PROYECTO FINAL DE CARRERA.

Multirresolución de Harten, aplicada a la compresión de imágenes digitales.

Comparación, en bits, con los formatos standard JPEG y PNG.

Autor: Jorge Cayuela Garcia.
Director: Juan Carlos Trillo Moya.

Departamento de matemática aplicada y estadística.
Escuela técnica superior de ingeniería industrial.
Universidad Politécnica de Cartagena.

Multirresolución de Harten, aplicada a la compresión de imágenes digitales.
Comparación, en bits, con los formatos standard JPEG y PNG.

Índice:

Multirresolución de Harten aplicada a la compresión de imágenes digitales. Comparación, en bits, con los formatos standard JPEG y PNG.

Parte I: Multirresolución de Harten aplicada a la compresión de imágenes digitales.

Introducción:

En la actualidad, se está usando bastante tecnología que es fácil de transportar, y de usar en cualquier lugar. Éste es el caso de los teléfonos móviles y de los ordenadores portátiles. Otra máquina móvil es la máquina fotográfica. Existen dos grupos de máquinas fotográficas: la digital y la analógica. Interesa que toda máquina transportable pese, ocupe, y, gaste lo menos posible. La máquina fotográfica digital se beneficia de las mejoras en este campo. Con los datos de este proyecto, o, continuándolo, se puede programar un método de compresión de imágenes en el circuito programable más adecuado para la aplicación requerida; adaptando, para ello, la lengua empleada en Matlab a la requerida por el citado circuito (procesado digital de señal, microcontrolador, FPGA, etcétera –como puede ser: D.S.P, lengua ensamblador, TRAC, V.H.D.L, etcétera–).

La imagen, cuando es digitalizada, es transformada en una matriz de números, números que representan la media de intensidad lumínica que existe en un determinado punto de la fotografía, punto que recibe el nombre de "píxel". Aplicando la multirresolución de Ami Harten a estos datos conseguimos una versión de los mismos adaptada a procesos de compresión mediante el uso de técnicas de cuantización. El éxito de estas técnicas depende en gran medida de la localización de los elementos menores que una cierta tolerancia de cuantización, pues la versión comprimida de la imagen requiere guardar los elementos mayores que la tolerancia y poder localizar dónde están. Así uno de los pasos de la descompresión de la imagen consistirá en volver a situar exactamente en el mismo lugar en el que estaban en la matriz de multirresolución los valores que se han considerado significantes (mayores que la tolerancia), y el resto de las posiciones de los valores de la matriz rellenarlos con ceros .

Hemos utilizado el método de Ami Harten con una discretización por valores en celda, como se explicará más adelante. La reconstrucción que utilizamos será lineal, y se basa en el polinomio interpolador de Lagrange. A partir de estos operadores se definen otros dos operadores que actuan entre dos niveles consecutivos de resolución.
A estos dos operadores se les llama operadores decimación, que siempre es lineal, y predicción, que puede ser no lineal y que permite mayor adaptación a las características inherentes de la imagen en concreto. El operador predicción da lugar a unos errores entre los valores exactos y los predichos, de manera que lo que se busca es conseguir buenos operadores de predicción. El usado en este proyecto es ampliamente utilizado, y da lugar a los conocidos 'Wavelets Biortogonales'. Al conjunto de errores a veces se les llama también detalles, los cuales son necesarios para recuperar una escala fina a partir de una grosera.

La calidad de la imagen reconstruida depende de la tolerancia escogida en la compresión. Cuando las imágenes son ruidosas (nosotros interpretamos que es ruido blanco, es decir, ruido gaussiano) el algoritmo implementado de compresión supone también el llevar a cabo una reducción del ruido. Hemos realizado algunos tests

introduciendo nosotros más ruido en las imágenes, el ruido empleado en este proyecto como acabamos de mencionar es el ruido blanco, que es el introducido por los mismos circuitos que captan la imagen (por los de la cámara fotográfica).

A continuación, se pasa a explicar el fundamento matemático de este proyecto.

S.1. Sección uno. Redes de multirresolución por medias en celda en el intervalo [0,1]:

Se considera $F = B[0,1]$ *el espacio de funciones acotadas en el intervalo cerrado unidad [0,1] , y sea* X^L, *una partición uniforme del intervalo cerrado [0,1]. Considerando lo aquí expuesto, se tiene lo siguiente:*

$$X^L = \{x_i^L\}_{i=0}^{i=J_L},\ x_0^L,\ x_i^L = i\,h_L,\ h_L = \frac{1}{J_L},\ J_L = 2^L J_0 \qquad (S.1.1)$$

donde "J_0" es un entero. Se definen las mallas de menor re $X^k = \{x_i^k\}_{i=0}^{i=J_k}$, *para k = L, L-1, ... , 1:*

$$x_i^{k-1} = x_{2i}^k,\ i = 0,\ 1,\ \ldots,\ J_{k-1} \triangleq \frac{J_k}{2}, \qquad (S.1.2)$$

y se considera esto:

$$(\bar{f}_i)^k = (D_k f)_i = \frac{1}{h_k}\int_{x_{i-1}^k}^{x_i^k} f(x)\,dx, \qquad (D_k^{k-1}(\bar{f})^k)_i = \frac{1}{2}\left((\bar{f})_{2i-1}^k + (\bar{f})_{2i}^k\right).$$

"k" es el número de elementos de la imagen original.

"k-1" es el número de elementos, después de que cada dos segmentos contiguos se conviertan en un único segmento; de esta manera, el vector ha reducido su tamaño original.

"D_k" es el operador "decimación", que es el que se encarga de bajar de nivel.

"f " es la función que está siendo analizada.

"h_k" es la distancia que existe entre dos celdas consecutivas, en el nivel "k".

"$(f)_{2i-1}^k$ " representa los elementos con índice impar de la discretización de la función "f " por medias en celda en el nivel "k".

"$(f)_{2i}^k$ " representa los elementos con índice par de la función "f " en el nivel "k".

Los errores (e^k) producidos por el operador predicción pertenecen al núcleo del operador decimación. Entonces:

" $0 = (D_k^{k-1} e^k)_i = \frac{e_{2i-1}^k + e_{2i}^k}{2}$ ", al bajar cada nivel.

" D_k^{k-1} " es una aplicación que transforma elementos del nivel "k" en elementos del nivel "k-1", es decir, reduce la dimensión del vector. Se le llama operador decimación.

De esta manera, se tiene que:

$$(d_j^k \equiv e_{2j-1}^k) \leftrightarrow (e_{2j}^k \equiv -d_j^k) \qquad 1 \le j \le \frac{J_k}{2} = J_{k-1} \qquad (S.1.3).$$

Así de los errores, sólo son guardados los impares, ya que los pares pueden ser calculados, fácilmente, a partir de los impares.

En una dimensión, el "camino de reconstrucción usando la función primitiva" nos permite establecer una relación con la teoría de interpolación que será útil en el análisis,

el cual va a ser descrito a continuación. Si se quieren ver más detalles, se pueden consultar las siguientes referencias bibliográficas [2], [3].

Se define la secuencia: " $\{F_i^k\}$ ", sobre la malla k-ésima como:

$F_i^k = h_k \cdot \sum_{j=1}^{j=i} \bar{f}_j^k = \int_0^{x_i^k} f(y)\,dy$, entonces : $F_i^k = F(x_i^k)$, donde: $F(x) = \int_0^x f(y)\,dy$

$(F(x) \in C[0,1])$ es una "primitiva" de f(x) (observar que: $f(x) \in L^1[0,1]$ implica que: $F(x) \in C[0,1]$). La secuencia: $\{F_i^k\}_{i=0}^{i=J_k}$ corresponde a la obtención de valores (puntos) concretos de F(x) en la malla k-ésima. Además: " $\bar{f}_i^k = \frac{F_i^k - F_{i-1}^k}{h_k}$ (S.1.4), y, esta relación, establece una correspondencia uno a uno entre los valores (puntos) de F(x) y las medias en celda de f(x) en la misma malla. Nótese que:

$F_0^k = 0, \quad k$

Se va a denotar así: "I(x; F^k)", al operador "reconstrucción" para los valores de la función primitiva, que son valores puntuales. Es decir que en este caso es una técnica de interpolación. Entonces, definiendo esto:

$$R_k(x; \{\bar{f}^k\}) \triangleq \frac{d}{dx} I(x; F^k) \qquad \text{(S.1.5)},$$

se obtiene, fácilmente esto otro:

$$D_k(R_k(x; \{\bar{f}^k\}))_j = \frac{1}{h_k} \cdot \int_{x_{j-1}^k}^{x_j^k} \frac{d}{dx} I(x; F^k)\,dx = \frac{F_j^k - F_{j-1}^k}{h_k} = \bar{f}_j^k \qquad \text{(S.1.6)}.$$

Así : " $D_k R_k = I_k$ (observar que: $R_k(\{\bar{f}^k\}) \in L^1[0,1]$ ").

.

Con estas definiciones la transformada de multirresolución (S.1.7), y su inversa (S.1.8):

Codificación: $v^L \to \{M_v\}^L = (v^0, d^1, \ldots, d^L)$ $\left\{ \begin{array}{l} \textit{Hacer: } k = L, \ldots, 1. \\ v^{k-1} = D_k^{k-1} \cdot v^k \\ d^k = G_k(v^k - P_{k-1}^k \cdot v^{k-1}) \end{array} \right\}$ *(S.1.7).*

Decodificación: $\{M_v\}^L \to M^{-1} \cdot M \cdot v^L$ $\left\{ \begin{array}{l} \textit{Hacer: } k = 1, \ldots, L. \\ v^k = P_{k-1}^k \cdot v^{k-1} + E_k \cdot d^k \end{array} \right\}$ *(S.1.8),*

pueden ser descritas de la siguiente manera:

$\{\bar{f}^L\} \mu M \cdot \{\bar{f}^L\}$ $\left\{ \begin{array}{l} \text{Hacer: } k = L, \ldots, 1. \\ (\bar{f}_i)^{k-1} = \frac{1}{2} \cdot (\bar{f}_{2i-1})^k + (\bar{f}_{2i})^k \qquad i = 1, \ldots, J_{k-1} \ . \end{array} \right\}$

$$d_i^k = \left|\bar{f}_{2i-1}\right|^k - A\left(P_{k-1}^k \times \left|\bar{\bar{f}}\right|^{k-1}\right)_{2i-1} \quad i = 1, \ldots, J_{k-1} \ . \qquad \text{(S.1.9).}$$

$$M^a\left|\bar{\bar{f}}\right|^L \mu\ M^{-1a} M^a\left|\bar{\bar{f}}\right|^L \left\{ \begin{array}{ll} \text{Hacer: } k = 1, \ldots, L. & \\ \left|\bar{f}_{2i-1}\right|^k = A\left(P_{k-1}^k\right)^a \left|\bar{f}\right|^{k-1}_{2i-1} + d_i^k & i = 1, \ldots, J_{k-1} \ . \\ \left|\bar{f}_{2i}\right|^k = 2^a \left|\bar{f}_i\right|^{k-1} - \left|\bar{f}_{2i-1}\right|^k & i = 1, \ldots, J_{k-1} \ . \end{array} \right\} \qquad \text{(S.1.10).}$$

Hay una relación directa entre los siguientes dos términos: "$d^k(f\ )$" (que son los coeficientes de detalles de la multirresolución por medias en celda), y "$d^k(F)$" (que son los coeficientes de detalles de la multirresolución por valores puntuales cuando se usa la función primitiva correspondiente); efectivamente:

$$d_j^k(f) = e_{2j-1}^k = \left|\bar{f}_{2j-1}\right|^k - A\left(P_{k-1}^k\right)^a D_k^{k-1a} \left|\bar{f}\right|^k_{2j-1} = \left|\bar{f}\right|^k_{2j-1} - A\left(D_k{}^a R_{k-1}\right)^a \left|\bar{f}\right|^{k-1}_{2j-1} =$$

$$\frac{1}{h_k}{}^a A\left|\left(F_{2j-1}\right)^k - \left(F_{2j-2}\right)^k\right| - \frac{1}{h_k}{}^a \int_{x_{2j-2}^k}^{x_{2j-1}^k} \frac{\partial}{\partial x} A\left|I\left|x;\ F^{k-1}\right|\right|^a dx$$

$$= \frac{1}{h_k}{}^a A\left|\left(F_{2j-1}\right)^k - \left(F_{2j-2}\right)^k\right| - \left|I\left|x_{2j-1}^k;\ F^{k-1}\right| - I\left|x_{2j-2}^k;\ F^{k-1}\right|\right|$$

$$= \frac{1}{h_k} \times \left[\left[F_{2j-1}\right]^k - I\left(x_{2j-1}^k;\ F^{k-1}\right)\right] = \frac{\left[d_j(F)\right]^k}{h_k} \rightarrow d_j^k(f) = \frac{\left[d_j(F)\right]^k}{h_k} \qquad \text{(S.1.11)}$$

$$d_j^k(f) = \frac{1}{h_k}{}^a d_j^k(F) \rightarrow d^k(f) = \frac{1}{h_k}{}^a d^k(F) \rightarrow h_k{}^a d^k(f) = d^k(F).$$

Vamos a restringir este estudio a las técnicas de interpolación segmentarias usando el polinomio interpolador de Lagrange, las cuales conducen a los operadores de reconstrucción : "$R_k\left|x;\ f\right|^k$", que son funciones polinómicas a trozos.

Observación:

Si "$I(x; F^k)$" es una función de interpolación polinómica a trozos, de grado "r +1", es decir:
"$I\left|x;\ F^k\right| = q_i\left|x;\ F^k\right| \quad x \varepsilon A x_{i-1}^k, x_i^k E \ \ddot{\imath}\ \deg(q_i) = r+1$ ", entonces: "$R_k\left|x;\ \bar{f}\right|^k$" es una función polinómica a trozos de grado "r".

Por causa de unicidad, esta reconstrucción es idéntica a la que está definida por (S.1.12).

Se va a denotar: " $p_i(x; D_k f)$ " al único polinomio de grado "r" que logra los promedios " $(D_k f)_{im}$ " en " S_i^k " , por ejemplo, el que satisface el siguiente sistema de ecuaciones lineales para sus coeficientes:

$$\frac{1}{\Delta x\, c_{im}^k} \sum_{c_{im}^k} p_i(x; D_k f)\, dx = (D_k f)_{im} \qquad m = 1, \ldots, s(r). \tag{S.1.12.}$$

El proceso de reconstrucción de la función primitiva implica que "R_k" es exacto para polinomios de grado no superior a "r". Por otra parte, debido a que I(x; F^k) es siempre un grado mayor que la reconstrucción para las medias en celda, y como los trozos del polinomio I(x; F^k) son de grado: "r + 1", los resultados de lo explicado hasta ahora, implican que, en regiones suaves, se cumple esto: "$d^k(f) = O(h_{k-1}^r)$ ".

Si el conjunto de celdas usadas para construir el trozo de polinomio envuelto en la computación de un coeficiente de escala o detalle
atraviesa una discontinuidad en $f^{(p)}$ $p \le r$, se tendrá esto:

$$d^k(f) = O \mid A f^{(p)} \mid h_{k-1}^p \ .$$

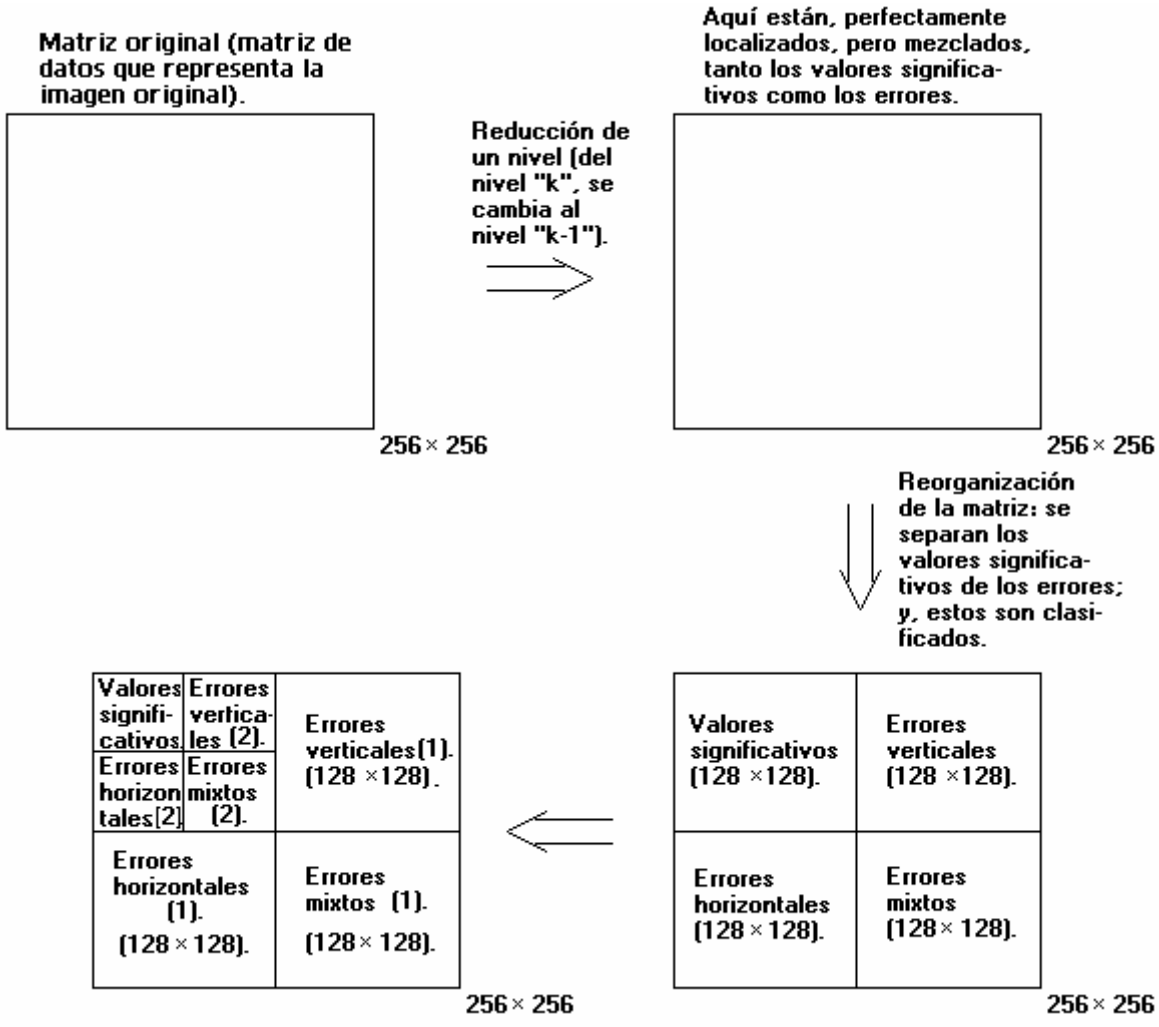
El grado de los trozos del polinomio (es decir, lo exacto de la reconstrucción) determina la velocidad en que decaen los coeficientes de escala o detalles, exactamente, de la misma manera que en el contexto de la multirresolución interpolatoria. Como se puede consultar en la bibliografía facilitada [1], el uso de las técnicas de interpolación no lineal "E.N.O" ayuda a reducir el componente de error de la aproximación de los coeficientes de escala. Esto conduce a esquemas multirresolutivos con capacidades de comprensión más altas cuando las señales con las que se trabaja presentan discontinuidades aisladas.

A diferencia de la discretización por valores puntuales, donde la información sobre la localización "exacta" de una discontinuidad está perdida, el proceso de discretización por medias en celda retiene esta información. En [4], Harten muestra cómo recobrar la localización de una discontinuidad en una función suave a trozos a partir de sus datos de medias en celda. La observación clave es que un salto discontinuo en f(x) llega a ser una "esquina" en F(x). Esta observación lleva a aplicar técnicas "E.N.O-S.R" para obtener esquemas de multirresolución para funciones suaves a trozos con saltos con capacidades de compresión cercanas a las óptimas.

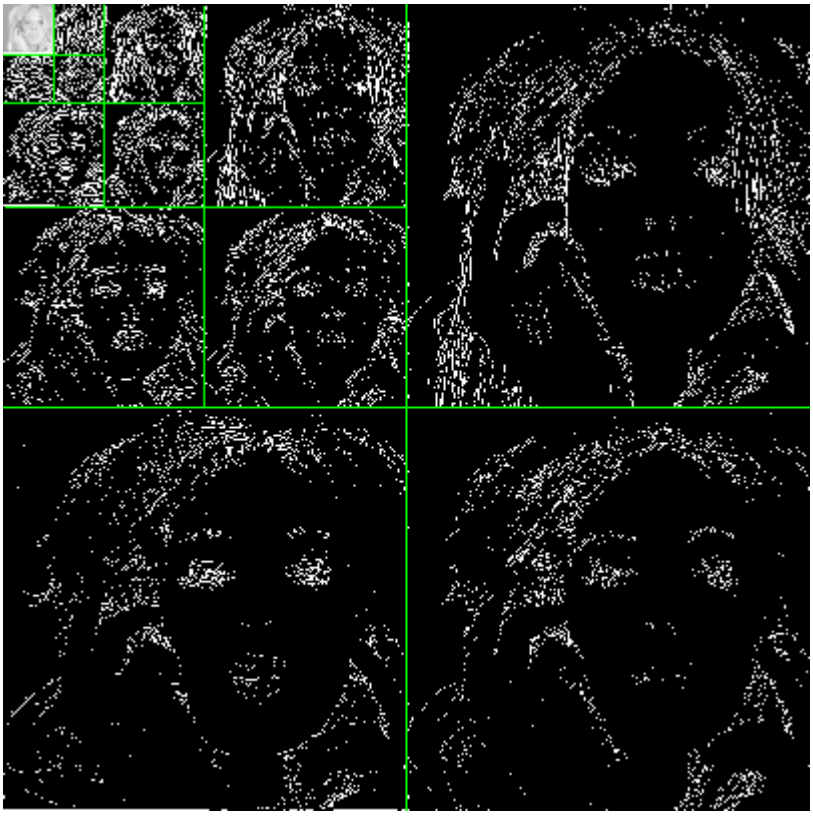
S.2. Sección dos. Aplicación, en el programa informático "Matlab", del desarrollo matemático realizado en la anterior sección:

S.2.1. Idea gráfica de la sección S.1:

En primer lugar damos una explicación gráfica del proceso de multirresolución llevado a cabo con la matriz original. Para ello explicamos como bajar de un nivel de multirresolución a otro.



Ejemplo con 4 pasos de multirresolución:



Los "valores significativos" son los valores que provienen directamente de la matriz original a través del operador de decimación.

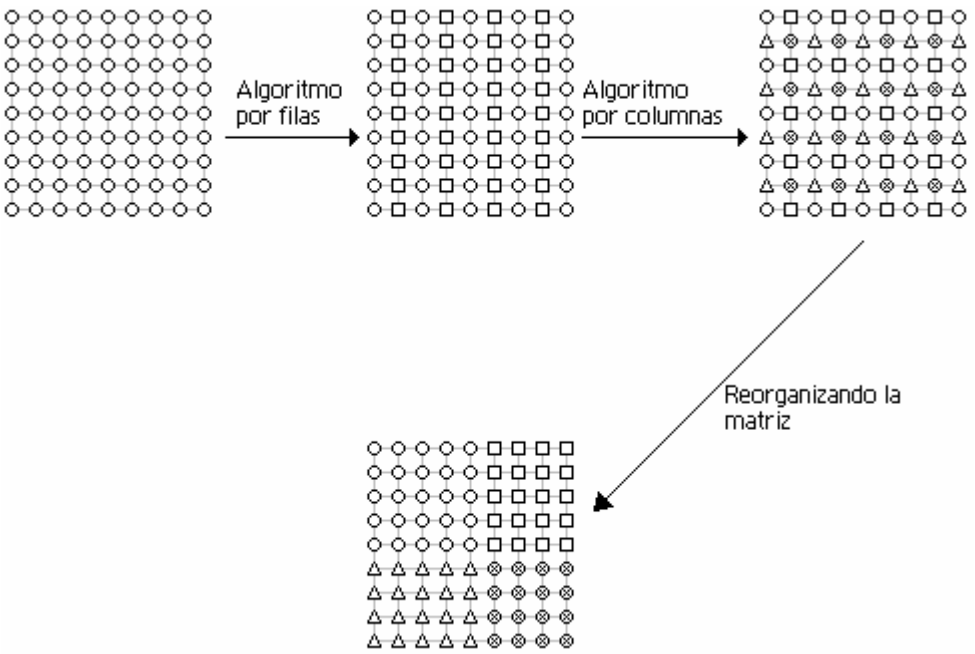
La aplicación del algoritmo de multirresolución en 1D explicado en la sección anterior en 2D se hace utilizando el llamado producto tensorial. En esta aproximación los errores son clasificados en tres grupos, que son el resultado de cómo es aplicada la aproximación por producto tensorial. Siempre se procede de la misma manera. Lo primero es procesar la matriz fila a fila, lo que permite hallar los errores horizontales; después, se procesa la matriz columna a columna, lo que permite hallar los errores verticales. Cuando los valores que sirven para interpolar los valores medios de las celdas son valores significativos de la matriz original, los valores que son hallados son llamados, o "errores verticales", o "errores horizontales" (según cómo hallan sido calculados); pero, cuando los valores que son usados en la interpolación son errores (tanto si son verticales, como si son horizontales), al resultado de la interpolación se le asigna el nombre de: "error mixto".

En cuanto han sido hallados los errores, la matriz es reorganizada siguiendo los criterios del gráfico de arriba.

Si se quiere bajar más de un nivel –o subir–, lo que ha sido llamado así: "valores significativos", pasarán a ser considerados la matriz original, y, esa submatriz será tratada como arriba se indicó que se trataba la matriz original, sin separar la submatriz de la matriz original; el resultado es que en la esquina superior izquierda de la matriz, quedarán los puntos de la matriz más relevantes, mientras que en el resto de la matriz, se almacenarán los errores que permitirán reconstruir la matriz original. Cuando esos errores sean inferiores a un determinado valor, que indica el máximo error de reconstrucción admisible en una determinada aplicación, ese coeficiente será puesto a cero. En cuanto sean localizados y almacenados todos los valores distintos de cero de la matriz, podrá comprimirse y reconstruirse la imagen original, sin que exista una pérdida importante de información; de hecho, la información que se perdería es escogida por el usuario –por quien comprime la imagen–, al seleccionar el máximo valor hasta el que podrá aproximarse un error por el cero. En cuanto sean localizados los ceros de la matriz, reorganizando la información que quede se obtendrá una estructura de datos que será más pequeña que la original (por lo que se habrá comprimido la matriz original), pero que contendrá la suficiente información como para reconstruir la imagen original (con lo que la pérdida de información será mínima).

La idea gráfica que ha sido usada representa una matriz de $256 \cdot 256$, que es uno de los dos tamaños empleados en este proyecto; el otro es de $512 \cdot 512$; y, para adaptar el gráfico al nuevo tamaño, no hay que hacer nada más que multiplicar por dos los números: 128, y, 256 del anterior gráfico.

Aunque este proyecto ha sido desarrollado para tratar la multirresolución por medias en celda, por hacer más claro el proceso, a continuación, se va a mostrar una representación simbólica de la multirresolución de una matriz usando valores puntuales; es decir, en esta matriz, se va a representar lo mismo que se hace con medias en celda, pero aplicado a valores puntuales.



◯ ≡ valores significativos.

□ ≡ errores verticales.

△ ≡ errores horizontales.

⊗ ≡ errores mixtos.

S.2.2. Funciones del programa *Matlab* que han sido desarrolladas para tratar las imágenes según la aproximación de Ami Harten:

Estos cuadros muestran las relaciones existentes entre las funciones que son utilizadas en este proyecto:

Carpeta "unioncomdescom":

Función:	Llama a la siguiente función:	Esta función ha sido programada para aplicar el método de Ami Harten:	Es llamada por la siguiente función:
unioncomdescom	compresor	Sí.	Ninguna función.
	descompresor	Sí.	
compresor	imread size fopen		unioncomdescom
	descenderc	Sí.	
	valorescom	Sí.	
descompresor	valoresdescom	Sí.	unioncomdescom
	ascenderc	Sí.	
	size max norm sqrt log		

El resto de los programas de la carpeta llamada así: "unioncomdescom" ("codifli1c.m", "decodiflic.m", "linealde1c.m", "lineale1c.m"), sólo se diferencia de sus homónimos de la carpeta llamada así: "comdescom" en que, en la carpeta: "unioncomdescom", el único programa con comentarios (y, con ayuda) es el llamado así: "unioncomdescom", mientras que en la carpeta "comdescom", todo programa (toda función) tiene comentarios; como mínimo, tiene ayuda.
La carpeta "comdescom" mantiene por separado los programas de compresión y los de descompresión. Por el contrario, en la carpeta "unioncomdescom" se han unido ambos códigos de manera que resulte más sencillo obtener una reconstrucción a la imagen original a partir de la versión comprimida, y así poder hacer comparaciones sobre la calidad de las reconstrucciones.

Carpeta “comdescom”:

Función:	Llama a la siguiente función:	Esta función ha sido programada para aplicar el método de Ami Harten:	Es llamada por la siguiente función:
compresor	imread size fopen		Ninguna función.
	descenderc	Sí.	
	valorescom	Sí.	
descompresor	valoresdescom	Sí.	Ninguna función.
	ascenderc	Sí.	
valorescom	find length		compresor
valoresdescom	length size max		descompresor
descenderc	size max randn		compresor
	codifli1c	Sí.	
codifli1c	lineale1c	Sí.	descenderc
	abs find length nnz		
lineale1c		Sí.	codifli1c
ascenderc	decodiflic	Sí.	descompresor
decodiflic	round zeros		ascenderc
	linealde1c	Sí.	
linealde1c	size zeros		decodiflic

S.2.2.1. unioncomdescom:

La función que ha sido programada es ésta:

```
function [a,a1,mra,v,i,j,ta]=unioncomdescom(l,tol,ruido,im,met)

% La sintaxis de la funcion "unioncomdescom" es esta:
% "[a,a1,mra,v,i,j,ta]=unioncomdescom(l,tol,ruido,im,met);".
% "l", son los niveles de multirresolucion.   Escribir un valor entero, que
%    sugerimos que pertenezca a este intervalo:  [0,7].   El cero indica que
%    no se va a comprimir la imagen, mientras que el siete es la maxima
%    compresion que puede hacerse con esta funcion.
% "tol" es la tolerancia de truncamiento.   Escribir un valor que puede ser,
%       tanto entero, como decimal.   Este valor determina la cantidad de
%       errores que guarda la funcion para reconstruir la imagen original.
%       Cuanto mayor sea este valor, menos datos tendra la funcion para
%       reconstruir la imagen original.   La funcion no guarda error alguno
%       si la tolerancia es suficientemente grande, por lo que la imagen
%       reconstruida esta difuminada.
% "ruido" es el nivel de ruido que se suma a la imagen.   Este ruido representa
%         el ruido introducido por la camara.
% "im" es la imagen que utiliza la funcion "compresor", escribir lo siguiente
%      (es una guia):  'nombre de la imagen.extension de la imagen'.
% "met" metodo que se quiere utilizar: 'lin'(significa:  "lineal", y,
%       actualmente, no ha sido programado ningun otro metodo).
% Esta funcion genera estas salidas:
% "rat" es el numero total de elementos de la imagen, dividido por los
%       elementos de la imagen que han sido guardados en la compresion
%       de la misma.
% "Com" es el tanto por ciento que es comprimida la imagen.
% "PSNR" es una medida de la calidad de la recontruccion realizada de la
%        imagen despues de haber sido comprimida.   Cuanto mayor es el valor de
%        "PSNR", mejor es la calidad de la imagen que ha sido reconstruida.
% "a" es la variable donde es guardada la imagen que primero ha sido comprimida,
%     y, despues, descomprimida.
% "a1" es la variable donde es guardada la imagen original.
% "mra" es donde es guardada la imagen comprimida, junto con los errores
%       que son necesarios para poder reconstruirla.
% "v" es el vector que contiene los valores significativos de la
%     imagen reducida y los errores guardados ("v" contiene la imagen
%     comprimida).
% "i", "j" son los vectores que contienen la ubicacion de los valores del
%          vector "v" (son imprescindibles en la descompresion de la imagen).
% "ta" es el tamaño de la imagen (512 significa:  "512·512", 256, signi-
%      fica:  "256·256).
```

```
% medimos el tiempo que tarda en hacer la compresion

to=clock;


% Se llama a la funcion compresor:
[v,i,j,l,met,ta,a1]=compresor(l,tol,ruido,im,met);


t1=etime(clock,to)

% Se llama a la funcion descompresor:
[a,mra]=descompresor(i,j,v,l,met,ta,a1);

% Se muestran:  la imagen original, la comprimida, y posteriormente
%           reconstruida, la reducida, los errores que seran usados
%           para reconstruir la imagen original, la imagen reducida
%           junto con los errores.

figure(1)
imagesc(a1,[0 255])
colormap gray
title('Imagen original.')
axis('square')


figure(2)
imagesc(a,[0,255])
colormap gray
axis('square')
title('Imagen comprimida, y, posteriormente, reconstruida.')

figure(3)
imagesc(mra,[0 255])
colormap gray
title('Imagen reducida.')
axis('square')

figure(4)
imshow(mra)
title('Errores.')
axis('square')

figure(5)
imagesc(mra,[0 255])
```

```
colormap pink
title('Imagen reducida, y, errores.')
axis('square')
```

Éste es el análisis de la función "unioncomdescom":

```
Se llama a la funcion compresor.
Se llama a la funcion descompresor.
Se muestran:  la imagen original, la comprimida, y posteriormente
              reconstruida, la reducida, los errores que seran usados
              para reconstruir la imagen original, la imagen reducida
              junto con los errores.
```

≡ Comentarios internos de la función (no aparecen en la ayuda).

Otras funciones utilizadas son:

to=clock;	≡ capta la fecha y la hora en el instante en el que es usada.
t1=etime(clock,to)	≡ proporciona la diferencia de segundos que han transcurrido entre el instante "to" y la nueva llamada a clock.
figure(1)	≡ hace aparecer una ventana, donde aparecerá la imagen.
imagesc(a1,[0 255])	≡ hace aparecer la imagen de "a1" en la anterior ventana (en: "figure(1)").
colormap gray	≡ hace que la imagen aparezca en escala de grises.
title('Imagen original.')	≡ le coloca un título a la imagen.
axis('square')	≡ provoca que la imagen esté limitada por un cuadrado.

La función: "unioncomdescom" ha sido desarrollada para realizar los procesos de compresión y descompresión en una misma función y permitir medir la calidad de la imagen reconstruida resultante.

S.2.2.2. compresor:

La función que ha sido programada es ésta:

```
function [v,i,j,l,met,ta]=compresor(l,tol,ruido,im,met)

% La sintaxis de la funcion "compresor" es esta:
% "[v,i,j,l,met,ta]=compresor(l,tol,ruido,im,met);".
% "l", son los niveles de multirresolucion.  Escribir un valor entero, que
%   sugerimos que pertenezca a este intervalo:  [0,7].  El cero indica que
%   no se va a comprimir la imagen, mientras que el siete es la maxima
```

```
%    compresion que puede hacerse con esta funcion.
% "tol" es la tolerancia de truncamiento.   Escribir un valor que puede ser,
%       tanto entero, como decimal.   Este valor determina la cantidad de
%       errores que guarda la funcion para reconstruir la imagen original.
%       Cuanto mayor sea este valor, menos datos tendra la funcion para
%       reconstruir la imagen original.   La funcion no guarda error alguno
%       si la tolerancia es suficientemente grande, por lo que la imagen
%       reconstruida esta difuminada.
% "ruido" es el nivel de ruido que se suma a la imagen.   Este ruido representa
%         el ruido introducido por la camara.   Se sugiere escribir un valor
%         entero que pertenezca a este intervalo:  [0,12].
% "im" es la imagen que utiliza la funcion "compresor", escribir lo siguiente
%      (es una guia):  'nombre de la imagen.extension de la imagen'.
% "met" metodo que se quiere utilizar: 'lin'(significa:  "lineal", y,
%       actualmente, no ha sido programado ningun otro metodo).
% Esta funcion genera estas salidas:
% "rat" es el numero total de elementos de la imagen, dividido por los
%       elementos de la imagen que han sido guardados en la compresion
%       de la misma.
% "Com" es el tanto por ciento que es comprimida la imagen.
% "v" es el vector que contiene los valores significativos de la
%     imagen reducida y los errores guardados ("v" contiene la imagen
%     comprimida).
% "i", "j" son los vectores que contienen la ubicacion de los valores del
%          vector "v" (son imprescindibles en la descompresion de la imagen).
% "ta" es el tamaño de la imagen (ta=512, significa:  imagen=512·512.
%      ta=256, significa:  imagen=256·256).




% Tamaño de la imagen:
a=imread(im);
[tax,tay]=size(a);
ta=tax;
a=double(a);
% Definicion de la matriz, la guardamos en la variable "a".
% y calculo del tamaño.


% La matriz original es guardada en "a1", para, despues, calcular
% los errores cometidos.

% Se desciende por la piramide de multiresolucion.
```

```
[a1,a]=descenderc(a,l,tol,ruido,met);
mra=a;

% Los errores son calculados.

% Compresion, tanto de la imagen reducida, como de los errores guardados:
[v,i,j]=valorescom(mra);
```

Éste es el análisis de la función "compresor":

Ésta es la función que se encarga de comprimir la imagen. Se comienza descendiendo en la escala de multirresolución, para, posteriormente, truncar todos los ceros y guardar todos los valores significativos de la imagen en un vector.

Algunas de las sentencias, comandos y funciones que aparecen se explican a continuación.

a=imread(im);	≡ la matriz de la imagen (im) es guardada en la variable "a".
[tax,tay]=size(a);	≡ el tamaño de "a" es guardado en las variables: "tax" (longitud de la matriz, medida en el eje horizontal), y, "tay" (longitud de la matriz, medida en el eje vertical).
ta=tax;	≡ la variable "tax" es guardada en la variable "ta" (que, al ser –las fotografías– matrices cuadradas, indica el tamaño de la imagen).
[a1,a]=descenderc(a,l,tol,ruido,met);	≡ es llamada la función "descenderc".
mra=a;	≡ el contenido de la variable "a" es guardado en la variable: "mra" (multirresolución de la matriz "a").
[v,i,j]=valorescom(mra);	≡ es llamada la función "valorescom".

S.2.2.3. descompresor:

Existen dos funciones "descompresor": una función es la que menos código tiene, y se encuentra en la carpeta "comdescom", que está preparada para descomprimir los datos que reciba de la función compresor (la que no proporciona la variable "a1", que es la imagen original; ya que, si, cuando es comprimida una imagen, también, se debe llevar la imagen original sin comprimir, el compresor es inútil), y, la otra función es la que más código tiene que es la que ha sido preparada para mostrar, de manera cómoda, cómo funcionan las funciones: "compresor", y, "descompresor" y hacer medidas para averiguar cuánto de buena es la reconstrucción.

La parte del código que no comparten ambas funciones "descompresor" será destacado en cursiva.

La función es ésta:

```
function [a]=descompresor(i,j,v,l,met,ta)

% Esta funcion expande la imagen que comprimio la funcion llamada asi:
% "compresor".
% La sintaxis de la funcion es esta:  "[a]=descompresor(i,j,v,l,met,ta);".
% "i", y, "j" son los vectores que contienen la ubicacion de los valores
%            significativos de la imagen original, y de los errores no
%            truncados.
% "v" es el vector que contiene los valores que han sido guardados para
%     poder reconstruir la imagen original.
% "l" son los niveles de multirresolucion (es un valor entero)
% "met" es el metodo de multirresolucion;  actualmente, solo esta programado
%       el metodo lineal ('lin').
% "ta" es el tamaño de la imagen (ta=512, significa: imagen=512·512.
%      ta=256, significa:  256·256).




% Se llama a la funcion que descomprime, tanto la imagen reducida, como los
% errores que seran usados para reconstruir la imagen original:
[mra]=valoresdescom(i,j,v,ta);

% Se llama a la funcion que, ascendiendo por la escala de multirresolucion,
% reconstruira la imagen original:
[a]=ascenderc(mra,l,met);
```

La función "descompresor" con más código, que se encuentra en la carpeta "unioncomdescom", es ésta:

```
function [a,mra]=descompresor(i,j,v,l,met,ta,a1)

[mra]=valoresdescom(i,j,v,ta);          ≡ se llama a la función:  "valoresdescom".
[a]=ascenderc(mra,l,met);               ≡ se llama a la función:  "ascenderc".
```

El resto de este código no está en ambas funciones:

```
n1=max(size(a));                        ≡ el máximo valor de las dos longitudes
                                          (horizontal, y, vertical) que proporciona la
                                          función:  "size".
```

MSE=(norm(a-a1,'fro')^2)/n1/n1; ≡ error cuadrático medio usado para definir el PSNR.

$$M.S.E = \frac{\sum (a_{i.j.} - a1_{i.j.})^2}{(n1)^2} .$$

PSNR=20*log10(255/sqrt(MSE)); ≡ medida de la calidad de la imagen. A mayor PSNR, mayor calidad de imagen.

$$P.S.N.R = 20 \cdot \log\left(\frac{255}{\sqrt{M.S.E.}}\right) = 20 \cdot \log\left(\frac{255}{\sqrt{\frac{\sum |a_{i.j.} - a1_{i.j.}|^2}{(n1)^2}}}\right) .$$

S.2.2.4. valorescom:

La función que ha sido programada es ésta:

```
function [v,i,j]=valorescom(mra)
% Esta funcion comprime la matriz "mra", eliminando todos sus ceros,
% pero localizando cada valor.
% Sintaxis:  "[v,i,j]=valorescom(mra);".

function [v,i,j]=valorescom(mra)

[i,j,v]=find(mra);

i=uint32(i); j=uint32(j);
```

Ésta es la explicación de las funciones y sentencias que aparecen en el programa.

[i,j,v]=find(mra); ≡ esta función interna del programa "Matlab" localiza dónde está cada valor distinto de cero; y, la citada ubicación queda escrita en los vectores "i", y, "j". En el vector "i" son guardados los valores de las abscisas, y en el "j", los de las ordenadas. Conviene saber que el programa "Matlab", siempre, lee las matrices, desde arriba hacia abajo (lee todas las columnas siguiendo este criterio), y, desde la izquierda hacia la derecha. En el vector "v" se guardan los valores distintos de cero.

i=uint32(i); j=uint32(j); ≡ los índices "i", "j" se convierten en enteros largos.

S.2.2.5. valoresdescom:

La función que ha sido programada es ésta:

```
function [mra]=valoresdescom(i,j,v,ta)
% Esta funcion descomprime los valores comprimidos
% con el programa:  "valorescom".
% Sintaxis:  "[mra]=valoresdescom(i,j,v,ta);".

mra=zeros(ta,ta);
m=length(i);
for (k=1:m)
   mra(i(k),j(k))=v(k);
end
```

Éste es el análisis de la función "valoresdescom":

mra=zeros(ta,ta); ≡ se construye una matriz de ceros de dimensión ta·ta.

m=length(i); ≡ se mide la longitud del vector "i", longitud que es guardada en la variable "m".

```
for (k=1:m)
   mra(i(k),j(k))=v(k);
end
```

} este bucle reconstruye la matriz original.

Por seguridad, han sido añadidas las siguientes líneas de código, que no harán falta si existe un solo elemento significativo (elemento del vector "v") en el extremo de la matriz; lo que implicaría que la matriz completa sería reconstruida.

[mramax]=max(size(mra)); ≡ esta línea de código calcula la máxima longitud (entre la horizontal y la vertical) de la matriz que acaba de construirse.

S.2.2.6. descenderc:

La función que ha sido programada es ésta:

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

% La sintaxis de la funcion descender es esta:
% "[ru,c]=descenderc(a,l,tol,ruido,met);".
% "a" es la imagen a la que le aplicamos el algorimo descendente de
%    multirresolucion (debe de ser "double").
% "l" son los niveles de multirresolucion.
% "tol" es la tolerancia de truncamiento.
% "ruido" es el nivel de ruido que se le suma a la imagen.
% "met" es el metodo que se quiere utilizar: 'lin' (actualmente, es el
%      unico metodo que esta programado).
%   Si se quieren obtener mas datos de las variables de entrada de esta
% funcion, escribir:  "help compresor";  los parametros que comparten ambas
% funciones estan adecuadamente explicados en la ayuda de la mencionada
% funcion.
%   Las variables de salida son estas:
% "c" contiene la version de multirresolucion (mra) de la imagen.
% "ru" contiene la imagen que entra en esta funcion a la que se le ha
%     añadido ruido blanco.
% Tanto "Com" como "rat" estan explicados en:  "compresor".

n=max(size(a));
c=a;

% Se le introduce ruido a la imagen;  el ruido introducido el "ruido
% blanco", que es el mismo ruido que introducen los captadores electronicos
% (camaras fotograficas, y, camaras de video) en las imagenes que captan.

c=c+ruido*randn(n);

ru=c;

% Este programa esta preparado para ser ampliado cuando se decida programar
% mas de un metodo;  el que esta programado es el metodo lineal.
if(met=='lin')
    [c,nzlin]=codifli1c(c,n,l,tol);
    % "nzlin" es el numero de no ceros que existen entre los detalles (entre
    %        los errores).
    na=n*n;   % "na" es la dimension de la matriz (de la imagen).
nb=nzlin+(n/2^l)*(n/2^l);
rat=na/nb
end
```

Com=(1-(1/rat))*100

Éste es el análisis de la función "descenderc":

```
n=max(size(a));
```
≡ se calcula el número de filas y de columnas de la matriz "a" (con la función: "size"), se calcula el máximo valor entre los dos valores que acaban de calcularse (con la función: "max"), y ese máximo valor es guardado en la variable "n".

```
c=a;
```
≡ se guarda la matriz "a" en el fichero "c".

```
c=c+ruido*randn(n);
```
≡ esta orden añade ruido blanco a la imagen. El ruido blanco es el nombre que recibe el ruido electrónico que el captador (la cámara) introduce en la imagen que capta (en la fotografía).

```
ru=c;
```
≡ la matriz de la imagen más el ruido blanco añadido, imagen que está en el fichero "c", es introducida en el fichero "ru".

```
if(met=='lin')
   [c,nzlin]=codifli1c(c,n,l,tol);
   na=n*n;
   nb=nzlin+(n/2^l)*(n/2^l);
   rat=na/nb
end
```
esto es un condicional que comienza con "if ", y termina con "end".
Se lee así: si el método (met) es igual a: 'lin', entonces, ejecutar estas sentencias, si es distinto de 'lin', salir. Cada vez que se ejecuta este condicional, se llama a la función: "codifli1c", que genera dos salidas: "c", y, "nzlin". "nzlin" es el número de elementos distintos de cero (no ceros, método lineal) que existen entre los detalles de la imagen analizada. Hay que considerar que: "$(n/2^l)\cdot(n/2^l)$" es la dimensión de la imagen reducción (el dos está elevado a "l", que es el nivel de multirresolución).

```
Com=(1-(1/rat))*100
```
≡ este valor es el tanto por ciento de compresión de la imagen, tanto por ciento al que se ha llegado con el método utilizado (el de Ami Harten).

S.2.2.7. codifli1c:

La función que ha sido programada es ésta:

```
function [d,nnceros]=codifli1c(a,n,l,tol)


% La sintaxis de la funcion es esta:  "[d,nnceros]=codifli1c(a,n,l,tol);".
%    Estas son las variables de entrada de esta funcion:
% "a" es la matriz (la imagen) a la que se le aplica la multirresolucion.
%     "a" es una matriz "double".
% "n" es el numero, tanto de filas, como de columnas de la matriz "a"
%     (o n=256, o, n=512).
% "l" son los niveles de multirresolucion, "l" es un valor entero.
% "tol" es la tolerancia de truncamiento.
%   Estas son las variables de salida de esta funcion:
% "d" es la imagen comprimida, junto con los errores guardados.
% "nnceros" es el numero de no ceros.
%
%    Si se quiere mas informacion, leer la ayuda de:  "compresor".



% Inicializacion de las variables:
nnceros=0;
d=a;

% Bucle para los niveles de multirresolucion:

for k=1:l

% Se comienza, haciendo la multirresolucion por filas:

for i=1:n
[f1,f2]=lineale1c(d(i,1:n),n);
d(i,1:2:n)=f1;
d(i,2:2:n)=f2;
end
clear f1; clear f2;


% Se sigue, haciendo la multirresolucion por columnas:

for j=1:n
[f1,f2]=lineale1c(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, y se cuentan los elementos no cero (distintos de cero):

b1=d(1:2:n,1:2:n);

b2=d(1:2:n-1,2:2:n);
[I2,J2]=find(abs(b2)<tol);
for i=1:length(I2)
b2(I2(i),J2(i))=0;
end
clear I2; clear J2;
nzb2=nnz(b2);
nnceros=nnceros+nnz(b2);



b3=d(2:2:n,1:2:n);
[I3,J3]=find(abs(b3)<tol);
for i=1:length(I3)
b3(I3(i),J3(i))=0;
end
clear I3; clear J3;
nzb3=nnz(b3);
nnceros=nnceros+nnz(b3);

b4=d(2:2:n,2:2:n);
[I4,J4]=find(abs(b4)<tol);
for i=1:length(I4)
b4(I4(i),J4(i))=0;
end
clear I4; clear J4;
nzb4=nnz(b4);
nnceros=nnceros+nnz(b4);

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

clear b1; clear b2; clear b3; clear b4;
```

```
% Se modifica el valor de "n":

n=n/2;

% Para la media en celda, la tolerancia se divide por dos, despues de cada
% nivel.

tol=tol/2;

% Se cierra el bucle de los niveles de multirresolucion.
end

return
```

Éste es el <u>análisis</u> de la función “codifli1c”:

```
nnceros=0;
```
≡ como se va a sumar valores a la variable “nnceros”, ésta debe de comenzar por un valor que no afecte a la suma, por ese motivo comienza en el cero.

```
d=a;
```
≡ se introduce el valor de la matriz “a” en la variable “d”.

Comienzo del bucle de los niveles de multirresolución, dentro del cual, los errores serán clasificados, al igual que lo serán los valores significativos:

```
for k=1:l
```
≡ para “k” igual a un valor que comienza en el valor uno, se incrementa de uno en uno, y termina en “l” (que es el valor del nivel más fino de resolución de la multirresolución).

```
for i=1:n
    [f1,f2]=lineale1c(d(i,1:n),n);
    d(i,1:2:n)=f1;
    d(i,2:2:n)=f2;
end
```
} En cada fila, de una en una, desde la primera hasta la n-ésima, se llama a la siguiente función: “lineale1c”, que genera dos salidas: “f1” (cuyo valor es introducido en las ubicaciones impares de las filas de la matriz “d”), y, “f2” (cuyo valor es introducido en las ubicaciones pares de las filas de la matriz “d”).

```
clear f1; clear f2;
```
≡ borrar el contenido de los ficheros: "f1", y, "f2".

```
for j=1:n
    [f1,f2]=lineale1c(d(1:n,j)',n);
    d(1:2:n,j)=f1';
    d(2:2:n,j)=f2';
end
```
En cada columna, de una en una, desde la primera hasta la n-ésima, se llama a la siguiente función: "lineale1c", que genera dos salidas: "f1" (cuya traspuesta es introducida en las ubicaciones impares de las columnas de la matriz "d"), y, "f2" (cuya traspuesta es introducida en las ubicaciones pares de las columnas de la matriz "d").

.

```
clear f1; clear f2;
```
≡ borrar el contenido de los ficheros: "f1", y, "f2".

```
b1=d(1:2:n,1:2:n);

b2=d(1:2:n-1,2:2:n);
```
formación de las submatrices "b1", y, "b2", a partir de la matriz original. La submatriz "b1" está formada por los valores significativos, y, la submatriz "b2" está formada por los errores verticales.

```
[I2,J2]=find(abs(b2)<tol);
```
≡ buscar los elementos de "b2" cuyo valor absoluto es inferior a la tolerancia que se le introdujo (tol). "abs(b2)" convierte, en positivos, todos los valores de la submatriz "b2". La función "find" genera dos salidas, que indican la posición, dentro de la submatriz "b2", que ocupan los diferentes elementos de la submatriz "b2" que cumplen la condición: "abs(b2)<tol"; las dos salidas son éstas: "I2" contiene el valor de las filas, y, "J2" contiene el valor de las columnas.

```
for i=1:length(I2)
    b2(I2(i),J2(i))=0;
end
```
Se ponen a cero los elementos de la submatriz "b2" que ocupan los índices (I2,J2). Esta aproximación, que está determinada por la tolerancia (tol), permitirá reducir el espacio que ocupa la imagen original.

```
clear I2; clear J2;
```
≡ borrar el contenido de los ficheros: "I2", y, "J2".

```
nzb2=nnz(b2);
```
≡ número de elementos no cero (distintos de cero) que contiene la submatriz "b2".

```
nnceros=nnceros+nnz(b2);   ≡ actualización de la variable:  "nnceros".

b3=d(2:2:n,1:2:n);         ≡ formación de las submatriz "b3", a partir de la
                             matriz original.  La submatriz "b3" está formada por
                             los errores horizontales.

[I3,J3]=find(abs(b3)<tol); ≡ obtención de las posiciones cuyo valor absoluto es
                             inferior a la tolerancia prescrita (tol).

for i=1:length(I3)
    b3(I3(i),J3(i))=0;     } Se ponen a cero los elementos de la submatriz "b3"
end                          que ocupan los índices (I3,J3).  Esta aproximación,
                             que está determinada por la tolerancia (tol), permitirá
                             reducir el espacio que ocupa la imagen original.

clear I3; clear J3;        ≡ borrar el contenido de los ficheros:  "I3", y, "J3".

nzb3=nnz(b3);              ≡ número de elementos no cero (distintos de cero) que
                             contiene la submatriz "b3".

nnceros=nnceros+nnz(b3);   ≡ actualización de la variable:  "nnceros".

b4=d(2:2:n,2:2:n);         ≡ formación de las submatriz "b4", a partir de la
                             matriz original.  La submatriz "b4"está formada por
                             los errores mixtos.

[I4,J4]=find(abs(b4)<tol); ≡ obtención de las posiciones cuyo valor absoluto es
                             inferior a la tolerancia prescrita (tol).

for i=1:length(I4)
    b4(I4(i),J4(i))=0;     } Se ponen a cero los elementos de la submatriz "b4"
end                          que ocupan los índices (I4,J4).  Esta aproximación,
                             que está determinada por la tolerancia (tol), permitirá
                             reducir el espacio que ocupa la imagen original.

clear I4; clear J4;        ≡ borrar el contenido de los ficheros:  "I4", y, "J4".

nzb4=nnz(b4);              ≡ número de elementos no cero (distintos de cero) que
                             contiene la submatriz "b4".

nnceros=nnceros+nnz(b4);   ≡ actualización de la variable:  "nnceros".
d(1:n,1:n)=[b1,b2;b3,b4];  ≡ redistribución de la matriz original.  Esta línea de
                             código es la que junta las cuatro submatrices en el
                             orden en el que se especificó en el apartado:
                             "S.2.1.".
```

```
    clear b1; clear b2; clear b3; clear b4;   ≡ borrado de las submatrices auxiliares:
                                                "b1", "b2", "b3", y, "b4".
    n=n/2;                                    ≡ actualización del tamaño de la
                                                submatriz que contiene los valores
                                                significativos.
    tol=tol/2;                                ≡ actualización de la tolerancia de la
                                                submatriz que contiene los valores
                                                significativos.
end                                           ≡ final de este bucle de multirresolución.

return                                        ≡ esta orden sirve para que la función
                                                (codifli1c) regrese a la siguiente línea del
                                                código de la función que la llamó.
```

S.2.2.8. lineale1c:

La función que ha sido programada es ésta:

```
function [f1,f2]=lineale1c(f,n)

% Esta es la sintaxis de la funcion:  "[f1,f2]=lineale1c(f,n);".
% Esta funcion calcula los valores significativos de la escala inferior
% "k-1" y los errores cometidos al pasar de la escala "k-1" a la escala
% "k".
%    Estas son las entradas:
% "f" es un vector.
% "n" es la logitud del vector.
%    Estas son las salidas:
% "f1" son los valores significativos de la escala inferior.
% "f2" son los errores al pasar de la escala "k-1" a la escala "k".



f1=(f(1:2:n)+f(2:2:n))/2;    %Calculo de los valores significativos de la escala inferior.



nk1=n/2;


% Calculo del primer detalle (error):

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

% Calculo de los detalles (errores) intermedios:

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

% Calculo del ultimo detalle (error):

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

Éste es el <u>análisis</u> de la función "lineale1c":

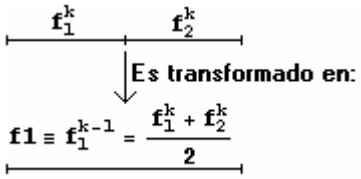
Se ha comenzado por considerar que todos los puntos significativos (los que pertenecen a la imagen que hay que comprimir) están equidistantes entre sí. La anterior consideración reduce, notablemente, los cálculos.

Como puede apreciarse en la función "lineale1c", los coeficientes con los que se calcula la predicción en los extremos del vector no son simétricos; mientras que sí lo son los que sí pertenecen al interior del vector.

A continuación se detallan los comandos utilizados.

f1=(f(1:2:n)+f(2:2:n))/2; ≡ decimación. Cálculo de los valores significativos de la escala inferior. Esta línea de código, también, puede escribirse así:

$$\mathbf{f1} \equiv f_1^{k-1} = \frac{f_1^k + f_2^k}{2} \equiv \frac{\text{valores impares} + \text{valores pares}}{2} \quad .$$



nk1=n/2; ≡ se impone que la escala inferior (k-1) tenga la mitad de puntos que la escala superior (k).

q(1)=(11/8)*f1(1)-(4/8)*f1(2)+(1/8)*f1(3); ≡ expresión del operador predicción para el extremo izquierdo usando las máscaras correspondientes al esquema definido en la sección S.1.

f2(1)=f(1)-q(1); ≡ Cálculo del primer error. Valor exacto menos predicción.

q(2:nk1-1)=(1/8)*f1(1:nk1-2)+f1(2:nk-1)-(1/8)*f1(3:nk-1);≡ expresión del operador predicción para los intervalos centrales usando las máscaras correspondientes al esquema definido en la sección S.1.

f2(2:nk1-1)=f(3:2:n-2)-q(2:nk1-1); ≡ Cálculo de los errores intermedios. Valores exactos menos predicciones.

q(nk1)=-(1/8)*f1(nk1-2)+(4/8)*f1(nk1-1)+(5/8)*f1(nk1);≡ expresión del operador predicción para el extremo derecho usando las máscaras correspondientes al esquema definido en la sección S.1.

f2(nk1)=f(n-1)-q(nk1); ≡ Cálculo del último error. Valor exacto menos predicción.

S.2.2.9. ascenderc:

La función que ha sido programada es ésta:

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

% La sintaxis de la funcion es esta:  "c=ascenderc(a,l,met);".
% "a" es la matriz a la que se le aplica el algoritmo ascendente de
%     multirresolucion.
% "l" es el nivel de multirresolucion.   "l" es un numero entero.
% "met" metodo que se quiere utilizar:  'lin'(actualmente, este es el unico
%       metodo que esta programado).   "lin" es el metodo lineal.


%    Esta es una funcion intermedia que prepara al conjunto de funciones
% usadas en este proyecto para que el usuario de las mismas decida que
% metodo quiere que se le aplique a la imagen que va a ser comprimida.
% Esta funcion selectora de funciones que aplican diversos metodos de
% compresion estara justificada en cuanto sea programada una sola funcion
% mas que aplique un metodo distinto del lineal.

n=max(size(a));
```

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

Éste es el análisis de la función "ascenderc":

```
n=max(size(a));
```
≡ "n" es el máximo valor de la dimensión de la matriz de la entrada (matriz "a").

```
if(met=='lin')
    c=decodiflic(a,n,l);
end
```
} Parte del código de esta función que activa la función "decodiflic" si el método seleccionado es el lineal ('lin').

S.2.2.10. decodiflic:

La función que ha sido programada es ésta:

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


% La sintaxis de esta funcion es la que sigue:  "c=decodiflic(a,n,l);".
% "a" es la matriz a la que se le aplica la multirresolucion.
% "n" es el numero, tanto de filas, como de columnas de la matriz "a"
%     (o n=256, o n=512).
% "l" son los niveles de multirresolucion.   "l" es un numero entero.

c=a;
nl=round(n/(2^l));
% Inicializacion de las variables.

for k=l:-1:1
% Bucle para los niveles de multiresolucion.

% Se Reordena la matriz, haciendo el proceso inverso que en la codificacion.

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

```
% Se hace el algoritmo de multirresolucion inverso para las columnas.
for j=1:nb
c(1:nb,j)=linealde1c(c(1:nb,j)',nb)';
end

% Se hace el algoritmo de multirresolucion inverso para las filas.

for i=1:nb
c(i,1:nb)=linealde1c(c(i,1:nb),nb);
end


% Se calcula "nl" para subir al siguiente nivel.

nl=2*nl;

end
```

El análisis de la función “decodiflic” es éste:

c=a;	≡ la matriz “a” (que es “double”) es introducida en la variable “c”.
nl=round(n/(2^l));	≡ el número entero más cercano a “n” dividido por dos elevado al número de niveles de multirresolución es guardado en la variable “nl” (nivel de multirresolución más bajo).

Las operaciones realizadas por la función “decodiflic” son las inversas de las realizadas por la función “codifli1c”. A continuación, comienza el bucle que deshará el trabajo realizado por la función “codifli1c”.

for k=l:-1:1	≡ comenzando el valor de “k” siendo el número de niveles de multirresolución, y, disminuyendo en cada ciclo una unidad. Se saldrá del bucle cuando “k” valga uno.
nb=2*nl;	≡ “nb” es dos veces el tamaño anterior. Se sube un nivel de multirresolución.
b=zeros(nb);	≡ en la variable “b”, es guardada una matriz formada solamente por ceros; y, cuya dimensión es nb·nb.

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

} son seleccionadas diversas partes de la matriz "c", y son redistribuidas en la matriz "b".

```
clear b;
```

≡ se borra el contenido de la matriz "b".

```
for j=1:nb
   c(1:nb,j)=linealde1c(c(1:nb,j)',nb)';
end
```

} es llamada la función: "linealde1c" en cada columna. La función "linealde1c" deshace el trabajo de la función: "lineale1c".

```
for i=1:nb
   c(i,1:nb)=linealde1c(c(i,1:nb),nb);
end
```

} es llamada la función "linealde1c" en cada fila.

```
nl=2*nl;
```

≡ al multiplicar el nivel de multirresolución por dos, se sube un nivel en la escala de multirresolución (se cambia del nivel "k-1" al nivel "k").

```
end
```

≡ esta orden finaliza el bucle que ha deshecho el trabajo realizado por la función "codifli1c".

S.2.2.11. linealde1c:

La <u>función</u> que ha sido programada es ésta:

```
function f=linealde1c(v,n)

% La sintaxis de la funcion es esta:  "f=linealde1c(v,n);".
% "v" es el vector que contiene los valores significativos de la escala
%     inferior(k-1), y los detalles para subir a la escala superior(k).
% "n" es la dimension del vector "v".

f=zeros(size(v));


nk1=n/2;
```

```
% Prediccion de los valores impares de "f":

% Prediccion del primer valor:
q(1)=(11/8)*v(1)-(4/8)*v(3)+(1/8)*v(5);
f(1)=v(2)+q(1);

% Prediccion de los valores intermedios:
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);

% Prediccion del ultimo valor:
q(nk1)=-(1/8)*v(n-5)+(4/8)*v(n-3)+(5/8)*v(n-1);
f(n-1)=v(n)+q(nk1);


% Prediccion de los valores pares de "f":

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

Éste es el <u>análisis</u> de la función “linealde1c”:

Esta función deshace el trabajo realizado por la función llamada así: “lineale1c”.
Las máscaras utilizadas en el operador predicción son las mismas en ambas funciones.

```
f=zeros(size(v));                          ≡ un vector formado, sólo, por
                                             ceros (cuya dimensión es la
                                             misma que la del vector “v”) es
                                             guardado en la variable “f ”.

nk1=n/2;                                   ≡ se le asigna el valor n/2 a la varia-
                                              ble “nk1”.
q(1)=(11/8)*v(1)-(4/8)*v(3)+(1/8)*v(5);    ≡ predicción de los valores impares
                                             usando las máscaras que se
                                             deducen de la sección S.1.
f(1)=v(2)+q(1);                            ≡ sumamos predicción más error.

q(2:nk1-1)=(1/8)*v(1:2:n-5)+v(3:2:n-3)-(1/8)*v(5:2:n-1); ≡ predicción de los valores
                                                           impares usando las
                                                           máscaras que se deducen de
                                                           la sección S.1.

f(3:2:n-3)=v(4:2:n-2)+q(2:nk1-1);          ≡ sumamos predicción más error.
```

q(nk1)=-(1/8)*v(n-5)+(4/8)*v(n-3)+(5/8)*v(n-1);	≡ predicción de los valores impares usando las máscaras que se deducen de la sección S.1.
f(n-1)=v(n)+q(nk1);	≡ sumamos predicción más error.
f(2:2:n)=2*v(1:2:n)-f(1:2:n);	≡ definición de los valores pares de la función "f ", valores que comienzan con el valor dos, y, terminan con el valor: "n". Se calculan a partir de los impares.

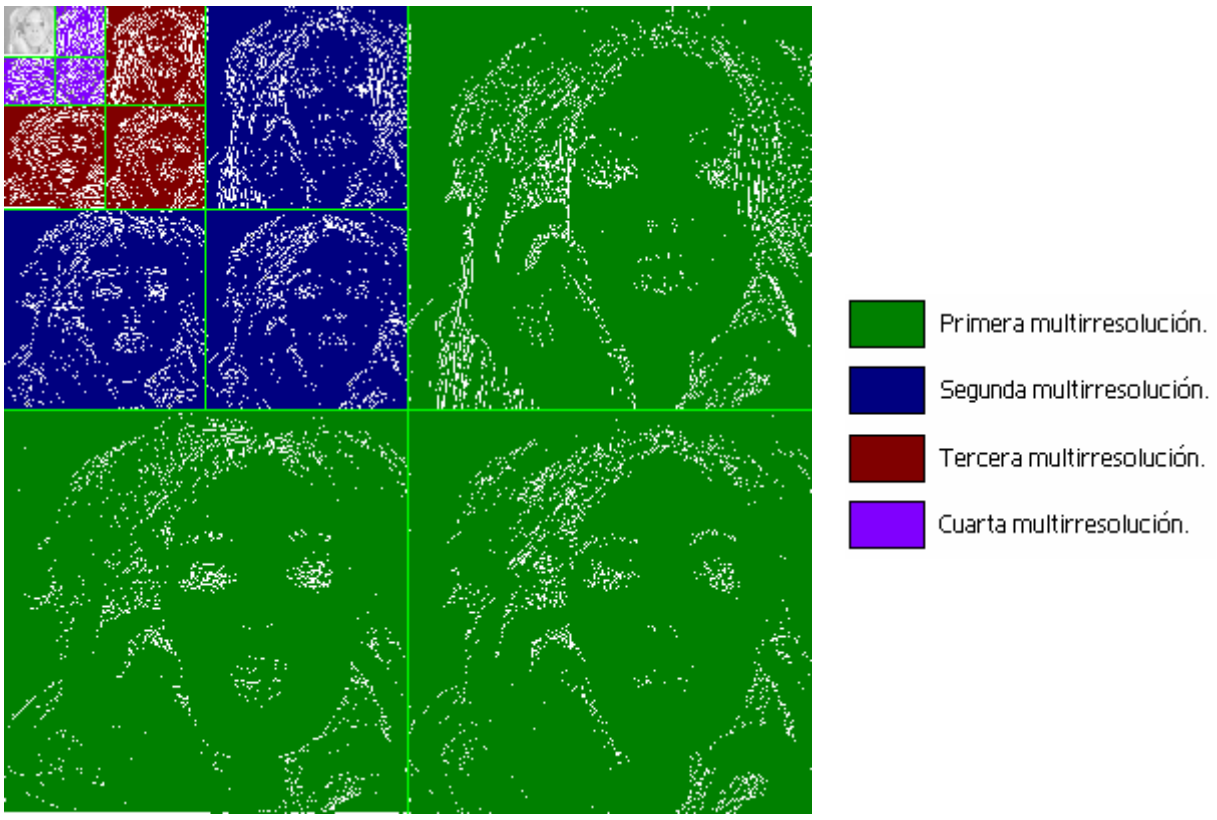
S.3. Sección tres. Tablas:

Estos datos han sido obtenidos con un Intel Pentium IV, CPU 2,80 GHz, 504 Mbytes de RAM, y con un disco duro de 74,5 Gbytes.

En primer lugar trabajamos con la imagen: Tiffany5.pgm .

Para evitar la necesidad de leer el apartado S.2.2. de la sección dos de este proyecto (apartado en el que se explican pormenorizadamente los programas informáticos que han sido desarrollados para este proyecto, y que son los que han sido utilizados para obtener las tablas y gráficas que son mostradas, tanto en esta sección como en la siguiente sección), se pasa a explicar qué significa cada término de las siguientes gráficas.

"**l nivel**" es el nivel de multirresolución que se quiere realizar.



La imagen que está en escala de grises es la imagen original reducida.
Todos los puntos blancos de las multirresoluciones son los errores necesarios para devolver la imagen original reducida a su tamaño original (cualquier color distinto del blanco es un cero –información eliminada cuando la imagen original sea comprimida–).
La gráfica ha sido realizada con cuatro niveles de multirresolución.

"**tol**" es la tolerancia de truncación, es decir, es el valor hasta el cual todo valor inferior será igualado a cero. Cuanto mayor es este valor, más comprimida queda la imagen original, y peor es la reconstrucción de la imagen original.

"**ruido**" es el nivel de ruido blanco. "0" indica que no hay ruido blanco. Cuanto mayor sea este número, peor es la compresión, ya que se guardarán más valores como errores.

"**rat**" es el numero total de elementos de la imagen, dividido por los elementos de la imagen que han sido guardados en la compresión de la misma. Éste es el ratio de compresión de la imagen. Es una medida utilizada en diversos estudios.

"**Com**" es el tanto por ciento que es comprimida la imagen.

"**PSNR**" es una medida de calidad de reconstrucción realizada de la imagen utilizando la versión comprimida. Cuanto mayor es el valor de "PSNR", mejor es la calidad de la imagen que ha sido reconstruida. Se considera un valor aceptable de esta medida un valor mayor que 25 aunque depende mucho de la imagen comprimida.

$$P.S.N.R = 20 \cdot \log\left(\frac{255}{\sqrt{\frac{\sum(|\text{Imagen comprimida y, después, descomprimida}_{i,j} - \text{Imagen original}_{i,j}|)^2}{(\text{Máximo valor, de entre las filas y las columnas, de la imagen original})^2}}}\right).$$

"**v**" es el vector que contiene los valores significativos, tanto de la imagen reducida, como de los errores guardados ("v" contiene la imagen comprimida).

" **i** ", " **j** " son los vectores que contienen la ubicación de los valores del vector "v" (son imprescindibles en la descompresión de la imagen).

"**ta**" es el tamaño de la imagen (ta = 512, significa: imagen = 512·512. ta = 256, significa: imagen = 256·256).

"**met**" es el método de multirresolución. En este proyecto, únicamente, ha sido desarrollado un método ('lin', que quiere decir esto: lineal), pero algunas funciones están preparadas para que sea fácil incorporar nuevos métodos no necesariamente lineales.

"**tamaño**" es la dimensión de la matriz. Cuando uno de los dos números que indican el tamaño es un uno, el otro número indica, exactamente, cuántos datos han sido guardados.

"**Bytes**" son los bytes que ocupa la matriz de la columna en la que está esta palabra.

Las matrices que, tanto tienen el mismo tamaño, como ocupan el mismo espacio, han sido agrupadas en la misma columna. En esa columna, hay dos divisiones, en la superior está el tamaño que ocupa cada una de las matrices de esa columna, mientras que en la inferior, está el número de bytes que ocupa cada una de las matrices de esa columna.

"$\mathbf{t_{compresión}}$" es el tiempo que el compresor tardó en comprimir la imagen original.

En esta tabla se muestra la **relación existente entre** el ratio de compresión (**rat**), el tanto por ciento de compresión (**Com**), la medida **PSNR**, la dimensión (**tamaño**) de la imagen comprimida, los **bytes** que ocupa la imagen comprimida, y el **tiempo de compresión.**

El tamaño de la imagen original es éste: 512·512.
El tamaño de la imagen reconstruida, siempre es igual al de la imagen original, es decir, 512·512.

Tabla 1:

Parámetros de entrada						Imagen comprimida		Parámetros de entrada		
l nivel	**tol**	**ruido**	**rat**	**Com**	**PSNR**	**Long int i,j tamaño**	**Double v tamaño**	**Int l, ta tamaño**	**Char met tamaño**	**$t_{compresión}$ (segundos)**
						Bytes	**Bytes**	**Bytes**	**Bytes**	
3	3	0	6.3376	84.2213	41.7372	41363x1 165452	41363x1 330904	1x1 l=1, ta=2	1 x 3 6	0.5000
3	5	0	10.4203	90.4034	38.9455	25157x1 100628	25157x1 201256	1x1 l=1, ta=2	1 x 3 6	0.5000
3	10	0	19.3693	94.8372	35.6106	13534x1 54136	13534x1 108272	1x1 l=1, ta=2	1 x 3 6	0.5000
3	15	0	26.8370	96.2738	33.7235	9768x1 39072	9768x1 78144	1x1 l=1, ta=2	1 x 3 6	0.5150
3	20	0	33.8250	97.0436	32.3340	7750x1 31000	7750x1 62000	1x1 l=1, ta=2	1 x 3 6	0.5000
3	25	0	42.7153	97.6589	31.0256	6137x1 24548	6137x1 49096	1x1 l=1, ta=2	1 x 3 6	0.5000
3	3	10	1.6568	39.6431	41.3598	158222x1 632888	158222x1 1265776	1x1 l=1, ta=2	1 x 3 6	0.5470
3	5	10	2.5350	60.5518	35.4367	103411x1 413644	103411x1 827288	1x1 l=1, ta=2	1 x 3 6	0.5320
3	10	10	8.8715	88.7280	29.3421	29549x1 118196	29549x1 236392	1x1 l=1, ta=2	1 x 3 6	0.5150
3	15	10	21.7836	95.4094	27.6175	12034x1 48136	12034x1 96272	1x1 l=1, ta=2	1 x 3 6	0.5000

Parámetros de entrada						Imagen comprimida		Parámetros de entrada		
l nivel	**tol**	**ruido**	**rat**	**Com**	**PSNR**	**Long int i,j tamaño**	**Double v tamaño**	**Int l, ta tamaño**	**Char met tamaño**	**$t_{compresión}$**
						Bytes	**Bytes**	**Bytes**	**Bytes**	
3	20	10	31.9610	96.8712	27.0023	8202x1 32808	8202x1 65616	1x1 l=1, ta=2	1 x 3 6	0.5000
3	25	10	39.9793	97.4987	26.5978	6557x1 26228	6557x1 52456	1x1 l=1, ta=2	1 x 3 6	0.5150
4	3	0	6.4644	84.5306	41.6968	40552x1 162208	40552x1 324416	1x1 l=1, ta=2	1 x 3 6	0.5310
4	5	0	10.9404	90.8596	38.8637	23961x1 95844	23961x1 191688	1x1 l=1, ta=2	1 x 3 6	0.5160
4	10	0	22.1705	95.4895	35.4421	11824x1 47296	11824x1 94592	1x1 l=1, ta=2	1 x 3 6	0.5310
4	15	0	33.8338	97.0444	33.4692	7748x1 30992	7748x1 61984	1x1 l=1, ta=2	1 x 3 6	0.5310
4	20	0	47.4899	97.8943	32.0063	5520x1 22080	5520x1 44160	1x1 l=1, ta=2	1 x 3 6	0.5160
4	25	0	69.9797	98.5710	30.6502	3746x1 14984	3746x1 29968	1x1 l=1, ta=2	1 x 3 6	0.5310
4	3	10	1.6579	39.6835	41.3600	158116x1 632464	158116x1 1264928	1x1 l=1, ta=2	1 x 3 6	0.5470
4	5	10	2.5642	61.0008	35.4138	102234x1 408936	102234x1 817872	1x1 l=1, ta=2	1 x 3 6	0.5320
4	10	10	9.3306	89.2826	29.3001	28095x1 112380	28095x1 224760	1x1 l=1, ta=2	1 x 3 6	0.5470
4	15	10	26.1230	96.1720	27.5512	10035x1 40140	10035x1 80280	1x1 l=1, ta=2	1 x 3 6	0.5470
4	20	10	43.6252	97.7077	26.8948	6009x1 24036	6009x1 48072	1x1 l=1, ta=2	1 x 3 6	0.5160
4	25	10	63.0002	98.4127	26.4124	4161x1 16644	4161x1 33288	1x1 l=1, ta=2	1 x 3 6	0.5310

Conforme aumenta el ratio de compresión (rat), aumenta el tanto por ciento que es comprimida la imagen (Com).

Conforme aumenta el ratio de compresión (rat), disminuye el valor de la medida de la calidad de la reconstrucción (PSNR).

Cuanto mayor es el tanto por ciento de compresión de la imagen (Com), menor es la cantidad de bytes que ocupa la imagen comprimida.

Tanto el tamaño como los bytes que ocupan "l", "ta", y, "met", son valores independientes de los parámetros de entrada.

Tabla 2:

Comparación de **bytes**:

Esta tabla ha sido obtenida también con la imagen Tiffany5.pgm .

l nivel	**tol**	**ruido**	**Uint8 a1 Bytes. Imagen original**	**i,j,v,l,met,ta Bytes. Imagen comprimida, junto con los datos necesarios para reconstruirla**	**Tanto por ciento de compresión**
3	3	0	262144	496364	84.2213
	5			301892	90.4034
	10			162416	94.8372
	15			117224	96.2738
	20			93008	97.0436
	25			73652	97.6589
3	3	10	262144	1898672	39.6431
	5			1240940	60.5518
	10			354596	88.7280
	15			144416	95.4094
	20			98432	96.8712
	25			78692	97.4987
4	3	0	262144	486632	84.5306
	5			287540	90.8596
	10			141896	95.4895
	15			92984	97.0444
	20			66248	97.8943
	25			44960	98.5710
4	3	10	262144	1897400	39.6835
	5			1226816	61.0008
	10			337148	89.2826
	15			120428	96.1720
	20			72116	97.7077
	25			49940	98.4127

Tanto la imagen reconstruida como la imagen original ocupan siempre la misma cantidad de bytes.

El efecto del ruido implica una tasa de compresión menor.

Tabla 3:

Variación del **tiempo de compresión y** de los **bytes** que ocupa la **imagen comprimida** cuando cambia el **ruido**:

Esta tabla ha sido obtenida con esta imagen: Tiffany5.pgm .

l nivel	**tol**	**ruido**	**i,j Bytes**	**v Bytes**	**l, ta Bytes**	**met Bytes**	**$t_{compresión}$**	**Imagen comprimida Bytes**	**Tanto por ciento de compresión**
4	3	0	162208	324416	l=1,ta=2	6	0.5160	486631	84.5306
4	3	1	167020	334040	l=1,ta=2	6	0.5160	501067	84.0717
4	3	2	190288	380576	l=1,ta=2	6	0.5150	570871	81.8527
4	3	3	242948	485896	l=1,ta=2	6	0.5150	728851	76.8307
4	3	4	311996	623992	l=1,ta=2	6	0.5310	935995	70.2457
4	3	5	383728	767456	l=1,ta=2	6	0.5470	1151191	63.4048
4	3	6	448524	897048	l=1,ta=2	6	0.5310	1345579	57.2254
4	3	7	506196	1012392	l=1,ta=2	6	0.5310	1518595	51.7254
4	3	8	552224	1104448	l=1,ta=2	6	0.5470	1656679	47.3358
4	3	9	596632	1193264	l=1,ta=2	6	0.5470	1789903	43.1007
4	3	10	634476	1268952	l=1,ta=2	6	0.5310	1903435	39.4917

Conforme aumenta el ruido en la imagen original, aumenta la cantidad de bytes que ocupa la imagen comprimida. El tiempo requerido para la compresión no depende del ruido.

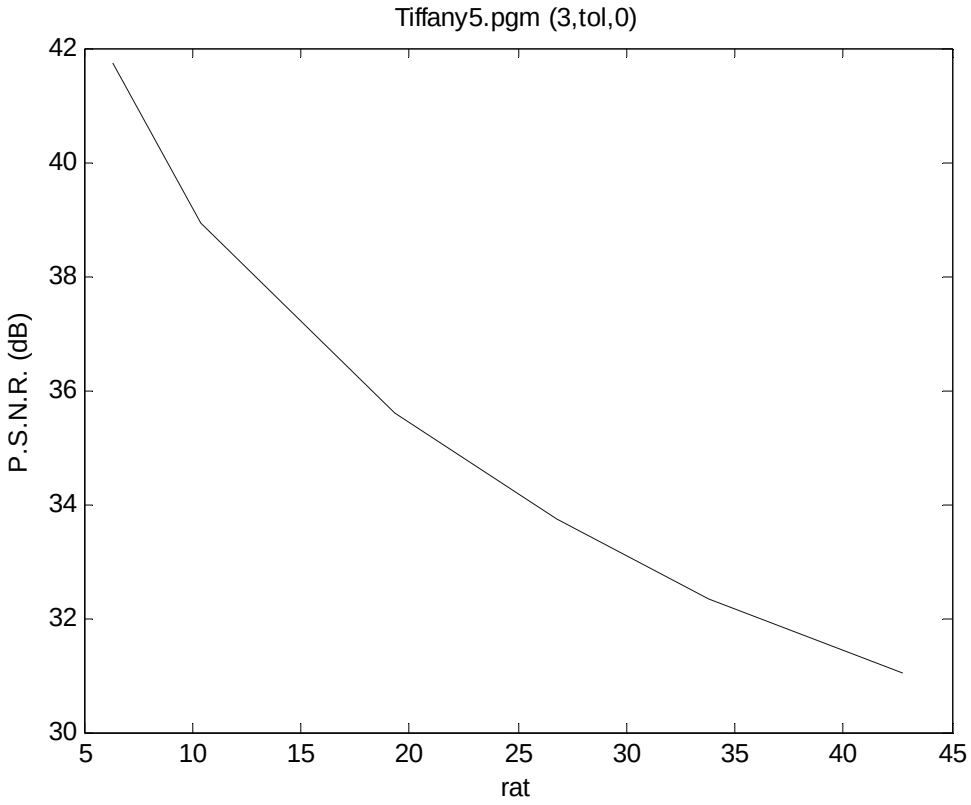
S.4. Sección cuatro. Gráficas:

Esta sección está directamente relacionada con la anterior sección, ya que estas gráficas han sido obtenidas de las tablas de la anterior sección.

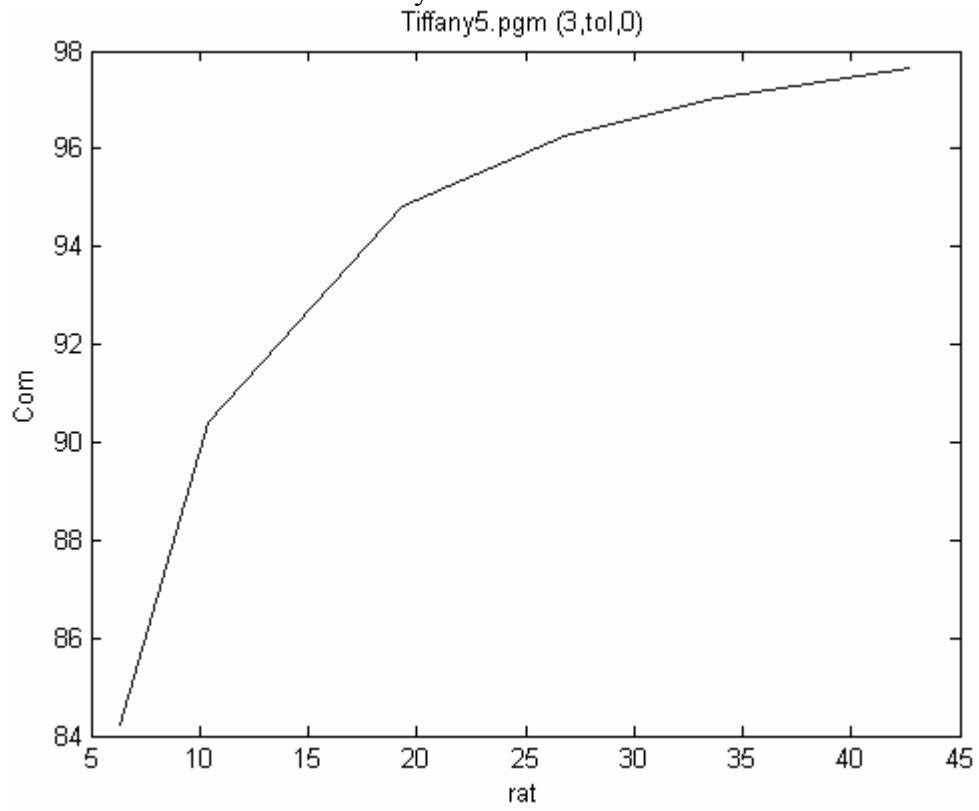
Si se accede directamente a esta sección, una ayuda de lo que significan las distintas variables que se comparan en estas gráficas está en la anterior sección.

Gráficas obtenidas de la tabla uno:

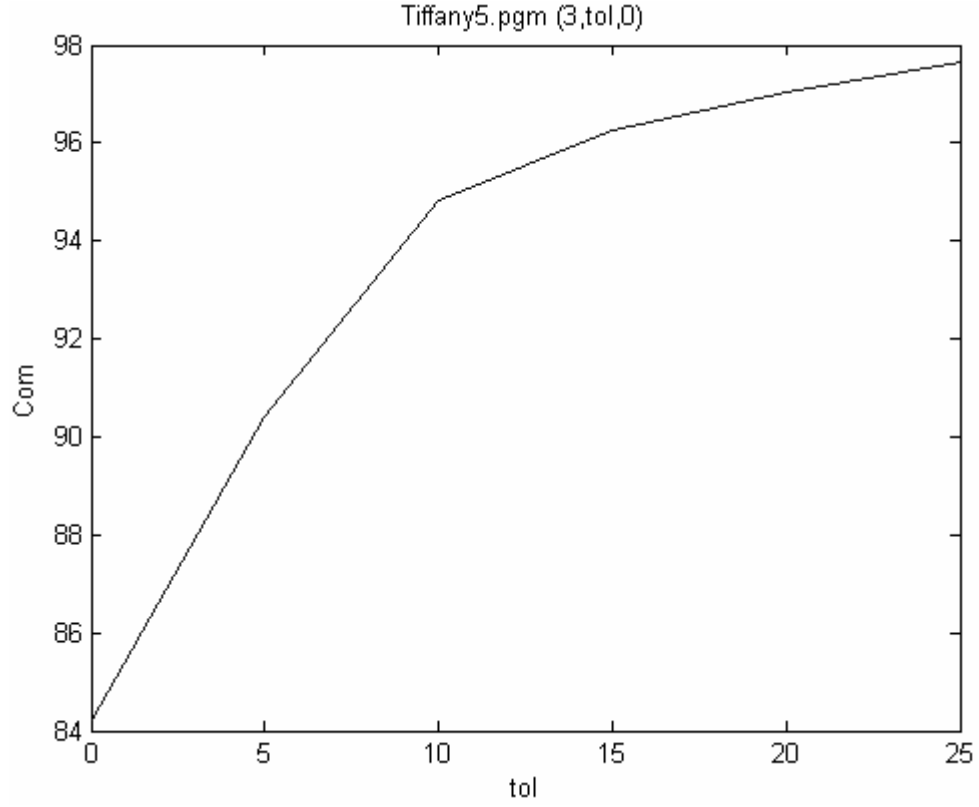
Gráfica uno. Relación entre "**rat**" y "**PSNR**":



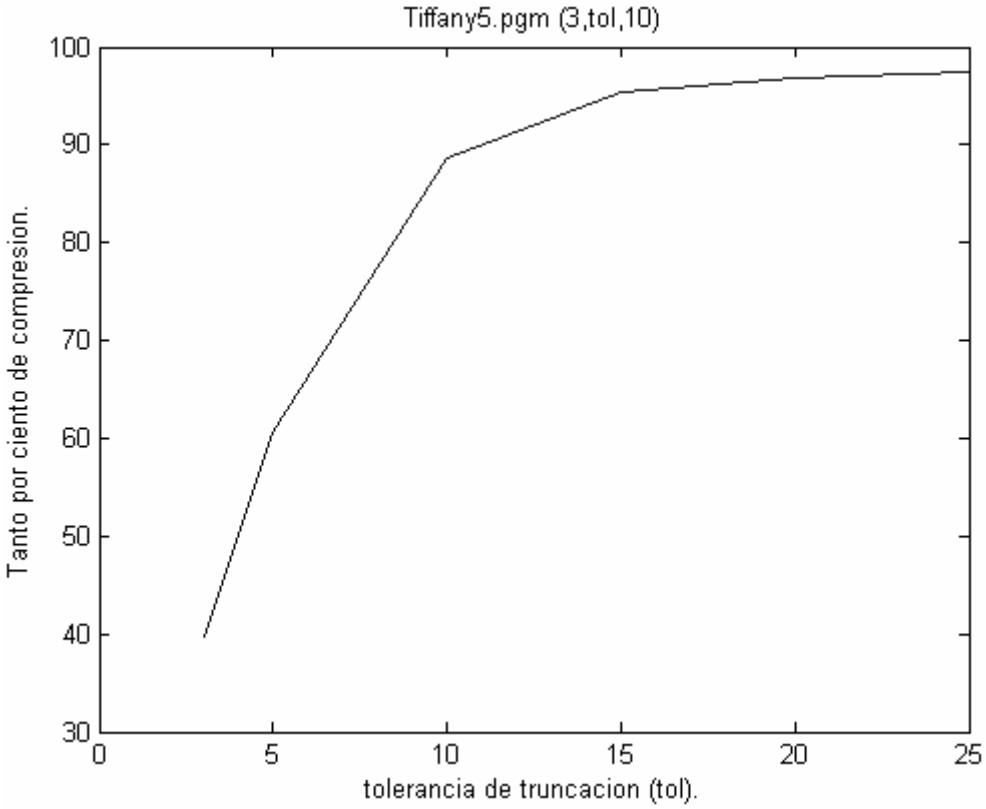
Gráfica dos. Relación entre "**rat**" y "**Com**":



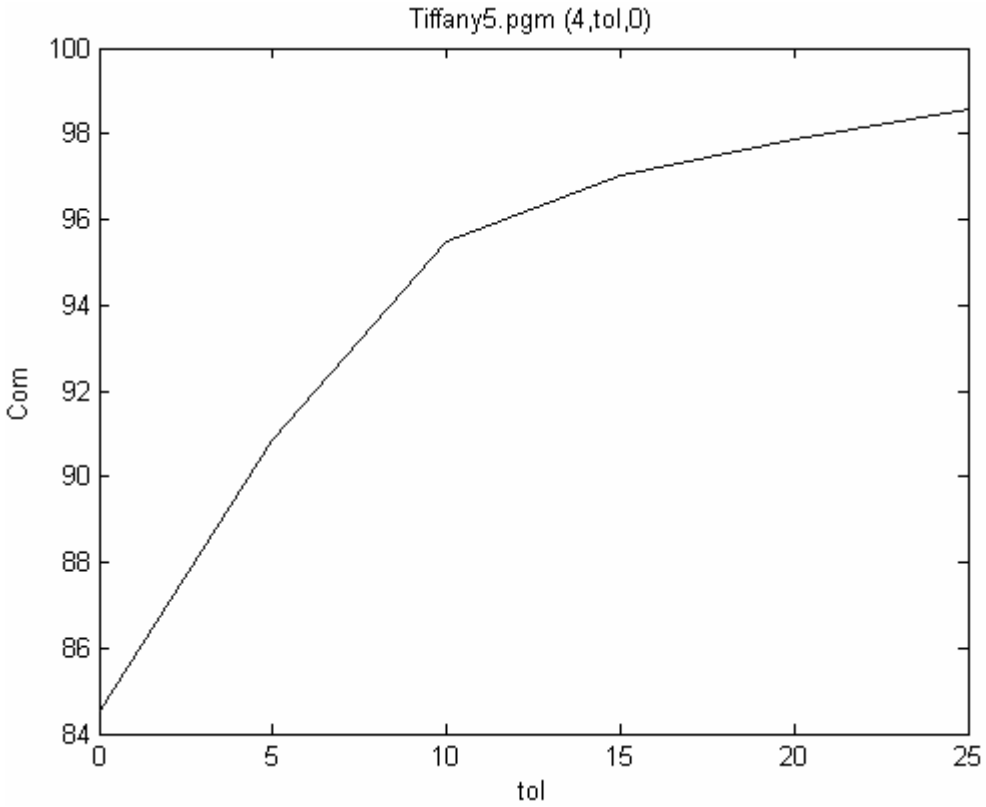
Gráfica tres. Relación entre "**Com**" (l = 3, ruido = 0) y "**tol**":



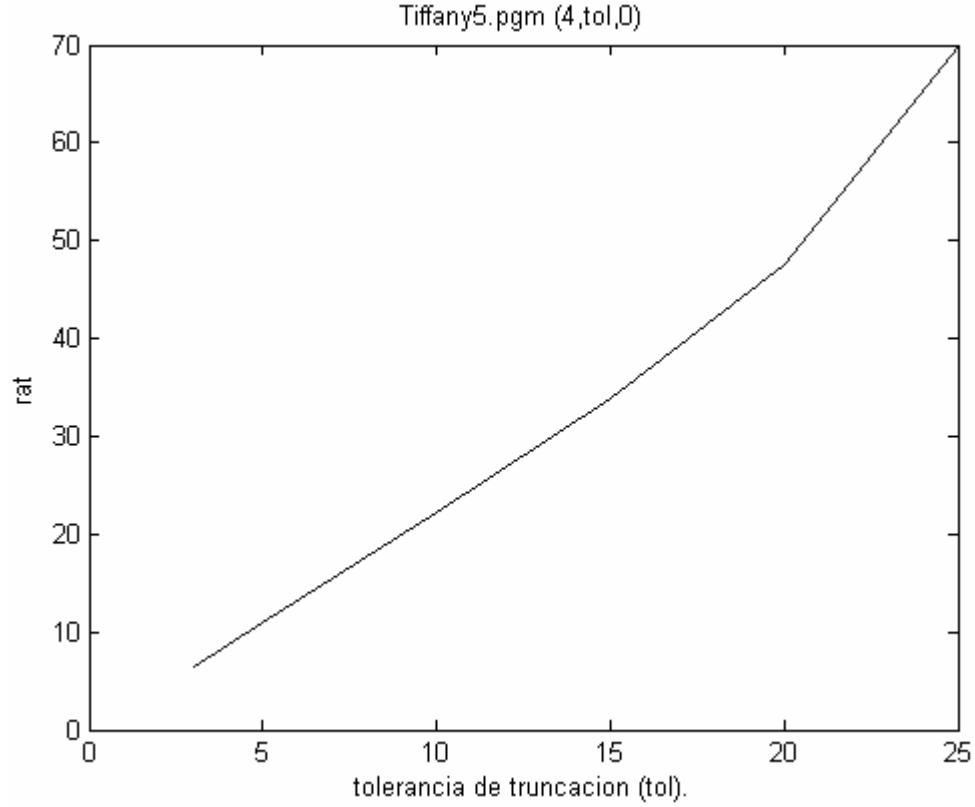
Gráfica cuatro. Relación entre "**Com**" (l = 3, ruido = 10) y "**tol**":



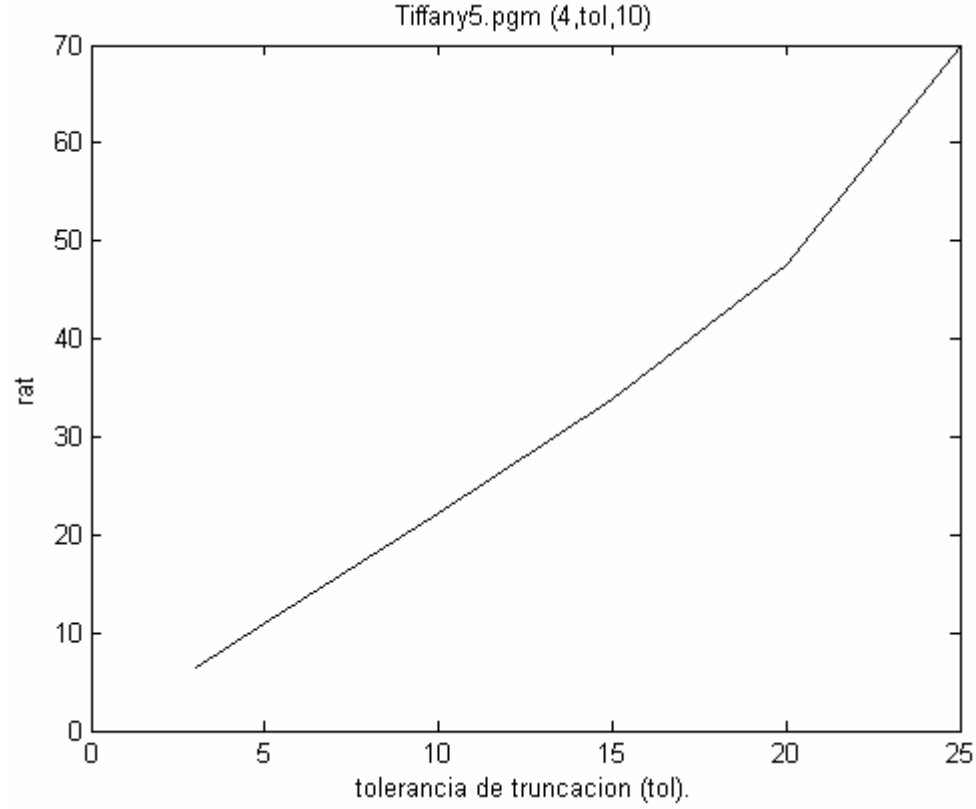
Gráfica cinco. Relación entre "**Com**" (l = 4, ruido = 0) y "**tol**":



Gráfica seis. Relación entre "**rat**" (l = 4, ruido = 0) y "**tol**":

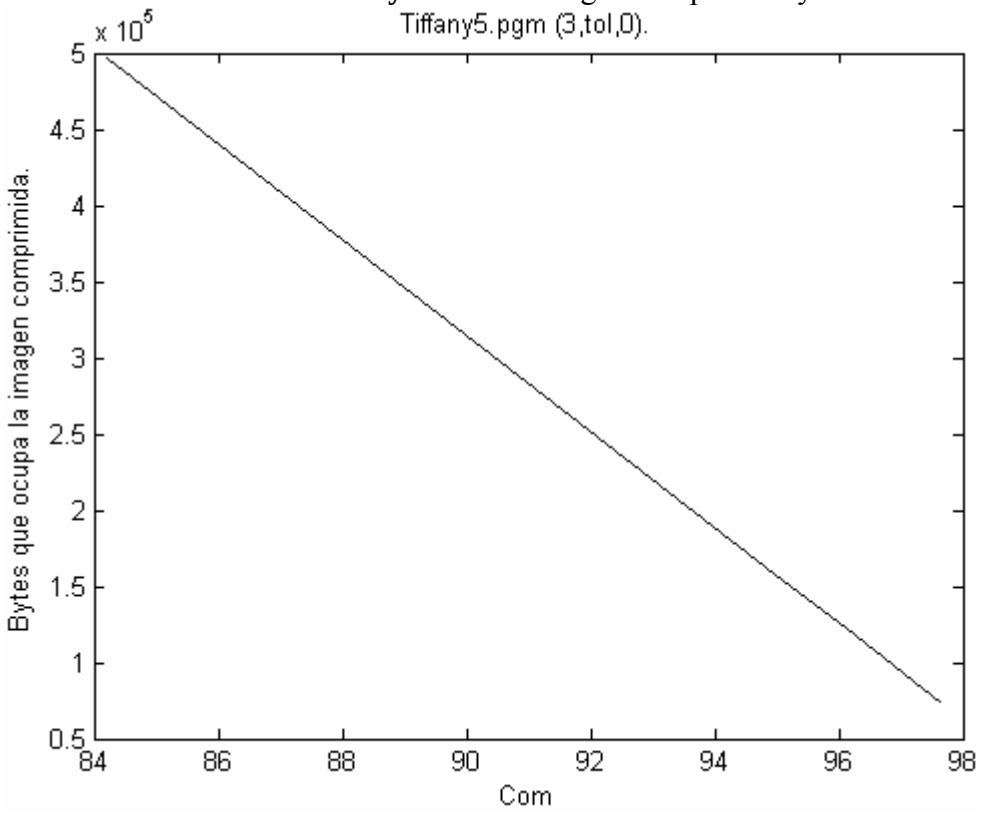


Gráfica siete. Relación entre el **ratio** y la **tolerancia**:

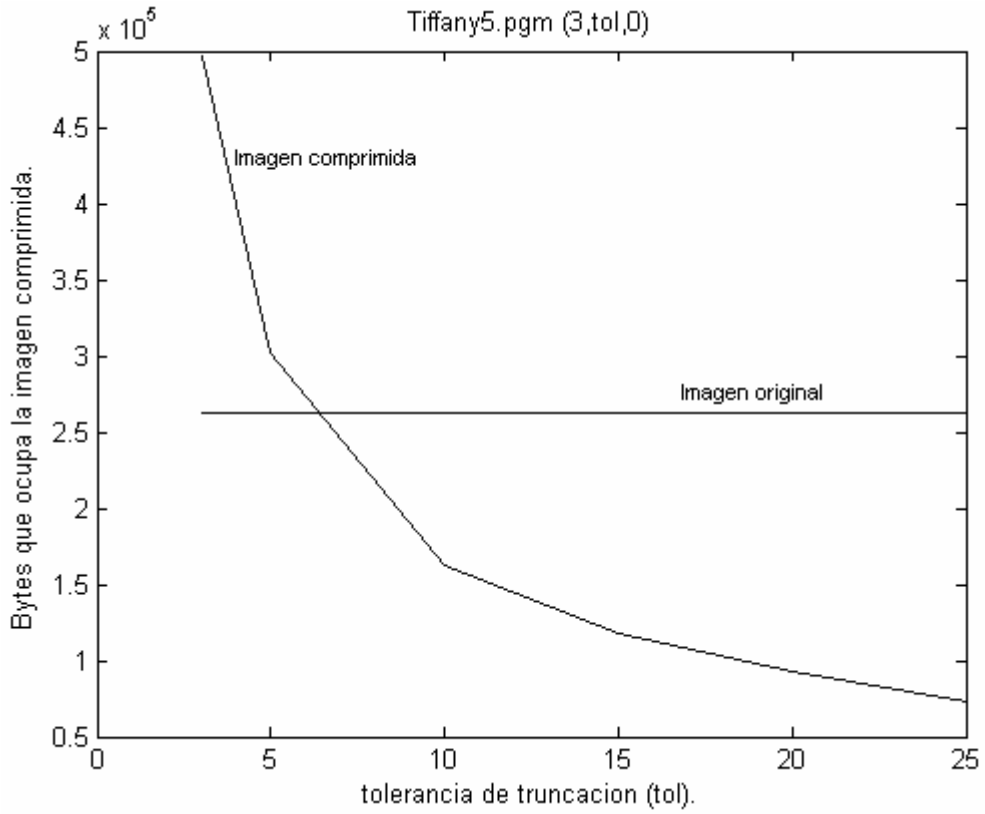


Gráficas obtenidas de la tabla dos:

Gráfica ocho. Relación entre "**Bytes**" de la imagen comprimida y "**Com**":

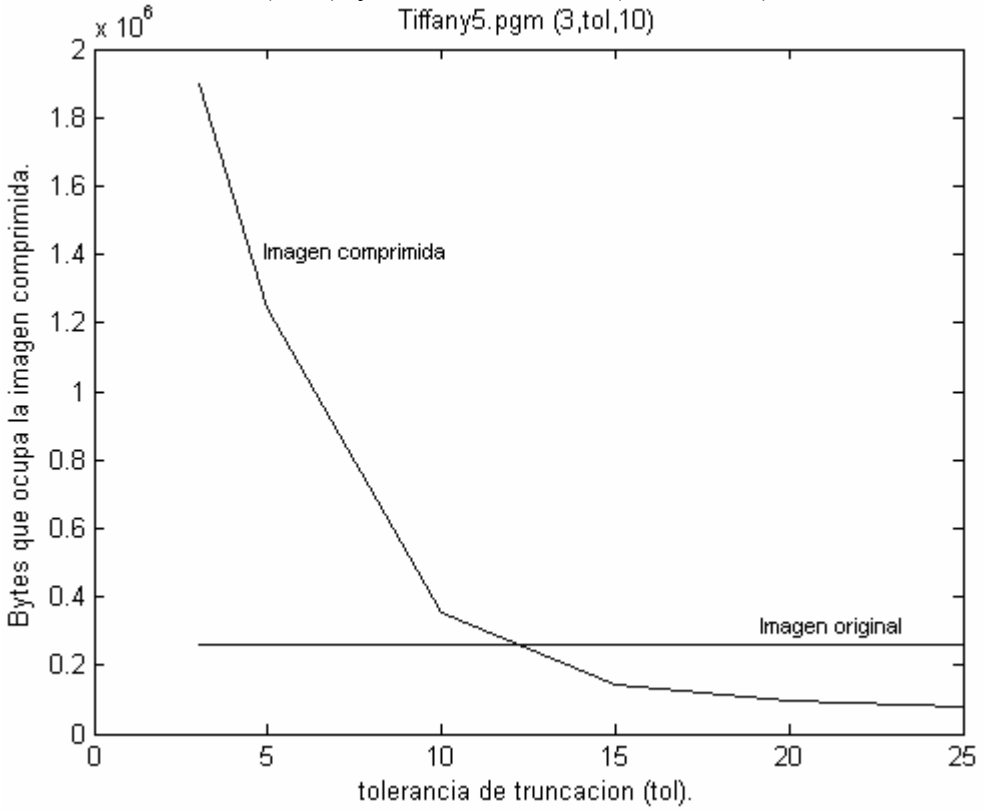


Gráfica nueve. **Bytes** que ocupa la imagen comprimida cuando el nivel de multirresolución vale tres (**l = 3**), y el ruido vale cero (**ruido = 0**):



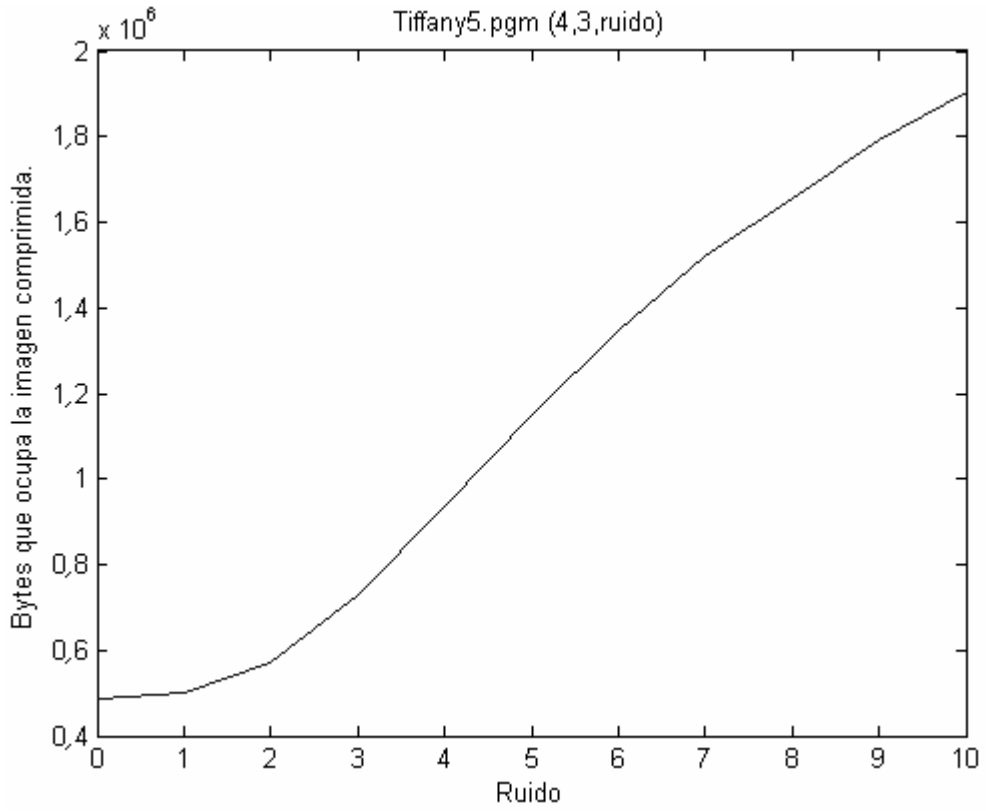
Hemos de tener en cuenta que la imagen original ha sido considerada con formato uint8, mientras que el vector v de la versión comprimida es tipo double.

Gráfica diez. **Bytes** que ocupa la imagen comprimida cuando el nivel de multirresolución vale tres (**l = 3**), y el ruido vale diez (**ruido = 10**):

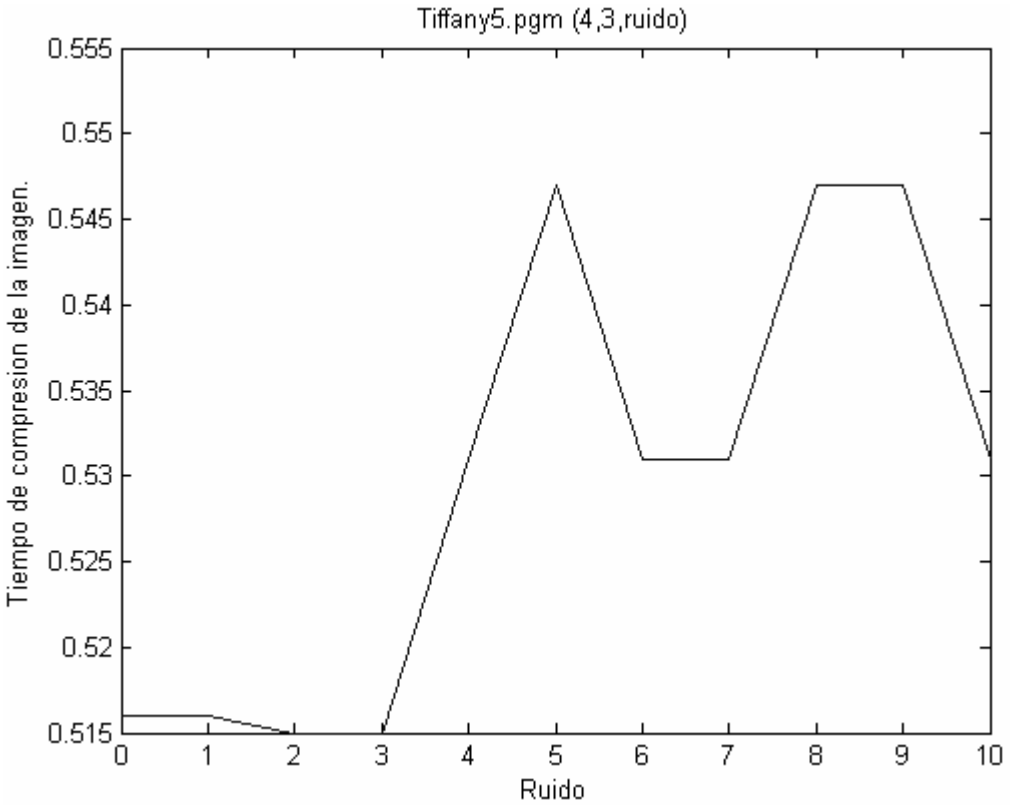


Gráficas obtenidas de la tabla tres:

Gráfica once. **Bytes** que ocupa la imagen comprimida frente a la variación del **ruido**:



Gráfica doce. **Tiempo de compresión** de la imagen frente a la variación del **ruido**:



S.5. Sección cinco. Imágenes:

Para obtener estas imágenes de muestra, se ha empleado la carpeta llamada así: "unioncomdescom".

Las imágenes geométricas se comprimen más y con mayor rapidez que las imágenes que reproducen la realidad (fotografías de paisajes, de personas, de animales, etcétera).

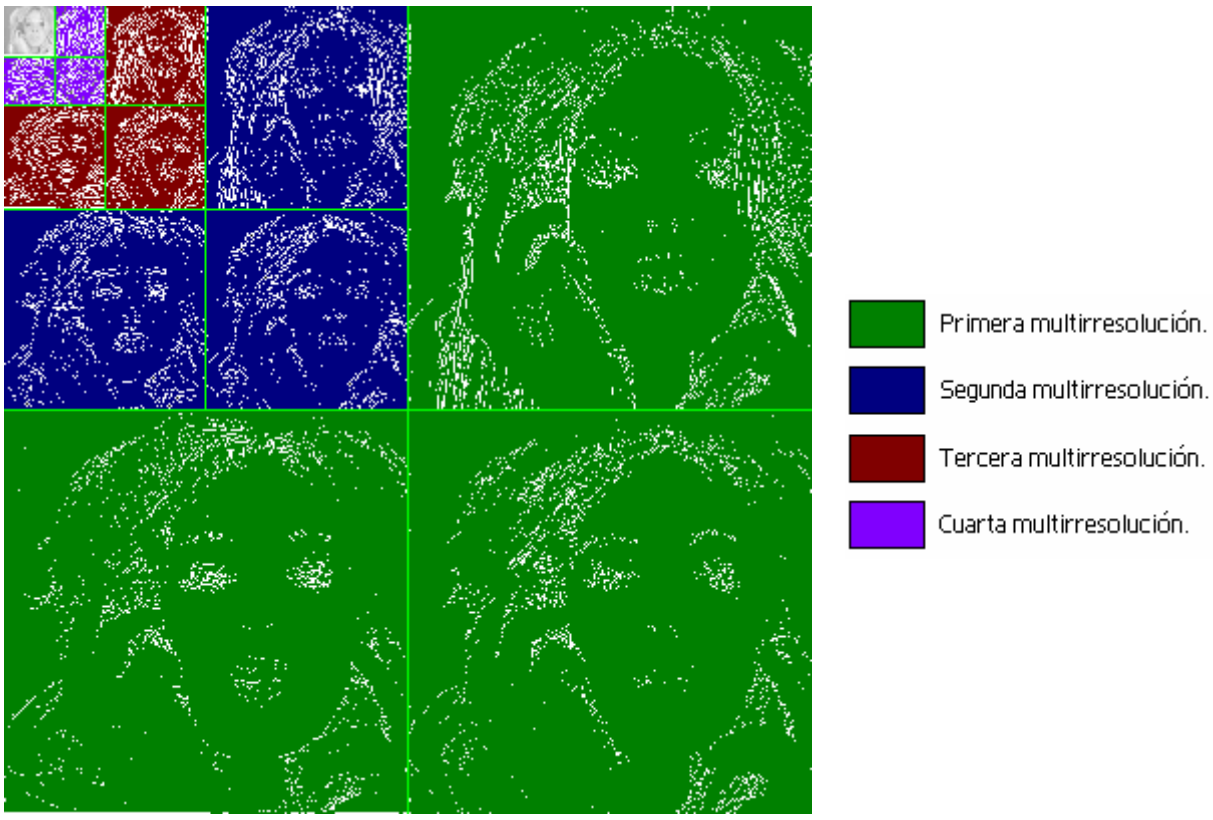
La orden que se ha usado es ésta:

```
>> [a,a1,mra,v,i,j,ta]=unioncomdescom(l,tol,ruido,im,met);
```

Por si no se ha leído ninguna de las anteriores secciones, ésta es una explicación de las variables de entrada y salida de la función "unioncomdescom":

Parámetros de entrada (están entre paréntesis):

" **l** " es el nivel de multirresolución que se quiere que exista.



La imagen que está en escala de grises es la imagen original reducida.

Todos los puntos blancos de las multirresoluciones son los errores necesarios para devolver la imagen original reducida a su tamaño original (cualquier color distinto del blanco es un cero –información eliminada cuando la imagen original sea comprimida–).

La gráfica ha sido realizada con cuatro niveles de multirresolución.

"**tol**" es la tolerancia de truncación, es decir, es el valor hasta el cual todo valor será igualado a cero. Cuanto mayor es este valor, más comprimida queda la imagen original, y peor es la reconstrucción de la imagen original.

"**ruido**" es el nivel de ruido blanco. "0" indica que no hay ruido blanco. Cuanto mayor sea este número, peor es la compresión, ya que se guardarán más valores como errores.

"**im"** es la imagen que utiliza la función "compresor" (función que es llamada dentro de la función "unioncomdescom"), escribir lo siguiente (es una guía):
'nombre de la imagen.extensión de la imagen'

"**met**" es el método de multirresolución. En este proyecto, únicamente, ha sido desarrollado un método ('lin', que quiere decir esto: lineal), pero algunas funciones están preparadas para que sea fácil incorporar nuevos métodos no necesariamente lineales.

Parámetros de salida:

Los parámetros que aparecen cuando se escribe la siguiente orden en Matlab: "**whos**" son los siguientes (en la llamada a la función, **aparecen entre corchetes**):

"**a"** es la variable donde es guardada la imagen que primero ha sido comprimida, y, después, descomprimida.

"**a1"** es la variable donde es guardada la imagen original.

"**mra"** es donde es guardada la imagen comprimida, junto con los errores que son necesarios para poder reconstruirla.

"**v**" es el vector que contiene los valores significativos, tanto de la imagen reducida, como de los errores guardados ("v" contiene la imagen comprimida).

" **i** ", " **j** " son los vectores que contienen la ubicación de los valores del vector "v" (son imprescindibles en la descompresión de la imagen).

"**ta**" es el tamaño de la imagen (ta = 512, significa: imagen = 512·512. ta = 256, significa: imagen = 256·256).
Parámetros que **aparecen en la pantalla** cuando es ejecutada la función "unioncomdescom":

"**rat**" es el numero total de elementos de la imagen, dividido por los elementos de la imagen que han sido guardados en la compresión de la misma. Éste es el ratio de compresión de la imagen. Es una medida utilizada en diversos estudios.

"**Com**" es el tanto por ciento que es comprimida la imagen.

"**PSNR**" es una medida de calidad de reconstrucción realizada de la imagen utilizando la versión comprimida. Cuanto mayor es el valor de "PSNR", mejor es la calidad de la imagen que ha sido reconstruida. Se considera un valor aceptable de esta medida un valor mayor que 25 aunque depende mucho de la imagen comprimida.

$$P.S.N.R = 20 \cdot \log\left(\frac{255}{\sqrt{\frac{\sum(|\text{Imagen comprimida y, después, descomprimida}_{i,j.} - \text{Imagen original}_{i,j.}|)^2}{(\text{Máximo valor, de entre las filas y las columnas, de la imagen original})^2}}}\right).$$

Sin ruido:

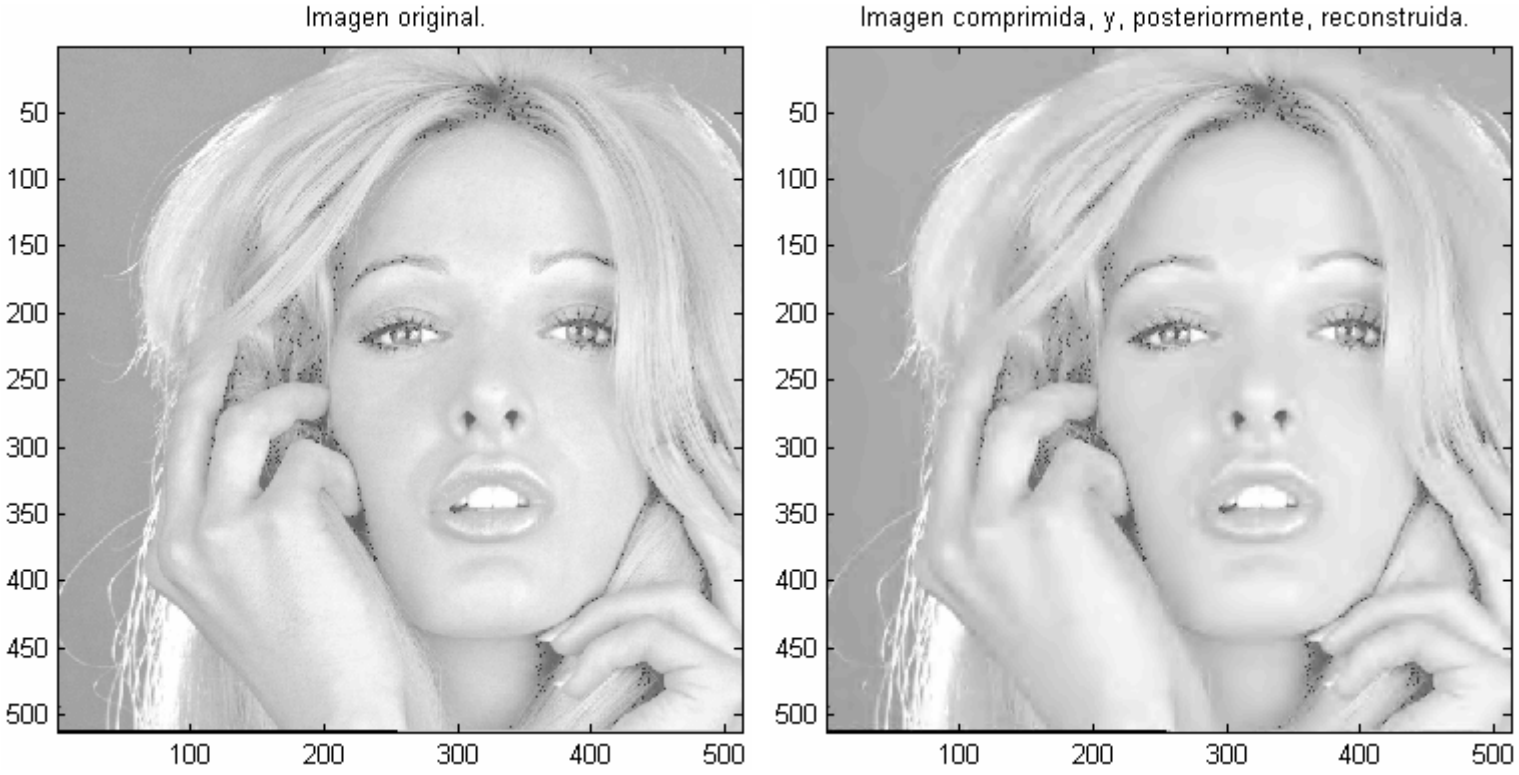
Primera imagen:

```
>> [a,a1,mra,v,i,j,ta]=unioncomdescom(4,10,0,'tiffany5.pgm','lin');

rat = 22.1705      Com = 95.4895      PSNR = 35.4421

>> a=uint8(a);
>> ta=uint16(ta);
>> whos
  Name      Size                Bytes  Class

  a       512x512              262144  uint8 array
  a1      512x512             2097152  double array
  i     11824x1                47296  uint32 array
  j     11824x1                47296  uint32 array
  mra     512x512              2097152  double array
  ta        1x1                    2  uint16 array
  v     11824x1                94592  double array
```



Visualmente, la reconstrucción parece ser buena.

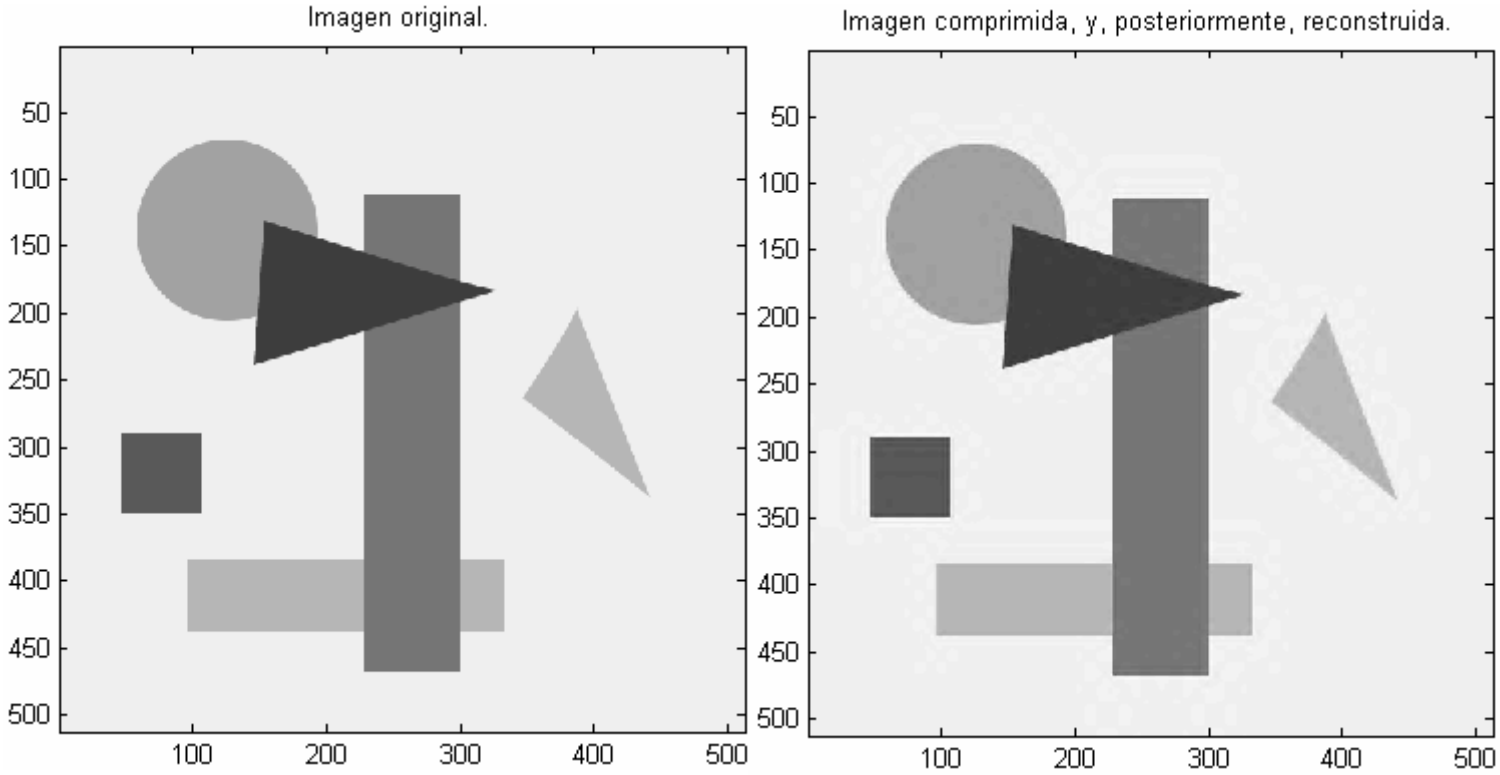
Segunda imagen:

```
>> [a,a1,mra,v,i,j,ta]=unioncomdescom(4,10,0,'seis5.pgm','lin');

rat = 52.6394      Com = 98.1003      PSNR =43.8760

>> a=uint8(a);
>> ta=uint16(ta);
>> whos
  Name      Size                Bytes  Class

  a       512x512              262144  uint8 array
  a1      512x512             2097152  double array
  i      4980x1                19920  uint32 array
  j      4980x1                19920  uint32 array
  mra     512x512              2097152  double array
  ta        1x1                    2  uint16 array
  v      4980x1                39840  double array
```



Visualmente, la reconstrucción tiene una enorme similitud con el original.

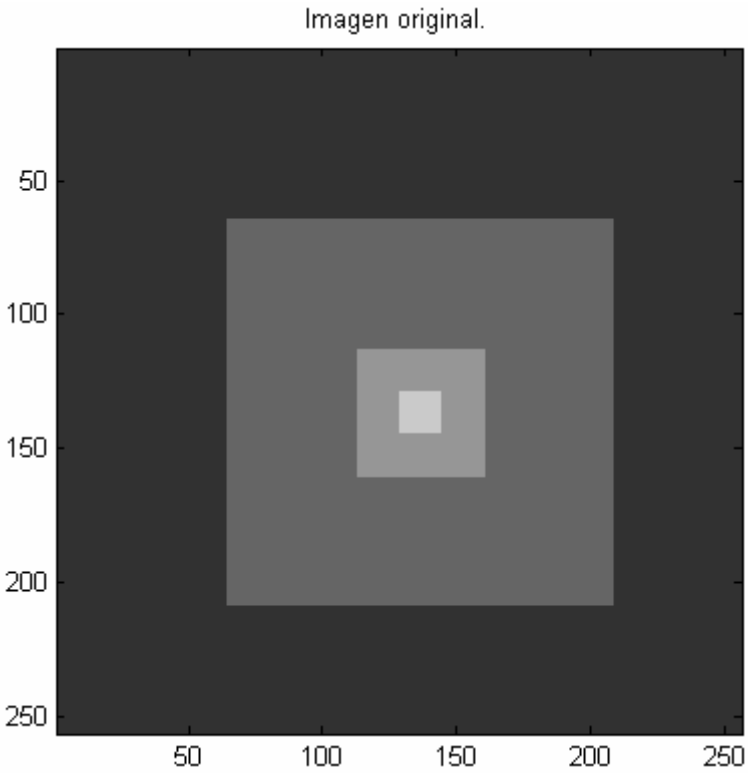
Tercera imagen:

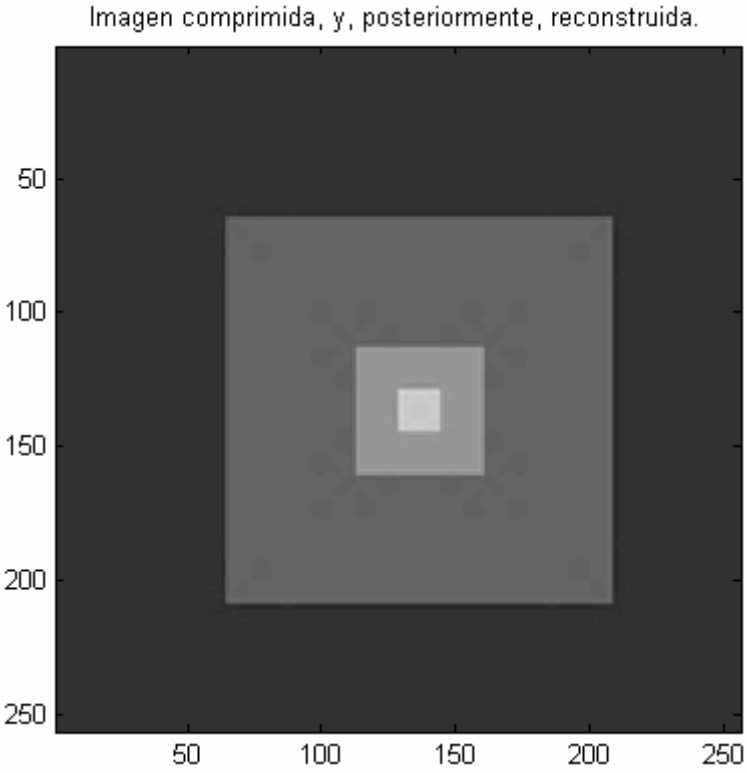
```
>> [a,a1,mra,v,i,j,ta]=unioncomdescom(4,10,0,'squares2.pgm','lin');

rat = 66.8735      Com = 98.5046      PSNR = 44.8951

>> a=uint8(a);
>> ta=uint16(ta);
>> whos
  Name      Size                Bytes  Class

  a       256x256              65536  uint8 array
  a1      256x256             524288  double array
  i       980x1                3920  uint32 array
  j       980x1                3920  uint32 array
  mra     256x256              524288  double array
  ta      1x1                  2  uint16 array
  v       980x1                7840  double array
```





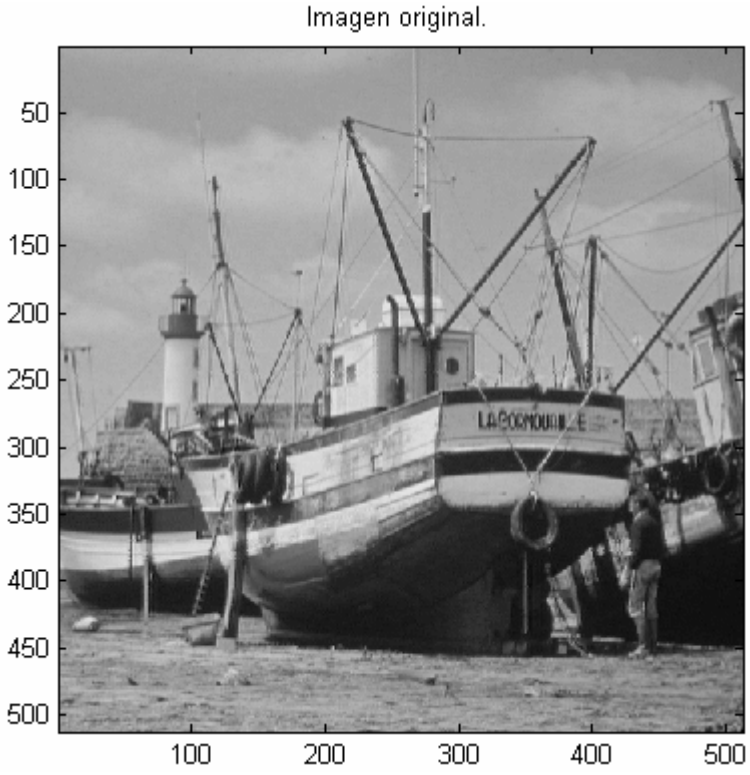
Cuarta imagen:

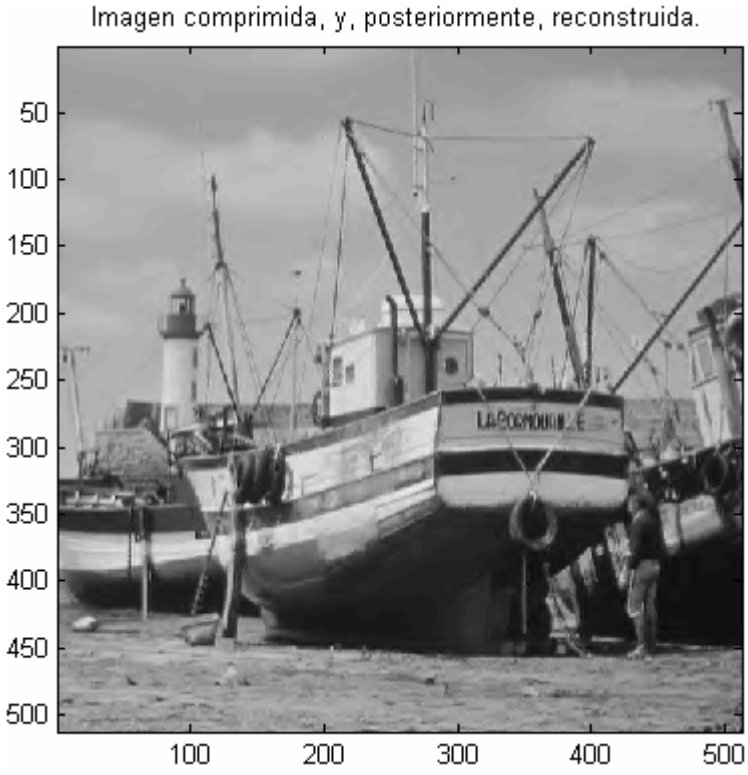
```
>> [a,a1,mra,v,i,j,ta]=unioncomdescom(4,10,0,'boat5.pgm','lin');

rat = 12.6268      Com = 92.0803      PSNR = 33.8562

>> a=uint8(a);
>> ta=uint16(ta);
>> whos
  Name      Size                Bytes  Class

  a       512x512              262144  uint8 array
  a1      512x512             2097152  double array
  i     20761x1                83044  uint32 array
  j     20761x1                83044  uint32 array
  mra     512x512             2097152  double array
  ta        1x1                   2  uint16 array
  v     20761x1               166088  double array
```





Con ruido:

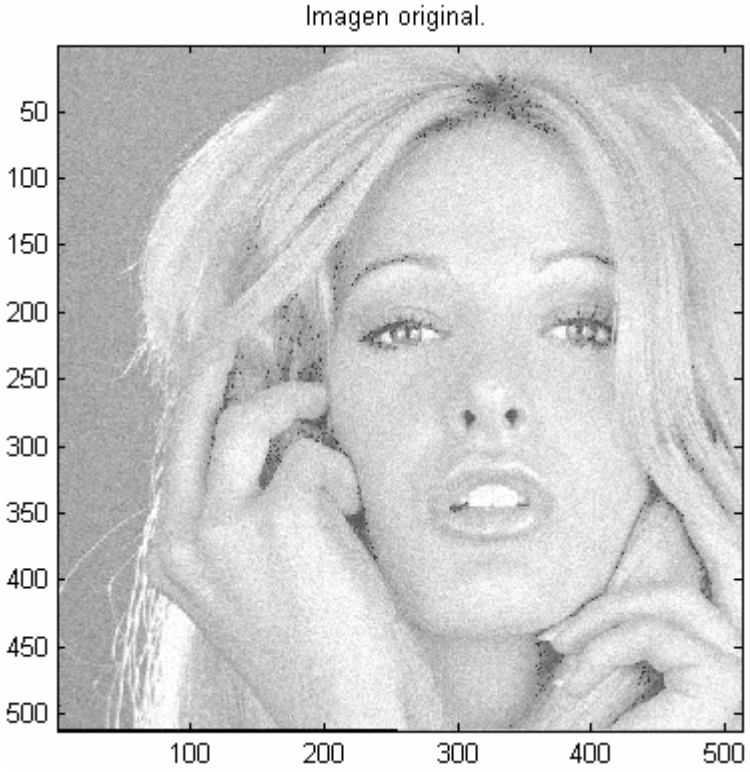
Quinta imagen:

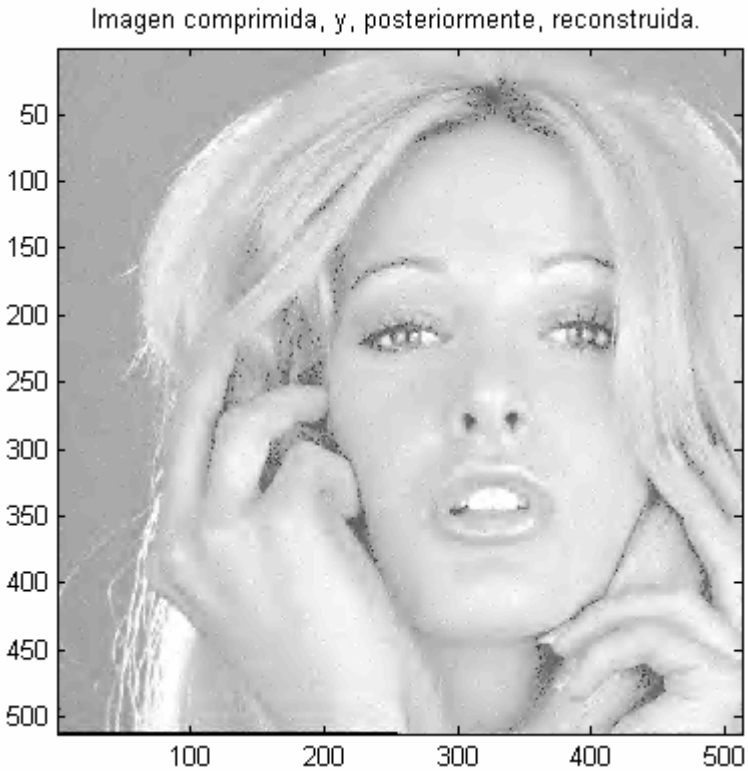
```
>> [a,a1,mra,v,i,j,ta]=unioncomdescom(4,15,10,'tiffany5.pgm','lin');

rat = 26.1022      Com = 96.1689      PSNR = 27.5370

>> a=uint8(a);
>> ta=uint16(ta);
>> whos
  Name      Size                Bytes  Class

  a       512x512              262144  uint8 array
  a1      512x512             2097152  double array
  i     10043x1                40172  uint32 array
  j     10043x1                40172  uint32 array
  mra     512x512              2097152  double array
  ta        1x1                    2  uint16 array
  v     10043x1                80344  double array
```





Igual que cuando no hay ruido, la imagen reconstruida tiene una calidad aceptable.

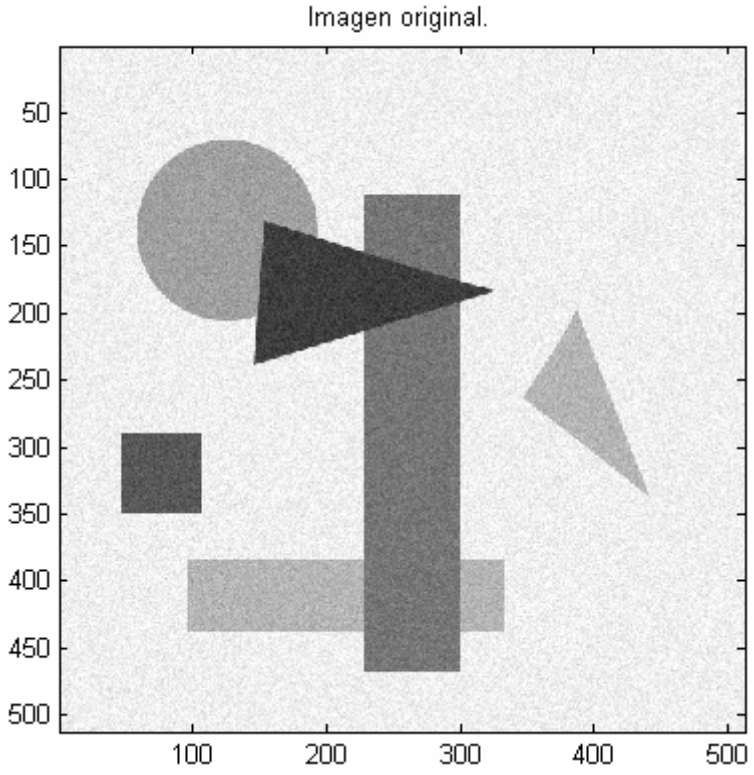
Sexta imagen:

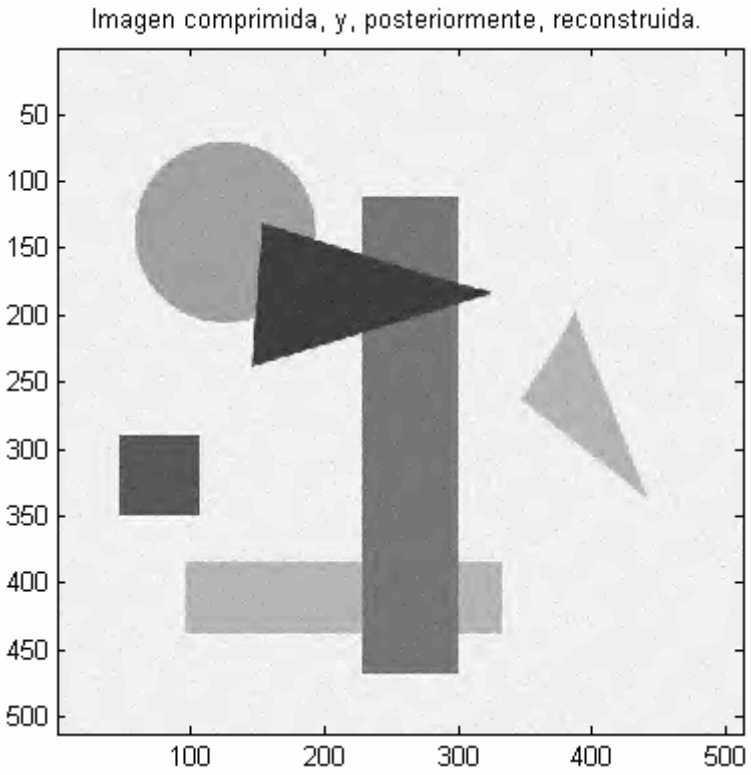
```
>> [a,a1,mra,v,i,j,ta]=unioncomdescom(4,15,10,'seis5.pgm','lin');

rat = 51.7764      Com = 98.0686      PSNR = 28.1833

>> a=uint8(a);
>> ta=uint16(ta);
>> whos
  Name      Size                Bytes  Class

  a       512x512              262144  uint8 array
  a1      512x512             2097152  double array
  i      5063x1                20252  uint32 array
  j      5063x1                20252  uint32 array
  mra     512x512             2097152  double array
  ta        1x1                   2  uint16 array
  v      5063x1                40504  double array
```





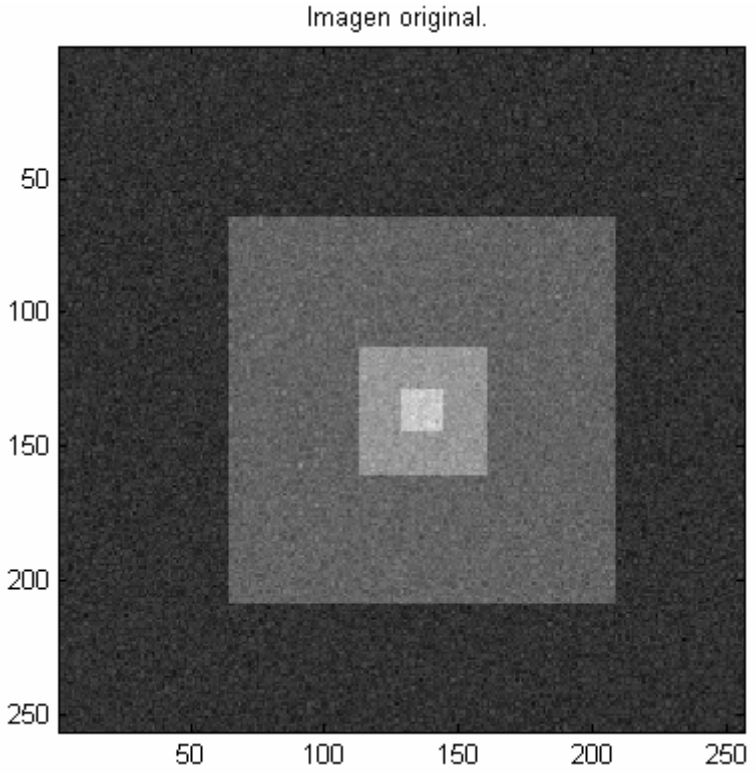
Séptima imagen:

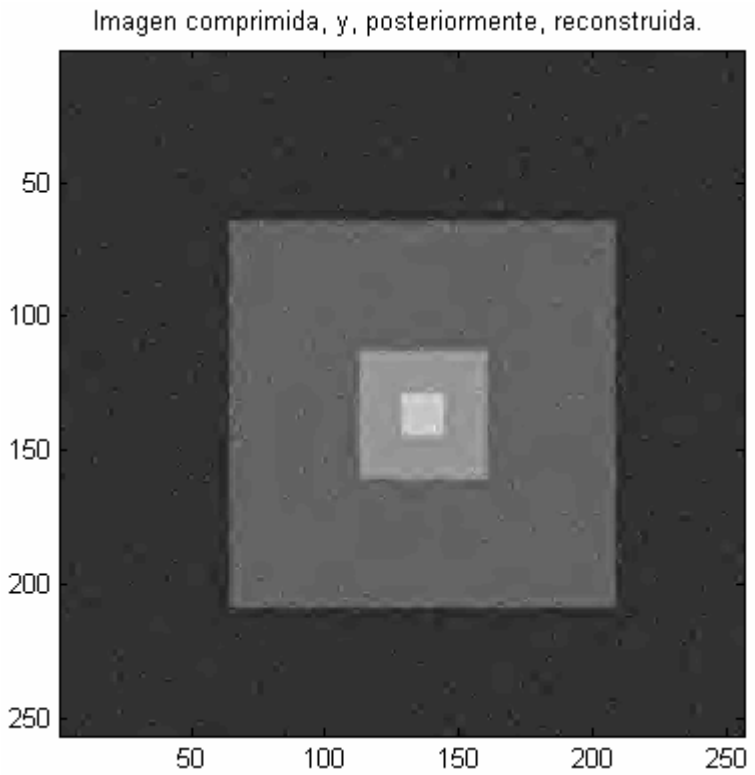
```
>> [a,a1,mra,v,i,j,ta]=unioncomdescom(4,15,10,'squares2.pgm','lin');

rat = 69.5711       Com = 98.5626       PSNR = 28.1748

>> a=uint8(a);
>> ta=uint16(ta);
>> whos
  Name      Size                Bytes  Class

  a       256x256               65536  uint8 array
  a1      256x256              524288  double array
  i       942x1                 3768  uint32 array
  j       942x1                 3768  uint32 array
  mra     256x256              524288  double array
  ta      1x1                       2  uint16 array
  v       942x1                 7536  double array
```





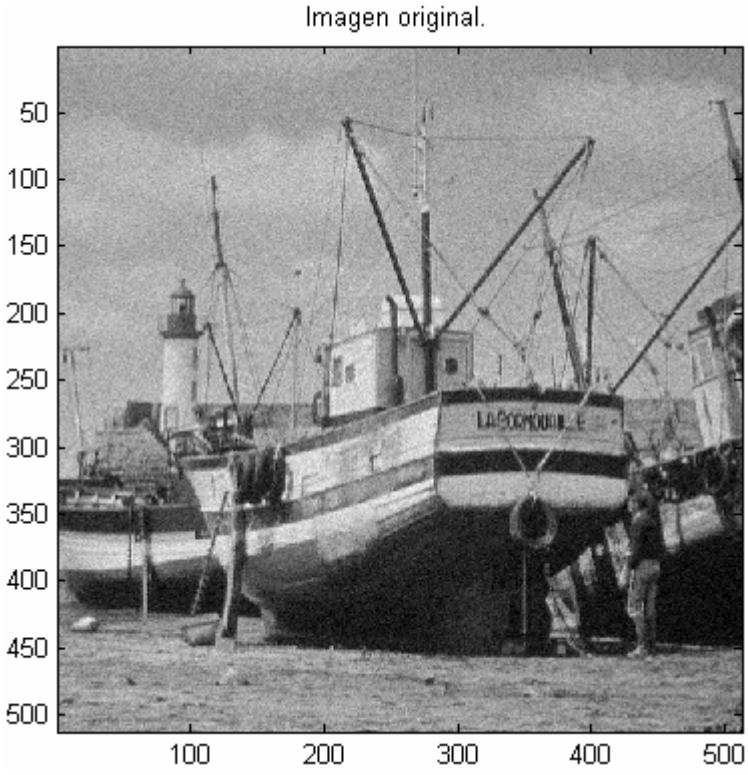
Octava imagen:

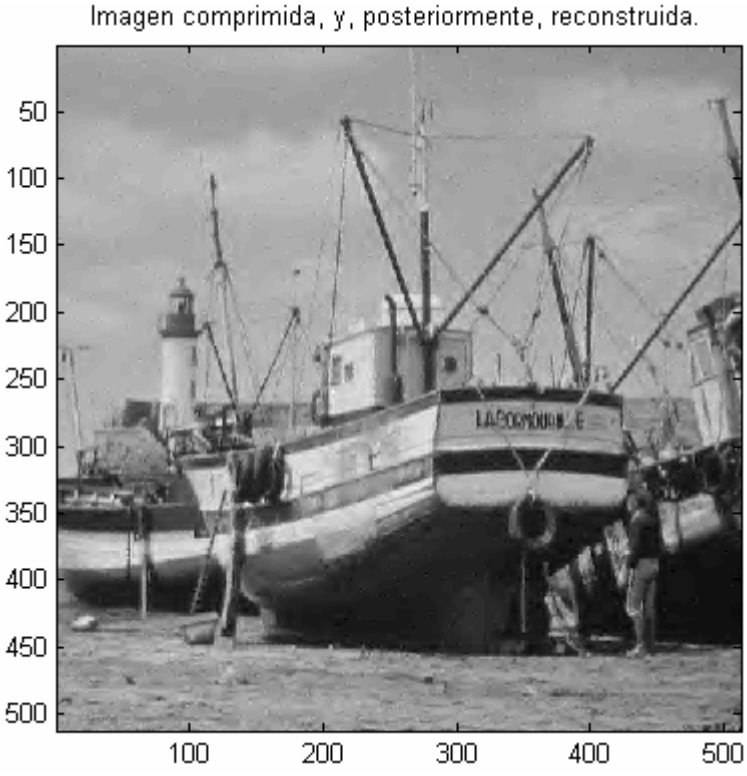
```
>> [a,a1,mra,v,i,j,ta]=unioncomdescom(4,15,10,'boat5.pgm','lin');

rat = 16.0657      Com = 93.7756      PSNR = 27.2200

>> a=uint8(a);
>> ta=uint16(ta);
>> whos
  Name      Size                Bytes  Class

  a       512x512              262144  uint8 array
  a1      512x512             2097152  double array
  i     16317x1                65268  uint32 array
  j     16317x1                65268  uint32 array
  mra     512x512              2097152  double array
  ta        1x1                    2  uint16 array
  v     16317x1               130536  double array
```





Parte II: Comparación, en bits, con los formatos standard JPEG y PNG.

Introducción:

En esta sección, lo que se pretende es combinar el método bassado en la multirresolución de Ami Harten con el método P.N.G. El método de Ami Harten elimina información redundante de la imagen, mientrasque el método P.N.G. no elimina información, es un método sin pérdida. Se ha pensado en compararlo con el método J.P.E.G. (que combina dos métodos: el primero elimina información que se considera que puede ser despreciable para el ojo humano; esta eliminación está basada en estudios fisiológicos; el segundo método está basado en el método ideado por Huffman, que es un método que comprime sin eliminar información).

Se ha preparado una función llamada así: "guionmet2". Esta función genera tres ficheros: "v1", vector binario que contiene la localización tanto de los valores positivos como de los negativos del vector v2; "v2", que contiene los valores absolutos de la matriz que se quiere comprimir; y, "apos", que contiene la matriz de posiciones de todos los elementos distintos de cero.
Estos tres ficheros juntos constituyen la matriz comprimida.
Los formatos JPEG y PNG fueron escogidos por su masiva difusión, como puede comprobarse en la web, por ejemplo consultar www.iiia.csic.es

Formatos de datos y contextos de uso.

Multimedia en la Web:

Existen numerosos formatos en que se pueden explotar las capacidades multimedia de www (world wide web). Cada uno de ellos tiene sus ventajas y desventajas, y algunos cumplen más normas que otros. Además, es importante saber cuándo usar un formato multimedia.

Formatos de imagen:
Los formatos digitales de imágenes se dividen en vectoriales y bitmapped.
El formato vectorial para la web de preferencia es SVG. "SVG" es un formato que soporta formas vectoriales bidimensionales, texto, y bitmaps. También se pueden poner eventos a los elementos, puesto que su tratamiento es tipo DOM, igual que un documento XHTML.
Una imagen digital bitmapped es una imagen que ha pasado por un proceso de conversión para que pueda ser almacenada en forma de bits en un computador.

La unidad mínima de una imagen digital es un píxel, que es un pequeño punto; un píxel es la menor unidad de medida de una pantalla. Mientras más puntos tenga una imagen, mayor será su detalle.

La resolución de pantalla mide el número de píxeles a lo ancho y alto de la pantalla. Mientras más píxeles tenga la imagen, mejor calidad tendrá ésta.

La resolución de colores describe el número de colores que pueden ser vistos en la pantalla al mismo tiempo. Un mayor número de colores produce imágenes que se perciben más reales, pero, al mismo tiempo, aumenta el espacio que ocupa la imagen en el disco. Típicamente, un sistema puede mostrar 16, 256, o más colores, dependiendo tanto del tipo de computador como de la tarjeta de video.

Veamos los formatos de imagen bitmapped más usados:

1. **JPEG** (Joint Photographic Experts Group):

"JPEG" nace para cubrir el almacenamiento masivo de imágenes debido a su elevada compresión sin pérdida aparente. Tras la compresión, se habrá sufrido una considerable pérdida de calidad respecto al original, pero, para aplicaciones de tipo web, trabajará a la perfección. "JPEG" se diseñó con el fin de poder comprimir imágenes, tanto de color, como en escala de grises, que representaran, o fotografías, o imágenes del mundo real. La compresión JPEG funciona descartando, selectivamente, datos de la imagen. Empieza eliminando datos que no son perceptibles, pero, si se comprime demasiado, se desvirtúa la imagen. Ya que se eliminan datos, a la compresión JPEG se la llama: "compresión con pérdida".

Funciona correctamente con imágenes realistas (imágenes escaneadas, fotografías, arte natural), pero incorrectamente con, o caricaturas, o dibujos simples.

El formato JPEG soporta 24 bits por píxel, y 8 bits por píxel en imágenes con escala de grises. El formato JPEG soporta paletas de varios millones de colores, lo que lo hace perfecto para imágenes fotográficas. Carece de transparencia. El formato JPEG está muy difundido.

2. **GIF** (Graphic Interchange Format):

Formato Gráfico desarrollado por CompuServe en 1987 para resolver el problema del intercambio de imágenes a través de diferentes plataformas (se usó bastante en el envío de imágenes a través de la línea telefónica). Se ha convertido en el formato normalizado de Internet.

El formato original (GIF87a) soportaba paletas de 256 colores (8bits) y compresión de imagen con una variante del algoritmo LZW. Utilizaba el algoritmo LZW para comprimir las áreas de color sólido, preservando los pequeños detalles, por eso se conoce como compresión sin pérdida. Este estándar fue revisado en 1989, resultando un nuevo standard llamado así: GIF89a.

En general, se recomienda para las imágenes simples. Para los fondos texturizados no son muy útiles, puesto que, al tener pocos colores disponibles, el computador que recibe la imagen GIF intenta encontrar el color más cercano al de la textura del fondo, produciéndose distorsiones que impiden que el texto sea visto en forma adecuada.

La compresión de GIF no es adecuada en imágenes con cambios graduales de colores.
Para ficheros de animación, lo más extendido es GIF. Altísima compresión y capacidad para guardar transparencias y animaciones.

CompuServe, recientemente, ha anunciado el desarrollo de un nuevo formato gráfico comprimido llamado GIF24, como sucesor de la actual especificación GIF89a.
"GIF24" será de dominio público, libre de patentes de compresión y con soporte para modernas capacidades gráficas, incluyendo imágenes de 24 bits (16 millones de colores). La especificación gráfica PNG será la base para el nuevo GIF24.
Jean-Loup Gailly, el desarrollador que proporcionó el código de compresión usado en PNG, participará, también, en el nuevo GIF24, que será de libre distribución, y de código abierto.
Este tipo de compresión hace que los GIF sean aptos para representar tipografías, logotipos, e ilustraciones sencillas.

Este formato está soportado por todos los navegadores gráficos desde 1995.
Esta tecnología patentada requiere una licencia para su uso, y, la patente expirará en agosto de 2.006 .

3. **PNG**:
El formato PNG se creó como una alternativa abierta y potente para los gráficos de internet. Tiene versiones de 8 y de 24 bits. Soporta transparencias de canal alfa. Proporciona imágenes perfectamente nítidas. El formato "PNG" está basado en una tecnología de compresión llamada de "deflacción", usada en programas de dominio público Info-Zip. "PNG" fue desarrollado como software de dominio público y continuará siéndolo.

4. Diferencias entre GIF, JPG, y PNG:
- Paleta de colores (JPG: siempre es rgb, GIF: paleta de colores, PNG: paleta de colores).
- Compresión (JPG: con pérdida, GIF: sin pérdida, PNG: sin pérdida).
- Transparencia (JPG: sin transparencia, GIF: color de transparencia, PNG: canal alfa).
- Licencia (JPG: gpl, GIF: compuserve, PNG: gpl).
- Interlacing (JPG: completa, GIF: horizontal, PNG: completa).
- Animación (JPG: no, GIF: sí, PNG: no).
- Corrección gamma (JPG: no, GIF: no, PNG: sí).

"PNG" puede sustituir tanto a GIF como a JPEG, ya que tiene versiones de 8 y de 24 bits. "PNG" soporta transparencias más sofisticadas (de canal alfa) que lo que pueden soportar los otros dos formatos, y proporciona imágenes perfectamente nítidas, no como JPEG.

5. Comparación entre JPEG y GIF:

Las imágenes JPEG (de extensión, o JPEG, o JPG) son más pequeñas que los GIF, y, por lo tanto, mejores para su uso en el Web. Sin embargo, cuando se trata de imágenes simples o de pocos colores, con el formato GIF se consigue un resultado que no alterará los colores del original, mientras que, en el formato JPEG, los colores sí quedan alterados.

"JPEG", al igual que GIF, está muy difundido.

Aunque se suele preferir el formato JPEG para las imágenes fotográficas, si la nitidez es importante hay que recurrir a GIF. "JPEG" tiende a suavizar las imágenes al comprimirlas.

6. Comparación entre PNG y JPEG:

El formato PNG no está tan difundido como el formato JPEG porque la imagen PNG de 24 bits suele ser más grande que la imagen JPEG equivalente. Pero, sobre todo, es porque el soporte que ha tenido PNG no ha sido suficiente hasta hace poco, e incluso, todavía, Internet Explorer 6.0 para Windows no lo soporta correctamente. Sin embargo, el resto de navegadores actuales (Netscape Navigator, Mozilla, Opera, Konqueror, Safari, ...) lo soportan correctamente. Así que poder usar PNG sin más problemas es cuestión de tiempo.

El formato PNG es más útil cuando se trabaja con imágenes simples o imágenes geométricas.

S.6. Sección seis. Funciones desarrolladas para unir el método basado en la multirresolución de Ami Harten con el PNG:

S.6.1. Guionmet2:

La función que ha sido programada es ésta:

```
function [a1]=guionmet2(l,tol,ruido,im,met)

% function guionmet2(l,tol,ruido,im,met)

[a1]=compresor(l,tol,ruido,im,met);
[a]=met2descom(l,met);
psnr(a,a1);
dibuja(a,a1);
```

Éste es el análisis de la función:

`function [a1]=guionmet2(l,tol,ruido,im,met)`	≡ nombre de la función.
`% function guionmet2(l,tol,ruido,im,met)`	≡ ayuda.
`[a1]=compresor(l,tol,ruido,im,met);`	≡ llamada a la función: "compresor".
`[a]=met2descom(l,met);`	≡ llamada a la función: "met2descom", que descomprime.
`psnr(a,a1);`	≡ llamada a la función: "psnr".
`dibuja(a,a1);`	≡ llamada a la función: "dibuja".

S.6.2. Compresor:

La función programada es ésta:

```
function [a1]=compresor(l,tol,ruido,im,met)

% La sintaxis de la funcion "compresor" es esta:
% [a1]=compresor(l,tol,ruido,im,met);
% "l", son los niveles de multirresolucion.   Escribir un valor entero que
%    pertenezca a este intervalo:  [0,7].   El cero indica que no se va a
%    comprimir la imagen, mientras que el siete es la maxima compresion que
%    puede hacerse con esta funcion.
% "tol" es la tolerancia de truncamiento.   Escribir un valor entero. Este
%       valor determina la cantidad de errores que guarda la funcion para
%       reconstruir la imagen original. Cuanto mayor sea este valor, menos
%       datos tendra la funcion para reconstruir la imagen original.
```

```
% "ruido" es el nivel de ruido que se suma a la imagen.   Este ruido representa
%        el ruido blanco introducido por una camara u otro dispositivo digital.
%        Valores grandes de ruido generan una imagen, excesivamente distorsionada.
% "im" es la imagen que utiliza la funcion "compresor", escribir lo siguiente
%     (es una guia):  'nombre de la imagen.extension de la imagen'.
% "met" metodo que se quiere utilizar: 'lin'(significa:  "lineal", y,
%      actualmente, no ha sido programado ningun otro metodo).
% Esta funcion genera estas salidas:
% "a1" matriz que representa la imagen original mas el ruido
% La version comprimida de la imagen es grabada en el disco duro

% lectura de la imagen y calculo de su tamaño
a=imread(im);
a=double(a);

% La matriz original es guardada en "a1", para, despues, calcular
% los errores cometidos.

% Se desciende por la piramide de multiresolucion.
[a1,a]=descenderc(a,l,tol,ruido,met);
mra=round(a);


% METODO DE COMPRESIÓN PROPUESTO

apos=(mra~=0);  % matriz que contiene las posiciones de los elementos no nulos
[i,j,v]=find(mra);   % guarda en v los valores que son no cero en mra
v1=(v>0);    % vector que contiene los signos de v

v2=abs(v);   % contiene los valores absolutos de los elementos de v
max(v2)
min(v2)
v2=uint8(v2);  % valores entre 0 y 255

imwrite(apos,'apos.png','png','BitDepth',1);  % como imagen binaria
imwrite(v1,'v1.png','png','BitDepth',1);   % como imagen binaria
imwrite(v2,'v2.png','png');  % como imagen uint8
```

El análisis de esta función es el siguiente:

```
function [a1]=compresor(l,tol,ruido,im,met)   ≡ nombre de la función
                                                "compresor".
a=imread(im);              ≡ se lee la imagen (im), y los números que la representan (que
                             representan su intensidad lumínica) son guardados en la matriz
                             "a".
```

a=double(a); ≡ se fuerza a que el tipo de datos guardados en la matriz "a" sea del tipo "double", esto garantiza que no existirán problemas cuando se usen algunas de las funciones de Matlab que son usadas, y que necesitan que los datos sean del tipo "double" para poder funcionar.

[a1,a]=descenderc(a,l,tol,ruido,met); ≡ se llama a la función "descenderc". La matriz original es guardada en "a1", para, después, calcular los errores cometidos.

mra=round(a); ≡ la matriz "mra" es la matriz "a", pero con todos sus valores redondeados convertidos a enteros.

apos = (mra ~ = 0); ≡ la matriz "apos" contiene las posiciones de los elementos no nulos de la matriz de multirresolución (mra).

[i,j,v]=find(mra); ≡ guarda en v los valores que son no cero en mra.

v1 = (v>0); ≡ la matriz "v1" es un vector que contiene las posiciones de los valores positivos de v (la matriz "v" contiene los valores no nulos de la matriz de multirresolución –mra–). Se ha escogido los valores positivos de la matriz "v", pero, en su lugar, podrían haberse escogido los valores negativos. Lo importante es que se sabe el signo de cualquier valor de la matriz "v" (en el caso escogido, cualquier valor de "v" que no esté entre los escogidos tiene valor negativo; y cualquier valor de "mra" que no esté localizado por la matriz "v" es un cero).

v2=abs(v); ≡ la matriz "v2" contiene los valores absolutos de los elementos de la matriz "v".

max(v2) ≡ valor máximo de la matriz "v2".

min(v2) ≡ valor mínimo de la matriz "v2".

La matriz "v2" está preparada para poder cambiar el tipo de los datos sin perder información alguna (se va a transformar del tipo "double" al tipo "uint8", que ocupa menos espacio que el tipo "double").

v2=uint8(v2); ≡ transformación del tipo "double" al tipo "uint8" [el tipo "uint8" limita los valores que contiene entre el 0 y el 255. Si algún valor de la matriz "v2" –double– superase el valor 255, en su lugar, la matriz "v2" –uint8– contendría el siguiente valor: 255; por lo que se perdería información, lo que se interpreta de la siguiente manera: la imagen reconstruida no sería la misma que la imagen original], en tal caso habría que considerar uint16.

imwrite(apos,'apos.png','png','BitDepth',1); ≡ la matriz "apos" es guardada como una imagen binaria usando el compresor PNG.

imwrite(v1,'v1.png','png','BitDepth',1); ≡ la matriz "v1" es guardada como una imagen binaria usando el compresor PNG.

imwrite(v2,'v2.png','png'); ≡ la imagen "v2" es guardada como imagen "uint8" usando el compresor PNG.

S.6.3. Descenderc:

La función programada es ésta:

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

% La sintaxis de la funcion descender es esta:
% "[ru,c]=descenderc(a,l,tol,ruido,met);".
% "a" es la imagen a la que le aplicamos el algorimo descendente de
%    multirresolucion (debe de ser "double").
% "l" son los niveles de multirresolucion.
% "tol" es la tolerancia de truncamiento.
% "ruido" es el nivel de ruido que se le suma a la imagen.
% "met" es el metodo que se quiere utilizar: 'lin' (actualmente, es el
%      unico metodo que esta programado).
%   Si se quieren obtener mas datos de las variables de entrada de esta
% funcion, escribir:  "help compresor";  los parametros que comparten ambas
% funciones estan adecuadamente explicados en la ayuda de la mencionada
% funcion.
%   Las variables de salida son estas:
% "c" contiene la version de multirresolucion de la imagen
% "ru" contiene la imagen original con ruido


n=max(size(a));
c=a;

% Se le introduce ruido a la imagen;  el ruido introducido es "ruido
% blanco", que es el mismo ruido que introducen los captadores electronicos
% (camaras fotograficas, y, camaras de video) en las imagenes que captan.

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


% Este programa esta preparado para ser ampliado cuando se decida programar
% mas de un metodo;  el que esta programado es el metodo lineal.
if(met=='lin')
    [c,nzlin]=codifli1c(c,n,l,tol);
```

```
    % "nzlin" es el numero de no ceros que existen entre los detalles (entre
    %        los errores).
end
```

Éste es el análisis de la función "descenderc":

```
function [ru,c] = descenderc(a,l,tol,ruido,met)   ≡ nombre de la función "descenderc".
n = max(size(a));                                 ≡ el tamaño máximo de la dimensión
                                                    de la matriz "a" es guardado en la
                                                    variable "n".
c = a;                                            ≡ la matriz "a" es guardada en la
                                                    variable "c" (las matrices "a" y "c"
                                                    son la misma matriz).
```

Se le introduce ruido a la imagen; el ruido introducido es "ruido blanco", que es el mismo ruido que introducen los captadores electrónicos (cámaras fotográficas, y, cámaras de vídeo) en las imágenes que captan.

```
c = c + ruido * randn(n);                         ≡ introducción del ruido blanco en la
                                                    matriz "c".
c = round(c);                                     ≡ los valores de la nueva matriz "c"
                                                    son los valores redondeados de la
                                                    antigua matriz "c".
c = ((c > = 0) & (c < = 255)).*c + (c>255)*255;   ≡ nos aseguramos que los valores de
                                                    "c" estén entre 0 y 255.
ru = c;                                           ≡ la última matriz "c" que ha sido
                                                    hallada es guardada con el siguiente
                                                    nombre: ru.
```

Este programa está preparado para ser ampliado cuando se decida programar más de un método; el que está programado es el método lineal.

```
if(met = = 'lin')                                 ≡ comienzo del bucle al que se accede
                                                    cuando se quiere usar el método
                                                    lineal (lin) –que es el único que ha
                                                    sido programado en este proyecto–.
   [c,nzlin]=codifli1c(c,n,l,tol);                ≡ llamada a la función "codifli1c".
                                                    "nzlin" es el número de no ceros
                                                    que existen entre los detalles (entre
                                                    los errores).
end                                               ≡ final del bucle del método lineal.
```

S.6.4. Codifli1c:

La función que se utiliza en esta parte del proyecto es esencialmente la misma que ya utilizamos en la primera parte. La única pero importante diferencia se trata de la manera en que los detalles (errores) poco significativos son puestos a cero. En el caso anterior se truncaba, es decir utilizabamos la fórmula:

$$tr(d, tol) = \begin{cases} d & |d| \geq tol \\ 0 & |d| < tol \end{cases}$$

Mientras que ahora lo que hacemos es cuantizar, lo cual es más adaptado a procesos de compresión ya que se pasa a trabajar con números enteros en lugar de doubles. La fórmula que nos da la cuantización es la siguiente:

$$qu(d, tol) = 2\, tol\, round\left(\frac{d}{2\, tol}\right)$$

S.6.5. Met2descom:

La función que ha sido programada es ésta:

```
function [a]=met2descom(l,met)

% Esta funcion expande la imagen que se comprimio usando la funcion
% compresor.
% La sintaxis de la funcion es esta:  [a]=met2descom(l,met);
% "l" son los niveles de multirresolucion(es un valor entero).
% "met" es el metodo de multirresolucion;  actualmente, solo esta programado
%       el metodo lineal ('lin').


% Se pasa de la version comprimida a la version de multirresolucion
[mra]=met2read;

% Se llama a la funcion que, ascendiendo por la escala de multirresolucion,
% reconstruira la imagen original:
[a]=ascenderc(mra,l,met);
```

Éste es el análisis de la función:

function [a]=met2descom(l,met)	≡ nombre de la function.
[mra]=met2read;	≡ llamada a la función: “met2read”.
[a]=ascenderc(mra,l,met);	≡ llamada a la función: “ascenderc”(ya fue explicada).

S.6.6. Met2read:

La función que ha sido programada es ésta:

```
function [mra]=met2read()

% function [mra]=met2read();
% esta funcion lee los ficheros apos.png, v1.png, v2.png y compone
% la matriz de multirresoluticion haciendo el proceso inverso del
% MET2 utilizado en la compresion

apos=imread('apos.png','png');
v1=imread('v1.png','png');
v2=imread('v2.png','png');
v2=double(v2);
v=v1.*v2 -(~v1).*v2;

[i,j]=find(apos);
mra=zeros(size(apos));
ma=sparse(i,j,v);
mra(1:max(i),1:max(j))=full(ma);
```

Éste es el análisis de la función:

`function [mra]=met2read()`	≡ nombre de la function, sin entradas especificadas.
`apos=imread('apos.png','png');`	≡ lectura del fichero “apos”.
`v1=imread('v1.png','png');`	≡ lectura del fichero “v1”.
`v2=imread('v2.png','png');`	≡ lectura del fichero “v2”.
`v2=double(v2);`	≡ transformación, de los datos de “v2”, en el formato “double”.
`v=v1.*v2 -(~v1).*v2;`	≡ formación de la matriz “v”, a partir de las matrices “v1”, y, “v2”.
`[i,j]=find(apos);`	≡ localización de los valores que son distintos de cero, en la matriz “apos”.
`mra=zeros(size(apos));`	≡ formación de una matriz de ceros con un tamaño igual al de la matriz “apos”.
`ma=sparse(i,j,v);`	≡ genera una matriz m·n, de forma que “m” sea lo siguiente: “m = max(i)”, y, “n”, sea lo siguiente: “n = max(j)”, conteniendo los valores en v.
`mra(1:max(i),1:max(j))=full(ma);`	≡ la matriz “mra”(que se extiende desde el valor uno hasta el máximo valor de “j ” en las columnas, y de “i ” en las filas) recibe los valores de la matriz completa que está formada por la salida de la anterior orden.

S.6.7. Psnr:

Ésta es la función que ha sido programada:

```
function psnr(a,a1)

n1=max(size(a));
MSE=(norm(a-a1,'fro')^2)/n1/n1;
PSNR=20*log10(255/sqrt(MSE))
```

El análisis de esta función es el siguiente:

Esta función fue desarrollada para calcular el P.S.N.R.

`function psnr(a,a1)` ≡ nombre de la función.

`n1=max(size(a));` ≡ "n1" es el máximo valor del tamaño de la matriz "a".

`MSE=(norm(a-a1,'fro')^2)/n1/n1;` ≡ error cuadrático medio entre "a" y "a1".

$$M.S.E = \frac{\sum \left| a_{i.j.} - a1_{i.j.} \right|^2}{(n1)^2} .$$

`PSNR=20*log10(255/sqrt(MSE))` ≡ Peak Signal to Noise Ratio. Medida de la calidad de la imagen, A mayor PSNR mayor calidad. La formula que se ha usado para calcular el PSNR es ésta:

$$P.S.N.R = 20 \cdot \log\left(\frac{255}{\sqrt{M.S.E.}}\right) = 20 \cdot \log\left(\frac{255}{\sqrt{\frac{\sum \left| a_{i.j.} - a1_{i.j.} \right|^2}{(n1)^2}}}\right)$$

S.6.8. Dibuja:

La función que ha sido programada es ésta:

```
function dibuja(a,a1)

figure
imagesc(a,[0 255])
colormap gray
axis('square')
title('Imagen comprimida, y, posteriormente, reconstruida.')

figure
imagesc(a1,[0 255])
colormap gray
axis('square')
title('Imagen original')
```

El análisis de esta función es el siguiente:

Ésta es una función que ha sido desarrollada para comparar la imagen reconstruida con la original de la que procede.

function dibuja(a,a1) ≡ nombre de la function "dibuja".

figure ≡ hace que aparezca una ventana en la que se podrá introducir una imagen.

imagesc(a,[0 255]) ≡ hace que aparezca la imagen en la ventana que apareció con la anterior orden.

colormap gray ≡ provoca que el color de la imagen esté en escala de grises.

axis('square') ≡ provoca que la imagen tenga una forma cuadrada.

title('Imagen comprimida, y, posteriormente, reconstruida.') ≡ hace que, sobre la imagen, aparezca el título que hay escrito entre comillas simples.

figure
imagesc(a1,[0 255])
colormap gray
axis('square')
title('Imagen original')

} ≡ este conjunto de órdenes ejecuta lo mismo que las órdenes homónimas que acaban de ser explicadas.

El resto de las funciones que se utilizan (ascenderc.m, decodiflic.m, linealde1c.m, lineale1c.m) están comentadas ya en la parte I de este proyecto y no han sido modificadas.

S.7. Sección siete. Imágenes:

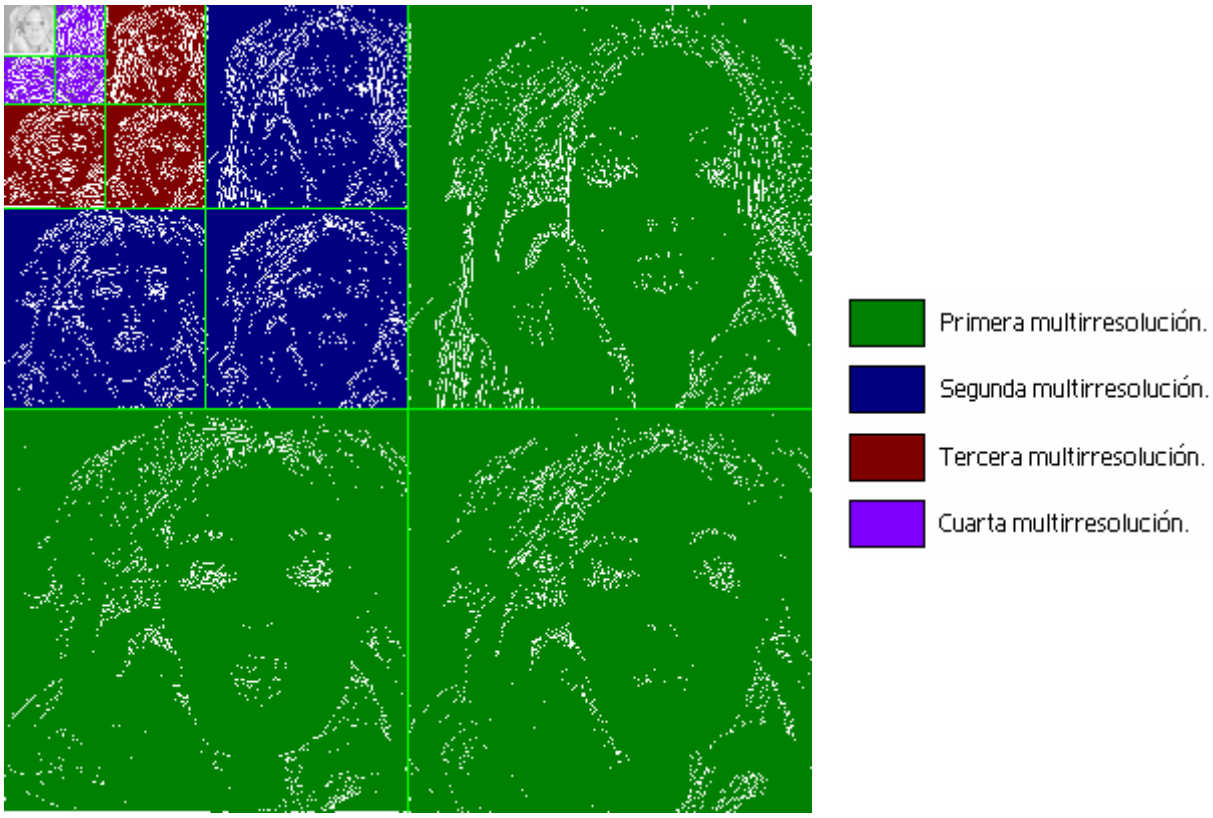
La orden que se ha usado para generar los datos y las imágenes es ésta:

>> guionmet2(l,tol,ruido,im,met);

Por si no se ha leído ninguna de las anteriores secciones, ésta es una explicación de las variables de entrada y salida de la función "guionmet2":

Parámetros de entrada (están entre paréntesis):

" **l** " es el nivel de multirresolución que se quiere que exista.



La imagen que está en escala de grises es la imagen original reducida.

Todos los puntos blancos de las multirresoluciones son los errores necesarios para devolver la imagen original reducida a su tamaño original (cualquier color distinto del blanco es un cero –información eliminada cuando la imagen original sea comprimida–).

La gráfica ha sido realizada con cuatro niveles de multirresolución.

.

"**tol**" es la tolerancia de truncación, es decir, es el valor hasta el cual todo valor será igualado a cero. Cuanto mayor es este valor, más comprimida queda la imagen original, y peor es la reconstrucción de la imagen original.

"**ruido**" es el nivel de ruido blanco. "0" indica que no hay ruido blanco. Cuanto mayor sea este número, peor es la compresión, ya que se guardarán más valores como errores.

"**im"** es la imagen que utiliza la función "compresor" (función que es llamada dentro de la función "unioncomdescom"), escribir lo siguiente (es una guía):
'nombre de la imagen.extensión de la imagen'

"**met**" es el método de multirresolución. En este proyecto, únicamente, ha sido desarrollado un método ('lin', que quiere decir esto: lineal), pero algunas funciones están preparadas para que sea fácil incorporar nuevos métodos no necesariamente lineales.

Parámetros que **aparecen en la pantalla** cuando es ejecutada la función "guionmet2":

"**PSNR**" es una medida de calidad de reconstrucción realizada de la imagen utilizando la versión comprimida. Cuanto mayor es el valor de "PSNR", mejor es la calidad de la imagen que ha sido reconstruida. Se considera un valor aceptable de esta medida un valor mayor que 25 aunque depende mucho de la imagen comprimida.

$$P.S.N.R = 20 * \log\left(\frac{255}{\sqrt{\frac{\sum(|\text{Imagen comprimida y, después, descomprimida}_{i,j.} - \text{Imagen original}_{i,j.}|)^2}{(\text{Máximo valor, de entre las filas y las columnas, de la imagen original})^2}} \right).$$

También aparecen el mínimo y el máximo en valor absoluto que toma el vector v. Se trata sólo de comprobar que efectivamente están entre 0 y 255.

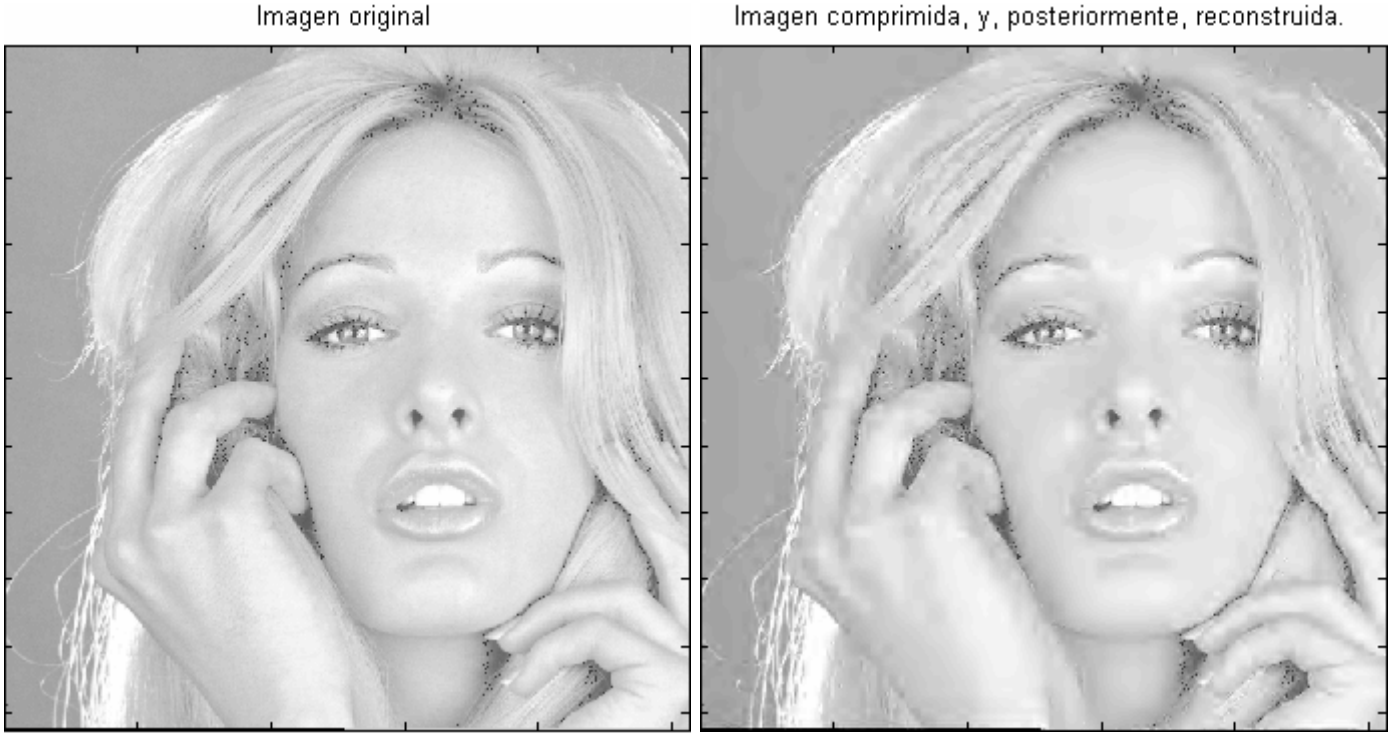
Las imágenes comprimidas aparecen en la carpeta donde están las funciones de este proyecto.

Imagen 1. Tiffany5.pgm.

Tolerancia 9. Tiffany5 (4,9,0):
>> guionmet2(4,9,0,'tiffany5.pgm','lin');

PSNR = 33.6628
Dimensión de la imagen = 512 · 512.
v1.png (2,26 KB = 2,26 Kbytes = 2,26 $\cdot 2^{10}$ = 2,26 · 1.024 @ 2.319 bytes).
v2.png (4,11 KB @ 4.218 bytes).
apos.png (7,80 KB @ 7.991 bytes).

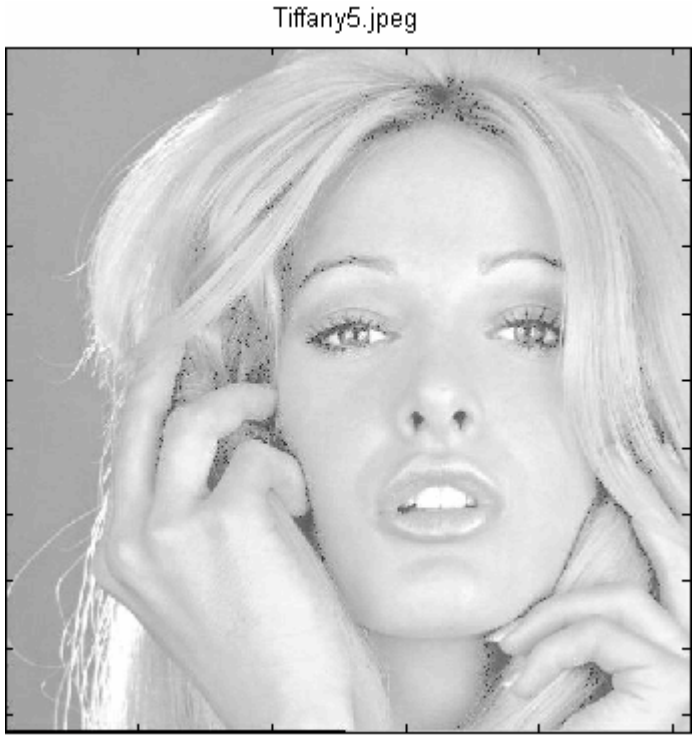
Total: 2.319 + 4.218 + 7.991 = 14.528 bytes (es lo que ocupa la imagen comprimida).



```
>> a1=imread('Tiffany5.pgm');
>> a2=double(a1);
>> imwrite(a1,'Tiffany5.jpeg','jpeg')
>> a3=imread('Tiffany5.jpeg','jpeg');
>> a4=double(a3);
>> psnr(a2,a4);
```

PSNR = 36,6608.

```
>> figure
>> imagesc(a3,[0,255])
>> axis('square')
>> colormap gray
>> title('Tiffany5.jpeg')
```

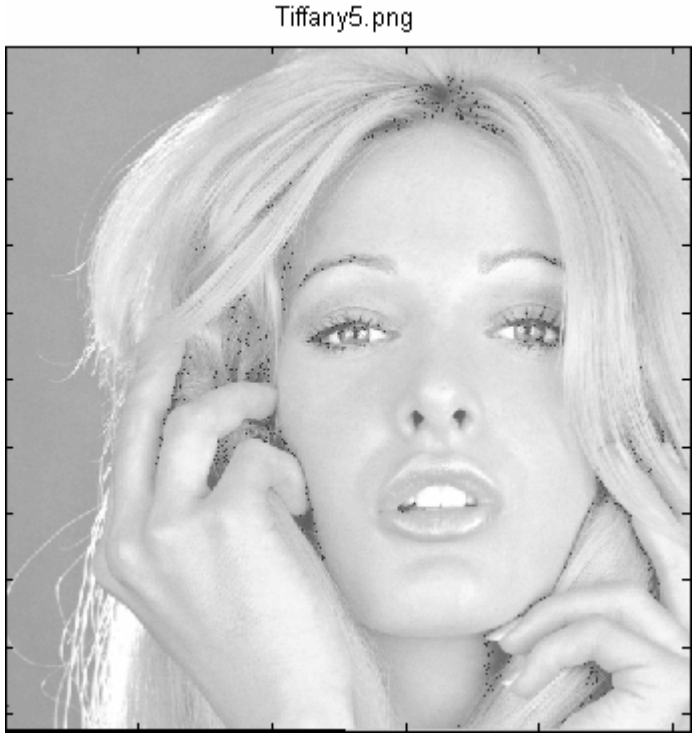


Cuánto ocupa esta imagen: 29,3 KB @ 30.031 bytes.

Comprimida con un parámetro de calidad de 75 de un rango de posibles valores en el intervalo [0, 100].

```
>> imwrite(a1,'Tiffany5.png','png')
>> a5=imread('Tiffany5.png','png');
>> figure
>> imagesc(a5)
>> colormap gray
>> axis('square')
```

PSNR = Inf SIN PÉRDIDA



Cuánto ocupa esta imagen: 138 KB @142.170 bytes.

Comparación de resultados:

Imagen de este proyecto:

Imagen original

Imagen comprimida, y, posteriormente, reconstruida.

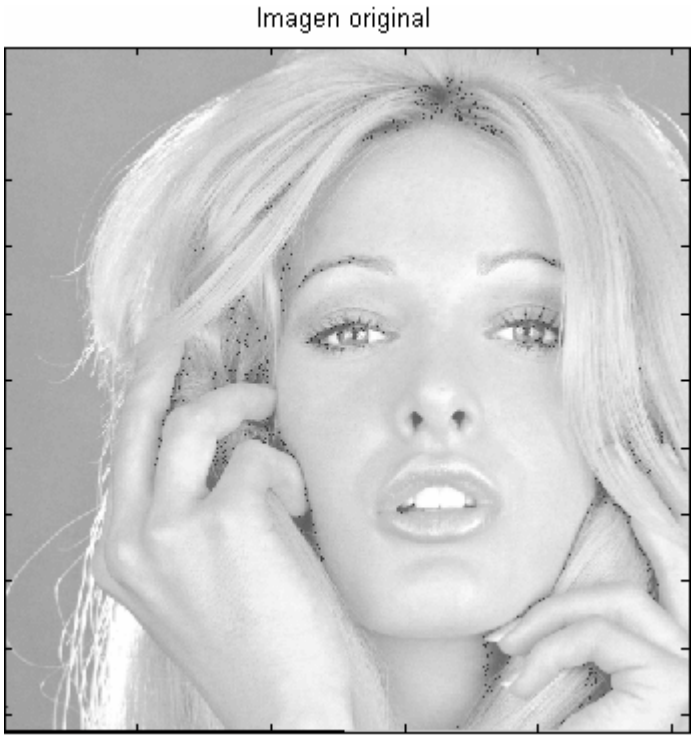
Tiffany5.jpeg

Tiffany5.png

Tolerancia 10. Tiffany5 (4,10,0):
>> guionmet2(4,10,0,'tiffany5.pgm','lin');

PSNR = 33.5557
Dimensión de la imagen = 512 · 512.
v1.png (2,15 KB @ 2.209 bytes).
v2.png (3,47 KB @ 3.560 bytes).
apos.png (7,50 KB @ 7.686 bytes).

Total = 13.455bytes (es lo que ocupa la imagen comprimida).



Comparación de resultados:

Imagen de este proyecto:

Imagen original

Imagen comprimida, y, posteriormente, reconstruida.

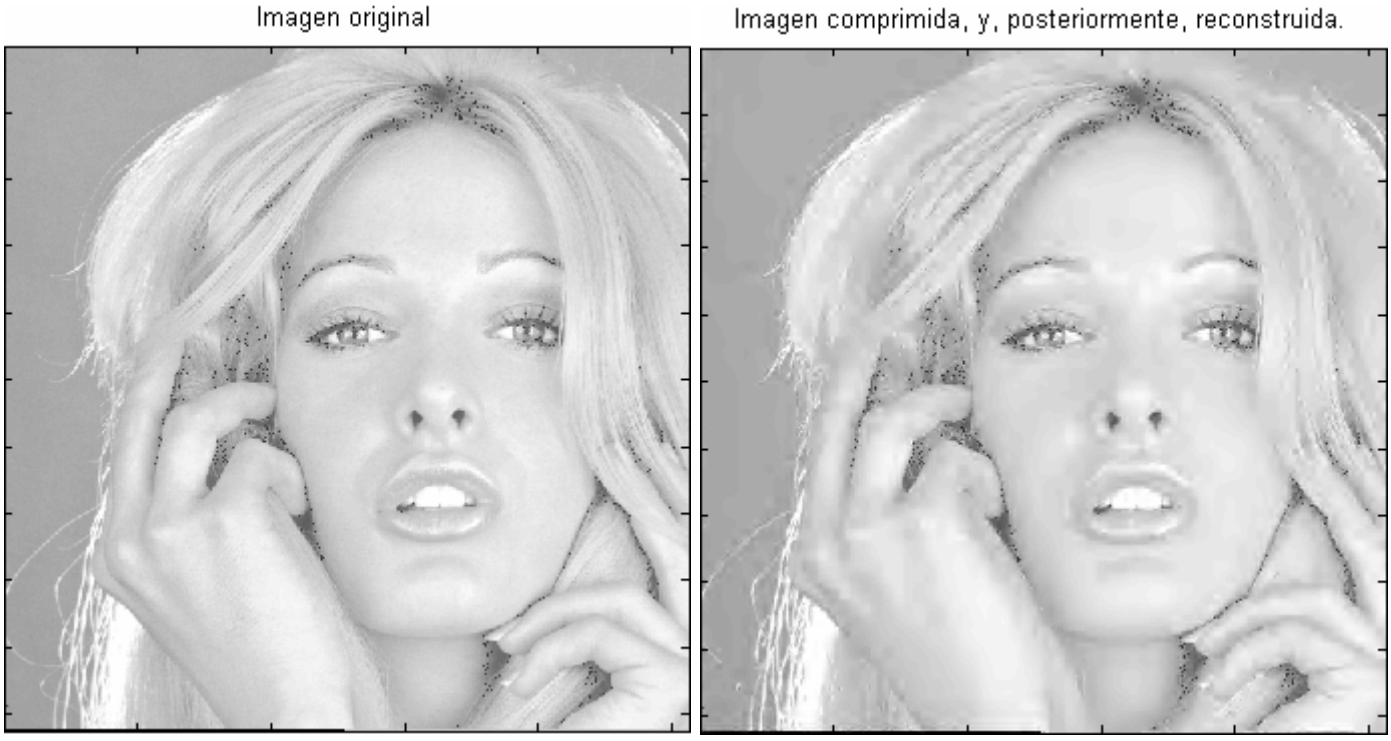
Tiffany5.jpeg

Tiffany5.png

Tolerancia 11. Tiffany5 (4,11,0):
>> guionmet2(4,11,0,'tiffany5.pgm','lin');

PSNR = 33.1850
Dimensión de la imagen = 512 · 512.
Imagen comprimida de este proyecto:
v1.png (1,91 KB @ 1.962 bytes).
v2.png (3,47 KB @ 3.560 bytes).
apos.png (6,86 KB @ 7.030 bytes).

Total: 12.552 bytes.



Comparación de resultados:

Imagen de este proyecto:

Imagen original

Imagen comprimida, y, posteriormente, reconstruida.

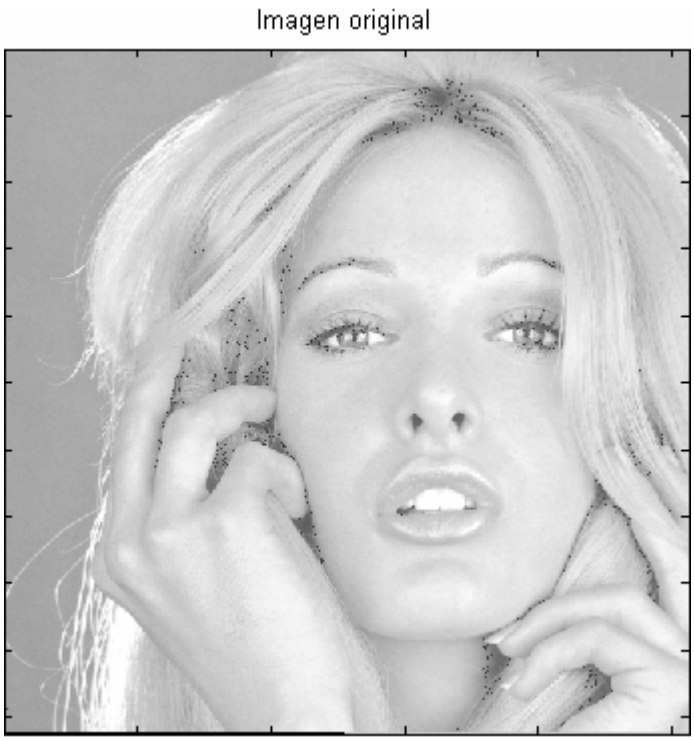
Tiffany5.jpeg

Tiffany5.png

Tolerancia 12. Tiffany5 (4,12,0):
>> guionmet2(4,12,0,'tiffany5.pgm','lin');

PSNR = 33.0711
Dimensión de la imagen = 256 · 256.
Imagen comprimida de este proyecto:
v1.png (1,86 KB @ 1.908 bytes).
v2.png (3,35 KB @ 3.440 bytes).
apos.png (6,66 KB @ 6.824 bytes).

Total: 12.172 bytes.



Comparación de resultados:

Imagen de este proyecto:

Imagen original

Imagen comprimida, y, posteriormente, reconstruida.

Tiffany5.jpeg

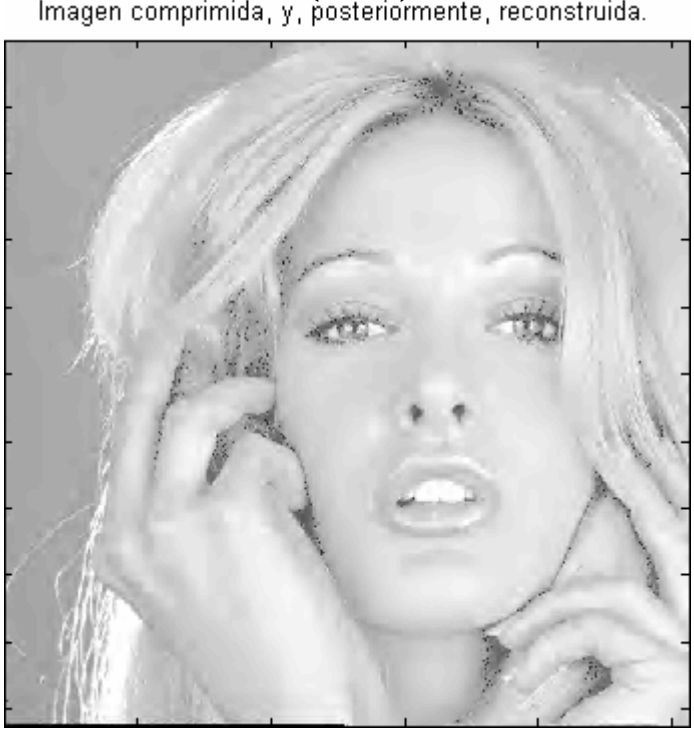
Tiffany5.png

Evolución de la imagen:

(4,9,0):

(4,10,0):

(4,11,0):



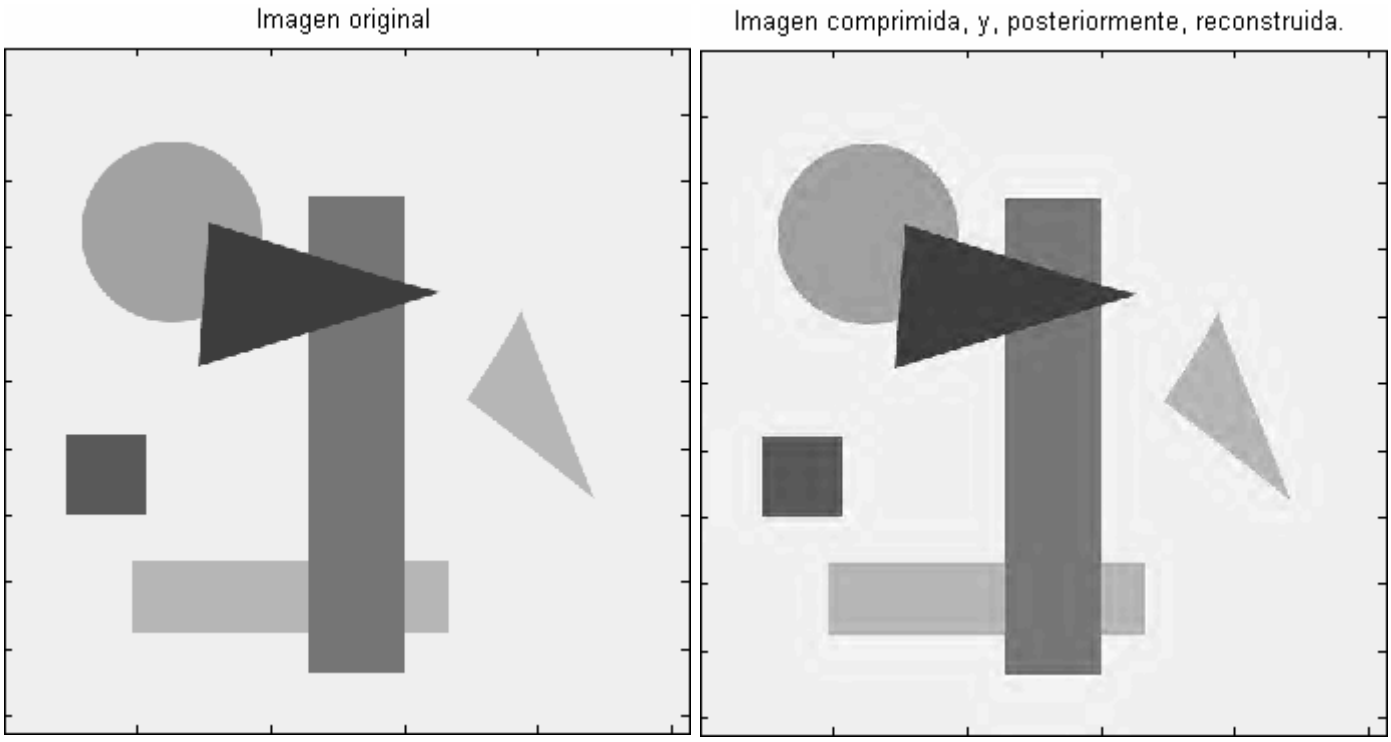
(4,12,0):

Imagen 2. Seis5.pgm.

Tolerancia 9. Seis5 (4,9,0):
>> guionmet2(4,9,0,'seis5.pgm','lin');

PSNR = 40.3087
Dimensión de la imagen = 512 · 512.
Imagen comprimida de este proyecto:
v1.png (736 bytes).
v2.png (1,72 KB @ 1.764 bytes).
apos.png (2,27 KB @ 2.325 bytes).

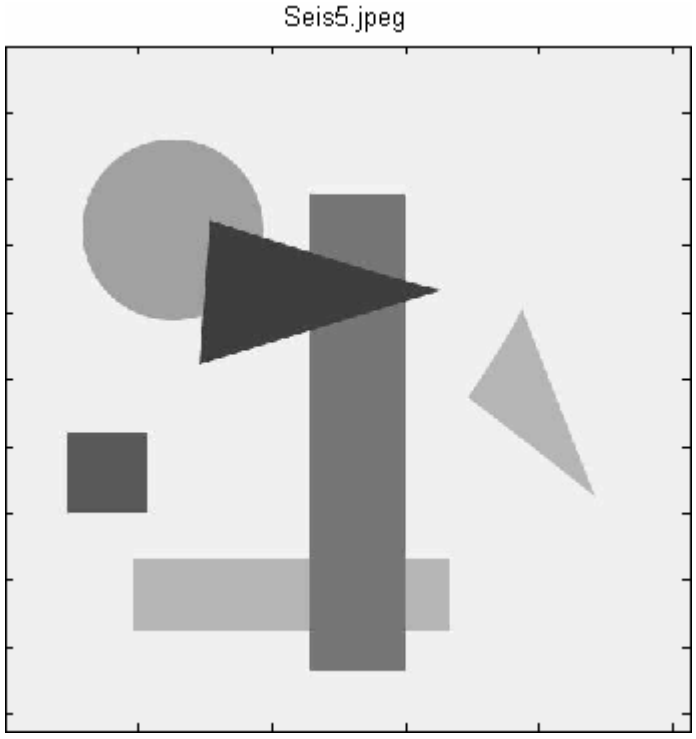
Total: 4825 bytes.



```
>> a1=imread('Seis5.pgm');
>> a2=double(a1);
>> imwrite(a1,'Seis5.jpeg','jpeg')
>> a3=imread('Seis5.jpeg','jpeg');
>> a4=double(a3);
>> psnr(a2,a4);
```

PSNR = 47,5620.

```
>> figure
>> imagesc(a3,[0,255])
>> axis('square')
>> colormap gray
>> title('Seis5.jpeg')
```

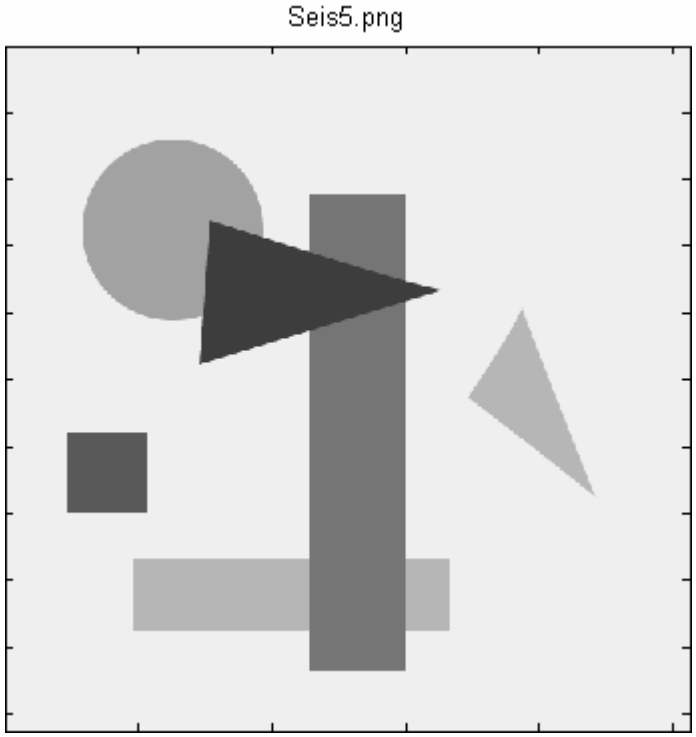


Cuánto ocupa esta imagen: 7,89 KB @ 8.085 bytes.

Comprimida con un parámetro de calidad de 75.

```
>> imwrite(a1,'Seis5.png','png')
>> a5=imread('Seis5.png','png');
>> figure
>> imagesc(a5)
>> colormap gray
>> axis('square')
>> title('Seis5.png')
```

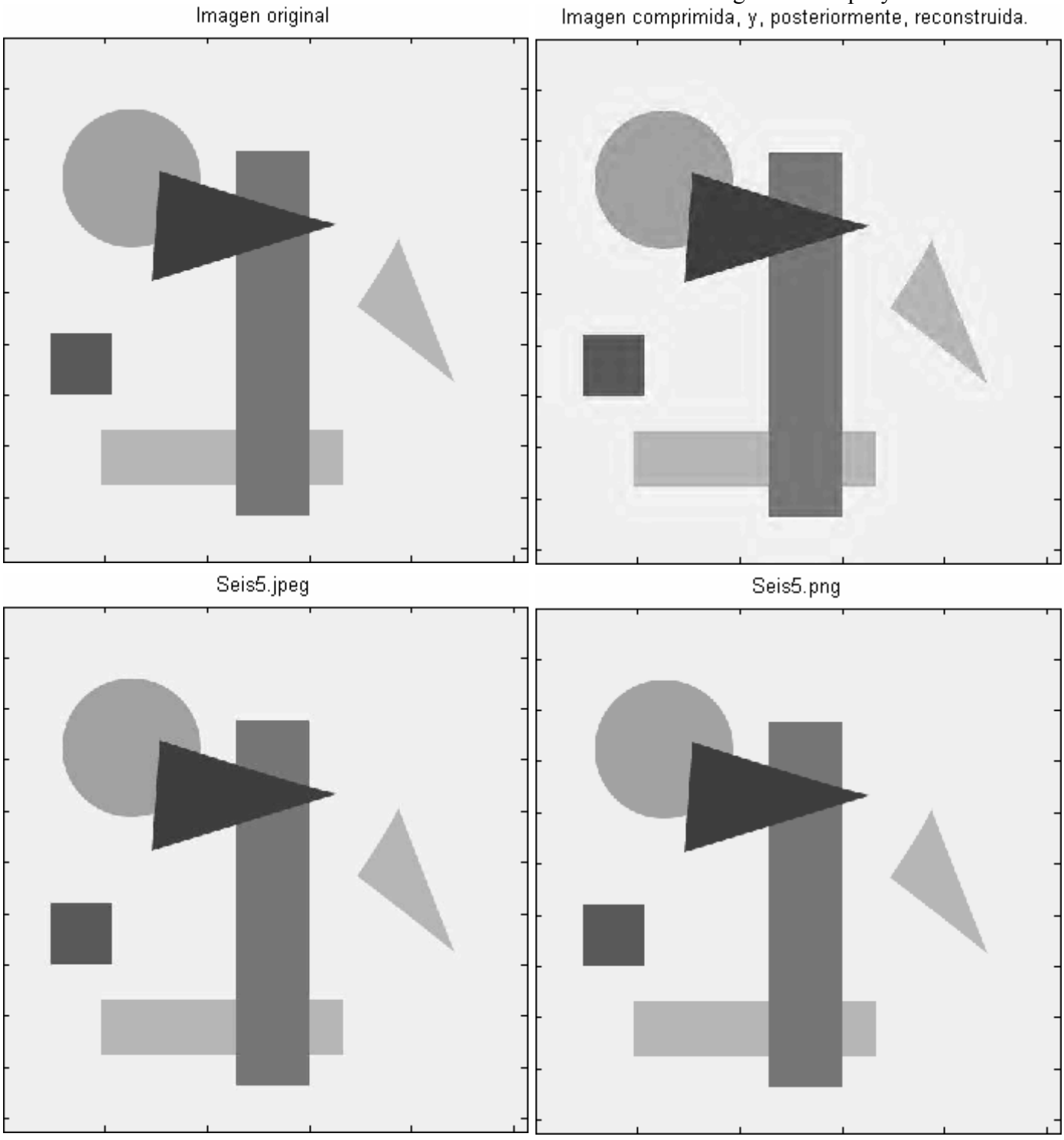
PSNR = Inf SIN PÉRDIDA



Cuánto ocupa esta imagen: 4,47 KB @ 4.580 bytes..

Comparación de resultados:

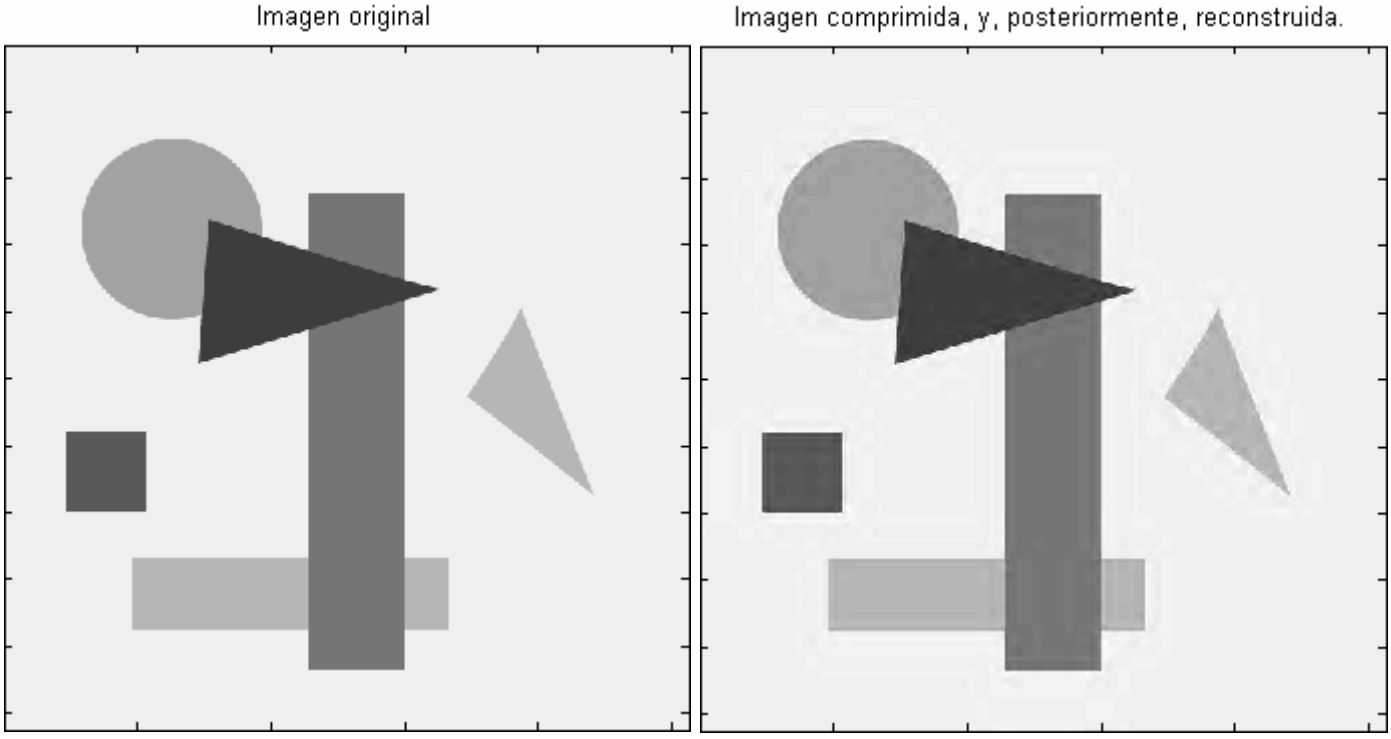
Imagen de este proyecto:



Tolerancia 10. Seis5 (4,10,0):

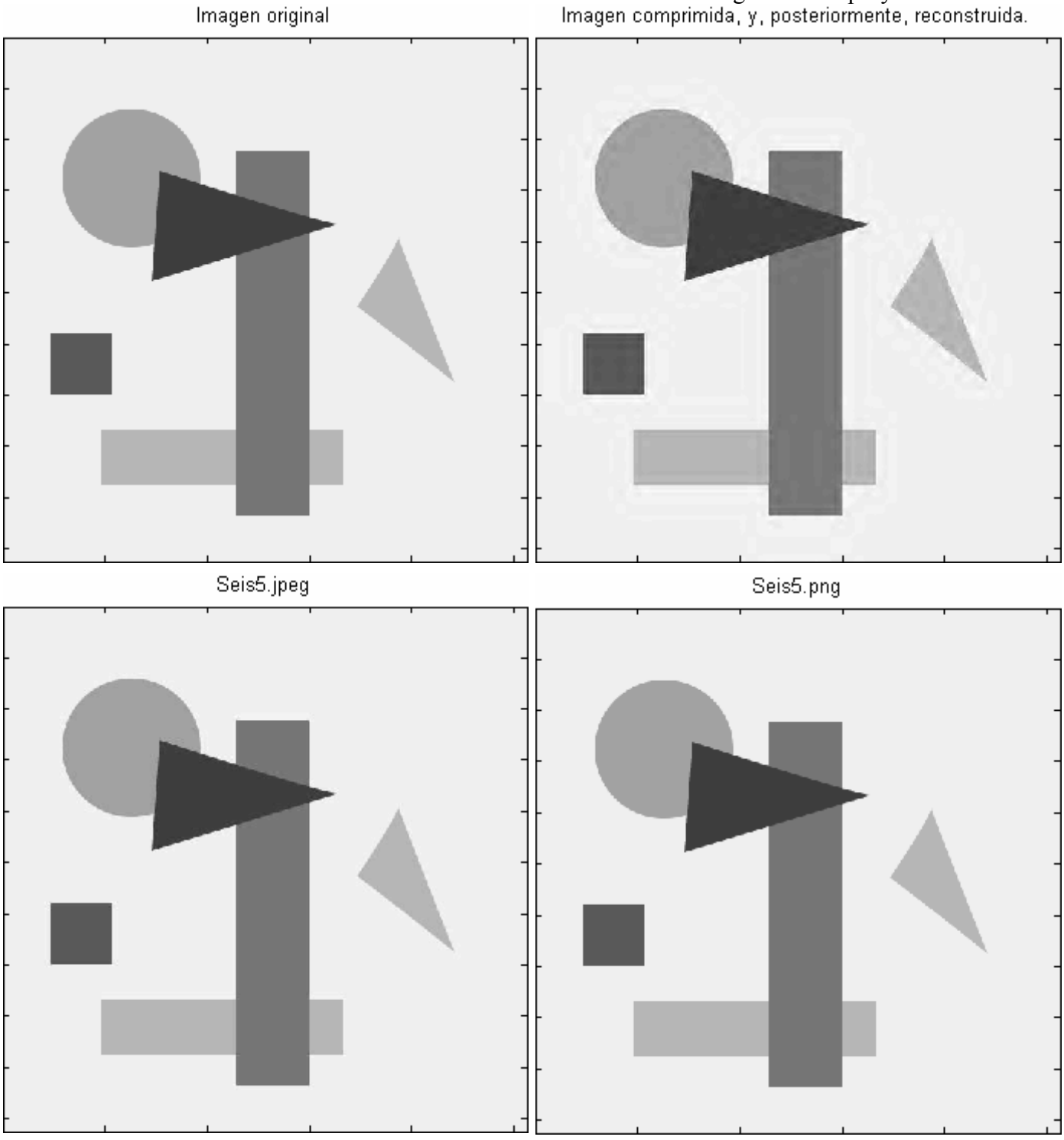
```
>> guionmet2(4,10,0,'seis5.pgm','lin');

PSNR = 40.3305
Dimensión de la imagen = 512 · 512.
Imagen comprimida de este proyecto:
v1.png  (701 bytes).
v2.png  (1,64 KB @ 1.688 bytes).
apos.png  (2,17 KB @ 2.226 bytes).
Total:  4.615 bytes.
```



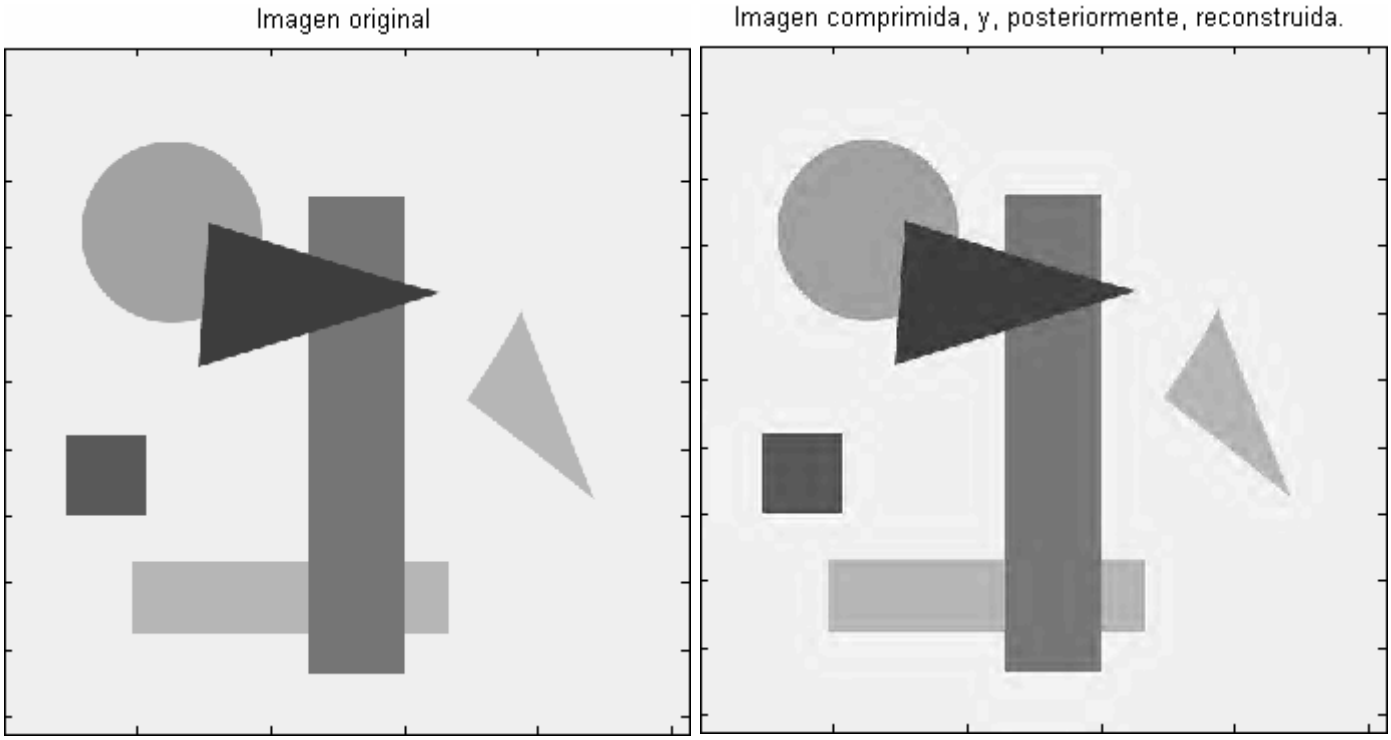
Comparación de resultados:

Imagen de este proyecto:



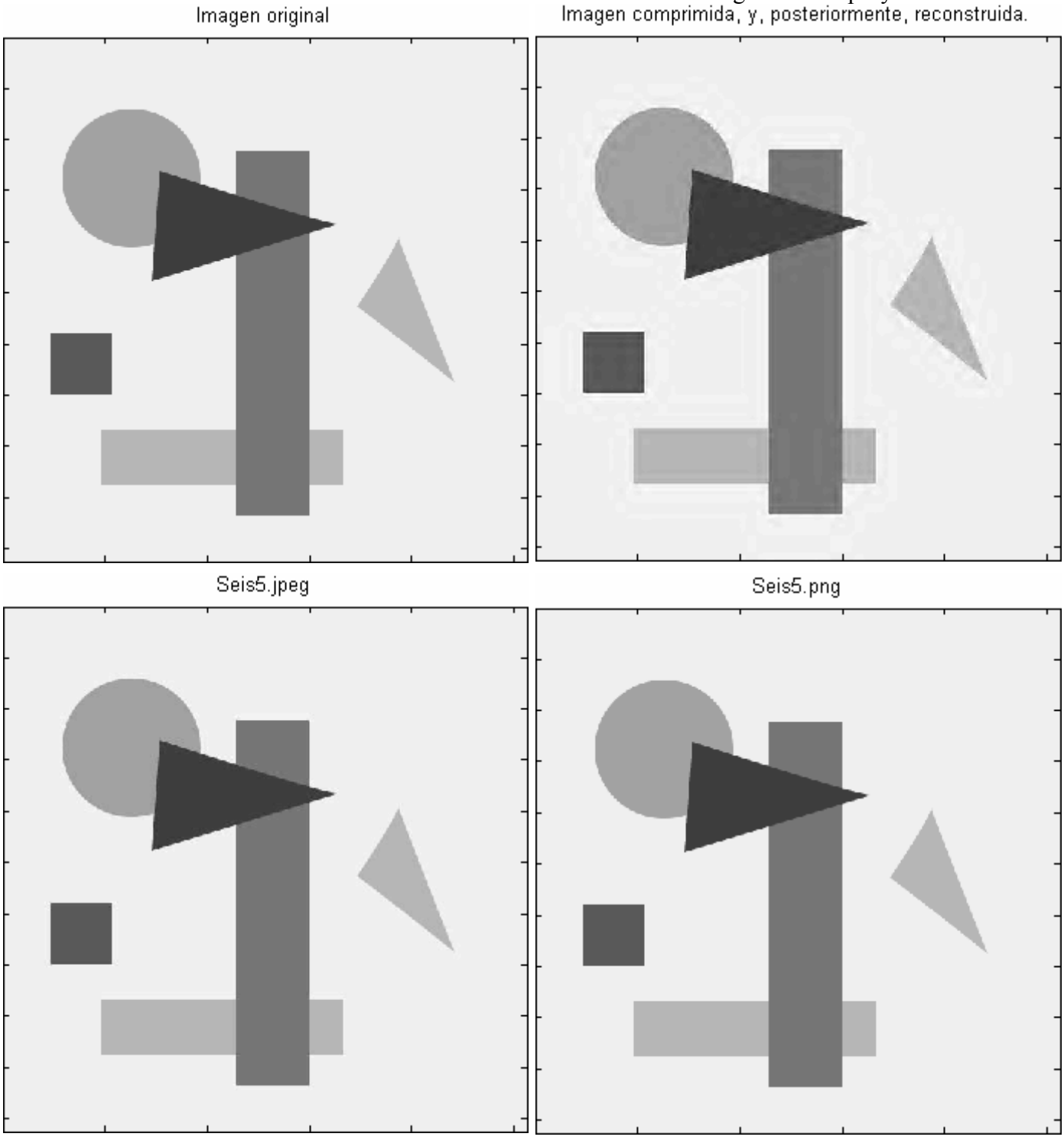
Tolerancia 11. Seis5 (4,11,0):
>> guionmet2(4,11,0,'seis5.pgm','lin');

PSNR = 39.9057
Dimensión de la imagen = 512 · 512.
Imagen comprimida de este proyecto:
v1.png (655 bytes).
v2.png (1,53 KB @ 1.568 bytes).
apos.png (2,04 KB @ 2.092 bytes).
Total: 4.315 bytes.



Comparación de resultados:

Imagen de este proyecto:

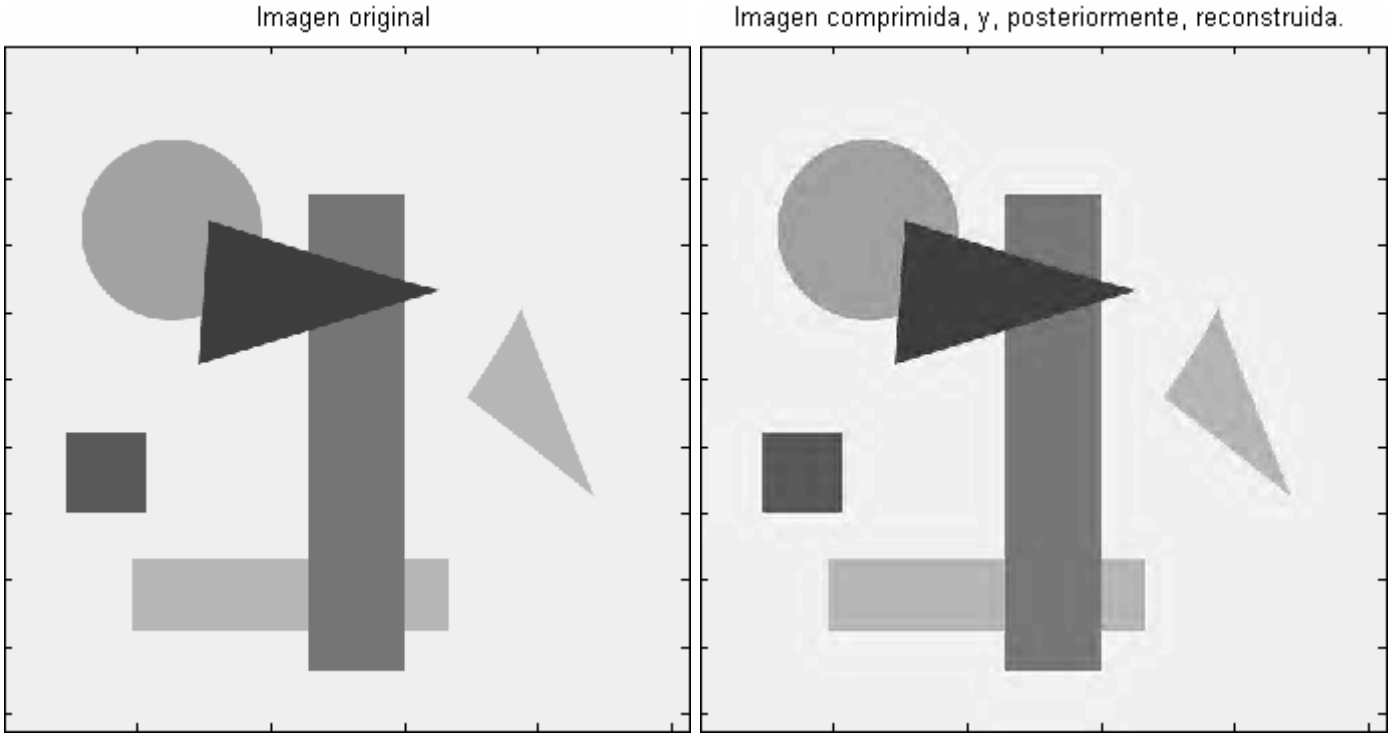


Tolerancia 12. Seis5 (4,12,0):

>> guionmet2(4,12,0,'seis5.pgm','lin');

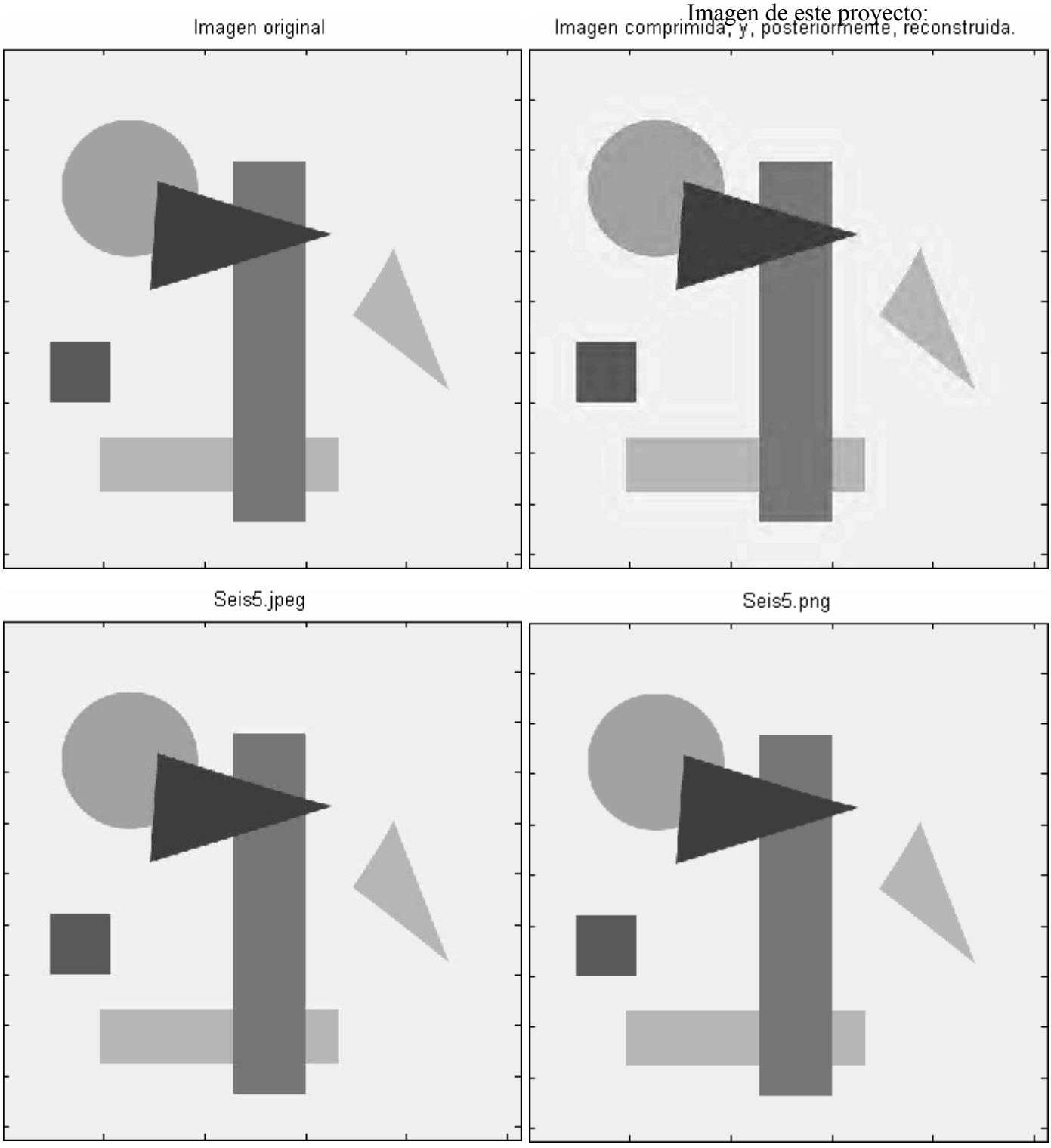
PSNR = 39.7905
Dimensión de la imagen = 512 · 512.
Imagen comprimida de este proyecto:
v1.png (657 bytes).
v2.png (1,47 KB @ 1.511 bytes).
apos.png (1,98 KB @ 2.028 bytes).

Total: 4.196 bytes.



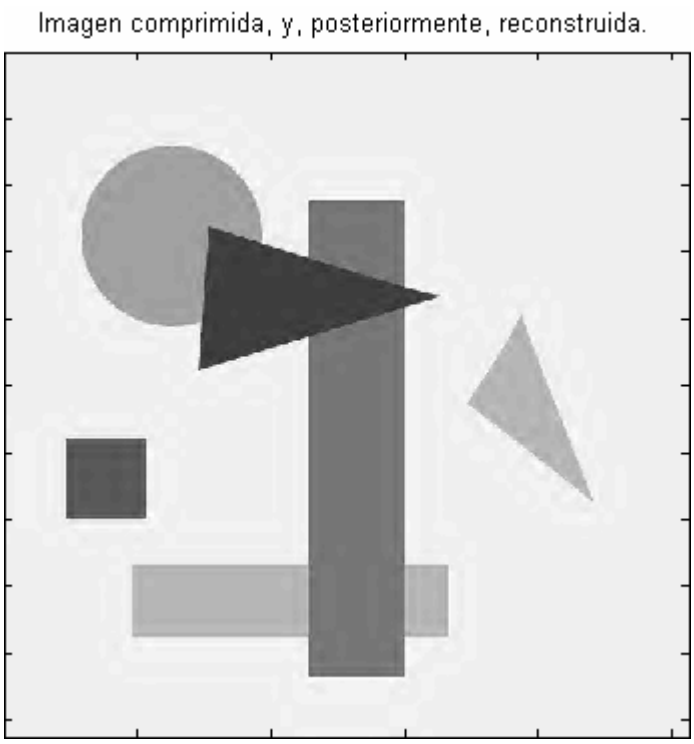
Comparación de resultados:

Imagen de este proyecto:

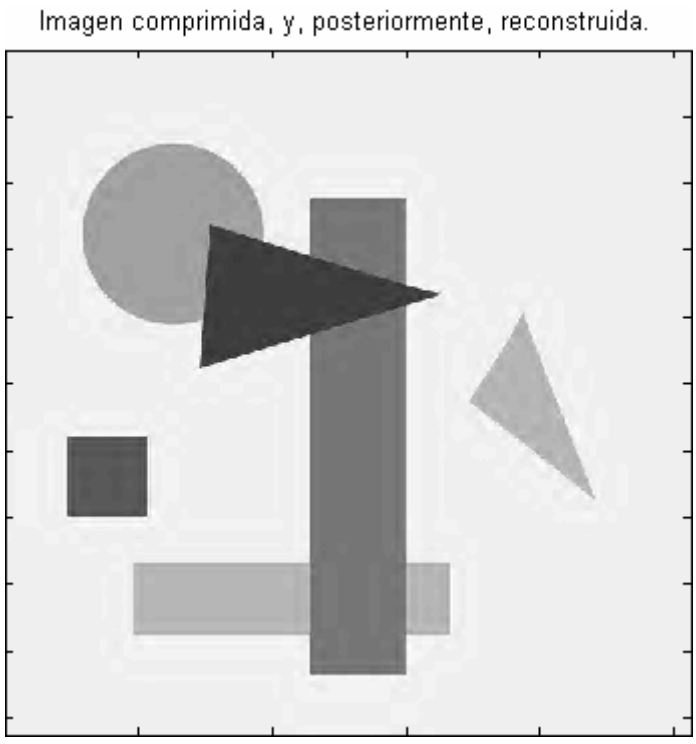


Evolución de la imagen:

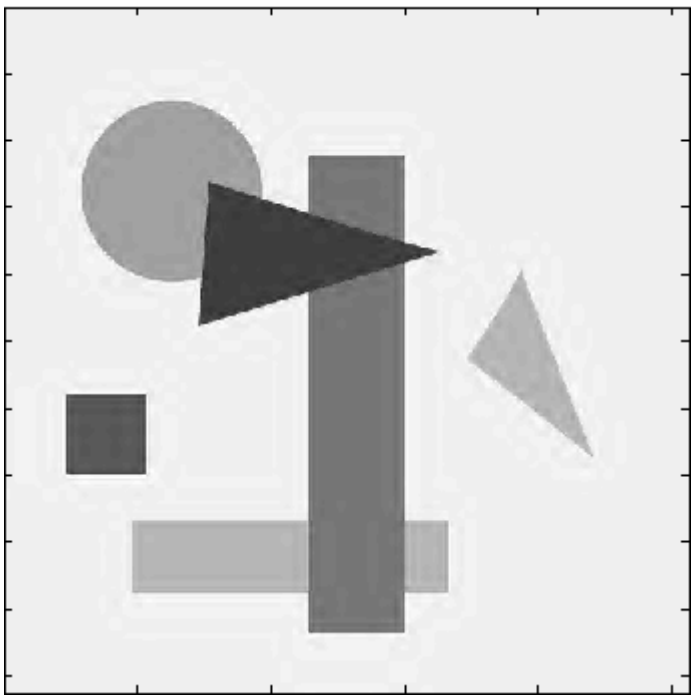
(4,9,0):



(4,10,0):



(4,11,0):



(4,12,0):

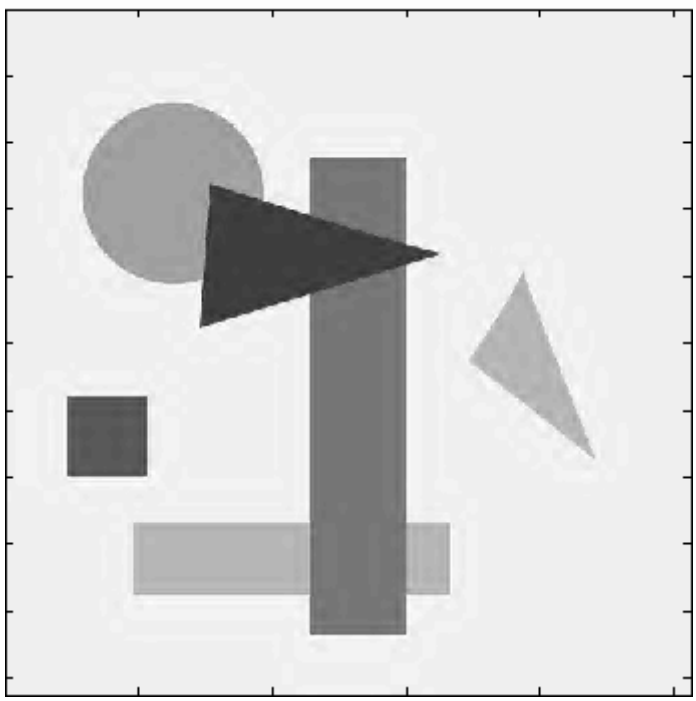
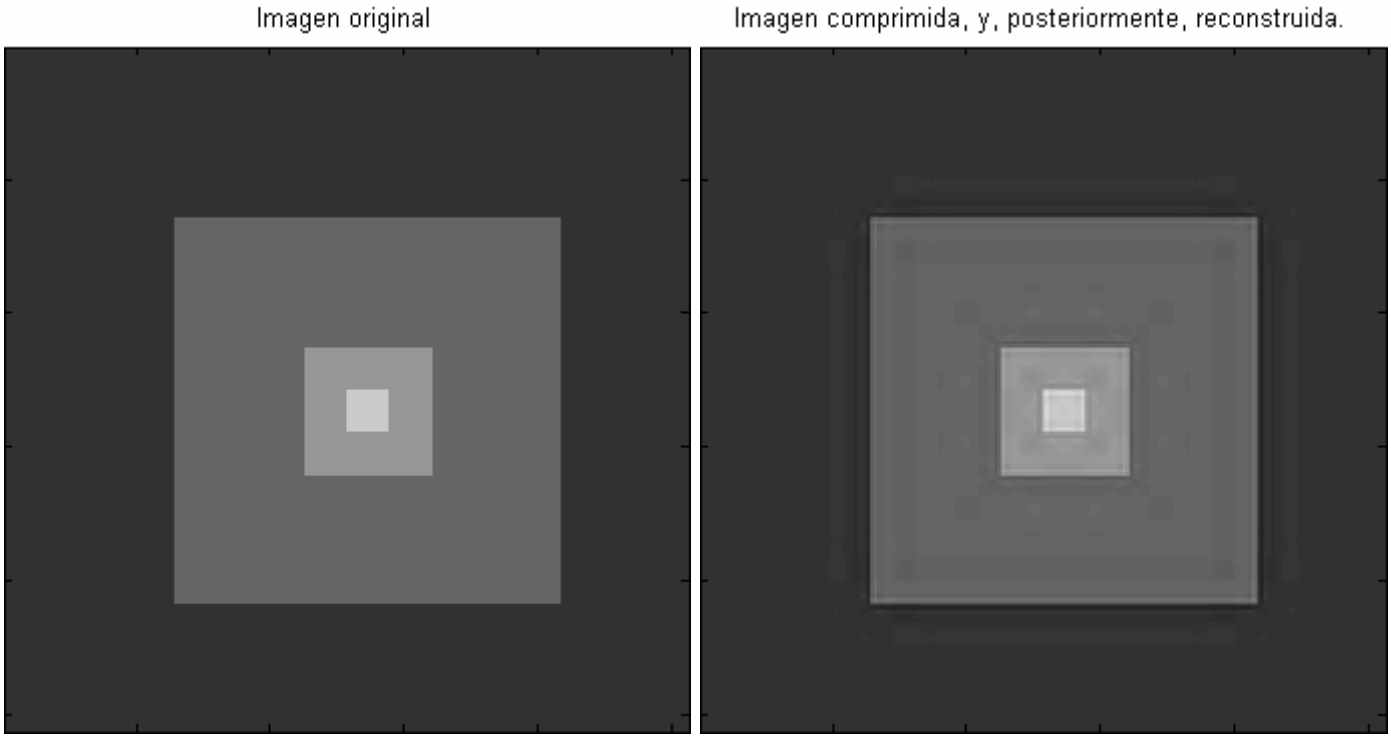


Imagen 3. Squares2.pgm.

Tolerancia 9. Squares2 (4,9,0):
>> guionmet2(4,9,0,'Squares2.pgm','lin');

PSNR = 41.2523
Dimensión de la imagen = 256 · 256.
Imagen comprimida de este proyecto:
v1.png (176 bytes).
v2.png (217 bytes).
apos.png (263 bytes).

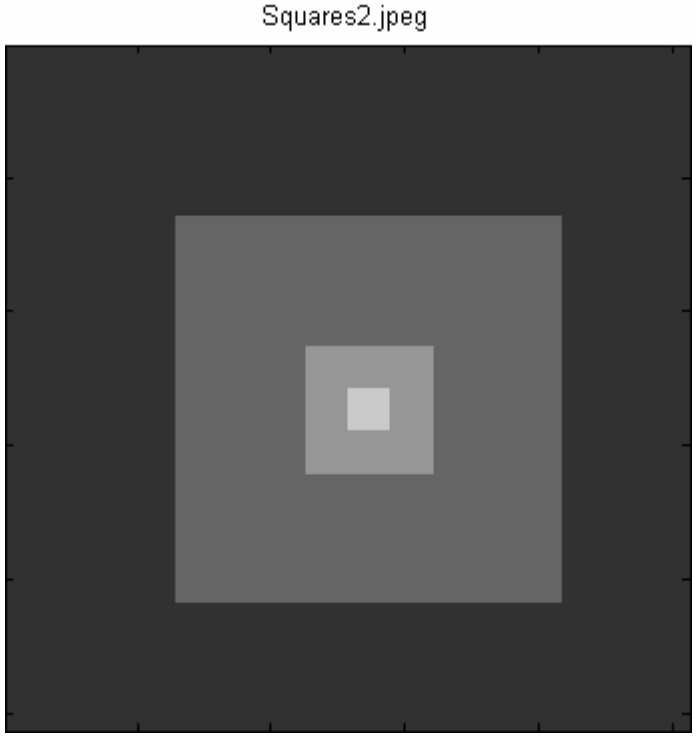
Total: = 656 bytes.



```
>> a1=imread('Squares2.pgm');
>> a2=double(a1);
>> imwrite(a1,'Squares2.jpeg','jpeg')
>> a3=imread('Squares2.jpeg','jpeg');
>> a4=double(a3);
>> psnr(a2,a4);

PSNR = Inf

>> figure
>> imagesc(a3,[0,255])
>> axis('square')
>> colormap gray
>> title('Squares2.jpeg')
```

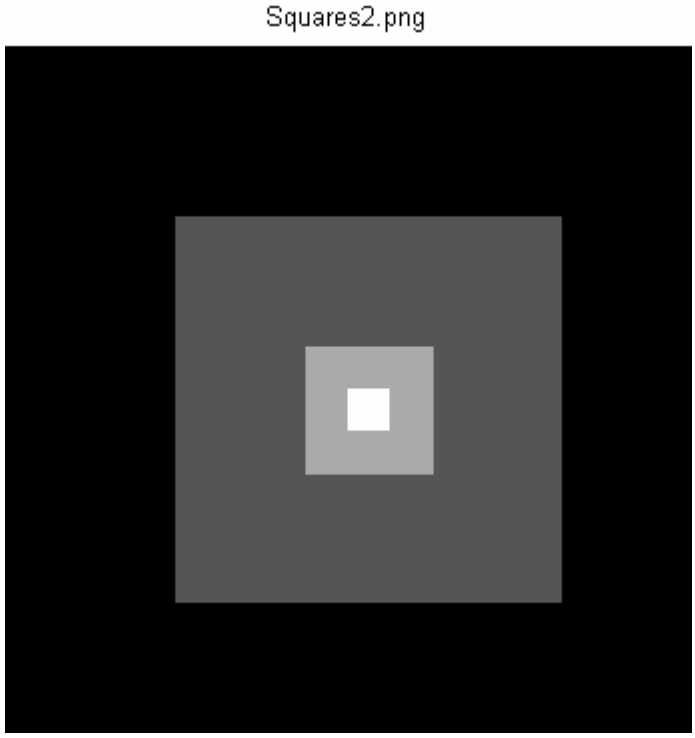


Cuánto ocupa esta imagen: 1,12 KB @ 1.152 bytes.

Comprimida con un parámetro de calidad de 75.

```
>> imwrite(a1,'Squares2.png','png')
>> a5=imread('Squares2.png','png');
>> figure
>> imagesc(a5)
>> colormap gray
>> axis('square')
>> title('Squares2.png')
```

PSNR = Inf SIN PÉRDIDA



Cuánto ocupa esta imagen: 659 bytes.

Comparación de resultados:

Imagen de este proyecto:

Imagen original

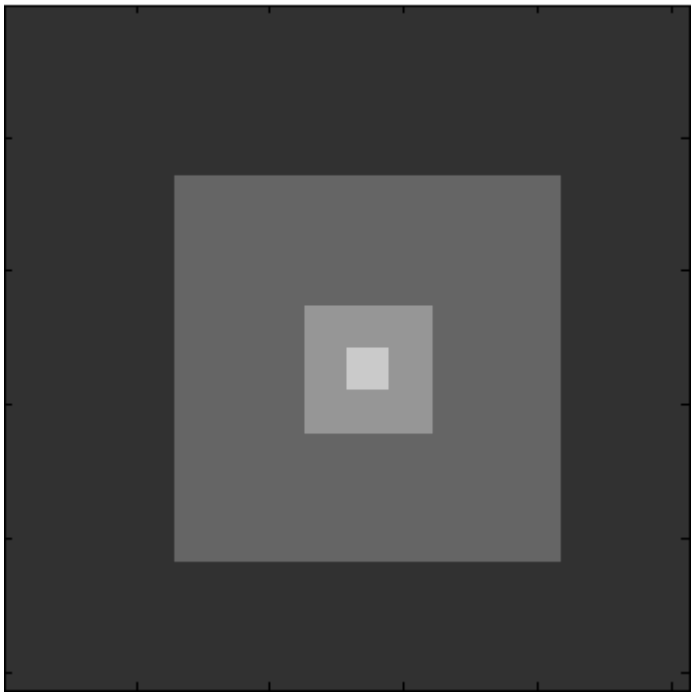
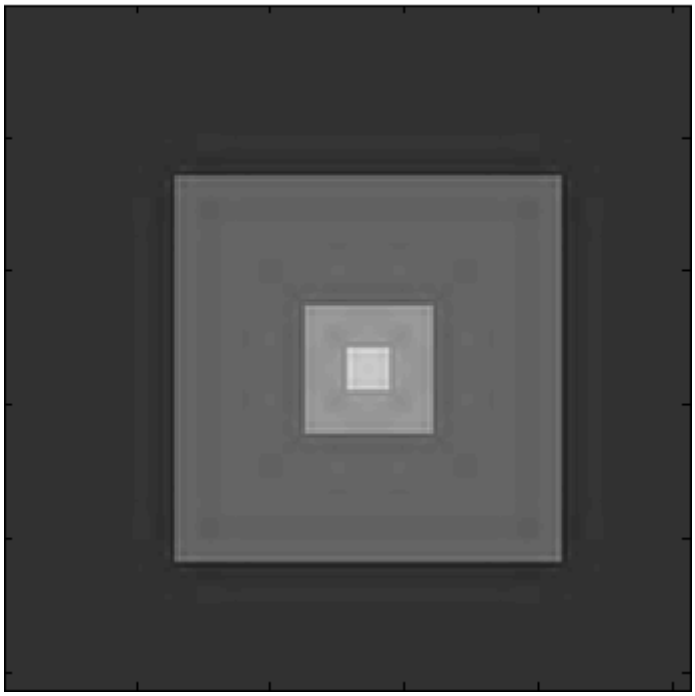
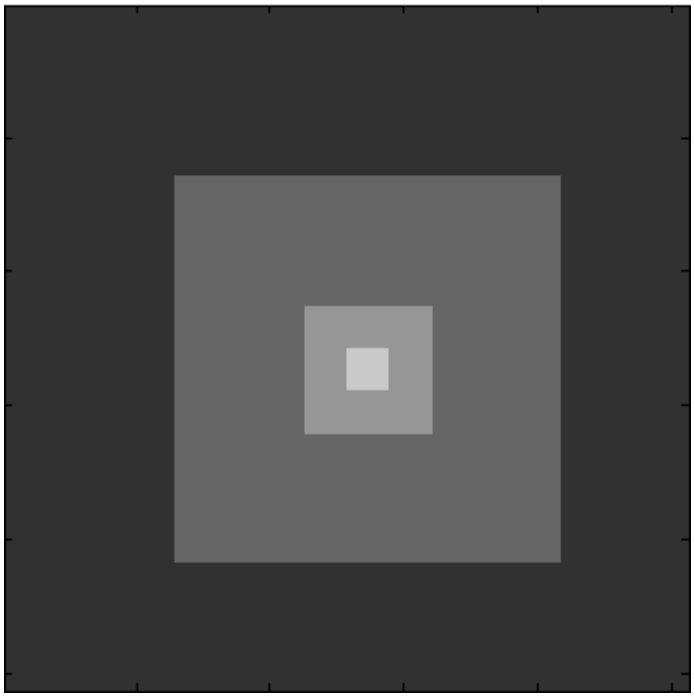


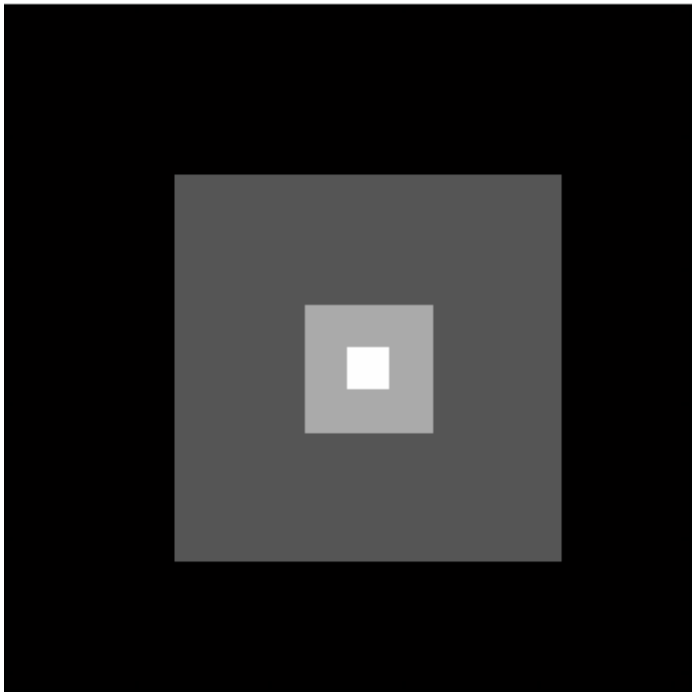
Imagen comprimida, y, posteriormente, reconstruida.



Squares2.jpeg



Squares2.png

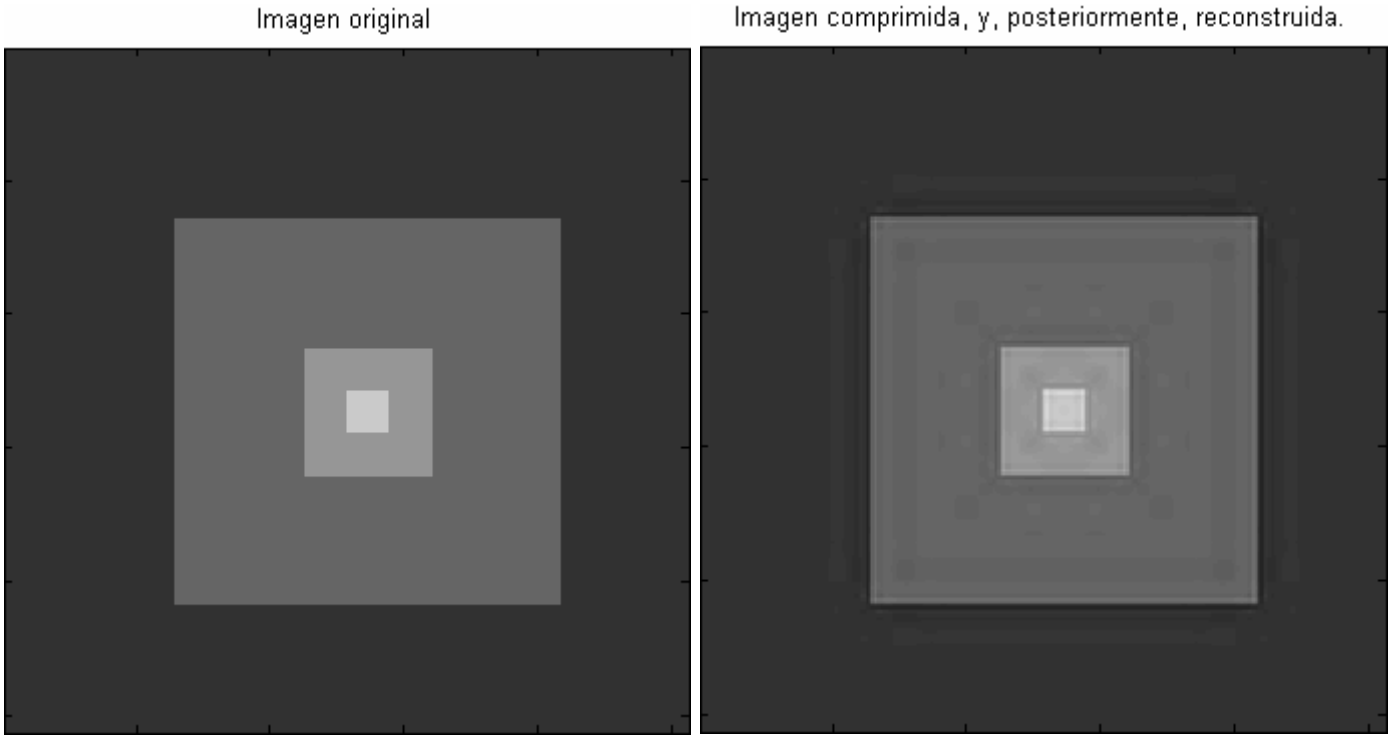


Tolerancia 10. Squares2 (4,10,0):

```
>> guionmet2(4,10,0,'Squares2.pgm','lin');

PSNR = 41.2523
Dimensión de la imagen = 256 · 256.
Imagen comprimida de este proyecto:
v1.png  (176 bytes).
v2.png  (217 bytes).
apos.png  (263 bytes).

Total:   656 bytes.
```



Comparación de resultados:

Imagen de este proyecto:

Imagen original

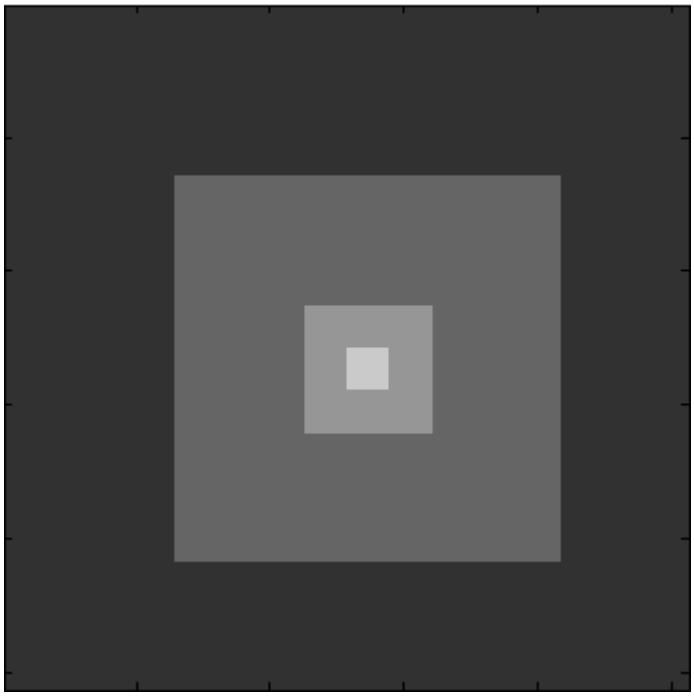
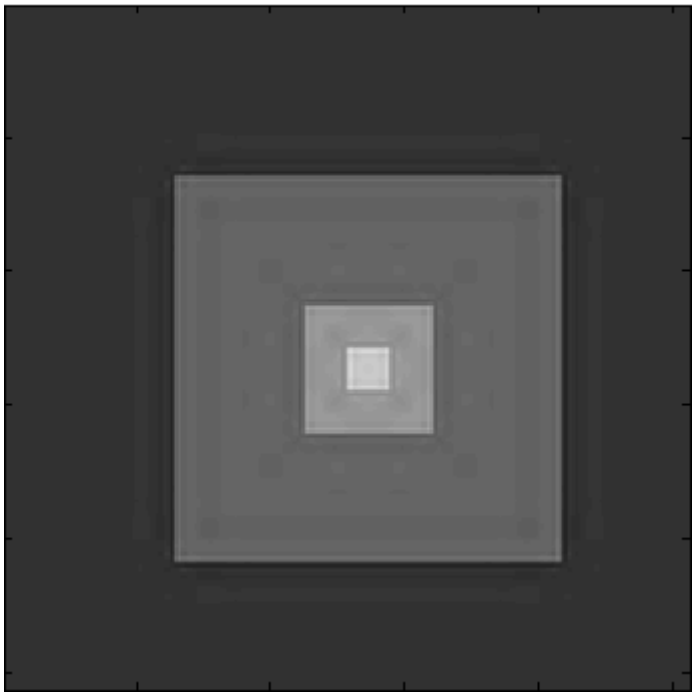
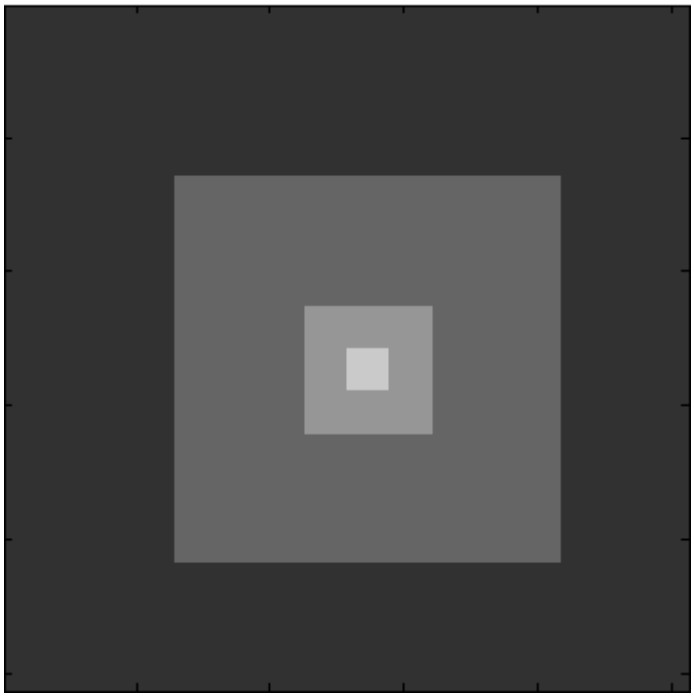


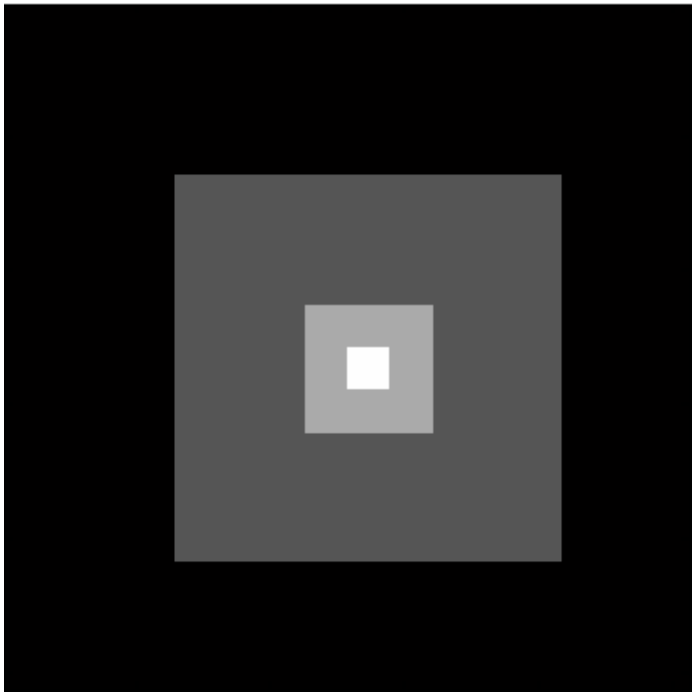
Imagen comprimida, y, posteriormente, reconstruida.



Squares2.jpeg



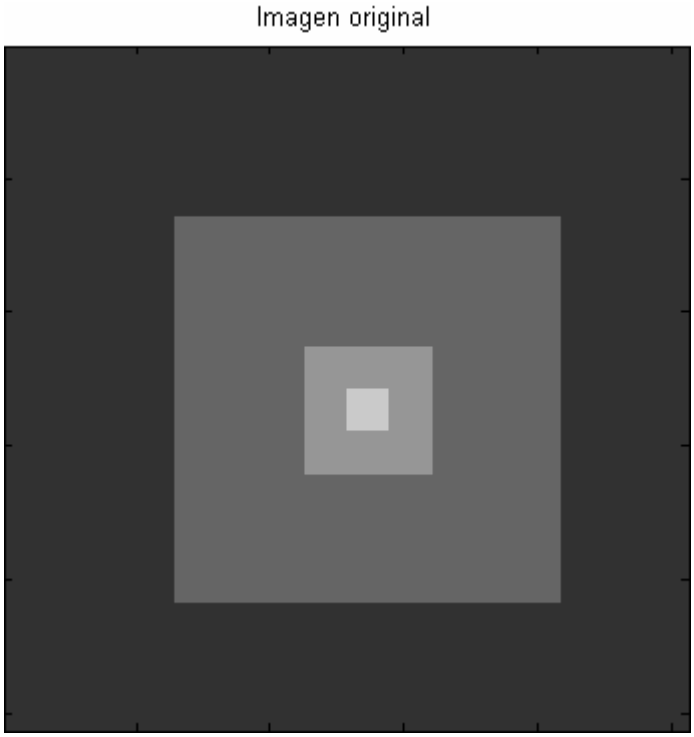
Squares2.png

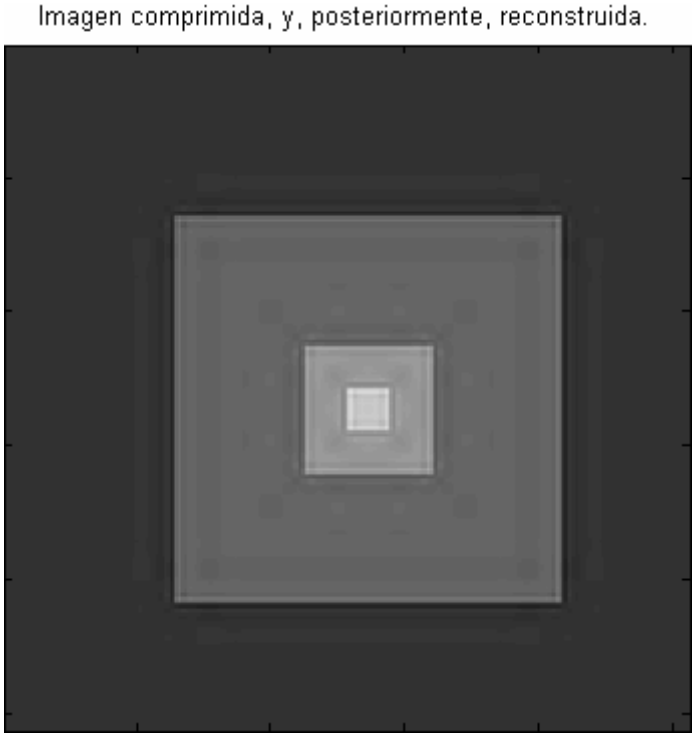


Tolerancia 11. Squares2 (4,11,0):
>> guionmet2(4,11,0,'Squares2.pgm','lin');

PSNR = 39.8679
Dimensión de la imagen = 256 · 256.
Imagen comprimida de este proyecto:
v1.png (176 bytes).
v2.png (219 bytes).
apos.png (263 bytes).

Total: 658 bytes.





Comparación de resultados:

Imagen de este proyecto:

Imagen original

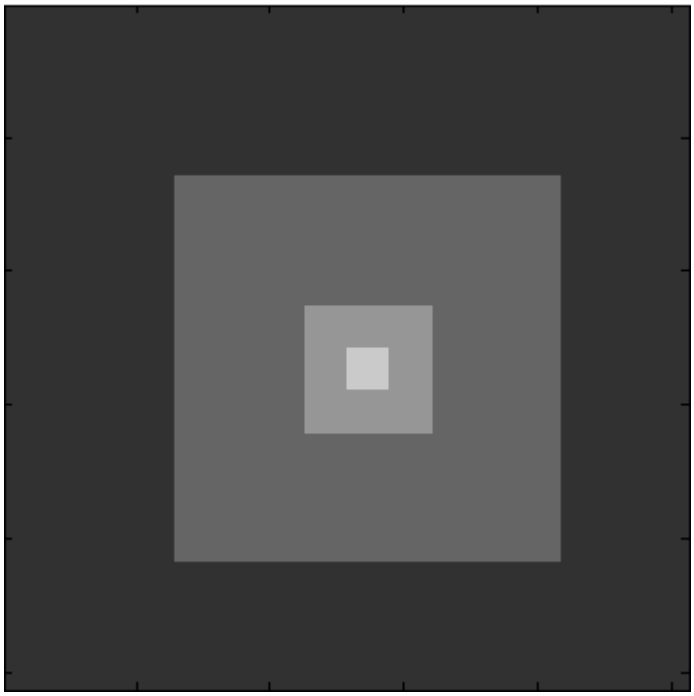
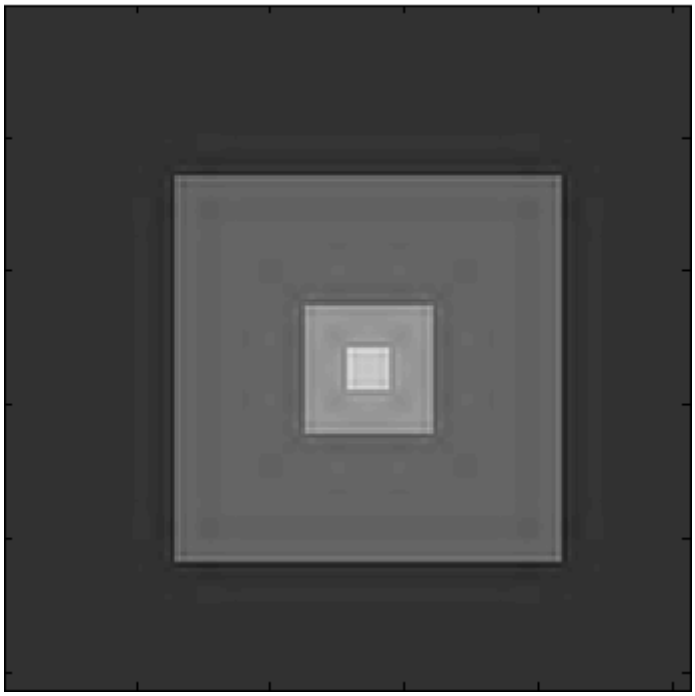
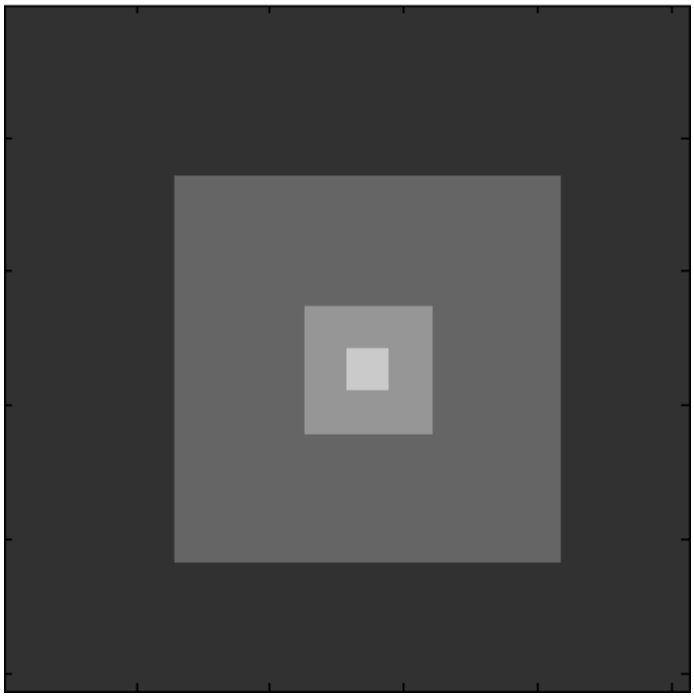


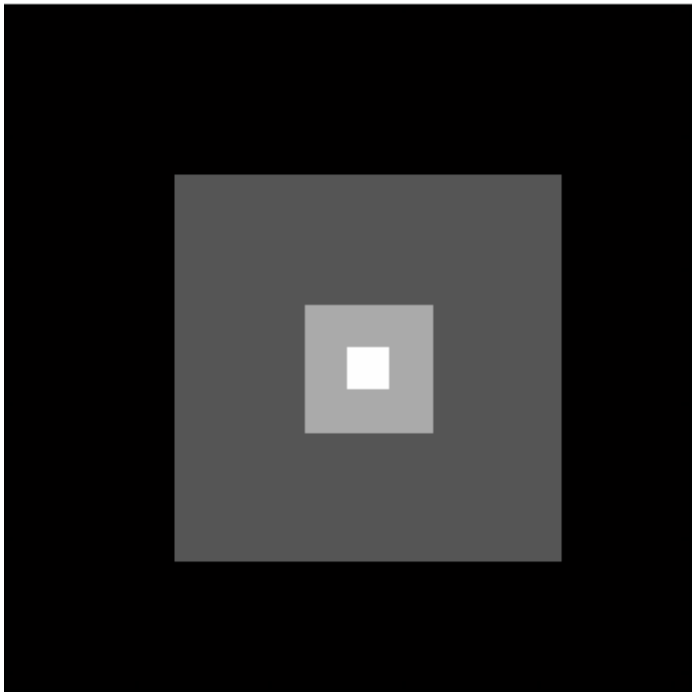
Imagen comprimida, y, posteriormente, reconstruida.



Squares2.jpeg



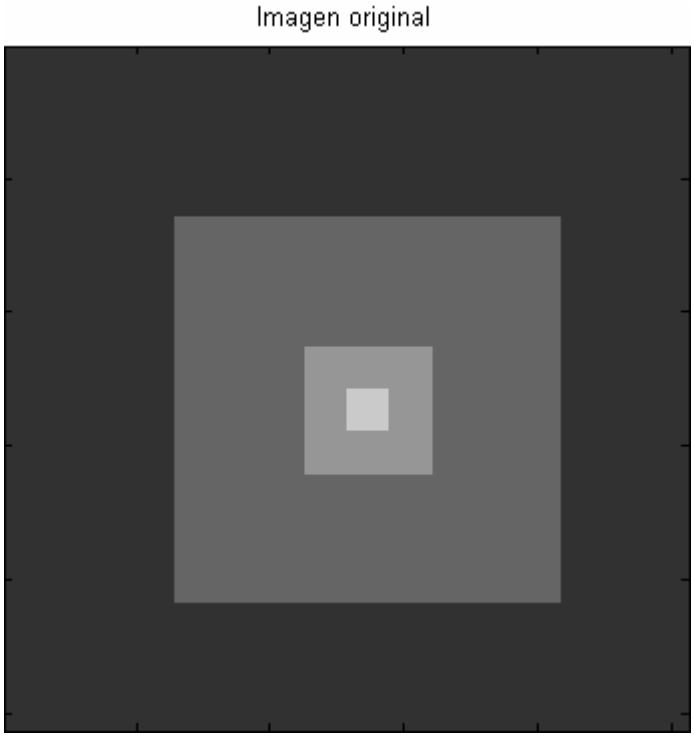
Squares2.png

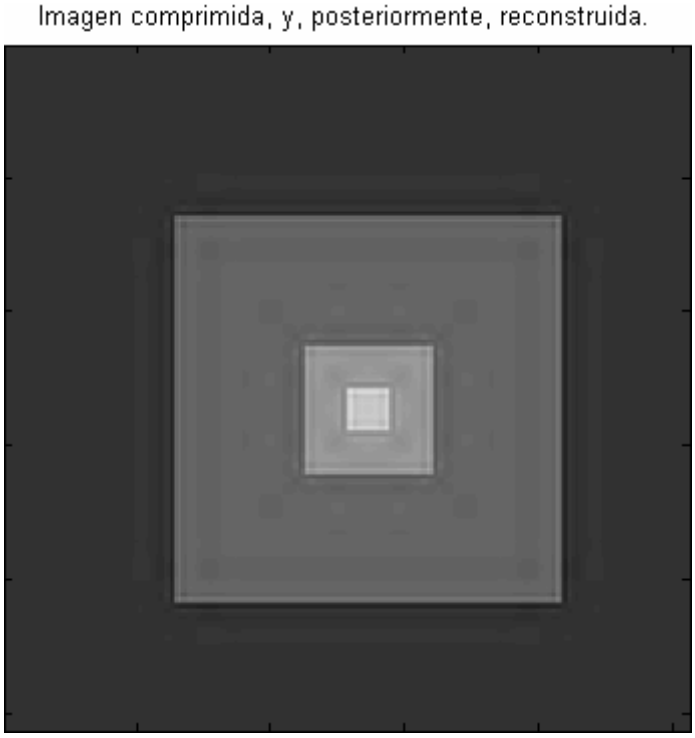


Tolerancia 12. Squares2 (4,12,0):
>> guionmet2(4,12,0,'Squares2.pgm','lin');

PSNR = 39.8679
Dimensión de la imagen = 256 · 256.
Imagen comprimida de este proyecto:
v1.png (176 bytes).
v2.png (219 bytes).
apos.png (263 bytes).

Total: 658 bytes.





Comparación de resultados:

Imagen de este proyecto:

Imagen original

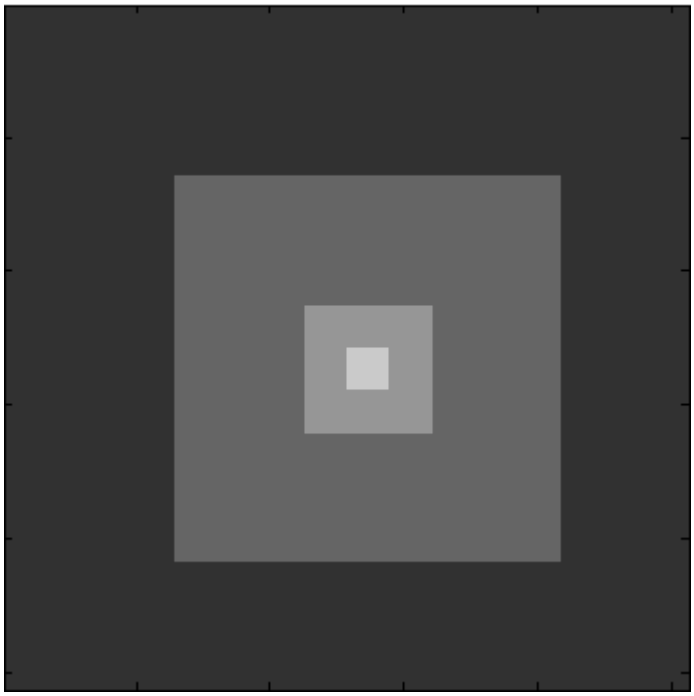
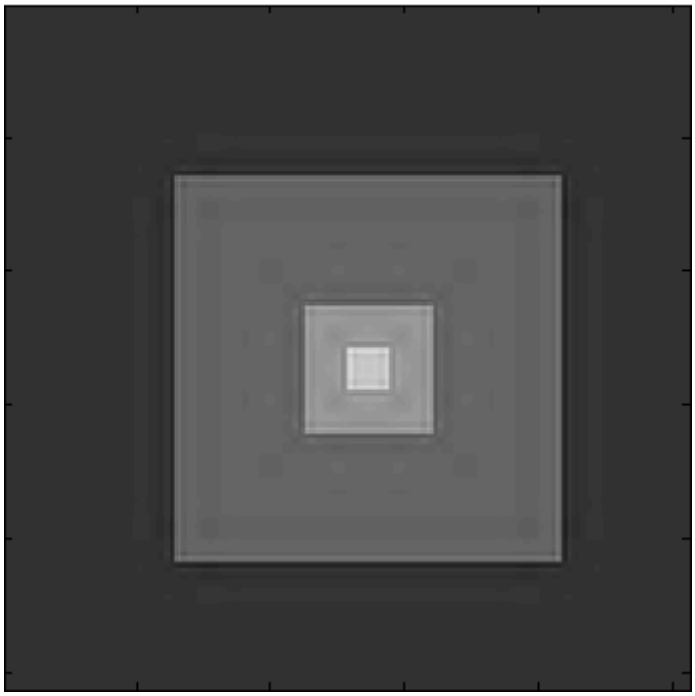
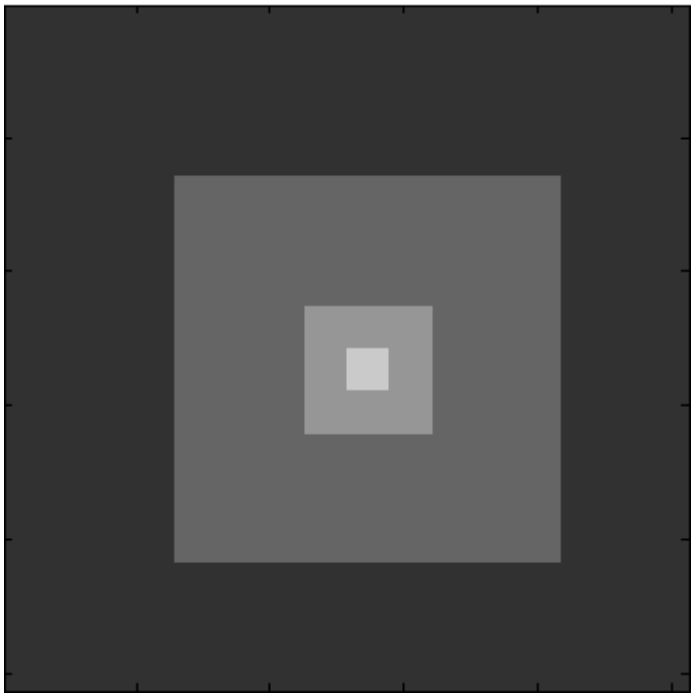


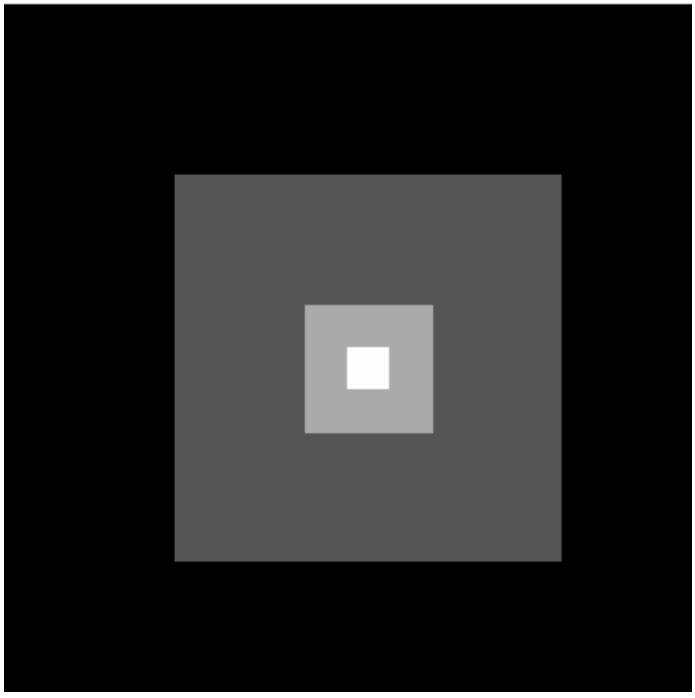
Imagen comprimida, y, posteriormente, reconstruida.



Squares2.jpeg

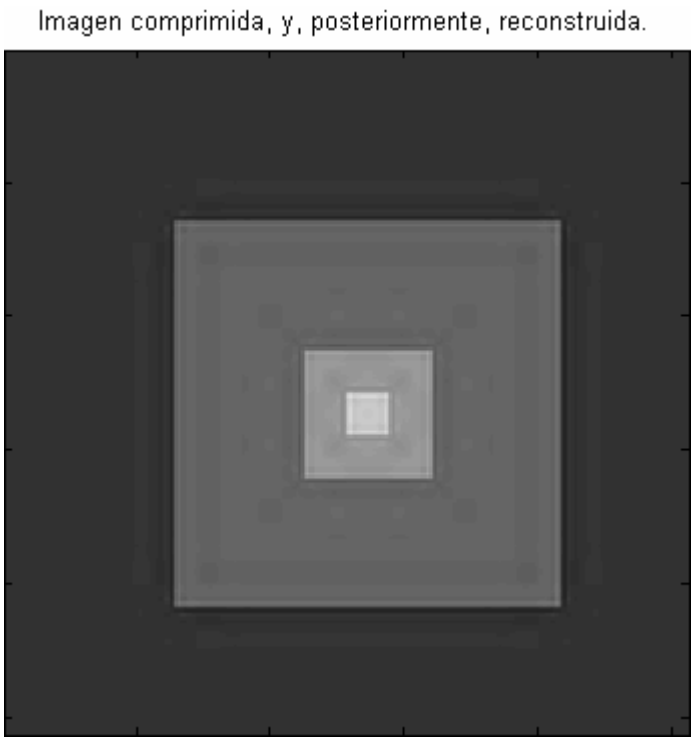


Squares2.png

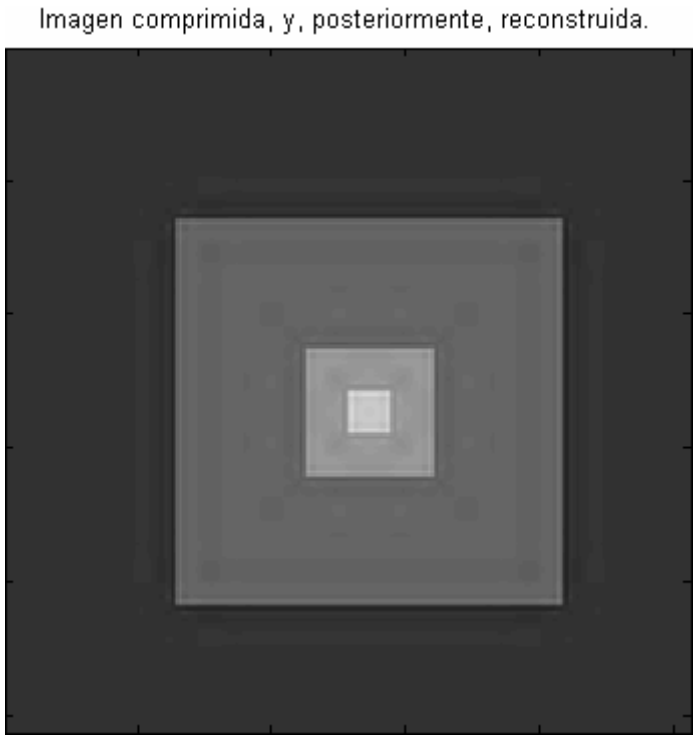


Evolución de la imagen:

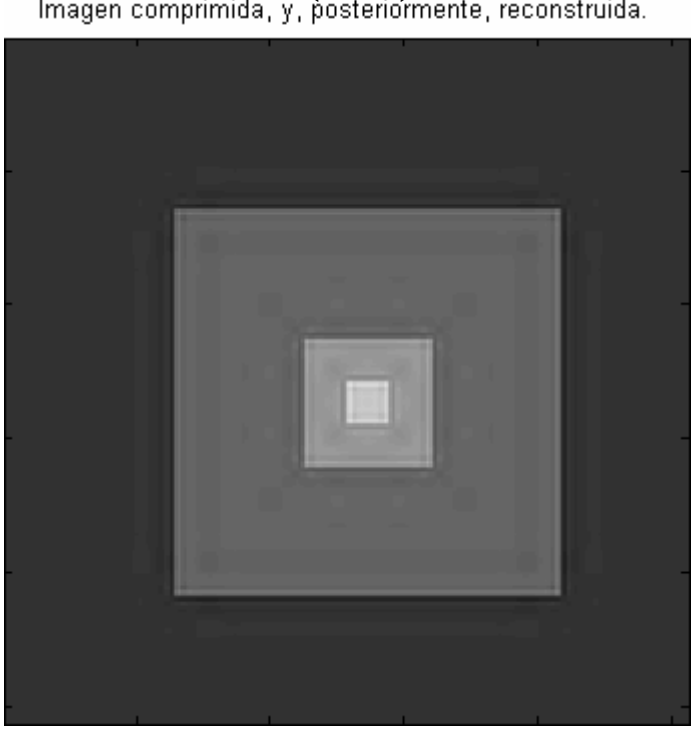
(4,9,0):



(4,10,0):



(4,11,0):



(4,12,0):

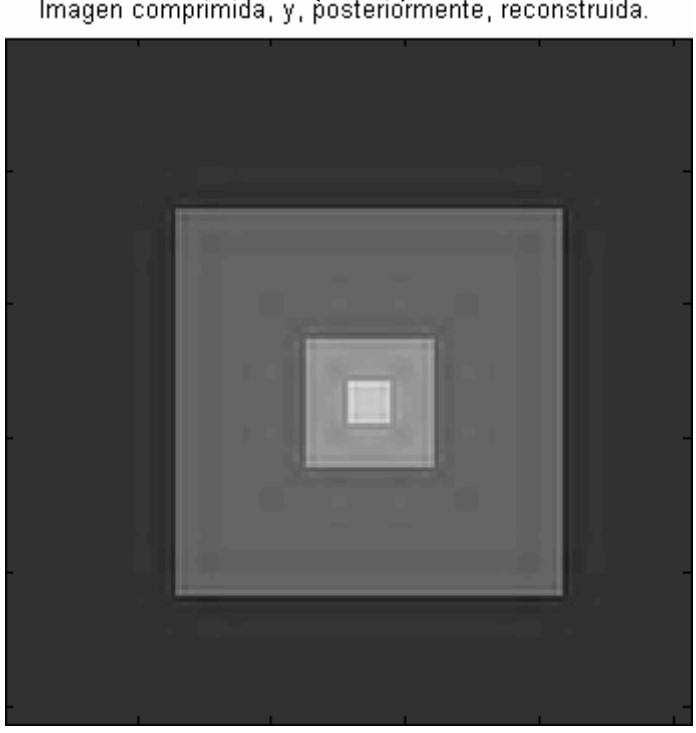


Imagen 4. Boat5.pgm.

Tolerancia 9. Boat5 (4,9,0):
>> guionmet2(4,9,0,'Boat5.pgm','lin');

PSNR = 31.9098
Dimensión de la imagen = 512 · 512.
Imagen comprimida de este proyecto:
v1.png (4,05 KB @ 4.155 bytes).
v2.png (6,97 KB @ 7.144 bytes).
apos.png (10,2 KB @10.500 bytes).

Total: 21.799 bytes.

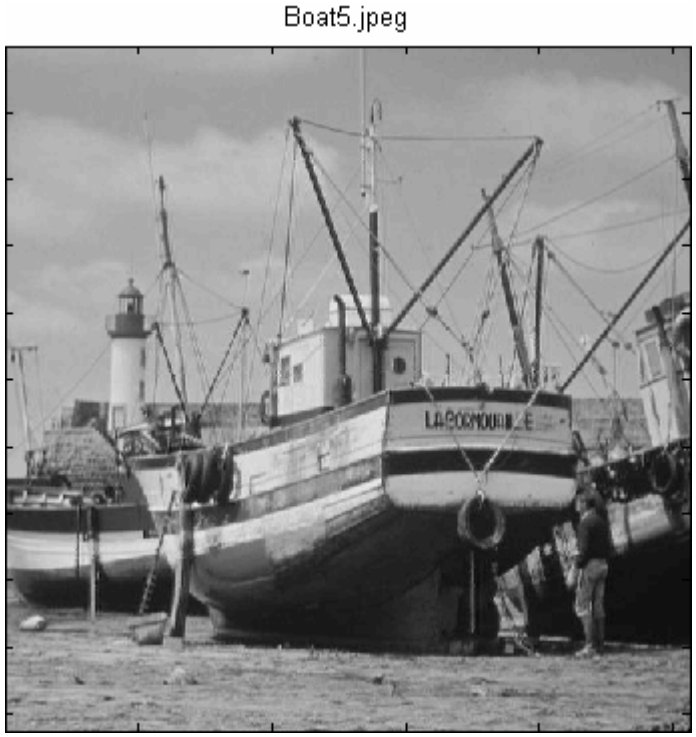
Imagen original

Imagen comprimida, y, posteriormente, reconstruida.

```
>> a1=imread('Boat5.pgm');
>> a2=double(a1);
>> imwrite(a1,'Boat5.jpeg','jpeg')
>> a3=imread('Boat5.jpeg','jpeg');
>> a4=double(a3);
>> psnr(a2,a4);
```

PSNR = 37,2704.

```
>> figure
>> imagesc(a3,[0,255])
>> axis('square')
>> colormap gray
>> title('Boat5.jpeg')
```

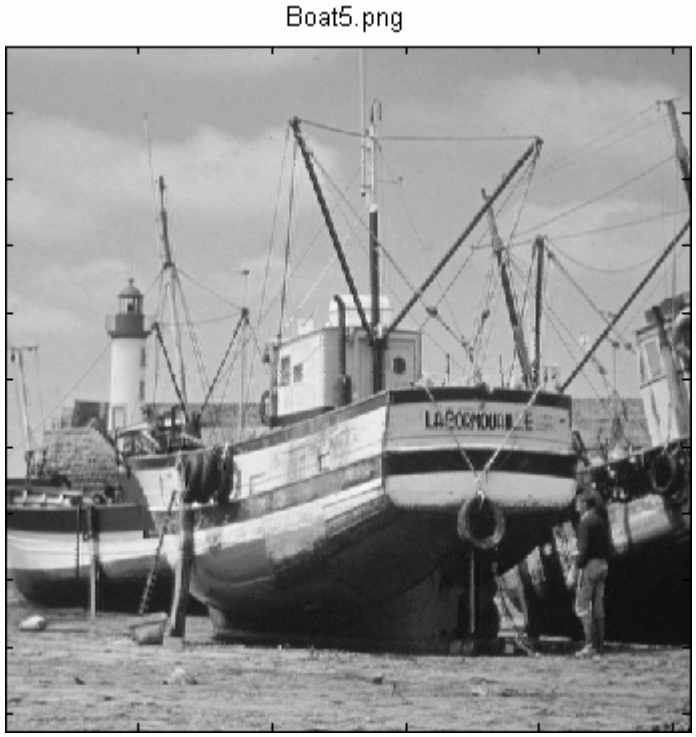


Cuánto ocupa esta imagen: 35,4 KB @ 36.272 bytes.

Comprimida con un parámetro de calidad de 75.

```
>> imwrite(a1,'Boat5.png','png')
>> a5=imread('Boat5.png','png');
>> figure
>> imagesc(a5)
>> colormap gray
>> axis('square')
>> title('Boat5.png')
```

PSNR = Inf SIN PÉRDIDA



Cuánto ocupa esta imagen: 148 KB @ 151.933 bytes.

Comparación de resultados:

Imagen original

Imagen de este proyecto:
Imagen comprimida, y, posteriormente, reconstruida.

Boat5.jpeg

Boat5.png

Tolerancia 10. Boat5 (4,10,0):

```
>> guionmet2(4,10,0,'Boat5.pgm','lin');
```

PSNR = 31.6993
Dimensión de la imagen = 512 · 512.
Imagen comprimida de este proyecto:
v1.png (3,83 KB @ 3.927 bytes).
v2.png (6,70 KB @ 6.869 bytes).
apos.png (9,62 KB @ 9.860 bytes).

Total: 20.656 bytes.

Imagen original

Imagen comprimida, y, posteriormente, reconstruida.

Comparación de resultados:

Imagen original

Imagen de este proyecto:
Imagen comprimida, y, posteriormente, reconstruida.

Boat5.jpeg

Boat5.png

Tolerancia 11. Boat5 (4,11,0):
>> guionmet2(4,11,0,'Boat5.pgm','lin');

PSNR = 31.1469
Dimensión de la imagen = 512 · 512.
Imagen comprimida de este proyecto:
v1.png (3,83 KB @ 3.927 bytes).
v2.png (6,70 KB @ 6.869 bytes).
apos.png (8,73 KB @ 8.942 bytes).

Total: 19.738 bytes.

Imagen original

Imagen comprimida, y, posteriormente, reconstruida.

Comparación de resultados:

Imagen original

Imagen de este proyecto:
Imagen comprimida, y, posteriormente, reconstruida.

Boat5.jpeg

Boat5.png

Tolerancia 12. Boat5 (4,12,0):
>> guionmet2(4,12,0,'Boat5.pgm','lin');

PSNR = 30.9921
Dimensión de la imagen = 512 · 512.
Imagen comprimida de este proyecto:
v1.png (3,16 KB @ 3.241 bytes).
v2.png (5,71 KB @ 5.857 bytes).
apos.png (8,29 KB @ 8.499 bytes).

Total: 17.597 bytes.

Imagen original

Imagen comprimida, y, posteriormente, reconstruida.

Comparación de resultados:

Imagen original

Imagen de este proyecto:
Imagen comprimida, y, posteriormente, reconstruida.

Boat5.jpeg

Boat5.png

Evolución de la imagen:

(4,9,0):

Imagen comprimida, y, posteriormente, reconstruida.

(4,10,0):

Imagen comprimida, y, posteriormente, reconstruida.

(4,11,0):

Imagen comprimida, y, posteriormente, reconstruida.

(4,12,0):

Imagen comprimida, y, posteriormente, reconstruida.

A continuación realizamos tests con ruido blanco. Para ello introduciremos ruido blanco en las imágenes consideradas y correremos los algoritmos tal y como hicimos previamente en la anterior batería de experimentos. En el algoritmo de multirresolución utilizaremos tolerancias tol=11, 12. Hemos adaptado los programas "guionmet2.m", "compresor.m" y "descenderc.m" para poder pasar como argumento directamente una matriz, y así pasar la imagen con ruido. A los nuevos programas los llamamos "guionmet2m.m " , "compresorm.m" y "descendercm.m" .

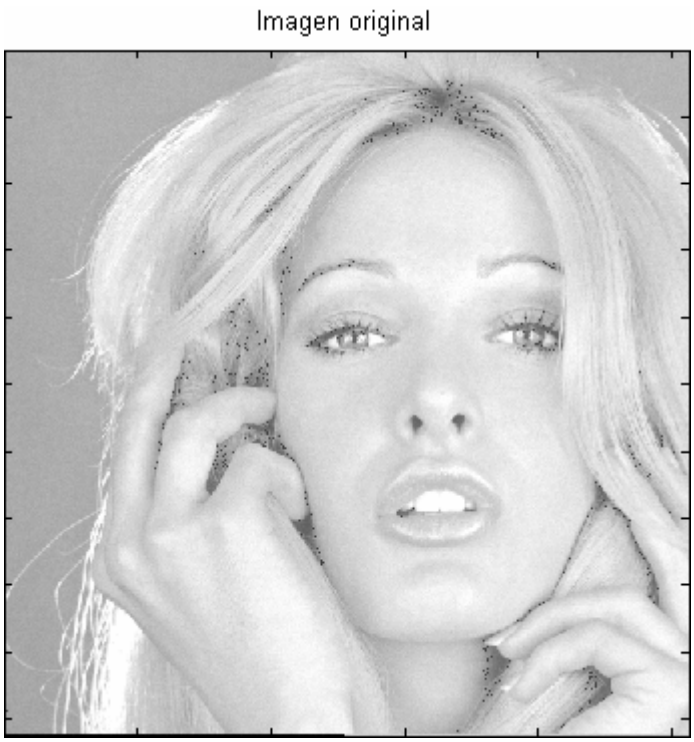
Imagen 1 + ruido blanco. Tiffany5.pgm + ruido blanco.

Tolerancia 11. Tiffany5 (4,11,8)=(l, tol, ruido):

```
>> a=imread('tiffany5.pgm');                % leemos la imagen tiffany5.pgm
>> ruido=8; n=max(size(a));                 % introducimos el ruido en la imagen
>> a=double(a); a=a+ruido*rand(n);
>> a=round(a);
>> a=((a>=0)&(a<=255)).*a+(a>255)*255;
>> guionmet2m(4,11,a,'lin');                % comprimimos con nuestro algoritmo
```

PSNR = 32.5545
Dimensión de la imagen = 512 · 512.
v1.png (1,94 KB = 1,94 Kbytes @ 1.993 bytes).
v2.png (3,50 KB @ 3.588 bytes).
apos.png (7,01 KB @ 7.186 bytes).

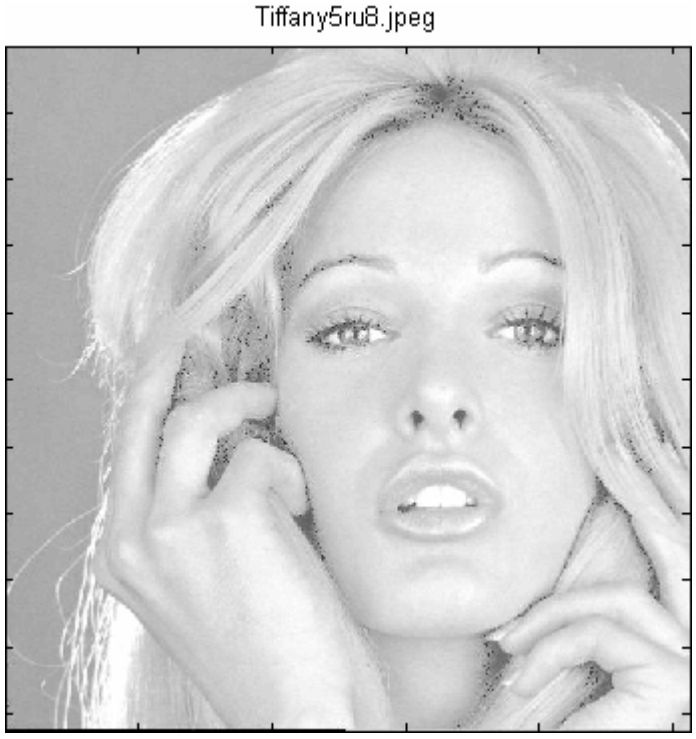
Total: 1.993 + 3.588 + 7.186 = 12.767 bytes (es lo que ocupa la imagen comprimida).



```
>> a2=a; a=uint8(a);
>> imwrite(a,'Tiffany5ru8.jpeg','jpeg')
>> a3=imread('Tiffany5ru8.jpeg','jpeg');
>> a4=double(a3);
>> psnr(a2,a4);
```

PSNR = 35,6922.

```
>> figure
>> imagesc(a3,[0,255])
>> axis('square')
>> colormap gray
>> title('Tiffany5ru8.jpeg')
```

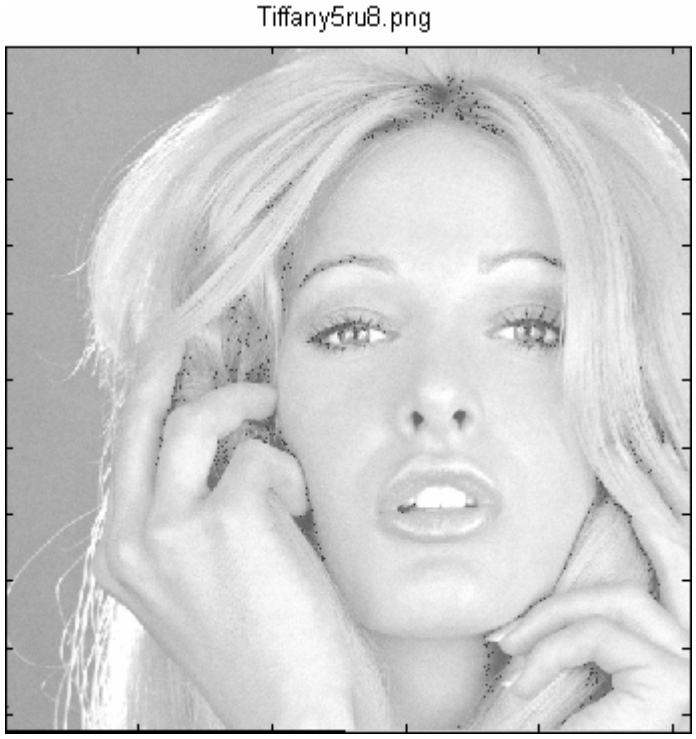


Cuánto ocupa esta imagen: 32,0 KB @ 32.854 bytes.

Comprimida con un parámetro de calidad de 75 de un rango de posibles valores en el intervalo [0, 100].

```
>> imwrite(a,'Tiffany5ru8.png','png')
>> a5=imread('Tiffany5ru8.png','png');
>> figure
>> imagesc(a5)
>> colormap gray
>> axis('square')
>> title('Tiffany5ru8.jpeg')
```

PSNR = Inf SIN PÉRDIDA



Cuánto ocupa esta imagen: 150 KB @154.198 bytes.

Comparación de resultados:

Imagen de este proyecto:

Imagen original

Imagen comprimida, y, posteriormente, reconstruida.

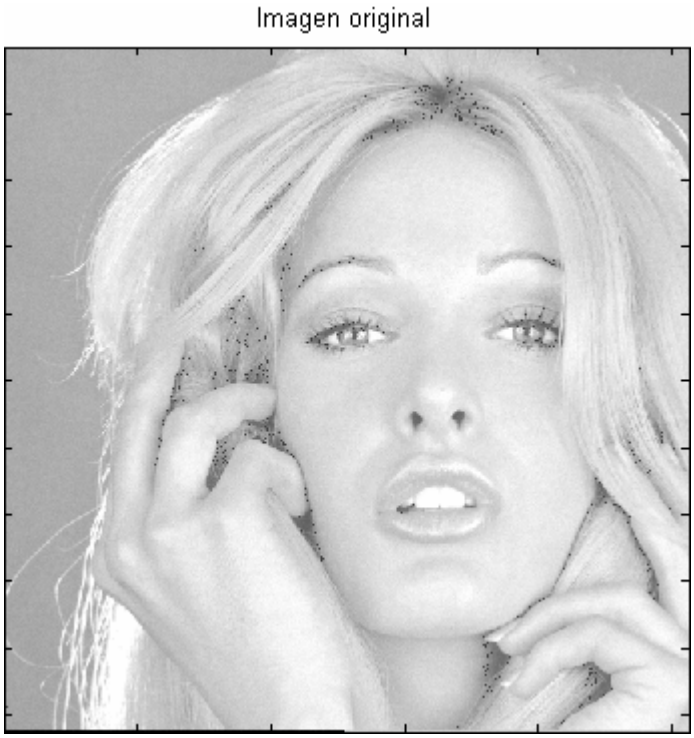
Tiffany5ru8.jpeg

Tiffany5ru8.png

Tolerancia 12. Tiffany5 (4,12,8):
>> guionmet2m(4,12,a,'lin');

PSNR = 32.4562
Dimensión de la imagen = 512 · 512.
v1.png (1,88 KB @ 1.926 bytes).
v2.png (3,33 KB @ 3.420 bytes).
apos.png (6,79 KB @ 6.954 bytes).

Total = 12.300 bytes (es lo que ocupa la imagen comprimida).



Comparación de resultados:

Imagen de este proyecto:

Imagen original

Imagen comprimida, y, posteriormente, reconstruida.

Tiffany5ru8.jpeg

Tiffany5ru8.png

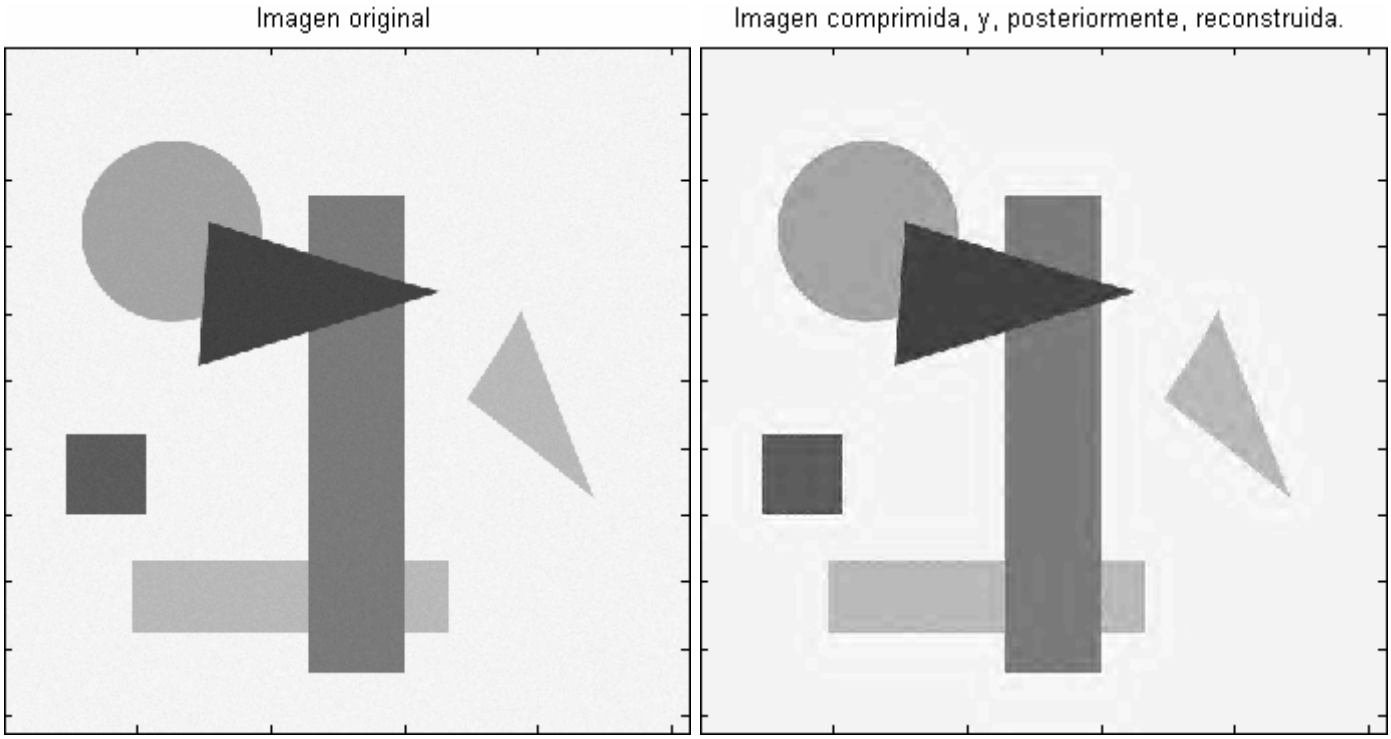
Imagen 2 + ruido blanco. Seis5.pgm + ruido blanco.

Tolerancia 11. Seis5 (4,11,8)=(l, tol, ruido):

```
>> a=imread('seis5.pgm');                    % leemos la imagen seis5.pgm
>> ruido=8; n=max(size(a));                  % introducimos el ruido en la imagen
>> a=double(a); a=a+ruido*rand(n);
>> a=round(a);
>> a=((a>=0)&(a<=255)).*a+(a>255)*255;
>> guionmet2m(4,11,a,'lin');                 % comprimimos con nuestro algoritmo
```

PSNR = 37.3135
Dimensión de la imagen = 512 · 512.
v1.png (668 bytes).
v2.png (1,67 KB @ 1.712 bytes).
apos.png (2,15 KB @ 2.203 bytes).

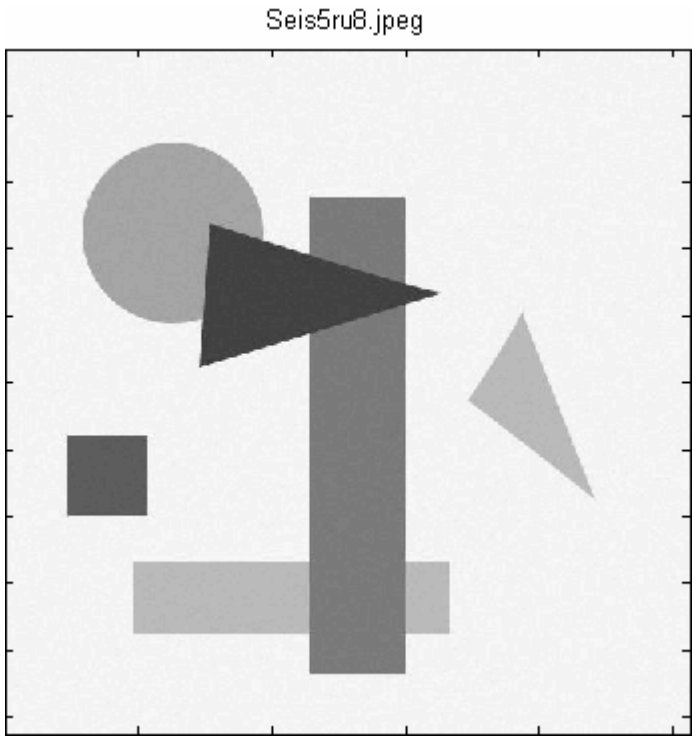
Total: 668 + 1.712+ 2.203 = 4583 bytes (es lo que ocupa la imagen comprimida).



```
>> a2=a;  a=uint8(a);
>> imwrite(a,'Seis5ru8.jpeg','jpeg')
>> a3=imread('Seis5ru8.jpeg','jpeg');
>> a4=double(a3);
>> psnr(a2,a4);
```

PSNR = 40,2438.

```
>> figure
>> imagesc(a3,[0,255])
>> axis('square')
>> colormap gray
>> title('Seis5ru8.jpeg')
```

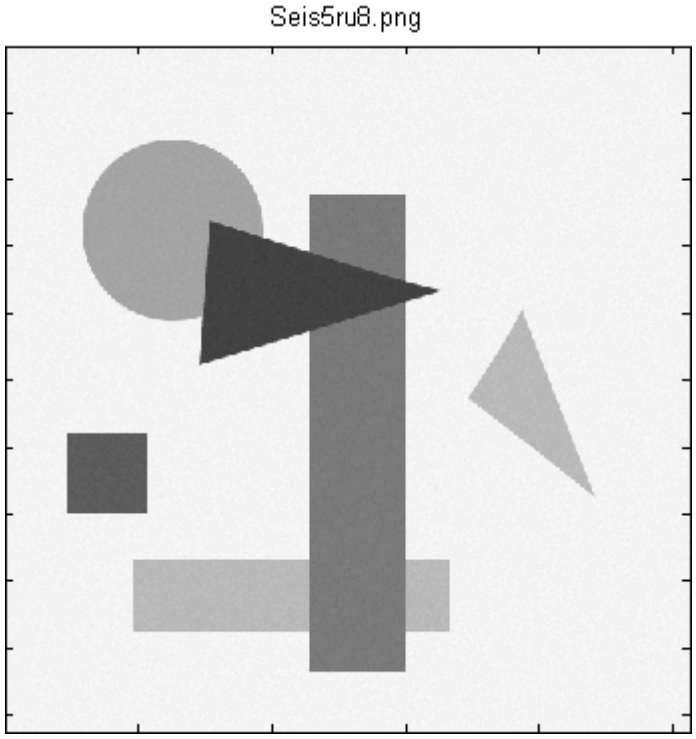


Cuánto ocupa esta imagen: 13,4 KB @ 13.797 bytes.

Comprimida con un parámetro de calidad de 75 de un rango de posibles valores en el intervalo [0, 100].

```
>> imwrite(a,'Seis5ru8.png','png')
>> a5=imread('Seis5ru8.png','png');
>> figure
>> imagesc(a5)
>> colormap gray
>> axis('square')
>> title('Seis5ru8.jpeg')
```

PSNR = Inf SIN PÉRDIDA



Cuánto ocupa esta imagen: 123 KB @126.385 bytes.

Comparación de resultados:

Imagen de este proyecto:

Imagen original

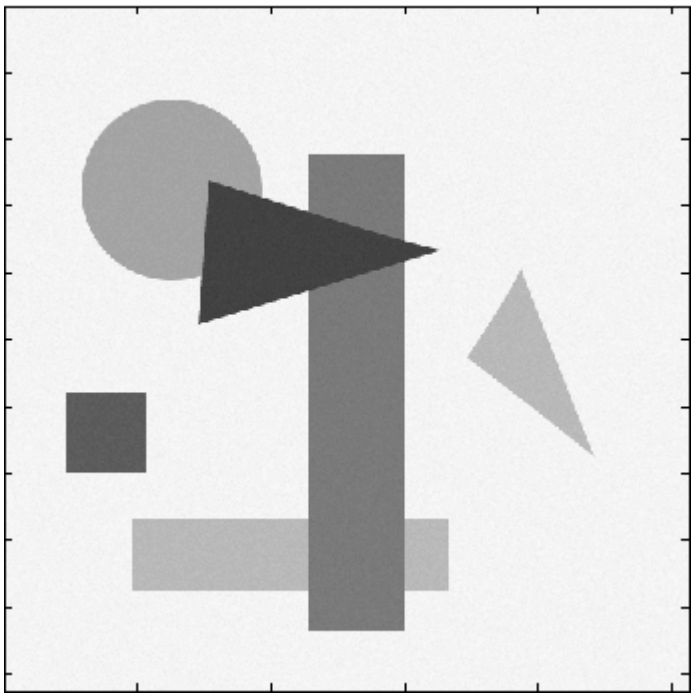
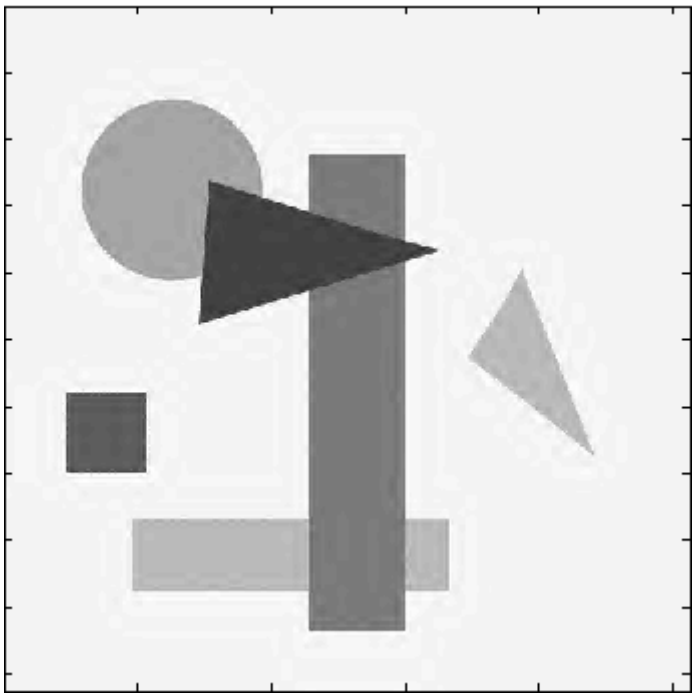
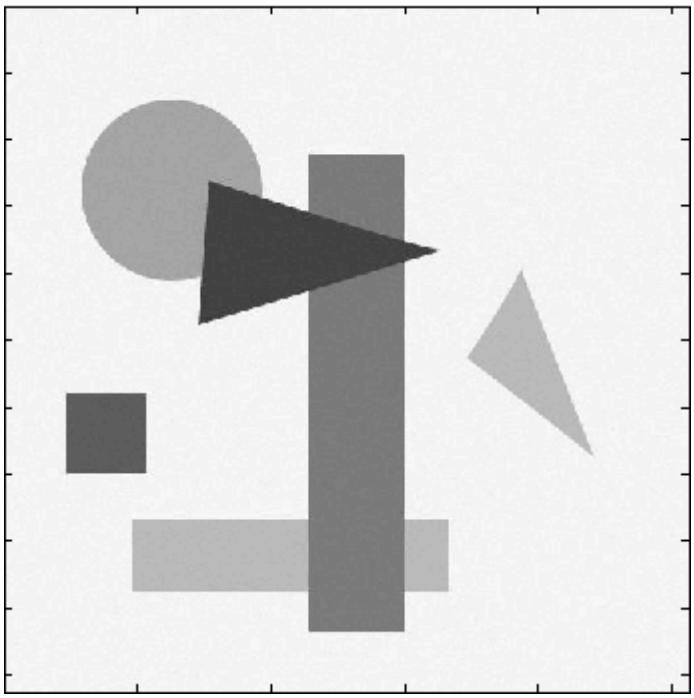


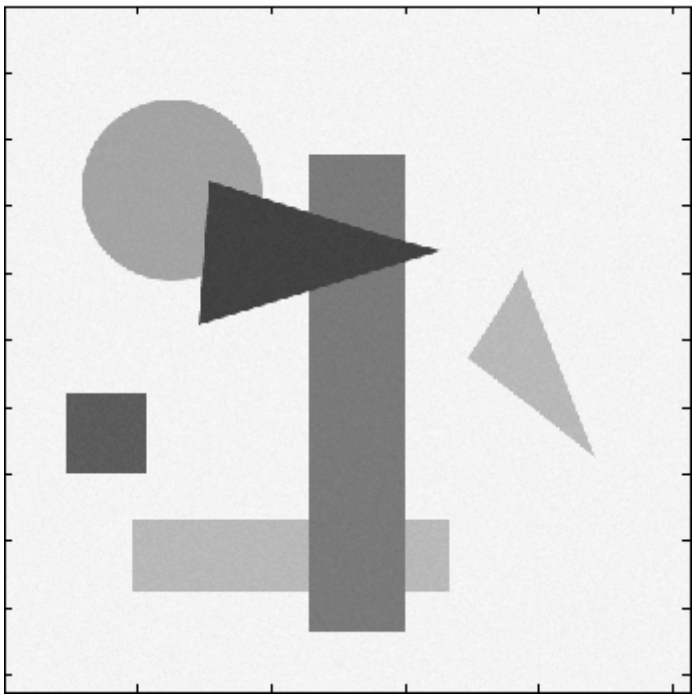
Imagen comprimida, y, posteriormente, reconstruida.



Seis5ru8.jpeg



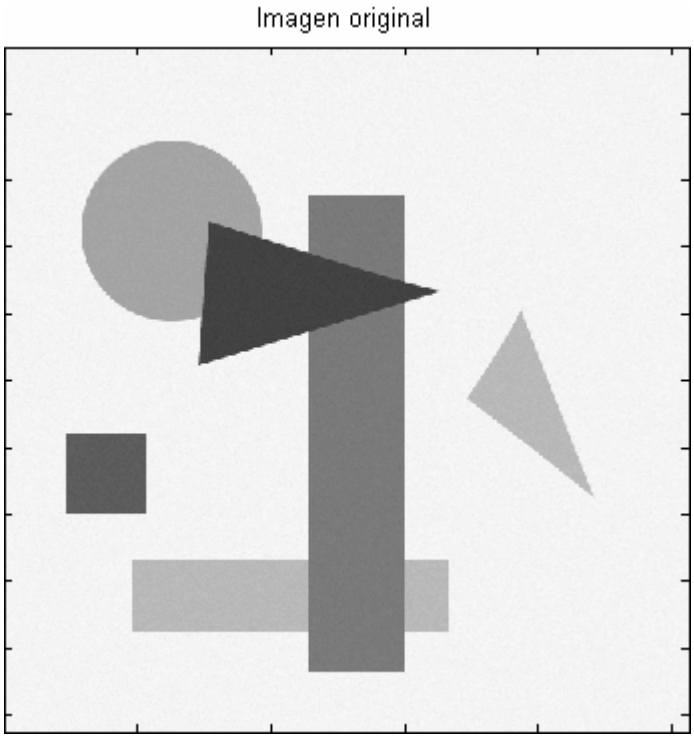
Seis5ru8.png

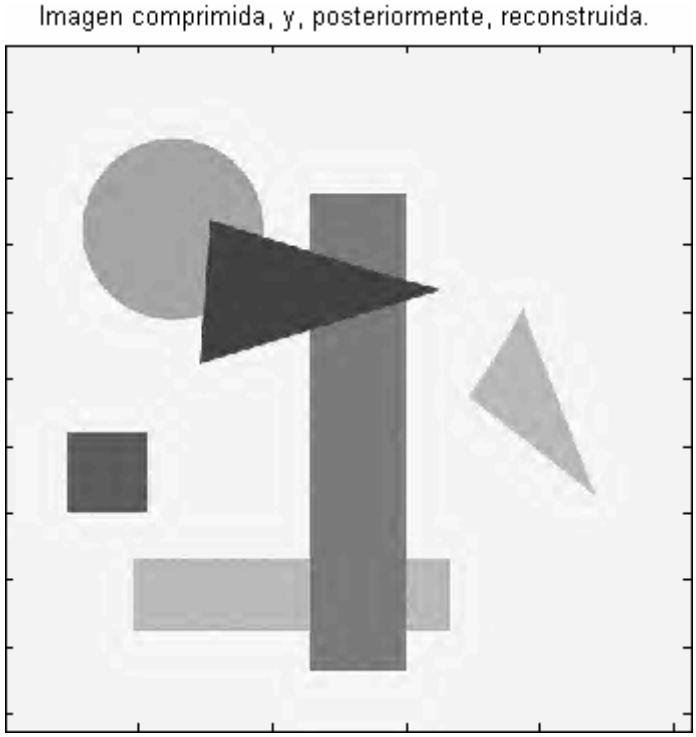


Tolerancia 12. Seis5 (4,12,8):
>> guionmet2m(4,12,a,'lin');

PSNR = 37.3135
Dimensión de la imagen = 512 · 512.
v1.png (653 bytes).
v2.png (1,55 KB @ 1.594 bytes).
apos.png (2,02 KB @ 2.072 bytes).

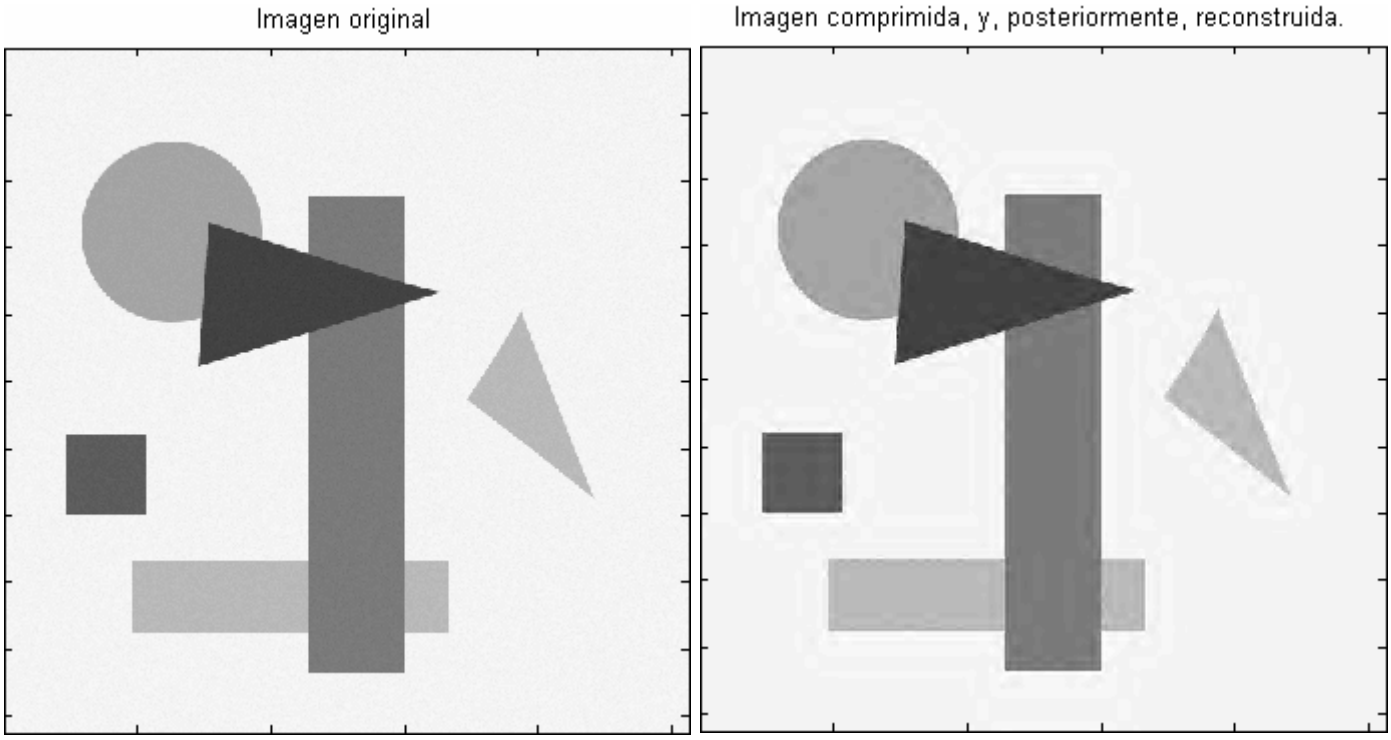
Total = 4.319 bytes (es lo que ocupa la imagen comprimida).





Comparación de resultados:

Imagen de este proyecto:



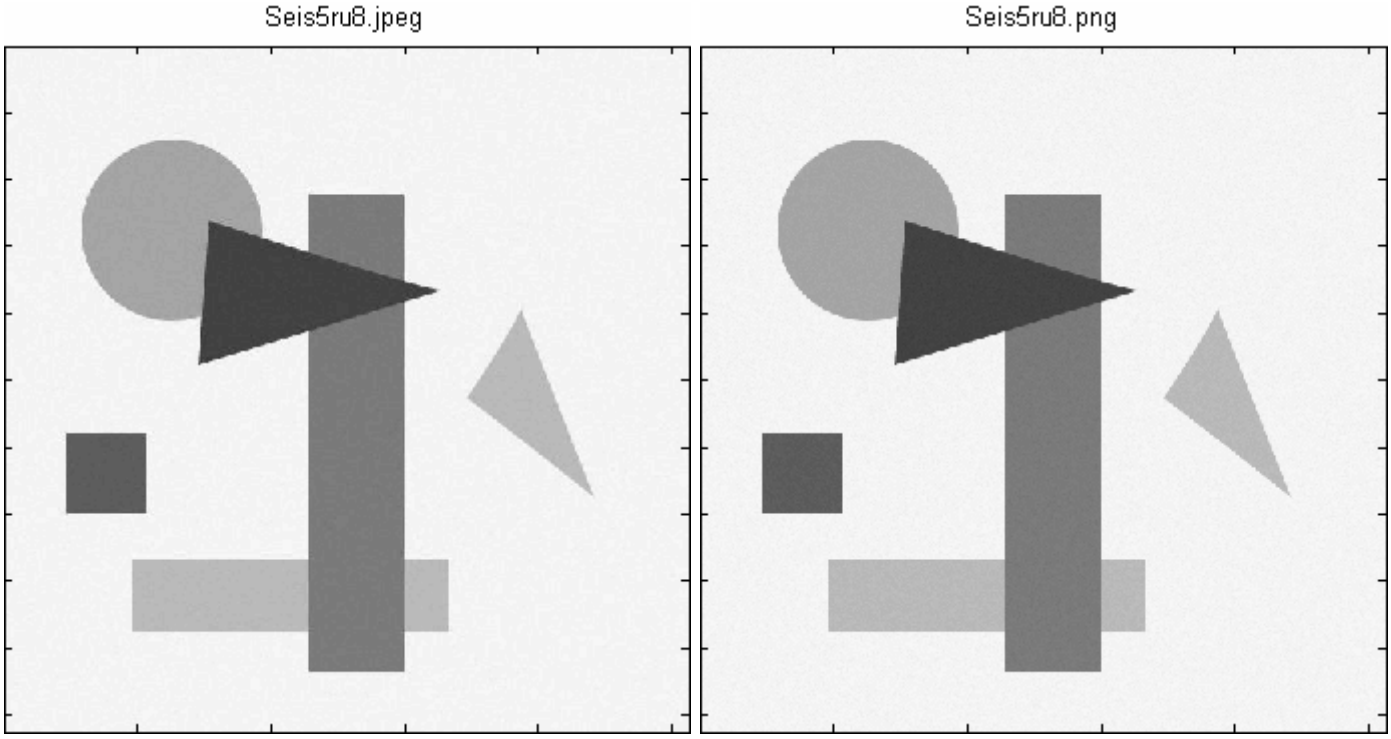


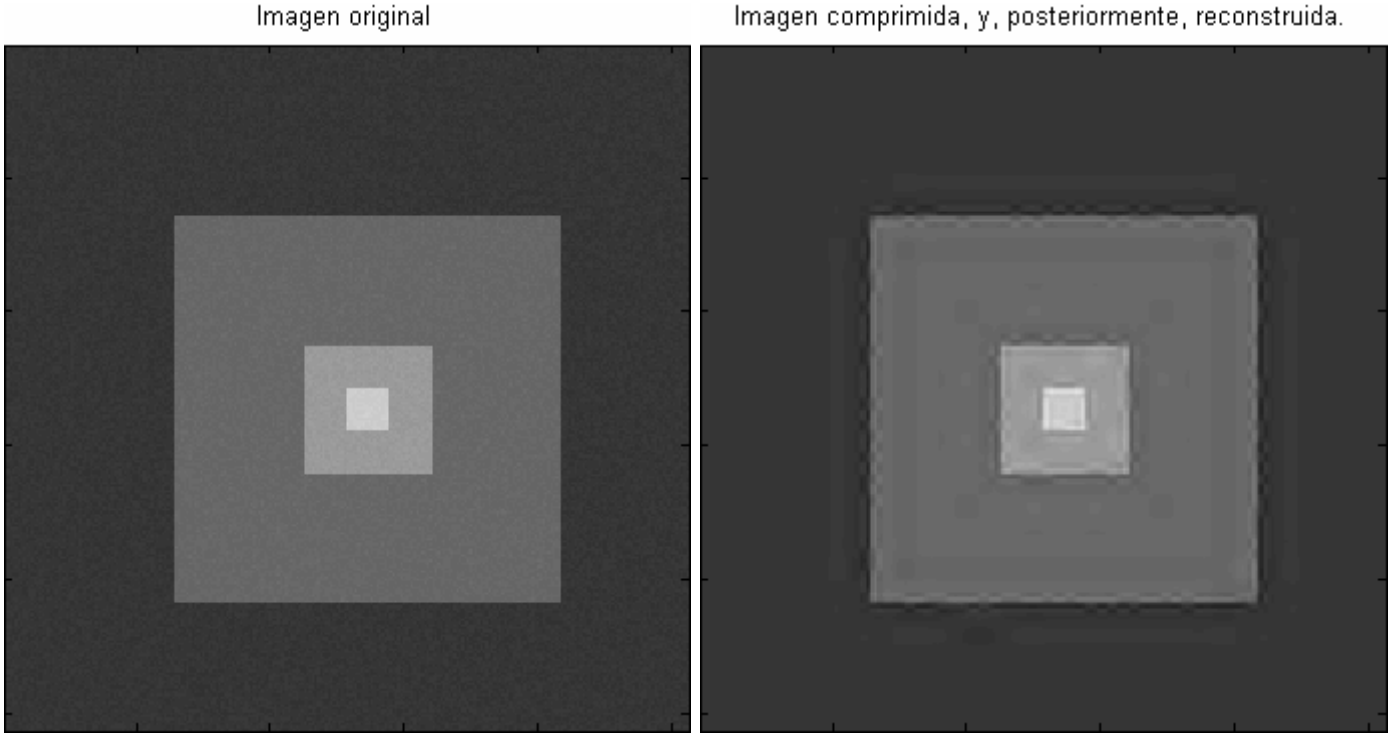
Imagen 3 + ruido blanco. Squares2.pgm + ruido blanco.

Tolerancia 11. Squares2 (4,11,8)=(l, tol, ruido):

```
>> a=imread('squares2.pgm');                 % leemos la imagen squares2.pgm
>> ruido=8; n=max(size(a));                  % introducimos el ruido en la imagen
>> a=double(a); a=a+ruido*rand(n);
>> a=round(a);
>> a=((a>=0)&(a<=255)).*a+(a>255)*255;
>> guionmet2m(4,11,a,'lin');                 % comprimimos con nuestro algoritmo
```

PSNR = 37.3135
Dimensión de la imagen = 256 · 256.
v1.png (178 bytes).
v2.png (235 bytes).
apos.png (398 bytes).

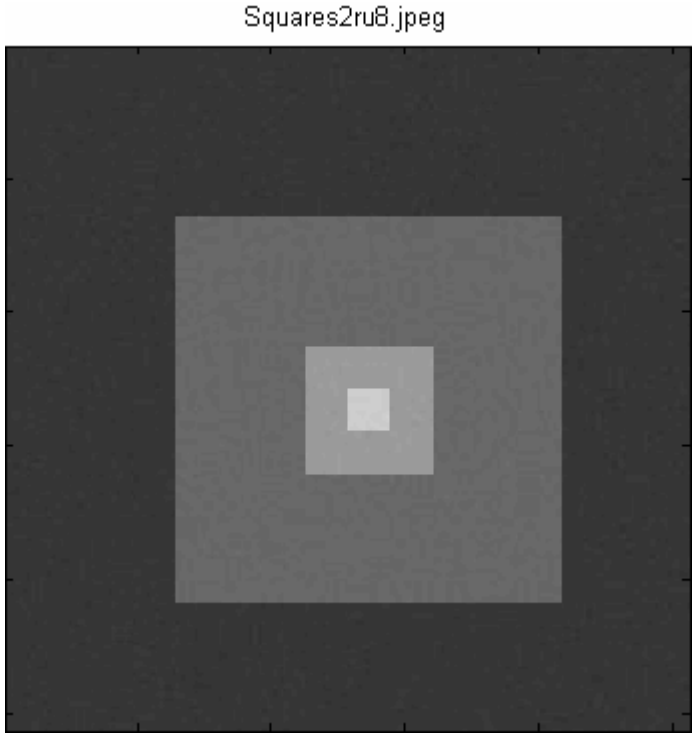
Total: 178 + 235+ 398 = 811 bytes (es lo que ocupa la imagen comprimida).



```
>> a2=a;  a=uint8(a);
>> imwrite(a,'Squares2ru8.jpeg','jpeg')
>> a3=imread('Squares2ru8.jpeg','jpeg');
>> a4=double(a3);
>> psnr(a2,a4);
```

PSNR = 41,0035.

```
>> figure
>> imagesc(a3,[0,255])
>> axis('square')
>> colormap gray
>> title('Squares2ru8.jpeg')
```

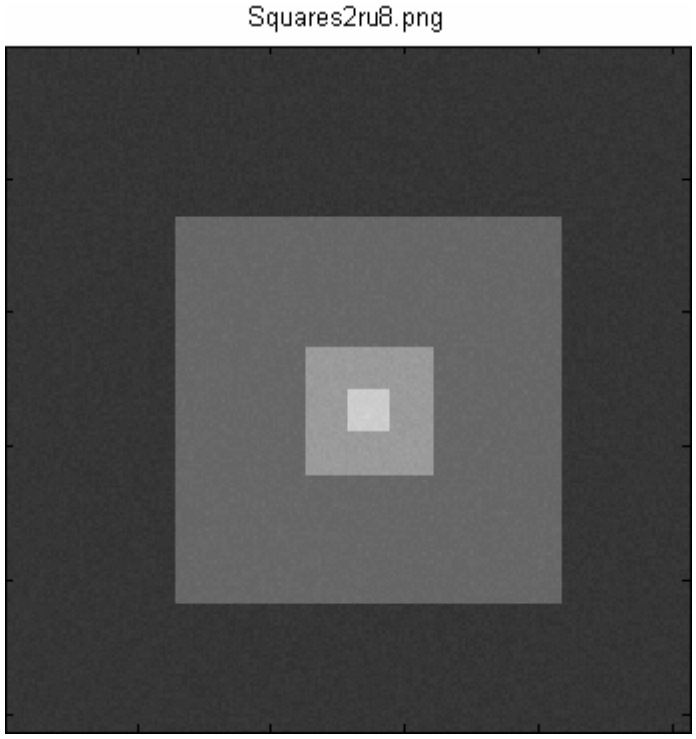


Cuánto ocupa esta imagen: 2,58 KB @ 2.643 bytes.

Comprimida con un parámetro de calidad de 75 de un rango de posibles valores en el intervalo [0, 100].

```
>> imwrite(a,'Squares2ru8.png','png')
>> a5=imread('Squares2ru8.png','png');
>> figure
>> imagesc(a5)
>> colormap gray
>> axis('square')
>> title('Squares2ru8.jpeg')
```

PSNR = Inf SIN PÉRDIDA



Cuánto ocupa esta imagen: 30’1 KB @30.896 bytes.

Comparación de resultados:

Imagen de este proyecto:

Imagen original

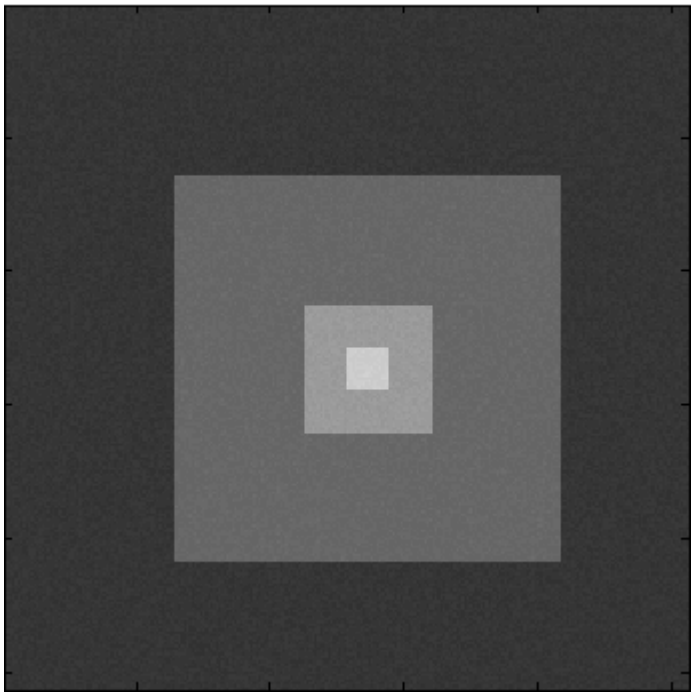
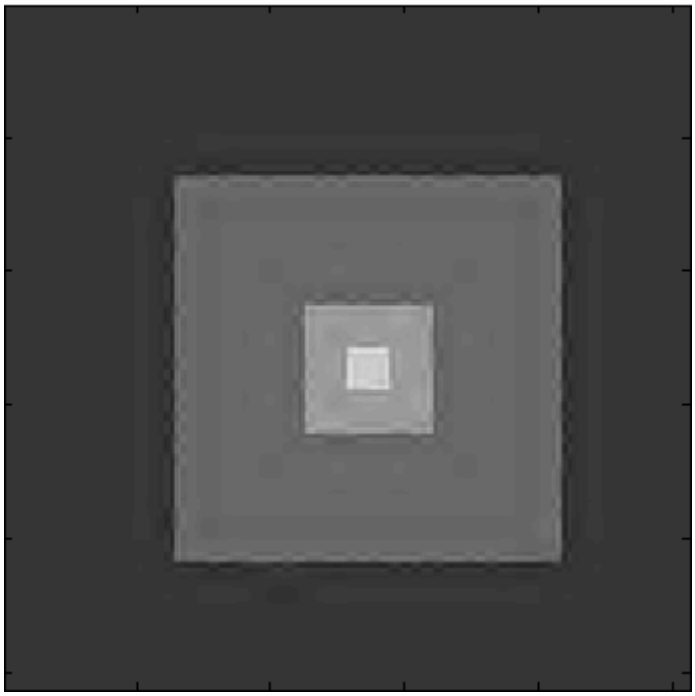
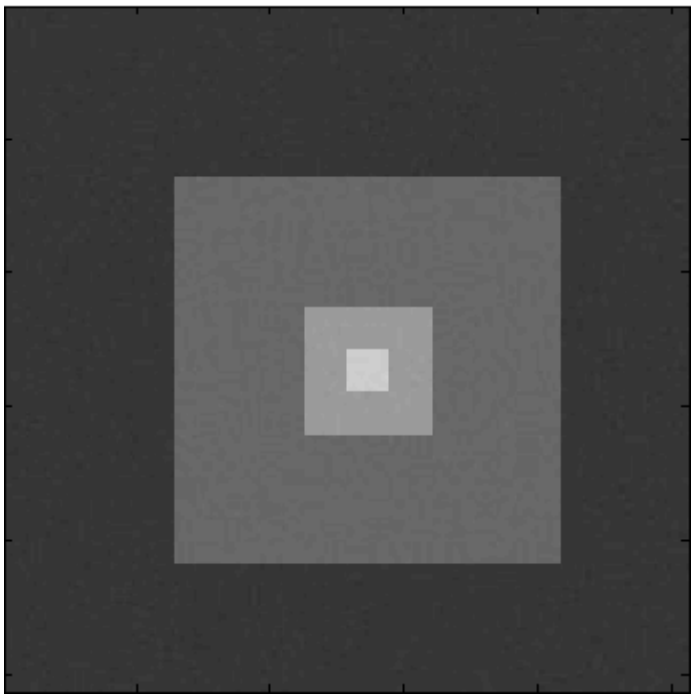


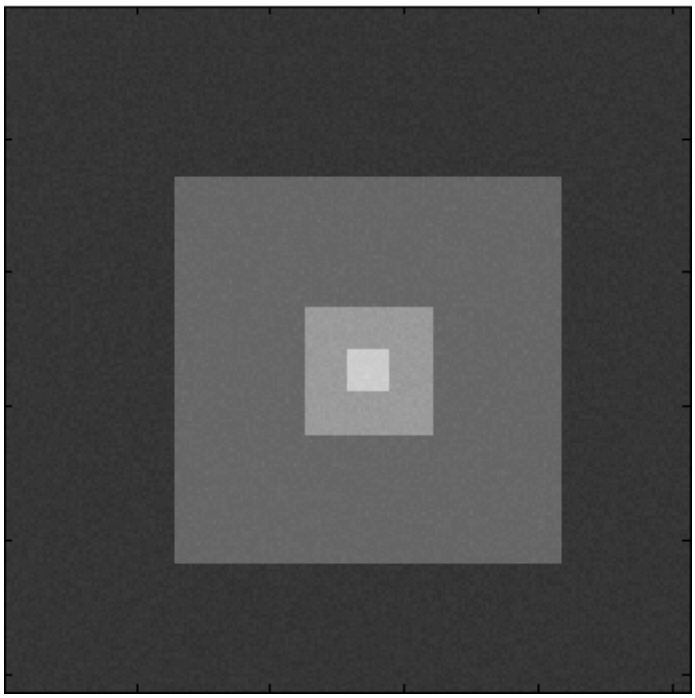
Imagen comprimida, y, posteriormente, reconstruida.



Squares2ru8.jpeg



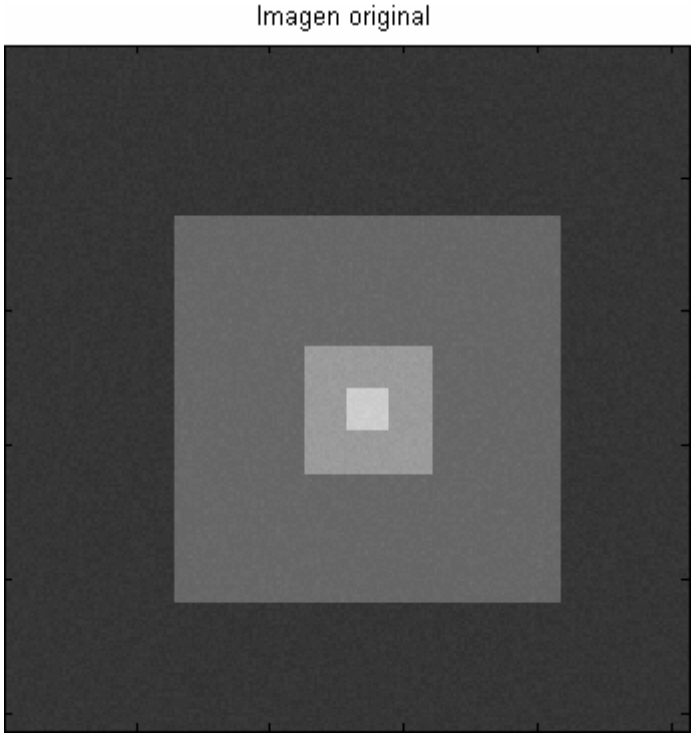
Squares2ru8.png

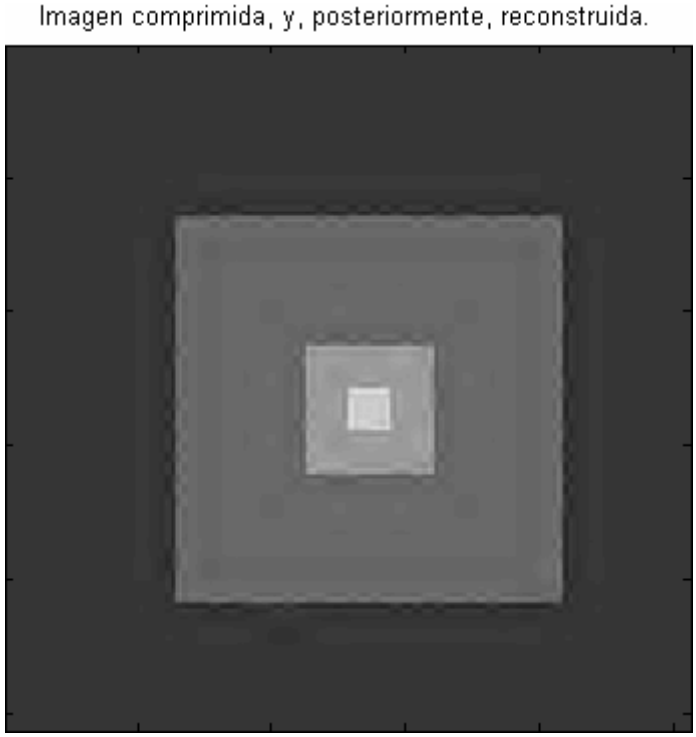


Tolerancia 12. Squares2 (4,12,8):
>> guionmet2m(4,12,a,'lin');

PSNR = 37.1667
Dimensión de la imagen = 256 · 256.
v1.png (178 bytes).
v2.png (235 bytes).
apos.png (398 bytes).

Total = 811 bytes (es lo que ocupa la imagen comprimida).





Comparación de resultados:

Imagen de este proyecto:

Imagen original

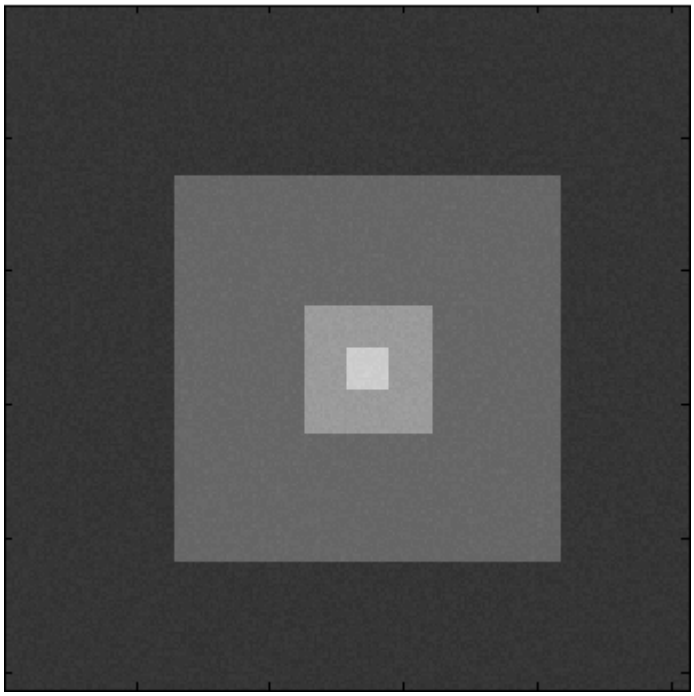
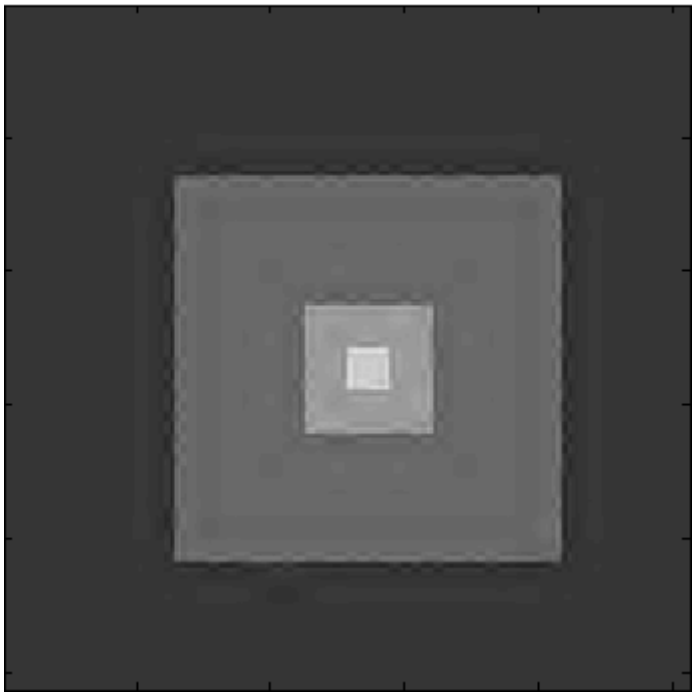
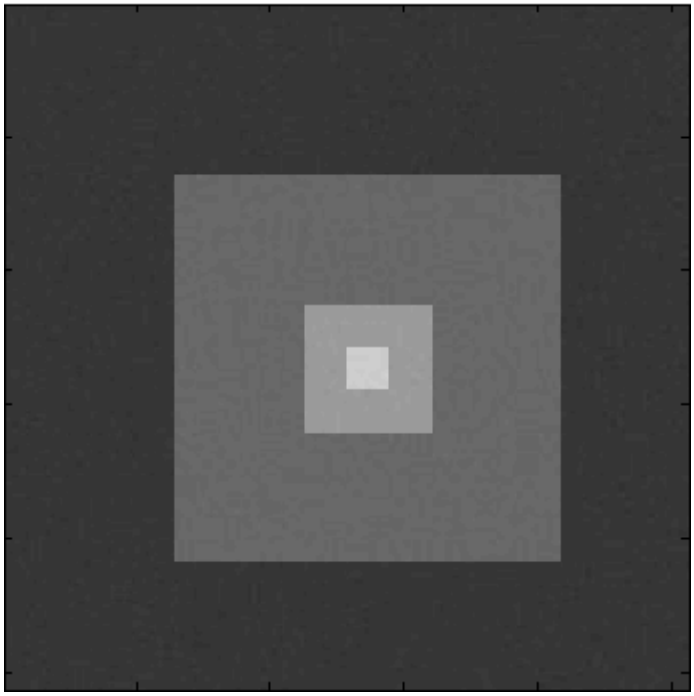


Imagen comprimida, y, posteriormente, reconstruida.



Squares2ru8.jpeg



Squares2ru8.png

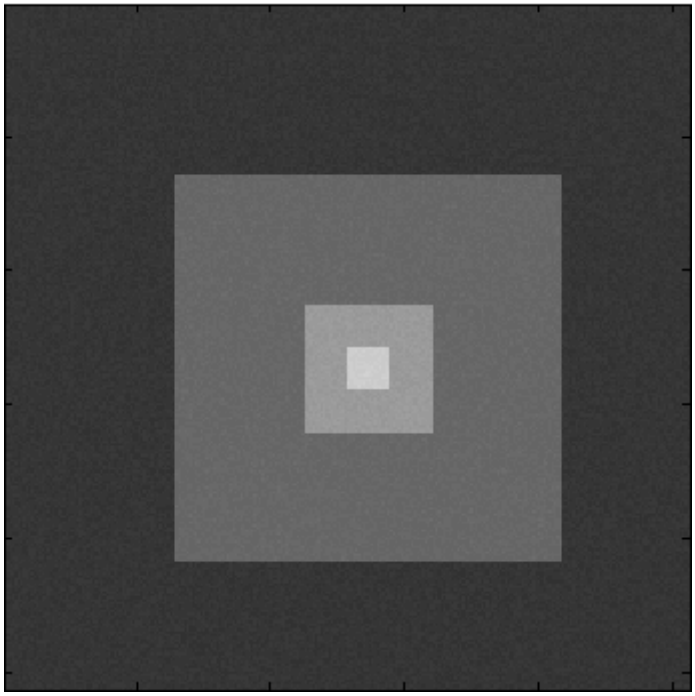


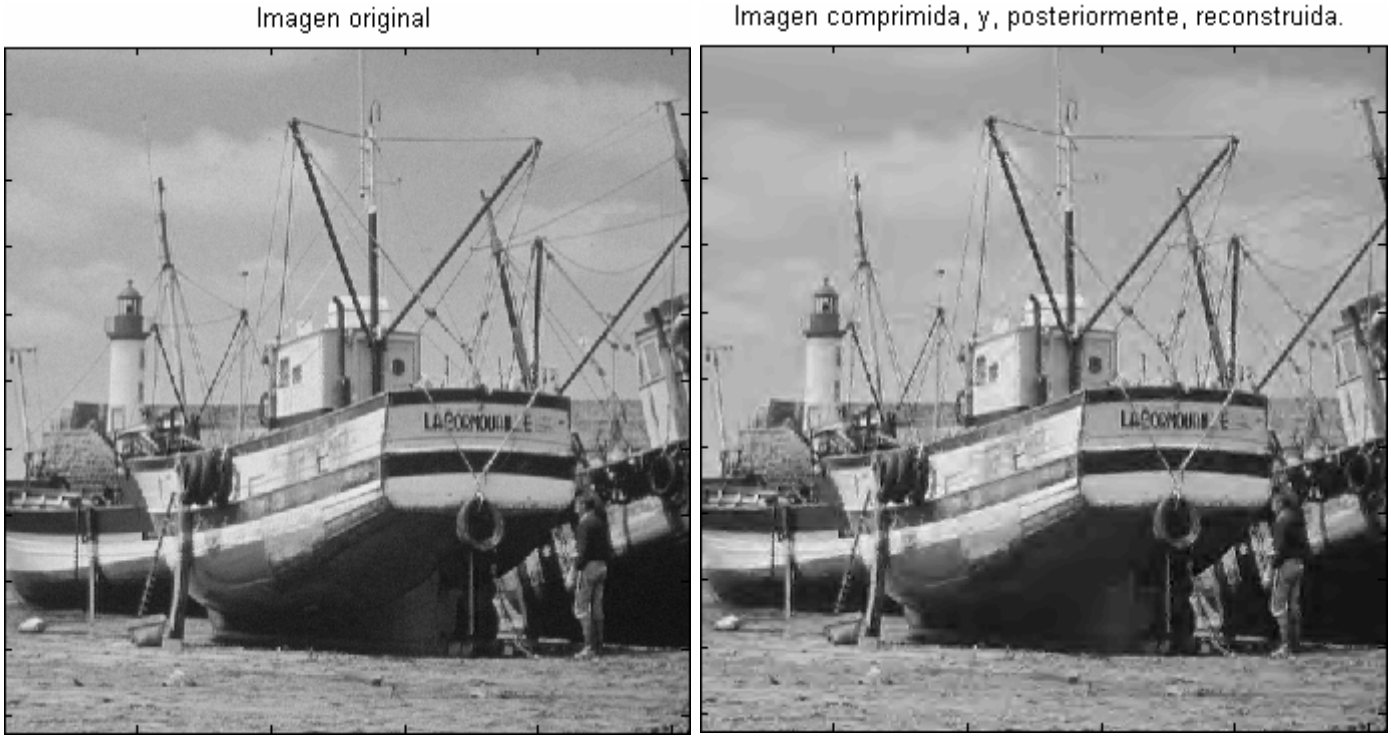
Imagen 4 + ruido blanco. Boat5.pgm + ruido blanco.

Tolerancia 11. Boat5 (4,11,8)=(l, tol, ruido):

```
>> a=imread('boat5.pgm');                    % leemos la imagen boat5.pgm
>> ruido=8; n=max(size(a));                  % introducimos el ruido en la imagen
>> a=double(a); a=a+ruido*rand(n);
>> a=round(a);
>> a=((a>=0)&(a<=255)).*a+(a>255)*255;
>> guionmet2m(4,11,a,'lin');                 % comprimimos con nuestro algoritmo
```

PSNR = 30.7825
Dimensión de la imagen = 512 · 512.
v1.png (3,38 KB @ 3.471 bytes).
v2.png (6,05 KB @ 6.196 bytes).
apos.png (8,94 KB @ 9.158 bytes).

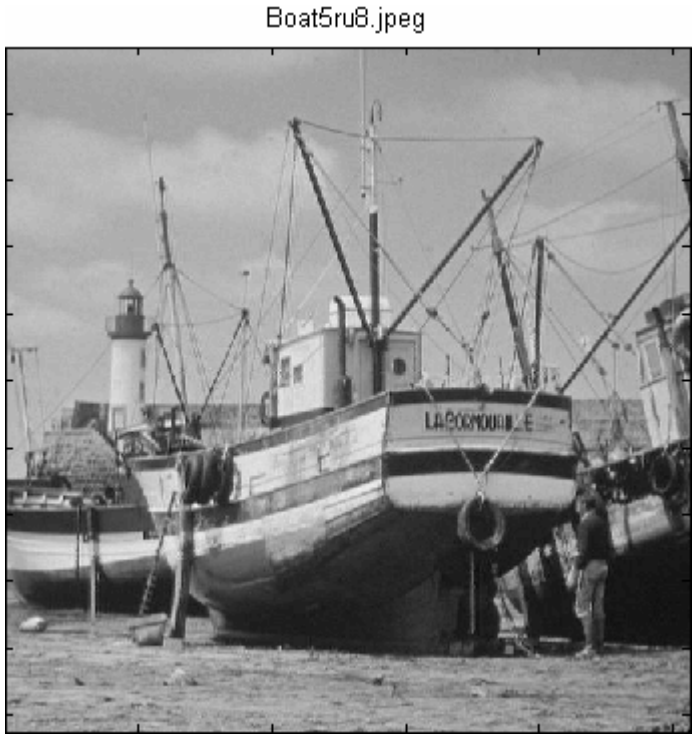
Total: 3.471 + 6.196+ 9.158 = 18.825 bytes (es lo que ocupa la imagen comprimida).



```
>> a2=a;  a=uint8(a);
>> imwrite(a,'Boat5ru8.jpeg','jpeg')
>> a3=imread('Boat5ru8.jpeg','jpeg');
>> a4=double(a3);
>> psnr(a2,a4);
```

PSNR = 36,1607.

```
>> figure
>> imagesc(a3,[0,255])
>> axis('square')
>> colormap gray
>> title('Boat5ru8.jpeg')
```

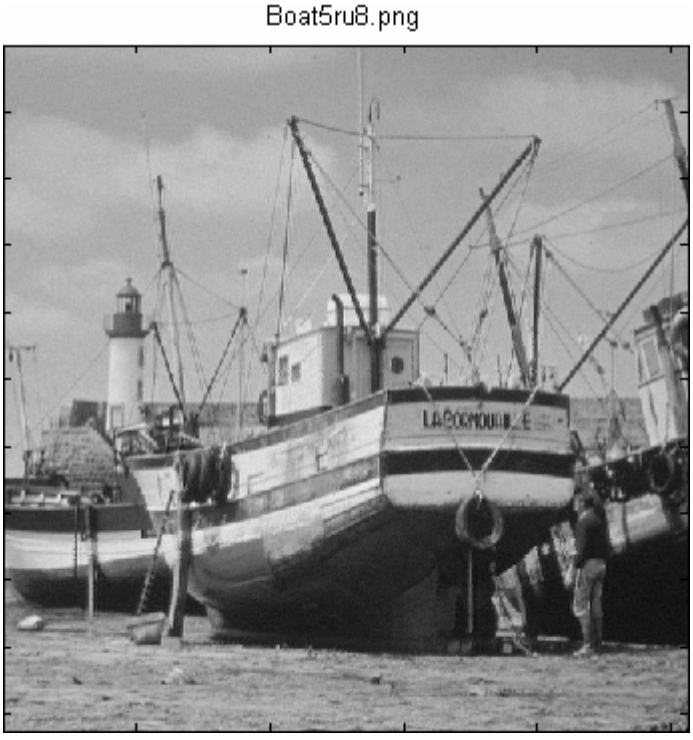


Cuánto ocupa esta imagen: 37,9 KB @ 38.824 bytes.

Comprimida con un parámetro de calidad de 75 de un rango de posibles valores en el intervalo [0, 100].

```
>> imwrite(a,'Boat5ru8.png','png')
>> a5=imread('Boat5ru8.png','png');
>> figure
>> imagesc(a5)
>> colormap gray
>> axis('square')
>> title('Boat5ru8.jpeg')
```

PSNR = Inf SIN PÉRDIDA



Cuánto ocupa esta imagen: 158 KB @162.576 bytes.

Comparación de resultados:

Imagen de este proyecto:

Imagen original

Imagen comprimida, y, posteriormente, reconstruida.

Boat5ru8.jpeg

Boat5ru8.png

Tolerancia 12. Boat5 (4,12,8):
>> guionmet2m(4,12,a,'lin');

PSNR = 37.1667
Dimensión de la imagen = 512 · 512.
v1.png (3,21 KB @ 3.296 bytes).
v2.png (5,80 KB @ 5.942 bytes).
apos.png (8,48 KB @ 8.689 bytes).

Total = 17927 bytes (es lo que ocupa la imagen comprimida).

Imagen original

Imagen comprimida, y, posteriormente, reconstruida.

Comparación de resultados:

Imagen de este proyecto:

Imagen original

Imagen comprimida, y, posteriormente, reconstruida.

Boat5ru8.jpeg

Boat5ru8.png

S.8. Sección ocho. Tablas:

En esta sección aparecen las tablas que ha sido rellenadas usando los datos obtenidos en la anterior sección.

El significado de cada parámetro de estas tablas está convenientemente explicado en las páginas 89 y 90, inicio de la anterior sección.

Sin ruido:

Tabla 1 (4,9,0):

Imagen	Método de este proyecto		JPEG		PNG	
	PSNR	Bytes	PSNR	Bytes	PSNR	Bytes
Tiffany5	33.6628	14528	36.6608	30031	Inf	142170
Seis5	40.3087	4825	47.5620	8085	Inf	4580
Squares2	41.2523	656	Inf	1152	Inf	659
Boat5	31.9098	21799	37.2704	36272	Inf	151933

Tabla 2 (4,10,0):

Imagen	Método de este proyecto		JPEG		PNG	
	PSNR	Bytes	PSNR	Bytes	PSNR	Bytes
Tiffany5	33.5557	13455	36.6608	30031	Inf	142170
Seis5	40.3305	4615	47.5620	8085	Inf	4580
Squares2	41.2523	656	Inf	1152	Inf	659
Boat5	31.6993	20656	37.2704	36272	Inf	151933

Tabla 3 (4,11,0):

Imagen	Método de este proyecto		JPEG		PNG	
	PSNR	Bytes	PSNR	Bytes	PSNR	Bytes
Tiffany5	33.1850	12552	36.6608	30031	Inf	142170
Seis5	39.9057	4315	47.5620	8085	Inf	4580
Squares2	39.8679	658	Inf	1152	Inf	659
Boat5	31.1469	19738	37.2704	36272	Inf	151933

Tabla 4 (4,12,0):

Imagen	Método de este proyecto		JPEG		PNG	
	PSNR	Bytes	PSNR	Bytes	PSNR	Bytes
Tiffany5	33.0711	12172	36.6608	30031	Inf	142170
Seis5	39.7905	4196	47.5620	8085	Inf	4580
Squares2	39.8679	658	Inf	1152	Inf	659
Boat5	30.9921	17597	37.2704	36272	Inf	151933

Tabla 5 (Tiffany5):

Imagen	Método de este proyecto		JPEG		PNG	
	PSNR	Bytes	PSNR	Bytes	PSNR	Bytes
4,9,0	33.6628	14528	36.6608	30031	Inf	142170
4,10,0	33.5557	13455	36.6608	30031	Inf	142170
4,11,0	33.1850	12552	36.6608	30031	Inf	142170
4,12,0	33.0711	12172	36.6608	30031	Inf	142170

Tabla 6 (Seis5):

Imagen	Método de este proyecto		JPEG		PNG	
	PSNR	Bytes	PSNR	Bytes	PSNR	Bytes
4,9,0	40.3087	4825	47.5620	8085	Inf	4580
4,10,0	40.3305	4615	47.5620	8085	Inf	4580
4,11,0	39.9057	4315	47.5620	8085	Inf	4580
4,12,0	39.7905	4196	47.5620	8085	Inf	4580

Tabla 7 (Squares2):

Imagen	Método de este proyecto		JPEG		PNG	
	PSNR	Bytes	PSNR	Bytes	PSNR	Bytes
4,9,0	41.2523	656	Inf	1152	Inf	659
4,10,0	41.2523	656	Inf	1152	Inf	659
4,11,0	39.8679	658	Inf	1152	Inf	659
4,12,0	39.8679	658	Inf	1152	Inf	659

Tabla 8 (Boat5):

Imagen	Método de este proyecto		JPEG		PNG	
	PSNR	Bytes	PSNR	Bytes	PSNR	Bytes
4,9,0	31.9098	21799	37.2704	36272	Inf	151933
4,10,0	31.6993	20656	37.2704	36272	Inf	151933
4,11,0	31.1469	19738	37.2704	36272	Inf	151933
4,12,0	30.9921	17597	37.2704	36272	Inf	151933

Ofrecemos a continuación una tabla donde se hace una comparativa del método presentado en el proyecto con el JPEG para similar calidad de imagen PSNR.

Tabla 9 (P.S.N.R.s similares):

Imagen	Método de este proyecto		JPEG	
	PSNR	Bytes	PSNR	Bytes
Tiffany5 (4,5,0)	36.3089	24566	36.6608	30031
Seis5 (4,4.2,0)	47.5975	7398	47.5620	8085
Boat5 (4,3.75,0)	37.1192	47916	37.2704	36272

Con ruido:

Tabla 10 (4,11,8):

Imagen	Método de este proyecto		JPEG		PNG	
	PSNR	Bytes	PSNR	Bytes	PSNR	Bytes
Tiffany5	32.5545	12767	35.6922	32854	Inf	154198
Seis5	37.3135	4583	40.2438	13797	Inf	126385
Squares2	37.3135	811	41.0035	2643	Inf	30896
Boat5	30.7825	18825	36.1607	38824	Inf	162576

Tabla 11 (4,12,8):

Imagen	Método de este proyecto		JPEG		PNG	
	PSNR	Bytes	PSNR	Bytes	PSNR	Bytes
Tiffany5	32.4562	12300	35.6922	32854	Inf	154198
Seis5	37.3135	4319	40.2438	13797	Inf	126385
Squares2	37.1667	811	41.0035	2643	Inf	30896
Boat5	37.1667	17927	36.1607	38824	Inf	162576

Tabla 12 (Tiffany5):

Imagen	Método de este proyecto		JPEG		PNG	
	PSNR	Bytes	PSNR	Bytes	PSNR	Bytes
4,11,8	32.5545	12767	35.6922	32854	Inf	154198
4,12,8	32.4562	12300	35.6922	32854	Inf	154198

Tabla 13 (Seis5):

Imagen	Método de este proyecto		JPEG		PNG	
	PSNR	Bytes	PSNR	Bytes	PSNR	Bytes
4,11,8	37.3135	4583	40.2438	13797	Inf	126385
4,12,8	37.3135	4319	40.2438	13797	Inf	126385

Tabla 14 (Squares2):

Imagen	Método de este proyecto		JPEG		PNG	
	PSNR	Bytes	PSNR	Bytes	PSNR	Bytes
4,11,8	37.3135	811	41.0035	2643	Inf	30896
4,12,8	37.1667	811	41.0035	2643	Inf	30896

Tabla 15 (Boat5):

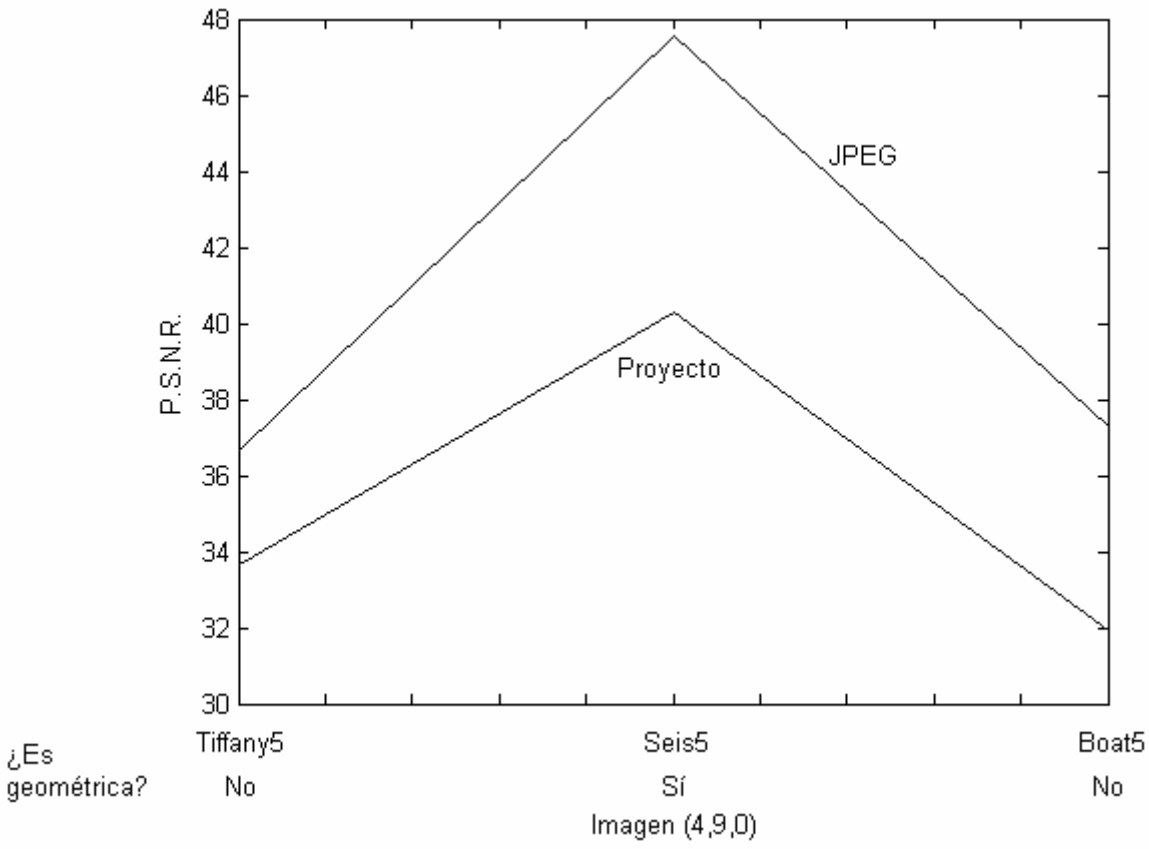
Imagen	Método de este proyecto		JPEG		PNG	
	PSNR	Bytes	PSNR	Bytes	PSNR	Bytes
4,11,8	30.7825	18825	36.1607	38824	Inf	162576
4,12,8	37.1667	17927	36.1607	38824	Inf	162576

S.9. Sección nueve. Gráficas:

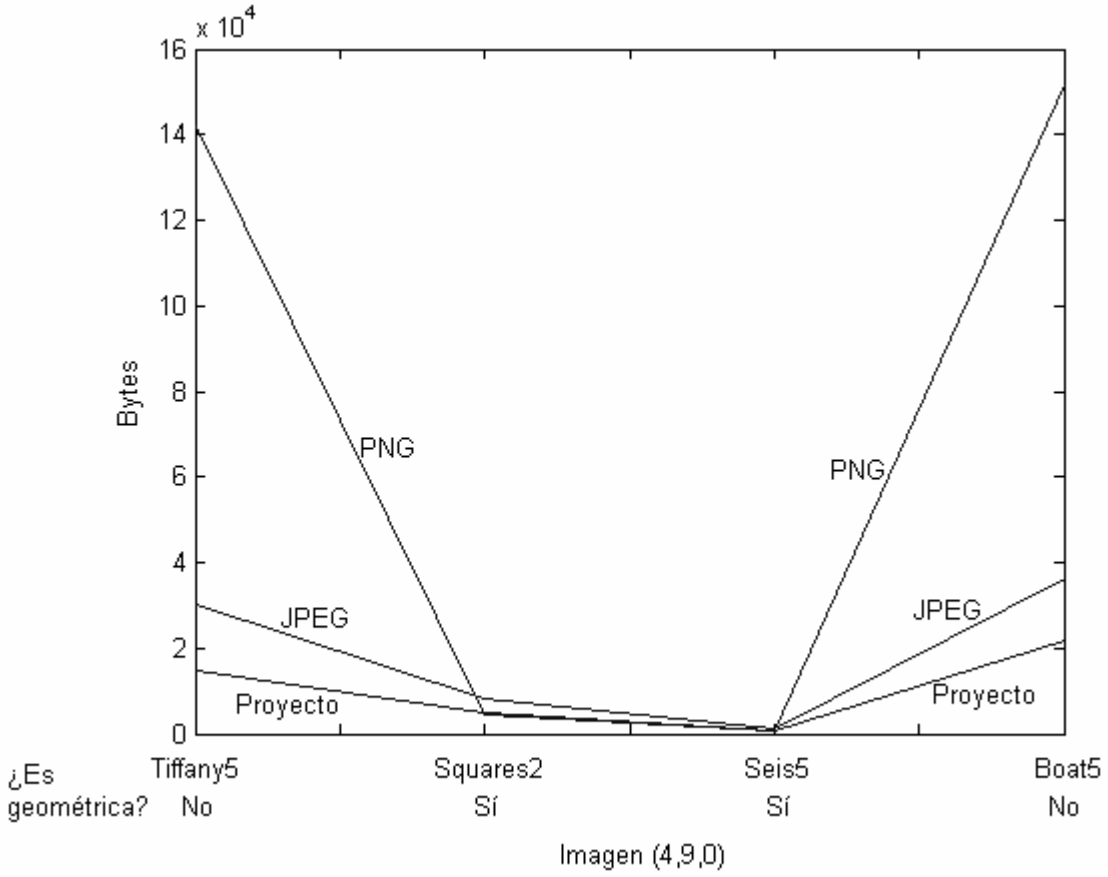
En esta sección se han obtenido gráficas usando los datos de las tablas de la anterior sección.

Gráficas de la tabla uno (4,9,0):

Gráfica uno. En esta gráfica, se muestra el valor del **PSNR** que el formato de compresión JPEG y el propuesto en el proyecto han obtenido para las diferentes **imágenes**.

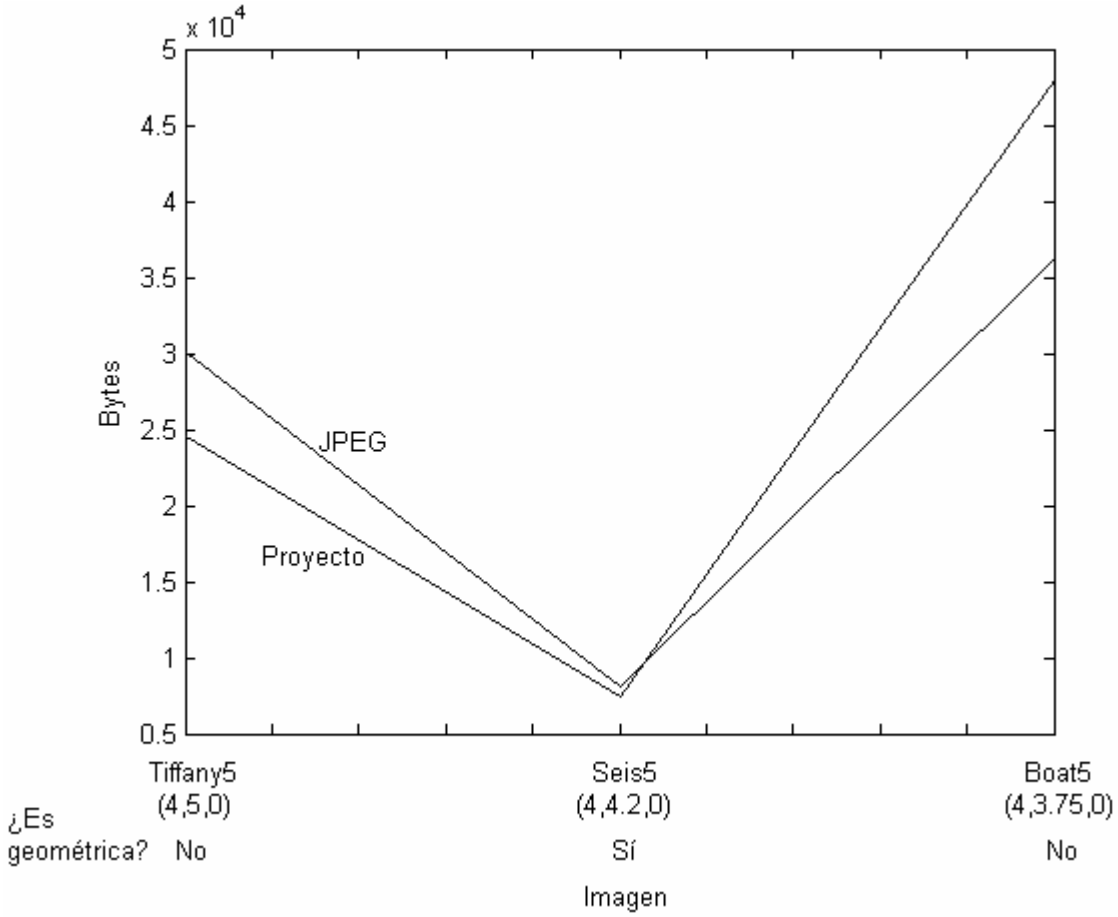


Gráfica dos. En esta gráfica, se muestra el valor de los **bytes** que cada formato ha ocupado con cada una de las **imágenes** de la tabla de la anterior sección.



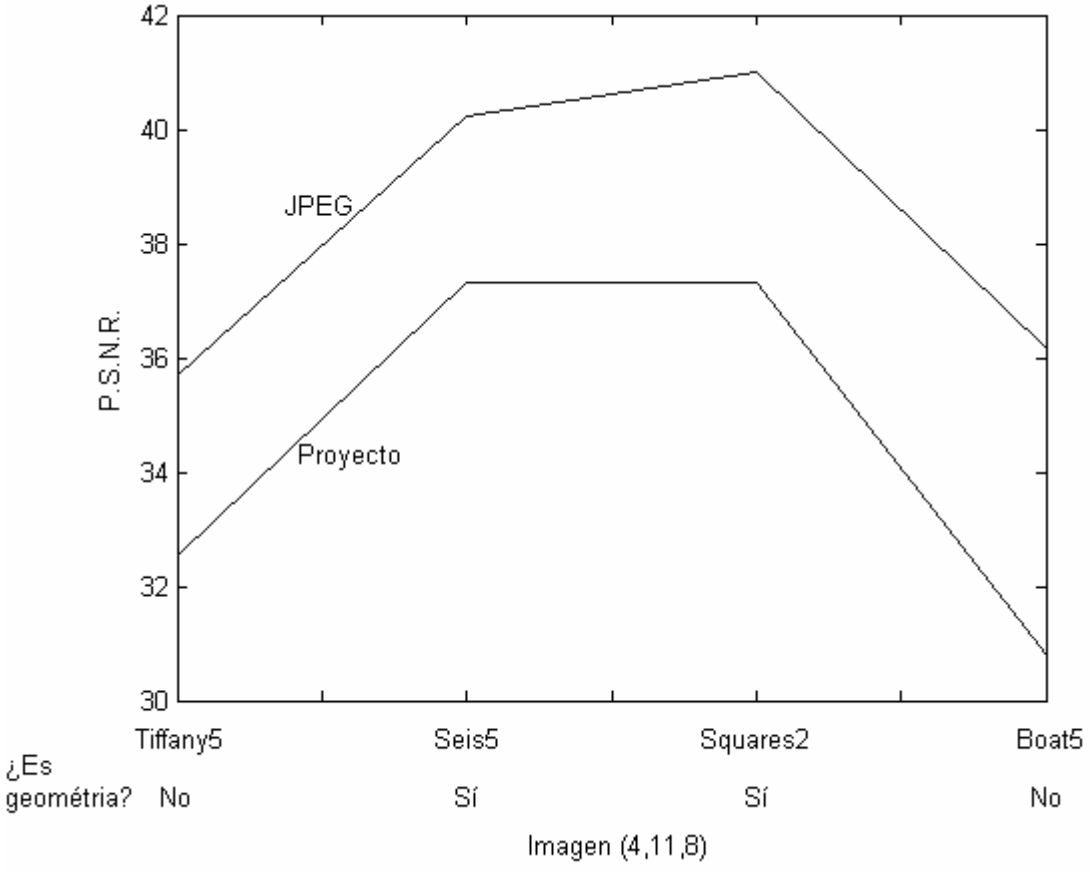
Gráfica de la tabla nueve (P.S.N.R.s similares):

Gráfica tres.

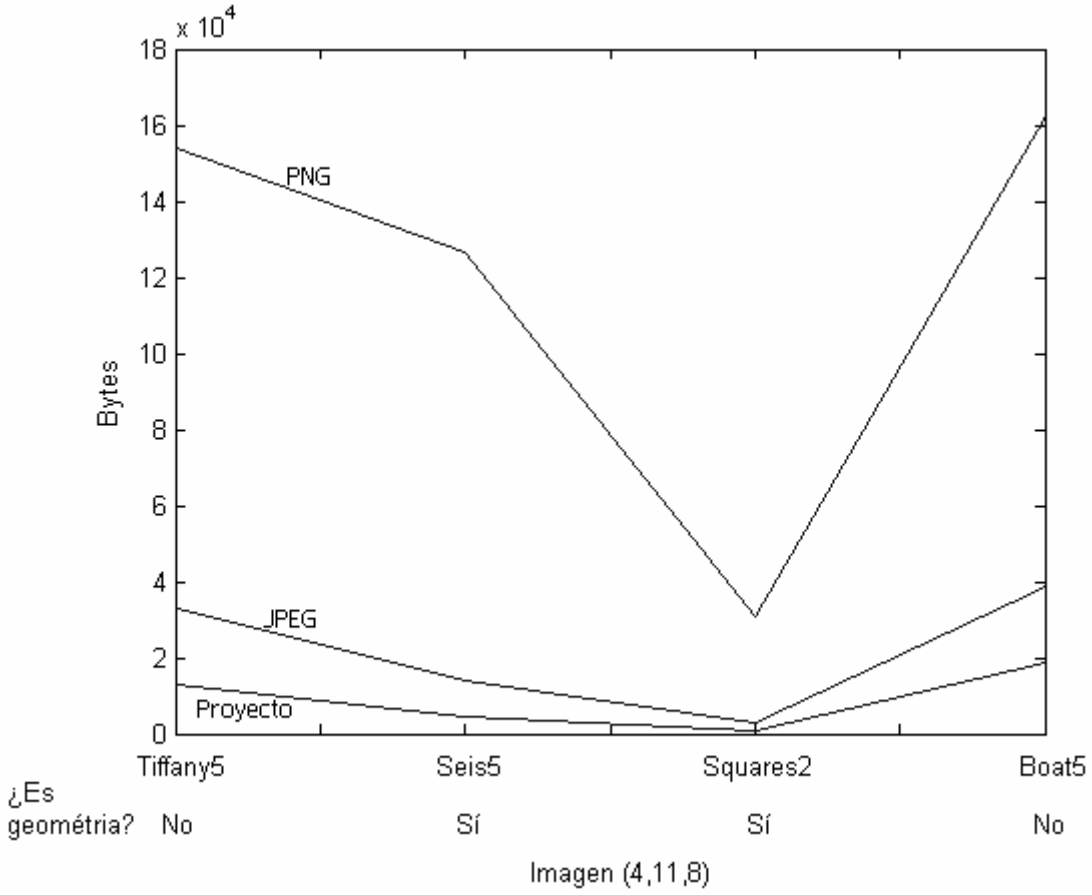


Gráficas de la tabla diez (4,11,8):

Gráfica cuatro:



Gráfica cinco:



Parte III (Conclusión):

En la primera parte de este proyecto hemos explicado y aplicado el método de compresión de imágenes que se deriva de la multirresolución de Harten cuando se utiliza una discretización por medias en celda (que es conocido que es la adecuada para trabajar con imágenes) y una reconstrucción via función primitiva que hace uso de una interpolación segmentaria de Lagrange segmentaria de orden 4. El tipo de reconstrucción escogido es lineal, y así no dependiente de los datos. Se podría haber implementado también el algoritmo con una reconstrucción no lineal [1],[4],[7].

En la segunda parte del proyecto, la cual es novedosa, tratamos de hacer utilizables en la práctica los algoritmos de compresión presentados anteriormente. Para ello hemos de utilizar cuantización en vez de truncación, y hemos de comprimir al final datos del tipo entero en lugar de double. Además hemos implementado un algoritmo para codificar en bits los elementos no cero de la matriz de multirresolución. Esta será nuestra versión comprimida de la imagen. Este algoritmo de codificación en bits hace uso del conocido standard de compresión PNG (formato de compresión muy competitivo con imágenes simples o geométricas), y se basa en guardar por un lado la localización de los coeficientes no nulos de la matriz de multirresolución (en una matriz binaria), y por otro los valores que ocupan dichas posiciones (en un vector de enteros con los valores absolutos más un vector binario con los signos). El algoritmo ha sido testado con las 4 imágenes test ‘tiffany5.pgm’, ‘boat5.pgm’, ‘seis5.pgm’, y ‘squares2.pgm’. De las 4 imágenes las dos primeras son imágenes reales, mientras que las otras dos son imágenes geométricas. Los resultados obtenidos han sido comparados con los formatos standard de compresión JPEG y PNG. Para facilitar una comparación adecuada hemos ofrecido tanto las imágenes reconstruidas después de la compresión como tablas con los datos de tasa de compresión y PSNR, así como gráficas que ayudan a visualizar de manera rápida el comportamiento general de los diferentes algoritmos.

Así hemos podido observar que el algoritmo propuesto en el proyecto es competitivo frente a ambos JPEG y PNG cuando se trabaja con imágenes geométricas. En este caso el mejor sin duda es PNG, si bien es verdad, que cuando la imagen, aunque sea geométrica, contiene ruido, entonces el PNG pierde toda su eficacia y pasa a ser el peor, siendo por tanto preferible aplicar el formato del proyecto. Aquí debemos comentar que la imagen ‘square2.pgm’ tiene una estructura particularmente simple y adaptada a que el JPEG funcione bien. Sin embargo, el comentario anterior se cumplió con otras imágenes geométricas con ejes inclinados.

También observamos que con imágenes reales el JPEG da los mejores resultados. El algoritmo propuesto no es excesivamente peor, mientras que el PNG debe de evitarse. Si la imagen es ruidosa, entonces nuestro algoritmo no empeora tanto como el JPEG o el PNG, aunque continua siendo preferible aplicar JPEG. También debemos matizar que algunas imágenes, aunque reales, presentan una geometría sencilla, y en tales casos, es posible que el algoritmo del proyecto obtenga mejores resultados, como por ejemplo con ’tiffany5.pgm’.

Por último comentar que lo esperable si se plantea un proceso de cuantización adecuado es ganar al formato de compresión JPEG en todos los casos puesto que el algoritmo utilizado de multirresolución viene de los Wavelets Biortogonales, y se sabe

que tienen mayores capacidades de compresión que la Transformada Coseno de Fourier (teoría en la que está basado el JPEG). Por tanto, los resultados aportados en este proyecto son mejorables.

Parte IV (Referencias):

Multirresolución de Harten, aplicada a la compresión de imágenes digitales (conceptos matemáticos):

[1]: Autor: Francesc Aràndiga y Rosa Donat. Título: "Nonlinear Multi-scale Decompositions: The Approach of A. Harten". Revista: "Numerical Algorithms", número 23, páginas: 175-216. Fecha de publicación: año 2.000.

[2]: Autor: Ami Harten. Título: "Discrete multiresolution analysis and generalized wavelets". Revista: "J. Appl. Numer. Math.", número 12, páginas: 153-192. Fecha de publicación: año 1.993.

[3]: Autor: Ami Harten. Título: "Multiresolution representation of data II: General framework". Revista: "SIAM J. Numer. Anal.", número 33 (3), páginas: 1.205-1.256. Fecha de publicación: año 1.996.

[4]: Autor: Ami Harten. Título: "E.N.O. schemes with subcell resolution". Revista: "J. Comput. Phys.", número 83, páginas: 148-184. Fecha de publicación: año 1989.

[5]: Programa informático: "Matlab", versión 6.5.

[6]: Autor: Sergio Amat, Sonia Busquier y Juan Carlos Trillo. Título: "Nonlinear Harten's Multiresolution on the Quincuna Pyramid". Revista: " JCAM, to appear". Fecha de aceptación: año 2005.

[7]: Autor: Sergio Amat, Rosa Donat, Jacques Liandrat y Juan Carlos Trillo. Título: "Analysis of a new nonlinear subdivision scheme. Applications in image processing". Revista: "Foundations of Computational Mathematics". Fecha de aceptación: año 2.005.

Comparación, en bits, con los formatos standard JPEG y PNG (introducción):

[8]: Autor: Pier Luigi Dragotti y Guillermo Barrenechea. Título: "Image Transform Coding". Fecha de publicación: 10 de Enero de 2.002 .

[9]: http://www.psychology.nottinghan.ac.uk/staff/cr1/graphics.html
Web Graphics Notes: GIF, JPEG, or PNG.

[10]: www.iiia.csic.es Created by awstats