The JPEG 2000 Compression Standard

Juan José Bonet Ramis

Adviser: Javier Soria

Advanced Mathematics Master
Universitat de Barcelona
Barcelona, June 2015

Contents

Acknowledgements

I would like to thank the tireless adviser of this Master Thesis, Javier Soria, for his patience and his work. Without him this manuscript would not have been possible.

Chapter 1

Introduction

Nowadays digital images and videos are used in many fields of our live. From the simpler digital camera to the nets of servers in Internet, all of them work with this kind of objects. Since images are stored in these devices, we need to reduce the size of the pictures in order to accumulate as many images as possible in the minimum storage space without losing quality. This necessity has derived in the appearance of several algorithms dedicated to compress images with or without loss of data [2, 5, 11, 17].

A simple image is a matrix of points (pixels) each one indicating a level of tone or color at its spatial position. Therefore, an image can be represented mathematically as a matrix with numbers indicating the value of the pixels at each position. There exist several types of representations for images, but we are going to consider gray scale images where each pixel can take a value between 0 and 255, where the 0 value represents the black color and the 255 value represents the white color. More complex images may have several components, for instance in colored images often each pixel value is given by three components R, G, and B corresponding to its tone of red color, green color, and blue color, respectively. Before processing them, the components are decorrelated transforming them into other three components more suitable to be treated. They are also usually normalized into a symmetric range of values, normally between -1 and 1, to exploit better the capabilities of the ulterior operations. These tasks are performed by the block named Pre-Processing in Figure 1.1, where we can observe the general scheme of an image compression system.

Each image is stored in a device as a vector of bits (elements that only can take the values 0 or 1) known as bit-stream and hence each pixel value is converted into a binary code using a certain number of bits. In general, images can have pixels with higher values at any position, so if we want to reduce the stored size of the image we have to compact the information contained in all the pixels into a reduced number of samples. To accomplish this task most of the algorithms apply a transform to the image that concentrates its energy in a few number of samples.

The DT block (Discrete Transform) in Figure 1.1 performs this operation. In Chapter 2 we introduce several of the transforms used in image processing start-

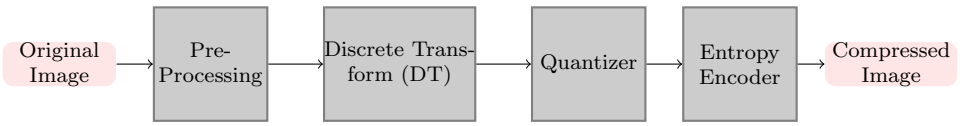

Figure 1.1: Diagram of an image compression system.

ing by the well-known Fourier transform and continuing with the Discrete Cosine Transform [1, 4, 15], used in the JPEG algorithm for image compression, and ending with various types of Discrete Wavelet Transforms [3, 6, 9], including the Haar and Daubechies' D6 wavelets. In this chapter we also develop some interesting properties of these transforms and justify the preference of some of them above the others. We also present several examples of signals transformed using the explained methods.

The ISO/IEC JTC1/SC29/WG1 body, which stands for Working Group 1 of Study Committee 1 of ISO/IEC, more popularly known as Joint Photographic Expert Group (JPEG), developed an image compression standard named JPEG [5] that has become one of the image format most used in the world. In Chapter 3, we explore the basis of this compression format that uses the DCT as discrete transform and we also implement a simple version of the algorithm using Matlab. We also apply this JPEG version to images to study its compression capabilities and the quality of the recovered images after compression. The function of the quantizer block in Figure 1.1 used by the JPEG algorithm to reduce the number of significant samples in order to compress the image is also explained in this chapter. Finally, we close Chapter 3 explaining the entropy encoder which plays the role of generating a compressed bit-stream from the coefficients created by the quantizer block. In particular two techniques of coding are developed, the run length encoding (RLE) and the Huffmann code [10, 16], enriched with an illustrative example.

The Joint Photographic Expert Group created, in 2000, a new still image compression standard, the JPEG 2000 [17], designated to replace the old JPEG. This new format was designed to fulfill several characteristics to improve the older format and satisfy new necessities that appeared later in time. Some of the innovating features of the new format are a better compression efficiency, the ability to enhance the quality associated to selected spatial regions in the image, the ability to work with enormous images without breaking them into smaller tiles, and the progressive lossy to lossless performance within a single data stream. Hence, one of main characteristics of the new format is scalability. This property is due to the combination of two facts: the use of Discrete Wavelet Transforms and the application of an entropy coder, named MQ-coder, to the samples coming from the wavelet transform, that constructs an embedded bit-stream that can be truncated at several selected points in order to obtain an image with less quality. Chapter 4 is devoted to explain the JPEG 2000 standard and a simple implementation is also constructed using Matlab. The key aspects of JPEG 2000 studied in this chapter are the standard wavelet transform used by the algorithm (Bi-orthogonal 9-7 wavelet) and the entropy coder. The Bi-orthogonal 9-7 wavelet is analyzed from the filtering view point and a fast

implementation known as *lifting scheme* [7] is presented. Before the entropy coder is explained, we introduce some theory about arithmetic coding (Elias code), its finite precision implementation and the context labeling used by the JPEG 2000 in order to compute the probabilities of the samples needed by any entropy coder. Next, we explain the coding order followed by the algorithm to exploit the context labels and feed the MQ-coder [12, 13] in an efficient manner. We finish Chapter 4 with the construction of the bit-stream in a layer fashion way, so that each layer in the bit-stream contains information of the image that permit its reconstruction at increasing resolution levels, each layer decoded provide more information letting us to obtain a higher resolution version of the image.

Finally, in Chapter 5 we show a comparative between JPEG and JPEG 2000 standards using the algorithms that we have implemented using Matlab that can be found in Appendix A. We apply the two compression methods to four test images presenting the recovered images after compression at different bit-rates. We analyze and compare the obtained results from a subjective view point and using a distortion measure.

Chapter 2

Preliminaries

In Science and Engineering people work with signals coming from sensors to monitor and control some processes in order to ensure good product quality. These signals usually are time dependent or can depend on several variables, space variables for example, as in the case of images. For mathematicians these signals are just one or several variables real or complex functions. The goal of signal processing is to extract information, which is not directly available, and reveal the underlying dynamics from a given signal using transformations to see the signal in another way. Mathematically, this can be achieved by representing the time-domain signal as a series of coefficients, based on a comparison between the signal and a set of template functions which will be a basis for the space of the signal. Signals obtained experimentally are usually sampled at discrete time intervals, instead of continuously, along a total measurement time T, so they can be represented by a vector containing these samples.

Through history, several basis have been used in order to achieve this goal. In this chapter we will introduce three of the most popular basis: the discrete Fourier transform, the discrete cosine transform and the wavelets. Most of the results presented in this chapter can be found in [9].

2.1 The Discrete Fourier Transform

The Fourier transform is probably one of the most widely used instruments in signal processing. This tool reveals the frequency composition of a time series signal by transforming it from the time domain to the frequency domain. This technique was originally used by the French mathematician Joseph Fourier in 1807, when he found that any periodically function could be expressed as a weighted sum of sinus and cosines functions. This section presents the discrete Fourier transform and states some of its properties.

2.1.1 Definition and properties

Definition 2.1.1. Define $\mathbb{Z}_N = \{0, 1, \ldots, N-1\}$.

Definition 2.1.2. Define the vector space over $\mathbb{C}$

$$\ell^2(\mathbb{Z}_N) = \big\{ z = (z(0), z(1), \ldots, z(N-1)) \;\big|\; z(j) \in \mathbb{C},\ 0 \le j \le N-1 \big\},$$

endowed with the scalar product

$$\langle z, v \rangle = \sum_{n=0}^{N-1} z(n)\overline{v(n)}.$$

This inner product induces the ℓ^2-norm given by

$$\|z\| = \sqrt{\langle z, z \rangle} = \sqrt{\sum_{n=0}^{N-1} z(n)\overline{z(n)}} = \sqrt{\sum_{n=0}^{N-1} |z(n)|^2}. \tag{2.1}$$

Definition 2.1.3. Define $E_m(n) = \frac{1}{\sqrt{N}} e^{2\pi i m n/N} \in \ell^2(\mathbb{Z}_N)$, for $0 \le m, n \le N-1$.

Lemma 2.1.4. *The set* $\{E_m : 0 \le m \le N-1\}$ *is an orthonormal basis for* $\ell^2(\mathbb{Z}_N)$.

Proof. If we compute the inner product of E_m and E_n, $0 \le m, n \le N-1$ and $n \ne m$, in $\ell^2(\mathbb{Z}_N)$

$$\begin{aligned}\langle E_m, E_n \rangle &= \frac{1}{N} \sum_{k=0}^{N-1} e^{-2\pi i k n/N} e^{2\pi i m k/N} = \frac{1}{N} \sum_{k=0}^{N-1} e^{-2\pi i k (n-m)/N} = \frac{1}{N} \frac{1 - e^{-2\pi i N (n-m)/N}}{1 - e^{-2\pi i (n-m)/N}} \\ &= \frac{1}{N} \frac{1-1}{1 - e^{-2\pi i (n-m)/N}} = 0,\end{aligned}$$

since $e^{-2\pi i N(n-m)/N} = e^{-2\pi i (n-m)} = 1$ and $e^{-2\pi i (n-m)/N} \ne 0$, because $n \ne m$. Now, if $n = m$ we have

$$\langle E_n, E_n \rangle = \frac{1}{N} \sum_{k=0}^{N-1} e^{-2\pi i k n/N} e^{2\pi i k n/N} = \frac{1}{N} \sum_{k=0}^{N-1} 1 = \frac{1}{N} N = 1.$$

□

Definition 2.1.5. Let $\mathbf{z} = (z(0), \ldots, z(N-1)) \in \ell^2(\mathbb{Z}_N)$. For $0 \le m \le N-1$, we define the *Discrete Fourier Transform (DFT)* of $\mathbf{z}$ as the map

$$\begin{aligned}\ell^2(\mathbb{Z}_N) &\xrightarrow{\widehat{}} \ell^2(\mathbb{Z}_N) \\ z &\mapsto (\hat{z}(0), \ldots, \hat{z}(N-1)),\end{aligned}$$

where

$$\hat{z}(m) = \sum_{n=0}^{N-1} z(n) e^{-i 2\pi n m/N}.$$

Observe that we can extend this definition to all $m \in \mathbb{Z}$. Since

$$\begin{aligned}\hat{z}(m+N) &= \sum_{n=0}^{N-1} z(n)e^{-i2\pi n(m+N)/N} = \sum_{n=0}^{N-1} z(n)e^{-i2\pi nm/N}e^{-i2\pi n} \\ &= \sum_{n=0}^{N-1} z(n)e^{-i2\pi nm/N} = \hat{z}(m),\end{aligned}$$

we have that the DFT is periodic of period N.

The DFT has multiple applications in signal and image processing. In fact, it maps the a time domain signal into its frequency domain.

Theorem 2.1.6. *Let $z, v \in \ell^2(\mathbb{Z}_N)$. Then,*

1. *(Inversion Formula). For all $n \in \{0, \dots, N-1\}$*

$$z(n) = \frac{1}{N}\sum_{k=0}^{N-1} \hat{z}(k)e^{i2\pi kn/N}. \tag{2.2}$$

2. *(Parseval's identity)*

$$\langle z, v\rangle = \sum_{k=0}^{N-1} z(k)\overline{v(k)} = \frac{1}{N}\sum_{k=0}^{N-1} \hat{z}(k)\overline{\hat{v}(k)} = \frac{1}{N}\langle \hat{z}, \hat{v}\rangle.$$

3. *(Plancherel's formula)*

$$\|z\|^2 = \sum_{k=0}^{N-1} |z(k)|^2 = \frac{1}{N}\sum_{k=0}^{N-1} |\hat{z}(k)|^2 = \frac{1}{N}\|\hat{z}\|^2.$$

Proof. 1. Since $\{E_0, \dots, E_{N-1}\}$ is an orthonormal basis we can write $z \in \ell^2(\mathbb{Z}_N)$ as

$$z(n) = \sum_{k=0}^{N-1} \langle z, E_k\rangle E_k(n).$$

But, $\langle z, E_k\rangle = \sum_{m=0}^{N-1} z(m)\overline{E_k(m)} = \frac{1}{\sqrt{N}}\hat{z}(k)$, and hence

$$\begin{aligned}z(n) &= \sum_{k=0}^{N-1} \langle z, E_k\rangle E_k(n) = \sum_{k=0}^{N-1} \frac{1}{\sqrt{N}}\hat{z}(k)E_k(n) = \sum_{k=0}^{N-1} \frac{1}{\sqrt{N}}\hat{z}(k)\frac{1}{\sqrt{N}}e^{i2\pi kn/N} \\ &= \frac{1}{N}\sum_{k=0}^{N-1} \hat{z}(k)e^{i2\pi kn/N}.\end{aligned}$$

2. Using the fact that $\{E_0, \ldots, E_{N-1}\}$ is an orthonormal basis we have that

$$\begin{aligned}\langle z, v\rangle &= \left\langle \sum_{n=0}^{N-1} \langle z, E_n\rangle E_n, \sum_{k=0}^{N-1} \langle v, E_k\rangle E_k \right\rangle = \sum_{n=0}^{N-1}\sum_{k=0}^{N-1} \overline{\langle v, E_k\rangle}\langle z, E_n\rangle \langle E_n, E_k\rangle \\ &= \sum_{n=0}^{N-1} \frac{1}{\sqrt{N}}\overline{\hat{v}(n)}\frac{1}{\sqrt{N}}\hat{z}(n) = \frac{1}{N}\sum_{n=0}^{N-1}\overline{\hat{v}(n)}\hat{z}(n).\end{aligned}$$

3. The last property can be proved from Parseval's identity imposing $v = z$:

$$\langle z, z\rangle = \|z\|^2 = \frac{1}{N}\sum_{n=0}^{N-1}\overline{\hat{z}(n)}\hat{z}(n) = \frac{1}{N}\|\hat{z}\|^2.$$

□

Definition 2.1.7. The Fourier basis for $\ell^2(\mathbb{Z}_N)$ is the set $F = \{F_m : 0 \leq m \leq N-1\}$ where $F_m \in \ell^2(\mathbb{Z}_N)$ is given by

$$F_m(n) = \frac{1}{N}e^{-i2\pi nm/N} = \frac{1}{N}w^{nm},$$

where we have defined $w = e^{-i2\pi/N}$.

Observe that, since $F_m = \frac{1}{\sqrt{N}}E_m$, F is an orthogonal basis of $\ell^2(\mathbb{Z}_N)$ and we can write $z = \sum_{m=0}^{N-1}\hat{z}(m)F_m = F\hat{z}$, where F is a matrix which columns are the vectors F_m, with $m = 0, \ldots, N-1$. We say that the DFT components $\hat{z}(m)$ are the components of z in the Fourier basis.

From the inversion formula we know that the the map $\hat{}\colon \ell^2(\mathbb{Z}_N) \to \ell^2(\mathbb{Z}_N)$ is a one to one map, i.e., if $\hat{z} = \hat{v}$, then $z = v$. Hence, we can define the inverse of the DFT.

Definition 2.1.8. Let $z \in \ell^2(\mathbb{Z}_N)$, we define the Inverse Discrete Fourier Transform of z as the vector $\check{z} \in \ell^2(\mathbb{Z}_N)$, where

$$\check{z}(n) = \frac{1}{N}\sum_{k=0}^{N-1} z(k)e^{i2\pi kn/N}, \text{ for } n \in \{0, \ldots, N-1\}.$$

Now, since the DFT is an invertible linear transformation, we have that $z = F^{-1}\hat{z}$. We can determine the elements of F^{-1} using the previous definition.

$$\check{z}(n) = \sum_{k=0}^{N-1} z(k)\frac{1}{N}e^{i2\pi kn/N} = \sum_{k=0}^{N-1} z(k)\frac{1}{N}w^{-kn} = Fz.$$

Therefore, the elements of the inverse matrix F^{-1} are $(F^{-1})_{kn} = \frac{1}{N}w^{-kn} = \frac{1}{N}\overline{w^{kn}}$ and we get

$$F^{-1} = \frac{1}{N}\overline{F}.$$

Definition 2.1.9. Given a vector $z \in \ell^2(\mathbb{Z}_N)$ and $k \in \mathbb{Z}$. We define the translate of z by k as

$$R_k z(n) = z(n-k), \quad n \in \mathbb{Z}.$$

Lemma 2.1.10. *Let $z \in \ell^2(Z_N)$ and $k \in \mathbb{N}$. Then,*

$$\widehat{(R_k z)}(n) = \hat{z}(n) e^{-i2\pi kn/N}, \quad \textit{for all } n \in \mathbb{Z}.$$

Proof.

$$\widehat{(R_k z)}(n) = \sum_{m=0}^{N-1} z(m-k) e^{-i2\pi mn/N}.$$

We change variables, $l = m - k$ and we get

$$\widehat{(R_k z)}(n) = \sum_{l=-k}^{N-1-k} z(l) e^{-i2\pi(l+k)n/N} = e^{-i2\pi kn/N} \sum_{l=-k}^{N-1-k} z(l) e^{-i2\pi ln/N}.$$

Now, since $z(l)$ and $e^{-i2\pi ln/N}$ are both periodic functions of period N we claim that

$$\sum_{l=-k}^{N-1-k} z(l) e^{-i2\pi ln/N} = \sum_{l=0}^{N-1} z(l) e^{-i2\pi ln/N} = \hat{z}(n)$$

and

$$\widehat{(R_k z)}(n) = e^{-i2\pi kn/N} \sum_{l=0}^{N-1} z(l) e^{-i2\pi ln/N} = e^{-i2\pi kn/N} \hat{z}(n).$$

If $k = 0$, the result is trivial. Let $0 < k \leq N-1$, then

$$\begin{aligned}
\sum_{l=-k}^{N-1-k} z(l) e^{-i2\pi ln/N} &= \sum_{l=-k}^{-1} z(l) e^{-i2\pi ln/N} + \sum_{l=0}^{N-1-k} z(l) e^{-i2\pi ln/N} \\
&= \sum_{l=-k}^{-1} z(l+N) e^{-i2\pi(l+N)n/N} + \sum_{l=0}^{N-1-k} z(l) e^{-i2\pi ln/N} \\
&= \sum_{m=N-k}^{N-1} z(m) e^{-i2\pi mn/N} + \sum_{m=0}^{N-1-k} z(m) e^{-i2\pi mn/N} \\
&= \sum_{m=0}^{N-1} z(m) e^{-i2\pi mn/N}.
\end{aligned}$$

Now, if $k \in \mathbb{Z}$, then there is some $r \in \mathbb{Z}$ such that $k' = k + rN \in \{0, \ldots, N-1\}$. If we change the summation variable $l' = l - rN$, then

$$\sum_{l=-k}^{N-1-k} z(l) e^{-i2\pi ln/N} = \sum_{l'=-k-rN}^{N-1-k-rN} z(l'+rN) e^{-i2\pi(l'+rN)n/N} = \sum_{l'=-k'}^{N-1-k'} z(l') e^{-i2\pi l'n/N},$$

and we are in the case $k' \in \{0, \ldots, N-1\}$ considered above. □

Definition 2.1.11. Let $u, v \in \ell^2(\mathbb{Z}_N)$. The convolution of u and v is the vector $u * v \in \ell^2(\mathbb{Z}_N)$ with components given by

$$(u * v)(n) = \sum_{k=0}^{N-1} u(k)v(n-k),$$

for all n.

Lemma 2.1.12. *Let $u, v \in \ell^2(\mathbb{Z}_N)$, then for all k*

$$\widehat{(u * v)}(k) = \hat{u}(k)\hat{v}(k). \tag{2.3}$$

Proof. From the definition of convolution

$$\begin{aligned}
\widehat{(u * v)}(k) &= \sum_{n=0}^{N-1} (u * v)(n) e^{-i2\pi nk/N} = \sum_{n=0}^{N-1}\sum_{m=0}^{N-1} u(m)v(n-m)e^{-i2\pi nk/N} \\
&= \sum_{n=0}^{N-1}\sum_{m=0}^{N-1} u(m)v(n-m)e^{-i2\pi(n-m)k/N}e^{-i2\pi mk/N} \\
&= \sum_{m=0}^{N-1} u(m)e^{-i2\pi mk/N} \sum_{n=0}^{N-1} v(n-m)e^{-i2\pi(n-m)k/N} = \hat{u}(k)\hat{v}(k).
\end{aligned}$$

□

We have defined the DFT in one dimension. This tool let us analyze one dimensional signals, for instance signals which depend on time such as audio signals. If we want to deal with images we can also use this tool in image processing, but since images are bi-dimensional, we have to define the 2-dimensional Fourier transform.

Definition 2.1.13. Let $z \in \ell^2(\mathbb{Z}_M \times \mathbb{Z}_N)$. The Discrete Fourier Transform of z is defined as

$$\hat{z}(k,l) = \sum_{n=0}^{N-1}\sum_{m=0}^{M-1} z(m,n)e^{-i2\pi(km/M+ln/N)},$$

for $k \in \{0, \ldots, M-1\}$ and $l \in \{0, \ldots, N-1\}$.

2.1.2 The Fast Fourier Transform

In signal processing we need to process signals fast and the DFT requires a large amount of operations and time. For instance, if $z \in \ell^2(\mathbb{Z}_N)$ and we want to compute its DFT, $\hat{z} \in \ell^2(\mathbb{Z}_N)$, we have to multiply a $N \times N$ matrix F with a vector $z \in \ell^2(\mathbb{Z}_N)$: $\hat{z} = Fz$. This requires N^2 complex multiplications. A priori one can think that multiplying two complex numbers requires four real multiplications, but in fact, using a little trick, we can perform this product with only three real multiplications. If $z_1 = a_1 + ib_1 \in \mathbb{C}$ and $z_2 = a_2 + ib_2 \in \mathbb{C}$ and we compute the product

$$z_1 z_2 = (a_1 a_2 - b_1 b_2) + i(a_1 b_2 + a_2 b_1),$$

we do 4 operations, but if we define

$$m = a_1(a_2 + b_2), \quad n = b_2(a_1 + b_1), \quad \textit{and} \quad p = b_1(a_2 - b_2),$$

then $z_1 z_2 = (m - n) + i(n + p)$. This technique enlarges the number of additions, but since the computational cost of multiplications is much larger than the cost of additions we do not consider them. One of the main advantages of the DFT is that there exists a fast algorithm, called the Fast Fourier Transform (FFT), to compute it. This algorithm is specially suitable when N is a power of 2.

We start with a simpler case considering that our vectors have even dimension.

Lemma 2.1.14. *Suppose $M \in \mathbb{N}$ and $N = 2M$. Let $z \in \ell^2(\mathbb{Z}_N)$. Define $u, v \in \ell^2(\mathbb{Z}_M)$ by*

$$u(k) = z(2k), \ \textit{for } k = 0, \dots, M-1,$$

and

$$v(k) = z(2k+1), \ \textit{for } k = 0, \dots, M-1.$$

Let $\hat{z}$ be the DFT of z on N points, i.e., $\hat{z} = F_N z$, and $\hat{u}$, $\hat{v}$ the DFT of u and v, respectively, on $M = N/2$ points, that is, $\hat{u} = F_M u$ and $\hat{v} = F_M v$. Then, for $m = 0, \dots, M-1$,

$$\hat{z}(m) = \hat{u}(m) + \hat{v}(m) e^{-i2\pi m/N}, \tag{2.4}$$

and for $m = M, M+1, \dots, N-1$, let $l = m - M$. Then

$$\hat{z}(m) = \hat{z}(l + M) = \hat{u}(l) - \hat{v}(l) e^{-i2\pi l/N}. \tag{2.5}$$

Proof. For $m = 0, \dots, N-1$,

$$\begin{aligned}
\hat{z}(m) &= \sum_{n=0}^{N-1} z(n) e^{-i2\pi nm/N} = \sum_{k=0}^{M-1} z(2k) e^{-i2\pi 2km/N} + \sum_{k=0}^{M-1} z(2k+1) e^{-i2\pi(2k+1)m/N} \\
&= \sum_{k=0}^{M-1} u(k) e^{-i2\pi km/(N/2)} + e^{-i2\pi m/N} \sum_{k=0}^{M-1} v(k) e^{-i2\pi km/(N/2)} \\
&= \sum_{k=0}^{M-1} u(k) e^{-i2\pi km/M} + e^{-i2\pi m/N} \sum_{k=0}^{M-1} v(k) e^{-i2\pi km/M},
\end{aligned}$$

and for $m = 0, \dots, M-1$ this expression coincides with $\hat{u}(m) + e^{-i2\pi m/N} \hat{v}(m)$. Now, if $m = M, M+1, \dots, N-1$ we make the variable change $m = l + M$ and we get

$$\begin{aligned}
\hat{z}(m) &= \sum_{k=0}^{M-1} u(k) e^{-i2\pi k(l+M)/M} + e^{-i2\pi(l+M)/N} \sum_{k=0}^{M-1} v(k) e^{-i2\pi k(l+M)/M} \\
&= \sum_{k=0}^{M-1} u(k) e^{-i2\pi kl/M} - e^{-i2\pi l/N} \sum_{k=0}^{M-1} v(k) e^{-i2\pi kl/M} = \hat{u}(l) - e^{-i2\pi l/N} \hat{v}(l),
\end{aligned}$$

because the function $e^{-i2\pi kl/M}$ is periodic of period M and $e^{-i2\pi M/N} = -1$, when $M = N/2$. □

Given $z \in \ell^2(\mathbb{Z}_N)$ the procedure to compute the FFT consists on compute the FFT for u and v which requires M^2 multiplications for each one. After that, we compute $\hat{z}(m)$ for $m = 0, \ldots, M-1$ using (2.4), which requires M multiplications of the terms $\hat{v}(m)e^{-i2\pi m/N}$. Finally, using (2.5), we compute $\hat{z}(m)$ for $m = M, \ldots, N-1$ from the previous values of $\hat{z}(m)$, for $m = 0, \ldots, M-1$. This last part requires only additions, because we already know all the terms of the equation. Hence, we can compute the FFT with a total of $2M^2 + M = \frac{1}{2}(N^2 + N) = \frac{N}{2}(N+1)$ products. For N large this number of multiplications is approximately equal to $\frac{N^2}{2}$. Hence, we can reduce in half the amount of operations to compute the DFT. In fact, if $M = 4N$ we can apply the same technique to compute the DFT of u and v and we will need M^2 operations for both terms instead of $2M^2$. The method can be extended to the limit if N is a power of two. In this case we can see that the number of complex multiplications to compute will be of the order of $\frac{1}{2}N \log_2 N$.

Lemma 2.1.15. *Let $N = 2^n$, for some $n \in \mathbb{N}$. Then, the number of complex multiplications $\#_N$ to compute the FFT of a vector of length N satisfies*

$$\#_N \leq \frac{1}{2} N \log_2 N.$$

Proof. We proceed by induction. For $n = 1$ we have a vector $z = (a, b)$ of length 2. The FFT will be $\hat{z} = (a + b, a - b)$ and we do not need any multiplication. It holds that

$$\#_2 = 0 \leq \frac{1}{2} 2 \log_2 2 = 1.$$

Suppose, by induction, that the result holds for $n = k - 1$. Then, for $n = k$ we have that

$$\begin{aligned} \#_N &\leq 2\#_{N/2} + N/2 = 2\#_{2^{k-1}} + 2^{k-1} \leq 2\frac{1}{2}2^{k-1}\log_2(2^{k-1}) + 2^{k-1} \\ &= (k-1)2^{k-1} + 2^{k-1} = k2^{k-1} = \frac{1}{2}k2^k = \frac{1}{2}N \log_2 N. \end{aligned}$$

□

The FFT has many applications in signal processing, but one of the most important consequences of this algorithm is given by equation (2.3) which implies that we can compute a complex operation, as convolution, much faster using the FFT.

2.2 The Discrete Cosine Transform

As the Discrete Fourier Transform, the Discrete Cosine Transform (DCT) uses an orthogonal basis constructed from cosine functions which elements are real valued. This transform was originally introduced in 1974 by N. Ahmed, T. Natarajan and K. R. Rao, see [1]. There are several DCT basis that can be derived from the eigenvectors of symmetric second-difference matrices, see [15], yielding to DCT-1,

DCT-2, DCT-3 and DCT-4 types, but we are going to focus on the DCT-2 basis, which we call DCT, because it is the one used in the standard JPEG for image compression.

Definition 2.2.1. Given $M \in \mathbb{N}$, the Discrete Cosine basis for $\ell^2(\mathbb{Z}_M)$ is defined as

$$\left\{ \left\{ \frac{1}{\sqrt{2}} \right\} \cup \left\{ \cos j(k+\frac{1}{2})\frac{\pi}{N} \right\} : \quad 0 \leq j \leq M-1, 0 < k \leq M-1 \right\}.$$

Definition 2.2.2. Given $z \in \ell^2(\mathbb{Z}_M)$, the Discrete Cosine Transform (DCT) of z is defined as

$$\begin{cases} G_z(0) = \frac{\sqrt{2}}{M} \sum_{m=0}^{M-1} z(m) \\ G_z(k) = \frac{2}{M} \sum_{m=0}^{M-1} z(m) \cos k(m+\frac{1}{2})\frac{\pi}{M}, \quad k = 1, \ldots, M-1, \end{cases}$$

where $G_z(k)$ is the kth DCT coefficient.

Definition 2.2.3. The Inverse Discrete Cosine Transform (IDCT) is defined as

$$z(k) = \frac{1}{\sqrt{2}} G_z(0) + \sum_{m=1}^{M-1} G_z(m) \cos m \left(k+\frac{1}{2}\right) \frac{\pi}{M}, \quad k = 0, \ldots, M-1,$$

where $G_z(k)$ is the kth DCT coefficient.

Proposition 2.2.4. *Let $z \in \ell^2(\mathbb{Z}_M)$ and $N = 2M$, then*

$$G_z(k) = \frac{2}{M} \Re \left\{ e^{ik\pi/N} \sum_{m=0}^{N-1} z(m) w^{km} \right\}, k = 1, \ldots, M-1, \tag{2.6}$$

where $w = e^{-i2\pi/N}$ and $z(m) = 0$, for $m = M, M+1, \ldots, N-1$.

Equation (2.6) shows us that there is a relation between the DCT and the DFT. This is an important fact because it let us to compute the DCT and the IDCT very fast using the FFT algorithm.

The DCT is used in the JPEG algorithm for image compression. The image is decomposed into blocks of 8×8 pixels and a DCT is applied to each block. Hence, we need to define the bi-dimensional DCT in order to study, in the next chapter, the JPEG algorithm.

Definition 2.2.5. Let $A \in \ell^2(\mathbb{Z}_M \times \mathbb{Z}_N)$. The Discrete Cosine Transform (DCT) of A is given by

$$G_A(m,n) = \frac{4C_1(m)C_2(n)}{MN} \sum_{j=0}^{M-1} \sum_{k=0}^{N-1} A(j,k) \cos m \left(j+\frac{1}{2}\right) \frac{\pi}{M} \cos n \left(k+\frac{1}{2}\right) \frac{\pi}{N}, \tag{2.7}$$

where $C_1(M-1) = C_2(N-1) = \frac{1}{\sqrt{2}}$, $C_1(m) = 1$, for $0 \leq m < M-1$ and $C_2(n) = 1$, for $0 \leq n < N-1$.

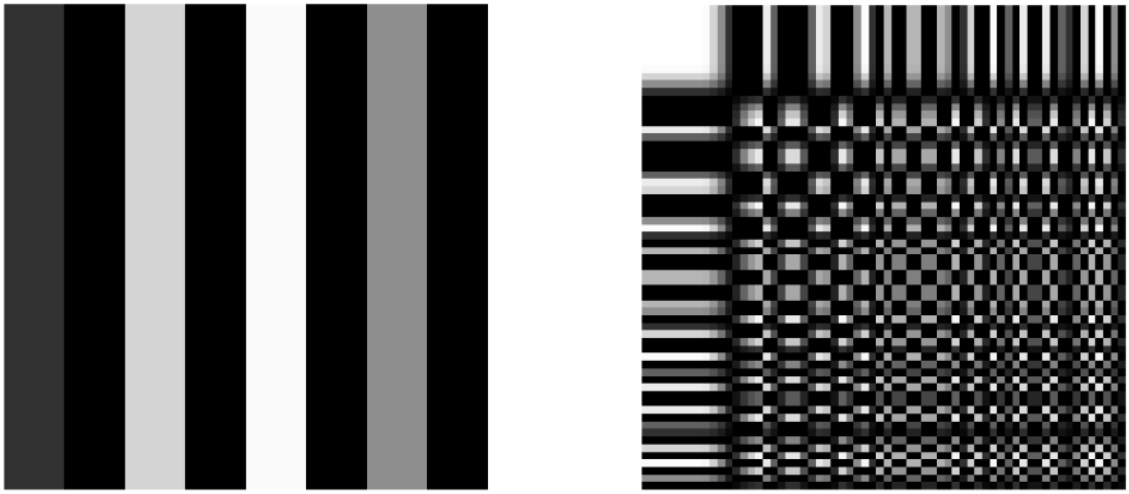

Figure 2.1: Discrete cosine for $m = 1$, $n = 8$. Discrete cosine basis for $N = M = 8$.

Definition 2.2.6. Let $G_A \in \ell^2(\mathbb{Z}_M \times \mathbb{Z}_N)$. The Inverse Discrete Cosine Transform (IDCT) of G_A is given by

$$A(m,n) = \sum_{j=0}^{M-1} \sum_{k=0}^{N-1} C_1(j) C_2(k) G_A(j,k) \cos m \left(j + \frac{1}{2}\right) \frac{\pi}{M} \cos n \left(k + \frac{1}{2}\right) \frac{\pi}{N},$$

where $C_1(M-1) = C_2(N-1) = \frac{1}{\sqrt{2}}$, $C_1(m) = 1$, for $0 \leq m < M-1$ and $C_2(n) = 1$, for $0 \leq n < N-1$.

The left image of Figure 2.1 shows the discrete cosine function for $m = 1$ and $n = 8$, when $N = M = 8$. On the right hand side we show the Discrete Cosine basis used in JPEG compression.

2.2.1 Comparison between DFT and DCT

The reason why the DCT is preferred to DFT in JPEG is that the DCT requires fewer coefficients than DFT to get a good approximation to a typical signal [4]. The goal of compression is to reduce the size of a signal or an image minimizing the loss of quality, hence we expect that after a suitable transformation of our signal we get smaller coefficients corresponding to the higher frequencies that can be rejected, keeping only some coefficients from the low frequencies.

Suppose we have the ramp signal in Figure 2.2 (above) and we apply a DCT and a DFT to this signal. After that, we truncate both transformed signals keeping only the two first coefficients and we apply the corresponding inverse transform to reconstruct an approximation of the original signal. Figure 2.2 shows the reconstructed signal using both the DCT (middle) and the FFT (below). We can observe how the DCT approximates better than the DFT to the original signal only keeping the

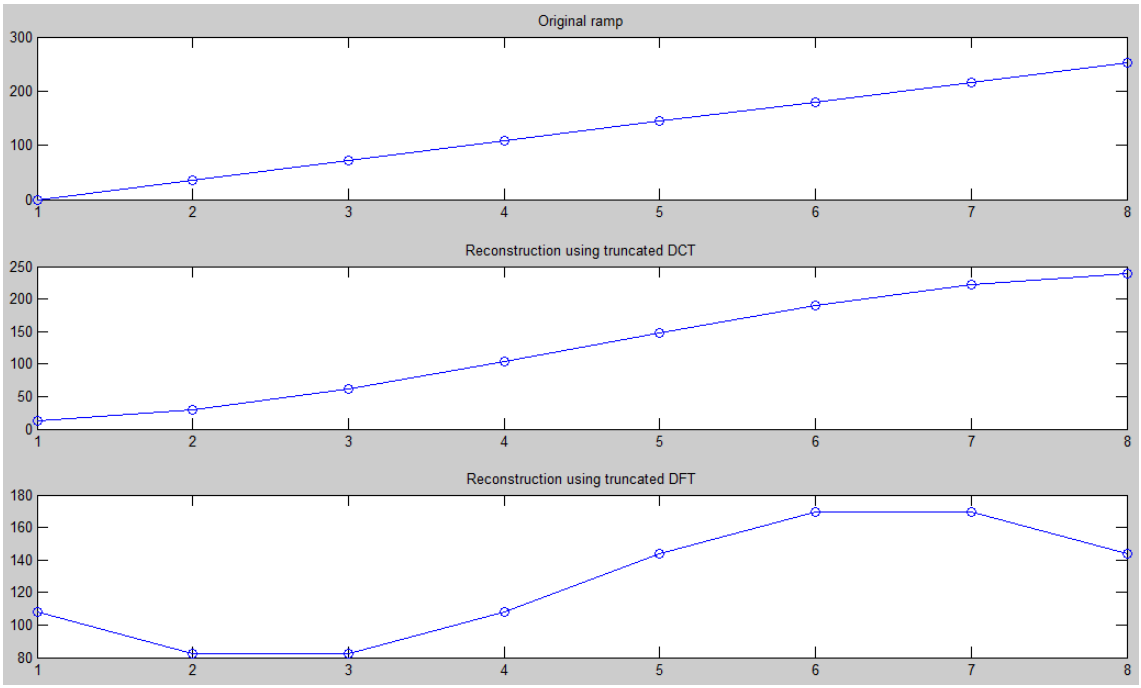

Figure 2.2: Comparison of accuracy of truncated DCT and DFT.

two first coefficients. Table 2.1 shows the eight values of the ramp signal with the corresponding coefficients of the DCT and the DFT (since the DFT has complex values we show the absolute value of the coefficients) and it can be observed how the DCT keeps the most part of the energy of the signal in the low-frequency coefficients while the DFT has larger coefficients corresponding to high frequencies.

x	**DCT**	**FFT**
0	356.3818	1008.0
36	-231.9236	376.3
72	0	203.6
108	-24.2444	155.9
144	0	144.0
180	-7.2325	155.9
216	0	203.6
252	-1.8253	376.3

Table 2.1: DCT and DFT coefficients for the ramp signal.

The explanation to this fact is easy: the DFT approximation is trying to reproduce a periodic ramp signal, i.e. a sawtooth wave, while the DCT reproduces a triangle wave, i.e. a symmetric ramp. Therefore, the DFT has greater high-frequency coefficients to reproduce the spurious discontinuity at the end of the ramp in the sawtooth wave, while the DCT has no high-frequency coefficients because the trian-

gle wave has no discontinuity. The DCT is thus better modeling finite-length signals that have fairly different values on their right and left sides.

To finish this section it is worth to present another point of view about the process that the DCT develops over images. If we consider an 8×8 block of an image, we can see each row of 8 pixels as a point in the eight-dimensional space; the eight-dimensional points of a typical image will form a cluster of points in this space not uniformly dispersed, on the contrary it is squashed fairly flat in some directions. The DCT rotates this cluster to line up this directions along some of the coordinate axes. Then we can represent the points of the cluster in a lower dimensional space. The idea behind this is called the Karhunen-Loève transform (KLT) [1] and DCT approximates it pretty close.

Definition 2.2.7. Given a random vector $z = (z(1), \ldots, z(N))^T \in \ell^2(\mathbb{Z}_N)$, we define its covariance matrix as

$$C_z = \mathrm{E}[(z - m)\overline{(z - m)}],$$

where $\mathrm{E}(\cdot)$ denotes the expectation operator and $m = \mathrm{E}(z)$ is the mean of the vector.

We define now the Karhunen-Loève transform.

Definition 2.2.8. Given a random vector $z = (z(1), \ldots, z(N))^T \in \ell^2(\mathbb{Z}_N)$, with covariance matrix C_z, its Karhunen-Loève transform is given by

$$y = Wz,$$

where W is an orthonormal matrix such that the covariance C_y of the vector y is a diagonal matrix.

The idea behind this transform is the following: the pixels of an image are correlated with their neighbors and hence there is redundant information in the image. If we decorrelate the pixels we can compress the image without losing information. Let us see an example: Figure 2.3 shows the scatter plot of the gray values of the pixels for Lenna's gray image. The x component shows the gray level of a pixel and the component y is the gray level of its right neighbor. We can observe the strong correlation along the line $x = y$. If we rotate the figure 45^o we obtain the decorrelated version of the distribution as we can see in Figure 2.4. Now the y components are uncorrelated from the x ones and we can encode these components using a smaller number of bits.

Now suppose that the mean of the z vector is zero. Hence, the problem consists on finding an orthogonal transformation represented by the matrix W ($W\overline{W} = I$) such that

$$C_y = \mathrm{E}(y\overline{y}) = \mathrm{E}(Wz\overline{Wz}) = \mathrm{E}(Wz\overline{z}\,\overline{W}) = W\,\mathrm{E}(z\overline{z})\overline{W} = WC_z\overline{W}.$$

In other words, since $W\overline{W} = I$,

$$C_yW = WC_z.$$

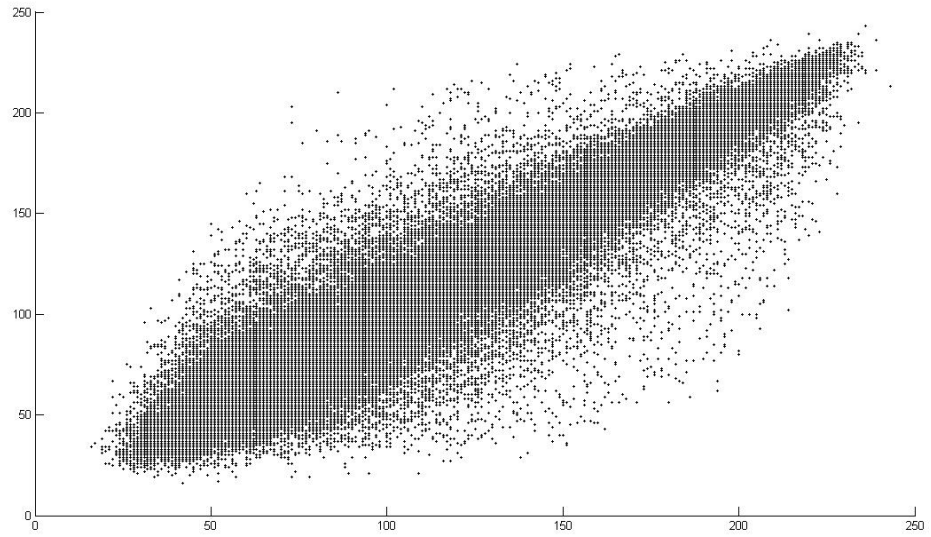

Figure 2.3: Scatter plot of adjacent pixel value pairs for Lenna's image.

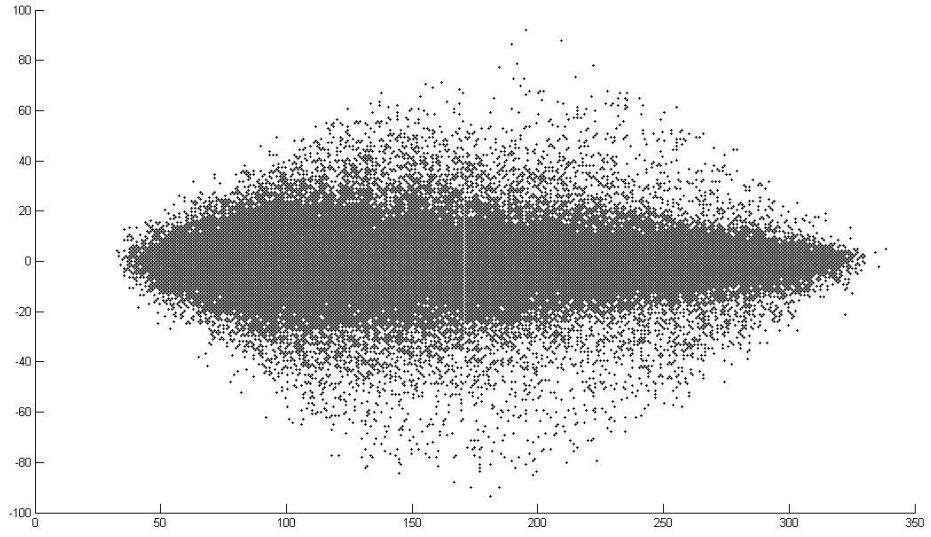

Figure 2.4: Scatter plot of adjacent pixel value pairs for Lenna's decorrelated image.

Now, if we impose that C_y is diagonal with elements λ_i, for $i = 1, \ldots, N$, we have that

$$\lambda_i w_i = C_z w_i, \quad i \in \{1, \ldots, N\},$$

where w_i is the *i-th* column of the matrix W. Therefore, the columns of the matrix transform W are the eigenvectors of the covariance matrix of z. This transformation defined by the eigenvectors of the covariance matrix of the original data is the Karhunen-Loève transform (see [8]).

Since we consider an image as a vector sample of a random variable which distribution we do not know, we have to estimate its covariance matrix from the data. We can compute such estimate from n data samples, for $z_k \in \ell^2(\mathbb{Z}_N)$, $k = 1, \ldots, n$, as

$$\tilde{C}_z = \frac{1}{n} \sum_{k=1}^{N} z_k \overline{z_k}.$$

This fact makes the KLT content dependent and it lacks fasts computation algorithm, therefore it is not suitable for compression purposes.

2.3 The Discrete Wavelet Transform

We have seen in previous sections that the FFT is very fast to compute and can approximate fairly well periodic signals. On the other hand the DCT can be computed using the FFT algorithm and is well equipped to model finite-length pieces of functions that have fairly different values on their left and right sides. The main problem of these transforms is that their basis elements are not well localized in space in the sense that they have many elements different from, or not close to, zero.

Definition 2.3.1. Given a vector $z \in \ell^2(\mathbb{Z}_N)$ we will say that it is localized near $n_0 \in \mathbb{N}$ if most of the components $z(n)$ of z are zero or at least relatively small, except for a few values of n near n_0.

A Fourier basis element $F_m(n) = \frac{1}{N} e^{i2\pi mn/N}$ is not localized in space because all its elements have the same magnitude $\frac{1}{N} \neq 0$.

Having a spatially localized basis is very useful because it provides a local analysis of a signal. For example in medical image processing we can study better a tumour if we focus on the region where it is localized. But the application we are interested in is image compression, for instance, if we have a localized basis in frequency, we can remove high-frequency coefficients which are small or that are not humanly perceptible. Thus, the goal is to produce a basis both localized in space and frequency, because a vector expansion's in this basis will make available both spatial and frequency information. Wavelets will be such a basis.

2.3.1 Wavelets on $\mathbb{Z}_N$

Another feature we will want for our basis is a fast change between the standard one and ours. In order to find this fast algorithm, since we can use the FFT to compute convolutions, we are going to relate the convolution with the inner product to use the FFT for the change of basis.

Definition 2.3.2. Given a vector $z \in \ell^2(\mathbb{Z}_N)$. We define $\tilde{z} \in \ell^2(\mathbb{Z}_N)$ by

$$\tilde{z}(n) = \overline{z(-n)} = \overline{z(N-n)}, \qquad \text{for all } n.$$

We call $\tilde{z}$ the conjugate reflection of z.

Definition 2.3.3. Let N be an even integer, $N = 2M$ for some $M \in \mathbb{N}$. An orthonormal basis for $\ell^2(\mathbb{Z}_N)$ of the form

$$\{R_{2k}u\}_{k=0}^{M-1} \cup \{R_{2k}v\}_{k=0}^{M-1},$$

for some $u, v \in \ell^2(\mathbb{Z}_N)$, is called a first-stage wavelet basis for $\ell^2(\mathbb{Z}_N)$. We call u and v the generators of this basis.

Lemma 2.3.4. *Let $u, v \in \ell^2(Z_N)$. For any $k \in Z$,*

$$u * \tilde{v}(k) = \langle u, R_k v \rangle$$

and

$$u * v(k) = \langle u, R_k \tilde{v} \rangle.$$

Proof. Applying the definition of inner product

$$\langle u, R_k v \rangle = \sum_{n=0}^{N-1} u(n)\overline{v(n-k)} = \sum_{n=0}^{N-1} u(n)\tilde{v}(k-n) = u * \tilde{v}(k).$$

For the second part we only have to apply the same proof changing v by $\tilde{v}$ and noting that $v = \tilde{\tilde{v}}$. □

Lemma 2.3.5. *Let $u \in \ell^2(Z_N)$. Then $\{R_k u\}_{k=0}^{N-1}$ is an orthonormal basis for $\ell^2(Z_N)$ if and only if $|\hat{u}(n)| = 1$ for all $n \in \mathbb{Z}_N$.*

Proof. Using the Dirac $\delta \in \ell^2(\mathbb{Z}_N)$ function

$$\delta(n) = \begin{cases} 1, & \text{if } n = 0, \\ 0, & \text{if } n = 1, \dots, N-1, \end{cases}$$

we know that $\hat{\delta}(n) = 1$, for all n. We also know that $\{R_k u\}_{k=0}^{N-1}$ is an orthonormal basis for $\ell^2(Z_N)$ if and only if $\langle u, R_k u \rangle = \delta(k)$. We have in Lemma 2.3.4 that

$$\langle u, R_k u \rangle = u * \tilde{u}(k).$$

Hence,

$$u * \tilde{u} = \delta.$$

By (2.2) and the fact that $\widehat{(\tilde{u})}(n) = \overline{\hat{u}(n)}$, we have that

$$1 = \hat{\delta}(n) = \widehat{(u * \tilde{u})}(n) = \hat{u}(n)\widehat{(\tilde{u})}(n) = \hat{u}(n)\overline{\hat{u}(n)} = |\hat{u}(n)|^2,$$

for all n. □

This lemma states that we cannot have a frequency localized orthonormal basis of the form $\{R_k u\}_{k=0}^{N-1}$, because $\hat{u}(n)$ will have magnitude 1, for all n and, since $|\widehat{(R_k u)}(n)| = |\hat{u}(n)|$, every element of the basis will share the same property.

Lemma 2.3.6. *Let $M \in \mathbb{N}$, $N = 2M$ and $z \in \ell^2(Z_N)$. Define $z^* \in \ell^2(\mathbb{Z}_N)$ by*

$$z^*(n) = (-1)^n z(n), \quad \text{for all } n.$$

Then,

$$\widehat{(z^*)}(k) = \hat{z}(k + M), \quad \text{for all } k.$$

Proof.

$$\begin{aligned}\widehat{(z^*)}(k) &= \sum_{n=0}^{N-1} (-1)^n z(n) e^{-i2\pi nk/N} = \sum_{n=0}^{N-1} z(n) e^{-i\pi n} e^{-i2\pi nk/N} \\ &= \sum_{n=0}^{N-1} z(n) e^{-i2\pi n(k+M)/N} = \hat{z}(k + M).\end{aligned}$$

□

Observe that if $z \in \ell^2(\mathbb{Z}_N)$, N even, then

$$(z + z^*)(n) = \begin{cases} 2z(n), & \text{if } n \text{ is even,} \\ 0, & \text{if } n \text{ is odd.} \end{cases}$$

Lemma 2.3.7. *Let $M \in \mathbb{N}$, $N = 2M$ and $v \in \ell^2(Z_N)$. Then $\{R_{2k} v\}_{k=0}^{M-1}$ is an orthonormal set with M elements if and only if*

$$|\hat{v}(n)|^2 + |\hat{v}(n + M)|^2 = 2, \quad \text{for } n = 0, \ldots, M - 1.$$

Proof. From Lemma 2.3.4 we already know that $\{R_{2k} v\}_{k=0}^{M-1}$ is an orthonormal set with M elements if and only if

$$v * \tilde{v}(2k) = \begin{cases} 1, & \text{if } k = 0, \\ 0, & \text{if } k = 1, 2, \ldots, M - 1, \end{cases} \tag{2.8}$$

and,

$$(v * \tilde{v} + (v * \tilde{v})^*)(n) = \begin{cases} 2(v * \tilde{v}), & \text{if } n \text{ is even,} \\ 0, & \text{if } n \text{ is odd.} \end{cases}$$

Hence, for n even, say $n = 2k$, equation (2.8) holds if and only if

$$(v * \tilde{v} + (v * \tilde{v})^*)(n) = 2(v * \tilde{v})(2k) = \begin{cases} 2, & \text{if } k = 0, \\ 0, & \text{if } k = 1, \ldots, M-1. \end{cases}$$

For odd values of n, $(v * \tilde{v} + (v * \tilde{v})^*)(n) = 0$. Therefore, equation (2.8) holds if and only if

$$(v * \tilde{v} + (v * \tilde{v})^*) = 2\delta.$$

Applying (2.2) and the fact that $\hat{\delta}(n) = 1$, for all n, we get that equation (2.8) holds if and only if

$$(\widehat{(v * \tilde{v})^*})(n) + \widehat{(v * \tilde{v})}(n) = 2, \quad \text{for all } n = 0, \ldots, N-1.$$

We also know that

$$\widehat{(v * \tilde{v})}(n) = \hat{v}(n)\widehat{(\tilde{v})}(n) = \hat{v}(n)\overline{\hat{v}(n)} = |\hat{v}(n)|^2,$$

which implies that

$$(\widehat{(v * \tilde{v})^*})(n) = (\widehat{(v * \tilde{v})})(n + M) = |\hat{v}(n + M)|^2.$$

Hence,

$$(\widehat{(v * \tilde{v})^*})(n) + \widehat{(v * \tilde{v})}(n) = |\hat{v}(n)|^2 + |\hat{v}(n + M)|^2 = 2, \quad \text{for all } n = 0, \ldots, N-1,$$

because $|\hat{v}(n)|^2 + |\hat{v}(n + M)|^2 = |\hat{v}(n + M)|^2 + |\hat{v}(n + M + M)|^2$, since $\hat{v}$ has period $N = 2M$. □

Definition 2.3.8. Let N be an even integer, $N = 2M$ for some $M \in \mathbb{N}$, and $u, v \in \ell^2(\mathbb{Z}_N)$. For $n \in \mathbb{N}$ we define $A(n)$, the system matrix of u and v, by

$$A(n) = \frac{1}{\sqrt{2}} \begin{pmatrix} \hat{u}(n) & \hat{v}(n) \\ \hat{u}(n + M) & \hat{v}(n + M) \end{pmatrix}. \tag{2.9}$$

Theorem 2.3.9. *Let N be an even integer, $N = 2M$ for some $M \in \mathbb{N}$, and $u, v \in \ell^2(\mathbb{Z}_N)$. Then*

$$B = \{R_{2k}u\}_{k=0}^{M-1} \cup \{R_{2k}v\}_{k=0}^{M-1}$$

is an orthonormal basis for $\ell^2(\mathbb{Z}_N)$ if and only if the system matrix $A(n)$ of u and v is unitary for each $n = 0, \ldots, M-1$. Equivalently, B is a first-stage wavelet basis for $\ell^2(\mathbb{Z}_N)$ if and only if

$$|\hat{u}(n)|^2 + |\hat{u}(n + M)|^2 = 2,$$

$$|\hat{v}(n)|^2 + |\hat{v}(n+M)|^2 = 2,$$

and

$$\hat{u}(n)\overline{\hat{v}(n)} + \hat{u}(n+M)\overline{\hat{v}(n+M)} = 0,$$

for all $n = 0, \dots, M-1$.

Proof. Since a 2×2 matrix is unitary if and only if its columns are orthonormal in $\mathbb{C}^2$, we get that $\{R_{2k}u\}_{k=0}^{M-1}$ is orthonormal if and only if $|\hat{u}(n)|^2 + |\hat{u}(n+M)|^2 = 2$, for $n = 0, \dots, M-1$. Similarly, $\{R_{2k}v\}_{k=0}^{M-1}$ is orthonormal if and only if $|\hat{v}(n)|^2 + |\hat{v}(n+M)|^2 = 2$, for $n = 0, \dots, M-1$. Finally, let us prove that

$$\langle R_{2k}u, R_{2j}v \rangle = 0, \qquad \text{for all } j, k = 0, \dots, M-1, \tag{2.10}$$

if and only if

$$\hat{u}(n)\overline{\hat{v}(n)} + \hat{u}(n+M)\overline{\hat{v}(n+M)} = 0,$$

for all $n = 0, \dots, M-1$. If this claim were true, then the columns of $A(n)$ would be orthogonal and B would be an orthogonal set and, hence, it would be a basis for $\ell^2(\mathbb{Z}_N)$. Since (2.10) is equivalent to

$$u * \tilde{v}(2k) = \langle u, R_{2k}v \rangle = 0, \qquad \text{for all } k = 0, \dots, M-1,$$

then it is also equivalent to

$$u * \tilde{v} + (u * \tilde{v})^* = 0,$$

because the values at odd indices are automatically zero. If we apply the DFT to this equation, then

$$\widehat{(u * \tilde{v})} + (\widehat{(u * \tilde{v})^*}) = 0,$$

and by (2.3),

$$\widehat{(u * \tilde{v})}(n) = \hat{u}(n)\overline{\hat{v}(n)}.$$

By Lemma 2.3.6 we have that

$$(\widehat{(u * \tilde{v})^*})(n) = \hat{u}(n+M)\overline{\hat{v}(n+M)},$$

and hence

$$\hat{u}(n)\overline{\hat{v}(n)} + \hat{u}(n+M)\overline{\hat{v}(n+M)} = \widehat{(u * \tilde{v})}(n) + (\widehat{(u * \tilde{v})^*})(n) = 0.$$

□

2.3.2 Examples

We are going to discuss three examples of wavelets, starting by the simplest one, the Haar wavelet, we will continue with an intermediate one, the Shannon wavelet and finally we will present the Daubechies wavelet, used in the JPEG2000 compression algorithm.

Example 2.3.10. The Haar wavelet transform.

Let us motivate this example in a different way. Suppose we want to transmit a data vector of size N (even) and we want to dispatch only $N/2$ numbers. One way to do it would be to send pairwise averages of the numbers. That is, if $z \in \mathbb{Z}_N$, N even, we can write $z = \{z_0, \ldots, z_{N-1}\}$, then we send a vector $a \in \mathbb{Z}_{N/2}$ such that

$$a_k = \frac{z_{2k} + z_{2k+1}}{2}, \quad \text{for all } k = 0, \ldots, N/2 - 1.$$

It is clear that it is impossible to recover the original information from the averaged transmitted set, but now, suppose that we also send another vector d of length $N/2$ containing the averaged pairwise differences, i.e.

$$d_k = \frac{z_{2k} - z_{2k+1}}{2}, \quad \text{for all } k = 0, \ldots, N/2 - 1,$$

then the receiver could recover the original information. The advantage of using this method is that, if the data vector is largely homogeneous, then a lot of elements of the vector d will be small and could be quantized to zero in order to compress the data with a small loss of quality.

The technique explained before is what the Haar wavelet is based on. Let $u, v \in \mathbb{Z}_N$, N even, such that

$$u = \left\{ \frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, 0, \ldots, 0 \right\} \quad \text{and} \quad v = \left\{ \frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}}, 0, \ldots, 0 \right\},$$

then the Haar wavelet basis is given by $B_H = \{R_{2k}u\}_{k=0}^{M} \cup \{R_{2k}v\}_{k=0}^{M}$, where $M = N/2$. Hence, if $z \in \ell^2(\mathbb{Z}_N)$, its Haar transform will be given by

$$a^1 = (\langle R_0 u, z\rangle, \ldots, \langle R_{N-2}u, z\rangle),$$
$$d^1 = (\langle R_0 v, z\rangle, \ldots, \langle R_{N-2}v, z\rangle),$$

where the vector a^1 (tendency) keeps most of the information contained in the original vector and the vector d^1 (details) gives the details.

This transformation can be written in matrix form

$$W_N = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & & 0 & 0 \\ & & & \vdots & & & \\ 0 & 0 & 0 & 0 & \dots & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & & 0 & 0 \\ & & & \vdots & & & \\ 0 & 0 & 0 & 0 & \dots & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix},$$

and then, the Haar transform of a vector z can be expressed by

$$\text{Haar}(z) = W_N z.$$

Observe that the rows of the matrix W_N correspond to the vectors u and v shifted, and hence the product $W_N z$ can be implemented as convolutions, which will be constructed via the FFT algorithm.

Since the matrix W_N is unitary, we can recover the original vector applying the inverse transform just by multiplying by the transposed matrix W_N^T

$$z = W_N^T \,\text{Haar}(z).$$

We can give an interpretation to the vectors u and v if we compute their Fourier transforms [3],

$$\hat{u}(n) = \frac{1}{\sqrt{2}}(1 + e^{-i2\pi n}) \quad \text{and} \quad \hat{v}(n) = \frac{1}{\sqrt{2}}(1 - e^{-i2\pi n}).$$

Figure 2.5 shows the absolute value of their DFT for $N = 256$. We can see how $|\hat{u}|$ attenuates the high frequencies, corresponding to the values closed to $N/2 = 128$. On the other hand, $|\hat{v}|$ does the contrary, it allows high-frequency signals to pass, but attenuates the amplitudes of low-frequency data. Hence, u is a lowpass filter and v is a highpass filter. In general, every wavelet transform has a vector u who acts as a lowpass filter and a vector v who behaves as a highpass filter.

If $N = 2^p$, $p \in \mathbb{N}$, we can iterate the process obtaining the kth-Haar transform of z applying the transformation to the tendency vector that we obtained in the previous operation and keeping the details. Hence, in the 2nd-transform we will obtain (a^2, d^2, d^1) and in the kth-transform we will have $(a^k, d^k, \dots, d^1)$.

Figure 2.6 shows a time signal $z \in Z_{256}$ (above) and its Haar transform (middle), the first 256 coefficients correspond to the tendency and the next 256 correspond to the details. A quantization has been applied to the transform keeping the 40% larger coefficients and after implementing the inverse transform we recover an approximation to the original signal (below).

To perform a Haar transform on images, i.e. a 2D-Haar transform, we apply the transform to each row of the image and after that to each column.

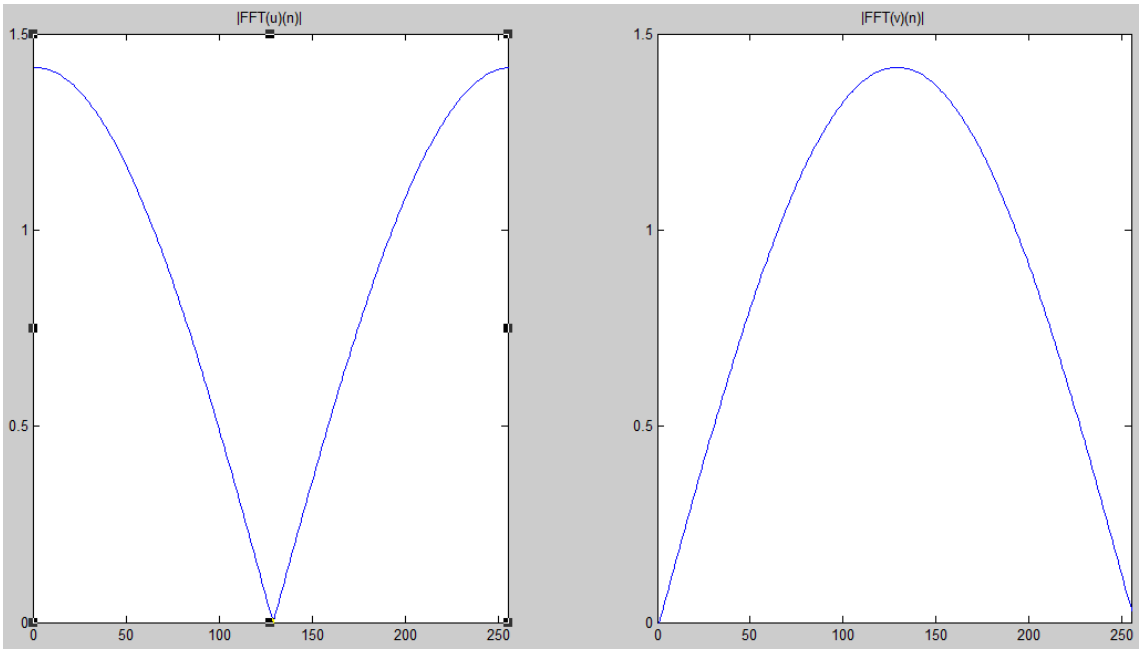

Figure 2.5: Lowpass filter $|\hat{u}(n)|$ (left) and highpass filter $|\hat{v}(n)|$ of the Haar wavelet for $N = 256$.

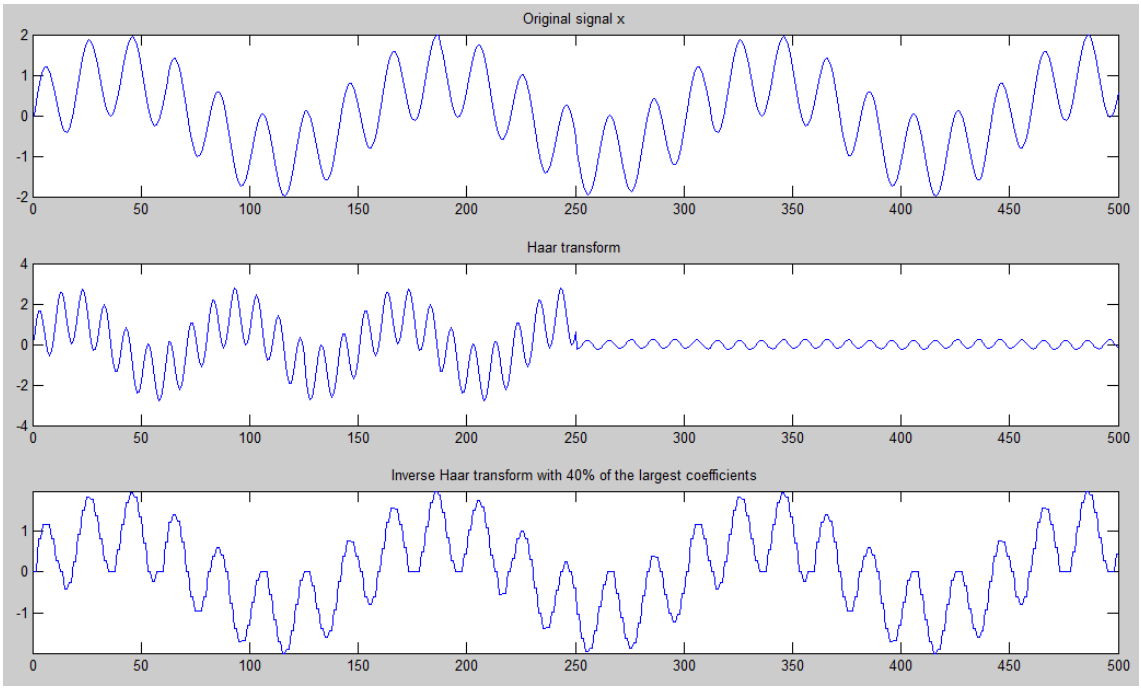

Figure 2.6: Original signal (above), Haar transform (middle) and recovered signal keeping 40% of the largest coefficients (below).

Figure 2.7: Image of Lenna (left) and its Haar transform (right).

Figure 2.7 shows the image of Lenna (right) with its Haar transform (left). We can see that the Haar transform is composed by four different blocks, the first block (left upper corner) shows the details, second block (right upper corner) contains the averaged differences in horizontal direction, the third block (left down corner) represents the averaged differences in vertical direction and, finally, the fourth block contains the averaged differences in diagonal direction.

In Figure 2.8 we show the 2nd Haar transform on Lenna's image (right).

Example 2.3.11. The real Shannon wavelet transform.

If N is divisible by 4 we define the real Shannon basis by

$$\hat{u}(n) = \begin{cases} \sqrt{\frac{2}{N}}, & \text{if } n = 0, 1, \ldots, \frac{N}{4} - 1 \text{ or } n = \frac{3N}{4} + 1, \ldots, N - 1, \\ \frac{i}{\sqrt{N}}, & \text{if } n = \frac{N}{4}, \\ \frac{-i}{\sqrt{N}}, & \text{if } n = \frac{3N}{4}, \\ 0, & \text{if } n = \frac{N}{4} + 1, \ldots, \frac{3N}{4} - 1, \end{cases}$$

and

$$\hat{v}(n) = \begin{cases} 0, & \text{if } n = 0, 1, \ldots, \frac{N}{4} - 1 \text{ or } n = \frac{3N}{4} + 1, \ldots, N - 1, \\ \frac{1}{\sqrt{N}}, & \text{if } n = \frac{N}{4}, \frac{3N}{4}, \\ \sqrt{\frac{2}{N}}, & \text{if } n = \frac{N}{4} + 1, \ldots, \frac{3N}{4} - 1. \end{cases}$$

Observe that we have defined the basis in the frequency domain and, since $\hat{u}$ and $\hat{v}$ are symmetric, u and v are real-valued vectors. To compute the values of these vectors we should use the IFFT algorithm. Figure 2.9 shows these vectors for $N = 256$, observe that they are relatively well localized around their center points, $N/2$. It is easy to check that, with this definition, the vectors u and v satisfy the conditions of Theorem 2.3.9 to construct a first-stage wavelet basis.

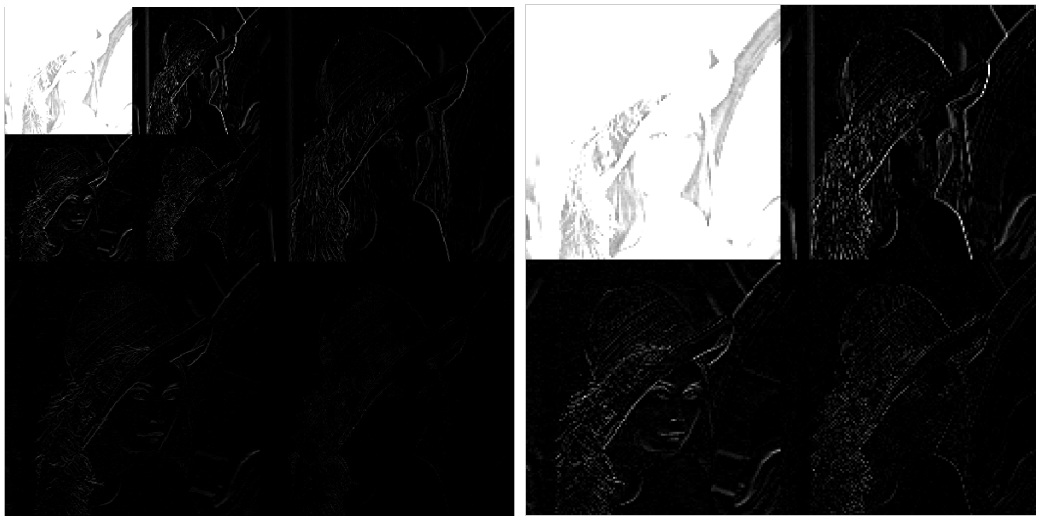

Figure 2.8: Second Haar iterate on Lenna (left) and its zoom (right).

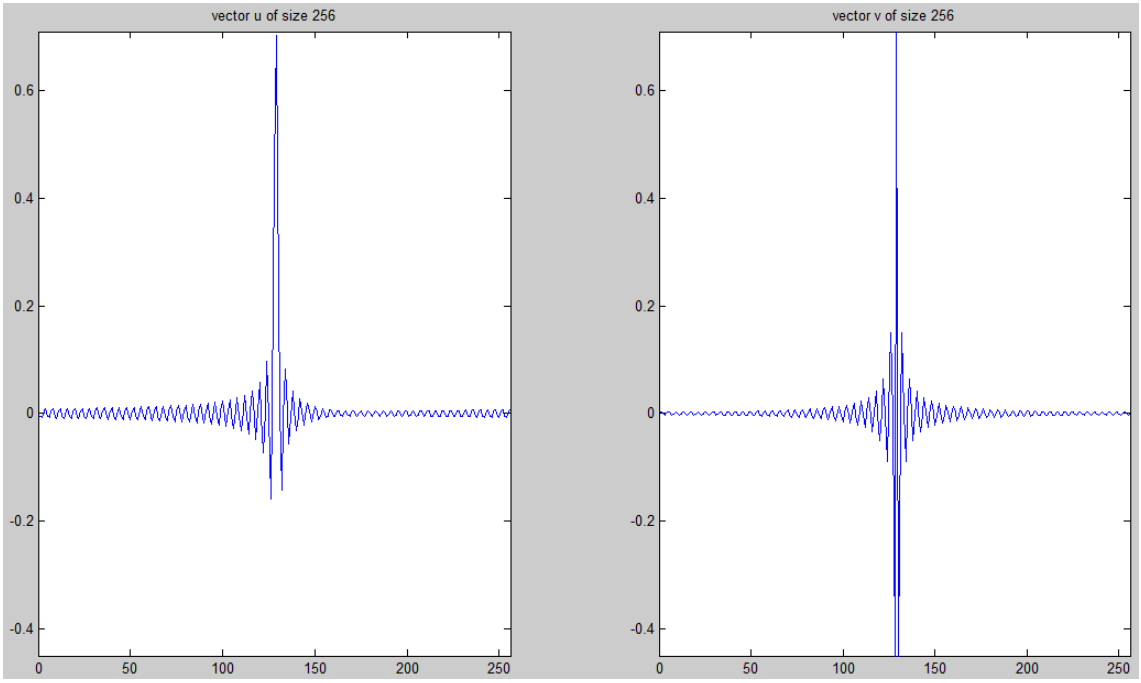

Figure 2.9: Vectors u (left) and v (right) of the real Shannon basis for $N = 256$.

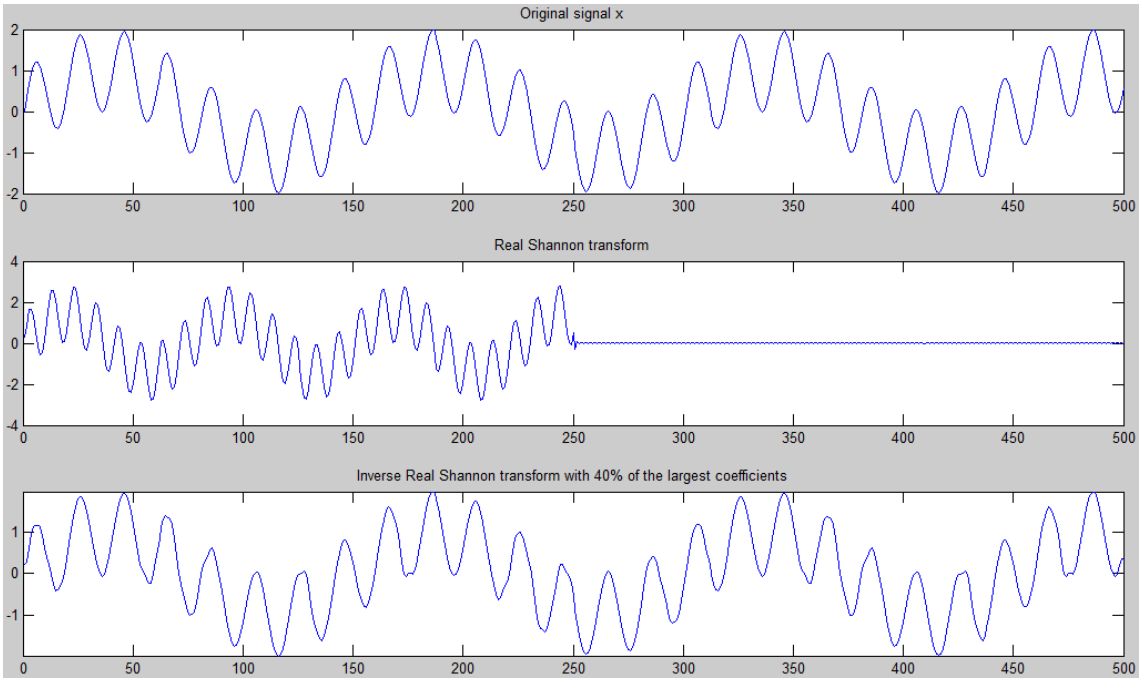

Figure 2.10: Original signal (above), Shannon transform (middle) and recovered signal keeping 40% of the largest coefficients (below).

As in the previous example, we show in Figure 2.10 how the Shannon wavelet transform operates on a time signal of size $N = 512$. Again we see the original signal (above), its Shannon transform (middle) and the approximation to the original signal (compressed signal) after quantizing to zero the 60% of the smaller coefficients in the Shannon transform and applying the inverse Shannon transform.

Finally, we apply this transform to the image of Lenna. The results are shown in Figure 2.11 for the first iteration and in Figure 2.12 for the second iteration.

Example 2.3.12. Daubechies' D6 wavelet transform.

The real Shannon wavelets have well localized DFTs by definition. Ingrid Daubechies obtained families of wavelets very well localized in space rather than in frequency, in $\mathbb{Z}$ and $\mathbb{R}$, but they can be adapted to $\mathbb{Z}_N$. Let N be divisible by 2^p, for some $p \in \mathbb{N}$, and $N/2^p > 6$. The Daubechies' D6 wavelet is generated by the vectors

$$\begin{aligned} u &= \{u(0), u(1), u(2), u(3), u(4), u(5), 0, \ldots, 0\} \\ &= \frac{\sqrt{2}}{32}\{b + c, 2a + 3b + 3c, 6a + 4b + 2c, 6a + 4b - 2c, 2a + 3b - 3c, b - c, 0, \ldots, 0\}, \end{aligned}$$

where

$$a = 1 - \sqrt{10}, \quad b = 1 + \sqrt{10}, \quad \textit{and} \quad c = \sqrt{5 + 2\sqrt{10}}.$$

Using this vector we can construct the vector v applying Lemma 2.3.13.

Figure 2.11: Image of Lenna (left) and its Shannon transform (right).

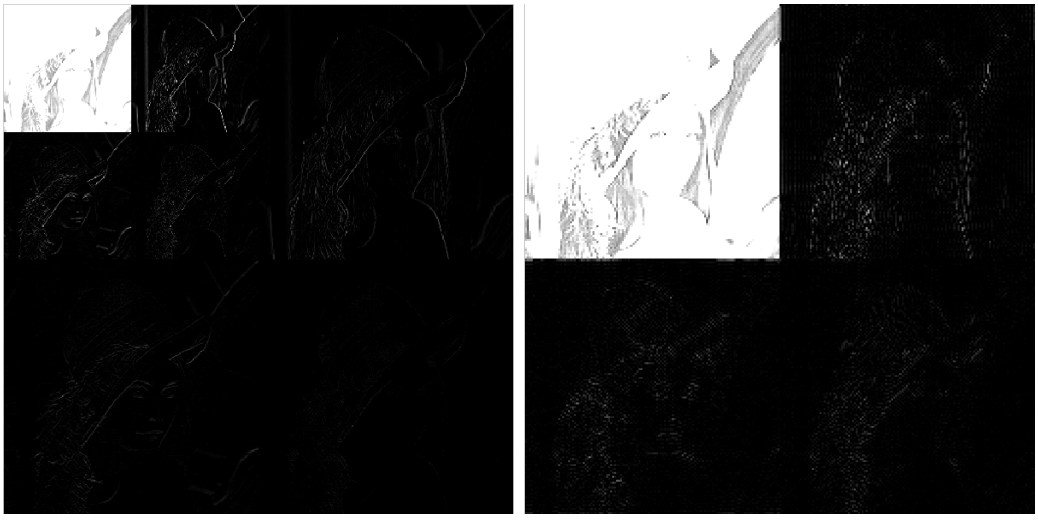

Figure 2.12: Second Shannon iterate on Lenna (left) and its zoom (right).

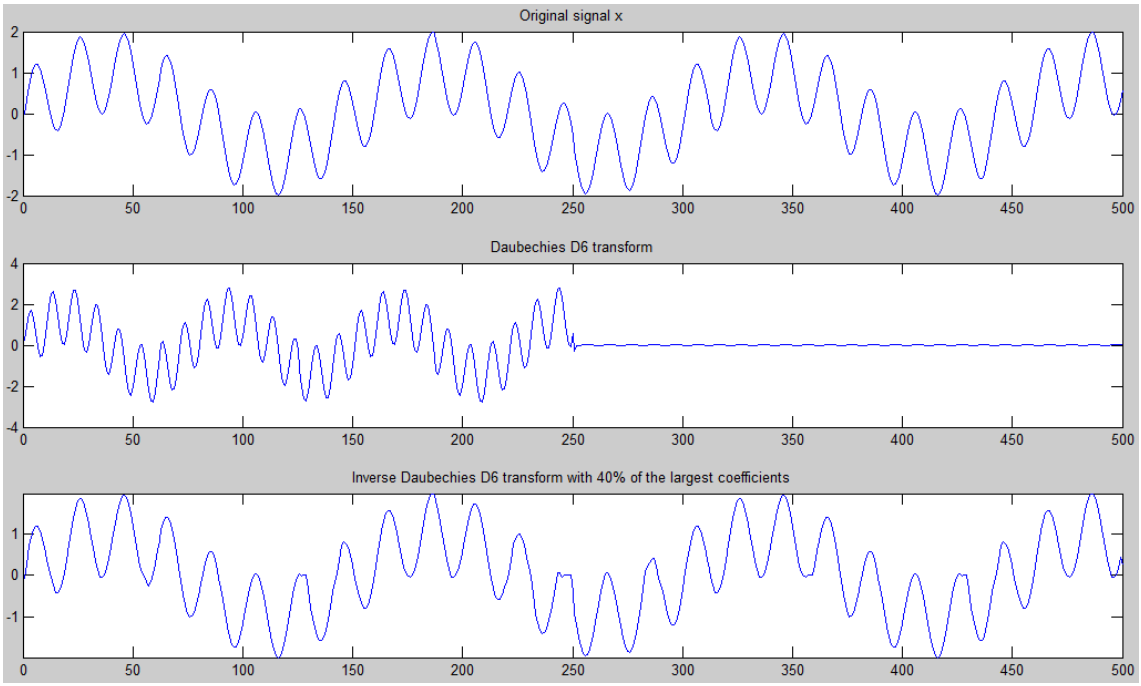

Figure 2.13: Original signal (above), Daubechies' D6 transform (middle) and recovered signal keeping the 40% of the largest coefficients (below).

Lemma 2.3.13. *Let $M \in \mathbb{N}$, $N = 2M$ and $u \in \ell^2(\mathbb{Z}_N)$ such that $\{R_{2u}u\}_{k=0}^{M-1}$ is an orthogonal set with M elements. Define $v \in \ell^2(\mathbb{Z}_N)$ by*

$$v(k) = (-1)^{k-1}\overline{u(1-k)}, \quad \text{for all } k.$$

Then, $\{R_{2u}u\}_{k=0}^{M-1} \cup \{R_{2v}v\}_{k=0}^{M-1}$ is a first-stage wavelet basis for $\ell^2(\mathbb{Z}_N)$.

Applying Lemma 2.3.13 we obtain

$$v = (-u(1), u(0), 0, \ldots, 0, -u(5), u(4), -u(3), u(2)).$$

Observe that the vectors u and v have only 6 non-zero elements. In Figure 2.13 we show again the effect of the Daubechies' D6 transform on a signal in one dimension and the recovered signal after apply the same quantization of the previous examples.

Finally, we apply this transform to the image of Lenna. The results are shown in Figure 2.14 for the first iteration and in Figure 2.15 for the second iteration.

In order to compare the properties of localization of the presented wavelets Figure 2.16 shows the histograms of the matrices W_N and $\widehat{W_N}$, for $N = 64$, for each basis: standard, Haar, Shannon, and Daubechies' D6. In each histogram we represent, in the x-axis, the value of the signal coefficients in a given basis and, in the y-axis, the number of coefficients that have each of these values.

We can observe that the standard basis has a very good localization in space, all coefficients are zero except one, but it is very bad localized in frequency, since

Figure 2.14: Image of Lenna (left) and its Daubechies' D6 transform (right).

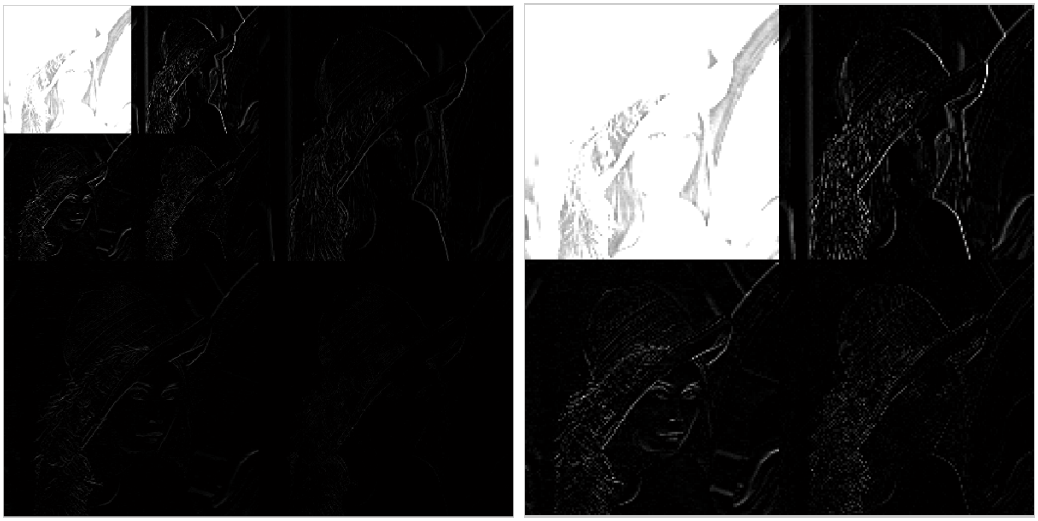

Figure 2.15: Second Daubechies' wavelet D6 iterate on Lenna (left) and its zoom (right).

Figure 2.16: Histograms for the transformation matrices and their DFTs. Standard (first), Haar (second), Shannon (third), and Daubechies' D6 (fourth).

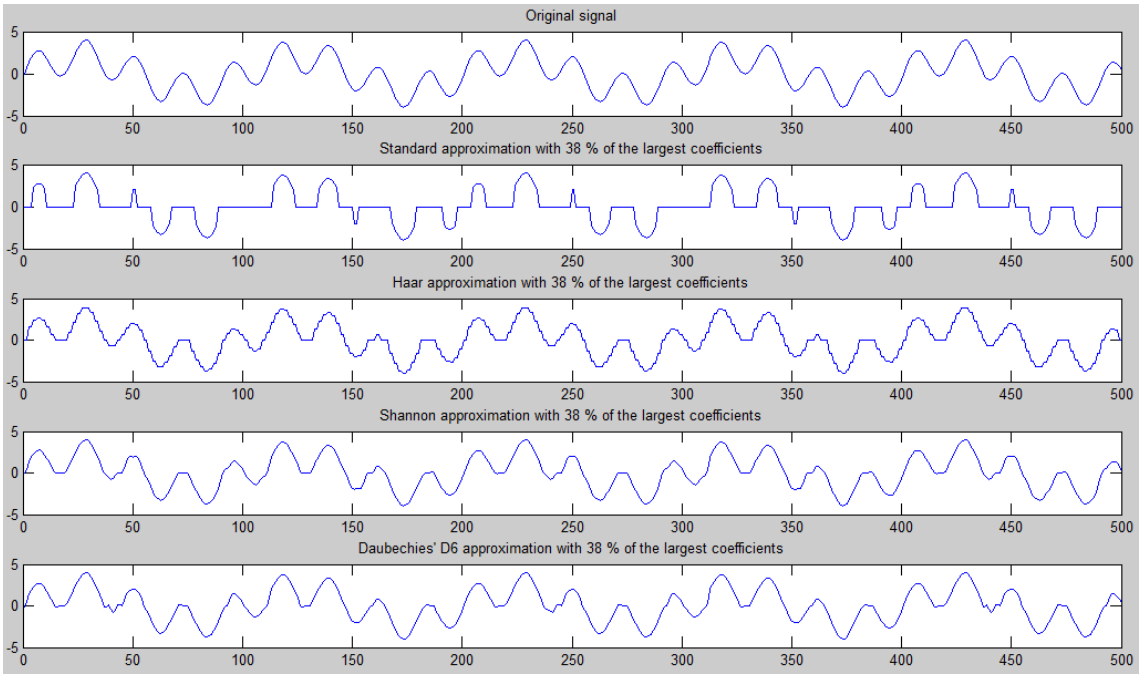

Figure 2.17: Original signal z_1 (above), approximations keeping 38% of the largest coefficients in standard, Haar, Shannon, and Daubechies' D6 basis.

all coefficients have value $1/N = 0.125$. The Haar basis is also well localized in space, but is not very well localized in frequency, although it is better localized than the standard one. Finally, the Shannon and the Daubechies basis are well localized in space and have a lot of zero coefficients in frequency, both have relatively good localization in frequency, but we see that the D6 wavelets are much more sharply localized in space than the real Shannon wavelets. On the other hand, the D6 wavelets are not as precisely localized in frequency.

Finally, Figures 2.17, 2.18, and 2.19 show three original signals,

$$z_1(n) = 2\sin(0.2\pi n) + 2\sin(0.9\pi n), \quad z_2(n) = \sin(10\pi(n^{1.5})/64),$$

and

$$z_3(n) = \begin{cases} 0, & \text{if } 0 \le n \le 127, \\ \sin(|n-128|^{1.7}/128), & \text{if } 128 \le n \le 255, \\ 0, & \text{if } 256 \le n \le 383, \\ \sin(|n-128|^{2}/128), & \text{if } 384 \le n \le 447, \\ 0, & \text{if } 448 \le n \le 511, \end{cases}$$

respectively, and the approximations obtained using the standard, the Haar, the Shannon and the Daubechies' D6 wavelets when we compress the signal keeping the 38% of the largest coefficients in each transformation.

For all these signals we can appreciate that the standard basis behaves very bad when we quantize the lower coefficients; the Haar approximation is neither very good,

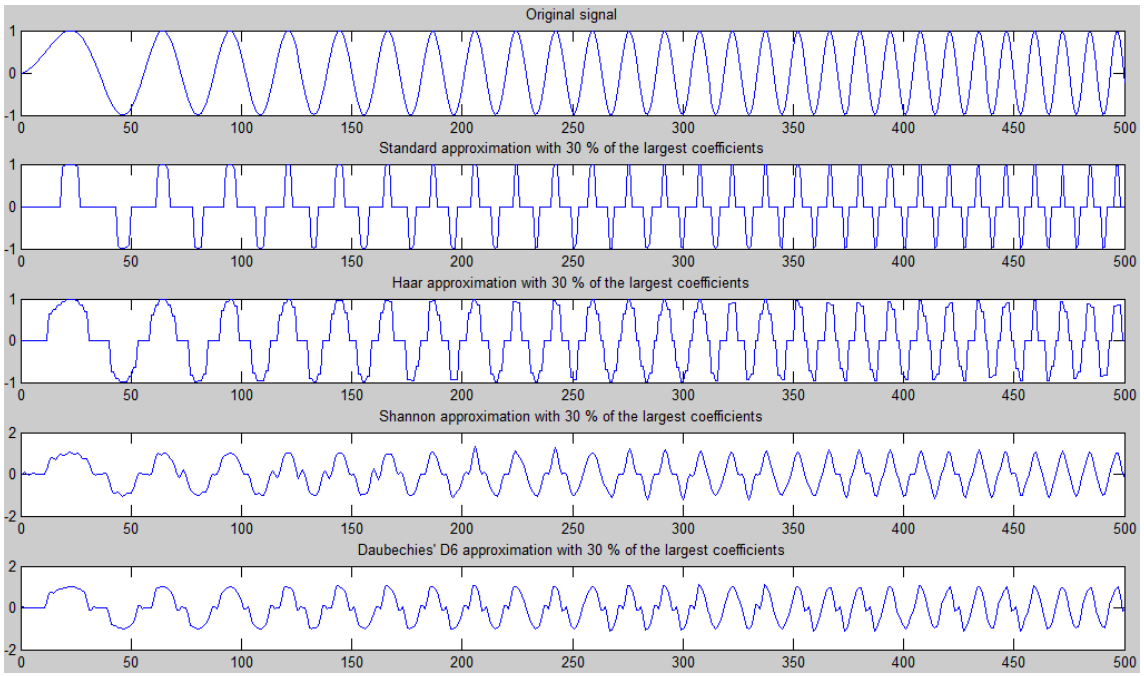

Figure 2.18: Original signal z_2 (above), approximations keeping 38% of the largest coefficients in standard, Haar, Shannon, and Daubechies' D6 basis.

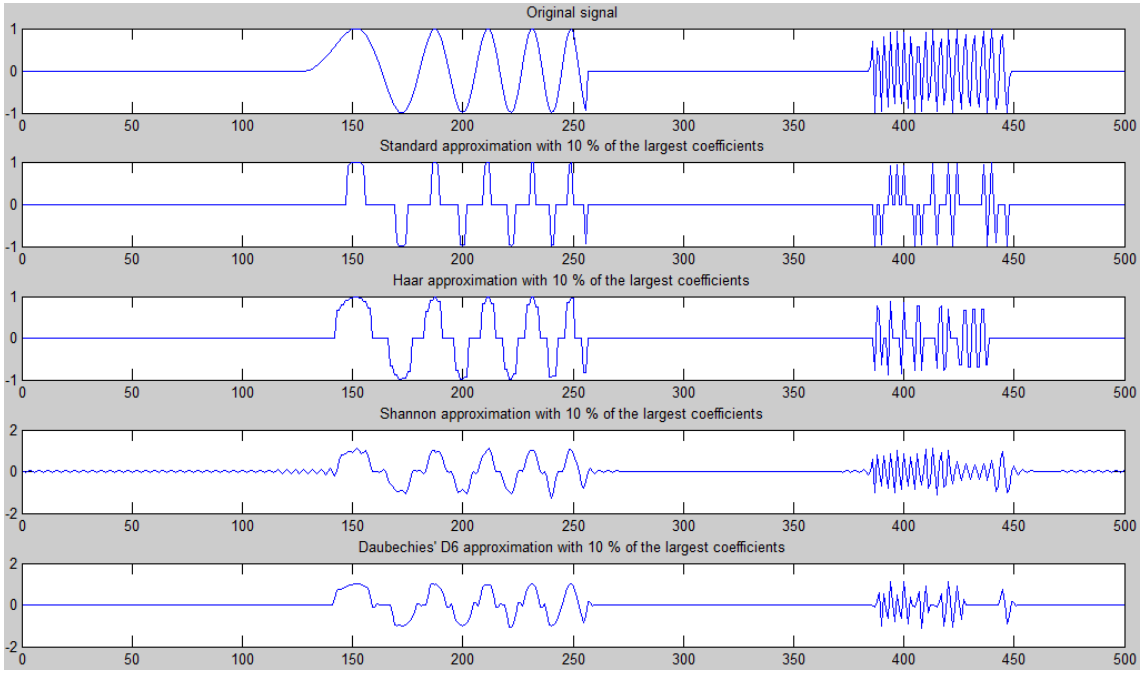

Figure 2.19: Original signal z_3 (above), approximations keeping 38% of the largest coefficients in standard, Haar, Shannon, and Daubechies' D6 basis.

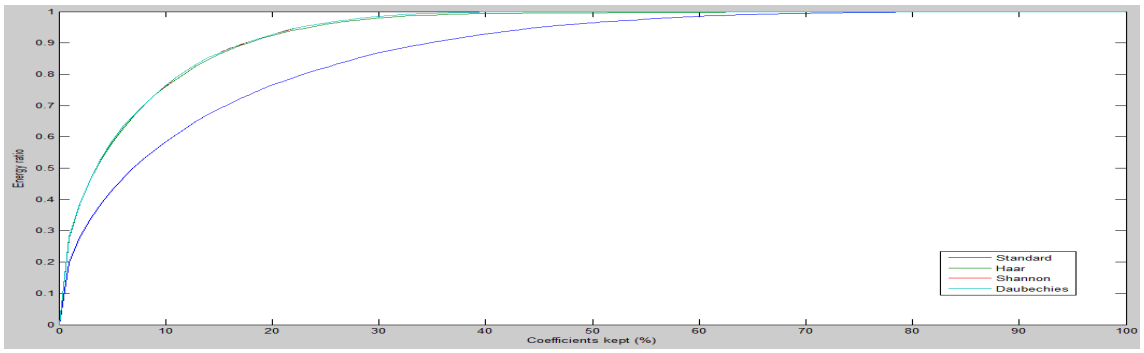

Figure 2.20: Energy ratio for signal 1 as a function of the kept coefficients in the approximation.

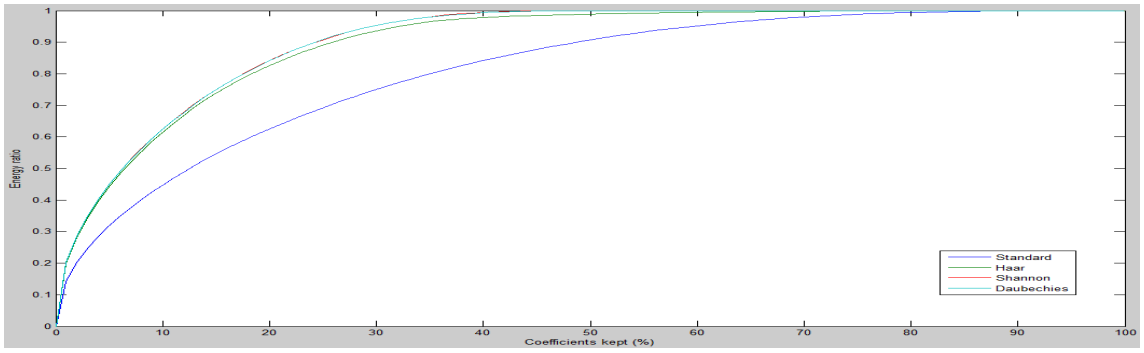

Figure 2.21: Energy ratio for signal 2 as a function of the kept coefficients in the approximation.

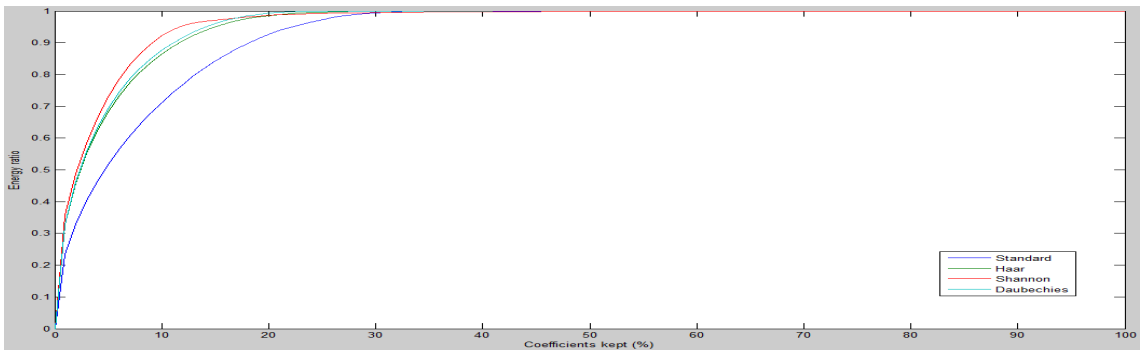

Figure 2.22: Energy ratio for signal 3 as a function of the kept coefficients in the approximation.

but the Shannon and Daubechies wavelet provide an approximation to the original signal that seems pretty similar to it. To confirm this point, for each one of the three original signals, Figures 2.20, 2.21, and 2.22 show the energy ratio (quotient between the ℓ^2-norms defined in (2.1)) between the approximated (compressed) signal and the original one, for each basis, as a function of the kept coefficients in the compressed signal. One can confirm that the Shannon and Daubechies wavelets maintain a better energy relation than the Haar wavelet, but it is not clear which one is the best. For signal 2 both wavelets seem to behave similar with respect to the energy ratio, but for signal 3 it looks like the Shannon approximation has a better energy relation if we keep less than 15% of the coefficients, but from 15% coefficients kept on, Daubechies' D6 approximation beats Shannon's one.

Chapter 3

The JPEG standard image format

3.1 Introduction

An image is a collection of $N \times M$ pixels, where N is the width and M is the height of the image. Each pixel is represented by a number between 0 and 255 (gray scale), if the image is black and white, or by three numbers R, G, and B between 0 and 255 indicating the intensity of colors red, green, and blue of the pixel, respectively. For example, Figure 3.1 shows the color version of the image of Lenna (left) and the black an white version (right). These images have 512×512 pixels and, since each pixel is encoded with 1 byte (8 bits), we may expect that the black and white version of Lenna has a size of 262144 bytes, but surprisingly, the size of the image in the computer is only 32768 bytes. The reason for this discrepancy comes from the use of a compression algorithm to store the image named JPEG, developed by the Joint Photographic Experts Group (JPEG) [5] and established as an international standard in 1992. This format remains one of most popular for photographic-type images.

For color images, rather than work with the red, green, and blue components of a color, it is better to use three different quantities: luminance Y, which is closely related to the brightness of the color, and blue and red chrominances C_b and C_r, which determine the hue. The relation between this quantities is given by the following affine transformation [2]

$$\begin{pmatrix} Y \\ C_b \\ C_r \end{pmatrix} = \begin{pmatrix} 0.299 & 0.587 & 0.114 \\ -0.1687 & -0.3313 & 0.5 \\ 0.5 & -0.4187 & -0.0813 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix} + \begin{pmatrix} 0 \\ 128 \\ 128 \end{pmatrix}$$

and the inverse transformation

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 1.0 & 0.0 & 1.40210 \\ 1.0 & -0.34414 & -0.71414 \\ 1.0 & 1.77180 & 0.0 \end{pmatrix} \begin{pmatrix} Y \\ C_b - 128 \\ C_r - 128 \end{pmatrix}. \tag{3.1}$$

We can observe how the luminance Y contributes with the same weight to all the three color components red, green, and blue. Figure 3.2 shows the components RGB

Figure 3.1: Lenna in color JPEG format (left) and gray scale JPEG format (right).

Figure 3.2: Components RGB for Lenna's color image.

for the color image of Lenna. If we add the values of each pixel of the three images we will recover the original image of Figure 3.1 (left). In Figure 3.3 we can observe the values (Y, C_r, C_b) for Lenna's color image, where the lighter pixels correspond to larger values of the component. We notice how the luminance produces a gray scale version of the image.

In this chapter we are going to develop the mathematics describing the JPEG algorithm and we will implement the algorithm to use it in an example.

3.2 The JPEG algorithm

In the JPEG algorithm, the encoder is shown in Figure 3.4, the image is divided into blocks of 8×8 pixels in order to apply the DCT (Discrete Cosine Transform), since this transform is bad localized. Each block is processed independently from the others and, since the values of most blocks do not change rapidly, and the human

Figure 3.3: Luminance (left) and chrominances C_r (middle) and C_b (right) for Lenna's color image.

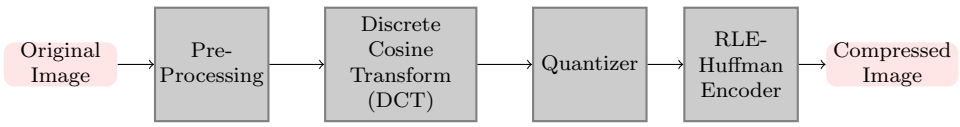

Figure 3.4: Diagram of the JPEG encoder.

eye is not particularly sensitive to these changes, the DCT components of the higher frequencies will be small and may be ignored without affecting our perception of the image. Therefore, the coefficients obtained by the DCT transform are quantized using an 8×8 matrix Q, in order to store them as integers. Moreover, to compress the data we use a parameter k that will determine the amount of compression and the quality of the image: larger values of k will lead to high compressed but lower quality images. If A is our 8×8 block matrix and G_A is its DCT, given by (2.7), then the quantized matrix $F = (F(m,n))$ is given by

$$F(m,n) = \frac{G_A(m,n)}{kQ(m,n)}.$$

If we work with a color image, we have to apply this process to each of the 8×8 block matrices obtained from the luminance and the chrominances C_r and C_b. Since the human eye is more sensitive to the luminance values, the quantize matrix Q_{lum} used for the luminance is not the same that the quantize matrix Q_{chr} used for the chrominances. These matrices come from empirically results obtained from

Figure 3.5: Lenna luminance (left) and 8×8 pixels block (right).

psycho-visual experiments and are given by [16]

$$Q_{\text{lum}} = \begin{pmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{pmatrix},$$

for the luminance quantization and

$$Q_{\text{chr}} = \begin{pmatrix} 17 & 18 & 24 & 47 & 99 & 99 & 99 & 99 \\ 18 & 21 & 26 & 66 & 99 & 99 & 99 & 99 \\ 24 & 26 & 56 & 99 & 99 & 99 & 99 & 99 \\ 47 & 66 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \end{pmatrix},$$

for the chrominances quantization. Let us see how it works with an example. Consider the 8×8 pixels block showed in Figure 3.5 (right), that corresponds to the white square in the luminance of Lenna's image in Figure 3.5 (left).

After applying the DCT to the block we get the result showed in Figure 3.6 (left) and when we quantize this DCT with $k = 1$, we can observe the result in Figure 3.6 (right).

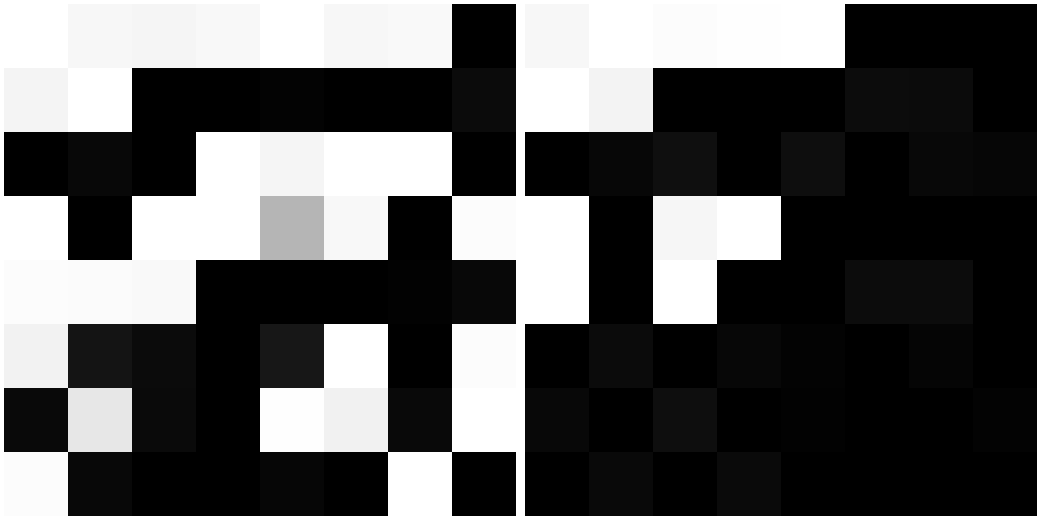

Figure 3.6: DCT of the 8×8 block (left) and quantized block with $k = 1$ (right).

Observe that in the quantized image the smaller coefficients correspond to dark colors and they are concentrated in the higher frequencies. The matrix after quantization obtained is

$$F = \begin{pmatrix} 68 & 13 & 2 & 1 & 1 & 0 & 0 & 0 \\ 23 & 6 & -4 & -1 & 0 & 0 & 0 & 0 \\ -2 & -6 & -5 & 0 & 0 & 0 & 0 & 0 \\ 1 & -2 & 2 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix},$$

where many coefficients have been turn into zero.

Continuing with the algorithm, the next step consists in ordering the coefficients in such a way that the low-frequency components appear first. The scheme to organize them is given in Figure 3.7 and the following matrix shows the new position in the order for each coefficient.

0	1	5	6	14	15	27	28
2	4	7	13	16	26	29	42
3	8	12	17	25	30	41	43
9	11	18	24	31	40	44	53
10	19	23	32	39	45	52	54
20	22	33	38	46	51	55	60
21	34	37	47	50	56	59	61
35	36	48	49	57	58	62	63

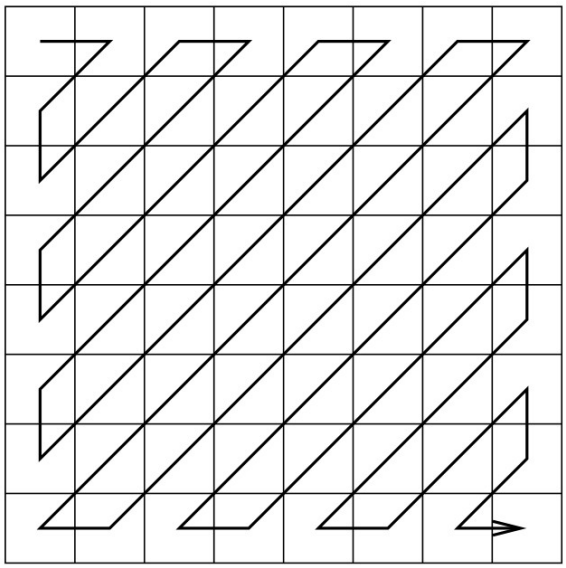

Figure 3.7: Zigzag ordering.

Therefore, in our example we will have the following vector of coefficients after ordering them

$$(68, 13, 23, -2, 6, 2, 1, -4, -6, 1, 0, -2, -5, -1, 1, 0, 0, 0, 2, 0, 0, 0, 0, 0, 2, \overbrace{0, \ldots, 0}^{39 \text{ zeroes}}).$$

The first element in the sequence of coefficients is called the DC component and the others are called the AC components. They are treated separately in the entropy coding process. Obviously, the last 39 zeroes will not be stored or transmitted, instead we will just record the number of them and it will reduce the storage requirement significantly. Now, we apply a zero run length encoding (RLE) to the 63 AC components which consists on saving, for each coefficient a_i different from zero, a pair of numbers (N_{a_i}, a_i), where N_{a_i} is the number of zeros preceding the coefficient a_i in the sequence. For instance, in our vector we will record $(0, 13)$, $(0, 23)$, $(0, -2)$, $(0, 6)$, $(0, 2)$, $(0, 1)$, $(0, -4)$, $(0, -6)$, $(0, 1)$, $(1, -2)$, $(0, -5)$, $(0, -1)$, $(0, 1)$, $(3, 2)$, $(5, 2)$, and $(0, 0)$, where the pair $(0, 0)$ indicates that the remainder coefficients are all zero. Now, in order to give a quantification of the compression for the system, we will suppose that each number in these pairs is encoded using eight bits and hence, if there are m pairs, we will need $2m$ bytes to encode them. We define the bit compression ratio.

Definition 3.2.1. Given a Run Length Encoding with m pairs of the previous type used to encode N pixels, the bit compression ratio is given by

$$C_b = \frac{N}{2m}.$$

Observe that we have encoded the number of zeroes preceding each coefficient using eight bits, but it is clear that the number of zeroes cannot be greater than 64, since there are only 64 coefficients in each 8×8 block. Hence, it suffices 6 bits to encode one of the numbers of each pair. In fact, it would be reasonable to suppose that there will not be more than 16 zeroes between coefficients different from zero and therefore we could use only 4 bits. Using this definition in our example and supposing that we also encode the DC component using RLE we would obtain a bit compression ratio for our 8×8 block equal to

$$C_b = \frac{64}{2 \cdot 17} = 1.88.$$

If we apply the JPEG algorithm to the same block with a compression level $k = 5$ we get the matrix

$$F_5 = \begin{pmatrix} 14 & 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 5 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix},$$

and the vector $(14, 3, 5, 0, 1, 0, 0, -1, -1, 0, 0, 0, -1, \overbrace{0, \ldots, 0}^{53 \text{ zeroes}})$, which produces a bit compression ratio of

$$C_b(k = 5) = \frac{64}{2 \cdot 8} = 4.$$

Increasing the compression with a factor $k = 10$ gives us the DCT quantized matrix

$$F_{10} = \begin{pmatrix} 7 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix},$$

obtaining a bit compression not much greater, $C_b(k = 10) = \frac{64}{2 \cdot 7} = 4.57$.

The DC components are stored considering the DC components of other blocks using differential encoding. We expect the DC components of neighboring blocks to be similar, and hence we go over the DC components of the blocks row-by-row, left-to-right manner, recording the difference with the previous DC's, except for the first one, which remains the same. Finally, in order to obtain further compression,

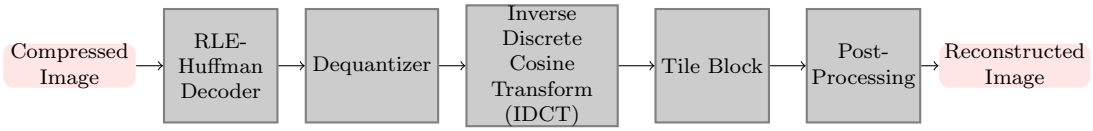

Figure 3.8: Diagram of the JPEG decoder.

a Huffman code (entropy code) [10, 16] is used to save the difference sequence of DC's coefficients.

The process to recover the image is implemented by the decoder shown in Figure 3.8 and goes through reversing the previous algorithm. From the compressed coefficients we recover the DCT blocks and apply an inverse discrete cosine transform to them in order to obtain the matrices (Y, C_r, C_b) of each block. From these matrices we apply the inverse transformation defined in (3.1) to get the RGB components. Finally, we only have to concatenate the resulting 8×8 pixels blocks to compose the reconstructed image.

Figure 3.9 shows the recovered images of Lenna after JPEG compression with compression levels $k = 1$ (above, left), $k = 2$ (above, right), $k = 10$ (below, left), and $k = 20$ (below, right). We can see how, for $k = 1$ the image almost remains the same. In Figure 3.10 we can see the quantized blocks of the image for the above values of the compression ratio k.

Now, we present the same example but with the color image of Lenna. Figure 3.11 shows the luminance, blue chrominance, and red chrominance recovered after compression using the ratio $k = 5$.

In Figure 3.12 we see the compressed color images of Lenna using the JPEG algorithm for compression ratios $k = 1$, 5, 10, and 20. We notice that for large compression ratios we can observe the appearance of block artifacts in the recovered image coming from the original 8×8 pixels blocks that we have extracted from the original image and processed independently. Since the information contained in one block is not used to process the others, when the compression is high discontinuities at the edges of the blocks become more apparent. Figure 3.13 shows the error (difference) between the original image and the recovered one for the previous compression ratios. We observe the negative of the difference images, i.e., dark pixels correspond to higher values of the difference. In Figure 3.14 we plot the mean square error between the original Lenna's image and the compressed one, using the JPEG algorithm, as a function of the compression ratio k.

The algorithms to compute the JPEG standard have been developed using MATLAB and can be found in Appendix A.

3.3 Entropy Code (Huffman)

We have seen that an image is an $N \times M$ matrix of pixels, each one codified with a byte (8 bits). Thus, another way to interpret an image is as a bit stream (sequence of

Figure 3.9: Recovered images after JPEG compression with ratios $k = 1, 5, 10, 20$.

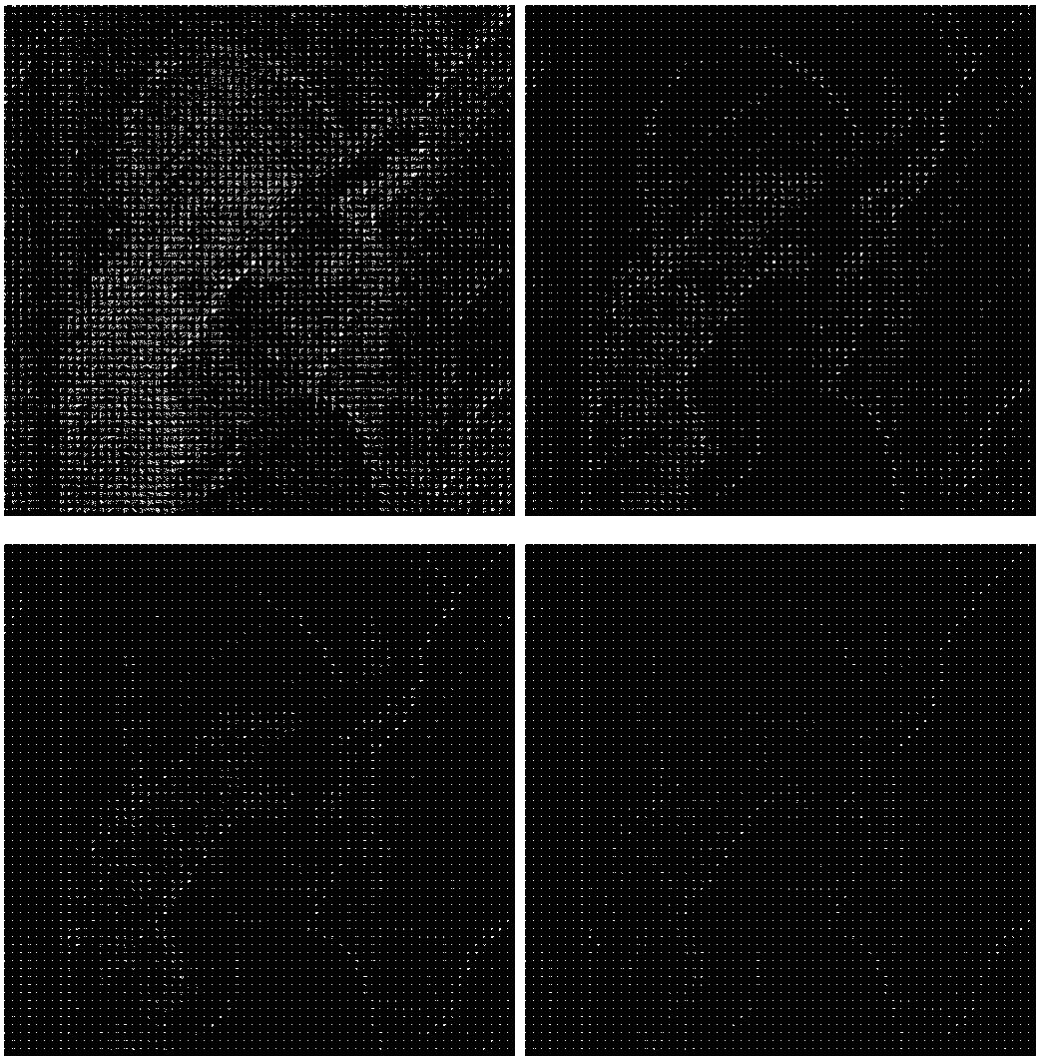

Figure 3.10: Quantized DCT coefficients with compression ratios $k = 1, 5, 10, 20$.

Figure 3.11: Recovered luminance (left) and chrominances C_r (middle) and C_b (right) for Lenna's color image using a level compression $k = 5$.

Figure 3.12: Recovered images after JPEG compression with ratios $k = 1, 5, 10, 20$.

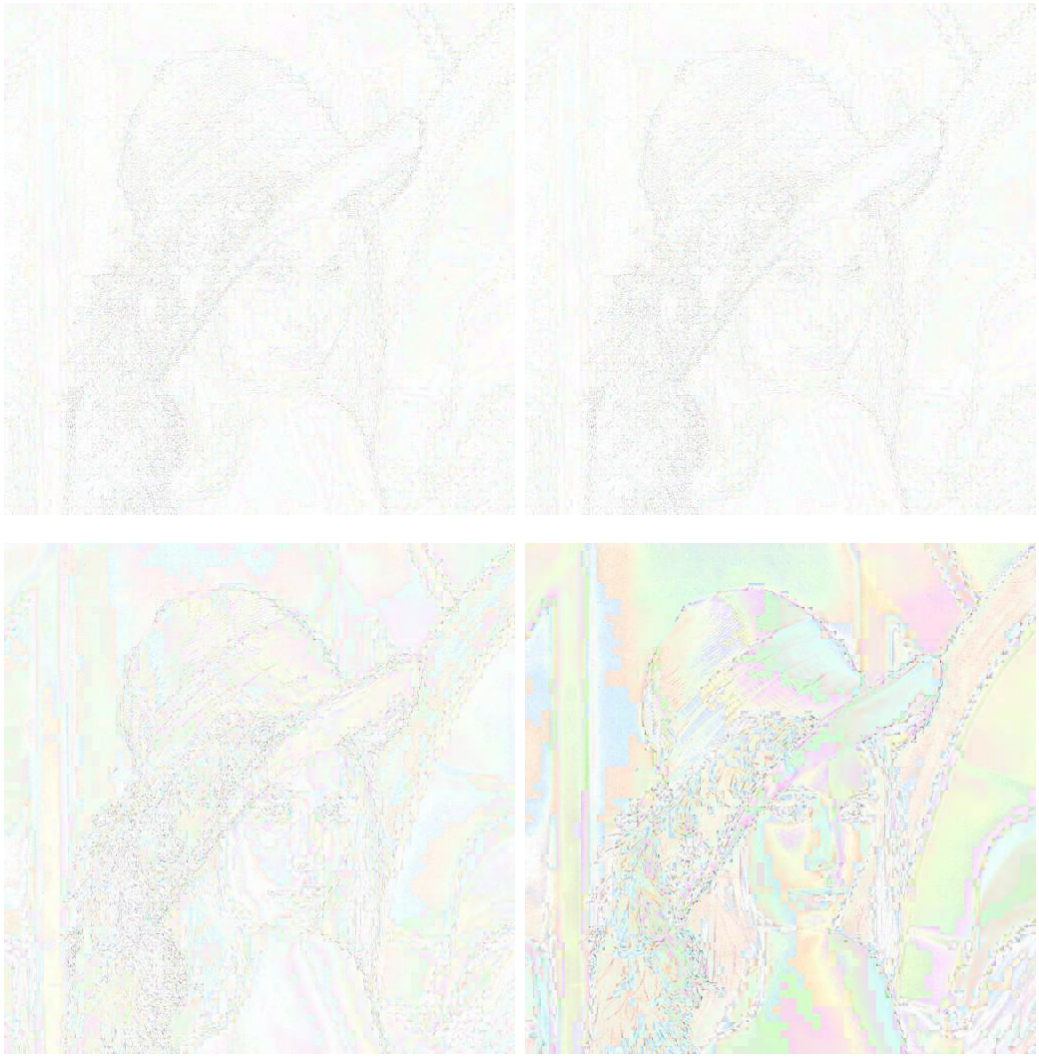

Figure 3.13: Error after JPEG compression with ratios $k = 1, 5, 10, 20$.

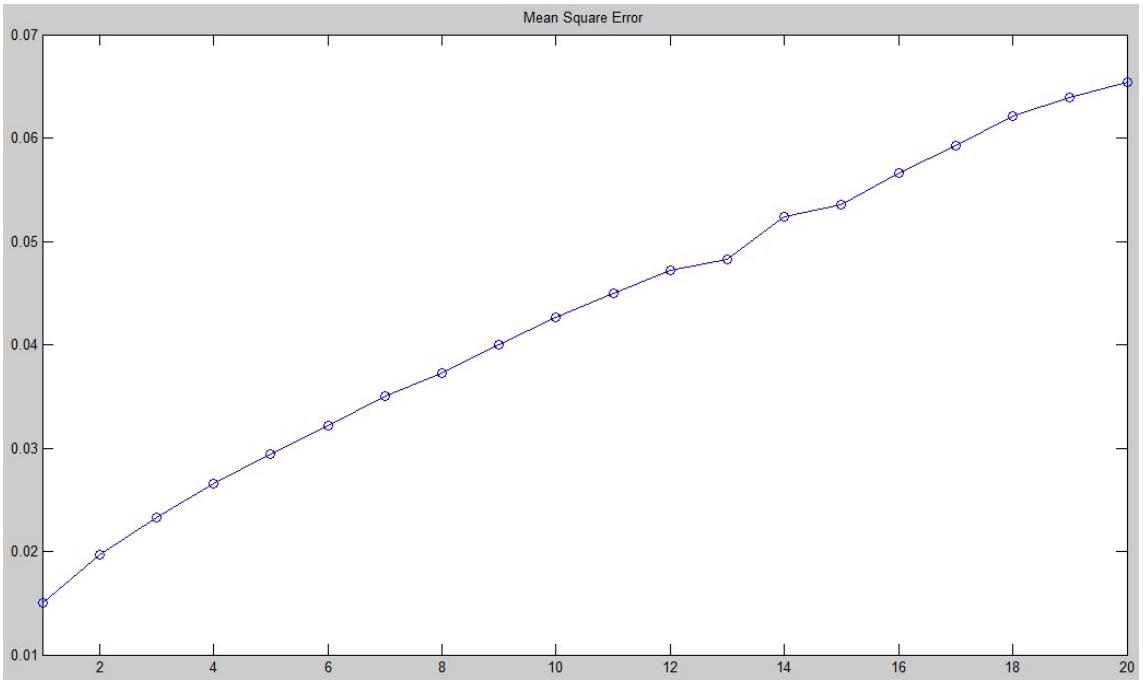

Figure 3.14: Mean square error as a function of compression ratio k for Lenna's image.

bits). We can view this sequence of bits as a random variable that takes the values 0 and 1 with certain probabilities p and $1-p$. Now, since an image has certain redundancies, we can exploit them to reduce the number of bits needed to store it. Suppose that X is a discrete random variable that can take values $\{x_1, \ldots, x_n\}$ with probabilities $p_k = P(X = x_k)$, for all $k \in \{1, \ldots, n\}$ representing the values of the DC components of the blocks in the DCT. Then, we can assume

$$p_k = \frac{n_k}{N}, \quad \text{for all } k \in \{1, \ldots, n\},$$

where n_k is the number of times that the value x_k appears in the DC sequence and $N = \sum_{k=1}^{n} n_k$. Hence, if l_k is the number of bits used to represent the value x_k, then the average length of the DC sequence will be

$$L_{\text{avg}} = \sum_{k=1}^{n} p_k l_k.$$

Now, if we choose to encode all DC values with the same number of bits m, then $l_k = m$, for all $k \in \{1, \ldots, n\}$ and the average length will be

$$L_{\text{avg}} = m \sum_{k=1}^{n} p_k = m.$$

But in this case we are not exploiting the redundancies in the image that will be translated on redundancies in the DC sequence. As a consequence of them, the

probabilities p_k are not uniformly distributed and, therefore, there will be privileged values of the variable X that will appear more frequently. If somehow we could encode the more frequent values with a smaller number of bits, then we would obtain a smaller average length for our DC sequence. Let us see this with a simple example. Suppose that the random variable X can take 4 values that can be encoded with 2 bits with probabilities shown in Table 3.1. The table also shows two possible codes for the values of the variable, the first code consists on using the same number of bits, in this case 2 bits, to encode each value. In the second code we choose to use only 1 bit for the value with greater probability, 2 bits for the second value with greater probability and 3 bits for the other 2 values. If we compute the average length of the sequence for each code we obtain, for the first code $L_1 = 2$ and for the second one

$$L_2 = \frac{1}{2} \cdot 1 + \frac{1}{4} \cdot 2 + \frac{1}{8} \cdot 3 + \frac{1}{8} \cdot 3 = 1.75.$$

X	p_k	**Code 1**	$l_1(x_k)$	**Code 2**	$l_2(x_k)$
x_1	1/8	00	2	010	13
x_2	1/2	01	2	1	1
x_3	1/8	10	2	011	3
x_4	1/4	11	2	00	2

Table 3.1: Random variable with 2 possible codes.

In order to compare the two codes we define the bit compression ratio.

Definition 3.3.1. Let X be a random variable taking values $\{x_1, \ldots, x_k\}$ that can be encoded using a maximum of m bits, $2^m = k$, for each value and let L_{av} be the average length for an alternative code for the values of the variable. We define the bit compression ratio of the alternative code by

$$C_b = \frac{m}{L_{\text{av}}}.$$

In our example we obtain a bit compression ratio equal to

$$C_b = \frac{m}{L_2} = \frac{2}{1.75} = 1.1428.$$

But now we could ask ourselves, how many bits do we need to encode these values? There exists an answer to this question given by Information Theory. Its fundamental premise is that the generation of information can be modeled as a probabilistic process that can be measured by the Shannon entropy.

Definition 3.3.2. Let X be a random variable that represents a source of random information that can take values $\{x_1, \ldots, x_k\}$ with probabilities p_k, for all $k \in \{1, \ldots, n\}$. The average information per source output, called the entropy of the source, is given by

$$H = \sum_{k=1}^{n} p_k \log_2 \frac{1}{p_k} = -\sum_{k=1}^{n} p_k \log_2 p_k.$$

Observe that if the random variable takes one value with probability 1, then the entropy value is $H = 0$, which means that no information is attributed to it because there is no uncertainty associated to the source. Now, if we compute the entropy for our example we get

$$H = -\frac{1}{2}\log_2\frac{1}{2} - \frac{1}{4}\log_2\frac{1}{4} - \frac{1}{8}\log_2\frac{1}{8} - \frac{1}{8}\log_2\frac{1}{8} = 1.75,$$

which coincides with the average length of the 2nd code in Table 3.1 we have used. This means that we cannot compress more the sequence than we have already done. Thus, in order to use this method we can approximate the probabilities p_k for $k \in \{1, \ldots, n\}$ using the histogram of the DC sequence. To apply the Huffman code to our sequence we have to rearrange the sequence of values (we call them symbols) of the variable in descending order of probability. Now we add the last two probabilities to obtain a new list of symbols in which the two last symbols of the previous list become a new symbol in the new list with probability equal to the sum of probabilities of the original symbols it comes from. We iterate the process until we obtain a final list with only two symbols and we assign the code 1 to the greater probability symbol and 0 to the other. Now, we step back to the previous list, where there will be two symbols that lead to one of the symbols of the last list with the code s. We assign to these symbols the codes $1s$, to the symbol with greater probability, and $0s$ to the other. The process is iterated until we reach the first list. For example, the code 2 shown in Table 3.1 is constructed in the following way: firstly we rearrange the symbols for X as in Table 3.2, secondly we combine the two last symbols x_1 and x_3 into the symbol x_{13} adding their probabilities to obtain the list for the variable X_1. Since the list for X_2 is already descending ordered by probability we do not have to rearrange the symbols. Thirdly we combine the two last symbols, x_{13} and x_4 in list for X_1, into the symbol x_{413} in the list for the variable X_2, adding their probabilities. Finally, we have a list with only two symbols, x_2 and x_{413}, with the same probability equal to $1/2$, thus we do not need to change the order of the symbols. Now we should choose the greatest probability symbol and assign the code 0 to it and the code 1 to the other one. We choose x_2 to codify with the code 1 and x_{413} with 0 as it is shown in the seventh column (code1) of Table 3.2.

X	p_k	X_1	p_k	X_2	p_k	Code1	Code2	Code3
x_2	1/2	x_2	1/2	x_2	1/2	1	1	1
x_4	1/4	x_4	1/4	x_{413}	1/2	0	00	00
x_1	1/8	x_{13}	1/4	-	-	-	01	010
x_3	1/8	-	-	-	-	-	-	011

Table 3.2: Random variable with 2 possible codes.

Now, we look at the previous list, for variable X_1, and we see that the symbol x_{413} came from the variables x_4 and x_{13} both with the same probability equal to $1/4$. Thus, we choose x_4 to assign a 0 and x_{13} to assign a 1, therefore the codes for x_4 and x_{13} will be 00 and 01, respectively, where the first zero in each code comes

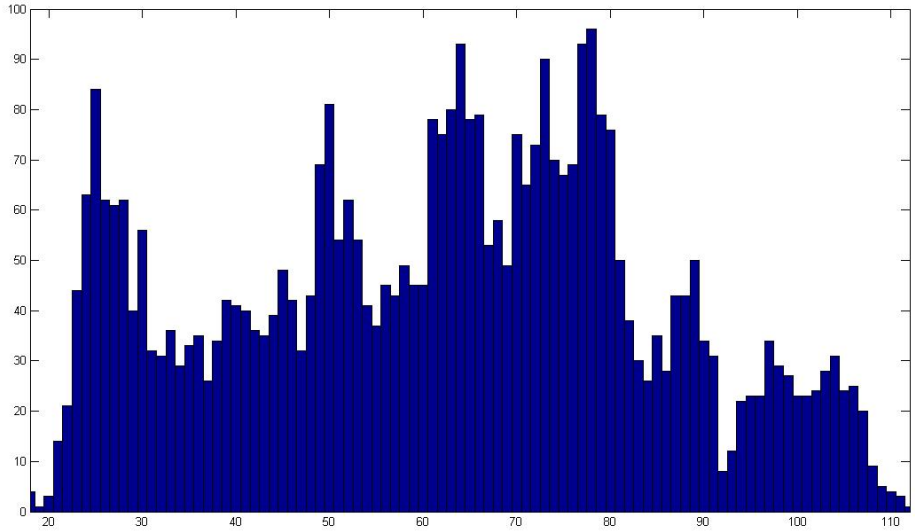

Figure 3.15: Histogram of the quantized coefficients DC components of the DCT blocks for gray scale Lenna's image.

from the zero of the symbol x_{413}. The results can be observed in column Code2 of Table 3.2. In the last step, we jump back from column X_1 to column X and we see that the variable x_{13} came from the variables x_1 and x_3, again both with the same probability, so we can choose any of them to assign the 1. We assign 0 to x_1 and 1 to x_3, and therefore, the codes for x_1 and x_3 will be 010 and 011, respectively. The resulting code corresponds to the one shown in the last column in Table 3.2. The original symbol sequence can be uniquely recovered from the coded bit stream sequence.

Figure 3.15 shows the histogram of the quantized DCT coefficients for all the blocks (left) and the histogram of the DC components (right) for Lenna's gray image that would be used to approximate the probabilities for the values of DC sequence to encode using the Huffman code.

But, since the JPEG uses differential Huffman encoding, Figure 3.16 shows the histogram (left) and the probability approximation obtained from this histogram for the difference between the DC components where only the first one remains the same. We notice that the high probabilities are around the zero value and the farther the coefficient is from zero the smaller its probability is.

Now in Figure 3.17 we can see the length of the bit string assigned to each value of the differential DC coefficients; we notice that coefficients with small probability have greater lengths. We have not deleted the coefficients with zero probability because they will not intervene in the computation of the average length of the bit stream, since they will be multiplied by their probability and it is zero. In Figure 3.17 (left) we observe averaged length of the bit sequence assigned to each

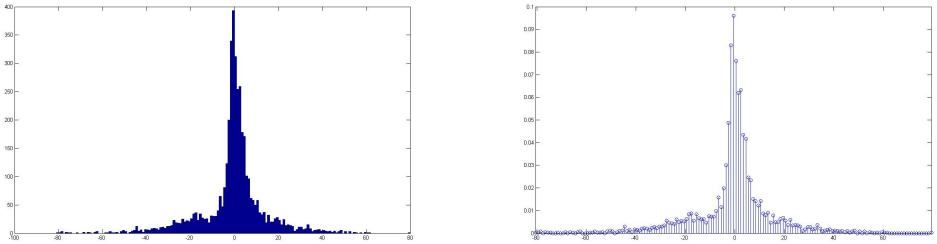

Figure 3.16: Histogram of the differential DC coefficients (left) and their probabilities (right).

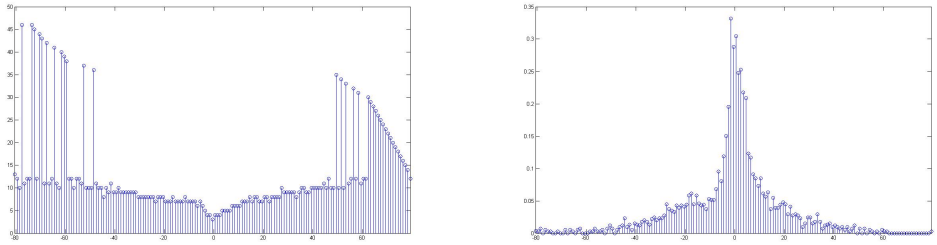

Figure 3.17: Number of bits assigned to each differential DC coefficient (left) and their averaged lengths (right).

coefficient; if we add all these lengths then we will obtain the total number of bits used in the compressed bit stream to encode the DC coefficients. This computation gives us a total averaged length of 5.4705. Hence, we obtain a bit compression ratio for the DC coefficients equal to

$$C_b = \frac{m}{L_{\text{av}}} = \frac{8}{5.4705} = 1.4624.$$

Now, computing the experimental entropy for this sequence we get

$$H_{\text{DC}} = \sum_{i=1}^{n} p_i \log_2 \frac{1}{p_i} = 5.4373,$$

which is very close to the average length $L_{\text{av}} = 5.4705$ we have obtained.

Finally, Figure 3.18 shows the total number of AC coefficients different from zero for each one of the 4094 quantized DCT blocks of Lenna's luminance. Hence, the number of bits needed to encode each block using RLE will be doubled. Observe that there are less than 30 coefficients different from zero in each block and hence we will need less than $2 \times 30 \times 8 = 480$ bits to encode each block, number to be compared with the $64 \times 8 = 252$ bits for the standard codification. Figure 3.19 shows the bit compression ratio obtained using RLE to encode the AC coefficients of each 8×8 pixels DCT block of Lenna's image.

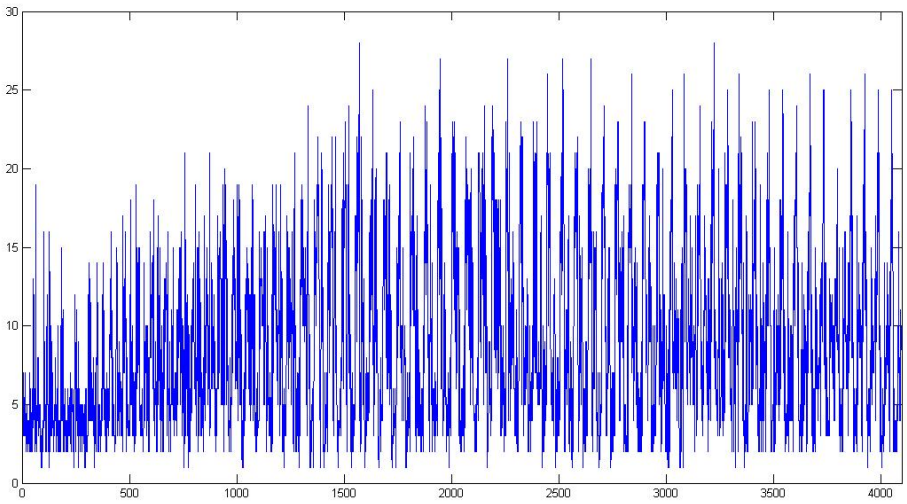

Figure 3.18: Number of AC coefficients different from zero for each 8×8 pixels block.

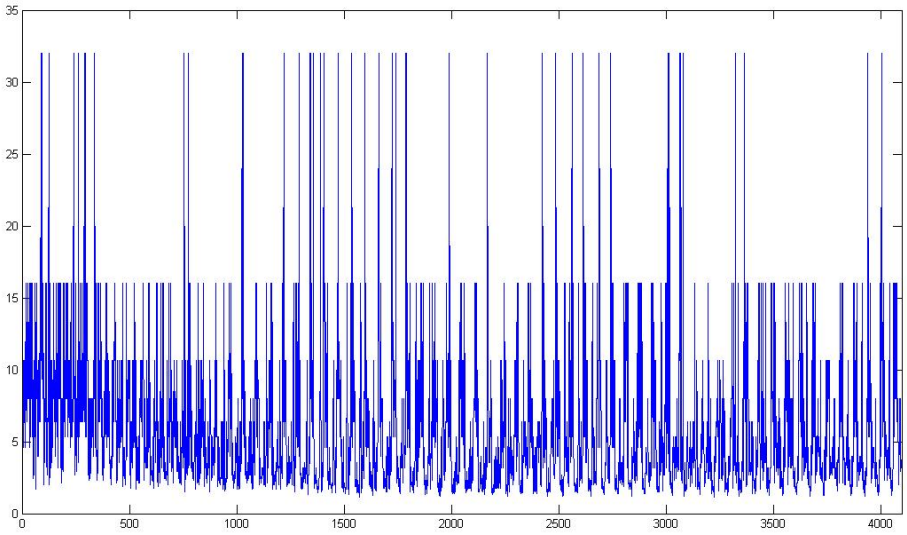

Figure 3.19: Bit compression ratio for the AC coefficients for each 8×8 pixels block.

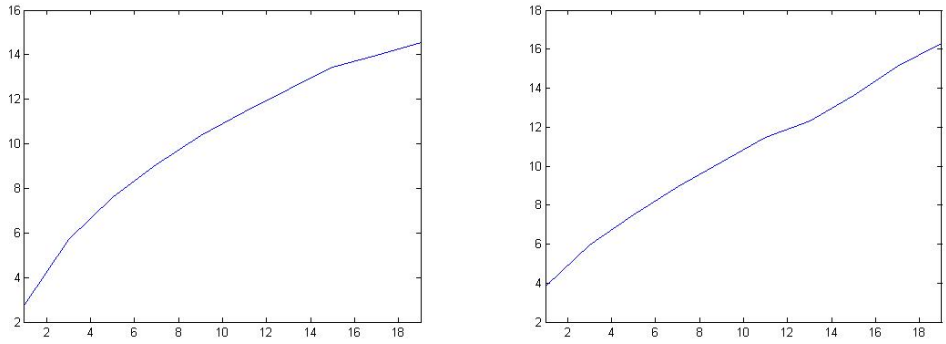

Figure 3.20: Bit compression ratio for gray version of Lenna (left) and the mean square error for the compressed image (right) as a function of the compression parameter k.

Combining the compression obtained with the Huffman code for the DC components and the one achieved using RLE for the AC coefficients we can compute the total amount of compression for the whole image using the JPEG algorithm. The total number of bits N_b needed to store the compressed image will be the sum of the number of bits N_{DC} used to encode the DC coefficients and the number N_{AC} used for the AC coefficients

$$N_b = N_{\text{DC}} + N_{\text{AC}} = 22407 + 570304 = 592711,$$

obtaining in that way a bit compression ratio for Lenna's image equal to

$$C_b = \frac{N_H N_W 8}{N_b} = \frac{512 \times 512 \times 8}{592711} = 3.5382,$$

where $N_H = 512$ and $N_W = 512$ are the height and the width in pixels of the image, respectively.

Figure 3.20 (left) shows the bit compression ratio obtained for the gray version of Lenna's image using JPEG compression with values of the compression parameter from 1 to 20. We observe, as it was expected, that the compression increases for larger values of k. In the right side of Figure 3.20 we can see the mean square error for the compressed images as a function of the parameter k.

3.4 Conclusion

The JPEG standard for still image compression uses the DCT basis to describe the image; this basis has the advantage to adapt well to images, but the disadvantage of complete non-locality. To compensate this, JPEG uses partitions of 8×8 blocks from the image and encodes them separately. This procedure induces the appearance of block artifacts in the compressed image that become more apparent when the

compression parameter k increases. A lot of literature has appeared in order to reduce this phenomenon when using JPEG compression, see for instance [14].

In order to achieve greater compression the algorithm uses the Run Length Encoding for the AC coefficients of the DCT blocks and the Huffman entropy code for the differential DC coefficients, but it is important to remark that the JPEG standard stores information data about the compression, such as the Huffman code used for each symbol, in the headers of the file and therefore, the compressions computed in the previous section will slightly diverge from the real ones because of this fact. On the other hand the JPEG standard sometimes uses encoding Huffman tables from statistics obtained using many images in order to avoid larger headings in the files, but this implies that the compression for our image is not as nicely adapted as it would be using our own Huffman codification.

Chapter 4

The JPEG 2000 standard image format

4.1 Introduction

We have seen in the previous chapter how breaking an image into 8×8 blocks and applying the DCT transform to each one independently produced artifacts in the reconstructed image. This is one of the reasons the JPEG 2000 has been developed. We will introduce now a second, but not less important argument. Suppose we are on holidays and have our digital camera to take a lot of pictures that we save on our memory card using the JPEG format. Since we are not very cautious we save the pictures with high quality and when there are many days left to finish our travel we have filled our memory card up. And we cannot buy another one! Therefore we have to erase some pictures or sacrifice the future ones. Imagine how amazing it will be if we could select some pictures and save them changing their quality in order to make room for new images. This property, the possibility of changing the compression ratio after compression, is known as scalability and it is one of the features provided by the JPEG 2000 standard. Another application of scalability can be found in the Internet. The pictures that we examine in a web browser are stored in a server and they are downloaded from this server to our computer. When the picture has high resolution, it has great size in the computer, the download speed cannot be enough and it produces long delays in the visualization. In order to avoid that problem, the web page owners usually save the images in the server with different resolutions to deliver them through the net as the user requires more quality. This problem can be solved using scalable image formats as JPEG 2000: since we can reconstruct the image at different resolution/quality ratios from the compressed bit stream, we do not need to store several copies of one image at different compression ratios, it suffices to store one image with high quality and deliver parts of the bit stream as the request for quality increases. Moreover, during the browsing the user can specify a spatial region of interest in the image and only the part of the bit stream corresponding to this region is downloaded in a progressive fashion so that a coarse version of this region is firstly delivered very quickly and then refined by sending

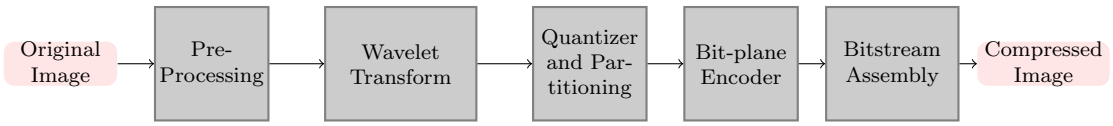

Figure 4.1: Diagram of the JPEG 2000 encoder.

more parts of its bit stream.

The development of JPEG 2000 was initiated in 1996, after the emerging of some compression algorithms early in 1990 with better compression performance and new features unseen before. After some technical contributions and the creation of some verification models, the JPEG 2000 became an international standard in December 2000.

In order to avoid the appearance of artifacts JPEG 2000 uses a wavelet transform with lifting implementation, as we will see in the next section, which has good localization properties, to process greater blocks than JPEG's. The wavelet also provides the resolution scalability property. Another characteristic is that the blocks processed in JPEG 2000 come from the wavelet domain instead of the space domain and the Run Length Encoding used in JPEG is substituted by bit-plane codification, which ensures the reconstruction of the image with graceful degradation from any truncated point of the bit stream formed after the bit-plane encoding. Finally, the compression is not produced by the quantification module, whose function is only to convert float coefficients into integer coefficients, instead it is generated by the bitstream assembly module and for that reason, only by reassembling the bitstream, the JPEG 2000 compressed bitstream can be converted to a higher compressed one without entropy coding and transform.

Figure 4.1 shows the operation flow for the JPEG 2000 algorithm that we will describe in detail in the next sections.

4.2 Wavelet transform by lifting

We have seen in Lemma 2.3.4 that we can write the coefficients of any vector $z \in \ell^2(\mathbb{Z}_N)$ in a wavelet basis $B = \{R_{2k}u\} \cup \{R_{2k}v\}$ as a convolution, given by

$$(z)_B = (z * \tilde{v}(0), z * \tilde{v}(2), \ldots, z * \tilde{v}(N-2), z * \tilde{u}(0), z * \tilde{u}(2), \ldots, z * \tilde{u}(N-2)),$$

where $(z)_B$ is the vector in the wavelet basis B. Therefore, we can compute the wavelet transform as two convolutions of the vector z with the vectors $\tilde{v}$ and $\tilde{u}$ keeping only the coefficients with even indexes. This last operation is known as decimating.

Definition 4.2.1. Let $z \in \ell^2(\mathbb{Z}_N)$, $N = 2M$, $M \in \mathbb{Z}$. We define the downsampling or decimation operator $\mathrm{D} : \ell^2(\mathbb{Z}_N) \to \ell^2(\mathbb{Z}_M)$ by

$$\mathrm{D}\, z(n) = z(2n), \quad \text{for } n = 0, 1, \ldots, M-1.$$

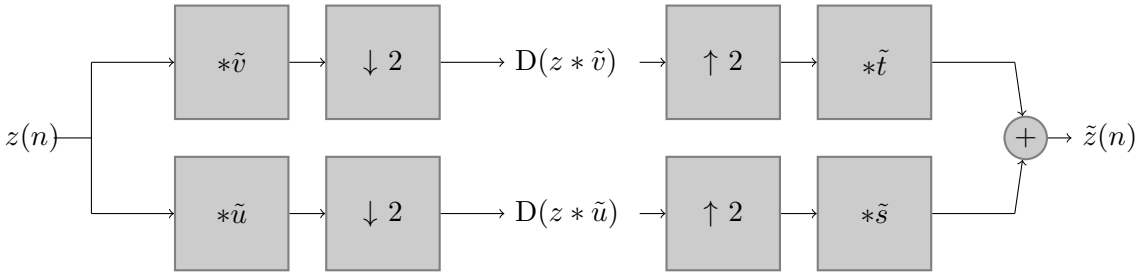

Figure 4.2: Diagram of the wavelet transform using a filtering bank.

We will represent this operation in diagrams by $\downarrow 2$.

Definition 4.2.2. Let $z \in \ell^2(\mathbb{Z}_M)$, $N = 2M$, $M \in \mathbb{Z}$. We define the upsampling operator $\mathrm{U} : \ell^2(\mathbb{Z}_M) \to \ell^2(\mathbb{Z}_N)$ by

$$\mathrm{U}\, z(n) = \begin{cases} z(n/2), & \text{if } n \text{ is even,} \\ 0, & \text{if } n \text{ is odd.} \end{cases}$$

We will represent this operation in diagrams by $\uparrow 2$.

Figure 4.2 shows the diagram for the wavelet transform of a vector z based on filter banks. The first blocks represent the filtering using FIR (Finite Impulse Response) filters with impulsive responses $\tilde{v}$ and $\tilde{u}$, respectively. The output of these filters is computed by convolution of the input vector with the impulse response of the filter. The second blocks compute the downsampled vectors. Therefore, we obtain the wavelet transform via filter banks,

$$(z)_B = (\mathrm{D}(z * \tilde{v}), \mathrm{D}(z * \tilde{u})).$$

Remark 4.2.3. Observe that $\mathrm{D} \circ \mathrm{U}(z) = z$, but in general $\mathrm{U} \circ \mathrm{D}(z) \neq z$. Recall that in Lemma 2.3.6 we defined $z^*(n) = (-1)^n z(n)$ and we have that

$$\mathrm{U} \circ \mathrm{D}(z) = \frac{1}{2}(z + z^*).$$

To reconstruct the original vector z we use the recovering filters $t, s \in \ell^2(\mathbb{Z}_N)$ which we need to find. The following lemma states the conditions for the filter bank to have a perfect reconstruction of our vector.

Lemma 4.2.4. *Let $M \in \mathbb{N}$, $N = 2M$ and $u, v, s, t \in \ell^2(\mathbb{Z}_N)$. Let $A(n)$ be the system matrix given by equation (2.9) for u and v. For all $z \in \ell^2(\mathbb{Z}_N)$, we have a perfect reconstruction in the filter bank of Figure 4.2 if and only if*

$$A(n) \begin{pmatrix} \hat{s}(n) \\ \hat{t}(n) \end{pmatrix} = \begin{pmatrix} \sqrt{2} \\ 0 \end{pmatrix}.$$

Moreover, if $A(n)$ is unitary, then $\hat{t}(n) = \hat{v}(n)$ and $\hat{s}(n) = \hat{u}(n)$. If $A(n)$ is unitary for all n, i.e., $B = \{R_{2k}u\} \cup \{R_{2k}v\}$ is an orthonormal basis, then $t = \tilde{v}$ and $s = \tilde{u}$.

Proof. We have that $\mathrm{U}\circ\mathrm{D}(z*\tilde{v}) = \frac{1}{2}(z*\tilde{v}+(z*\tilde{v})^*)$ and $\mathrm{U}\circ\mathrm{D}(z*\tilde{u}) = \frac{1}{2}(z*\tilde{u}+(z*\tilde{u})^*)$. On the other hand,

$$\widehat{\mathrm{U}\circ\mathrm{D}(z*\tilde{u})}(n) = \frac{1}{2}(\hat{z}(n)\overline{\hat{u}(n)} + \hat{z}(n+M)\overline{\hat{u}(n+M)})$$

and

$$\widehat{\mathrm{U}\circ\mathrm{D}(z*\tilde{v})}(n) = \frac{1}{2}(\hat{z}(n)\overline{\hat{v}(n)} + \hat{z}(n+M)\overline{\hat{v}(n+M)}).$$

Combining both expressions we get

$$\begin{aligned}(\tilde{t}*\mathrm{U}(\mathrm{D}(z*\tilde{v})) + \tilde{s}*\mathrm{U}(\mathrm{D}(z*\tilde{u})))\widehat{\ }(n) = &\frac{1}{2}\overline{\hat{t}(n)}\left(\hat{z}(n)\overline{\hat{v}(n)} + \hat{z}(n+M)\overline{\hat{v}(n+M)}\right)\\ &+ \frac{1}{2}\overline{\hat{s}(n)}\left(\hat{z}(n)\overline{\hat{u}(n)} + \hat{z}(n+M)\overline{\hat{u}(n+M)}\right).\end{aligned}$$

By the Fourier inverse theorem (see Equation (2.2)) this expression has to be equal to $\hat{z}(n)$, for all n and for all $z\in\ell^2(\mathbb{Z}_N)$, if we want to recover this vector. Therefore,

$$\begin{aligned}\hat{z}(n) &= \frac{1}{2}\overline{\hat{t}(n)}\left(\hat{z}(n)\overline{\hat{v}(n)} + \hat{z}(n+M)\overline{\hat{v}(n+M)}\right)\\ &\quad+ \frac{1}{2}\overline{\hat{s}(n)}\left(\hat{z}(n)\overline{\hat{u}(n)} + \hat{z}(n+M)\overline{\hat{u}(n+M)}\right)\\ &= \frac{1}{2}\hat{z}(n)\left(\overline{\hat{t}(n)\hat{v}(n)} + \overline{\hat{s}(n)\hat{u}(n)}\right)\\ &\quad+ \frac{1}{2}\hat{z}(n+M)\left(\overline{\hat{s}(n)\hat{u}(n+M)} + \overline{\hat{t}(n)\hat{v}(n+M)}\right).\end{aligned} \tag{4.1}$$

This equation holds if and only if

$$\hat{t}(n)\hat{v}(n) + \hat{s}(n)\hat{u}(n) = 2$$

and

$$\hat{s}(n)\hat{u}(n+M) + \hat{t}(n)\hat{v}(n+M) = 0.$$

It is obvious that if the last two equations hold, then we recover z in equation (4.1). Now, suppose that the equation (4.1) holds for all n and for all $z\in\ell^2(\mathbb{Z}_N)$. Suppose that we fix n and choose z such that $\hat{z}(n)=1$ and $\hat{z}(n+M)=0$ then, substituting these values in equation (4.1) we obtain that

$$\hat{t}(n)\hat{v}(n) + \hat{s}(n)\hat{u}(n) = 2.$$

If we consider now another vector z such that $\hat{z}(n)=0$ and $\hat{z}(n+M)=1$, then we obtain the second condition

$$\hat{s}(n)\hat{u}(n+M) + \hat{t}(n)\hat{v}(n+M) = 0.$$

If we divide by $\sqrt{2}$ and write these two conditions in matrix form we get

$$\frac{1}{\sqrt{2}}\begin{pmatrix}\hat{u}(n) & \hat{v}(n)\\ \hat{u}(n+M) & \hat{v}(n+M)\end{pmatrix}\begin{pmatrix}\hat{s}(n)\\ \hat{t}(n)\end{pmatrix} = A(n)\begin{pmatrix}\hat{s}(n)\\ \hat{t}(n)\end{pmatrix} = \begin{pmatrix}\sqrt{2}\\ 0\end{pmatrix}.$$

Now, if $A(n)$ is unitary, $\overline{A(n)}A(n) = I$, then

$$\overline{A(n)}A(n)\begin{pmatrix}\hat{s}(n)\\ \hat{t}(n)\end{pmatrix} = \overline{A(n)}\begin{pmatrix}\sqrt{2}\\ 0\end{pmatrix}$$

and

$$\begin{pmatrix}\hat{s}(n)\\ \hat{t}(n)\end{pmatrix} = \overline{A(n)}\begin{pmatrix}\sqrt{2}\\ 0\end{pmatrix} = \frac{1}{\sqrt{2}}\overline{\begin{pmatrix}\hat{u}(n) & \hat{u}(n+M)\\ \hat{v}(n) & \hat{v}(n+M)\end{pmatrix}}\begin{pmatrix}\sqrt{2}\\ 0\end{pmatrix}.$$

Therefore, $\hat{s}(n) = \overline{\hat{u}(n)}$ and $\hat{t}(n) = \overline{\hat{v}(n)}$, and if $A(n)$ is unitary for all n, applying again the Fourier inverse theorem (Equation (2.2)) we obtain $s = \tilde{u}$ and $t = \tilde{v}$. □

Remark 4.2.5. Observe that to reconstruct the original vector z using the filter bank in Figure 4.2 we do not need to impose the matrix $A(n)$ to be unitary, i.e. the basis does not need to be orthogonal, so we are in a more general case that generalizes the orthogonal wavelets, that we will deal with below, called bi-orthogonal wavelets.

We have shown how to implement a first-stage wavelet transform using a bank of filters, where the vector u carries information about the low frequencies and the vector v about the high frequencies. Now we can apply the technique of bank of filters to obtain more refined scales of frequencies by iterating this process. If N is divisible by 4 we can apply the same type of filter to $y_1 = D(z * \tilde{u})$ and $x_1 = D(z * \tilde{v})$ to obtain a more refined scale in frequency domain, resulting in $y_2 = D(y_1 * \tilde{u_1})$ and $x_2 = D(x_1 * \tilde{v_1})$, where the filters $u_1, v_1 \in \ell^2(\mathbb{Z}_{N/2})$. If N is divisible by 2^p we can repeat this process up to p times. Usually the iteration process is only applied to the vector resulting from the lowpass filter.

Definition 4.2.6. Let $N \in \mathbb{N}$ be divisible by 2^p, $p \in \mathbb{N}$. A p-th stage wavelet filter is a sequence $\{u_1, v_1, u_2, v_2, \ldots, u_p, v_p\}$ such that, for each $l = 1, 2, \ldots, p$, $u_l, v_l \in \ell^2(Z_{N/2^{l-1}})$ and the system matrix

$$A_l(n) = \begin{pmatrix}\hat{u}_l(n) & \hat{v}_l(n)\\ \hat{u}_l(n + N/2^l) & \hat{v}_l(n + N/2^l)\end{pmatrix}$$

is unitary, for all $n = 0, 1, \ldots, N/2^l - 1$. If $z \in \ell^2(\mathbb{Z}_N)$, the output of the p-th filter bank is the set of vectors $\{x_1, x_2, \ldots, x_p, y_p\}$ given by

$$\begin{aligned} x_1 &= \mathrm{D}(z * \tilde{v}_1),\\ y_1 &= \mathrm{D}(z * \tilde{u}_1),\\ &\vdots\\ x_l &= \mathrm{D}(x_{l-1} * \tilde{v}_l),\\ y_l &= \mathrm{D}(y_{l-1} * \tilde{u}_l),\end{aligned}$$

for all $l = 2, 3, \ldots, p$.

Figure 4.3 shows a diagram of this construction for a second stage wavelet transform. The output of the low-pass filter y_1 is filtered again to produce the low frequency version, y_2, of the input data and a high frequency output x_2.

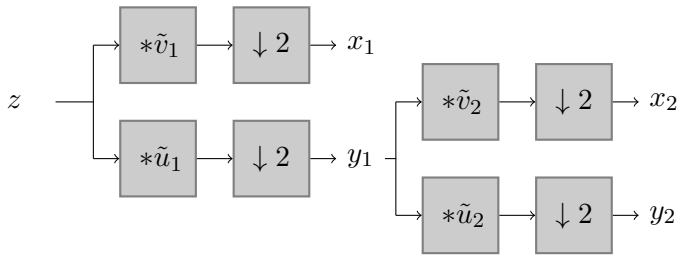

Figure 4.3: Diagram of 2 stage wavelet transform using a filtering bank.

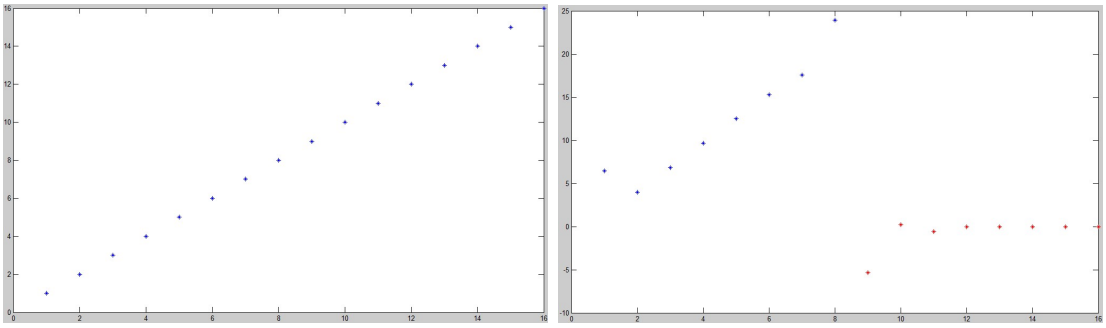

Figure 4.4: Ramp signal and its Daubechies D6 wavelet transform.

4.2.1 Bi-orthogonal scaling filters

We have seen, in JPEG compression standard, that given an image we transform it using some change of basis and then we apply a quantization scheme. This procedure is typical in image processing. When the transformation is orthogonal, it can be shown that the error in the quantized of the transformed coefficients is the same as the error between the original image and the compressed one. But orthogonal transformation also have disadvantages. In lossless image compression the quantization step is omitted and we need to use a transform that maps integers to integers. Although it is possible to modify an orthogonal transform to do such task, it is easier to use another class of wavelets, bi-orthogonal wavelets, that sacrifices the orthogonality but are much easier to adapt to map integers to integers. A second problem is similar to the explained when we talked about the DCT transform. Recall that the wavelet transform is a change of basis and it can be expressed as a product of a matrix W_N, the change of basis matrix, by our image. Since the matrix is formed by the basis vectors u and v shifted, we produce wrappings in the rows of the matrix which induce problems in many applications. Let us see it in the same example we used for the DCT. We consider the ramp vector $z = (z(k))_k \subset \ell^2(\mathbb{Z}_N)$, where $z(k) = k$, and we compute its wavelet transform using a Daubechies D6 wavelet. In Figure 4.4 we see the ramp vector (left) and the resulting transform (right). We can observe how the wrapping rows have an undesirable effect on the boundary of the signal.

This result can also be improved by using bi-orthogonal wavelets. The main idea is to renounce to the orthogonality of the basis, that gave us a way to produce a fast inversion of the transform, and substitute it by bi-orthogonality, i.e., we will look for two pair of filters $(\tilde{u}, \tilde{v})$ and $(\tilde{s}, \tilde{t})$, instead of one, with matrices of change of basis $\widetilde{W}_N$ and W_N, respectively, such that $\widetilde{W}_N^{-1} = W_N^T$, i.e.,

$$\widetilde{W}_N W_N^T = I.$$

Historically, the research about wavelets using filter banks has been developed using other techniques rather than Fourier transforms, in particular, the z-transform is a frequently used tool. For this reason, we are going to continue this analysis from this tool. We have already mentioned the concept of Finite Impulse Response Filter, that is characterized by its finite impulse response. Now we are going to define its z-transform.

Definition 4.2.7. A filter h is a linear time invariant operator completely defined by its impulse response $\{k(n) : n \in \mathbb{Z}\}$. The filter has finite response if only a finite number of the coefficients $h(n)$ are different from zero, i.e., $h \in \ell^2(\mathbb{Z}_N)$, for some $N \in \mathbb{Z}$. Let $p = \min\{n \in \mathbb{Z} : h(n) \neq 0\}$ and $q = \max\{n \in \mathbb{Z} : h(n) \neq 0\}$, the z-transform of the filter h is a Laurent polynomial given by

$$H(z) = \sum_{n=p}^{q} h(n) z^{-n}.$$

The degree of the Laurent polynomial is defined by $|h| = q - p$. Hence, the length of the filter h will be $|h| + 1$.

Remark 4.2.8.

1. The degree of the Laurent polynomial z^p is zero.
2. We set the degree of the 0 polynomial as $-\infty$.
3. The set of Laurent polynomials with the sum and the product has a ring structure. In this ring the division with remainder is possible, but not necessarily unique.

Definition 4.2.9. The set of 2×2 matrices of Laurent polynomials is denoted by $M(2; \mathbb{R}[z, z^{-1}])$. This set has a ring structure. If the determinant is a monomial, then the matrix is invertible. The set of invertible 2×2 matrices of Laurent polynomials is denoted by $GL(2; \mathbb{R}[z, z^{-1}])$. A matrix $M \in GL(2; \mathbb{R}[z, z^{-1}])$ is unitary if $M(z)^{-1} = M(z^{-1})^T$.

Proposition 4.2.10. *Let $\tilde{u}, \tilde{v}, \tilde{s}, \tilde{t} \in \ell^2(\mathbb{Z}_N)$ be the impulse responses of the FIR filters of Figure 4.2 with real coefficients. If the z-transforms of the filters satisfy*

$$T(z)\tilde{V}(z^{-1}) + S(z)\tilde{U}(z^{-1}) = 2$$

and

$$S(z)\tilde{U}(-z^{-1}) + T(z)\tilde{V}(-z^{-1}) = 0,$$

then we will have a perfect reconstruction of the input vector.

Proof. First observe that the discrete Fourier transform can be recovered from the z-transform by substituting $z = e^{2\pi in/N}$:

$$\hat{v}(n) = \sum_{k=0}^{N-1} v(k)e^{-2\pi ikn/N} = \sum_{k=0}^{N-1} v(k)z^{-k}\Big|_{z=e^{2\pi in/N}} = V(z)|_{z=e^{2\pi in/N}}.$$

It is enough to prove that the conditions in the proposition imply the conditions for perfect reconstruction

$$\hat{t}(n)\hat{v}(n) + \hat{s}(n)\hat{u}(n) = 2$$

and

$$\hat{s}(n)\hat{u}(n+M) + \hat{t}(n)\hat{v}(n+M) = 0.$$

Now, since the analysis filters have impulse responses $\tilde{u}$ and $\tilde{v}$, we have to relate their Fourier transforms with the Fourier transforms of u and v. Recall that $\tilde{v}(n) = \overline{v(-n)} = \overline{v(N-n)}$. Then, we claim that

$$\widehat{\tilde{v}}(n) = \overline{\hat{v}(n)}.$$

In fact,

$$\begin{aligned}\widehat{\tilde{v}}(n) &= \sum_{k=0}^{N-1} \tilde{v}(k)e^{-2\pi ink/N} = \sum_{k=0}^{N-1} \overline{v(N-k)}e^{-2\pi ink/N} = \sum_{j=0}^{N-1} \overline{v(j)}e^{-2\pi in(N-j)/N} \\ &= \sum_{j=0}^{N-1} \overline{v(j)}e^{2\pi inj/N} = \overline{\sum_{j=0}^{N-1} v(j)e^{-2\pi inj/N}} = \overline{\hat{v}(n)}.\end{aligned}$$

Therefore, $\overline{\widehat{\tilde{v}}(n)} = \hat{v}(n)$. Now, since the filter coefficients are real, i.e., $\overline{v(n)} = v(n)$, we have that

$$\overline{\widehat{\tilde{v}}(n)} = \overline{\sum_{k=0}^{N-1} \tilde{v}(k)e^{-2\pi ikn/N}} = \sum_{k=0}^{N-1} \overline{\tilde{v}(k)}e^{2\pi ikn/N} = \sum_{k=0}^{N-1} \tilde{v}(k)e^{2\pi ikn/N} = \tilde{V}(z^{-1})|_{z=e^{2\pi in/N}}.$$

Therefore, if we substitute $z = e^{2\pi in/N}$ in

$$T(z)\tilde{V}(z^{-1}) + S(z)\tilde{U}(z^{-1}) = 2,$$

we get

$$T(e^{2\pi in/N})\tilde{V}(e^{-2\pi in/N}) + S(e^{2\pi in/N})\tilde{U}(e^{-2\pi in/N}) = 2,$$

which is the same as

$$\hat{t}(n)\hat{v}(n) + \hat{s}(n)\hat{u}(n) = 2.$$

For the other condition we have

$$\begin{aligned}\hat{v}(n+M) = \overline{\widehat{\tilde{v}}(n+M)} &= \sum_{k=0}^{N-1} \tilde{v}(k) e^{2\pi i(n+M)k/N} = \sum_{k=0}^{N-1} \tilde{v}(k) e^{2\pi i(n+N/2)k/N} \\ &= \sum_{k=0}^{N-1} \tilde{v}(k) e^{2\pi i n k/N} (-1)^k = \sum_{k=0}^{N-1} \tilde{v}(k) (e^{-2\pi i n/N})^{-k} (-1)^k \\ &= \sum_{k=0}^{N-1} \tilde{v}(k) (-e^{-2\pi i n/N})^{-k} = \tilde{V}(-z^{-1})|_{z=e^{2\pi i n/N}}.\end{aligned}$$

Similarly, we obtain an analogous result for $\hat{u}(n+M) = \tilde{U}(-z^{-1})|_{z=e^{2\pi i n/N}}$. Therefore, substituting $z = e^{2\pi i n/N}$ in the equation

$$S(z)\tilde{U}(-z^{-1}) + T(z)\tilde{V}(-z^{-1}) = 0$$

we obtain the second condition for perfect reconstruction

$$\hat{t}(n)\hat{v}(n+M) + \hat{s}(n)\hat{u}(n+M) = 0.$$

□

Definition 4.2.11. The modulation matrix $M(z)$ is given by

$$M(z) = \begin{pmatrix} S(z) & S(-z) \\ T(z) & T(-z) \end{pmatrix}$$

and the dual modulation matrix $\widetilde{M}(z)$ is

$$\widetilde{M}(z) = \begin{pmatrix} \widetilde{U}(z) & \widetilde{U}(-z) \\ \widetilde{V}(z) & \widetilde{V}(-z) \end{pmatrix}.$$

With this definition, now the conditions for perfect reconstruction can be written in matrix form as

$$\widetilde{M}(z^{-1})^T M(z) = 2I,$$

where I is the 2×2 identity matrix.

Remark 4.2.12. Observe that for FIR filters $M, \widetilde{M} \in GL(2; \mathbb{R}[z, z^{-1}])$, i.e., they are invertible.

Definition 4.2.13. Given a filter h, its polyphase representation is given by

$$H(z) = H_e(z^2) + z^{-1} H_o(z^2),$$

where

$$H_e(z) = \sum_{k \in \mathbb{Z}} h(2k) z^{-k} \quad \text{and} \quad H_o(z) = \sum_{k \in \mathbb{Z}} h(2k+1) z^{-k}.$$

Observe that $H_e(z^2) = \sum_{k\in\mathbb{Z}} h(2k)z^{-2k}$ corresponds to the z-transform of the even coefficients of the filter and $H_o(z^2) = \sum_{k\in\mathbb{Z}} h(2k+1)z^{-2k} = z\sum_{k\in\mathbb{Z}} h(2k+1)z^{-(2k+1)}$ corresponds to the contribution of the odd coefficients. Therefore, we can recover $H_e(z)$ and $H_o(z)$ from $H(z)$

$$H_e(z^2) = \frac{H(z)+H(-z)}{2} \quad \text{and} \quad H_o(z^2) = \frac{H(z)-H(-z)}{2z^{-1}}.$$

Now we can write the outputs, after the subsampling operations, for the lowpass filter and the highpass filter represented in Figure 4.2 as

$$\begin{aligned} Y_l(z) &= U_e(z)X_e(z) + z^{-1}U_o(z)X_o(x), \\ Y_h(x) &= V_e(z)X_e(z) + z^{-1}V_o(z)X_o(x), \end{aligned}$$

where $x \in \ell^2(\mathbb{Z}_N)$ is the input vector, $X(z)$ is its z-transform and $Y_l(z)$ and $Y_h(z)$ are the low frequency and high frequency outputs, respectively. These expressions can be written in matrix form as

$$\begin{pmatrix} Y_l(z) \\ Y_h(z) \end{pmatrix} = \begin{pmatrix} \widetilde{U}_e(z) & \widetilde{U}_o(z) \\ \widetilde{V}_e(z) & \widetilde{V}_o(z) \end{pmatrix} \begin{pmatrix} X_e(z) \\ z^{-1}X_o(z) \end{pmatrix}.$$

The output $\tilde{x}$ of the synthesis filter (right part of Figure 4.2) can be computed by

$$\begin{pmatrix} \widetilde{X}_e(z) \\ z\widetilde{X}_o(z) \end{pmatrix} = \begin{pmatrix} S_e(z) & T_e(z) \\ S_o(z) & T_o(z) \end{pmatrix} \begin{pmatrix} Y_l(z) \\ Y_h(z) \end{pmatrix}.$$

Definition 4.2.14. Given the bank of filters of Figure 4.2, its polyphase matrix is given by

$$P(z) = \begin{pmatrix} S_e(z) & T_e(z) \\ S_o(z) & T_o(z) \end{pmatrix},$$

and its dual polyphase matrix is

$$\widetilde{P}(z) = \begin{pmatrix} \widetilde{U}_e(z) & \widetilde{V}_e(z) \\ \widetilde{U}_o(z) & \widetilde{V}_o(z) \end{pmatrix}.$$

Observe that

$$P(z^2)^T = \frac{1}{2}M(z)\begin{pmatrix} 1 & z \\ 1 & -z \end{pmatrix}$$

and

$$\widetilde{P}(z^2)^T = \frac{1}{2}\widetilde{M}(z)\begin{pmatrix} 1 & z \\ 1 & -z \end{pmatrix}.$$

Now the perfect reconstruction conditions can be expressed in terms of the polyphase matrices as

$$\begin{pmatrix} \widetilde{X}_e(z) \\ z\widetilde{X}_o(z) \end{pmatrix} = P(z)\begin{pmatrix} Y_l(z) \\ Y_h(z) \end{pmatrix} = P(z)\widetilde{P}(z)\begin{pmatrix} X_e(z) \\ X_o(z) \end{pmatrix}.$$

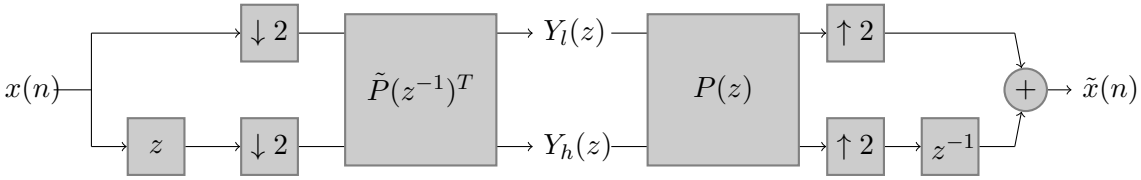

Figure 4.5: Polyphase representation of the wavelet transform.

$$P(z)\widetilde{P}(z^{-1})^T = I.$$

This last condition implies that $\det P(z)$ and $\det P(z)^{-1}$ are both Laurent polynomials, and this is possible only if both are monomials, i.e., $\det P(z) = C_1 z^l$ and $\det P(z)^{-1} = C_2 z^k$. We can assume without loss of generality that $\det P(z) = 1$, if not, we always can rescale the filter $T(z)$ to obtain this result. So, we have transformed the problem of finding the FIR wavelet transform into the problem of finding a polyphase matrix $P(z)$ with determinant one. Once we obtain $P(z)$, the filters can be found by the relation

$$P(z)\widetilde{P}(z^{-1})^T = I,$$

which implies

$$\begin{pmatrix} S_e(z) & T_e(z) \\ S_o(z) & T_o(z) \end{pmatrix} \begin{pmatrix} \widetilde{U}_e(z^{-1}) & \widetilde{V}_e(z^{-1}) \\ \widetilde{U}_o(z^{-1}) & \widetilde{V}_o(z^{-1}) \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

Using that $\det P(z) = 1$ and the Kramer's rule we solve the system to obtain

$$\widetilde{U}_e(z^{-1}) = \begin{vmatrix} 1 & T_e(z) \\ 0 & T_o(z) \end{vmatrix} = T_o(z), \quad \widetilde{V}_e(z^{-1}) = \begin{vmatrix} S_e(z) & 1 \\ S_o(z) & 0 \end{vmatrix} = -S_o(z),$$

$$\widetilde{U}_o(z^{-1}) = \begin{vmatrix} 0 & T_e(z) \\ 1 & T_o(z) \end{vmatrix} = -T_e(z), \quad \widetilde{V}_o(z^{-1}) = \begin{vmatrix} S_e(z) & 0 \\ S_o(z) & 1 \end{vmatrix} = S_e(z).$$

And finally,

$$\widetilde{U}(z) = \widetilde{U}_e(z^2) + z^{-1}\widetilde{U}_o(z^2) = -z^{-1}T(-z^{-1})$$

and

$$\widetilde{V}(z) = \widetilde{V}_e(z^2) + z^{-1}\widetilde{V}_o(z^2) = -z^{-1}S(-z^{-1}).$$

Figure 4.5 shows the wavelet transform operations in terms of the polyphase matrices. We first decimate the input vector and then compute the outputs of the lowpass (Y_l) and highpass (Y_h) filters by multiplying by the polyphase matrix $\tilde{P}(z^{-1})^T$, where the block with a z inside performs a product of the z-transform of the entry by z, producing a shift to the right of one position in the input vector in the space domain. So in the branch below, when we decimate the vector, we

keep the values at odd indexes because of the shift. The block with the z^{-1} label produces tho opposite effect, it shifts the vector one position to the left, so at the end we join the even and the odd samples of the vector. Observe that using this scheme we perform the subsampling before filtering, i.e. we subsample and after we convolve, this represents a great saving in computation if we compare it with the previous method, where we first filtered (convolved) and after that we decimated. So we were throwing away half of the coefficients after computing them.

Using the polyphase matrices we can factor the wavelet filter into lifting steps.

Definition 4.2.15. A filter pair (s,t) is complementary if the polyphase matrix $P(z)$ has determinant one. If the filter pair (s,t) is complementary also it is the filter $(\tilde{u},\tilde{v})$.

Theorem 4.2.16. *(Lifting theorem) Let (s,t) be a complementary pair filters. A filter t^{new} is complementary to s if and only if*

$$T^{\text{new}}(z) = T(z) + S(z)H(z^2),$$

where $H(z)$ is a Laurent polynomial.

Proof. We prove only the if part that is what we need. We can compute the polyphase components of $S(z)H(z^2)$:

$$\begin{aligned}
(S(z)H(z^2))_e &= \frac{1}{2}(S(z^{1/2})H((z^{1/2})^2) + S(-z^{1/2})H((-z^{1/2})^2)) \\
&= \frac{1}{2}(S(z^{1/2})H(z) + S(-z^{1/2})H(z)) = S_e(z)H(z), \\
(S(z)H(z^2))_o &= \frac{1}{2}z^{1/2}(S(z^{1/2})H((z^{1/2})^2) - S(-z^{1/2})H((-z^{1/2})^2)) \\
&= \frac{1}{2}z^{1/2}(S(z^{1/2})H(z) - S(-z^{1/2})H(z)) = S_o(z)H(z).
\end{aligned}$$

Then, the new polyphase matrix will be

$$\begin{aligned}
P^{\text{new}}(z) &= \begin{pmatrix} S_e(z) & T_e^{\text{new}}(z) \\ S_o(z) & T_o^{\text{new}}(z) \end{pmatrix} = \begin{pmatrix} S_e(z) & T_e(z) + S_e(z)H(z) \\ S_o(z) & T_o(z) + S_o(z)H(z) \end{pmatrix} \\
&= P(z)\begin{pmatrix} 1 & H(z) \\ 0 & 1 \end{pmatrix}.
\end{aligned}$$

And $\det P^{\text{new}}(z) = \det P = 1$. □

The new dual polyphase matrix will be

$$\widetilde{P}^{\text{new}}(z) = \widetilde{P}(z)\begin{pmatrix} 1 & 0 \\ -H(z^{-1}) & 1 \end{pmatrix}.$$

We have that

$$\begin{aligned}
P^{\text{new}}(z)\widetilde{P}^{\text{new}}(z^{-1})^T &= P(z)\begin{pmatrix} 1 & H(z) \\ 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & -H(z) \\ 0 & 1 \end{pmatrix}\widetilde{P}(z^{-1})^T \\
&= P(z)\widetilde{P}(z^{-1})^T = I.
\end{aligned}$$

Hence with the lifting we have a new filter $\widetilde{U}^{\text{new}}$ given by

$$\widetilde{U}^{\text{new}}(z) = \widetilde{U}(z) - V(z)H(z^{-2}).$$

Theorem 4.2.17. *(Dual Lifting theorem) Let (s,t) be a complementary pair filters. A filter s^{new} is complementary to t if and only if*

$$S^{\text{new}}(z) = S(z) + T(z)G(z^2),$$

where $G(z)$ is a Laurent polynomial.

There is a whole family of wavelets constructed using a dual lifting step and a normal lifting step (see [7]). Now, by iterating the process we can factorize the polyphase matrix into a sequence of lifting steps and we can write the polyphase matrix using this theorem

Theorem 4.2.18. *Let (s,t) be a complementary filter pair. Then there exist constants $K_1, K_2 \neq 0$, an integer $m > 0$ and Laurent polynomials H_i and G_i, for $i = 1, \ldots, m$, such that*

$$P(z) = \begin{pmatrix} K_1 & 0 \\ 0 & K_2 \end{pmatrix} \prod_{i=1}^{m} \begin{pmatrix} 1 & H_i(z) \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ G_i(z) & 1 \end{pmatrix}.$$

The lifting wavelet can be directly inversed by

$$P(z)^{-1} = \begin{pmatrix} 1/K_1 & 0 \\ 0 & 1/K_2 \end{pmatrix} \prod_{i=0}^{m} \begin{pmatrix} 1 & 0 \\ -G_i(z) & 1 \end{pmatrix} \begin{pmatrix} 1 & -H_i(z) \\ 0 & 1 \end{pmatrix}.$$

4.3 Bi-orthogonal 9-7 wavelet and Boundary Extensions

The JPEG 2000 standard uses, by default, the bi-orthogonal 9-7 wavelet. This is a 4-stage lifting wavelet implemented with a 9-coefficient analysis filter and a 7-coefficient synthesis filter. Starting from the analysis filter we can obtain the synthesis one and the apply the lifting scheme to perform the transform of the input data. Let

$$u = (u(-4), u(-3), u(-2), u(-1), u(0), u(1), u(2), u(3), u(4)) \in \ell^2(\mathbb{Z}_9)$$

be the analysis filter, with $u(n) = u(-n)$, for all $n = 1, 2, 3, 4$. The coefficients for this filter can be found in [6] and are given by

$$u(n) = \begin{cases} 0.602949018236, & \text{if} \quad n = 0, \\ 0.266864118443, & \text{if} \quad n = \pm 1, \\ -0.078223266529, & \text{if} \quad n = \pm 2, \\ -0.016864118443, & \text{if} \quad n = \pm 3, \\ 0.026748757411, & \text{if} \quad n = \pm 4. \end{cases}$$

Then

$$\widetilde{U}_e(z) = u(0) + u(2)(z + z^{-1}) + u(4)(z^2 + z^{-2})$$

and

$$\widetilde{U}_o(z) = u(1)(z + 1) + u(3)(z^2 + z^{-1}).$$

Now, $\widetilde{U}_e$ and $\widetilde{U}_e$ have to be relatively prime because $\det P(z) = 1$. Hence, we can apply the Euclidean algorithm for Laurent Polynomials in order to obtain a factorization. The algorithm goes as follows: first we set $a_0(z) = \widetilde{U}_e(z)$ and $b_0(z) = \widetilde{U}_o(z)$. Observe that $|a_0| > |b_0(z)|$. Next we divide to obtain $a_0(z) = b_0(z)q_0(z) + r_0(z)$, where $|r_0(z)| < |b_0(z)|$, and we set $a_1(z) = b_0(z)$ and $b_1(z) = r_0(z)$. We iterate the process and in the i-th step we obtain

$$a_{i+1}(z) = b_{i+1}(z)q_{i+1}(z) + r_{i+1}(z), \quad |r_{i+1}(z)| < |b_{i+1}(z)|.$$

The process ends at the step $n+1$, when we obtain $|r_n(z)| = 0$. Then, $a_n(z)$ will be the greatest common divisor of $a_0(z)$ and $b_0(z)$. Finally, we can write

$$\begin{pmatrix} \widetilde{U}_e(z) \\ \widetilde{U}_o(z) \end{pmatrix} = \prod_{i=0}^{n} \begin{pmatrix} q_i(z) & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} a_n(z) \\ 0 \end{pmatrix},$$

where $a_n(z)$ is a monomial due to the fact that $\widetilde{U}_e$ and $\widetilde{U}_e$ have to be relatively prime. After applying the algorithm to our filter, we get

$$\begin{aligned} a_0(z) &= u(4)z^2 + u(2)z + u(0) + u(2)z^{-2} + u(4)z^{-4}, \\ b_0(z) &= u(3)z^2 + u(1)z + u(1) + u(3)z^{-1}, \\ q_0(z) &= a(1 + z^{-1}), \\ r_0(z) &= w_1 z + w_0 + w_1, \end{aligned}$$

where

$$a = \frac{u(4)}{u(3)}, \quad w_0 = u(0) - 2au(1), \quad w_1 = u(2) - u(4) - au(1).$$

In the second iterate we have

$$\begin{aligned} a_1(z) &= u(3)z^2 + u(1)z + u(1) + u(3)z^{-1}, \\ b_1(z) &= w_1 z + w_0 + w_1, \\ q_1(z) &= b(1 + z), \\ r_1(z) &= d_0 z + d_0, \end{aligned}$$

where

$$b = \frac{u(3)}{w_1}, \quad d_0 = u(1) - u(3) - bw_0.$$

In the third iterate we get

$$\begin{aligned} a_2(z) &= w_1 z + w_0 + w_1, \\ b_2(z) &= d_0 z + d_0, \\ q_2(z) &= c(1 + z^{-1}), \\ r_2(z) &= w_0 - 2w_1, \end{aligned}$$

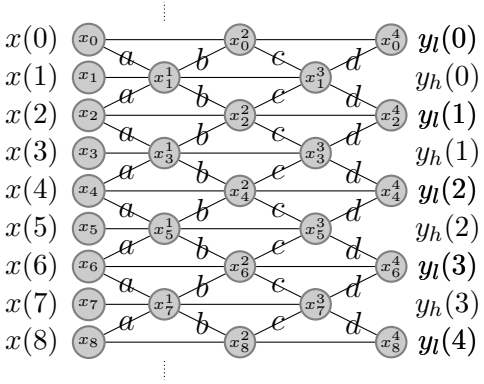

Figure 4.6: Bi-orthogonal 9-7 wavelet.

where

$$c = \frac{w_1}{d_0}.$$

And in the last step

$$\begin{aligned} a_3(z) &= d_0 z + d_0, \\ b_3(z) &= w_0 - 2w_1, \\ q_3(z) &= d(1+z), \\ r_3(z) &= 0, \end{aligned}$$

where

$$d = \frac{d_0}{K}, \quad K = w_0 - 2w_1.$$

Now, we can write

$$\widetilde{P}(z) = \begin{bmatrix} 1 & a(1+z^{-1}) \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \\ b(1+z) & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & c(1+z^{-1}) \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \\ d(1+z) & 1 \end{bmatrix} \cdot \begin{bmatrix} K & 0 \\ 0 & 1/K \end{bmatrix},$$

and therefore

$$H_1(z) = az^{-1} + a, \quad G_1(z) = bz^{-1} + b, \quad H_2(z) = cz^{-1} + c, \quad G_2(z) = dz^{-1} + d,$$

where $a, b, c, d \in \mathbb{R}$ are lifting parameters at each stage. Hence, using these filters we could obtain the complementary filter of $\widetilde{U}(z)$, $\widetilde{V}(z)$, and the pair of filters $\widetilde{S}(z)$ and $\widetilde{T}(z)$. Now we are going to show how to implement this transform via the lifting scheme. In order to simplify let us suppose first that our input vector has an infinite length $x = (\dots, x_0, x_1, \dots, x_8, \dots)$.

Figure 4.6 shows the scheme to implement the lifting. First we separate the input vector into two vectors, the first one containing the values for even indexes,

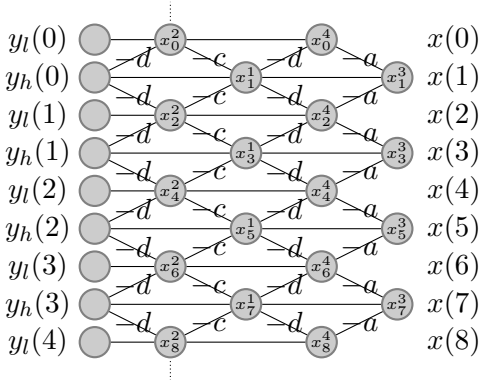

Figure 4.7: Inverse bi-orthogonal 9-7 wavelet.

$x(2n)$, and the second one with the values for the odd indexes, $x(2n+1)$, for all $n \in \mathbb{Z}$. At each lifting step we will update only one of this sequences, for instance at the first stage only the odd indexes vector is modified in order to obtain

$$x^1(2n+1) = x(2n+1) + a(x(2n) + x(2n+2)), \tag{4.2}$$

which corresponds to the filter $H_1(z)$. In the second stage we update the vector with even indexes using the output of the first stage

$$x^2(2n) = x(2n) + b(x^1(2n+1) + x^1(2n-1)), \tag{4.3}$$

where the superscripts stand for the number of the stage. For the third and fourth stages we obtain

$$\begin{aligned} x^3(2n+1) &= x^1(2n+1) + c(x^2(2n) + x^2(2n+2)), \\ x^4(2n) &= x^2(2n) + d(x^3(2n-1) + x^3(2n+1)). \end{aligned}$$

The coefficients $y_l(n) = x^4(2n)$ correspond to the lowpass filter and the coefficients $y_h(n) = x^3(2n+1)$ to the highpass filter. The parameters of the lifting stage wavelet are

$$a = -1.586, \quad b = -0.052, \quad c = 0.883, \quad d = 0.444.$$

So, we can write

$$y_l(n) = x^4(2n) = x^2(2n) + d(x^3(2n-1) + x^3(2n+1)) = x^2(2n) + d(y_h(n-1) + y_h(n)),$$

and we can find $x^2(2n)$ from the outputs

$$x^2(2n) = y_l(n) - d(y_h(n-1) + y_h(n)).$$

Therefore, we can inverse the lifting wavelet transform just using the scheme shown in Figure 4.7.

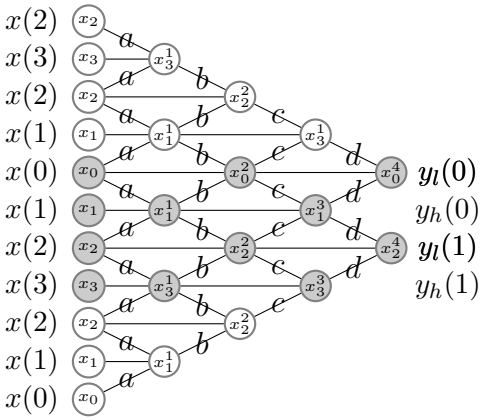

Figure 4.8: Symmetrical boundary extension for the bi-orthogonal 9-7 wavelet on 4 data points.

Now we will approach the boundary problem. Images are represented by finite matrices, so we have to deal with the boundary of the vectors in a special way. For instance, if the vector has infinite length, we have that

$$x^2(2n) = x(2n) + b(x^1(2n+1) + x^1(2n-1)),$$

and for $n = 0$,

$$x^2(0) = x(0) + b(x^1(1) + x^1(-1)),$$

with

$$x^1(-1) = x(-1) + a(x(-2) + x(0)).$$

But if the vector is finite in length, $x \in \ell^2(\mathbb{Z}_N)$, $x(n)$ is not defined for $n < 0$ or $n > N$. So we have to extend it in order to apply equations (4.2) and (4.3). Since the bi-orthogonal 9-7 wavelet is symmetrical and has an odd number of coefficients, we can apply a symmetrical extension to the data. Then, we define

$$x(-n) = x(n), \quad x(n+N) = x(N-n),$$

for $n = 1, \ldots, N$.

We can observe in Figure 4.8 how the symmetrical extension affects to the coefficients lying in the boundary, doubling the contribution of certain weights. For instance,

$$x^2(0) = x(0) + 2bx^1(1).$$

Hence, due to the symmetry we do not need to extend the data, we just only need to double the weights for the points that are in the boundary.

This method is computationally faster than the fast matrix multiplication using the FFT, which has $N \log N$ complexity, while from Figure 4.8 we see that the lifting scheme uses $\frac{N}{2}$ products in each column of the scheme and, since there are only 4 columns, the number of products is $4 \cdot \frac{N}{2} = 2N$, resulting in a linear complexity. The

Figure 4.9: 9-7 lifting wavelet transform for Lenna's gray image: transforms on columns (left) and final result (right).

lifting scheme also has the advantage of being easy to modify in order to construct an integer to integer map used in the lossless compression.

In order to apply this scheme to an image we proceed in the same fashion as in Chapter 2, we first apply the wavelet in the vertical direction and, after that, we apply it separately in the horizontal direction. This is called a 2D separable wavelet transform. Although it is possible to produce a 2D non-separable wavelet transform, its benefits are not worth comparing with its computational cost. Figure 4.9 shows the results of applying this wavelet to Lenna's gray image. We first transform the horizontal components to obtain a version of the image filtered in two subbands (left side of Figure 4.9), a horizontal low and a horizontal high frequencies subbands. Next we filter again the resulting image in the vertical direction to get four subbands (right side of Figure 4.9), the low horizontal and low vertical frequency subband (LL), the the low horizontal and high vertical frequency subband (LH), the high horizontal and low vertical frequency subband (HL) and the high horizontal and high vertical frequency subband (HH).

In Table 4.1 we can observe the computation times used by the bi-orthogonal 9-7 wavelet implemented via the lifting scheme and the Daubechies' D6 using the fast FFT to compute the convolutions for images of different sizes. Notice that for large images the improvement of the lifting scheme is considerable.

4.4 Quantization and partitioning

After applying the wavelet transform the coefficients obtained are quantized in a similar way as in the JPEG.

Height	Width	Time (DWT Bi-orthogonal 9-7)	Time (Daubechies D6)
128	128	0.410736	0.228973
256	256	0.411890	0.326355
480	400	0.446673	0.553753
512	512	0.528403	0.696883
1024	1024	0.844290	1.662423

Table 4.1: Comparison between the computation times (in seconds) of the lifting scheme and the fast matrix product for different images with different sizes.

Definition 4.4.1. Let $y \in \ell^2(\mathbb{Z}_N)$ be the vector of transformed coefficients. Then the quantized coefficients are given by

$$q(k) = \text{sign}(y(k)) \left\lfloor \frac{|y(k)|}{\delta} \right\rfloor, \text{ for all } k = 1, \ldots, N,$$

where $\lfloor x \rfloor$ represents the largest integer that is less than or equal to x and δ is a scale parameter called the step size of the quantizer.

Recall that in the JPEG algorithm the quantization operation introduces a compression in the image via the parameter δ because many coefficients become zero and they are neglected. In JPEG 2000, the parameter δ does not determine the compression ratio, instead it is given by the subsequent bitstream assembler. The default scale parameter in JPEG 2000 is $\delta = 1/128$. Hence, the main function of the quantization module is to map the float coefficients into integers in order to be efficiently processed by the entropy encoder.

After quantization the image is partitioned, the subbands HL, LH and HH are divided into non-overlapping rectangles of the same size. The rectangles from the same spatial region in each subband form a packet.

Definition 4.4.2. Let $q \in \ell^2(\mathbb{Z}_{M\times N})$ be the quantized coefficients after wavelet transform of an image at a certain stage $l \geq 1$ and let $m, n \in \mathbb{N}$ be integers such that m divides M and n divides N. Suppose that we partition each subband HL, LH and HH into $p = \frac{M}{m}\frac{N}{n}$ rectangles, R^j_{HL}, R^j_{LH} and R^j_{HH}, for $0 < j \leq p$, such that $R^j_{HL} \cap R^k_{HL} = \emptyset$, $R^j_{LH} \cap R^k_{LH} = \emptyset$ and $R^j_{HH} \cap R^k_{HH} = \emptyset$, for all $j \neq k$. Then the j-th packet is given by

$$P_j = R^j_{HL} \cup R^j_{LH} \cup R^j_{HH}, \quad \text{for all} \quad 0 < j \leq p.$$

Observe that each packet provides spatial locality information because it contains information needed for decoding the image at a certain spatial region at a certain resolution level.

Definition 4.4.3. Let $P = R_{HL} \cup R_{LH} \cup R_{HH}$ be a packet at a certain resolution level with $R_{HL}, R_{LH}, R_{HH} \in \ell^2(\mathbb{Z}_{M'\times N'})$ and let $m, n \in \mathbb{N}$ be integers such that m divides M' and n divides N'. Then we can partition the packet into rectangles of

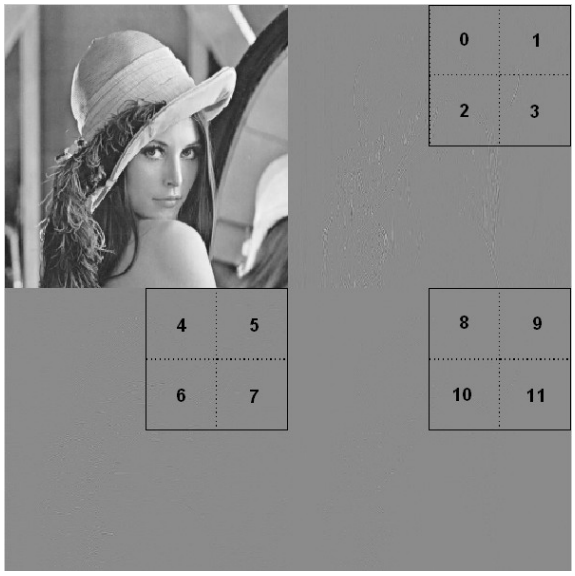

Figure 4.10: Lenna packet for the first resolution level and its 12 code-blocks.

equal size by dividing each rectangle R_{HL}, R_{LH}, R_{HH} into $p = \frac{M'}{m} \cdot \frac{N'}{n}$ rectangles of size $\frac{M'}{m} \times \frac{N'}{n}$, $C^j_{HL}, C^j_{LH}, C^j_{HH}$, for $0 < j \leq p$, such that

$$\begin{aligned} R_{HL} &= \cup_{j=1}^{p} C^j_{HL}, \quad C^j_{HL} \cap C^k_{HL} = \emptyset, \quad \text{for all } k \neq j, \quad j,k = 1,\dots p, \\ R_{LH} &= \cup_{j=1}^{p} C^j_{LH}, \quad C^j_{LH} \cap C^k_{LH} = \emptyset, \quad \text{for all } k \neq j, \quad j,k = 1,\dots p, \\ R_{HH} &= \cup_{j=1}^{p} C^j_{HH}, \quad C^j_{HH} \cap C^k_{HH} = \emptyset, \quad \text{for all } k \neq j, \quad j,k = 1,\dots p. \end{aligned}$$

Each one of these rectangles is called a code-block.

The code-blocks are the fundamental objects for the entropy encoder, in JPEG 2000 the standard size for the code-blocks is 64×64. In Figure 4.10 we can observe the wavelet transform for Lenna's image and one packet, draw with thick black line, of the first level resolution. We can see that the packet is formed by three rectangles, one in each subband HL, LH and HH. Each rectangle of the packet is further divided into four rectangles, the code-blocks of the packet, numbered from 0 to 11.

Observe that since the image has dimension 512×512, this packet contains the coefficients in the transform domain located in the zone $(0, 127) \times (255, 255)$ that corresponds to the spatial region given by the coordinates in the image $(0, 255) \times (511, 511)$.

4.5 Entropy coding

After obtaining all the code-blocks, each one is encoded separately. The coefficients of a code-block are transform into binary code. Hence, if we need $n + 1$ bits to represent the range of the quantized coefficients, then a quantized coefficient q will

be represented by a sequence of n bits $b = \{b_j\}_{j=0}^{n-1} \subset \{0,1\}^n$, where b_0 will be the most significant bit and b_{n-1} the least significant bit, indicating the magnitude or the modulus of the coefficient, plus a bit b_n for the sign of the coefficient.

10	4	−7	5	11	5	−3
−2	−1	6	−5	−4	3	−1
−5	4	−3	0	−2	−2	−3
3	1	−3	0	1	−3	−12
8	2	−1	−1	1	0	−14
−7	13	−6	0	−4	0	0
−6	8	−3	−2	−4	−1	0
−9	4	2	−6	−1	−5	3

Table 4.2: 8 coefficient array.

Table 4.2 shows the coefficients of an 8×8 code-block and in Table 4.3 we see the binary representation for the coefficients of the first column of the block.

w	b_0	b_1	b_2	b_3	b_4	b_5	b_6	Sign
10	0	0	0	1	0	1	0	+
−2	0	0	0	0	0	1	0	−
−5	0	0	0	0	1	0	1	−
3	0	0	0	0	0	1	1	+
8	0	0	0	1	0	0	0	+
−7	0	0	0	0	1	1	1	−
−6	0	0	0	0	1	1	0	−
−9	0	0	0	1	0	0	1	−

Table 4.3: Binary representation of coefficients.

Repeating the process with the coefficients of the other columns of the code-block we will obtain similar tables to Table 4.3, concatenating these tables with Table 4.3 in the third dimension we will get a cube. If we fix $p \in \{0, \ldots, n-1\}$ and we take in the cube a plane containing all the bits with significance level p for all the coefficients, then we will have constructed a bit-plane of significance level p.

Definition 4.5.1. Given a quantized coefficient vector $q \in \ell^2(\mathbb{Z}_{M\times N})$ we will represent the sign of the coefficients as $\mathcal{X} = \operatorname{sign}(q)$ and their magnitudes as $v = |q|$, i.e., $\mathcal{X}(i,j) = \operatorname{sign}(q(i,j))$ and $v(i,j) = |q(i,j)|$, for all $0 \le i < M$ and $0 \le j < N$.

Definition 4.5.2. Given a quantized coefficient vector $q \in \ell^2(\mathbb{Z}_{M\times N})$ with magnitude $v \in \ell^2(\mathbb{Z}_{M\times N})$, we will denote by

$$v^p = \left\lfloor \frac{v}{2^p} \right\rfloor \in \ell^2(\mathbb{Z}_{M\times N})$$

the vector of values formed by dropping p least significance bits from each component of the vector $v = v^0$, and v_p will denote the vector of least significant bits of v^p,

i.e., $v_p(i,j) \in \{0,1\}$ is the least significant bit of $v^p(i,j)$, for all $0 \leq i < M$ and $0 \leq j < N$.

Definition 4.5.3. Let $q \in \ell^2(\mathbb{Z}_{M\times N})$ be the coefficients of a code-block of size $M \times N$ and $v^p \in \ell^2(\mathbb{Z}_{M\times N})$ as defined above. The bit-plane of significance level p for this code-block is given by the set

$$S^p = \{v_p(i,j) \in \{0,1\}, \quad 0 \leq i \leq M-1, \quad 0 \leq j \leq N-1\}.$$

Definition 4.5.4. Let $b_p \in \{0,1\}$ be the p least significant bit of a coefficient $w \in \mathbb{R}$ to be encoded. If $b_j = 0$, for all $0 \leq j < p$, we say that the coefficient w is insignificant (if the bitstream is terminated at this point or before, then the coefficient will be reconstructed to zero) and we say that the bit b_p will be encoded in mode of significance identification. Otherwise, the coefficient w is said to be significant and the bit b_p is encoded in the mode of refinement.

The distinction of modes for encoding is useful for the entropy encoder. Recall that entropy encoders work with distributions of probability and observe that the bits laying in the mode of significance identification have greater probability to be zero than to be one. On the other hand, the bits on the refinement mode will take the values 0 and 1 with the same probability. We can observe in Table 4.3 the boundary, vertical line, that determines if a bit will belong to the significance identification mode or to the refinement mode. For instance, for the coefficient $w = 10$ the significance mode bits are the set $\{b_0, b_1, b_2, b_3\}$ and the refinement mode is formed by the bits $\{b_4, b_5, b_6\}$. Hence, the quantized coefficients of the code-block are encoded one bit at a time starting with the most significant bit and proceeding to the least significant bit. In this way, we will encode the bits of the same significance level for all the coefficients of the code-block at the same time, i.e., we process all the bit-plane corresponding to the significance level of the bits. When the coefficient we are encoding becomes significant, i.e. we get the first non-zero bit, the sign must be encoded. As it happens with the DCT transform, the Discrete Wavelet Transform packs most of the energy in the LL subband, hence it is reasonable to expect the coefficients of the other subbands to be small and, therefore, there will be many zeroes in the significance identification mode when we encode these coefficients. So, the earlier bit-planes will be formed by many zeroes, and therefore they will contain little information. In order to exploit this redundancy in the bit-planes, the JPEG 2000 uses an efficient coding called context-based adaptive binary arithmetic coding.

4.5.1 Context labeling

It turns out that the probability distribution of a bit in a quantized coefficient is not independent from all the previous bits in the same coefficient as well as the value of its immediate neighbors. In order to exploit this redundancy, JPEG 2000 uses a context to estimate the probability of a bit in a coefficient. This context is constructed from the current significance of the bit as well as the significance state of its vertical, horizontal and diagonal neighbors.

Definition 4.5.5. Let $q(i,j) \in \mathbb{R}$, $0 \leq i < M$ and $0 \leq j < N$, be a quantized coefficient and let $v^p(i,j) \in \mathbb{R}$ be its value after dropping the p least significant bits. Then, the significance of the coefficient is given by

$$\sigma^p(i,j) = \begin{cases} 1, & \text{if} \quad v^p(i,j) > 0, \\ 0, & \text{if} \quad v^p(i,j) = 0, \end{cases}$$

for all $0 \leq i < M$ and $0 \leq j < N$.

In order to simplify we will use the notion of binary significance state $\sigma(i,j)$ that will assume the value of $\sigma^p(i,j)$, where p is the most recent least significant bit for which information of the sample $q(i,j)$ has been coded up to that point. At the beginning, we will set $\sigma(i,j) = 0$ for all coefficients in the code-block and we will change the state to $\sigma(i,j) = 1$ after finding the first bit different from zero in the coefficient. Immediately after that we will code the sign of the coefficient. Hence we will identify the modes of coding by the value of $\sigma(i,j)$, if $\sigma(i,j) = 0$ we will code $v_p(i,j)$ in the significance mode, the sign will be coded in the sign mode and when $\sigma(i,j) = 1$, $v_p(i,j)$ will be coded in the refinement mode.

Definition 4.5.6. Let $q(i,j) \in \mathbb{R}$ be a quantized coefficient of a code-block and let $\sigma(i,j)$ be its binary significance state. We define the quantities

$$\begin{aligned} \kappa^h(i,j) &= \sigma(i,j-1) + \sigma(i,j+1), \\ \kappa^v(i,j) &= \sigma(i-1,j) + \sigma(i+1,j), \\ \kappa^d(i,j) &= \sum_{k_1=\pm 1} \sum_{k_2=\pm 1} \sigma(i+k_1, j+k_2), \end{aligned}$$

as the number of horizontal, vertical and diagonal coefficient neighbors of the coefficient, respectively, that have a binary significance state equal 1.

The coefficients which lie beyond the boundaries of the code-block are considered as insignificant for the purpose of constructing these quantities.

Definition 4.5.7. Let $v_p(i,j) \in \{0,1\}$ be the p least significant bit in a quantized coefficient of a code-block of one of the subbands LL, HL, LH or HH, belonging to the significance mode. Let κ^h, κ^v and κ^d be its number of neighbors in the horizontal, vertical, and diagonal directions, respectively, defined above. Then, the context label $\kappa^{sig}(i,j) \in \{0,1,\ldots,8\}$ is given by the values in Table 4.4.

Although the values in Table 4.4 come from empirically studies and the need for simple and efficient implementations using both software and hardware, we can present a qualitative interpretation for them.

In Figure 4.11 we can observe a test pattern image (left) and its bi-orthogonal 9-7 wavelet transform (left), with its subbands LL (top, left), HL (top, right), LH (bottom, left) and HH (bottom, right). If we are coding a coefficient in the LH (vertically high-pass) subband, then the rapid variations in the horizontal direction are killed

LH and LL subband				HL subband				HH subband		
κ^h	κ^v	κ^d	κ^{sig}	κ^h	κ^v	κ^d	κ^{sig}	$\kappa^h + \kappa^v$	κ^d	κ^{sig}
2	x	x	8	x	2	x	8	≥ 3	x	8
1	≥ 1	x	7	≥ 1	1	x	7	2	≥ 1	7
1	0	≥ 1	6	0	1	≥ 1	6	2	0	6
1	0	0	5	0	1	0	5	1	≥ 2	5
1	2	x	4	2	0	x	4	1	1	4
0	1	x	3	1	0	x	3	1	0	3
0	0	≥ 2	2	0	0	≥ 2	2	0	≥ 2	2
0	0	1	1	0	0	1	1	0	1	1
0	0	0	0	0	0	0	0	0	0	0

Table 4.4: Context for significance identification encoding.

by the filter, as we can observe in Figure 4.11 (right), and the significant coefficients will mostly derive from the horizontally oriented features in the image. Accordingly, horizontal neighbors are considered most indicative of the current coefficient significance. After the horizontal neighbors, the vertical neighbors are considered the most important indicators of significance and the diagonal neighbors are considered if at most one of the four diagonal neighbors is already significant. If the code-block belongs to the HL subband the argument is the same interchanging the roles of the horizontal and vertical neighbors. For a code-block in the HH subband we expect to encounter diagonal features emphasized, as we can observe in Figure 4.11 (right), and therefore we consider more relevant the significances of the diagonal neighbors and in a second plane the sum of significances in vertical and horizontal neighbors.

Notice that the significance information of the eight neighbors used to compute the context label depends on the order of codification of the bits. It may well happen that if we are coding the bit $v_p(i,j)$ of a coefficient, four of the neighbors have been coded until the $p+1$ least significant coefficient and we have no information about the p-th least bits for these coefficients. Hence, we must use the binary significance at the bit-plane $p+1$ for these neighbors.

We also must point out that sometimes a run length mode is also used to encode multiple insignificant coefficients using just a single binary symbol, see [17].

After the coefficient becomes significant, the sign is coded used a context since it turns out that the signs of neighbor coefficients present statistical redundancy. JPEG2000 employs 5 different contexts for the sign.

Definition 4.5.8. Given a coefficient $q(i,j) \in \mathbb{R}$ in a code-block of size $M \times N$ with sign $\mathcal{X}(i,j)$, for all $0 \leq i < M$ and $0 \leq j < N$, we define the quantities

$$\mathcal{X}^h(i,j) = \mathcal{X}(i,j-1)\sigma(i,j-1) + \mathcal{X}(i,j+1)\sigma(i,j+1),$$
$$\mathcal{X}^v(i,j) = \mathcal{X}(i-1,j)\sigma(i-1,j) + \mathcal{X}(i+1,j)\sigma(i+1,j),$$

representing the net sign bias of the horizontal and vertical neighbors, respectively.

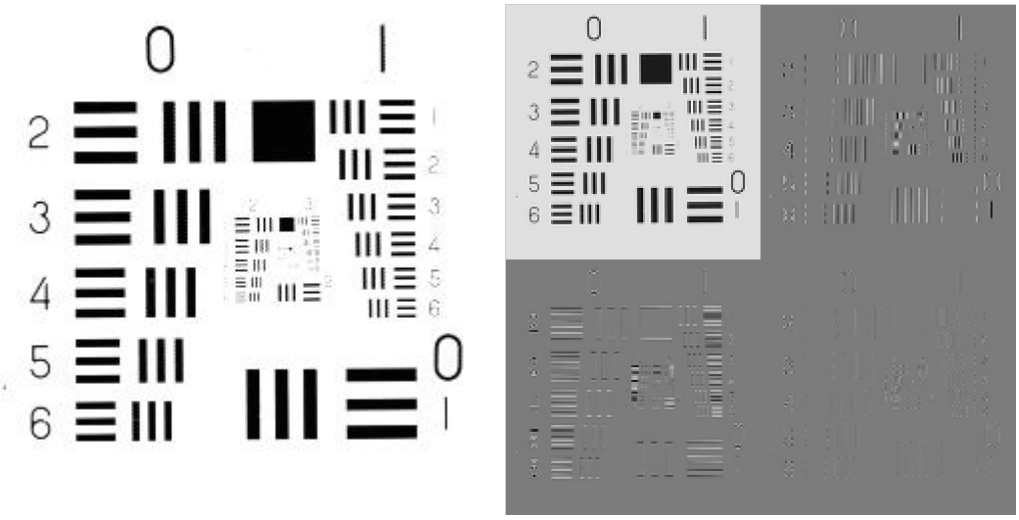

Figure 4.11: Test pattern image (left) and its 9-7 bi-orthogonal wavelet transform (right).

These quantities take values between -2 and 2, resulting in 25 neighbor configurations. But it is reasonable to assume that the conditional distribution of $\mathcal{X}(i,j)$ given a determined neighborhood is identical to the conditional distribution of $-\mathcal{X}(i,j)$ given a determined neighborhood in which the signs of all neighbors are flipped. In this way, the 25 different configurations become 13 different configurations, which can be reduced to 5 by truncating the horizontal and vertical biases to the range -1 through 1.

Definition 4.5.9. Given a coefficient $q(i,j) \in \mathbb{R}$ in a code-block of size $M \times N$ with sign $\mathcal{X}(i,j)$, for all $0 \leq i < M$ and $0 \leq j < N$, we define the quantities

$$\overline{\mathcal{X}}^h(i,j) = \text{sign}(\mathcal{X}^h(i,j)) \min(1, |\mathcal{X}^h(i,j)|),$$
$$\overline{\mathcal{X}}^v(i,j) = \text{sign}(\mathcal{X}^v(i,j)) \min(1, |\mathcal{X}^v(i,j)|),$$

representing the truncated sign bias of the horizontal and vertical neighbors, respectively.

Definition 4.5.10. Given a quantized coefficient $q(i,j)$ in a code-block, its sign context label, $\kappa^{\text{sign}} \in \{10, 11, 12, 13, 14\}$, and the sign-flipping factor, $\mathcal{X}^{\text{flip}} \in \{-1, 1\}$, are given in Table 4.5. The single binary symbol coded with respect to context κ^{sign} is given by

$$s = \begin{cases} 0, & \text{if} \quad \mathcal{X}(i,j)\mathcal{X}^{\text{flip}} = 1, \\ 1, & \text{if} \quad \mathcal{X}(i,j)\mathcal{X}^{\text{flip}} = -1. \end{cases}$$

Finally, we attempt to explain the codification in the refinement mode. This mode is applied to code the bit $v_p(i,j)$ when the coefficient has become significant, i.e., $\sigma^{p+1}(i,j) = 1$. Here we refine the coarser approximation $q^{p+1}(i,j)$ to a finer one

$\overline{\mathcal{X}}^h$	$\overline{\mathcal{X}}^v$	κ^{sign}	$\mathcal{X}^{\text{flip}}$
1	1	14	1
1	0	13	1
1	-1	12	1
0	1	11	1
0	0	10	1
0	-1	11	-1
-1	1	12	-1
-1	0	13	-1
-1	-1	14	-1

Table 4.5: Context for sign mode encoding.

$q^p(i,j)$. It turns out that the conditional probability distribution of the bits in the refinement mode given a a coarser representation of the coefficient, $f_{V_p|Q^{p+1}}(v_p|q^{p+1})$, is independent of the sign of the coefficient, $f_{V_p|Q^{p+1}}(v_p 1|q^{p+1}) > 1/2$ for all q^{p+1}, and $f_{V_p|Q^{p+1}}(v_p|q^{p+1}) \approx 1/2$, for large $|q^{p+1}|$. Therefore, it is convenient to condition the coding of $v_p(i,j)$ upon the value of $v^{p+1}(i,j)$ when $v^{p+1}(i,j)$ is small.

Definition 4.5.11. Given an approximation $v^p(i,j)$ of a quantized coefficient $q(i,j)$ in a code-block. If $\sigma(i,j) = \sigma^p(i,j)$, we define the delayed significance state $\overleftarrow{\sigma}(i,j) = \sigma^{p+1}(i,j)$.

When the coefficient becomes significant, $\sigma(i,j)$ is changed to 1, but $\overleftarrow{\sigma}(i,j)$ remains 0 until the first magnitude refinement bit has been coded. After that, $\overleftarrow{\sigma}(i,j)$ is toggled to 1. In fact, $\overleftarrow{\sigma}(i,j) = 1$ is an indicator that informs us that $v^{p+1}(i,j) \geq 2$, we already know that $v^{p+1}(i,j) > 0$, because we are in the refinement mode for $v_p(i,j)$.

Definition 4.5.12. Given a the p-th least significant bit $v_p(i,j)$ in a quantized coefficient $q(i,j)$ in a code-block, which belongs to the refinement mode, its refinement context label, $\kappa^{\text{mag}} \in \{15, 16, 17\}$ is given in Table 4.6.

$\overleftarrow{\sigma}(i,j)$	κ^{sign}	κ^{mag}
0	0	15
0	> 0	16
1	x	17

Table 4.6: Context for refinement mode encoding.

Remark 4.5.13. The 18 different contexts described above define 18 probability models that are few enough to be maintained in the high speed registers of a machine for a hardware implementation. These are the standard JPEG2000 contexts, but the algorithm permits several mode variations in order to adapt to different implementation requirements as parallelism of high bit-rates (see [17]).

4.5.2 Arithmetic coding and the Elias coder

After assigning a context label to each bit we obtain a bit vector accompanied by a context label vector indicating that the bits within the same context are independent and identically distributed. The entropy coder has the task to convert this bit-context pair sequence in a bit-stream with a length as close to the Shannon's limit as possible. There are a great variety of coders to perform this task and the selected for the JPEG2000 standard is the MQ-coder. In order to explain its operation, we start by the Elias coder. Although this coder is not practical as it needs infinite arithmetic precision, it is very useful in order to demonstrate how easily it approximates to the entropy rate of a random source.

Let $y \in \ell^2(\mathbb{Z}_N)$ be a random vector, with $y(n) \in \{0,1\}$, for all $n = 1, \ldots, N$. Suppose that the probabilities $p_n = P(y(n) = 0)$ are known for all $n = 1, \ldots, N$. This last assumption is, in general, false and here the context labeling explained above plays its role.

Definition 4.5.14. We define the n-th interval of probability $[c_n, c_n + a_n) \subseteq [0,1)$ associated to the bit sequence $y \in \ell^2(\mathbb{Z}_N)$ as $c_0 = 0$ and $a_0 = 1$, if $n = 0$, and for $n > 0$

$$c_n = \begin{cases} c_{n-1}, & \text{if } y(n) = 0, \\ c_{n-1} + a_{n-1} \cdot (1 - p_n), & \text{if } y(n) = 1, \end{cases}$$

$$a_n = \begin{cases} a_{n-1}(1 - p_n), & \text{if } y(n) = 0, \\ a_{n-1} p_n, & \text{if } y(n) = 1. \end{cases}$$

This definition shows how, for each bit $y(n)$ in the sequence, the previous probability interval $[c_{n-1}, c_{n-1} + a_{n-1})$ is divided into two disjoint intervals $[c_{n-1}, c_{n-1} + a_{n-1} \cdot (1 - p_n))$ and $[c_{n-1} + a_{n-1} \cdot (1 - p_n), c_{n-1} + a_{n-1})$, choosing the first one, if $y(n) = 0$, and the second one whenever $y(n) = 1$.

Proposition 4.5.15. *Let $y_1 \in \ell^2(\mathbb{Z}_N)$ and $y_2 \in \ell^2(\mathbb{Z}_N)$ be two different sequences of bits of the same length N. Let $[c_n, c_n + a_n)$ and $[c'_n, c'_n + a'_n)$, for all $n = 1, \ldots, N$, be their n-th probability interval, respectively. Then*

1. *$[c_n, c_n + a_n) \cap [c'_n, c'_n + a'_n) = \emptyset$, for all $n = 1, \ldots, N$,*
2. *$\cup_{n=1}^{N} [c_n, c_n + a_n) = \cup_{n=1}^{N} [c'_n, c'_n + a'_n) = [0,1)$,*
3. *$a_n = P(y(1), \ldots, y(n)) = \prod_{i=1}^{n} (p_i \mathbb{1}_{\{y(i)=0\}} + (1 - p_i)\mathbb{1}_{\{y(i)=1\}})$.*

From the first claim of the proposition we see that any vector $y \in \ell^2(\mathbb{Z}_N)$ of bits can be uniquely identified by any number in the interval $[c_N, c_N + a_N)$. From the third part we see that the length of the n-th probability interval gives the probability of the subsequence $y(1), \ldots, y(n)$. Let us see how it woks with an example.

Example 4.5.16. Consider the sequence $y = \{0, 1, 1, 0, 1, \ldots\}$ with the same probability for each bit $p_n = P(y(n) = 0) = \frac{1}{4}$, for all n. We start with the interval $[0,1)$ and, since $p_1 = 1/4$, we divide it into two intervals $[0,1) = [0, 1/4) \cup [1/4, 1)$.

Since $y(1) = 0$, we select the first subinterval, $[c_1, c_1 + a_1) = [0, 1/4)$. Hence, $c_1 = c_0 = 0$ and $a_1 = a_0 \cdot p_1 = 1 \cdot 1/4 = 1/4$. Now, we again divide the actual interval $[0, 1/4) = [0, 1/4^2) \cup [1/4^2, 1/4)$, choosing the second subinterval because $y(2) = 1$. Hence, $[c_2, c_2 + a_2) = [1/4^2, 1/4)$, obtaining $a_2 = 1/4 - 1/4^2 = 3/4^2$. Observe that $a_2 = P(y(1) = 0, y(1) = 1) = \frac{1}{4} \cdot \frac{3}{4} = \frac{3}{4^2}$. For the next bit we get $[1/4^2, 1/4) = [1/4^2, 7/4^3) \cup [7/4^3, 1/4)$, and since $y(3) = 1$ we chose $[c_3, c_3 + a_3) = [7/4^3, 1/4)$, where $a_3 = 1/4 \cdot 3/4 \cdot 3/4 = 9/4^3$. Finally, $[7/4^3, 1/4) = [7/4^3, 37/4^4) \cup [37/4^4, 1/4)$, and since $y(4) = 0$, we select $[c_4, c_4 + a_4) = [7/4^3, 37/4^4)$, with $a_4 = 1/4 \cdot 3/4 \cdot 3/4 \cdot 1/4 = 9/4^4$.

Suppose that we apply this coding to a sequence of N bits. Then, as we said before, the vector $y \in \ell^2(\mathbb{Z}_N)$ can be recovered from any number in the interval $[c_N, c_N + a_N)$. Since the interval has length a_N, it must contain at least one L_N bit fraction of the form

$$0.\underbrace{bbbbbbbb}_{L_N},$$

where L_N is any integer such that $2^{-L_N} < a_N$. Therefore, the number of bits needed to represent the vector is

$$L_N \approx -\log_2(a_N),$$

which is very close to the entropy of the source associated to the vector. Recall that a_N is the probability of occurrence of the sequence $y = (y(1), \ldots, y(N))$, $a_N = P(y)$, and its entropy is given by

$$H = -\sum_{(y(1),\ldots,y(N))} P\{(y(1), \ldots, y(N)\} \log_2(P\{(y(1), \ldots, y(N)\}).$$

There is a problem in the method explained above since we do not know a priori the length L_N needed to encode the sequence. Thus we need to provide a mechanism to signal this length and take into account the number of bits used by this method for the final bit count in compression. Usually a large number N of bits is coded in order to neglect the signaling cost. Nevertheless there is a termination policy in Elias coding that avoids the need for signaling. If N is the number of bits of the sequence we can code it with

$$L_N = \lceil -\log_2(a_N) \rceil + 1,$$

where $\lceil x \rceil$ indicates the smallest integer which is larger than x. Thus, we have

$$2^{-L_N} \leq \frac{1}{2} a_N.$$

Let $\hat{c}_N$ be the quantity formed by taking only the first L_N bits of c_N and adding 1 to the least significant bit position

$$\hat{c}_N = 2^{-L_N} \left\lfloor 2^{L_N} c_N + 1 \right\rfloor > c_N.$$

Then observe that $\hat{c}_N + 2^{-L_N} \leq c_N + 2 \cdot 2^{-L_N} \leq c_N + a_N$. Thus, if a decoder receives a sequence of bits which agrees with $\hat{c}_N$ in the first L_N bit positions, then processing this sequence as a binary fraction with value r we obtain

$$c_N < \hat{c}_N \leq r < \hat{c}_N + 2^{-L_N} \leq c_N + a_N.$$

So, since $r \in [c_N, c_N + a_N)$, it uniquely identifies the original sequence of N bits $y \in \ell^2(\mathbb{Z}_N)$.

Suppose now that we encode sequences of N bits to form a bit-stream concatenating several of these sequences. Let L_N^k represent the length of the k-th coded sequence, for $k = 0, 1, \ldots$ Then the decoder receives a fraction bit sequence $r^{(0)}$ of the form

$$r^{(0)} = 0.\underbrace{bbbbbbbb}_{L_N^0}\underbrace{cccccccc}_{L_N^1}\ldots$$

The decoder first estimates the probability interval $[c_N^0, c_N^0 + a_N^0)$ and determines the length L_N^0, after that it computes the new bit fraction sequence, $r^{(1)}$, by eliminating the first L_N^0 bits, i.e.,

$$r^{(1)} = 0.\underbrace{bbbbbbbb}_{L_N^1}\underbrace{cccccccc}_{L_N^2}\ldots,$$

and it finds the next probability interval $[c_N^1, c_N^1 + a_N^1)$ and the length L_N^1, and so on. In this way, the lengths L_N^k do not need to be transmitted and the average bit-rate is

$$\frac{1}{N}E[L_N] = \frac{1}{N}E[\lceil -\log_2(a_N) + 1\rceil] = H(y) + \frac{3}{2m}.$$

We see that as N goes to infinity the bit-rate approaches to the ideal limit.

It turns out that the Elias coding is impractical because it involves arithmetic operations whose precision is comparable to the number of code bits. Nevertheless, applying several modifications we can derive an algorithm which approaches to the ideal bit-rate limit and involves fixed, finite arithmetic precision for arbitrarily large values of N. On the other hand, the Elias code has the important property of being "incrementally decodable", i.e., given $r \in [c_N, c_N + a_N)$ we can decode the prefixes $(y(0), \ldots, y(n))$, for $n = 1, 2, \ldots, N$, one by one because $r \in [c_n, c_n + a_n)$. This leads to a recursive algorithm for incrementally decoding the source outputs.

4.5.3 Arithmetic coding and the MQ-coder

Although the Elias coder requires infinite arithmetic precision operations and it cannot be practically implemented, it can be modified to obtain a finite arithmetic precision coder. This requires to make a choice about the fixed-precision representation of the interval and devise a renormalization rule for the interval in order to maintain it within the bounds allowed by the fixed-precision representation (see [13]). We observe that the amplitude interval A becomes very small at each iteration and it allows us to normalize the coding interval parameters C and A in the following way

$$\begin{cases} C = 1.5 \cdot [0.k_1k_2 \dots k_L] + 2^{-L} \cdot 1.5 \cdot C_x, \\ A = 2^{-L} \cdot 1.5 \cdot A_x, \end{cases}$$

where L is a normalization factor which determines the magnitude of the interval A, C_x and A_x are fixed-point integers representing magnitudes in $(0.0, 1.5)$ and $(0.75, 1.5)$, respectively. The bit sequence $k_1k_2 \dots k_L$ are the output bits that are already determined. In this way it is possible to use a fixed-point arithmetic and normalization operations for the probability interval subdivision task. Since the value of A_x is close to 1.0, we can approximate the value $A \cdot P_i$ with P_i and then the interval sub-division operation in Definition 4.5.14 can be reduced to

$$\begin{cases} c_x(n) = c_x(n-1), \quad a_x(n) = a_x(n-1) - p(n), & \text{if } y(n) = 0, \\ c_x(n) = c_x(n-1) + a_x(n-1) - p(n), \quad a_x(n) = p(n), & \text{if } y(n) = 1. \end{cases}$$

These operations can be computed quickly without any multiplication. This method requires that the interval length A and the code string (pointer to the interval to decode the sequence) are periodically normalized in order to maintain A within its range $(0.75, 1.5)$. Since we are making approximations the compression performance may be affected a little bit because of the use of the fixed-point integer coding interval instead of the real one, and the change of $a_i \cdot p_i$ by p_i. However, experimental studies show that the degradation in compression performance is less than 3% and it is well worth the saving in computational complexity (see [11]).

4.5.4 Probability estimation

As we have seen the arithmetic coder uses the probability of zero (or one) of each bit. So, we need to estimate these probabilities because we do not know the statistics of the source (image). In context technique we suppose that each the symbols in the same context are independent and identically distributed (i.i.d.), so we can estimate the probability $p(i)$ of the bit $y(i)$ to be one from the number of bits, n, in the same context of the bit $y(i)$ and the number of bits with value one, n_1, in this context, using the Bayesian rule

$$p(i) = \frac{n_1 + 1}{n + 2}.$$

The MQ-coder uses state transition machines to estimate the probability of the context more efficiently by taking into consideration the non-stationary characteristics of the symbol sequence.

4.5.5 Coding order: Sub-bitplane entropy coder

In JPEG2000 the embedded bit-stream may be truncated at any point, thus the coding order of the bit-planes and inside each bit-plane is very important in order

to achieve little distortion. A non-optimal coding order leads to miss important information and produces great levels of distortion at the moment of truncating the bit-stream at some critical points. It turns out [12] that the optimal encoding order is to encode those bits with the steepest rate-distortion slope, i.e., the largest coding distortion decrease per bit spent.

Suppose that we have n bit-planes and that we are processing the i-th most significant one. Let us consider a bit $b_i \in \{0, 1\}$ in this bit-plane in the refinement mode. Then, supposing that the bit takes the value 1 with probability 1/2, before coding the bit the uncertainty interval of the coefficient will be $[c, c+2^{n-i})$ and after the the codification the coefficient will lie either in $[c, c + 2^{n-i-1})$ or $[c + 2^{n-i-1}, c + 2^{n-i-1})$. Since the bit has the same probability to be 0 than to be 1, the entropy of the refinement bit will be

$$R_{\text{ref}} = 1 \text{ bit.}$$

Now, if we assume that the value of the coefficient is uniformly distributed in the intervals, then the expected distortion before and after coding will be given by

$$D_{\text{ref,before}} = \frac{1}{2^{n-i}} \int_c^{c+2^{n-i}} (x - c - 2^{n-i-1})^2 dx = \frac{4^{n-i}}{12},$$
$$D_{\text{ref,after}} = \frac{1}{2^{n-i-1}} \int_c^{c+2^{n-i-1}} (x - c - 2^{n-i-2})^2 dx = \frac{4^{n-i-1}}{12},$$

and the slope for the rate-distortion curve will be given by

$$s_{\text{ref}} = \frac{D_{\text{ref,before}} - D_{\text{ref,after}}}{R_{\text{ref}}} = \frac{4^{n-i}}{12} - \frac{4^{n-i-1}}{12} = 4^{n-i-2}.$$

If the bit b_i in the i-th bit-plane is in the significance mode, before the coding of the bit the uncertainty interval for the coefficient will be $[-2^{n-i}, 2^{n-i})$ and after codification, if the bit becomes significant, the coefficient will lie either on $[-2^{n-i}, -2^{n-i-1})$ or $[2^{n-i-1}, 2^{n-i})$, depending on its sign. If the bit continues being insignificant, then after codification the coefficient will lie on the interval $[-2^{n-i-1}, 2^{n-i-1})$ and it will not affect to the distortion because the coefficient will still be zero. If we assume that the probability that the coefficient becomes significant is p, then the average number of bits to encode the significant identification can calculated as

$$R_{\text{sig}} = -p \log_2 p - (1 - p) \log_2(1 - p) + p \cdot 1 = H(p) + p,$$

where $H(p) = -p \log_2 p - (1 - p) \log_2(1 - p)$ and the last term p is due to the codification of the sign bit when the coefficient becomes significant. Assuming again a uniform distribution for the coefficient in the uncertainty intervals we can compute the distortion before and after encoding the bit that becomes significant as

$$D_{\text{sig,before}} - D_{\text{sig,after}} = p \cdot 2.25 \cdot 4^{n-i},$$

and the rate-distortion slope is

$$s_{\text{sig}} = \frac{D_{\text{sig,before}} - D_{\text{sig,after}}}{R_{\text{sig}}} = \frac{p \cdot 2.25 \cdot 4^{n-i}}{H(p) + p} = \frac{9 \cdot 4^{n-i-1}}{1 + H(p)/p}.$$

As it was expected we have that $s_{\text{sig}} > s_{\text{ref}}$ and therefore the conclusion is that we must encode the significant bits in the bit-plane before the refinement bits of the same bit-plane.

Remark 4.5.17. Within the same coding category (significance identification/ refinement), one more significance bit-plane translates into 4 times more contribution in distortion decrease per coding bit spent. Therefore, the code-block should be coded bit-plane by bit-plane, starting by the most significant ones.

Remark 4.5.18. Within the same bit-plane, we should encode the significance identification bits with a higher probability to become significant.

Remark 4.5.19. Within the same bit-plane, the significance identification bits with a probability of becoming significant higher than 0.01 should be encoded earlier than the refinement bits. It has been observed that the insignificant coefficients with no significant coefficients in its neighborhood have a probability of becoming significant below 0.01, while insignificant coefficients with at least one significant coefficient neighbor usually have greater significance probabilities.

As a consequence, the JPEG 2000 entropy coder encodes a code-block bit-plane by bit-plane from the most significant bit-plane to the least significant one. Within each bit-plane the bit array is subdivided into three sub-bitplanes: the predicted significance (PS), the refinement (REF) , and the predicted insignificance (PN). The next example illustrates the block-coding process in JPEG 2000.

Example 4.5.20. Suppose that we have the coefficients shown in Table 4.7. The coder starts by the most significant bit-plane b_1. At first all the coefficients are insignificant and they belong to the PN sub-bitplane. Whenever a bit '1' is found the sign of the coefficient is immediately encoded after the coding of the bit. The information about the already coded bits and the sign of the significant coefficients allow us to determine an interval within each coefficient lies and we can approximate the value of these coefficients by the middle value of their respective intervals. In Table 4.8 we can observe how we have coded the most significant bit-plane b_1 and the sign of the coefficient w_1, we can also find the range within the coefficient lies and its approximated value at this moment. As we proceed with the coding the uncertainty ranges shrink and the reconstruction values of the coefficients become more precise.

After coding the most significant bit-plane we proceed with the PS sub-bitplane of the second most significant bit-plane, b_2. The PS sub-bitplane consists on the bits of the coefficients that are not significant but has at least one significant neighbor. In our case these bits will correspond to the coefficients w_0 and w_2. Since the bit of the coefficient $w_0 = 45$ has value '1' we proceed to encode its sign. Figure 4.9 shows the results with the range for each coefficient and its predicted value.

In the next step the refinement (REF) sub-bitplane of the bit-plane b_2 is encoded, so we code the bits of the coefficients that are already significant; in our example the unique coefficient of this characteristics is w_1, so we code the bit of w_1 in the bit-plane b_2. We can observe the results in Table 4.10.

w	b_1	b_2	b_3	b_4	b_5	b_6	b_7	Sign
$45(w_0)$	0	1	0	1	1	0	1	+
$-74(w_1)$	1	0	0	1	0	1	0	−
$21(w_2)$	0	0	1	0	1	0	1	+
$14(w_3)$	0	0	0	1	1	1	0	+
$-4(w_4)$	0	0	0	0	1	0	0	−
$-18(w_5)$	0	0	1	0	0	1	0	−
$4(w_6)$	0	0	0	0	1	0	0	+
$-1(w_7)$	0	0	0	0	0	0	1	−

Table 4.7: Binary representation of coefficients.

w	b_1	b_2	b_3	b_4	b_5	b_6	b_7	Sign	Range	Value
45	0								[-64,63)	0
−74	1							−	[-128,-64)	-96
21	0								[-64,63)	0
14	0								[-64,63)	0
−4	0								[-64,63)	0
−18	0								[-64,63)	0
4	0								[-64,63)	0
−1	0								[-64,63)	0

Table 4.8: Bit plane b_1 coding.

To finish with the bit-plane b_2 the PN sub-bitplane is processed. The rest of bits from the bit-plane b_2, bits of the coefficients which are not significant, are coded and have no significant neighbors. The sign is also coded when a bit with value '1' is found. In Table 4.11 we can observe the results; now all bits in b_2 are encoded and their ranges have changed.

To finish the process we repeat the previous steps in the next bit-planes. We start coding ordered the PS, REF and PN sub-bitplanes for each bit-plane and coding the sign of the coefficients when we find a '1' for the first time in a coefficient, which will become significant. The entropy coder continues until all the bit-planes are coded or until certain criteria is satisfied, e.g., we have reached the desired coding rate or quality. The output bit-stream has the embedded property, so if the bit-stream is truncated the more significant bits will be recovered and an estimate for each coefficient can be obtained within the uncertainty ranges.

4.5.6 Bitstream assembler

Let i be the index of a code-block. When the entropy coder constructs the code-block bitstream it also computes the coding rate R_i^k and distortion D_i^k at the end of each sub-bitplane, where k is the index of the sub-bitplane in the bit-plane i. The bitstream assembler module determines the truncation points n_i, $i = 0, \ldots, M$, in the bit-stream of each code-block i in order to achieve a minimum distortion in

w	b_1	b_2	b_3	b_4	b_5	b_6	b_7	Sign	Range	Value
45	0	1						+	[32,63)	48
−74	1							−	[-128,-64)	-96
21	0	0							[-32,31)	0
14	0								[-64,63)	0
−4	0								[-64,63)	0
−18	0								[-64,63)	0
4	0								[-64,63)	0
−1	0								[-64,63)	0

Table 4.9: Bit plane b_2 coding PS.

w	b_1	b_2	b_3	b_4	b_5	b_6	b_7	Sign	Range	Value
45	0	1						+	[32,63)	48
−74	1	0						−	[-96,-64)	-80
21	0	0							[-32,31)	0
14	0								[-64,63)	0
−4	0								[-64,63)	0
−18	0								[-64,63)	0
4	0								[-64,63)	0
−1	0								[-64,63)	0

Table 4.10: Bit plane b_2 coding (REF).

the image provided a rate compression constrain B. Thus, the bitstream assembler solves the minimization problem

$$\begin{cases} \min \sum_i D_i^{n_i}, \\ \sum_i R_i^{n_i} \leq B. \end{cases} \tag{4.4}$$

In order to solve this problem we can proceed by distributing bits first to the code-blocks with the steepest distortion-rate spend.

Definition 4.5.21. Let i be the index of a code-block which has m_i sub-bitplanes. Let $n_i \geq 0$ be the actual truncation point for this code-block ($n_i = 0$ indicates that the corresponding bit-stream is truncated at the beginning and therefore, it has its maximal distortion). The maximum possible gain of distortion decrease per rate spent is called rate-distortion slope for this code-block and is given by

$$S_i = \max_{k>n_i} \frac{D_i^{n_i} - D_i^k}{R_i^{n_i} - R_i^k}.$$

If this code-block is truncated at this moment, the new truncation point is given by

$$n_i^{\text{new}} = \arg_{k>n_i} \left\{ \frac{D_i^{n_i} - D_i^k}{R_i^{n_i} - R_i^k} = S_i \right\}.$$

w	b_1	b_2	b_3	b_4	b_5	b_6	b_7	Sign	Range	Value
45	0	1						+	[32,63)	48
−74	1	0						−	[-96,-64)	-80
21	0	0							[-32,31)	0
14	0	0							[-32,31)	0
−4	0	0							[-32,31)	0
−18	0	0							[-32,31)	0
4	0	0							[-32,31)	0
−1	0	0							[-32,31)	0

Table 4.11: Bit plane b_2 coding (PN).

Proposition 4.5.22. *Let n_i and n_i^{new} be the previous truncation point and the new truncation point of the bitstream of the code-block i, respectively. If we add $R_i^{n_i} - R_i^{n_i^{new}}$ bits to the final bitstream, then we obtain a distortion decrease of $D_i^{n_i} - D_i^{n_i^{new}}$. Moreover, if i is a code-block with the maximum rate-distortion slope, i.e., $S_i > S_j$, for all $j \neq i$, then*

$$D_i^{n_i} - D_i^{n_i^{new}}$$

is the maximum distortion decrease achievable by spending $R_i^{n_i} - R_i^{n_i^{new}}$ bits.

Hence, the algorithm consists on initialize the truncating points $n_i = 0$, for all code-blocks, and compute their rate-distortion slope, S_i, for each code-block, choosing the one with the maximum slope and actualizing its new truncating point to n_i^{new}. The bits in this code-block until the truncating point are send to the bitstream assembler and we repeat the process until we obtain the desired rate B.

Figure 4.12 (left) shows the Rate-distortion curve for the bit-plane in Example 4.5.20 where each point in the graph corresponds to a truncating point for the bit-plane bitstream and coincides with the end coding of a sub-bitplane. The distortion has been computed as the mean square error between the coefficients w and the approximated coefficients $\hat{w}^k$ at the truncating point k,

$$D^k = \sqrt{\frac{1}{N}\|w - \hat{w}^k\|_2},$$

where $w, \hat{w}^k \in \ell^2(\mathbb{Z}_N)$.

This algorithm is highly time-consuming and the process is speeded up by using an alternative method that starts by computing the convex hull of the R-D slope for each code-block i as follows:

1. Determine the set $\mathcal{S} = \{k \in \mathbb{N} : k \text{ truncating point}\}$,
2. Set $p = 0$,
3. For $k = 1, 2, 3, \ldots$ If $k \in \mathcal{S}$, then compute

$$S_k^i = \frac{D_i^p - D_i^k}{R_i^p - R_i^k}.$$

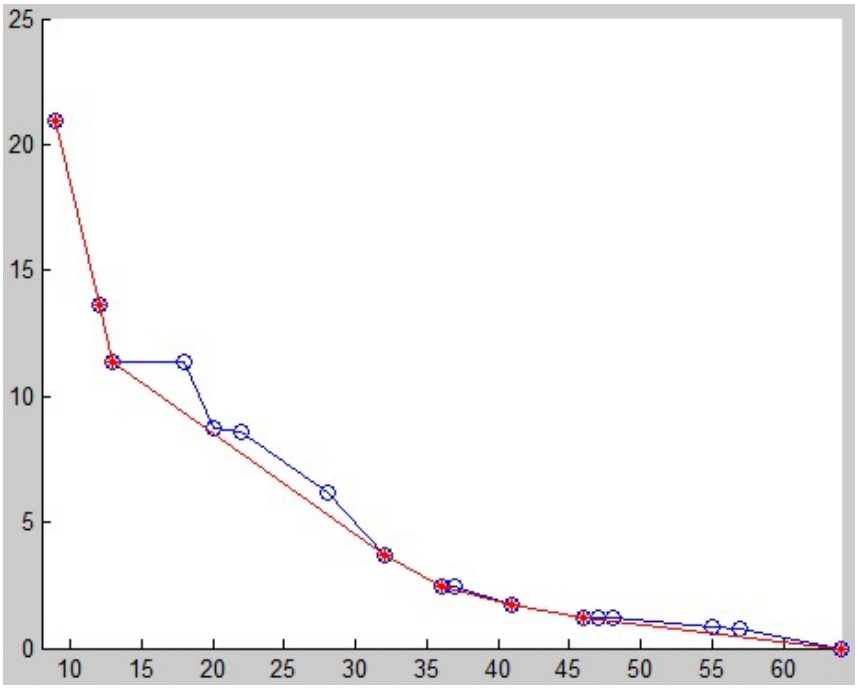

Figure 4.12: Rate-distortion curve for bit-plane in Example 4.5.20 (blue) and the same curve with its convex hull (red).

If $p > 0$ and $S_i^k > S_i^p$, then set $\mathcal{S} = \mathcal{S} \backslash \{p\}$ and go to step 2. Otherwise, set $p = k$ and repeat step 3.

In Figure 4.12 we can see the convex hull computed for the bit-plane in Example 4.5.20. We can observe how the use of the convex hull permits to avoid several truncating points. Once we have computed the convex hull for each code-block, the optimization is performed by searching a global slope λ, such that the rate constrain B is satisfied, and determine the truncating point n_i for each code-block i as

$$n_i = \arg \max_k \{S_i^k > \lambda\}.$$

Finally, putting all bitstreams of all code-blocks together we obtain an optimal compressed bitstream satisfying the restriction given by equation (4.4).

It is possible to construct a compressed image bitstream with progressive quality improvement property, so that we may gradually improve the quality of the received image just by sending more bitstreams. We just set different rate points $B_1, B_2, \ldots, B_n$ and determine the different slopes $\lambda_1, \lambda_2, \ldots, \lambda_n$ associated to each rate point and their corresponding code-block bitstreams. A typical sample rate point set is given by $\{0.0625, 0.125, 0.5, 1.0, 2.0\}$ bpp (bits per pixel), that for an image of size 512×512 corresponds to compressed bitstreams of sizes 2k, 4k, 8k, 16k, 32k and 64k bytes, respectively. Thus, the process begins by calculating the first global slope λ_1 associated to the first rate point B_1 and obtaining the truncating

point n_i^1 for the bitstream of each code-block i using this slope. The bitstreams of these code-blocks of one resolution level at one spatial location are grouped into a packet. All packets containing the first bitstreams of some code-blocks determine the first layer representing the first quality increment of the entire image at full resolution. After that, the second global slope λ_2 related to the rate point B_2 is computed, the second truncating points n_i^2 are determined for all code-blocks and the second resolution level bitstreams are formed by the bits compressed between the points n_i^1 and n_i^2 of the bitstream of each code-block i. We again assemble these bitstreams of the code-blocks into packets. The process is iterated until we obtain n layers of bitstreams, giving the full resolution image.

Chapter 5

Comparison between JPEG and JPEG 2000

In this chapter we present a comparison between the two compression algorithms, JPEG and JPEG 2000, we have introduced in chapters 3 and 4. We have applied both algorithms to four gray scale images shown in Figure 5.1 with different compression ratios obtaining images with rates of 0.0615, 0.125, 0.25, 0.5, 1, and 2 bpp (bits per pixel).

We have used a simple JPEG 2000 algorithm implemented using Matlab to obtain the compressed images. The algorithm applies a bi-orthogonal 9-7 wavelet transform to the images and uses an MQ-coder to compress the bit-stream coming from each processed code-block of size 64×64 samples. The discrete wavelet transform has been applied up to three scale levels in order to obtain the lower bit-rate of 0.0625 bpp. Figure 5.2 shows the result obtained applying both algorithms to Lenna's image for requesting rates of 0.0625, 0.125, and 0.25. The JPEG 2000 images at different rates have been obtained from the same bit-stream truncating each code block at different points depending on the bit-rate required. This is an example of the scalability feature of the JPEG 2000 standard. In the JPEG case, the different bit-rates have been produced applying the algorithm each time to the image varying the compression parameter k. Note that at a low bit-rate, 0.0625 bpp for instance, the artifacts appear in the JPEG format, while the JPEG 2000 image is free of them. We can also observe the improvement in the quality of the image while the bit-rate increases.

Figure 5.3 shows the recovered images from JPEG 2000 at bit-rates 0.5, 1, and 2 bpp. We cannot appreciate any difference between the images and, in fact, there are no differences, they are equal. We do not show the results for the JPEG because in our simple version of the algorithm it is difficult to control the bit-rate because the compression is determined by the value of the compression parameter k used in the quantization stage. In fact, the bit-rates for the JPEG images in all figures are a little bigger than those of the JPEG 2000.

In Figure 5.4 we can observe the results for the chronometer image. Again the results using the JPEG 2000 algorithm are better than JPEG's ones, at least for

Figure 5.1: Images used for comparison: Lenna, Bike, Chronometer and Pattern.

Figure 5.2: Lenna's image compressed using JPEG 2000 (left) and JPEG (right) at rates 0.0625 (up), 0.125 (middle), and 0.25 (down).

Figure 5.3: Lenna's image compressed using JPEG 2000 at rates 0.5 (left), 1 (middle), and 2 (right).

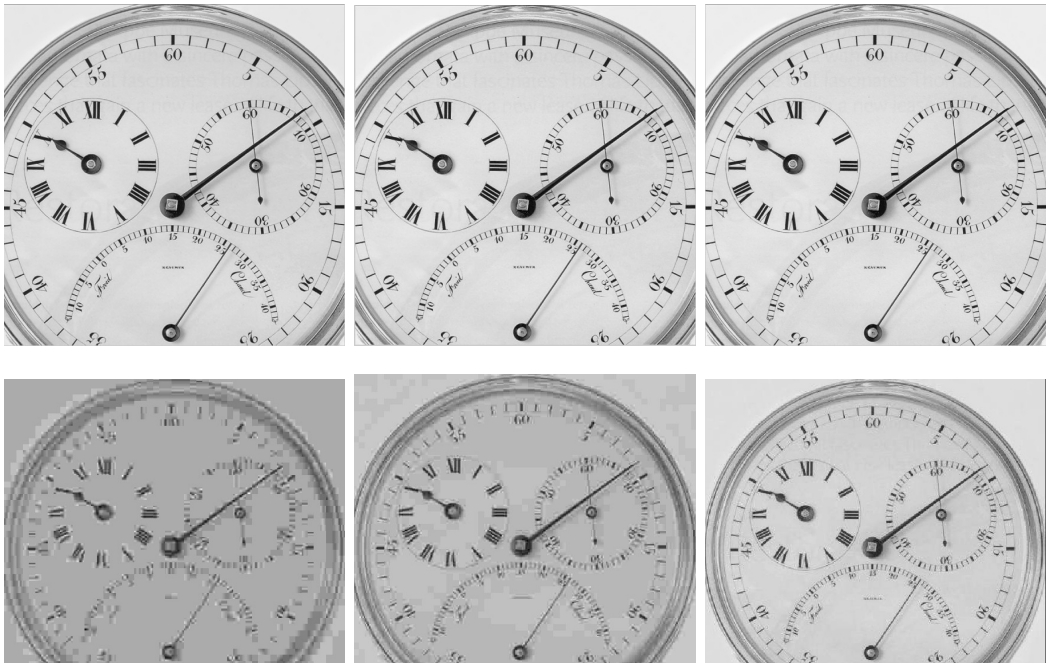

Figure 5.4: Chronometer image compressed using JPEG 2000 (up) and JPEG (down) at rates 0.0625 (left), 0.125 (middle), and 0.5 (right).

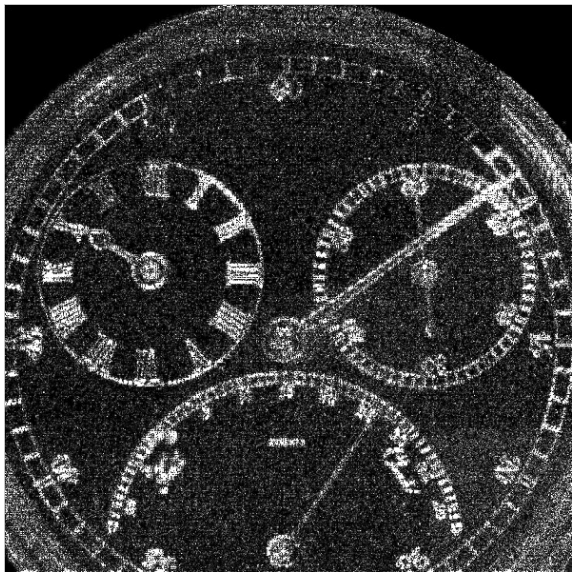

Figure 5.5: Difference between the above first and third images in Figure 5.4.

low bit-rates. Observe that in the JPEG 2000 results we cannot distinguish any difference between the images, but in Figure 5.5 we can see the difference between the first (0.0625 bpp) and the third (0.5 bpp) images.

Figure 5.6 shows the results for the Bike image. This image forms part of the test imagery used in the evaluation of the JPEG 2000 standard. We can see how for the 0.0626 bpp and 0.125 bpp bit-rates the JPEG 2000 shows better results than the JPEG. At 0.25 bpp the results seem to be very similar.

For the last image we can see the results in Figure 5.7. This image contains text and bars and the results, in this case, are very similar, but in Figure 5.8 we can observe the distortion measured for both standards as a function of the bit-rate and the results are better in JPEG 2000.

Finally, we compute the distortion between the original images and the recovered images using both algorithms for different bit-rates. The error d between the original image x and the reconstructed image $\hat{x}$ is computed as

$$d = \frac{1}{M \cdot N} \sum_{i=1}^{M} \sum_{j=1}^{N} (x(i,j) - \hat{x}(i,j))^2,$$

where M and N are the dimensions of the image. Figures 5.9 and 5.10 show the results for the images of Lenna and the Bike, respectively. In blue we can observe the distortion between the JPEG 2000 reconstructed image while the red curve represents the error using the JPEG standard. Notice how JPEG 2000 improves the results of JPEG.

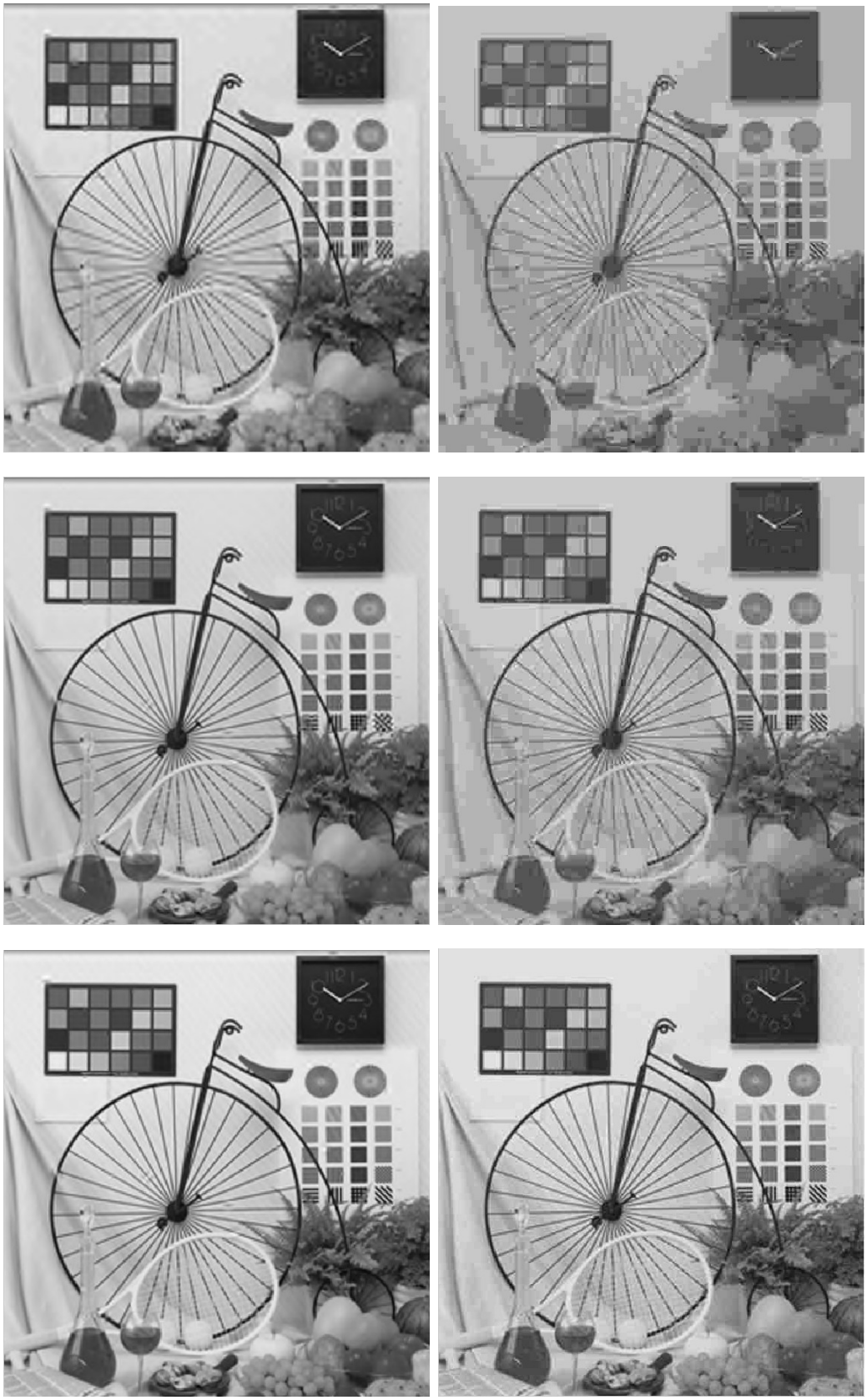

Figure 5.6: Bike image compressed using JPEG 2000 (left) and JPEG (right) at rates 0.0625 (up), 0.125 (middle), and 0.25 (down).

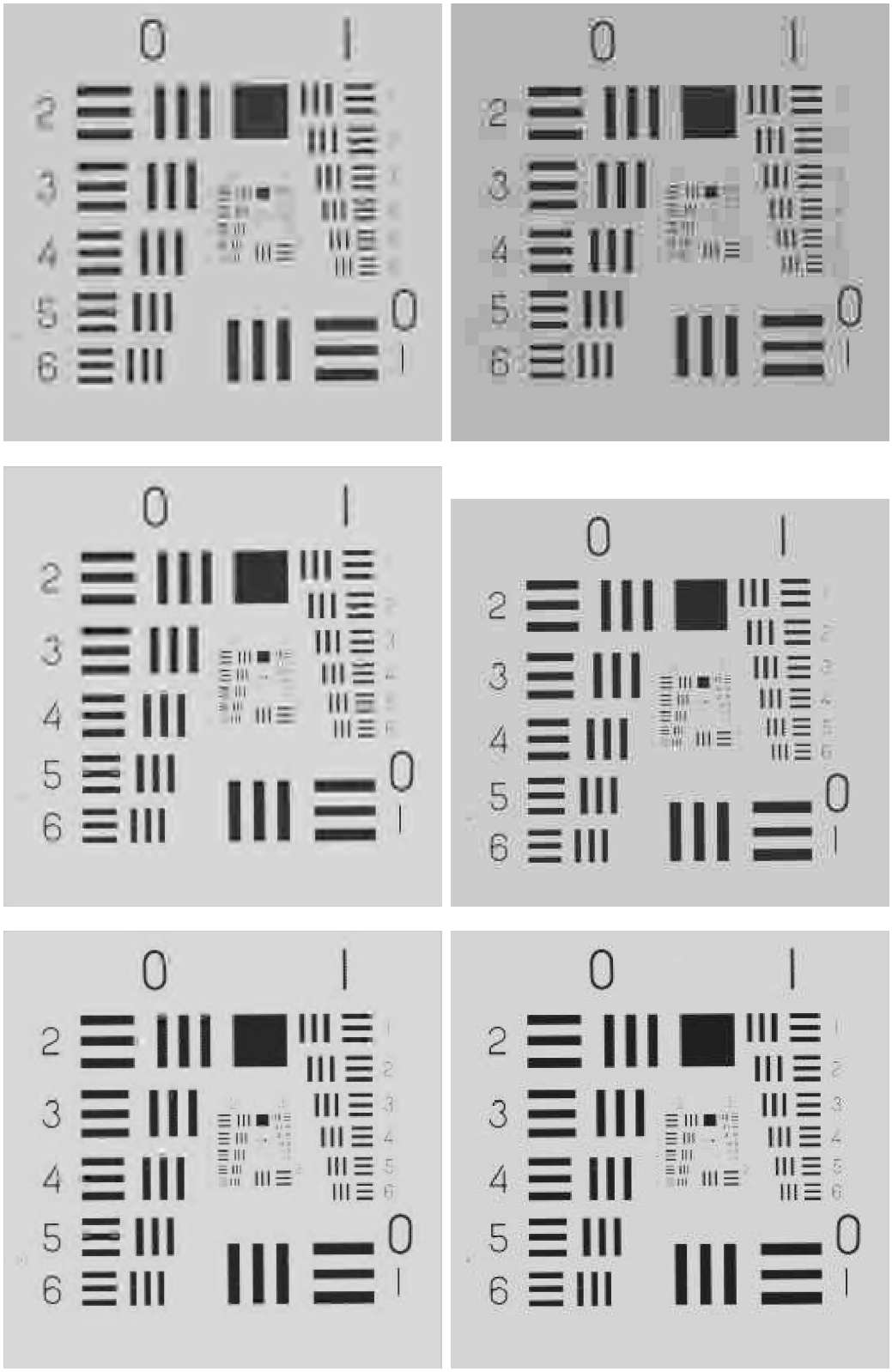

Figure 5.7: Pattern image compressed using JPEG 2000 (left) and JPEG (right) at rates 0.125 (up), 0.25 (middle), and 0.5 (down).

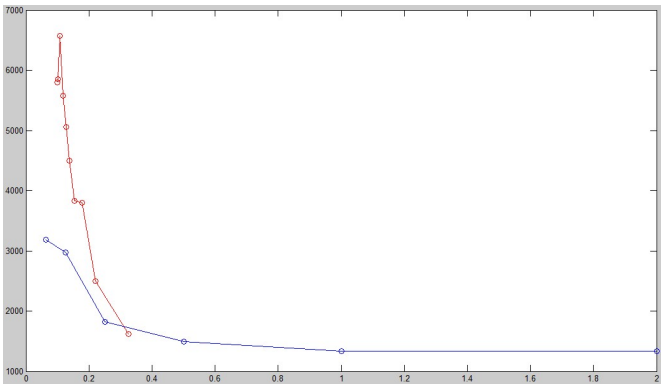

Figure 5.8: Error between the original test pattern image and the reconstructed image using JPEG (red) and JPEG2000 (blue) as a function of the bit-rate.

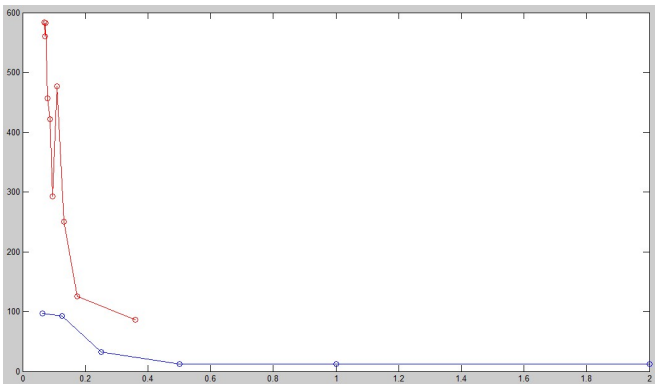

Figure 5.9: Error between the original Lenna image and the reconstructed image using JPEG (red) and JPEG2000 (blue) as a function of the bit-rate.

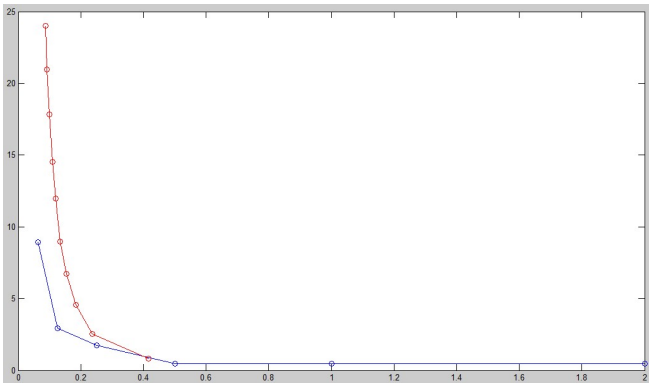

Figure 5.10: Error between the original Bike image and the reconstructed image using JPEG (red) and JPEG2000 (blue) as a function of the bit-rate.

We have shown how the JPEG 2000 standard improves the results of the JPEG algorithm and that it enriches the standard with several very useful characteristics as scalability. We have implemented a simple version of the JPEG 2000 algorithm using Matlab such that it constructs an embedding bit-stream. Hence, from this unique bit-stream we can extract versions of the original image at different bit-rates with different resolutions and qualities improving gradually their quality just by extracting more bits from the last truncation point of the suitable code-blocks in the bit-stream. In this way we obtain images with the least quality degradation given a fixed bit-rate.

Appendix A

Matlab Algorithms

A.1 JPEG implementation

```
function [recImg, error]=jpegColor(imgName,k)

 img2=imread(imgName);

 img=im2double(img2)*255;

 s=size(img);
 s2=size(s);

 recImg = img;

 if (s2(2)==3)
     % RGB Components
     R=img(:,:,1);
     G=img(:,:,2);
     B=img(:,:,3);
     % Compute luminance Y and chrominances Cr and Cb
     Y=0.299*R + 0.587*G + 0.114*B;
     Cb = -0.1687*R-0.3313*G+0.5*B+128;
     Cr = 0.5*R-0.4187*G-0.0813*B+128;

     % JPEG Compression for luminance
     [recImgLum,qMat]=jpeg1(Y,k,0);
     figure;
     subplot(1,2,1);imshow(Y/255);title('Original Luminance');
     subplot(1,2,2);imshow(recImgLum/255);title('Compressed Lum');
     % JPEG Compression for red chrominance
     [recImgCr,qMat]=jpeg1(Cr,k,1);
     figure;
```

```
    subplot(1,2,1);imshow(Cr/255);
    title('Original red Chrom');
    subplot(1,2,2);imshow(recImgCr/255);
    title('Compressed red Chrom');
    % JPEG Compression for blue chrominance
    [recImgCb,qMat]=jpeg1(Cb,k,1);
    figure;
    subplot(1,2,1);imshow(Cb/255);
    title('Original blue Chrominance');
    subplot(1,2,2);imshow(recImgCb/255);
    title('Compressed blue Chrominance');

    % Recover the compressed values for RGB
    R1=recImgLum+1.40210*(recImgCr-128);
    G1=recImgLum-0.34414*(recImgCb-128)-0.71414*(recImgCr-128);
    B1=recImgLum + 1.77180*(recImgCb-128);
    figure;
    subplot(1,3,1);imshow(R1/255);title('Compressed Red');
    subplot(1,3,2);imshow(G1/255);title('Compressed Green');
    subplot(1,3,3);imshow(B1/255);title('Compressed Blue');

    recImg(:,:,1) = R1;
    recImg(:,:,2) = G1;
    recImg(:,:,3) = B1;
 else
    [recImg,qMat]=jpeg1(img,k,0);
 end

 figure;
 subplot(1,2,1);imshow(img/255);
 subplot(1,2,2);imshow(recImg/255,[0,1]);

 % Mean quadratic error
 error = sum(sum((img-recImg).^2))/(s(1)*s(2));

 % Example
 % [jpegImg,error]=jpegColor('lenna.jpg',1);

function [recImg,qMat]=jpeg1(img,k,color)

s=size(img);
M=s(1);
N=s(2);

nRowBk = M/8;
```

```
nColBk = N/8;
qMat=zeros(M,N);
recImg=zeros(M,N);

for j=1:nRowBk
    for r=1:nColBk
        block = img((j-1)*8+1:j*8,(r-1)*8+1:r*8);
        [B,qB,dctB]=jpegBlock(block,k,color);
        qMat((j-1)*8+1:j*8,(r-1)*8+1:r*8)=qB;
        %recImg((j-1)*8+1:j*8,(r-1)*8+1:r*8)=B/255;
        recImg((j-1)*8+1:j*8,(r-1)*8+1:r*8)=B;
        %imshow(B);
    end
end
%figure;
%subplot(1,2,1);imshow(qMat);
%subplot(1,2,2);imshow(recImg);

%[recImg,qMat]=jpeg1(y,1.0);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Performs the jpeg compression on a 8x8 image block
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [B,qB,dctB]=jpegBlock(B,k,color)

% Quantization matrix for luminance
Ql= [16 11 10 16 24 40 51 61;
12 12 14 19 26 58 60 55;
14 13 16 24 40 57 69 56;
14 17 22 29 51 87 80 62;
18 22 37 56 68 109 103 77;
24 35 55 64 81 104 113 92;
49 64 78 87 103 121 120 101;
72 92 95 98 112 100 103 99];

% Quantization matrix for chrominances
Qc= [17 18 24 47 99 99 99 99;
18 21 26 66 99 99 99 99;
24 26 56 99 99 99 99 99;
47 66 99 99 99 99 99 99;
99 99 99 99 99 99 99 99;
99 99 99 99 99 99 99 99;
99 99 99 99 99 99 99 99;
99 99 99 99 99 99 99 99];
```

```
if (color == 1)
    Q = Qc;
else
    Q = Ql;
end

dctB = dct2(B);
qB = round(dctB./(k*Q));
B = (k*Q).*qB;
B = round(idct2(B));
```

A.2 Bi-orthogonal wavelet 9-7 using lifting scheme

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Bi-orthogonal wavelet using the lifting scheme
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function y=BiOrthogonalWavelet79Rows(x,col)

% Determine the type of signal (vector or image)
s=size(x);
M=s(1);
N=s(2);

y=0;

% Compute coefficients for the lifting scheme of the wavelet
a =-1.586134;
b=-0.05298011;
c=0.882911;
d=0.4435068;
K=1.149604398;

if (col)
    % Process columns first
    img.Even = x(:,1:2:N); % Even samples of columns
    img.Odd = x(:,2:2:N); % Odd samples of columns

    % First stage
    y.s1=zeros(M,N/2);
    y.s1=img.Odd+a*(img.Even+[img.Even(:,2:N/2) img.Even(:,N/2)]);

    % Second stage
```

```
    y.s2 = img.Even + b*(y.s1+[y.s1(:,1) y.s1(:,1:N/2-1)]);

    % Third stage
    y.H = y.s1 + c*(y.s2+[y.s2(:,2:N/2) y.s2(:,N/2)]);

    % Second stage
    y.L = y.s2 + d*(y.H+[y.H(:,1) y.H(:,1:N/2-1)]);

    y.L=K*y.L;
    y.H=y.H/K;
    y.comp = [y.L y.H];
else
    % Process columns first
    img.Even = x(1:2:M,:); % Even samples of columns
    img.Odd = x(2:2:M,:); % Odd samples of columns

    % First stage
    y.s1=zeros(M/2,N);
    y.s1=img.Odd+a*(img.Even+[img.Even(2:M/2,:);img.Even(M/2,:)]);

    % Second stage
    y.s2 = img.Even + b*(y.s1+[y.s1(1,:);y.s1(1:M/2-1,:)]);

    % Third stage
    y.H = y.s1 + c*(y.s2+[y.s2(2:M/2,:);y.s2(M/2,:)]);

    % Second stage
    y.L = y.s2 + d*(y.H+[y.H(1,:);y.H(1:M/2-1,:)]);

    y.L=K*y.L;
    y.H=y.H/K;
    y.comp = [y.L;y.H];
end
% y=BiOrthogonalWavelet79Rows(x,0); %For Rows
% y=BiOrthogonalWavelet79Rows(x,1); %For Columns

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Bi-orthogonal wavelet using the lifting scheme by rows or columns
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function z=BiOrthogonalWavelet79(x,plotFlag)

y=BiOrthogonalWavelet79Rows(x,1);

if (plotFlag)
    figure;
```

```
    imshow(y.comp);
    imwrite(y.comp,'g:\wavelet97Vertical.jpg','jpg');
end

z=BiOrthogonalWavelet79Rows(y.comp,0);

if (plotFlag)
    figure;
    imshow(z.comp);
    imwrite(z.comp,'g:\wavelet97.jpg','jpg');
end

%x=imread('lenna.jpg'); x=im2double(x);
%x=rgb2gray(x);z=BiOrthogonalWavelet79(x,1);
```

A.3 JPEG 2000 implementation

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [y2,y3,data]=im2jpeg2k(x1,res,bpp)

cb_size = 64; % Code Bolck size
%x=im2double(x1)*255-128;
x=im2double(x1)-0.5;
[M N]=size(x);
maxRate = M*N*bpp;

data.size = size(x);
data.res = res;
data.bpp=bpp;

z1=x;
% Compute multiresolution res bi-orthogonal wavelet 7-9
z=BiOrthogonalWavelet79(z1,0);
y=z.comp;
for k=2:res
    z1=z.comp(1:size(z.comp,1)/2,1:size(z.comp,2)/2);
    z=BiOrthogonalWavelet79(z1,0);
    y(1:size(z.comp,1),1:size(z.comp,2))=z.comp;
end

%imshow(y,[]);

% Quantization process ()
y=y/max(abs(y(:)));
```

```
y1=fix(127*y);
%figure;imshow(y1,[]);

numBlocks=0;
y2 = [];
slopes = ones(size(y1,1)/cb_size*size(y1,2)/cb_size,cb_size*cb_size);
indexSlopes = ones(1,size(y1,1)/cb_size*size(y1,2)/cb_size);

maxTruncPoints = 0;
slopesVect = [];
rateVect = [];
idBlockVect = [];
BitStream = [];

for resol=res:-1:1
    % Process LL subband of resoltion res
    s_res = size(y1)/2^resol;
    nBk = s_res/cb_size;
    if (resol==res)
        subband_start = 1;
% Only process subband 'LL' for the lower resolution
    else
        subband_start = 2;
    end

    for subband=subband_start:4
        if (subband==1)
            xStart = 0;
            yStart = 0;
            sband = 'LL';
        else
            if (subband==2)
               xStart = 0;
               yStart = s_res(2);
               sband = 'HL';
            else
                if (subband==3)
                   xStart = s_res(1);
                   yStart = 0;
                   sband = 'LH';
                else
                   xStart = s_res(1);
                   yStart = s_res(2);
                   sband = 'HH';
```

```
                end
            end
        end

        for j=1:nBk(1)
            xc = xStart +(j-1)*cb_size+1;
            for k=1:nBk(2)
                yc = yStart + (k-1)*cb_size+1;
                y3=Embedded_block_Encoder(y1(xc:xc+cb_size-1,
yc:yc+cb_size-1),sband);
                y3.x = xc; % x init point
                y3.y = yc; % y init point
                y3.n = 1; % Actual truncating point
                %y3.smax =  % Max slope from n
                y2 = [y2 y3];
                numBlocks = numBlocks + 1;
                slopes(numBlocks,1:size(y3.slope,2)) = y3.slope;
                rates(numBlocks,1:size(y3.R,2)) = y3.R;
                slopesVect = [slopesVect y3.slope];
                rateVect = [rateVect 0
(y3.R(2:end)-y3.R(1:end-1))];
                idBlockVect = [idBlockVect
numBlocks*ones(1,size(y3.R,2))];
                BitStream = [BitStream y3.bs];
                if (size(y3.R,2) > maxTruncPoints)
                    maxTruncPoints = size(y3.R,2);
                end
            end
        end
    end
end
slopes=slopes(:,1:maxTruncPoints);
tRate = 0;

%[lm2,p2,lm,prob,nelements]=huffmanEncoder(BitStream);

j=0;

%while (tRate < maxRate)
    %j = j + 1;
    %[C,I]=sort(slopes(:,j));
    %sumRate = cumsum(rates(I,j+1));
    %indRate = max(find(sumRate<=maxRate));
    %tRate = tRate + sumRate;
```

```
%end

data.slopesVect = slopesVect;
data.rateVect = rateVect;
data.idBlockVect = idBlockVect;
data.cb_size = cb_size;

[~,I]=sort(slopesVect);
% Vector I indexes the numbers of blocks with least slopes
sumRate = cumsum(rateVect(I));
indRate = find(sumRate<=maxRate,1,'last');

%layer1=y2(1).wRec
y3 = zeros(size(y1));
lx = size(y1,2)/cb_size;

nextPoint = ones(1,numBlocks);

for k=1:indRate
    j=idBlockVect(I(k));
    %q = rem(j-1,lx)+1;
    %p = fix((j-1)/lx) + 1;
    r = nextPoint(j);
    y3(y2(j).x:y2(j).x+cb_size-1,y2(j).y:y2(j).y+cb_size-1)
= reshape(y2(j).wRec(r,:),[cb_size cb_size]);
    nextPoint(j) = nextPoint(j) + 1;
end

figure;imshow(y3,[]);title(['DWT truncated at rate: ',
num2str(bpp)]);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function y=Embedded_block_Encoder(B,subband)
% B is the M x N block

context = struct([]);
tp = [0]; % Truncating points in the bit-stream

% Initialize MQ Encoder
A=hex2dec('8000');
C=0;
tbar=12;
T=dec2hex(0);
L=-1;
%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% Initialize context states
for k=0:18
    context(k+1).Sigma = 0;
    if (k==0)
        context(k+1).Sigma = 4;
    else
        if (k==9)
            context(k+1).Sigma = 3;
        else
            if (k==18)
                context(k+1).Sigma = 46;
            end
        end
    end

    context(k+1).s = 0;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[M N] = size(B);
v_tmp = 0;
sigma = zeros(M,N);
sigma_ant = zeros(M,N);
ppi = zeros(M,N);

wmax = max(abs(B(:)));
if (wmax > 0)
    K=fix(log2(wmax))+1; % Number of bit-planes needed
else
    K=1;
end

bitStream=[];
% Encode bit-planes p form K-1 to 0
Wup = ones(M,N)*(2^K-1);
%Wdown = -ones(M,N)*(2^K-1);
Wdown = zeros(M,N);
W_rec = zeros(1,M*N);
sgn = sign(B);
D=sum(sum(B.^2))/(M*N);

for p=K-1:-1:0
    bp = mod(fix(abs(B)/2^(p)),2);
    if (p<K-1)
        [bs,context,sigma,ppi,A,C,T,L,tbar,coded]=
```

```
MQ_Ecoder_Pass0(bp,sigma,sgn,ppi,context,A,C,
T,L,tbar,subband); % Encode pass 0
        if (sum(coded(:)) > 0)
            bitStream = [bitStream bs];
            tp = [tp size(bitStream,2)];
            index=find(coded);
            Wdown(index)=Wdown(index).*(1-bp(index))+
(2^p+Wdown(index)).*bp(index);
            Wup(index)=Wup(index).*bp(index)+
(Wup(index)-2^p).*(1-bp(index));
            W1=sigma.*sgn.*ceil((Wup+Wdown)/2);
            dist = sum(sum((W1-B).^2))/(N*M);
            D = [D dist];
            W_rec = [W_rec;reshape(W1,[1 size(W1,1)*size(W1,2)])];
        end
% Encode pass 1
        [bs,context,A,C,T,L,tbar,coded]=MQ_Ecoder_Pass1(bp,
sigma,sigma_ant,ppi,context,A,C,T,L,tbar,subband);
        if (sum(coded(:)) > 0)
            bitStream = [bitStream bs];
            tp = [tp size(bitStream,2)];
            index=find(coded);
            Wdown(index)=Wdown(index)
.*(1-bp(index))+(2^p+Wdown(index)).*bp(index);
            Wup(index)=Wup(index).*bp(index)+
(Wup(index)-2^p).*(1-bp(index));
            W1=sigma.*sgn.*ceil((Wup+Wdown)/2);
            dist = sum(sum((W1-B).^2))/(N*M);
            D = [D dist];
            W_rec = [W_rec;reshape(W1,[1 size(W1,1)*size(W1,2)])];
        end
    end
    [bs,context,sigma,A,C,T,L,tbar,coded]=MQ_Ecoder_Pass2(bp,
sigma,sgn,ppi,context,A,C,T,L,tbar,subband); % Encode pass 2

    if (sum(coded(:)) > 0)
        bitStream = [bitStream bs];
        tp = [tp size(bitStream,2)];
        index=find(coded);
        Wdown(index)=Wdown(index).*(1-bp(index))+
(2^p+Wdown(index)).*bp(index);
        Wup(index)=Wup(index).*bp(index)+
(Wup(index)-2^p).*(1-bp(index));
        W1=sigma.*sgn.*ceil((Wup+Wdown)/2);
```

```
        dist = sum(sum((W1-B).^2))/(N*M);
        D = [D dist];
        W_rec = [W_rec;reshape(W1,[1 size(W1,1)*size(W1,2)])];
    end
end

tp=tp/2; % Each char has 4 bits

% Determine the convex hull
if (abs(D(1)) > 0)
    if (size(D,2) > 2)
        khull=convhull(tp,D)';
        cont=1;
        while (khull(cont+1)>khull(cont))
         cont = cont +1;
        end
        khull=khull(1:cont);
    else
        khull = [1 size(tp,2)];
    end
else
    khull = [1 size(tp,2)];
end

% Determine the slopes Rate-Distortion
slope = (D(khull(1:end-1)) - D(khull(2:end)))./(tp(khull(1:end-1))
 - tp(khull(2:end)));

y.B=K;
y.bs = bs;
y.R = tp(khull);
y.D = D(khull);
y.K = khull;
y.slope = slope;
y.wRec=W_rec(khull,:);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [bst,context,sigma,ppi,A,C,T,L,tbar,coded]=MQ_Ecoder_Pass0(bp,
sigma,sgn,ppi,context,A,C,T,L,tbar,subband)

[M,N] = size(bp);
nStripes = ceil(M/4);
bst=[];
coded = zeros(M,N);
```

```
for n=1:nStripes
    str = bp(4*(n-1)+1:min(4*n,M),1:N);
    M2 = size(str,1);
    for j2=1:N
        for j1=1:M2
            j = 4*(n-1)+j1;
            k=find_ksig(sigma,j1+4*(n-1),j2,subband);
            if ((sigma(4*(n-1)+j1,j2)==0) && (k>0))
                x=str(j1,j2);
                [bs,context,A,C,T,L,tbar]=MQ_Encode(x,k,
context,A,C,T,L,tbar);
                bst = [bst bs];
                if (str(j1,j2)==1)
                   sigma(j1+4*(n-1),j2)=1;
                   [bs,context,A,C,T,L,tbar]=Encode_Sign(bp,
sigma,sgn,j1+4*(n-1),j2,context,A,C,T,L,tbar);
                   bst = [bst bs];
                end
                ppi(j1+4*(n-1),j2)=1;
                coded(j1+4*(n-1),j2)=1;
            else
                ppi(j1+4*(n-1),j2)=0;
            end
        end
    end
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [bst,context,A,C,T,L,tbar,coded]=MQ_Ecoder_Pass1(bp,
sigma,sigma_ant,ppi,context,A,C,T,L,tbar,subband)
% Magnitude refinement

[M,N] = size(bp);
nStripes = ceil(M/4);
bst=[];
coded = zeros(M,N);

for n=1:nStripes
    str = bp(4*(n-1)+1:min(4*n,M),1:N);
    M2 = size(str,1);
    for j2=1:N
        for j1=1:M2
            if ((sigma(4*(n-1)+j1,j2)==1) &&
(ppi(4*(n-1)+j1,j2)==0))
```

```
                ksig=find_ksig(sigma,j1+4*(n-1),j2,subband);
                if (sigma_ant(4*(n-1)+j1,j2)==0)
                    if (ksig > 0)
                        kmag = 16;
                    else
                        kmag = 15;
                    end
                else
                    kmag = 17;
                end
                x=str(j1,j2);
                coded(j1+4*(n-1),j2)=1;
                [bs,context,A,C,T,L,tbar]=MQ_Encode(x,kmag,
context,A,C,T,L,tbar);
                bst = [bst bs];
            end
        end
    end
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [bst,context,sigma,A,C,T,L,tbar,coded]=MQ_Ecoder_Pass2(bp,
sigma,sgn,ppi,context,A,C,T,L,tbar,subband)
% Magnitude refinement
% Define the context for k_run and kuni
krun = 9;
kuni = 18;

[M,N] = size(bp);
nStripes = ceil(M/4);
bst = [];
coded=zeros(M,N);

for n=1:nStripes
    str = bp(4*(n-1)+1:min(4*n,M),1:N);
    M2 = size(str,1);
    for j2=1:N
        for j1=1:M2
            r=-1;
            if (j1==1 && M2==4) % Entering a full stripe column
                i=0;
                ksig=find_ksig(sigma,j1+i+4*(n-1),j2,subband);
                while (and(ksig == 0,i<3))
                    i = i+1;
```

```
                    ksig=find_ksig(sigma,j1+i+4*(n-1),j2,subband);
                end
                if (and(i==3,ksig ==0)==1)
                   r=0;
%Find the first 1
                   r2=find(str(j1:j1+3,j2)==1,1,'first');
                   if (r2<=4)
                       r=r2-1;
                   else
                       r=4;
                   end
                   if (r==4)
                      [bs,context,A,C,T,L,tbar]=
MQ_Encode(0,krun,context,A,C,T,L,tbar);
                      bst = [bst bs];
                   else
                      [bs,context,A,C,T,L,tbar]=
MQ_Encode(1,krun,context,A,C,T,L,tbar);
                      bst = [bst bs];
                      [bs,context,A,C,T,L,tbar]=
MQ_Encode(fix(r/2),kuni,context,A,C,T,L,tbar);
                      bst = [bst bs];
                      [bs,context,A,C,T,L,tbar]=
MQ_Encode(mod(r,2),kuni,context,A,C,T,L,tbar);
                      bst = [bst bs];
                   end
                end
            end

            if ((sigma(j1+4*(n-1),j2)==0) && ppi(j1+4*(n-1),j2)==0)
               if (r >=0 )
                  r=r-1; % No need to code significance
                  coded(j1+4*(n-1):j1+4*(n-1)+3,j2)=1;
               else
                  ksig=find_ksig(sigma,j1+4*(n-1),j2,subband);
                  x=str(j1,j2);
                  coded(j1+4*(n-1),j2)=1;
                  [bs,context,A,C,T,L,tbar]=
MQ_Encode(x,ksig,context,A,C,T,L,tbar);
                  bst = [bst bs];
                  if (x==1)
                     sigma(j1+4*(n-1),j2)=1;
                     [bs,context,A,C,T,L,tbar]=
Encode_Sign(bp,sigma,sgn,j1+4*(n-1),j2,context,A,
```

```
C,T,L,tbar);
                    bst = [bst bs];
                end
            end
        end
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function ksig=find_ksig(sigma,j1,j2,subband)

[M,N]=size(sigma);
kh=0;
kv=0;
kd=0;
%j1
%j2
kh = sum(sigma(j1,max(j2-1,1):min(j2+1,M)))-sigma(j1,j2);
kv = sum(sigma(max(j1-1,1):min(j1+1,N),j2))-sigma(j1,j2);

kd =  sum(sum(sigma(max(j1-1,1):min(j1+1,N),
max(j2-1,1):min(j2+1,M))))-sigma(j1,j2)-kh-kv;

if (strcmp(subband,'HL')==1)
    k_aux = kh;
    kh = kv;
    kv = k_aux;
end

if (strcmp(subband,'HL')==1 || strcmp(subband,'LL')==1 ||
strcmp(subband,'LH')==1)
    if (kh==2)
       ksig = 8;
   else
      if (kh==1)
          if (kv>=1)
              ksig = 7;
          else
             if (kd >= 1)
                 ksig = 6;
             else
                 ksig = 5;
             end
          end
```

```
      else
          if (kv==2)
              ksig = 4;
          else
              if (kv==1)
                 ksig = 3;
              else
                  if (kd>=2)
                     ksig = 2;
                  else
                      ksig = kd;
                  end
              end
          end
      end
   end
end

if (strcmp(subband,'HH')==1)
    khv = kh + kv;
    if (kd >= 3)
        ksig = 8;
    else
        if (kd == 2)
            if (khv >= 1)
                ksig = 7;
            else
                ksig = 6;
            end
        else
            if (kd == 1)
               if (khv >= 2)
                  ksig = 5;
               else
                  if (khv == 1)
                     ksig = 4;
                  else
                      ksig = 3;
                  end
               end
            else
                if (khv >= 2)
                   ksig = 2;
                else
```

```
                    ksig = khv;
                end
            end
        end
    end
end


function [bst,context,A,C,T,L,tbar]=Encode_Sign(bp,sigma,
sgn,j1,j2,context,A,C,T,L,tbar)

[M,N]=size(bp);
bst=[];
Xh = 0;
if (j2>1)
    Xh = sgn(j1,j2-1)*sigma(j1,j2-1);
end
if (j2<N)
    Xh=Xh+sgn(j1,j2+1)*sigma(j1,j2+1);
end

Xv = 0;
if (j1>1)
    Xv = sgn(j1-1,j2)*sigma(j1-1,j2);
end
if (j1<M)
    Xh=Xh+sgn(j1+1,j2)*sigma(j1+1,j2);
end

Xh = sign(Xh)*min(1,abs(Xh));
Xv = sign(Xv)*min(1,abs(Xv));

if (Xh == 1)
   k_sign = 13 + Xv;
   X_flip = 1;
else
    if (Xh == 0)
        k_sign = 10 + abs(Xv);
        if (Xv == -1)
            X_flip = -1;
        else
            X_flip = 0;
        end
    else
        % Xh ==-1
```

```
        k_sign = 13 - Xv;
        X_flip = -1;
    end
end

if (sgn(j1,j2)*X_flip==1)
   [bs,context,A,C,T,L,tbar]=MQ_Encode(0,k_sign,context,A,
C,T,L,tbar);
   bst = [bst bs];
else
   [bs,context,A,C,T,L,tbar]=MQ_Encode(1,k_sign,context,A,
C,T,L,tbar);
   bst = [bst bs];
end


function [bs,context,A,C,T,L,tbar]=
MQ_Encode(x,k,context,A,C,T,L,tbar)

% x is the symbol to encode
% context(k) is the context for the symbol x

k=k+1; % To acces the table the index must start at 1, not at zero

[Sigma,Sigma_mps,Sigma_lps,Xs,pb,p]=findProbStateTrans();
pbar = pb(k);
s = context(k).s;
A=A-pbar;
bs=[];

if (A<pbar)
    s = 1-s;    % Conditional exchange of MPS and LPS
end

if (x==s)
   C = C + pbar; % Assign MPS the upper subinterval
else
    A = pbar;   % Assign LPS the lower subinterval
end

if (A<2^15)
   if (x==context(k).s)
   % The symbol was a real MPS
        context(k).Sigma=Sigma_mps(context(k).Sigma+1);
   else
```

```
   % The symbol was a real LPS
        if (Xs(context(k).Sigma+1)==1)
           context(k).s = 1-context(k).s; % Switch MPS/LPS
        end
        context(k).Sigma=Sigma_lps(context(k).Sigma+1);
   end
end

while (A < 2^15) % Perform a normalization shift
   A=2*A;
   C=2*C;
   tbar= tbar-1;
   if (tbar==0)
      [bs,T,C,L,tbar]=MQ_Transfer_Byte(T,C,L,tbar);
   end
end
```

Bibliography

[1] N. Ahmed, T. Natarajan, and K. R. Rao, *Discrete cosine transform,* IEEE Trans. Comput. **C-23** (1974), 90–93.

[2] D. Austin, *What is JPEG?,* Notices of the AMS **55** (2008), 226–229.

[3] C. Bénéteau and P.J. Van Fleet, *Discrete wavelet transformations and undergraduate education,* Notices of the AMS **58** (2011), 656–666.

[4] J. Blinn, *What's the deal with DCT,* IEEE Computer Graphics and Applications **13** (1993), 78–83.

[5] CCITT, *Information technology: Digital compression and coding of continuous-tone still images: Requirements and guidelines,* Recommendation T.81, International Telephone and Telegraph Consultative Committee, 1992. http://www.w3.org/Graphics/JPEG/itu-t81.pdf.

[6] I. Daubechies, *Ten Lectures on Wavelets,* CBMS-NSF Regional Conf. Series in Appl. Math. **61**, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1992.

[7] I. Daubechies and W. Sweldens, *Factoring wavelet transforms into lifting steps*, J. Fourier Anal. Appl. **4** (1998), 247-269.

[8] R. D. Dony, *The Karhunen-Loève transform,* in *The Transform and Data Compression Handbook,* 20–55, Ed. K. R. Rao and P.C. Yip, Boca Raton, CRC Press LLC, 2001.

[9] M. W. Frazier, *An Introduction to Wavelets Through Linear Algebra,* Springer, 2000.

[10] R. C. González, *Digital Image Processing using Matlab,* Gatesmark Publishing, 2009.

[11] J. Li, *Image compression - the mathematics of JPEG 2000,* Modern Signal Processing **46** (2003), 185–221.

[12] J. Li and S. Lei, *An embedded still image coder with rate-distortion optimization,* IEEE Trans. On Image Processing **7** (1999), 913–924.

[13] W. Pennebaker, J. Mitchell, G. Langdon, and R. Arps, *An overview of the basic by principles of the Q-Coder adaptive binary arithmetic coder,* IBM J. Res. Develop **32** (1998), 717–726.

[14] I. Popovici and D. Withers, *Locating edges and removing ringing artifacts in JPEG images by frequency-domain analysis,* IEEE Transactions on Image Processing **16** (2007), 1470–1474.

[15] G. Strang, *The discrete cosine transform,* SIAM **41** (1999), 135–147.

[16] A. Swart, *An introduction to JPEG compression using MATLAB,* http://www.math.uu.nl/~sleij101/Opgaven/FourWav/ExercisesWavelets/jpg.pdf.

[17] D. S. Taubman and M. W. Marcellin, *JPEG2000. Image Compression, Fundamentals, Standards and Practice,* Springer, 2002.

Index