
    %'Donoho_m3'

                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)))/((k+1)*(k+1));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)))/((k+1)*(k+1));
                tol4=2*ruido*sqrt(2*(log(b4m)+log(b4n)))/((k+1)*(k+1));
       end


% SOFT THRESHOLDING

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

case 3

    %'polinomial'

       switch(epsitmp)

           case 1

    %'usuario'

                epsilon=epsilon/2^(k-1);
                tol2=epsilon(1); tol3=epsilon(2); tol4=epsilon(3);
          case 2

    %'Donoho'

                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)));
                tol4=ruido*sqrt(2*(log(b4m)+log(b4n)));
          case 3

    %'Donoho_m1'

                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)));
                tol4=2*ruido*sqrt(2*(log(b4m)+log(b4n)));
          case 4

    %'Donoho_m2'
```

```
                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)))/((k+1)*(k+1));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)))/((k+1)*(k+1));
                tol4=ruido*sqrt(2*(log(b4m)+log(b4n)))/((k+1)*(k+1));
          case 5

    %'Donoho_m3'

                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)))/((k+1)*(k+1));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)))/((k+1)*(k+1));
                tol4=2*ruido*sqrt(2*(log(b4m)+log(b4n)))/((k+1)*(k+1));
       end


% FILTRO POLINOMIAL

       b2=((1-tol2^param\_filtro(1)./(b2.^param\_filtro(1)+...
       10^(-10))).*(abs(b2)>tol2)).*b2;
       b3=((1-tol3^param_\filtro(1)./(b3.^param\_filtro(1)+...
       10^(-10))).*(abs(b3)>tol3)).*b3;
       b4=((1-tol4^param\_filtro(1)./(b4.^param\_filtro(1)+...
       10^(-10))).*(abs(b4)>tol4)).*b4;

case 4

    %'exponencial'

       switch(epsitmp)

           case 1

    %'usuario'

                epsilon=epsilon/2^(k-1);
                tol2=epsilon(1); tol3=epsilon(2); tol4=epsilon(3);
          case 2

    %'Donoho'

                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)));
                tol4=ruido*sqrt(2*(log(b4m)+log(b4n)));
          case 3

    %'Donoho_m1'
```

```
                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)));
                tol4=2*ruido*sqrt(2*(log(b4m)+log(b4n)));
        case 4

    %'Donoho_m2'

                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)))/((k+1)*(k+1));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)))/((k+1)*(k+1));
                tol4=ruido*sqrt(2*(log(b4m)+log(b4n)))/((k+1)*(k+1));
         case 5

    %'Donoho_m3'

                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)))/((k+1)*(k+1));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)))/((k+1)*(k+1));
                tol4=2*ruido*sqrt(2*(log(b4m)+log(b4n)))/((k+1)*(k+1));
      end


% FILTRO EXPONENCIAL

     b2=b2.*param\_filtro(1).^((-param\_filtro(2)*tol2^2)./(b2.^2+10^(-10)));
     b3=b3.*param\_filtro(1).^((-param\_filtro(2)*tol3^2)./(b3.^2+10^(-10)));
     b4=b4.*param\_filtro(1).^((-param\_filtro(2)*tol4^2)./(b4.^2+10^(-10)));

end

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


% se modifica el valor de n

n=n/2; m=m/2;


% se cierra el bucle de los niveles de multirresolución

end
return
```

codifwenc.m

```
function [d]=codifwenc(a,l,ruido,fil,param\_filtro,epsi,epsilon)


% [d]=codifwenc(a,l,ruido,fil,param_filtro,epsi,epsilon);
% [d]=codiflic(a,l,ruido,fil,param_filtro,epsi,epsilon);
% d matriz de multirresolución truncada
% a matriz a la que se le aplica la multirresolución
% l son los niveles de multirresolución
% ruido es el nivel de ruido introducido
% fil filtro elegido por el usuario 'duro', 'suave', 'polinomial',
% 'exponencial'
% param_filtro parámetros necesarios para aplicar el filtro polinomial
% o exponencial
% epsi método de elección de la tolerancia de truncamiento elegido por
% el usuario
% 'usuario', 'Donoho', 'Donoho_m1', 'Donoho_m2', 'Donoho_m3'
% epsilon vector de tolerancias de truncamiento introducido por el usuario


[n,m]=size(a);
d=a;


% inicializacion de las variables
% bucle para los niveles de multirresolución

for k=1:l


% se hace la multirresolución por filas

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


% se hace la multirresolución por columnas
```

```
for j=1:m
[f1,f2]=wenoec(d(1:n,j)',n);
d(1:2:n,j)=f1';
d(2:2:n,j)=f2';
end
clear f1; clear f2;


% se ordena la matriz

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


% se pasan los diferentes filtros con la elección de tolerancia pedida

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

if strcmp(fil,'duro')
    fil\_ind=1;

elseif strcmp(fil,'suave')
    fil\_ind=2;

elseif strcmp(fil,'polinomial')
    fil\_ind=3;

elseif strcmp(fil,'exponencial')
    fil\_ind=4;

else
    display('Error al introducir el filtro')

end

if strcmp(epsi,'usuario')
    epsitmp=1;

elseif strcmp(epsi,'Donoho')
    epsitmp=2;
```

```
elseif strcmp(epsi,'Donoho\_m1')
    epsitmp=3;

elseif strcmp(epsi,'Donoho\_m2')
    epsitmp=4;

elseif strcmp(epsi,'Donoho\_m3')
    epsitmp=5;

else
    display('Error al introducir epsi')

end

switch(fil\_ind)

    case 1

   % 'duro'

      switch(epsitmp)

          case 1

   %'usuario'

                epsilon=epsilon/2^(k-1);
                tol2=epsilon(1); tol3=epsilon(2); tol4=epsilon(3);
         case 2

   %'Donoho'

                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)));
                tol4=ruido*sqrt(2*(log(b4m)+log(b4n)));
         case 3

   %'Donoho_m1'

                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)));
                tol4=2*ruido*sqrt(2*(log(b4m)+log(b4n)));
         case 4

   %'Donoho_m2'
```

```
                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)))/((k+1)*(k+1));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)))/((k+1)*(k+1));
                tol4=ruido*sqrt(2*(log(b4m)+log(b4n)))/((k+1)*(k+1));
          case 5

    %'Donoho_m3'

                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)))/((k+1)*(k+1));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)))/((k+1)*(k+1));
                tol4=2*ruido*sqrt(2*(log(b4m)+log(b4n)))/((k+1)*(k+1));
      end


% HARD THRESHOLDING

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

case 2

    %'suave'

      switch(epsitmp)

          case 1

    %'usuario'

                epsilon=epsilon/2^(k-1);
                tol2=epsilon(1); tol3=epsilon(2); tol4=epsilon(3);
          case 2

    %'Donoho'

                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)));
                tol4=ruido*sqrt(2*(log(b4m)+log(b4n)));
          case 3

    %'Donoho_m1'

                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)));
                tol4=2*ruido*sqrt(2*(log(b4m)+log(b4n)));
          case 4
```

```
    %'Donoho_m2'

                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)))/((k+1)*(k+1));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)))/((k+1)*(k+1));
                tol4=ruido*sqrt(2*(log(b4m)+log(b4n)))/((k+1)*(k+1));
          case 5

    %'Donoho_m3'

                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)))/((k+1)*(k+1));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)))/((k+1)*(k+1));
                tol4=2*ruido*sqrt(2*(log(b4m)+log(b4n)))/((k+1)*(k+1));
       end


% SOFT THRESHOLDING

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

case 3

    %'polinomial'

       switch(epsitmp)

           case 1

    %'usuario'

                epsilon=epsilon/2^(k-1);
                tol2=epsilon(1); tol3=epsilon(2); tol4=epsilon(3);
          case 2

    %'Donoho'

                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)));
                tol4=ruido*sqrt(2*(log(b4m)+log(b4n)));
          case 3

    %'Donoho_m1'
```

```
                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)));
                tol4=2*ruido*sqrt(2*(log(b4m)+log(b4n)));
        case 4

    %'Donoho_m2'

                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)))/((k+1)*(k+1));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)))/((k+1)*(k+1));
                tol4=ruido*sqrt(2*(log(b4m)+log(b4n)))/((k+1)*(k+1));
         case 5

    %'Donoho_m3'

                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)))/((k+1)*(k+1));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)))/((k+1)*(k+1));
                tol4=2*ruido*sqrt(2*(log(b4m)+log(b4n)))/((k+1)*(k+1));
      end


% FILTRO POLINOMIAL

      b2=((1-tol2^param\_filtro(1)./(b2.^param\_filtro(1)+...
      10^(-10))).*(abs(b2)>tol2)).*b2;
      b3=((1-tol3^param_\filtro(1)./(b3.^param\_filtro(1)+...
      10^(-10))).*(abs(b3)>tol3)).*b3;
      b4=((1-tol4^param\_filtro(1)./(b4.^param\_filtro(1)+...
      10^(-10))).*(abs(b4)>tol4)).*b4;

case 4

    %'exponencial'

      switch(epsitmp)

          case 1

    %'usuario'

                epsilon=epsilon/2^(k-1);
                tol2=epsilon(1); tol3=epsilon(2); tol4=epsilon(3);
        case 2

    %'Donoho'
```

```
                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)));
                tol4=ruido*sqrt(2*(log(b4m)+log(b4n)));
        case 3

    %'Donoho_m1'

                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)));
                tol4=2*ruido*sqrt(2*(log(b4m)+log(b4n)));
        case 4

    %'Donoho_m2'

                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)))/((k+1)*(k+1));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)))/((k+1)*(k+1));
                tol4=ruido*sqrt(2*(log(b4m)+log(b4n)))/((k+1)*(k+1));
         case 5

    %'Donoho_m3'

                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)))/((k+1)*(k+1));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)))/((k+1)*(k+1));
                tol4=2*ruido*sqrt(2*(log(b4m)+log(b4n)))/((k+1)*(k+1));
      end


% FILTRO EXPONENCIAL

     b2=b2.*param\_filtro(1).^((-param\_filtro(2)*tol2^2)./(b2.^2+10^(-10)));
     b3=b3.*param\_filtro(1).^((-param\_filtro(2)*tol3^2)./(b3.^2+10^(-10)));
     b4=b4.*param\_filtro(1).^((-param\_filtro(2)*tol4^2)./(b4.^2+10^(-10)));

end

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


% se modifica el valor de n

n=n/2; m=m/2;


% se cierra el bucle de los niveles de multirresolución
```

```
end
return
```

codifpphc.m

```
function [d]=codifpphc(a,l,ruido,fil,param\_filtro,epsi,epsilon)


% [d]=codifpphc(a,l,ruido,fil,param_filtro,epsi,epsilon);
% [d]=codiflic(a,l,ruido,fil,param_filtro,epsi,epsilon);
% d matriz de multirresolución truncada
% a matriz a la que se le aplica la multirresolución
% l son los niveles de multirresolución
% ruido es el nivel de ruido introducido
% fil filtro elegido por el usuario 'duro', 'suave', 'polinomial',
% 'exponencial'
% param_filtro parámetros necesarios para aplicar el filtro polinomial
% o exponencial
% epsi método de elección de la tolerancia de truncamiento elegido por
% el usuario
% 'usuario', 'Donoho', 'Donoho_m1', 'Donoho_m2', 'Donoho_m3'
% epsilon vector de tolerancias de truncamiento introducido por el usuario


[n,m]=size(a);
d=a;


% inicializacion de las variables
% bucle para los niveles de multirresolución

for k=1:l


% se hace la multirresolución por filas
```

```
for i=1:n
[f1,f2]=pphec(d(i,1:m),m);
d(i,1:2:m)=f1;
d(i,2:2:m)=f2;
end
clear f1; clear f2;


% se hace la multirresolución por columnas

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


% se ordena la matriz

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


% se pasan los diferentes filtros con la elección de tolerancia pedida

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

if strcmp(fil,'duro')
    fil\_ind=1;

elseif strcmp(fil,'suave')
    fil\_ind=2;

elseif strcmp(fil,'polinomial')
    fil\_ind=3;

elseif strcmp(fil,'exponencial')
    fil\_ind=4;

else
```

```
    display('Error al introducir el filtro')

end

if strcmp(epsi,'usuario')
    epsitmp=1;

elseif strcmp(epsi,'Donoho')
    epsitmp=2;

elseif strcmp(epsi,'Donoho\_m1')
    epsitmp=3;

elseif strcmp(epsi,'Donoho\_m2')
    epsitmp=4;

elseif strcmp(epsi,'Donoho\_m3')
    epsitmp=5;

else
    display('Error al introducir epsi')

end

switch(fil\_ind)

    case 1

    % 'duro'

      switch(epsitmp)

          case 1

    %'usuario'

                epsilon=epsilon/2^(k-1);
                tol2=epsilon(1); tol3=epsilon(2); tol4=epsilon(3);
          case 2

    %'Donoho'

                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)));
                tol4=ruido*sqrt(2*(log(b4m)+log(b4n)));
          case 3
```

```
    %'Donoho_m1'

                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)));
                tol4=2*ruido*sqrt(2*(log(b4m)+log(b4n)));
          case 4

    %'Donoho_m2'

                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)))/((k+1)*(k+1));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)))/((k+1)*(k+1));
                tol4=ruido*sqrt(2*(log(b4m)+log(b4n)))/((k+1)*(k+1));
           case 5

    %'Donoho_m3'

                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)))/((k+1)*(k+1));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)))/((k+1)*(k+1));
                tol4=2*ruido*sqrt(2*(log(b4m)+log(b4n)))/((k+1)*(k+1));
      end


% HARD THRESHOLDING

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

case 2

    %'suave'

      switch(epsitmp)

          case 1

    %'usuario'

                epsilon=epsilon/2^(k-1);
                tol2=epsilon(1); tol3=epsilon(2); tol4=epsilon(3);
          case 2

    %'Donoho'
```

```
                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)));
                tol4=ruido*sqrt(2*(log(b4m)+log(b4n)));
        case 3

    %'Donoho_m1'

                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)));
                tol4=2*ruido*sqrt(2*(log(b4m)+log(b4n)));
        case 4

    %'Donoho_m2'

                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)))/((k+1)*(k+1));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)))/((k+1)*(k+1));
                tol4=ruido*sqrt(2*(log(b4m)+log(b4n)))/((k+1)*(k+1));
         case 5

    %'Donoho_m3'

                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)))/((k+1)*(k+1));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)))/((k+1)*(k+1));
                tol4=2*ruido*sqrt(2*(log(b4m)+log(b4n)))/((k+1)*(k+1));
      end


% SOFT THRESHOLDING

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

case 3

    %'polinomial'

      switch(epsitmp)

          case 1

    %'usuario'

                epsilon=epsilon/2^(k-1);
                tol2=epsilon(1); tol3=epsilon(2); tol4=epsilon(3);
        case 2
```

```
    %'Donoho'

                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)));
                tol4=ruido*sqrt(2*(log(b4m)+log(b4n)));
          case 3

    %'Donoho_m1'

                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)));
                tol4=2*ruido*sqrt(2*(log(b4m)+log(b4n)));
          case 4

    %'Donoho_m2'

                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)))/((k+1)*(k+1));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)))/((k+1)*(k+1));
                tol4=ruido*sqrt(2*(log(b4m)+log(b4n)))/((k+1)*(k+1));
           case 5

    %'Donoho_m3'

                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)))/((k+1)*(k+1));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)))/((k+1)*(k+1));
                tol4=2*ruido*sqrt(2*(log(b4m)+log(b4n)))/((k+1)*(k+1));
       end


% FILTRO POLINOMIAL

       b2=((1-tol2^param\_filtro(1)./(b2.^param\_filtro(1)+...
       10^(-10))).*(abs(b2)>tol2)).*b2;
       b3=((1-tol3^param_\filtro(1)./(b3.^param\_filtro(1)+...
       10^(-10))).*(abs(b3)>tol3)).*b3;
       b4=((1-tol4^param\_filtro(1)./(b4.^param\_filtro(1)+...
       10^(-10))).*(abs(b4)>tol4)).*b4;

case 4

    %'exponencial'

       switch(epsitmp)

           case 1
```

```
    %'usuario'

                epsilon=epsilon/2^(k-1);
                tol2=epsilon(1); tol3=epsilon(2); tol4=epsilon(3);
          case 2

    %'Donoho'

                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)));
                tol4=ruido*sqrt(2*(log(b4m)+log(b4n)));
          case 3

    %'Donoho_m1'

                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)));
                tol4=2*ruido*sqrt(2*(log(b4m)+log(b4n)));
          case 4

    %'Donoho_m2'

                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)))/((k+1)*(k+1));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)))/((k+1)*(k+1));
                tol4=ruido*sqrt(2*(log(b4m)+log(b4n)))/((k+1)*(k+1));
           case 5

    %'Donoho_m3'

                tol2=ruido*sqrt(2*(log(b2m)+log(b2n)))/((k+1)*(k+1));
                tol3=ruido*sqrt(2*(log(b3n)+log(b3m)))/((k+1)*(k+1));
                tol4=2*ruido*sqrt(2*(log(b4m)+log(b4n)))/((k+1)*(k+1));
       end


% FILTRO EXPONENCIAL

      b2=b2.*param\_filtro(1).^((-param\_filtro(2)*tol2^2)./(b2.^2+10^(-10)));
      b3=b3.*param\_filtro(1).^((-param\_filtro(2)*tol3^2)./(b3.^2+10^(-10)));
      b4=b4.*param\_filtro(1).^((-param\_filtro(2)*tol4^2)./(b4.^2+10^(-10)));

end

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

```
% se modifica el valor de n

n=n/2; m=m/2;


% se cierra el bucle de los niveles de multirresolución

end
return
```

linealec.m

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


% valores significativos de la escala inferior

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


% calculo del primer detalle

q(1)=(11/8)*f1(1)-(4/8)*f1(2)+(1/8)*f1(3);
f2(1)=f(1)-q(1);


% calculo de los detalles intermedios

q(2:nk1-1)=(1/8)*f1(1:nk1-2)+f1(2:nk1-1)-(1/8)*f1(3:nk1);
f2(2:nk1-1)=f(3:2:n-2)-q(2:nk1-1);


% calculo de el ultimo detalle

q(nk1)=-(1/8)*f1(nk1-2)+(4/8)*f1(nk1-1)+(5/8)*f1(nk1);
f2(nk1)=f(n-1)-q(nk1);
```

enoec.m

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


% valores significativos de la escala inferior

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


% calculo del primer detalle

q(1)=(11/8)*f1(1)-(4/8)*f1(2)+(1/8)*f1(3);
f2(1)=f(1)-q(1);


% calculo del segundo detalle

ddc=abs(f1(1)-2*f1(2)+f1(3));
ddr=abs(f1(2)-2*f1(3)+f1(4));

if(ddc <= ddr)
   q(2)=(1/8)*f1(1)+f1(2)-(1/8)*f1(3);
else
   q(2)= (11/8)*f1(2)-(4/8)*f1(3)+(1/8)*f1(4);
end

f2(2)=f(3)-q(2);


% calculo de los detalles intermedios

for j=3:nk1-2
    ddl=abs(f1(j-2)-2*f1(j-1)+f1(j));
    ddc=abs(f1(j-1)-2*f1(j)+f1(j+1));
    ddr=abs(f1(j)-2*f1(j+1)+f1(j+2));


% prediccion y detalle

    if (ddc <= ddl \& ddc <= ddr)
        q(j)=(1/8)*f1(j-1)+f1(j)-(1/8)*f1(j+1);
    elseif(ddl<=ddr)
        q(j)=-(1/8)*f1(j-2)+(4/8)*f1(j-1)+(5/8)*f1(j);
    else
```

```
        q(j)=(11/8)*f1(j)-(4/8)*f1(j+1)+(1/8)*f1(j+2);
    end
    f2(j)=f(2*j-1)-q(j);
end


% calculo del penultimo detalle

ddl=abs(f1(nk1-3)-2*f1(nk1-2)+f1(nk1-1));
ddc=abs(f1(nk1-2)-2*f1(nk1-1)+f1(nk1));

if(ddc<=ddl)
  q(nk1-1)=(1/8)*f1(nk1-2)+f1(nk1-1)-(1/8)*f1(nk1);
else
  q(nk1-1)=-(1/8)*f1(nk1-3)+(4/8)*f1(nk1-2)+(5/8)*f1(nk1-1);
end
  f2(nk1-1)=f(n-3)-q(nk1-1);


% calculo de el ultimo detalle

q(nk1)=-(1/8)*f1(nk1-2)+(4/8)*f1(nk1-1)+(5/8)*f1(nk1);
f2(nk1)=f(n-1)-q(nk1);
```

enhec.m

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


% valores significativos de la escala inferior

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


% calculo del primer detalle
```

```
q(1)=(11/8)*f1(1)-(4/8)*f1(2)+(1/8)*f1(3);
f2(1)=f(1)-q(1);


% calculo del segundo detalle

dd1a=abs(f1(1)-f1(2));
dd1b=abs(f1(2)-f1(3));

if(dd1a <= dd1b)
    q(2)=(1/8)*f1(1)+f1(2)-(1/8)*f1(3);
else
    dd2a=abs(f1(1)-2*f1(2)+f1(3));
    dd2b=abs(f1(2)-2*f1(3)+f1(4));
    if(dd2a <= dd2b)
       q(2)=(1/8)*f1(1)+f1(2)-(1/8)*f1(3);
    else
       q(2)= (11/8)*f1(2)-(4/8)*f1(3)+(1/8)*f1(4);
    end
end

f2(2)=f(3)-q(2);


% calculo de los detalles intermedios

for j=3:nk1-2

    dd1a=abs(f1(j-1)-f1(j));
    dd1b=abs(f1(j)-f1(j+1));

   if(dd1a <= dd1b)
      dd2a=abs(f1(j-2)-2*f1(j-1)+f1(j));
      dd2b=abs(f1(j-1)-2*f1(j)+f1(j+1));
      if(dd2a <= dd2b)
          q(j)=-(1/8)*f1(j-2)+(4/8)*f1(j-1)+(5/8)*f1(j);
      else
          q(j)=(1/8)*f1(j-1)+f1(j)-(1/8)*f1(j+1);
      end
   else
     dd2a=abs(f1(j-1)-2*f1(j)+f1(j+1));
     dd2b=abs(f1(j)-2*f1(j+1)+f1(j+2));
     if(dd2a <= dd2b)
       q(j)=(1/8)*f1(j-1)+f1(j)-(1/8)*f1(j+1);
    else
```

```
       q(j)=(11/8)*f1(j)-(4/8)*f1(j+1)+(1/8)*f1(j+2);
    end
   end
    f2(j)=f(2*j-1)-q(j);
end


% calculo del penultimo detalle

dd1a=abs(f1(nk1-2)-f1(nk1-1));
dd1b=abs(f1(nk1-1)-f1(nk1));

if(dd1a <= dd1b)
    dd2a=abs(f1(nk1-3)-2*f1(nk1-2)+f1(nk1-1));
    dd2b=abs(f1(nk1-2)-2*f1(nk1-1)+f1(nk1));
    if(dd2a <= dd2b)
        q(nk1-1)=-(1/8)*f1(nk1-3)+(4/8)*f1(nk1-2)+(5/8)*f1(nk1-1);
    else
        q(nk1-1)=(1/8)*f1(nk1-2)+f1(nk1-1)-(1/8)*f1(nk1);
    end
else
        q(nk1-1)=(1/8)*f1(nk1-2)+f1(nk1-1)-(1/8)*f1(nk1);
end

  f2(nk1-1)=f(n-3)-q(nk1-1);


% calculo de el ultimo detalle

q(nk1)=-(1/8)*f1(nk1-2)+(4/8)*f1(nk1-1)+(5/8)*f1(nk1);
f2(nk1)=f(n-1)-q(nk1);
```

pphec.m

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


% valores significativos de la escala inferior

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


% calculo del primer detalle

q(1)=(11/8)*f1(1)-(4/8)*f1(2)+(1/8)*f1(3);
f2(1)=f(1)-q(1);


% calculo de los detalles intermedios


% calculo de las diferencias primeras


    delta1=diff(f1(1:nk1-1));
    delta2=diff(f1(2:nk1));


% parte de la prediccion que corresponde al intervalo central

        d=delta1.*delta2;
        d1=delta1+delta2;
        s=f1(2:nk1-1);
        d=(d>0).*d;
        d1=(d>0).*d1+(d<=0).*ones(size(d1));
        q(2:nk1-1)=s-0.5*d./d1;


% detalle

    f2(2:nk1-1)=f(3:2:n-3)-q(2:nk1-1);


% calculo de el ultimo detalle

q(nk1)=-(1/8)*f1(nk1-2)+(4/8)*f1(nk1-1)+(5/8)*f1(nk1);
f2(nk1)=f(n-1)-q(nk1);
```

wenoec.m

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


% valores significativos de la escala inferior

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


% calculo del primer detalle

q(1)=(11/8)*f1(1)-(4/8)*f1(2)+(1/8)*f1(3);
f2(1)=f(1)-q(1);


% calculo del segundo detalle

q(2)=(1/8)*f1(1)+f1(2)-(1/8)*f1(3);
f2(2)=f(3)-q(2);


% calculo de los detalles intermedios

epsilon=10^(-6);
for j=3:nk1-2


% los pesos optimos en este caso de valores en celda son c0=3/16,
% c1=10/16, c2=3/16

    c0=3/16; c1=10/16; c2=3/16;


% calculo de los indicadores de suavidad

    I0=0.5*((f1(j-2)-f1(j-1))^2+(f1(j)-f1(j-1))^2)+(f1(j)-2*f1(j-1)+f1(j-2))^2;
    I1=0.5*((f1(j)-f1(j-1))^2+(f1(j)-f1(j+1))^2)+(f1(j-1)-2*f1(j)+f1(j+1))^2;
    I2=0.5*((f1(j)-f1(j+1))^2+(f1(j+1)-f1(j+2))^2)+(f1(j)-2*f1(j+1)+f1(j+2))^2;


% calculo de alfa0, alfa1, alfa2

    alfa0=c0/((epsilon+I0)^3);
    alfa1=c1/((epsilon+I1)^3);
    alfa2=c2/((epsilon+I2)^3);
```

```
% calculo de los pesos wj's

    s=alfa0+alfa1+alfa2;
    w0=alfa0/s;  w1=alfa1/s; w2=alfa2/s;


% prediccion y detalle

    q(j)=w0*(-(1/8)*f1(j-2)+(4/8)*f1(j-1)+(5/8)*f1(j))+ w1*((1/8)*f1(j-1)+...
    f1(j)-(1/8)*f1(j+1))+w2*((11/8)*f1(j)-(4/8)*f1(j+1)+(1/8)*f1(j+2));
    f2(j)=f(2*j-1)-q(j);
end


% calculo del penultimo detalle

q(nk1-1)=(1/8)*f1(nk1-2)+f1(nk1-1)-(1/8)*f1(nk1);
f2(nk1-1)=f(n-3)-q(nk1-1);


% calculo de el último detalle

q(nk1)=-(1/8)*f1(nk1-2)+(4/8)*f1(nk1-1)+(5/8)*f1(nk1);
f2(nk1)=f(n-1)-q(nk1);
```

ascender.m

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


% c=ascenderc(a,l,met);
% c aproximación a la matriz original sin ruido
% a matriz a la que le aplicamos el algorimo ascendente de multiresolucion
% l son los niveles de multiresolucion
% met metodo que quieres utilizar 'lin', 'eno', 'enh', 'wen', 'pph'



% llamamos al metodo
```

```
if(met==’lin’)
     c=decodiflic(a,l);
elseif(met==’eno’)
     c=decodifenoc(a,l);
elseif(met==’enh’)
     c=decodifenhc(a,l);
elseif(met==’wen’)
     c=decodifwenc(a,l);
elseif(met==’pph’)
     c=decodifpphc(a,l);
end
```

decodiflic.m

```
function c=decodiflic(a,l)


% c=decodiflic(a,l)
% c aproximación a la matriz original sin ruido
% a matriz a la que se le aplica la multirresolución
% l son los niveles de multirresolución

[n,m]=size(a);
c=a;
nl=round(n/(2^l)); ml=round(m/(2^l));


% inicialización de las variables

for k=l:-1:1


% se trata del bucle para los niveles de multirresolución
% reordenamos la matriz haciendo el proceso inverso que en la codificacion

nb=2*nl; mb=2*ml;
b=zeros(nb,mb);
b(1:2:nb,1:2:mb)=c(1:nl,1:ml);
```

```
b(2:2:nb,2:2:mb)=c(nl+1:nb,ml+1:mb);
b(1:2:nb,2:2:mb)=c(1:nl,ml+1:mb);
b(2:2:nb,1:2:mb)=c(1+nl:nb,1:ml);
c(1:nb,1:mb)=b;
clear b;


% se hace el algoritmo de multirresolución inverso para las columnas

for j=1:mb
c(1:nb,j)=linealdec(c(1:nb,j)',nb)';
end


% se hace el algoritmo de multirresolución inverso para las filas

for i=1:nb
c(i,1:mb)=linealdec(c(i,1:mb),mb);
end


% se calcula el nl para subir al siguiente nivel

nl=2*nl; ml=2*ml;
end
```

decodifenoc.m

```
function c=decodifenoc(a,l)


% c=decodifenoc(a,l)
% c aproximación a la matriz original sin ruido
% a matriz a la que se le aplica la multirresolución
% l son los niveles de multirresolución

[n,m]=size(a);
c=a;
nl=round(n/(2^l)); ml=round(m/(2^l));
```

```
% inicialización de las variables

for k=l:-1:1


% se trata del bucle para los niveles de multirresolución
% reordenamos la matriz haciendo el proceso inverso que en la codificacion

nb=2*nl; mb=2*ml;
b=zeros(nb,mb);
b(1:2:nb,1:2:mb)=c(1:nl,1:ml);
b(2:2:nb,2:2:mb)=c(nl+1:nb,ml+1:mb);
b(1:2:nb,2:2:mb)=c(1:nl,ml+1:mb);
b(2:2:nb,1:2:mb)=c(1+nl:nb,1:ml);
c(1:nb,1:mb)=b;
clear b;


% se hace el algoritmo de multirresolución inverso para las columnas

for j=1:mb
c(1:nb,j)=enodec(c(1:nb,j)',nb)';
end


% se hace el algoritmo de multirresolución inverso para las filas

for i=1:nb
c(i,1:mb)=enodec(c(i,1:mb),mb);
end


% se calcula el nl para subir al siguiente nivel

nl=2*nl; ml=2*ml; end
```

decodifenhc.m

```
function c=decodifenhc(a,l)


% c=decodifenhc(a,l)
% c aproximación a la matriz original sin ruido
% a matriz a la que se le aplica la multirresolución
% l son los niveles de multirresolución

[n,m]=size(a);
c=a;
nl=round(n/(2^l)); ml=round(m/(2^l));


% inicialización de las variables

for k=l:-1:1


% se trata del bucle para los niveles de multirresolución
% reordenamos la matriz haciendo el proceso inverso que en la codificacion

nb=2*nl; mb=2*ml;
b=zeros(nb,mb);
b(1:2:nb,1:2:mb)=c(1:nl,1:ml);
b(2:2:nb,2:2:mb)=c(nl+1:nb,ml+1:mb);
b(1:2:nb,2:2:mb)=c(1:nl,ml+1:mb);
b(2:2:nb,1:2:mb)=c(1+nl:nb,1:ml);
c(1:nb,1:mb)=b;
clear b;


% se hace el algoritmo de multirresolución inverso para las columnas

for j=1:mb
c(1:nb,j)=enhdec(c(1:nb,j)',nb)';
end


% se hace el algoritmo de multirresolución inverso para las filas

for i=1:mb
c(i,1:mb)=enhdec(c(i,1:mb),mb);
end
```

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

nl=2*nl; ml=2*ml;
end
```

decodifwenc.m

```
function c=decodifwenc(a,l)


% c=decodifwenc(a,l)
% c aproximación a la matriz original sin ruido
% a matriz a la que se le aplica la multirresolución
% l son los niveles de multirresolución

[n,m]=size(a);
c=a;
nl=round(n/(2^l)); ml=round(m/(2^l));


% inicialización de las variables

for k=l:-1:1


% se trata del bucle para los niveles de multirresolución
% reordenamos la matriz haciendo el proceso inverso que en la codificación

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

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

for j=1:mb
c(1:nb,j)=wenodec(c(1:nb,j)',nb)';
end


% se hace el algoritmo de multirresolución inverso para las filas

for i=1:nb
c(i,1:mb)=wenodec(c(i,1:mb),mb);
end


% se calcula el nl para subir al siguiente nivel

nl=2*nl; ml=2*ml;
end
```

decodifpphc.m

```
function c=decodifpphc(a,l)


% c=decodifpphc(a,l)
% c aproximación a la matriz original sin ruido
% a matriz a la que se le aplica la multirresolución
% l son los niveles de multirresolución

[n,m]=size(a);
c=a;
nl=round(n/(2^l)); ml=round(m/(2^l));


% inicialización de las variables

for k=l:-1:1
```

```
% se trata del bucle para los niveles de multirresolución
% reordenamos la matriz haciendo el proceso inverso que en la codificación

nb=2*nl; mb=2*ml;
b=zeros(nb,mb);
b(1:2:nb,1:2:mb)=c(1:nl,1:ml);
b(2:2:nb,2:2:mb)=c(nl+1:nb,ml+1:mb);
b(1:2:nb,2:2:mb)=c(1:nl,ml+1:mb);
b(2:2:nb,1:2:mb)=c(1+nl:nb,1:ml);
c(1:nb,1:mb)=b;
clear b;


% se hace el algoritmo de multirresolución inverso para las columnas

for j=1:mb
c(1:nb,j)=pphdec(c(1:nb,j)',nb)';
end


% se hace el algoritmo de multirresolución inverso para las filas

for i=1:nb
c(i,1:mb)=pphdec(c(i,1:mb),mb);
end


% se calcula el nl para subir al siguiente nivel

nl=2*nl; ml=2*ml;
end
```

linealdec.m

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


% vector que contiene los valores significativos de la escala inferior
% y los detalles para subir
% n dimension de v

f=zeros(size(v));
nk1=n/2;


% predicción de los valores impares de f

q(1)=(11/8)*v(1)-(4/8)*v(3)+(1/8)*v(5);
f(1)=v(2)+q(1);

q(2:nk1-1)=(1/8)*v(1:2:n-5)+v(3:2:n-3)-(1/8)*v(5:2:n-1);
f(3:2:n-3)=v(4:2:n-2)+q(2:nk1-1);


q(nk1)=-(1/8)*v(n-5)+(4/8)*v(n-3)+(5/8)*v(n-1);
f(n-1)=v(n)+q(nk1);


% predicción de los valores pares de f

f(2:2:n)=2*v(1:2:n)-f(1:2:n);
```

enodec.m

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


% f=enodec(v,n);
% vector que contiene los valores significativos de la escala inferior
% y los detalles para subir
% n dimension de v
```

```
f=zeros(size(v));
nk1=n/2;


% predicción de los valores impares de f
% predicción del primer elemento

q(1)=(11/8)*v(1)-(4/8)*v(3)+(1/8)*v(5);
f(1)=v(2)+q(1);


% calculo del segundo detalle

ddc=abs(v(1)-2*v(3)+v(5));
ddr=abs(v(3)-2*v(5)+v(7));

if(ddc <= ddr)
   q(2)=(1/8)*v(1)+v(3)-(1/8)*v(5);
else
   q(2)= (11/8)*v(3)-(4/8)*v(5)+(1/8)*v(7);
end

f(3)=v(4)+q(2);


% predicción de los valores intermedios


for j=3:nk1-2

    ddl=abs(v(2*j-5)-2*v(2*j-3)+v(2*j-1));
    ddc=abs(v(2*j-3)-2*v(2*j-1)+v(2*j+1));
    ddr=abs(v(2*j-1)-2*v(2*j+1)+v(2*j+3));

    % predicción y detalle

    if (ddc <= ddl \& ddc <= ddr)
        q(j)=(1/8)*v(2*j-3)+v(2*j-1)-(1/8)*v(2*j+1);
    elseif(ddl<=ddr)
        q(j)=-(1/8)*v(2*j-5)+(4/8)*v(2*j-3)+(5/8)*v(2*j-1);
    else
        q(j)=(11/8)*v(2*j-1)-(4/8)*v(2*j+1)+(1/8)*v(2*j+3);
    end

    f(2*j-1)=v(2*j)+q(j);

end
```

```
% calculo del penultimo detalle

ddl=abs(v(n-7)-2*v(n-5)+v(n-3));
ddc=abs(v(n-5)-2*v(n-3)+v(n-1));

if(ddc<=ddl)
  q(nk1-1)=(1/8)*v(n-5)+v(n-3)-(1/8)*v(n-1);
else
  q(nk1-1)=-(1/8)*v(n-7)+(4/8)*v(n-5)+(5/8)*v(n-3);
end

f(n-3)=v(n-2)+q(nk1-1);


% predicción del ultimo elemento

q(nk1)=-(1/8)*v(n-5)+(4/8)*v(n-3)+(5/8)*v(n-1);
f(n-1)=v(n)+q(nk1);


% predicción de los valores pares de f

f(2:2:n)=2*v(1:2:n)-f(1:2:n);
```

enhdec.m

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


% f=enhdec(v,n);
% vector que contiene los valores significativos de la escala inferior
% y los detalles para subir
% n dimension de v

f=zeros(size(v));
nk1=n/2;
```

```
% predicción de los valores impares de f
% predicción del primer elemento

q(1)=(11/8)*v(1)-(4/8)*v(3)+(1/8)*v(5);
f(1)=v(2)+q(1);


% calculo del segundo detalle

dd1a=abs(v(3)-v(1));
dd1b=abs(v(3)-v(5));

if(dd1a <= dd1b)
    q(2)=(1/8)*v(1)+v(3)-(1/8)*v(5);
else
    dd2a=abs(v(1)-2*v(3)+v(5));
    dd2b=abs(v(3)-2*v(5)+v(7));
    if(dd2a <= dd2b)
       q(2)=(1/8)*v(1)+v(3)-(1/8)*v(5);
    else
       q(2)= (11/8)*v(3)-(4/8)*v(5)+(1/8)*v(7);
    end
end

f(3)=v(4)+q(2);


% predicción de los valores intermedios

for j=3:nk1-2
    dd1a=abs(v(2*j-3)-v(2*j-1));
    dd1b=abs(v(2*j-1)-v(2*j+1));

   if(dd1a <= dd1b)
      dd2a=abs(v(2*j-5)-2*v(2*j-3)+v(2*j-1));
      dd2b=abs(v(2*j-3)-2*v(2*j-1)+v(2*j+1));
      if(dd2a <= dd2b)
          q(j)=-(1/8)*v(2*j-5)+(4/8)*v(2*j-3)+(5/8)*v(2*j-1);
      else
          q(j)=(1/8)*v(2*j-3)+v(2*j-1)-(1/8)*v(2*j+1);
      end

   else
     dd2a=abs(v(2*j-3)-2*v(2*j-1)+v(2*j+1));
```

```
     dd2b=abs(v(2*j-1)-2*v(2*j+1)+v(2*j+3));
     if(dd2a <= dd2b)
       q(j)=(1/8)*v(2*j-3)+v(2*j-1)-(1/8)*v(2*j+1);
    else
       q(j)=(11/8)*v(2*j-1)-(4/8)*v(2*j+1)+(1/8)*v(2*j+3);
    end
   end

    f(2*j-1)=v(2*j)+q(j);
end


% calculo del penultimo detalle

dd1a=abs(v(n-5)-v(n-3));
dd1b=abs(v(n-3)-v(n-1));

if(dd1a <= dd1b)
    dd2a=abs(v(n-7)-2*v(n-5)+v(n-3));
    dd2b=abs(v(n-5)-2*v(n-3)+v(n-1));
    if(dd2a <= dd2b)
        q(nk1-1)=-(1/8)*v(n-7)+(4/8)*v(n-5)+(5/8)*v(n-3);
    else
        q(nk1-1)=(1/8)*v(n-5)+v(n-3)-(1/8)*v(n-1);
    end
else
        q(nk1-1)=(1/8)*v(n-5)+v(n-3)-(1/8)*v(n-1);
end
f(n-3)=v(n-2)+q(nk1-1);


% predicción del ultimo elemento

q(nk1)=-(1/8)*v(n-5)+(4/8)*v(n-3)+(5/8)*v(n-1);
f(n-1)=v(n)+q(nk1);


% predicción de los valores pares de f

f(2:2:n)=2*v(1:2:n)-f(1:2:n);
```

wenodec.m

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


% f=wenodec(v,n);
% vector que contiene los valores significativos de la escala inferior
% y los detalles para subir
% n dimension de v

f=zeros(size(v));
nk1=n/2;


% predicción de los valores impares de f
% predicción del primer elemento

q(1)=(11/8)*v(1)-(4/8)*v(3)+(1/8)*v(5);
f(1)=v(2)+q(1);


% calculo del segundo detalle

q(2)=(1/8)*v(1)+v(3)-(1/8)*v(5);
f(3)=v(4)+q(2);


% predicción de los valores intermedios

epsilon=10^(-6);
for j=3:nk1-2


% los pesos optimos en este caso de valores en celda son c0=3/16,
% c1=10/16, c2=3/16

    c0=3/16; c1=10/16; c2=3/16;


% calculo de los indicadores de suavidad
```

```
    I0=0.5*((v(2*j-5)-v(2*j-3))^2+(v(2*j-3)-v(2*j-1))^2)+...
    (v(2*j-5)-2*v(2*j-3)+v(2*j-1))^2;
    I1=0.5*((v(2*j-3)-v(2*j-1))^2+(v(2*j-1)-v(2*j+1))^2)+...
    (v(2*j-3)-2*v(2*j-1)+v(2*j+1))^2;
    I2=0.5*((v(2*j-1)-v(2*j+1))^2+(v(2*j+1)-v(2*j+3))^2)+...
    (v(2*j-1)-2*v(2*j+1)+v(2*j+3))^2;


% calculo de alfa0, alfa1, alfa2

    alfa0=c0/((epsilon+I0)^3);
    alfa1=c1/((epsilon+I1)^3);
    alfa2=c2/((epsilon+I2)^3);


% calculo de los pesos wj's

    s=alfa0+alfa1+alfa2;
    w0=alfa0/s;  w1=alfa1/s; w2=alfa2/s;


% predicción y detalle

    q(j)=w0*(-(1/8)*v(2*j-5)+(4/8)*v(2*j-3)+(5/8)*v(2*j-1))+ w1*((1/8)*v(2*j-3)+...
    v(2*j-1)-(1/8)*v(2*j+1))+w2*((11/8)*v(2*j-1)-(4/8)*v(2*j+1)+(1/8)*v(2*j+3));
    f(2*j-1)=v(2*j)+q(j);

end


% calculo del penultimo detalle

q(nk1-1)=(1/8)*v(n-5)+v(n-3)-(1/8)*v(n-1);
f(n-3)=v(n-2)+q(nk1-1);


% predicción del ultimo elemento

q(nk1)=-(1/8)*v(n-5)+(4/8)*v(n-3)+(5/8)*v(n-1);
f(n-1)=v(n)+q(nk1);


% predicción de los valores pares de f

f(2:2:n)=2*v(1:2:n)-f(1:2:n);
```

pphdec.m

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


% f=pphdec1(v,n);
% vector que contiene los valores significativos de la escala inferior
% y los detalles para subir
% n dimension de v

f=zeros(size(v));
nk1=n/2;


% predicción de los valores impares de f
% predicción del primer elemento

q(1)=(11/8)*v(1)-(4/8)*v(3)+(1/8)*v(5);
f(1)=v(2)+q(1);


% predicción de los valores intermedios
% calculo de las diferencias primeras

delta1=diff(v(1:2:n-3));
delta2=diff(v(3:2:n-1));


% parte de la predicción que corresponde al intervalo central

d=delta1.*delta2;
d1=delta1+delta2;
s=v(3:2:n-3);
d=(d>0).*d;
d1=(d>0).*d1+(d<=0).*ones(size(d1));
q(2:nk1-1)=s-0.5*d./d1;


% detalle

f(3:2:n-3)=v(4:2:n-2)+q(2:nk1-1);


% predicción del ultimo elemento
```

```
q(nk1)=-(1/8)*v(n-5)+(4/8)*v(n-3)+(5/8)*v(n-1);
f(n-1)=v(n)+q(nk1);


% predicción de los valores pares de f

f(2:2:n)=2*v(1:2:n)-f(1:2:n);
```

5.2. Código fuente del algoritmo de Suavizado

suaviza.m

```
function
suaviza(image\_file,tipo\_ruido,param\_ruido,filtro,param\_filtro)


% Esta función lee una imagen, le añade ruido y después la suaviza
% Da 2 imágenes de salida, la ruidosa original y la suavizada
% También escribe en un fichero datos sobre la calidad del suavizado
% suaviza(image_file,tipo_ruido,param_ruido,filtro,param_filtro);
% Variables de entrada
% image_file nombre de la imagen
% tipo_ruido identifica el tipo de ruido introducido a la imagen
% 1='gaussian', 2='salt & pepper', 3='speckle'
% param_ruido es un vector con dos componentes, y según el ruido introducido
nos da su media y varianza,
% densidad solamente en la primera componente sin importar la segunda,
% o varianza en la primera componente sin importar la segunda
% filtro 1=filtro gaussiano, 2=filtro de mediana, 3=filtro de vecindad
% param_filtro tamaño de la máscara del filtro y valor de la desviacion
tipica para el filtro gaussiano [N,M,sigma]



% leemos la imagen
```

```
a=imread(image\_file); [na,ma,ban]=size(a);


% le añadimos ruido

if tipo\_ruido==1
    a1=imnoise(a,'gaussian',param\_ruido(1),param\_ruido(2));
elseif tipo\_ruido==2
    a1=imnoise(a,'salt \& pepper',param\_ruido(1));
elseif tipo\_ruido==3
    a1=imnoise(a,'speckle',param\_ruido(1));
else
    a1=a;
end


% le pasamos el filtro a la imagen

N=param\_filtro(1); M=param\_filtro(2); sigma=param\_filtro(3);

if filtro==1
    mascara = fspecial('gaussian',[N,M],sigma);
    a2=imfilter(a1,mascara,'same','replicate');
elseif filtro==2
    for i=1:ban
       a2(:,:,i)=medfilt2(a1(:,:,i),[N,M]);
    end
elseif filtro==3
    mascara=1/N/M*ones(N,M);
    for i=1:ban
       a2(:,:,i)=conv2(double(a1(:,:,i)),double(mascara),'same');
    end
elseif filtro==4
    for i=1:ban
       a2(:,:,i)=wiener2(a1(:,:,i),[N,M]);
    end
end

figure('Tag','primera','Name',[blanks(6),'IMAGEN CON RUIDO'],...
'Position',[0 280 560 420]) if(ban==1)
 imagesc(a1,[0 255])
 colormap gray
else
 a1=uint8(a1);
 imshow(a1);
end
```

```
% title('imagen con ruido');

figure('Tag','segunda','Name',[blanks(6),'IMAGEN SUAVIZADA'],...
'Position', [0 16 560 420]) if(ban==1)
 imagesc(a2,[0 255])
 colormap gray
else
 a2=uint8(a2);
 imshow(a2);
end


% title('imagen suavizada');
```

6. Conclusión

En base a los estudios numéricos realizados en este proyecto se puede destacar que el método que mejores resultados ha dado es el basado en una reconstrucción PPH. El punto fuerte del algoritmo PPH son las imágenes con características geométricas pronunciadas, ya que en este tipo de imágenes consige reducir notablemente el ruido sin apenas afectar a las discontinuidades. En otro tipo de imágenes donde las características geométricas no son tan sobresalientes los resultados de los algoritmos lineales y del WENO son similares, pero el PPH elimina mejor los posibles artefactos numéricos como el efecto Gibbs o las difusiones en las discontinuidades, típicos efectos no deseados introducidos por algoritmos lineales.

Como hemos dicho, los resultados numéricos obtenidos son mejores para el algoritmo PPH, y esto se puede apreciar con una calidad de imagen mejor para las imágenes reconstruidas con este algoritmo con respecto al resto de algoritmos utilizados.

También se han analizado distintos filtros para la eliminación de ruido, y con los resultados obtenidos se puede considerar como mejor elección el filtro polinomial con un valor del parámetro m comprendido entre 2 y 3.

Así pues según nuestras observaciones el método que propondríamos utilizar para la eliminación de ruido gaussiano en imágenes digitales sería la combinación del algoritmo de multirresolución PPH con el filtro de truncación polinomial con $m = 2$. Notar que ambas elecciones comparten la misma filosofia de preservación de la calidad de la imagen alrededor de los ejes de la misma.

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., Busquier S. and Trillo J.C. (2006), Nonlinear Harten's Multiresolution on the Quincunx Pyramid, *J. of Comp. and App. Math.*, 189, 555-567.

[4] Amat S., Donat R., Liandrat J. and Trillo J.C. (2002), Analysis of a New Nonlinear Subdivision Scheme. Applications in Image Processing. *Foundations of Computational Mathematics*, 6(2), 193-225.

[5] Amat S., Cherif H. and Trillo J.C. (2005), Denoising using Linear and Nonlinear Multiresolutions, *Engineering Computations*, 22(7), 877-891.

[6] Amat S., Ruiz J. and Trillo J.C.(2008), Denoising using linear and nonlinear multiresolutions II: cell-average framework and color images, *Engineering Computations*, Accepted.

[7] Amat S., Ruiz J. and Trillo J.C., Fast multiresolution algorithms and their related variational problems for image denoising. Submitted.

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

[9] 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.

[10] 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.

[11] Bacchelli S. and Papi S., (2004). Filtered wavelets thresholding methods, *J. Comp. Appl. Math.*, 164-165, 39-52.

[12] Chambolle A., Devore R.A., Lee N. and Lucier B.J., (1998). Nonlinear wavelet image processing: variational problem, compression and noise removal through wavelet shrinkage, *IEEE Trans. Image. Processing.*, 7, 319-335.

[13] Cohen A., Daubechies I. anf Feauveau J.C., (1992). Biorthogonal bases of compactly supported wavelets, *Comm. in Pure and Appl. Math.*, 45, 485-560.

[14] Dahmen W. and Micchelli C.A., (1997). Biorthogonal wavelet expansions. *Constr. Approx.*, 13(3), 293-328.

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

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

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

[18] Fatemi E., Jerome J. and Osher S.,(1991). Solution of the hydrodynamic device model using high order non-oscillatory shock capturing algorithms. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 10, 232-244.

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

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

[21] 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.

[22] 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.

[23] Jiang G.S. and Shu C. W., (1996). Efficient implementation of weighted ENO schemes. *Journal of Computational Physics*, 126, 202-228.

[24] Lin E-B. and Ling Y., (2003). Image compression and denoising via nonseparable wavelet approximation, Journal of Computational and Applied Mathematics, to appear.

[25] Liu X. D., Osher S. and Chan T., (1994). Weighted essentially non-oscillatory schemes. *Journal of Computational Physics*, 115, 200-212.

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

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

[28] Shu C.W., (1990). Numerical experiments on the accuracy of ENO and modified ENO schemes. *Journal of Scientific Computing*, 5, 127-149.