UNIVERSITAT DE BARCELONA

Effective Algorithms in Harmonic Analysis and Applications to Signal Processing

Master in Advanced and Professional Mathematics

Course 2010/2011

Raquel GIL

Tutor:
Joan Carles Naranjo

Supervisor:
Javier Soria

Contents

Introduction

The interest in Fourier and Wavelets Transforms is because Harmonic Analysis is very useful in signal processing nowadays. Signal compression, de-noising, recognition are some examples of its range. In Mathematics, we know its development in continuous ($\mathbb{R}$) and discrete time ($\mathbb{Z}$), but here we will deal only with the discrete part. All the signals will be treated like vectors and matrices. We will not only deal with Harmonic Analysis, but also with other fields in signal processing like Morphological operations and Image filtering. All these topics have a theoretical part based on rigorous mathematical theories, that will be explained in detail. But our goal is to go further by implementing, on a computer, the most important techniques and algorithms used to develop in practice these methods.

In first and second chapter, *Discrete Fourier Transform* and *Fast Fourier Transform* respectively, we will prove why is used the fast algorithm to compute the Fourier transform. The fact is that, if we compute the discrete transform, the time required is of the order of N^2, where N is the length of the signal; but if we compute the fast algorithm, the time spend in it will be of $N \log N$. For example, if we spent $100s$ to compute the *Discrete Fourier Transform*, we will spend $33s$ in *Fast Fourier Transform*.

Morphological operations are used to determine features of the analysed objects, their form, their neighborhood. In the chapter of Intensity Images and Filtering we will work in the space domain and in the frequency domain, because in this moment we will have the necessary tools to compute the Fourier transform.

In each chapter there is a section of *Matlab Implementation*. All the functions

and GUI's (Graphical user interface's) have been implemented with the program *Matlab* (Matrix Laboratory). The *IDE* (Integrated development environment) used in the project has been *Microsoft Visual C++ 2008 Express Edition* and *Matlab R2009b*. The languages supported are *C/C++* and *Matlab*, its own language, respectively.

Matlab has a compile library in *C++* to compute the *Fast Fourier Transform*, so for this reason, we use *Microsoft Visual*. First, the code is implemented in *C++* with the libraries *OpenCV* (Open Computer Vision) and the header of *Matlab*, *mex.h*. Then a *mex* file is created from *Matlab* to make the code works on it. This new file works like another function in the program. The code of each function is at the end of its corresponding chapter to clarify its development. The GUI's code has not been included since it is to large, around 200 lines of code for each one.

Chapter 1

The Discrete Fourier Transform

In this chapter we will introduce and deal with the discrete Fourier transform. First, we will introduce the notation, definitions and properties, followed by its matrix representation, its code implementation in one and two dimensions, and finally some examples of the implemented algorithms.

1.1 Properties of the Discrete Fourier Transform

Notation 1.1. *In the discrete case we work with $\mathbb{C}^N$ vectors, sequences of N complex numbers. See [1].*

1. *Now we consider z as a function defined in the finite set $\mathbb{Z}_N = \{0, ..., N-1\}$. We will write the z's components in this way $z = (z(0), ..., z(N-1))$.*

2. *For the finite dimensional case we introduce the vector space over $\mathbb{C}$*

$$\ell^2(\mathbb{Z}_N) = \{z \mid z(j) \in \mathbb{C},\ 0 \leq j \leq N-1\}.$$

   Note that the Euclidean basis is an orthogonal basis of this space.

3. *Inner product*

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

4. $\ell^2(\mathbb{Z}_N) - norm$

$$\|z\| = \left(\sum_{k=0}^{N-1} |z(k)|^2 \right)^{1/2}.$$

5. *We can extend the sequence of* z *to any* $j \in \mathbb{Z}$ *assuming that* z *is periodic with period* N*:*

$$z(j+N) = z(j), \ for\ all\ j \in \mathbb{Z}.$$

The definition and lemma below are necessary for introducing the Discrete Fourier Transform (DFT).

Definition 1.2. *Define* $E_0, ..., E_{N-1} \in \ell^2(\mathbb{Z}_N)$ *by*

$$E_m(n) = \frac{1}{\sqrt{N}} e^{2\pi imn/N}, \ 0 \le m, n \le N-1. \tag{1.1}$$

Lemma 1.3. *The set* $\{E_0, ..., E_{N-1}\}$ *is an orthonormal basis of* $\ell^2(\mathbb{Z}_N)$.

Proof. Suppose $m, m' \in \{0, ..., N-1\}$. Then

$$\begin{aligned} \langle E_m, E_{m'} \rangle &= \sum_{n=0}^{N-1} E_m(n) \overline{E_{m'}(n)} = \sum_{n=0}^{N-1} \frac{1}{\sqrt{N}} e^{2\pi imn/N} \frac{1}{\sqrt{N}} e^{-2\pi im'n/N} \\ &= \frac{1}{N} \sum_{n=0}^{N-1} e^{2\pi i(m-m')n/N}. \end{aligned}$$

If $m = m'$

$$\langle E_m, E_m \rangle = \frac{1}{N} \sum_{n=0}^{N-1} e^{2\pi i(m-m)n/N} = \frac{1}{N} \sum_{n=0}^{N-1} 1 = 1.$$

If $m \neq m'$

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

□

Since $\{E_0, ..., E_{N-1}\}$ is an orthonormal basis for $\ell^2(\mathbb{Z}_N)$, we have for all $z, w \in \ell^2(\mathbb{Z}_N)$ the following properties:

$$z = \sum_{m=0}^{N-1} \langle z, E_m \rangle E_m \tag{1.2}$$

$$\langle z, w \rangle = \sum_{m=0}^{N-1} \langle z, E_m \rangle \overline{\langle w, E_m \rangle} \tag{1.3}$$

$$\|z\|^2 = \sum_{m=0}^{N-1} |\langle z, E_m \rangle|^2 \tag{1.4}$$

where

$$\langle z, E_m \rangle = \sum_{m=0}^{N-1} z(n) \overline{\frac{1}{\sqrt{N}} e^{2\pi imn/N}} = \frac{1}{\sqrt{N}} \sum_{m=0}^{N-1} z(n) e^{-2\pi imn/N} \tag{1.5}$$

Definition 1.4. *Suppose* $z = (z(0), ..., z(N-1)) \in \ell^2(\mathbb{Z}_N)$. *For* $m = 0, ..., N-1$ *define*

$$\hat{z}(m) = \sum_{n=0}^{N-1} z(n) e^{-2\pi imn/N}. \tag{1.6}$$

Let

$$\hat{z} = (\hat{z}(0), ..., \hat{z}(N-1)). \tag{1.7}$$

Then $\hat{z} \in \ell^2(\mathbb{Z}_N)$ *and the map* $z \longrightarrow \hat{z}$ *is called the discrete Fourier transform DFT.*

Remark 1.5. *It is easy to see that the map*

$$\ell^2(\mathbb{Z}_N) \xrightarrow{\hat{}} \ell^2(\mathbb{Z}_N)$$

$$z \longmapsto \hat{z}$$

is a linear transformation.

We observe that $\hat{z}$ is also an N-periodic vector:

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

Note that

$$\hat{z}(m) = \sqrt{N} \langle z, E_m \rangle . \tag{1.8}$$

Theorem 1.6. *Let $z = (z(0), ..., z(N-1))$, $w = (w(0), ..., w(N-1)) \in \ell^2(\mathbb{Z}_N)$. Then*

1. *Fourier inversion formula:*

$$z(n) = \frac{1}{N} \sum_{m=0}^{N-1} \hat{z}(m) e^{2\pi imn/N}, \ for\ n = 0, ..., N-1. \tag{1.9}$$

2. *Parseval's relation:*

$$\langle z, w \rangle = \frac{1}{N} \sum_{m=0}^{N-1} \hat{z}\overline{\hat{w}} = \frac{1}{N} \langle \hat{z}, \hat{w} \rangle . \tag{1.10}$$

3. *Plancherel's formula:*

$$\|z\|^2 = \frac{1}{N} \sum_{m=0}^{N-1} |\hat{z}(m)|^2 = \frac{1}{N} \|\hat{z}\|^2 . \tag{1.11}$$

Proof.

1.

$$z(n) = \sum_{m=0}^{N-1} \langle z, E_m \rangle E_m(n) = \sum_{m=0}^{N-1} \frac{1}{\sqrt{N}} \hat{z}(m) \frac{1}{\sqrt{N}} e^{2\pi imn/N} = \frac{1}{N} \sum_{m=0}^{N-1} \hat{z}(m) e^{2\pi imn/N}.$$

2.

$$\langle z, w \rangle = \sum_{m=0}^{N-1} \langle z, E_m \rangle \overline{\langle w, E_m \rangle} = \sum_{m=0}^{N-1} \frac{1}{\sqrt{N}} \hat{z}(m) \overline{\frac{1}{\sqrt{N}} \hat{w}(m)} = \frac{1}{N} \sum_{m=0}^{N-1} \hat{z}(m) \overline{\hat{w}(m)}.$$

3.
$$\|z\|^2 = \sum_{m=0}^{N-1} |\langle z, E_m\rangle|^2 = \frac{1}{N}\sum_{m=0}^{N-1} |\hat{z}\,(m)|^2\,.$$

□

Remark 1.7.

1. *We call* $\{E_0, ..., E_{N-1}\}$ *the Fourier basis for* $\ell^2(\mathbb{Z}_N)$.
2. *The vector representing* z *with respect to the Fourier basis is* $\hat{z}$.
3. *The Fourier inversion formula in the above theorem is the change of basis formula for the Fourier basis.*
4. *The DFT components* $\hat{z}\,(m)$ *are the components of* z *in the Fourier basis.*

Definition 1.8. *The convolution of* $z,\ w \in \ell^2(\mathbb{Z}_N)$ *is defined as*

$$(z * w)(n) = \sum_{l=0}^{N-1} z(l)\cdot w(n-l), \quad n = 0, ..., N-1. \tag{1.12}$$

Theorem 1.9. *Suppose* $z,\ w \in \ell^2(\mathbb{Z}_N)$. *Then,*

$$(z * w)^{\wedge}(n) = \hat{z}(n)\cdot\hat{w}(n), \quad n = 0, ..., N-1. \tag{1.13}$$

Proof.

$$\begin{aligned}
(z * w)^{\wedge}(n) &= \sum_{m=0}^{N-1}\left(\sum_{l=0}^{N-1} z(l)\cdot w(m-l)\right) e^{-2\pi imn/N} \\
&= \sum_{l=0}^{N-1} z(l)\left(\sum_{m=0}^{N-1} w(m-l)\cdot e^{-2\pi i(m-l)n/N}\right) e^{-2\pi iln/N} \\
&= \left(\sum_{l=0}^{N-1} z(l)\cdot e^{-2\pi iln/N}\right)\left(\sum_{m=0}^{N-1} w(m)\cdot e^{-2\pi imn/N}\right) \\
&= \hat{z}(n)\cdot\hat{w}(n).
\end{aligned}$$

□

1.1.1 Matrix representation of *DFT*

Notation 1.10.

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

Definition 1.11. *Let W_N be the matrix $(w_{mn})_{0\leq m,n\leq N-1}$ such that $w_{mn} = \omega_N^{mn}$*

$$\begin{pmatrix} 1 & 1 & \dots & 1 \\ 1 & \omega_N & \dots & \omega_N^{N-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_N^{N-1} & \dots & \omega_N^{(N-1)(N-1)} \end{pmatrix}$$

Regarding $z, \hat{z} \in \ell^2(\mathbb{Z}_N)$ as column vectors, we can write

$$\hat{z} = W_N \cdot z$$

Signal processing deals with images, sounds and others, so we will refer to images and sounds like signals.

Definition 1.12. *Let us consider a signal f of size $M \times N$. We define the 2D Discrete Fourier Transform*

$$\hat{f}(u,v) = \sum_{j=0}^{M-1}\sum_{k=0}^{N-1} f(j,k)e^{-2\pi i(uj/M+vk/N)}, \tag{1.14}$$

where $u = 0, ..., M-1$ and $v = 0, ..., N-1$.

1.2 *DFT* Matlab implementation

Implementation of *DFT* using the *W's* matrix.

Code

```
function Z = DFT_mat(z)

N = length(z);
W = ones(N);
w = exp(-2*pi*i/N);
% Auxiliar array&matrix %
x = 1:N-1;
mat = zeros(N-1);
for k=1:N-1
    mat(k,:) = x.*k;
end
%%%%%%%%%%%%%%%%%%%%%%%%%

% %% Matrix of w's %%
W(2:end,2:end) = w.^(mat);
%% Check if z is a column vector %%
[r,c] = size(z);
if c == N
        Z = W*z';
else
        Z = W*z;
end
```

Implementation of DFT using the definition of the map

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

Code

```
function Z=DFT(z)

N = length(z);
Z = zeros(1,N);
for m=1:N
       for n=1:N
           Z(m) = Z(m) + z(n)*exp(-2*pi*i*((m-1)*(n-1)/N));
       end
end
```

2D DFT Matlab implementation.

Code

```
 function F=DFT2(signal)

[r,c] = size(signal);
F = zeros(r,c);

for u=1:r
 for v=1:c
    for j=1:r
       for k=1:c
           F(u,v) = F(u,v) + signal(j,k)*exp(-2*pi*i*((v-1)*(k-1)/c)
                                            + (j-1)*(u-1)/r);
       end
    end
```

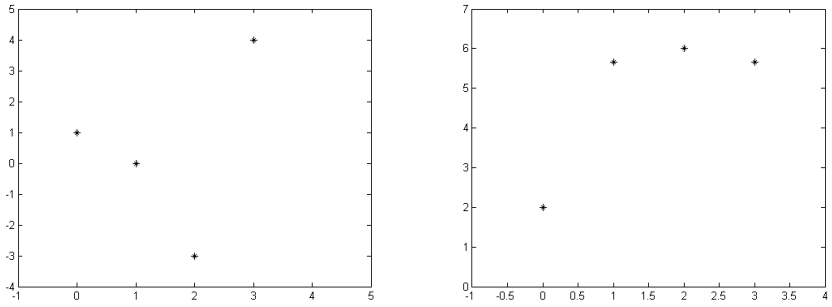
```
  end
end
```

1.3 Examples

Definition 1.13. *The spectrum of the Fourier Transform of a signal is its absolute values.*

Let us consider,

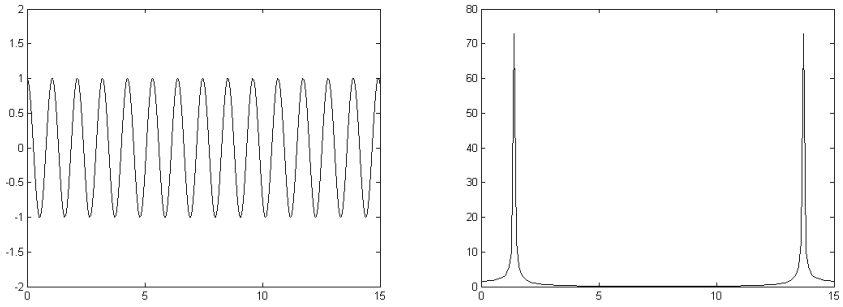
1. $z_0 = (1, 0, -3, 4) \in \ell^2(\mathbb{Z}_4)$,



(a) z_0 (b) Spectrum of $\hat{z}_0$

Figure 1.1: DFT of $z_0 = (1, 0, -3, 4)$

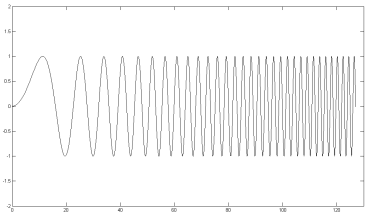
2. $z_1 = \cos(2\pi 15n/16) \in \ell^2(\mathbb{Z}_{16})$,



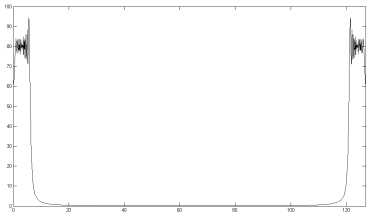
(a) z_1 (b) Spectrum of $\hat{z}_1$

Figure 1.2: DFT of $z_1(n) = \cos(2\pi 15n/16)$

3. $z_2 = \sin(\pi n^2/256) \in \ell^2(\mathbb{Z}_{128})$,



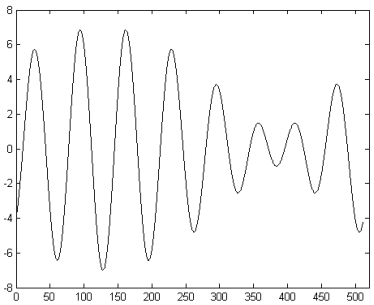
(a) z_2



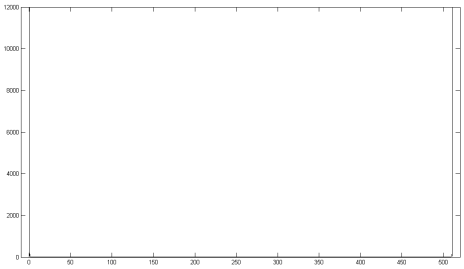
(b) Spectrum of $\hat{z}_2$

Figure 1.3: DFT of $z_2(n) = \sin(\pi n^2/256)$

4. $z_3(n) = 3\sin(2\pi 7n/512) - 4\cos(2\pi 8n/512) \in \ell^2(\mathbb{Z}_{512})$.



(a) z_3



(b) Spectrum of $\hat{z}_3$

Figure 1.4: DFT of $z_3(n) = 3\sin(2\pi 7n/512) - 4\cos(2\pi 8n/512)$

The following table represents the time (seconds) spent in processing the signals above. The first column *DFTmat* and the second column *DFT* represent the algorithms 1.1.1 and 1.6 respectively.

Length	*DFTmat (s)*	*DFT (s)*
4	$1.5142 \cdot 10^{-4}$	$9.1352 \cdot 10^{-5}$
16	$6.8640 \cdot 10^{-4}$	$4.1234 \cdot 10^{-4}$
128	0.0654	0.0179
512	0.6629	0.1993
1024	2.8469	0.6662

Figure 1.5 represents the graph of time varying the length of the signals. It can be observe that the algorithm that uses the matrix to calculate the Fourier Transform is slower than the one that applies the definition directly.

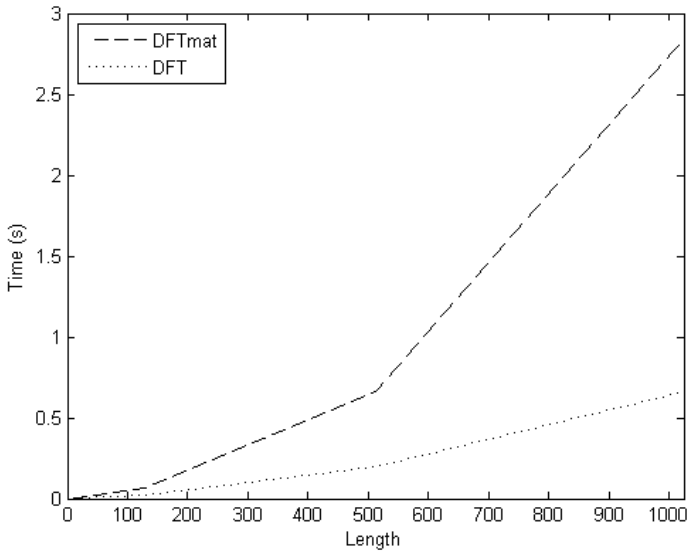


Figure 1.5: Time comparison

Chapter 2

The Fast Fourier Transform

In this chapter we will introduce the Fast Fourier Transform to compute Fourier definition explained above, to improve the execution time in computers.

2.1 Description of *FFT*

The DFT can be computed by a fast algorithm, known as the *fast Fourier transform (FFT)*. We will see the efficiency of FFT by comparing the number of complex multiplications in each algorithm, DFT vs FFT. We do not make reference to the number of complex additions because complex multiplications have slowly time computation. We have $\hat{z} = W_N \cdot z$, where the matrix W_N has dimensions $N \times N$, and hence the number of complex multiplications is N^2. See [3].

We consider only the case in which the length N of the vector is even, and in particular a power of 2. The lemma below shows the basic idea behind the FFT.

Lemma 2.1. *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*

$$u(k) = z(2k) \ \ for\ k = 0, ..., M-1,$$

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

In other words,

$$u = (z(0), z(2), ..., z(N-4), z(N-2)),$$

$$v = (z(1), z(3), ..., z(N-3), z(N-1)).$$

Let $\hat{z}$ denote the DFT of z, given by $\hat{z} = W_N z$. Let $\hat{u}$ and $\hat{v}$ denote the DFTs of u and v respectively, defined for $M = N/2$, $\hat{u} = W_M \cdot u$, $\hat{v} = W_M \cdot v$. Then, for $m = 0, ..., M-1$,

$$\hat{z}(m) = \hat{u}(m) + e^{-2\pi i m/N} \hat{v}(m); \tag{2.1}$$

and for $m = M, ..., N-1$ and $l = m - M$, we have

$$\hat{z}(l + M) = \hat{u}(l) - e^{-2\pi i l/N} \hat{v}(l). \tag{2.2}$$

Proof. For any $m = 0, ..., N-1$

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

The sum over $n = 0, ..., N-1$ can be broken into the sum over the even values $n = 2k$, $k = 0, ..., M-1$ plus the sum over the odd values $n = 2k+1$, $k = 0, ..., M-1$:

$$\begin{aligned}
\hat{z}(m) &= \sum_{k=0}^{M-1} z(2k) e^{-2\pi i 2km/N} + \sum_{k=0}^{M-1} z(2k+1) e^{-2\pi i (2k+1)m/N} \\
&= \sum_{k=0}^{M-1} u(k) e^{-2\pi i km/(N/2)} + e^{-2\pi i m/N} \sum_{k=0}^{M-1} v(k) e^{-2\pi i km/(N/2)} \\
&= \sum_{k=0}^{M-1} u(k) e^{-2\pi i km/M} + e^{-2\pi i m/N} \sum_{k=0}^{M-1} v(k) e^{-2\pi i km/M}. \qquad (2.3)
\end{aligned}$$

In the case of $m = 0, ..., M-1$ we obtain

$$\hat{u}(m) + e^{-2\pi i m/N} \hat{v}(m).$$

Suppose $m = M, ..., N-1$ and $l = m - M$, so that $l = 0, .., M-1$. By writing

$m = l + M$ in the expression 2.3 we get

$$\begin{aligned}\hat{z}(m) &= \sum_{k=0}^{M-1} u(k)\, e^{-2\pi ik(l+M)/M} + e^{-2\pi i(l+M)/N} \sum_{k=0}^{M-1} v(k)\, e^{-2\pi ik(l+M)/M} \\ &= \sum_{k=0}^{M-1} u(k)\, e^{-2\pi ikl/M} - e^{-2\pi il/N} \sum_{k=0}^{M-1} v(k)\, e^{-2\pi ikl/M},\end{aligned}$$

since $e^{-2\pi iM/N} = e^{-2\pi i(N/2)/N} = e^{-\pi i} = -1$. Then, in the case $m = M, ..., N-1$

$$\hat{u}(m) - e^{-2\pi im/N}\hat{v}(m).$$

□

2.2 *FFT* algorithm: Decimation in Time, Radix2

We will assume that N is a power of 2, $N = 2^n$, with $n \in \mathbb{Z}^+$, Walker [3]. By the Lemma 2.1 we write

$$\hat{z}(m) = \hat{u}(m) + \omega_N^m \cdot \hat{v}(m), \ m = 0, ..., M-1,$$

$$\hat{z}(m) = \hat{u}(m) - \omega_N^m \cdot \hat{v}(m), \ m = M, ..., N-1,$$

where $\omega_N^m = e^{-2\pi im/N}$. These calculations can be plotted as

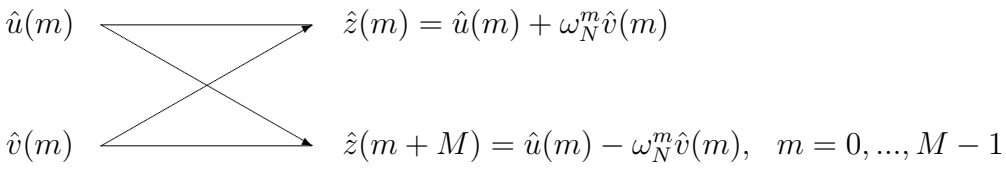


Figure 2.1: *Butterfly* diagram.

To apply equations (2.1) and (2.2), we first compute $\hat{u}$ and $\hat{v}$. Each of these is a vector of length $M = N/2$, so each can be computed with M^2 complex multiplications.

Then we compute the products $\omega_N^m \cdot \hat{v}(m)$, $m = 0, ..., M-1$. This requires M multiplications. So the total number of complex multiplications required to compute $\hat{z}$ is at most

$$2M^2 + M = 2\left(\frac{N}{2}\right)^2 + \frac{N}{2} = \frac{1}{2}\left(N^2 + N\right).$$

For N large, this is essentially $N^2/2$, whereas N^2 multiplications are required to compute $\hat{z}$ directly.

Notation 2.2. *We define $\#_N$ as the least number of complex multiplications required to compute the DFT of a vector of length $N \in \mathbb{Z}^+$.*

If $N = 2M$, then equations (2.1) and (2.2) reduce the computation of $\hat{z}$ to the computation of two *DFTs* of size M, plus M additional complex multiplications

$$\#_N \leq 2\#_M + M. \tag{2.4}$$

Lemma 2.3. *Suppose $N = 2^n$ for some $n \in \mathbb{N}$. Then,*

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

Proof. By induction on n:

Let $n = 1$. In this case we have a vector of length 2^1, $z = (z_1, z_2)$. Then,

$$\hat{z} = W_2 z = \begin{pmatrix} 1 & 1 \\ 1 & \omega_2 \end{pmatrix} \begin{pmatrix} z_1 \\ z_2 \end{pmatrix} = \begin{pmatrix} z_1 + z_1 \\ z_1 - z_2 \end{pmatrix},$$

where $\omega_2 = e^{-\pi i}$.

This computation does not require any complex multiplication, so

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

By induction, assume it holds for $n-1$. Then,

$$\#_{2^n} \underbrace{\leq}_{(2.4)} 2\#_{2^{n-1}} + 2^{n-1} \leq 2\left(\frac{1}{2} 2^{n-1} \log_2 2^{n-1}\right) + 2^{n-1}$$

$$= 2^{n-1}(n-1) + 2^{n-1} = n2^{n-1} = \frac{1}{2} n 2^n = \frac{1}{2} n \log_2 n.$$

□

Example 2.4. *Let us consider a vector of length* $N = 2^3$.

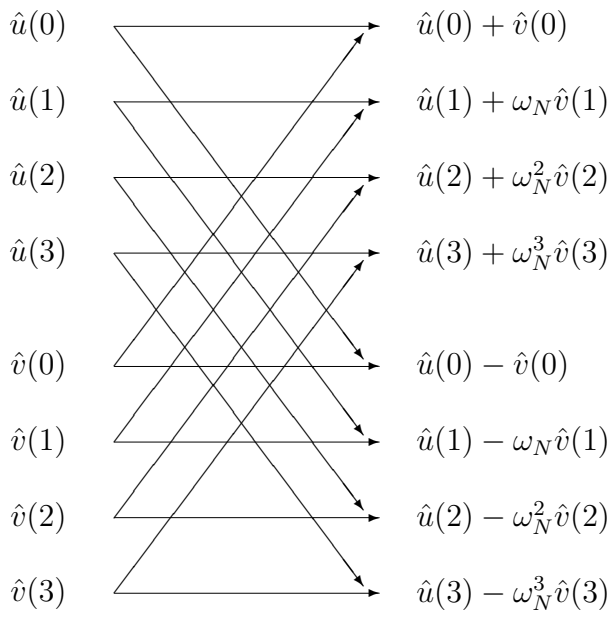


Figure 2.2: First reduction in *FFT* algorithm for $N = 8$.

We can split $\hat{u}$ and $\hat{v}$ into two DFT half-size of the original vector and so on. In this example we will consider the vector $z = (1, 2, 0, i, 1, -1, 3, -i)$. The length of the vector is $8 = 2^3$. This means that we can split $\hat{u}$ and $\hat{v}$ three times to obtain the DFT of vectors of length 1. Figure 2.3 shows the three splits to calculate its FFT.

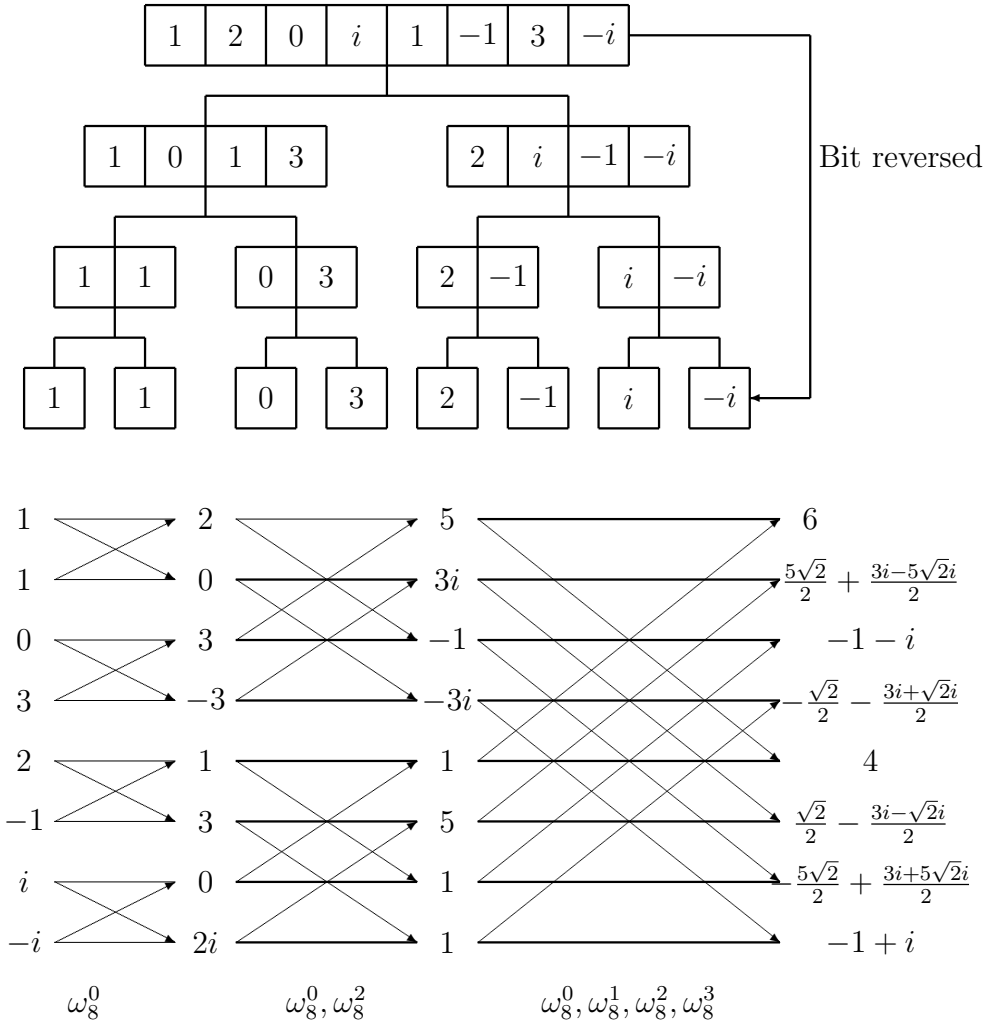


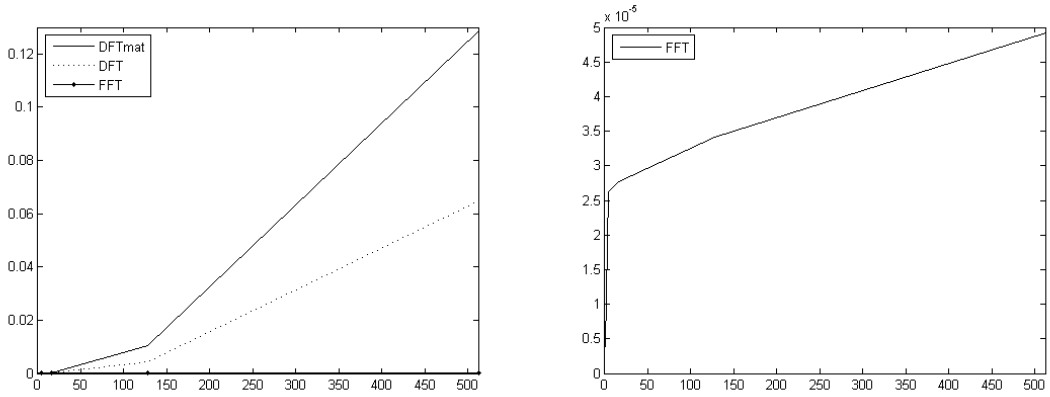
Figure 2.3: *FFT* for $N = 8$.

Example 2.5. *Let us consider the same data of the Section 1.3. In the table 2.1 we compare the times spent in the DFT and FFT algorithms.*

Length	*DFTmat (s)*	*DFT (s)*	*fft (s)*
4	$6.4813 \cdot 10^{-5}$	$3.9111 \cdot 10^{-5}$	$2.6260 \cdot 10^{-5}$
16	$1.5477 \cdot 10^{-4}$	$1.0392 \cdot 10^{-4}$	$2.7657 \cdot 10^{-5}$
128	0.0103	0.0042	$3.4083 \cdot 10^{-5}$
512	0.1286	0.0648	$4.9168 \cdot 10^{-5}$

Table 2.1: Comparison between *DFT* and *FFT* times.

Figure 2.4 represents the time spent in the differents algoritms: DFTmat, DFT and FFT. We represent only the FFT inside a graph because its values cannot be seen in the first graph.



(a) DFTmat, DFT and FFT time graph

(b) FFT time graph

Figure 2.4: Comparison of DFTmat, DFT and FFT algorithms

2.3 Code implementation

2.3.1 *FFT* Matlab implementation

Auxiliar functions: *padding1D* and *iterative_twiddle*.

Code

```
function l_pad = padding1D(n)

l_pad = 2;
while n>l_pad
    l_pad = l_pad*2;
end
```

Code

```
function y = iterative_twiddle(x,N,w)

y = bitrevorder(x);
for i=1:log2(N)

    d = pow2(i);
    ktwiddle = 1;

    for k=1:d/2
       for m=k:d:N
            t = w(ktwiddle)*y(m+d/2);
            z = y(m);
            y(m) = z + t;
            y(m+d/2) = z - t;
        end
        ktwiddle = ktwiddle + pow2(log2(N)-i);
```

```
    end
end
```

Main function *FFT_iterative*

Code

```
function FFT = FFT_iterative(signal)

N = length(signal);

%% Check N is a power of 2 %%
if isinteger(log2(N))
    N_pad = N;
else
    N_pad = padding1D(N);
end

w = twiddle(N_pad);
signal_pad = zeros(1,N_pad);
signal_pad(1:N) = signal;
FFT = iterative_twiddle(signal_pad,N_pad,w);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%    EXECUTION
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% z = [1 1 1 i 1 -1 1 -i];
%% tic;FFT_iterative(z);t=toc
%% tic;fft(z);t=toc
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

2.3.2 2D *FFT* Matlab implementation

The main program uses the same auxiliary functions as the one dimensional.

Code

```
function fft_2 = FFT2_iterative(im)

[rows,cols] = size(im);

%% Check N is a power of 2 %%
if isinteger(log2(rows))
    rows_pad = rows;
else
    rows_pad = padding1D(rows);
end
if isinteger(log2(cols))
    cols_pad = cols;
else
    cols_pad = padding1D(cols);
end
%% Pad image %%
im_pad = zeros(rows_pad,cols_pad);
im_pad(1:rows,1:cols) = im;

wc = twiddle(cols_pad);
wr = twiddle(rows_pad);

 for i=1:rows_pad
     fft(i,:) = iterative_twiddle(im_pad(i,:),cols_pad,wc);
 end
for j=1:cols_pad
    fft_2(:,j) = iterative_twiddle(im_pad(:,j),rows_pad,wr);
```

```
end
```

2.3.3 *FFT* C++ implementation

Twiddles function. In this function we calculate the ω_N values, so we only have to calculate the elements till $N/2$.

Code

```
void
        FFT::Compute_TWIDDLES(int N)
{

        for(int k = 0; k < N/2; k++)
        twiddles[k] = polar(1.0, -2*pi*(k/(double)N));

}
```

Bit reversal function. This function reorders the vector. First, we compute the positions of the vector in bits, and then we swap the bit string. For example, the position 1 in the vector corresponds with the bit string 001, and it becomes the bit string 100. It can be seen better in the Figure 2.3.

Code

```
void
        FFT::BitReversal(Mat &mat,int p)
{
        unsigned long x = mat.cols;
        unsigned long h = 0;
        int i = 0;
        int k, k_aux;
        complex<double> temp;
```

```
        int *v;
        v = new int[mat.cols];
        Mat m_aux = mat.clone();

        for(k = x-1; k >= 0 ;k--){
                k_aux = k;
                for(h = i = 0; i < p; i++){
                        h = (h << 1) + (k_aux & 1);
                        k_aux >>= 1;
                }
                v[k] = h;

        }
        for(int i=0; i<mat.cols;i++)
                mat.at<complex<double> >(0,i) =
                m_aux.at<complex<double> >(0,v[i]);

        delete[] v;

}
```

FFT function

Code

```
void
        FFT::Compute_FFT()
{
        complex<double> z(0.0,0.0), t(0.0,0.0);
        int r = 0;
        int d;
        int ktwiddle = 0;
        int p = 0;
```

```
        // Compute twiddle vector //
        Compute_TWIDDLES(cols);

        // Log2 of cols //
        unsigned int cols_aux = cols;

        while ( cols_aux >>= 1) // unroll for more speed...
        {
          r++;
        }

        // bitreversal //
        BitReversal(FFT_fft,r);

        for(int i=1; i <= r; i++){
         d = 1 << i;
         ktwiddle = 0;
         for(int k = 0;k < d/2; k++){
          for(int j = k; j < cols; j+=d){
           t = (FFT_fft.at<complex<double> >(0,j + d/2)).operator
                 *=(twiddles[ktwiddle]);
                 z = FFT_fft.at<complex<double> >(0,j);
                 FFT_fft.at<complex<double> >(0,j) = z + t;
                 FFT_fft.at<complex<double> >(0,j + d/2) = z - t;

         }
                 p = 1 << (r-i);
                 ktwiddle += p;
         }
  }
}
```

2.3.4 2D *FFT* C++ implementation

The bit reversal function and twiddles function used in the code are the same as the implemented in one dimension. See Section 2.3.3.

*FFT*2D function.

Code

```
void
        FFT::Compute_FFT2D( )
{
/// FFT1D for all the rows ///

        // Compute twiddle vector //
        Compute_TWIDDLES(cols);
        for(int i=0; i< rows; i++)
                Compute_FFT1D_r(FFT_fft.row(i));


/// FFT1D for all the columns ///

        // Compute twiddle vector //
        Compute_TWIDDLES(rows);
        for(int i=0; i< cols; i++)
                Compute_FFT1D_c(FFT_fft.col(i));

}
```

1D *FFT* function for rows.

Code

```
void
        FFT::Compute_FFT1D_r(Mat& fft_r )
{
        complex<double> z(0.0,0.0), t(0.0,0.0);
        int r = 0;
        int d;
        int ktwiddle = 0;
        int p;


        // Log2 of cols //
        int cols_aux = cols;

        while ( cols_aux >>= 1)
        {
          r++;
        }

        // bitreversal //
        BitReversal_1xN(fft_r,r);

        for(int i=1; i <= r; i++){
                d = 1 << i;
                ktwiddle = 0;
                for(int k = 0;k < d/2; k++){
                 for(int j = k; j < cols; j+=d){

                        t = twiddles[ktwiddle]
                                *fft_r.at<complex<double> >(0,j + d/2);
```

```
                    z = fft_r.at<complex<double> >(0,j);
                    fft_r.at<complex<double> >(0,j) = z + t;
                    fft_r.at<complex<double> >(0,j + d/2) = z - t;


                }p = 1 << (r-i);
                ktwiddle += p;

            }
        }



}
```

1D *FFT* function for columns.

Code

```
void
        FFT::Compute_FFT1D_c(Mat& fft_c)
{
        complex<double> z(0.0,0.0), t(0.0,0.0);
        int r = 0;
        int d;
        int ktwiddle = 0;
        int p;


        // Log2 of rows //
        int rows_aux = rows;

        while ( rows_aux >>= 1)
```

```
        {
          r++;
        }

        // bitreversal //
        BitReversal_Nx1(fft_c,r);

        for(int i=1; i <= r; i++){
         d = 1 << i;
         ktwiddle = 0;
         for(int k = 0;k < d/2; k++){
          for(int j = k; j < rows; j+=d){

                  t = twiddles[ktwiddle]
                  *fft_c.at<complex<double> >(j + d/2,0);
                  z = fft_c.at<complex<double> >(j,0);
                  fft_c.at<complex<double> >(j,0) = z + t;
                  fft_c.at<complex<double> >(j + d/2,0) = z - t;

                }
                p = 1 << (r-i);
                ktwiddle += p;
         }
 }
}
```

2.3.5 GUI for *FFT2D*

A graphical User Interface (*GUI*) has been implemented to improve the execution. In this section we explain how to run this *GUI* to compute the 2-dimensional *FFT* for an image.

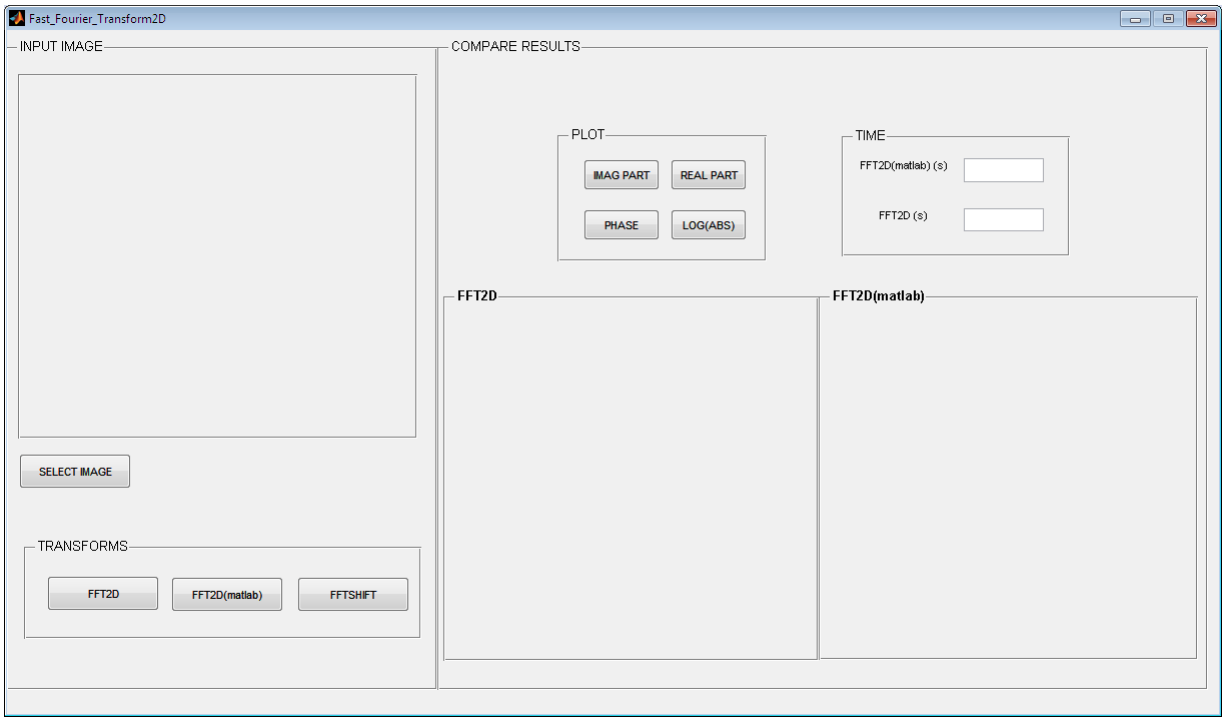


Figure 2.5: Two dimensional FFT GUI.

We can see in Figure 2.5 the diferent parts of the GUI:

- Input Image,

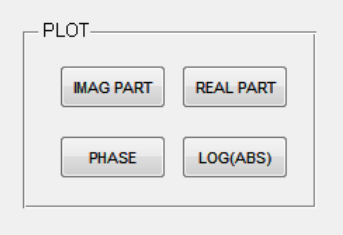
If you press *SELECT IMAGE*, then a file dialog will appear where you can choose the input image to be processed.

- Transforms,

We can compute the *Matlab FFT* and our algorithm. The button *FFTSHIFT* is used to centrate the result, and then you can see the simetry of the *FFT*.

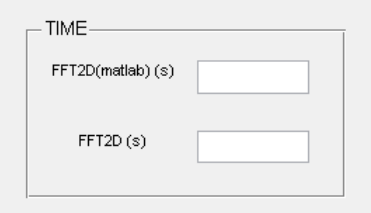
- Plot,

There are four different plots. The real part, the imaginary part, the phase, which represent the angle, and the default plot logarithm of the spectrum.



- Time.

Display the time spent in each algoritm in seconds.
They appear after the images are plotted.



Example 2.6. *In Figure 2.6 we compute the FFT of a RGB image and in Figure 2.7 we compute the FFT of a gray image, the size of the images is* 512×512*. In both results we plot the shifted spectrum of them.*

We can also see the time spent in Matlab and in our algorithm.

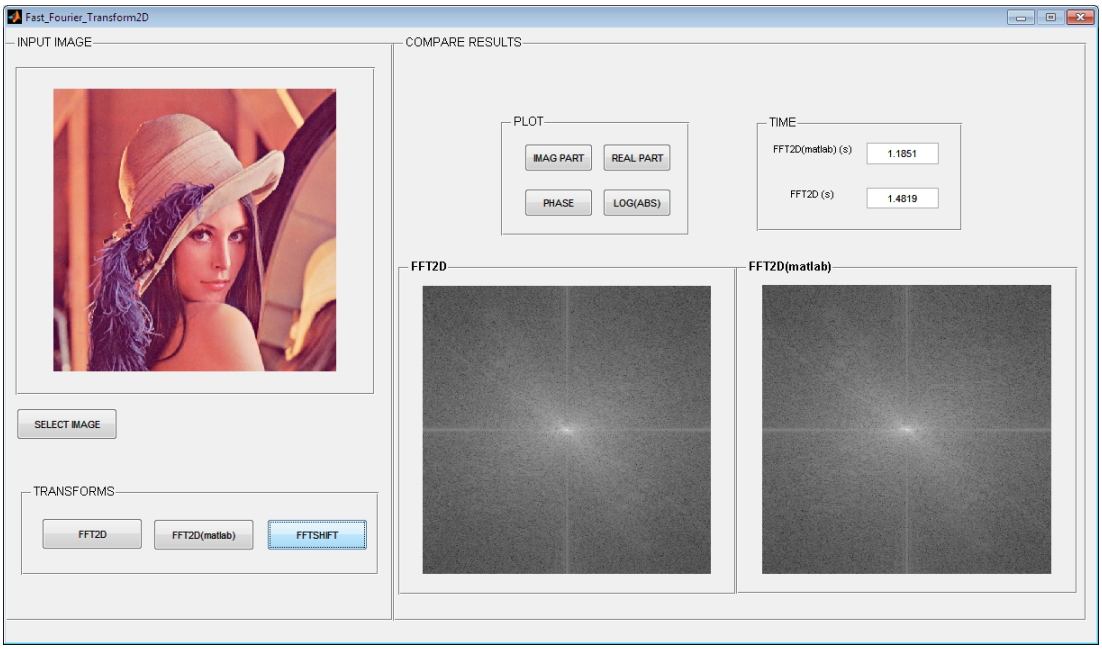


Figure 2.6: Lena.png.

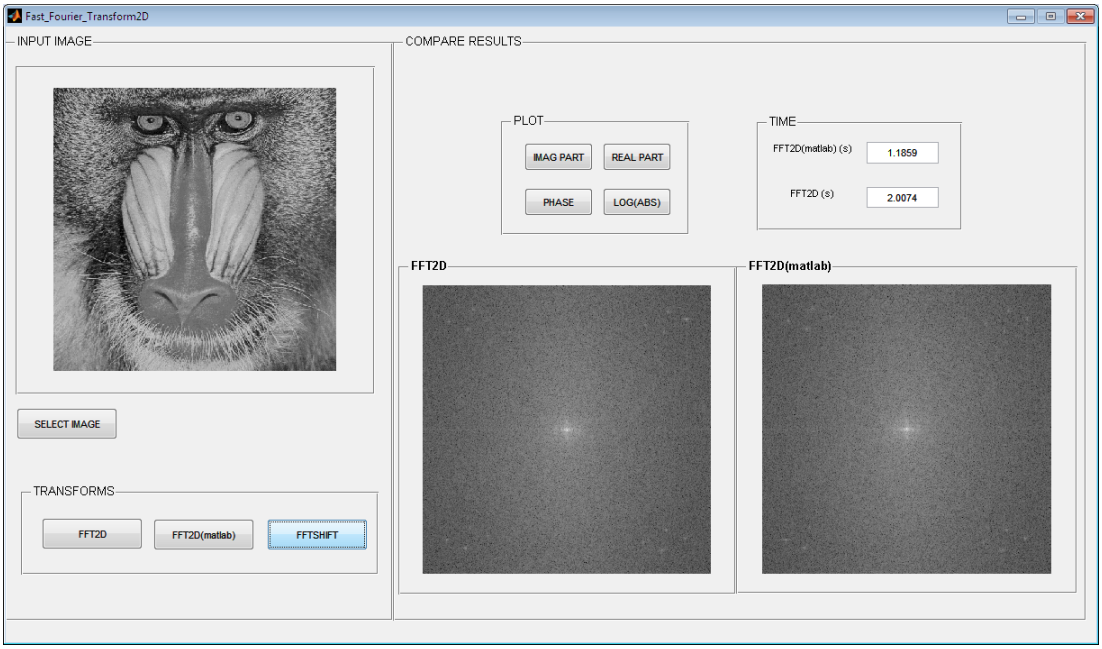


Figure 2.7: Mandril.png.

Chapter 3

Intensity Transformations and Filtering

Spatial domain techniques operate directly on the pixels of an image. Given an image of dimensions $M \times N$ we have

$$I_2(x, y) = T[I_1(x, y)], \quad (x, y) \in M \times N,$$

where I_1 is the input image, I_2 is the output image and T is the transformation on I_1.

Definition 3.1. *A spatial neighborhood about a point (x, y) is a square or a rectangular region centered at (x, y).*

Then, the operator T is applied in each location of the image in the neighborhood selected, to perform the output value of $I_2(x, y)$. See [2].

3.1 Intensity Transformations

The simplest form of the transformation T is when the size of the neighborhood is 1×1, and is called *intensity transformation* of the image, because the output value depends only on the value of the input.

1. Adjust image function.

   This function changes the input range of values of the image by a specified range.

$$T[I(x,y)] = \begin{cases} low\_out & \text{if } I(x,y) < low\_in \\ low\_out + (I(x,y) - low\_in)\frac{high\_out - low\_out}{high\_in - low\_in} & \text{if } I(x,y) \in C \\ high\_out & \text{if } I(x,y) > high\_in \end{cases}$$

   where $C = [low\_in, high_in]$.

   Here is the implementation of such transformation.

Code

```
function out_image = adimage(I,range_in, range_out,gamma)
%%%%%%%% INPUTS %%%%%%%%%%%%%%%%%%%%
disp(nargchk(1, 4, nargin));
if nargin == 1
    gamma = 1;
    range_in = [min(min(I)) max(max(I))];
    range_out = [0 1];
elseif nargin == 3
    gamma = 1;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[rows, cols, ch] = size(I);
% Check that the ranges are not empty
if isempty(range_in)
    range_in = [0 1];
end
if isempty(range_out)
    range_out = [0 1];
end

low_in = range_in(1);
```

```
high_in = range_in(2);
low_out = range_out(1);
high_out = range_out(2);

%%%%% Transform %%%%%
%% All the c_i are auxiliar matrix %%
c1 = (I <= low_in);
c2 = (I >= high_in);
c3 = c1|c2;
c4 = ( low_in < I < high_in);
out = (I-low_in).*((high_out-low_out)/(high_in-low_in)) + low_out;
c5 = c4.*out;
c6 = ~c3;
out_image = c6.*out + c1.*low_out + c2.*high_out;
out_image = out_image.^gamma;
figure;
subplot(1,2,1);imshow(I);title('Original image');
subplot(1,2,2);imshow(out_image);title('Adjust image');
```

Example 3.2. *Execution of the program with differents ranges and images.*

(a) In Figure 3.1 if we get all the possible intensity values as input range and transform it to a smaller one, then the output image has less contrast and the gray values are very similar.

(b) In Figure 3.2 the output range is reversed, so we obtain its corresponding negative image.

(c) In Figure 3.3 the contrast of the output image is achieved by selecting the intermediate intensity values and arranging them to $[0\ ,\ 1]$.

Figure 3.1: Input range: [0 1] & output range: [.3 .7].

Figure 3.2: Input range: [0 1] & output range: [.7 .3].

Figure 3.3: Input range: [.3 .7] & output range: [0 1].

2. Logarithmic transformation.

   It is defined by

$$T[I(x,y)] = c \cdot \log(1 + double(I(x,y))),$$

   where $double(I(x,y)) \in [0,1]$.

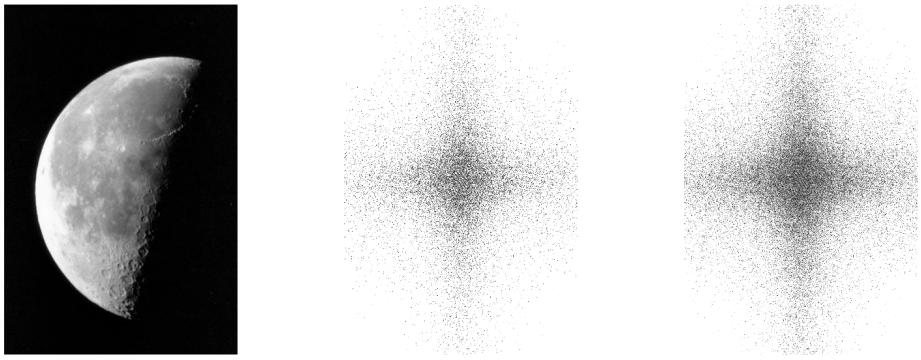
   The application of this transform is basically compression of the ranges. For example, it applies when we have to represent the spectrum of the Fourier Transform, whose values are usually high.

   Here is the implementation of such transformation.

Code

```
function out = logarithmic(I,c)

disp(nargchk(1, 2, nargin));
if nargin==1
    c = 1;
end

out = c.*(1 + log(I));

figure;
subplot(1,2,1);imshow(I);title('Original image');
subplot(1,2,2);imshow(out);title('Logarithmic image');
```

Example 3.3. *Suppose we want to plot the result of the Fast Fourier Transform of an image. The output is a complex image, and we will represent the absolute value of it. The problem is that there are high values that are not represented because of the range. Then, the logarithmic transform achieves to represent in the plot those high values.*

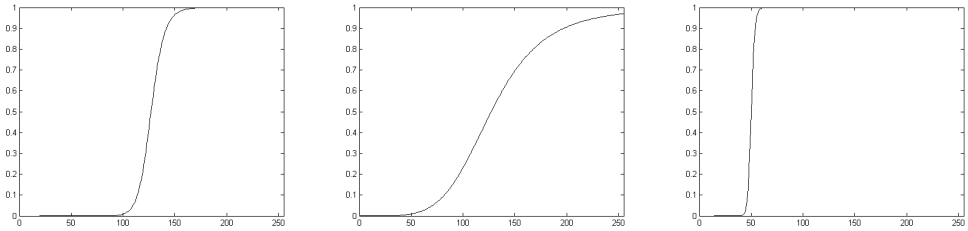


(a) Image *moon.tif*. (b) abs(*FFT(I)*). (c) $\log(1 + abs(FFT))$

Figure 3.4: Logarithmic transformation.

3. Contrast stretching transformation.

   It compresses the inputs levels lower than m into a narrow range of dark levels in the output image; and it compresses the values above m into narrow band of light levels in the output. The function for differents inputs of m and E can be seen in Figure 3.6.



(a) $m = 127 \;\; E = 20$ (b) $m = 127 \;\; E = 5$ (c) $m = 50 \;\; E = 30$

Figure 3.5: Constract stretching function

It is defined by means of the transformation:

$$T[I(x,y)] = \frac{1}{1 + (m/I(x,y))^E},$$

where E controls the slope of the function. Here is the implementation of such transformation.

Code

```
function out=contrast_stretching(I,m,E)
disp(nargchk(1, 3, nargin));
if nargin == 1
    m = (max(max(I))-min(min(I)))/2;
    E = 20;
end

out = 1./(1 + (m./I).^E);

figure;
subplot(1,2,1);imshow(I);title('Original image');
subplot(1,2,2);imshow(out);title('Contrast&Stretching image');
```

Example 3.4. *In the example below, we apply the transform to the image estatua.jpg. The values of* $m \in [0, 1]$ *because this image takes values in this range.*

(a) $m = .5 \;\; E = 20$ (b) $m = .5 \;\; E = 5$ (c) $m = .8 \;\; E = 20$

Figure 3.6: Constract stretching transformation

3.1.1 Histogram and Equalization

Histograms are used in image processing for enhancement, compression, segmentation, etc. The histogram h computes how many times an intensity value is repeated:

$$h(r_k) = n_k.$$

where r_k is the k intensity value and n_k is its corresponding absolute frequency.

When we calculate the histogram of an image it is necessary to specify how many bins (adjacent rectangles over discrete intervals) will be used; by default, the number of bins will be 256.

Equalization of the histogram is a transformation that generates an image whose intensity levels are basically equal and then all intensity value range is covered. There are many ways to compute the equalization and we will mostly consider the following:

Define

$$p_r(r_k) = \frac{n_k}{N},$$

with N all the intensity values, as the *probability density function (PDF)*. Then the transformation is given by

$$s_k = \sum_{j=1}^{k} p_r(r_j),$$

which is called the *cumulative distribution function (CDF)*.

Remark 3.5. *In the process of histogram equalization, information data is lost. So this transformation is useful for viewing the image better, but it is not convenient to use the output image for image processing later.*

The histogram function has been implemented for RGB, red-green-blue channels, color images, and gray-level images. Here is the code for each function and histogram equalization.

Histogram function of a gray-level image.

Code

```
function [h,x]=histdouble(I,bin)

disp(nargchk(1, 2, nargin));

if nargin == 1
    bin = 256;
end
if isa(I,'double')
    A=1;
    x = 0:1/255:1;
elseif isa(I,'uint8')
    A=255;
    x = 0:255;
end

[rows cols] = size(I);
h = zeros(1,256);

for i=1:rows
    for j=1:cols
        p = floor(I(i,j).*((bin-1)/A) + 1.5);
        if p>256
            p=256;
        end
        h(p) = h(p) + 1;
    end
end
```

Histogram function of a RGB image.

Code

```
function [h x] = hist_rgb(I)

[rows cols ch] = size(I);
h = zeros(256,ch);

[h(:,1),x] = histdouble(I(:,:,1));
h(:,2) = histdouble(I(:,:,2));
h(:,3) = histdouble(I(:,:,3));
```

Image equalization.

Code

```
function [im_eq,h_eq,s,h1]=histogram_eq(I,h,n)

disp(nargchk(1, 3, nargin));
if nargin == 2
    n = 256;
end
if isa(I,'double')
    A=1;
elseif isa(I,'uint8')
    A=255;
end
[rows cols] = size(I);

s = zeros(1,n);
for k=1:n
    s(k) = sum(h(1:k));
```

```
end

im_eq = (s(floor(I.*((n-1)/A)+1.5))-min(s))./(rows*cols-min(s));

h_eq = histdouble(im_eq);

h1 = zeros(1,n);
for k=1:n
    h1(k) = sum(h_eq(1:k));
end
```

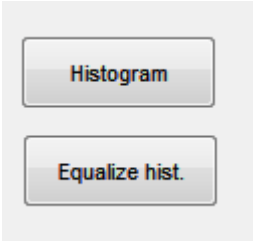
GUI of histogram equalization

In this section we explain how to run this *GUI*, enumerating its parts:

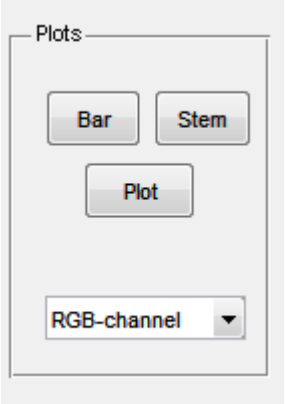
- Input Image.
- Histogram and equalization.
- Plot histograms.
- *PDF* and *CDF* plots.
- Equalize image.

If you press *SELECT IMAGE*, then a file dialog will appear where you can choose the input image to be processed.

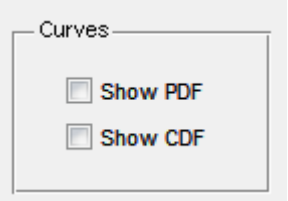
Each button compute the histogram and the equalized histogram and plot them.



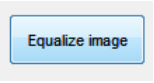
There are 3 different ways to plot the histograms: with bars, stems or with a simple plot. The menu list of RGB-channel is only for images of 3 channels, then you can choose which one will be plot.



Check the corresponding box to plot the PDF and CDF defined as above.



This button displays the equalized image.



Example 3.6. *Let us consider lena.png and estatua.jpg to show the image equalization of images of different number of channels. In Figure 3.7 we compute it for a gray-level image, and in Figure 3.8 we compute it for a color image.*

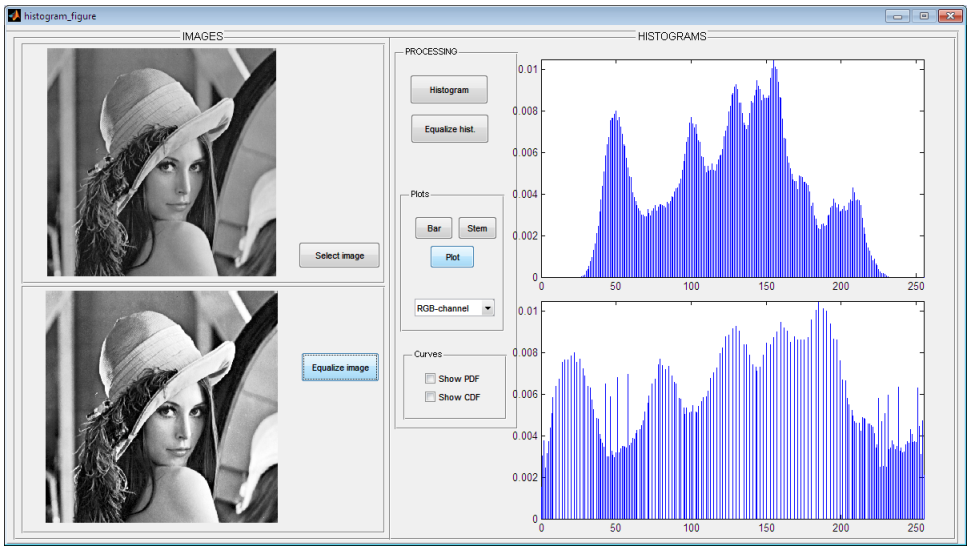


Figure 3.7: Histogram of a gray-level image.

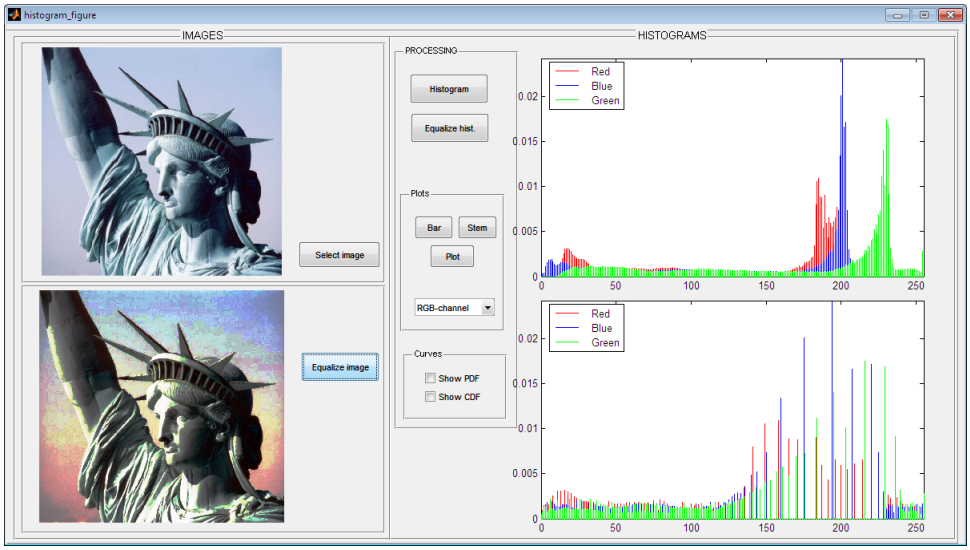


Figure 3.8: Histogram of an RGB image.

3.2 Filtering

3.2.1 Spatial Filter Domain

Definition 3.7. *Given a neighborhood as defined in last section, we define linear spatial filtering as the multiplication of each pixel in the neighborhood by a corresponding coefficient and summing the results to obtain a new value at pixel* (x, y).

The coefficients are arranged as a matrix called *mask*, *filter*, *filter mask*, *kernel*, *template* or *window*. The most common are the first three. Figure 3.9 shows how to process the mask filter accross the image.

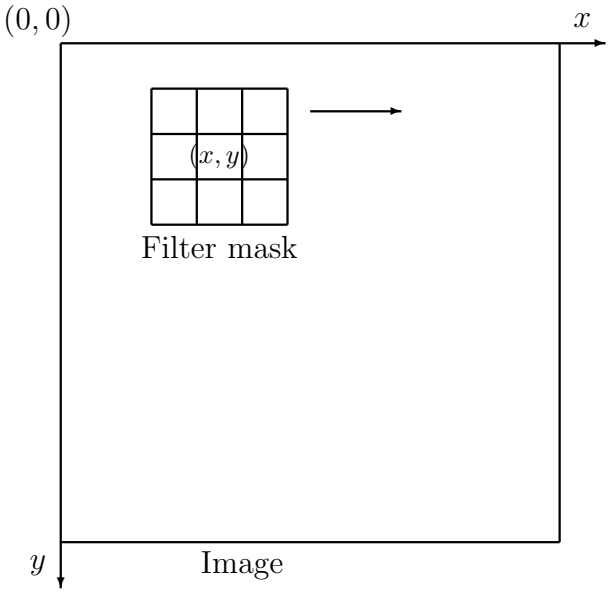


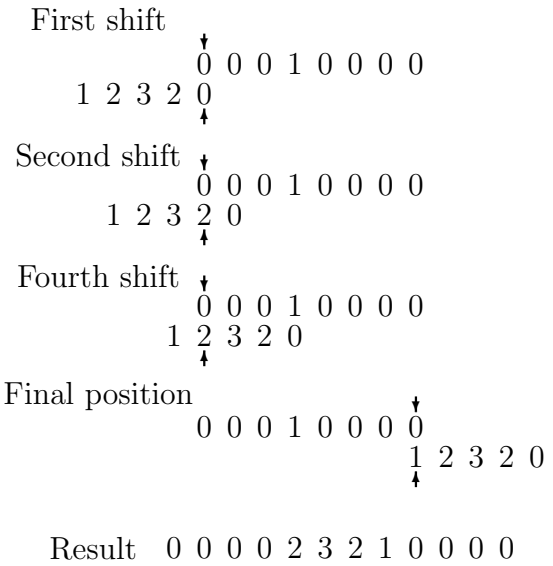
Figure 3.9: Filtering image

Definition 3.8. *1. Correlation is a process of passing the mask* w *by the image.*

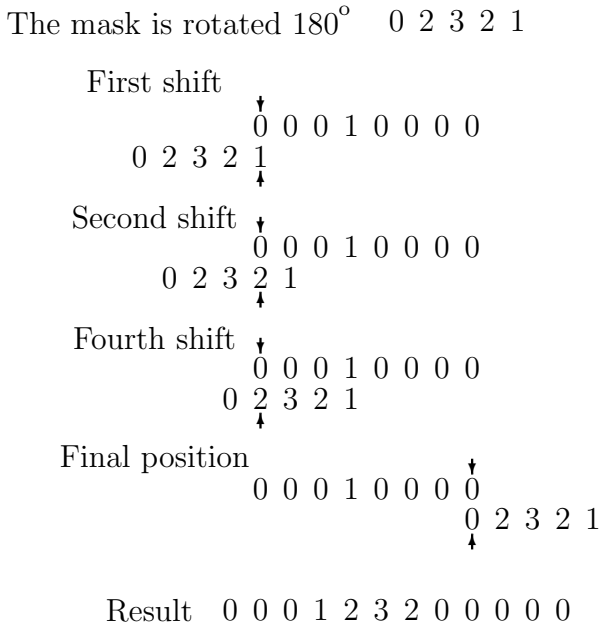
2. Convolution is a process of passing the mask w *rotated* 180^o *by the image.*

Example 3.9. *Consider a one dimensional signal* $f = 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0$ *and a mask* $w = 1\ 2\ 3\ 2\ 0$.

1. Correlation



2. Convolution



3.2.2 Boundary options in the filter

We have to extend the image boundaries because when we apply the filter, we put the filter center c in each pixel image. It can be seen in Figure 3.10.

PADDING: The boundaries are extended by the value p. In our case, the padding is made with zeros.
REPLICATE: The size of the image is extended by replicating its values in the borders.
REFLEXION: The size of the image is extended by symmetry followed by a mirror-reflecting across its borders.

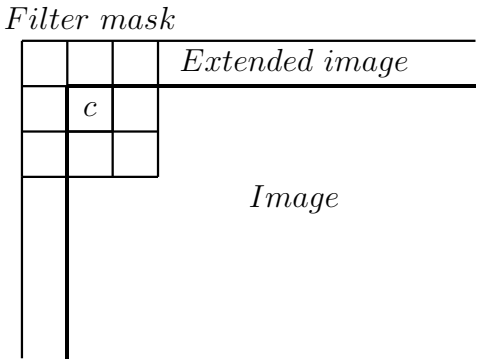


Figure 3.10: Image boundaries

3.2.3 Spatial filters

AVERAGE: A rectangular averaging filter of dimension $r \times c$, where r is the number of rows and c the number of columns.
DISK: A circular averaging filter of dimension $(2r + 1) \times (2r + 1)$, where r is the radius.
GAUSSIAN: A Gaussian filter of size $r \times c$ and standard deviation $\sigma \geq 0$.
LAPLACIAN: Laplacian filter of size 3×3 and $\alpha \in [0, 1]$.
LOG: Laplacian of Gaussian *(LoG)* filter of size $r \times c$ and standard deviation $\sigma \geq 0$.
PREWITT: Prewitt mask of size 3×3 that approximates a vertical gradient.
SOBEL: Sobel mask of size 3×3 that approximates an horitzontal gradient.

In Figure 3.11 and 3.12 the Gaussian, Laplacian, LoG and Prewitt, Sobel spatial filters are represented respectively.

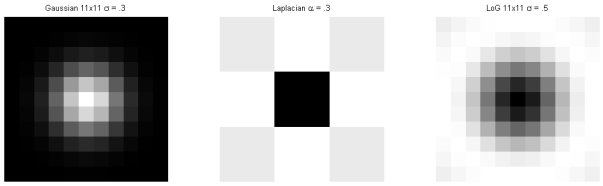


Figure 3.11: Gaussian, Laplacian and LoG Filters.

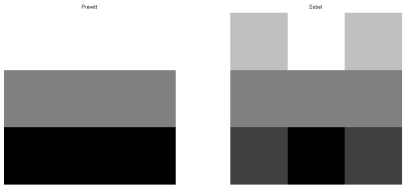


Figure 3.12: Gradient Filters.

Here is the code of the convolution in the spatial domain of an image.

Code

```
function conv2D = conv2D(image,filter,method)


% Convert image to double (if the image is double, it doesn't change)
image = im2double(image);
```

```
[rows,cols]=size(image);
[rows_f,cols_f]=size(filter);


r = floor(rows_f/2);
c = floor(cols_f/2);

newimage = zeros(rows+2*r,cols+2*c);
newimage(r+1:end-r,c+1:end-c) = image;

if (strcmp(method,'period'))
    % Fill newimage
    newimage(1:r,:) =
    [image(end-r+1:end,end-c+1:end)
    image(end-r+1:end,:)
    image(end-r+1:end,1:c)];
    newimage(end-r+1:end,:) =
    [image(1:r,end-c+1:end) image(1:r,:) image(1:r,1:c)];
    newimage(r+1:end-r,1:c) = image(:,end-c+1:end);
    newimage(r+1:end-r,end-c+1:end) = image(:,1:c);
    % Convolution
    for i=1:rows
        for j=1:cols
           conv2D(i,j) = sum(sum(newimage(i:i+2*r,j:j+2*c).*filter));
        end
    end
elseif (strcmp(method,'reflexion'))
    % Fill newimage
    newimage(1:r,:) =
    [image(r:-1:1,end:-1:end-c+1)
    image(r:-1:1,:)
    image(r:-1:1,end:-1:end-c+1)];
```

```
    newimage(end-r+1:end,:) =
    [image(end:-1:end-r+1,c:-1:1)
    image(end:-1:end-r+1,:)
    image(end:-1:end-r+1,end:-1:end-c+1)];
    newimage(r+1:end-r,1:c) = image(:,c:-1:1);
    newimage(r+1:end-r,end-c+1:end) = image(:,end:-1:end-c+1);
    % Convolution
    for i=1:rows
        for j=1:cols
            conv2D(i,j) = sum(sum(newimage(i:i+2*r,j:j+2*c).*filter));
        end
    end
elseif(strcmp(method,'padding'))
    % Convolution
    for i=1:rows
        for j=1:cols
            conv2D(i,j) = sum(sum(newimage(i:i+2*r,j:j+2*c).*filter));
        end
    end
else
    'method = period,reflexion,padding'
    return
end

% figure;
% subplot(1,2,1);imshow(image,[]);title('Image');
% subplot(1,2,2);imshow(conv2D,[]);title('Convolution 2D');
```

3.2.4 Frequency Domain

In the frequency domain we use Theorem 1.9.

$$f(x,y) * h(x,y) \Longleftrightarrow \hat{f}(u,v) \cdot \hat{h}(u,v),$$

where f is the image and h is the filter.

Remark 3.10. *1. If filter size is not equal to image size, then padd with zeros to achieve the same size.*

2. First we compute the convolution in the frequency domain and then we apply the IFFT to the result.

Here is the code for the convolution in the frequency domain.

Code

```
function conv2D = conv2D_f(I,filter)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%%  Convolution in the frequency domain
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[rows cols] = size(I);
[rows_f cols_f] = size(filter);
rows_pad = padding1D(rows);
cols_pad = padding1D(cols);

I_pad = zeros(rows_pad,cols_pad);
I_pad(1:rows,1:cols) = I;

filter_pad = zeros(rows_pad,cols_pad);
filter_pad(1:rows_f,1:cols_f) = filter;
```

```
[fft_image_r fft_image_i] = FFT2_mex.FFT(I_pad);
[fft_filter_r fft_filter_i] = FFT2_mex.FFT(filter_pad);

fft_image = complex(fft_image_r,fft_image_i);
fft_filter = complex(fft_filter_r,fft_filter_i);

conv2D_f = fft_image.*fft_filter;

[conv2D_r conv2D_i] = FFT2_mex.IFFT(conv2D_f);

conv2D1 = complex(conv2D_r,conv2D_i);
conv2D = conv2D1(floor(rows_f/2):rows+floor(rows_f/2),
floor(cols_f/2):cols+floor(cols_f/2));
```

3.2.5 GUI Image Filtering

In this section we explain how to run this *GUI* enumerating its parts:

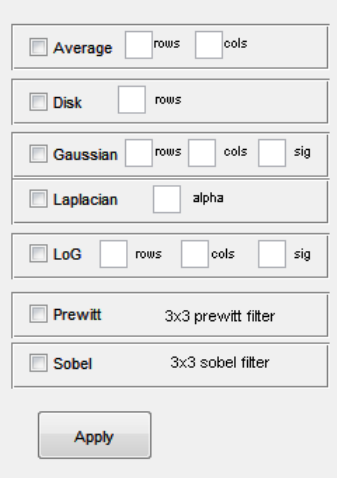
- Input Image,

- Choose parameters of the filter,

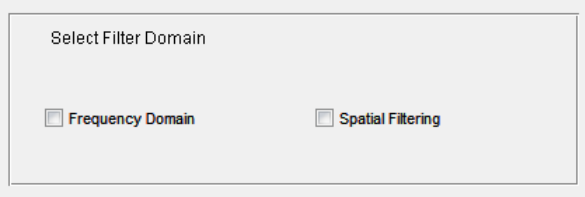
- Plot the filtered image.

If you press *SELECT IMAGE*, then a file dialog will appear where you can choose the input image to be processed.

Introduce all the parameters for the filter and select it (in this order). Then press *Apply* button to compute the transformations.



Choose spatial or frequency domain.



Example 3.11. *We have added 'salt and pepper'noise (randomly white and black pixels) in the picture of lena.png. In Figure 3.13 and Figure 3.14 we can observe that the image has been improved, because these filters are called low-pass filters. But in Figure 3.15 the noise is not erased, because it is called a high-pass filter.*

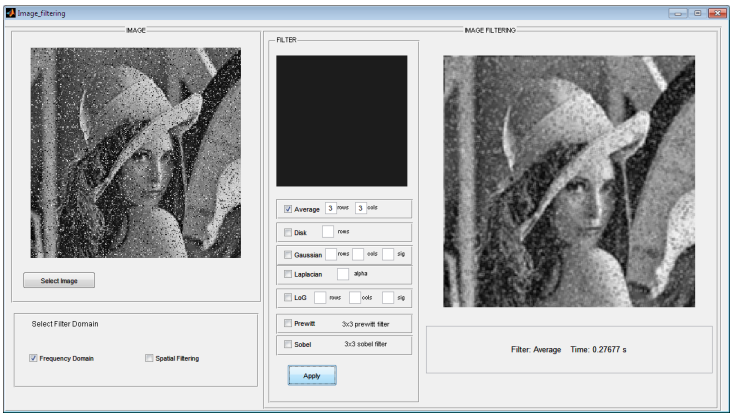


Figure 3.13: Average filter.

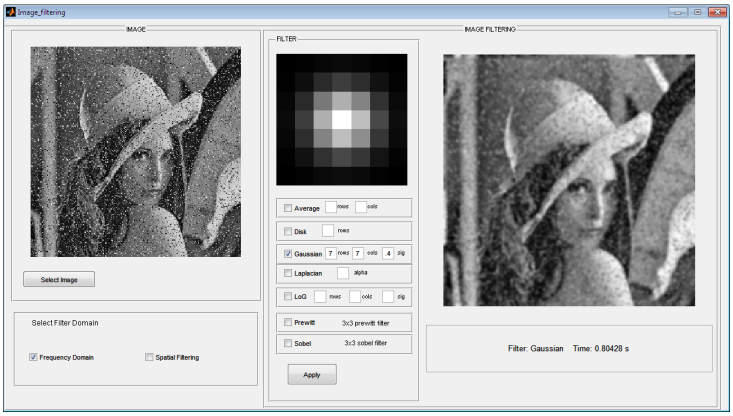


Figure 3.14: Gaussian filter.

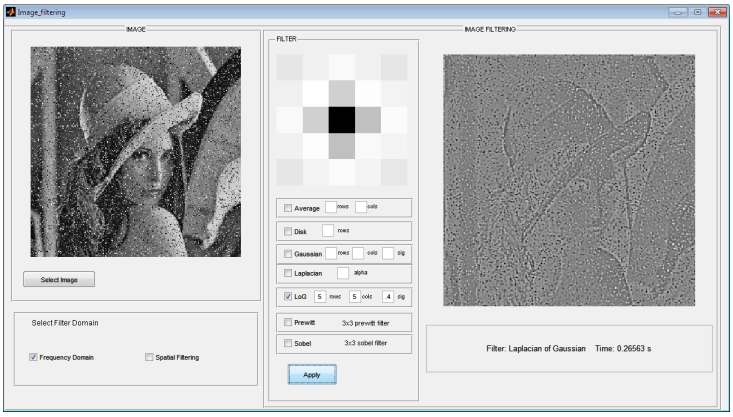


Figure 3.15: Laplacian of Gaussian filter.

Chapter 4

Morphological Image Processing

In this chapter we will introduce morphological operations in image processing, like dilation, erosion, openning, closing... See [2]. In the two first sections we will deal with binary images, and in the last section all the functions will be applied in gray level images.

4.1 Dilation and Erosion

First, we will introduce some correspondences between set theory and its equivalent with binary images.

Definition 4.1. *Reflexion of a set A is defined as:*

$$\hat{A} = \{w \mid w = -a,\ a \in A\}.$$

Traslation of a set A is defined by:

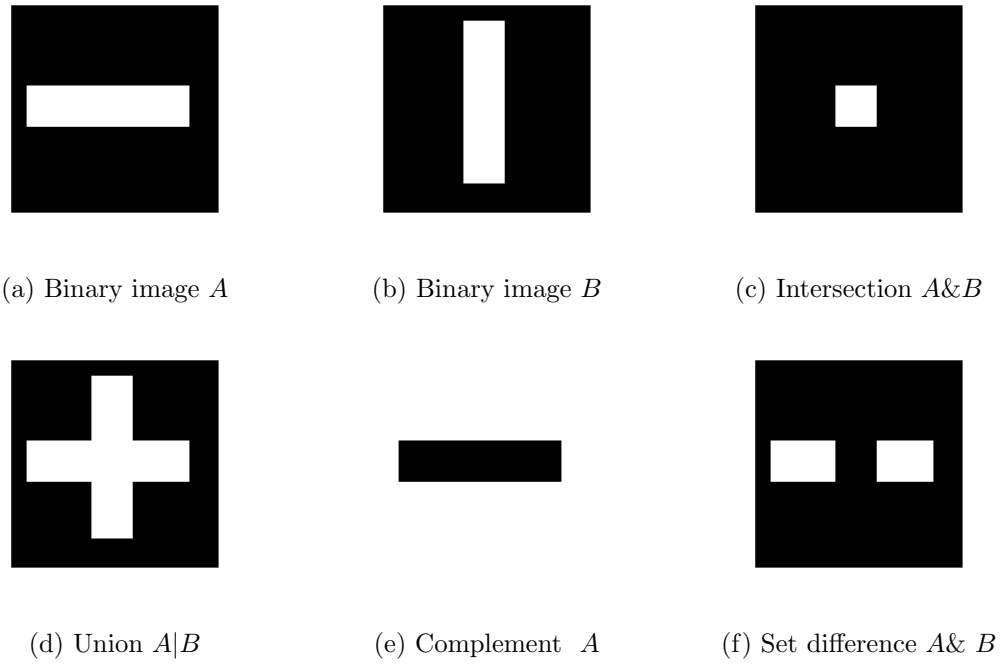
$$(A)_z = \{c \mid c = a + z,\ a \in A\}.$$

In table 4.1 we represent the equivalence between set theory and *Matlab* syntax.

Set theory	*Matlab*
$A \cap B$	$A\&B$
$A \cup B$	$A\|B$
A^c	$-A$
$A - B$	$A\& - B$

Table 4.1: Set theory & Matlab syntax

Example 4.2. *Given two binary images, A and B, calculate the intersection, union, complement and set difference.*



(a) Binary image A (b) Binary image B (c) Intersection $A\&B$

(d) Union $A|B$ (e) Complement A (f) Set difference $A\& B$

Figure 4.1

4.1.1 Dilation

Definition 4.3. *An structuring element is computationally represented as a matrix of* 0*s and* 1*s and by an origin.*

Definition 4.4. *Dilation is an operation that 'grows'or 'thickens'objects in binary images.*

$$A \oplus B = \left\{ z \mid (\hat{B})_z \cap A \neq \emptyset \right\}.$$

Dilation is a commutative operation $A \oplus B = B \oplus A$.

Example 4.5. *Given a binary image and an structuring element*

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

we calculate its dilation:

(a) Original image (b) Dilation

Figure 4.2: Dilation operation.

We can observe that the output image has increased the size of the rice. That is because dilation selects all those pixels that have intersection with the given structuring element.

We have implemented a function *structel* to calculate the most used structuring elements.

Syntax	*Description*
structel('diamond',R)	Diamond form of radius R
structel('disk',R)	Disk of radius R
structel('octagon',R)	Octagon of radius R
structel('rectangle',[M N])	Rectangle of size $M \times N$
structel('line',L)	Line of length L
structel('square',M)	Square of size $M \times M$

4.1.2 Erosion

Definition 4.6. *Erosion 'shrinks'or 'thins'objects in a binary image. It is defined as:*

$$A \ominus B = \{z \mid (B)_z \cap A^c \neq \emptyset\}$$

Example 4.7. *With the same binary image and structuring element (4.5) used in dilation we will calculate its erosion:*

(a) Original image (b) Erosion

Figure 4.3: Erosion operation.

The output image has decreased the rice size, because erosion deletes all those pixels that have not the same neighborhood as the given structuring element.

4.1.3 Implementation in Matlab

Dilation

Code

```
 function A2 = dilate(A,B)

[rows,cols]=size(A);
[rowsB,colsB] = size(B);

% Calcule the center coordinates of B
cx = ceil(colsB/2);
cy = ceil(rowsB/2);
% Calculate new image
new_image = zeros(rows+rowsB-1,cols+colsB-1);

if mod(colsB,2)~=0
    lastx = cols+colsB-1 -cx+1;
    deltax=cx-1;
else
    lastx = cols+colsB-1 -cx;
    deltax=cx;
end
if mod(rowsB,2)~=0
    lasty = rows+rowsB-1 -cy+1;
    deltay=cy-1;
else
    lasty = rows+rowsB-1 -cy;
    deltay= cy;
end

new_image(cy:lasty,cx:lastx) = A;
%% Calcule reflection of B
R_B = B(end:-1:1,end:-1:1);
%Calculate dilation
```

```
A2 = zeros(rows,cols);
for i=cy:lasty
    for j=cx:lastx
        a = sum(sum(new_image(i-cy+1:i+deltay,j-cx+1:j+deltax).*R_B));
        if a~=0
            A2(i-cy+1,j-cx+1)=1;
        end
    end
end
```

Erosion

Code

```
  function A2=erode(A,B)

[rows,cols]=size(A);
[rowsB,colsB] = size(B);
% Calcule the center coordinates of B
cx = ceil(colsB/2);
cy = ceil(rowsB/2);
% Calculate new image
new_image = zeros(rows+rowsB-1,cols+colsB-1);
if mod(colsB,2)~=0
    lastx = cols+colsB-1 -cx+1;
else
    lastx = cols+colsB-1 -cx;
end
if mod(rowsB,2)~=0
    lasty = rows+rowsB-1 -cy+1;
else
    lasty = rows+rowsB-1 -cy;
end
```

```
new_image(cy:lasty,cx:lastx) = A;
%Calculate
sumel=sum(sum(B));
A2 = zeros(rows,cols);

for i=cy:lasty
    for j=cx:lastx
        a = sum(sum(new_image(i-cy+1:i+cy-1,j-cx+1:j+cx-1).*B));
        if a == sumel
            A2(i-cy+1,j-cx+1) = 1;
        end
    end
end
```

4.2 Opening, Closing and Hit-or-Miss

In practice, in image processing we combine dilations and erosions with different structuring elements. We enhance the most common combinations: opening, closing and hit-or-miss.

4.2.1 Opening

Definition 4.8. *The morphological operation* openning *of the image A by the element B is denoted by $A \circ B$. It is defined as:*

$$A \circ B = (A \ominus B) \oplus B.$$

This operation eliminates the regions that do not contain the structuring element, smooth the countours and break weak connections between objects.

4.2.2 Closing

Definition 4.9. *The morphological operation of* closing *is defined as:*

$$A \bullet B = (A \oplus B) \ominus B.$$

This operation smooths the countours, and fill the holes that are smaller than the structuring elements.

Example 4.10. *Given a fingerprint image with noise, first we apply opening to delete noisy dots. But this operation produces holes in the fingerprint. Then, if we apply the closing operation in this output image we obtain the desire result.*

(a) Fingerprint with noise (b) Opening (c) Opening followed by Closing

Figure 4.4: Opening and Closing.

4.2.3 Hit-or-Miss

This transformation is useful for identifying configurations of pixels like isolated pixels and endings of lines.

Definition 4.11. *The* Hit-or-Miss *of the set A by the element $B = (B_1, B_2)$ is defined as:*

$$A \otimes B = (A \ominus B_1) \cap (A^c \ominus B_2).$$

The *Hit-or-Miss* consists of all pixels which are in B_1 (*hit*), and those that have not pixels in B_2 (*miss*).

4.2.4 Implementation in Matlab

We only present the implementation of *Hit-or-Miss* because *opening* and *closing* operations are form using *dilations* and *erosions*.

Hit-or-Miss

Code

```
function [A2,C2]=HitorMiss(A,B1,B2)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%%      Hit-or-Miss transformation
%%      Description:
%%      Structuring elements (B1,B2)
%%      HoM:=erode(A,B1) intersection erode(Ac B2)
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% In general, supose that B has odd dimension
A=im2double(A);
%% Image A must be a binary image %
A = im2bw(A);
% Calculate A supplementary
A_c = ~A;

C1 = erode(A,B1);
C2 = erode(A_c,B2);

% Calculate the intersection between C1 & C2:
A2 = C1&C2;
```

We introduce *bottom-hat* and *top-hat*. These are operations that extract small elements and details from a given image.

Bottom-hat returns the difference between its closing by a structuring element and the image.

Code

```
function A2 = bothat(A,B)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%% Bottom hat: closing and subtracting the image
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

A1 = close(A,B);
A2 = imsubtract(A1,A);
```

Top-hat returns the difference between the image and its opening by some structuring element.

Code

```
function A2 = bothat(A,B)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%% Top hat: opening and subtracting the image
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

A1 = open(A,B);
A2 = imsubtract(A1,A);
```

4.2.5 Lookup Tables

A lookup table is a data structure used to replace a runtime computation with a simpler array indexing operation. In image processing the nomenclature is LUT.

The technique is to precompute the output pixel value for every possible neighborhood configuration and then store the answer in a table for later use. For example, we have $2^9 = 512$ different 3×3 pixels configurations in a binary image.

To make the LUT we have to assign a unique value to each configuration, we will multiply each element of our neighborhood with the following matrix

$$\begin{matrix} 1 & 8 & 64 \\ 2 & 16 & 128 \\ 4 & 32 & 256 \end{matrix}$$

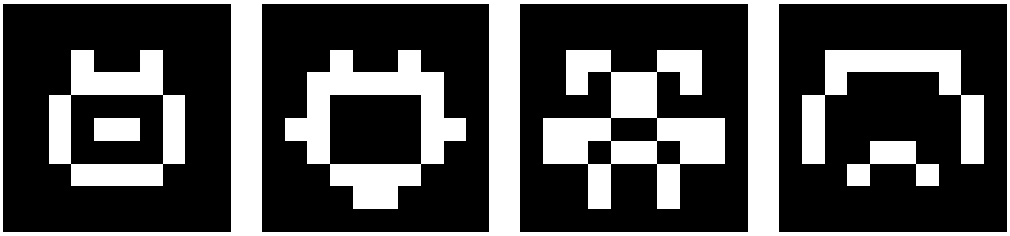
Example 4.12. *Let us consider the* 3×3 *neighborhood*

$$\begin{matrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \end{matrix}$$

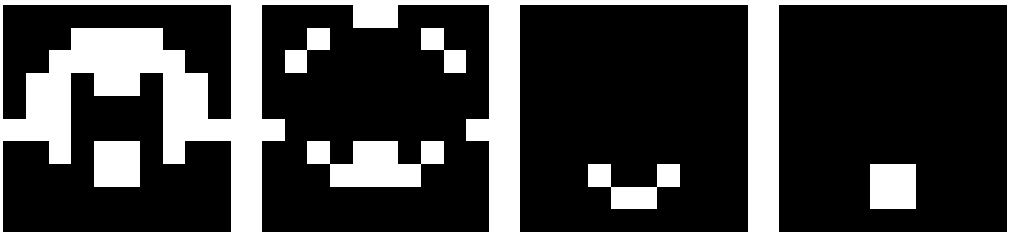
the value that will represent this matrix uniquely is perform as follows: $1(1)+2(1)+4(1)+8(1)+16(0)+32(0)+64(0)+128(1)+256(1) = 399$.

We will implement two functions: *makelookuptab* and *applylookuptab*. The first one constructs a lookup table and the last one processes the binary images using this lookup table.

Example 4.13. *The lookup table is contructed next by calling makelut with a function handle to conwaylaws*



(a) Binary image (b) First apply (c) Second apply (d) Third apply



(e) Fourth apply (f) Fiveth apply (g) Sixth apply (h) Seventh apply

Figure 4.5

4.2.6 Implementation in Matlab

Auxiliar function *matrix3x3*. Here, we create all the 3×3 matrices of $1's$ and $0's$, there are 2^9 different matrices.

Code

```
function m=matrix3x3( )

m = zeros(pow2(9),9);

m(1,:)= zeros(1,9);

for i=2:512
    m(i,:) = sum1(m(i-1,:));
end
```

Auxialiar function *sum1*. It calculates the value for each matrix in the above function *matrix3x3*.

Code

```
function sum = sum1(v)

sum = v;
k=0;
while v(end-k) == 1
    sum(end-k)=0;
    k=k+1;
end
if k==0
    sum(end)=1;
else
    sum(end-k)=1;
end
```

Make lookup table.

Code

```
function [lut,m]=makelookuptab(handle,n)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%%  Look up table
%%  First argument is the call of a function & n is the neighborhood (nxn)
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Check handle is a function handle
if isa(handle, 'function_handle');
else
    'First argument is not a function handle'
```

```
    return
end
if n==2
    lut = zeros(pow2(4),2,2,2);
    m = matrix2x2( );

    for i=1:pow2(4)
        lut(i,1) = handle(reshape(m(i,:),[2,2]));
        lut(i,2,:,:) = reshape(m(i,:),[2,2]);
    end
elseif n==3
    lut = zeros(pow2(9),1);
    m = matrix3x3( );

    for i=1:pow2(9)
        lut(i) = handle(reshape(m(i,:),[3,3]));
    end
else
    'Second argument must be 2 or 3'
    return
end
```

Apply lookup table.

Code

```
function fnew=applylookuptab(f,lut)

[r,c]=size(lut);
aux = f;
if r == 16
    m = matrix2x2();
    size(m)
```

```
    for i=1:r
        if lut(i)==1
            fnew = mask(aux,reshape(m(i,:),[2,2]));
            aux = fnew;
        end
    end
else
    m = matrix3x3();
    for i=1:r
        if lut(i)
            fnew = mask(aux,reshape(m(i,:),[3,3]));
            aux = fnew;
        end
    end
end
```

4.3 Morphological operations in gray images

All the binary morphological operations, except Hit-or-Miss transform, have extensions to gray images. In this section, we will work with dilation, erosion, opening and closing. Finally we will introduce how to obtain the gradient of the image with some of these operations.

4.3.1 Dilation and Erosion

Definition 4.14. *The gray-scale dilation of an image by a structuring element, is denoted by*

$$(I \oplus B)(x,y) = max\left\{I\left(x-x^{'},y-y^{'}\right)+B\left(x^{'},y^{'}\right)|(x^{'},y^{'})\in B\right\},$$

where I is the image and B is the structuring element. We assume that the value $I(x,y)=-\infty$ outside the image.

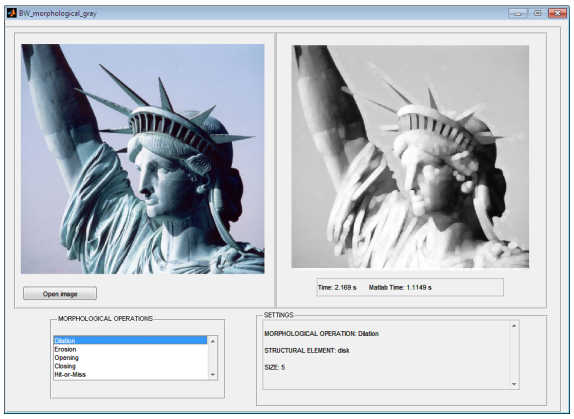


Figure 4.6: Dilation in gray images

Implementation in Matlab.

Code

```
function A2 = rkl_dilate_gray(A,B)

[rows,cols]=size(A);
[rowsB,colsB] = size(B);

% Calcule the center coordinates of B
cx = ceil(colsB/2);
cy = ceil(rowsB/2);
% Calculate new image
new_image = zeros(rows+rowsB-1,cols+colsB-1);
new_image(:) = -Inf;
if mod(colsB,2)~=0
    lastx = cols+colsB-1 -cx+1;
    deltax=cx-1;
else
    lastx = cols+colsB-1 -cx;
    deltax=cx;
end
```

```
if mod(rowsB,2)~=0
    lasty = rows+rowsB-1 -cy+1;
    deltay=cy-1;
else
    lasty = rows+rowsB-1 -cy;
    deltay= cy;
end

new_image(cy:lasty,cx:lastx) = A;
%% Calcule reflection of B
R_B = B(end:-1:1,end:-1:1);
%Calculate dilation
A2 = zeros(rows,cols);
for i=cy:lasty
    for j=cx:lastx
        a = new_image(i-cy+1:i+deltay,j-cx+1:j+deltax).*R_B;
        A2(i-cy+1,j-cx+1)=max(max(a));
    end
end
```

Definition 4.15. *The gray-scale erosion of an image by a structuring element, is denoted by*

$$(I \ominus B)(x,y) = min\left\{I\left(x+x^{'},y+y^{'}\right) - B\left(x^{'},y^{'}\right) | (x^{'},y^{'}) \in B\right\},$$

where I is the image and B is the structuring element. We assume that the value $I(x,y) = +\infty$ outside the image.

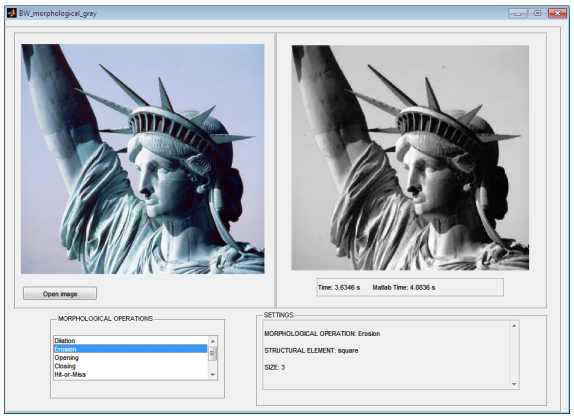


Figure 4.7: Erosion in gray images.

Implementation in Matlab.

Code

```
function A2 = rkl_erosion_gray(A,B)

[rows,cols]=size(A);
[rowsB,colsB] = size(B);
% Calcule the center coordinates of B
cx = ceil(colsB/2);
cy = ceil(rowsB/2);
% Calculate new image
new_image = zeros(rows+rowsB-1,cols+colsB-1);
new_image(:,:) = +Inf;
if mod(colsB,2)~=0
    lastx = cols+colsB-1 -cx+1;
else
    lastx = cols+colsB-1 -cx;
end
if mod(rowsB,2)~=0
    lasty = rows+rowsB-1 -cy+1;
else
```

```
    lasty = rows+rowsB-1 -cy;
end

new_image(cy:lasty,cx:lastx) = A;

A2 = zeros(rows,cols);
for i=cy:lasty
    for j=cx:lastx
        a = new_image(i-cy+1:i+cy-1,j-cx+1:j+cx-1).*B;
        a(a==0) = +Inf;
        A2(i-cy+1,j-cx+1) = min(min(a));
    end
end
```

4.3.2 Opening and Closing

Definition 4.16. *The opening of an image by a structuring element is denoted as*

$$I \circ B = (I \ominus B) \oplus B,$$

where I is the image, B the structuring element and the operations $\oplus, \ominus$ as defined above.

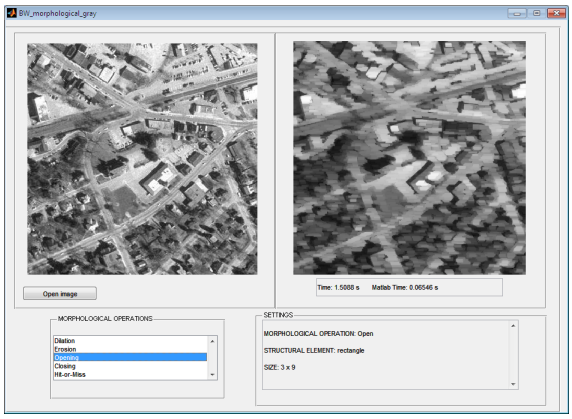


Figure 4.8: Opening in gray images.

Implementation in Matlab.

Code

```
function A2 = rkl_open_gray(A,B)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%% Opening in gray images
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
A1 = morph.rkl_erode_gray(A,B);
A2 = morph.rkl_dilate_gray(A1,B);
```

Definition 4.17. *The closing of an image by a structuring element is denoted as*

$$I \bullet B = (I \oplus B) \ominus B,$$

where I is the image, B the structuring element and the operations $\oplus, \ominus$ as defined above.

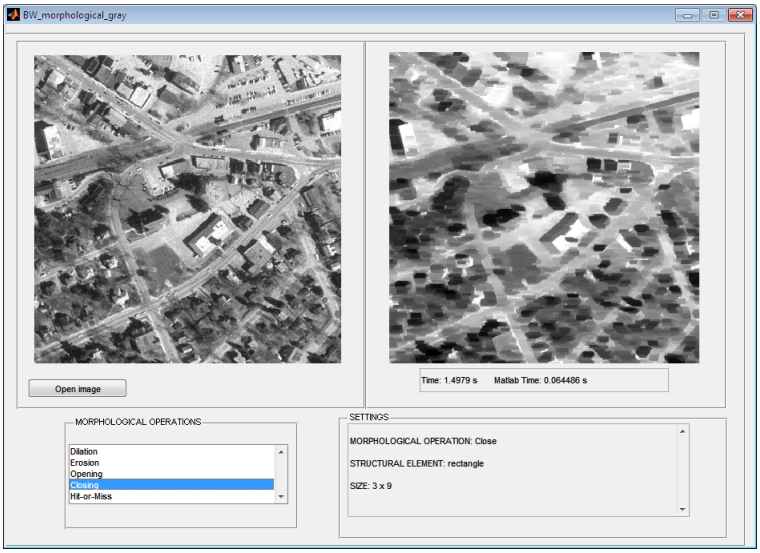


Figure 4.9: Closing in gray images.

Implementation in Matlab.

Code

```
function A2 = rkl_close_gray(A,B)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%% Closing in gray images
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

A1 = morph.rkl_dilate_gray(A,B);
A2 = morph.rkl_erode_gray(A1,B);
```

4.3.3 Applications: bottom-hat, top-hat and morphological gradient

Definition 4.18. *The* Bottom-hat *transformation is defined as*

$$I \bullet B - I,$$

where B is the structuring element, and I is the image.

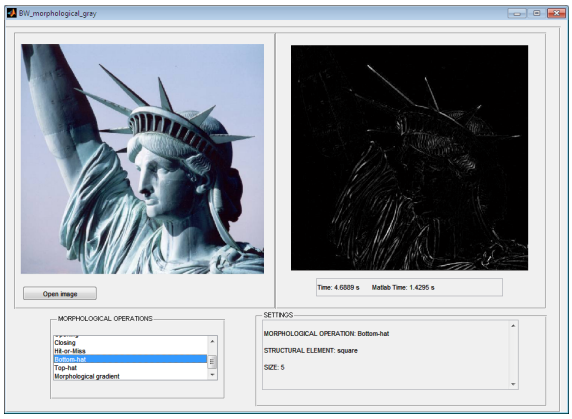


Figure 4.10: Bottom-hat of *estatua.jpg*

Implementation in Matlab.

Code

```
function A2 = rkl_bothat_gray(A,B)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%% Bottom hat: closing and subtracting the image
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

A1 = morph.rkl_close_gray(A,B);
A2 = imsubtract(A1,A);
```

Definition 4.19. *The* Top-hat *transformation is defined as*

$$I - I \circ B,$$

where B is the structuring element, and I is the image.

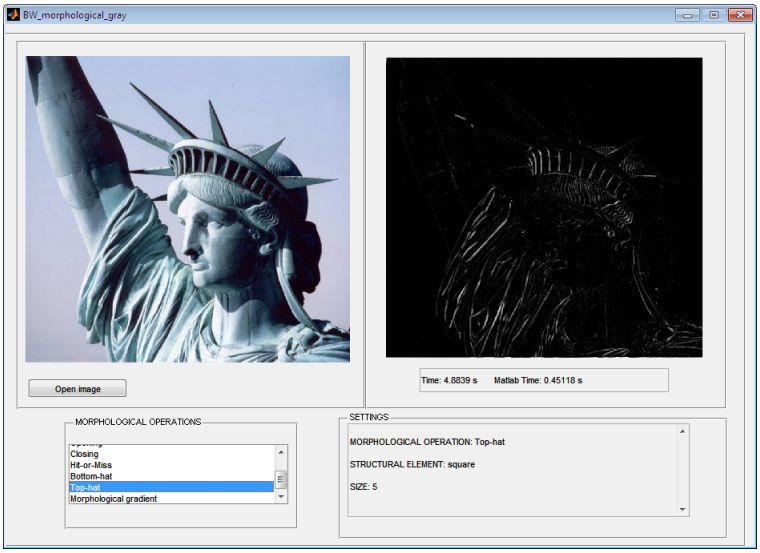


Figure 4.11: Top-hat of *estatua.jpg*

Implementation in Matlab.

Code

```
function A2 = rkl_tophat_gray(A,B)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%% Top hat: openning and subtracting the image
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

A1 = morph.rkl_open_gray(A,B);
A2 = imsubtract(A,A1);
```

Definition 4.20. *The* morphological gradient *transformation is defined as*

$$I \oplus B - I \ominus B,$$

where B is the structuring element, and I is the image.

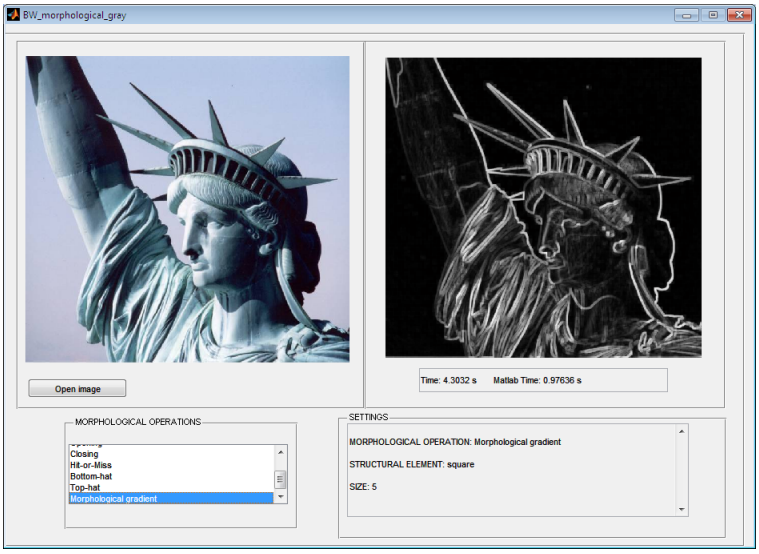


Figure 4.12: Morphological gradient of *estatua.jpg*.

Implementation in Matlab.

Code

```
function A2 = rkl_gradient_gray(A,B)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%% Gradient: subtracting dilation with erosion
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

C1 = morph.rkl_dilate_gray(A,B);
C2 = morph.rkl_erode_gray(A,B);
A2 = imsubtract(C1,C2);
```

4.3.4 GUI Morphological Operations

In this section we explain how to run the two *GUI*'s of morphological operations, one for binary images and the other for intensity images.

Morphological operations GUI for binary images.

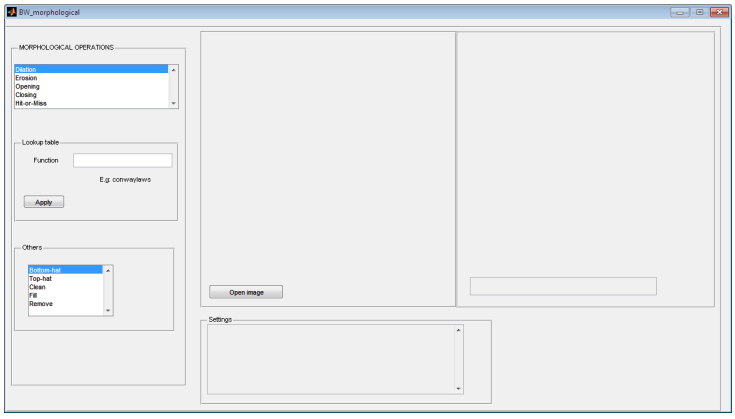


Figure 4.13: Binary images.

- Open the image to be processed.

- Choose the morphological operation. It will appear another window to choose the structuring element. Then press *Apply* button. See Figure 4.15.

- The output image will appear in the main window.

Morphological operations GUI for intensity images.

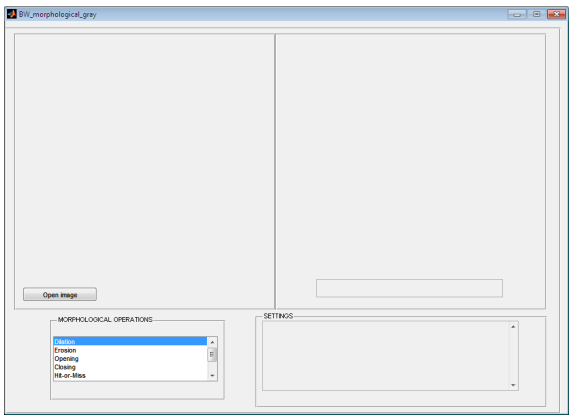


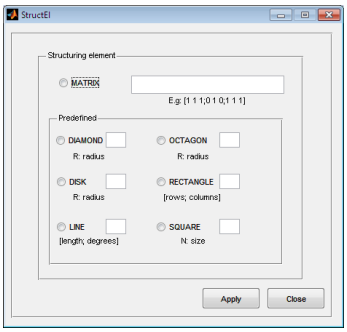
Figure 4.14: Intensity images.

- Open the image to be processed.

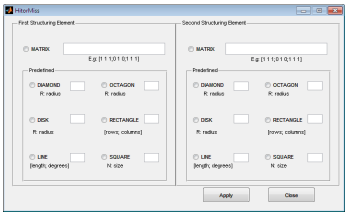
- Choose the morphological operation. It will appear another window to choose the structuring element. Then press *Apply* button. See Figure 4.15.

- The output image will appear in the main window.

There are two windows to choose the structuring element. The one that has two panels is only used for *Hit-or-Miss* transformation.



(a) One structuring element.



(b) Two structuring elements.

Figure 4.15: Structuring element windows.

Chapter 5

Wavelets on $\mathbb{Z}_N$

5.1 Introduction

In this section we introduce the motivation of *Wavelets* analysis in front of *Fourier* analysis. We will continue working with discrete signals in $\ell^2(\mathbb{Z}_N)$. See [4].

Definition 5.1. *We say that a vector $z \in \ell^2(\mathbb{Z}_N)$ is* localized in space *near $n_0 \in \mathbb{Z}_N$, if most of the components $z(n)$ of z are 0 or al least relatively small, except for a few values of n close to n_0.*

A *Fourier basis* element is not localized, because it has the same magnitude $\frac{1}{\sqrt{N}}$ for every $n \in \mathbb{Z}_N$. See 1.1.

Now, suppose $B = \{v_0, \ldots, v_{N-1}\}$ is a basis for $\ell^2(\mathbb{Z}_N)$ such that all the basis elements of B are *localized in space*. For a vector z, we can write $z = \sum_{n=0}^{N-1} a_n v_n$, $a_n \in \mathbb{R}$. Suppose that we wish to focus on the position of z near some particular point n_0. Terms involving basis vectors that are 0 or relatively small near n_0 can be deleted in without changing the behaviour. Then we can replace a sum over N terms by a smaller sum.

5.2 Wavelets on $\mathbb{Z}_N$

Notation 5.2. *We introduce new notation, and we recall the notation used above.*

1. $\tilde{w}(n) = \overline{w(-n)} = \overline{w(N-n)}$.
2. $\widehat{(\tilde{w})}(n) = \overline{\hat{w}(n)}$.
3. $(R_k z)(n) = z(n-k)$.
4. $(z * \tilde{w})(k) = \langle z, R_k w \rangle$.

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

Lemma 5.3. *Suppose $z, w \in \ell^2(\mathbb{Z}_N)$. For any $k \in \mathbb{Z}$*

$$z * \tilde{w}(k) = \langle z, R_k w \rangle ,$$

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

Remark 5.4. $\widetilde{(\tilde{w})} = w$: $\widetilde{(\tilde{w})} = \widetilde{(\overline{w(-n)})} = \overline{\overline{(w(n))}} = w(n)$.

Proof. By definition and commutativity,

1.
$$\begin{aligned}\langle z, R_k w \rangle &= \sum_{n=0}^{N-1} z(n)\overline{R_k w(n)} = \sum_{n=0}^{N-1} z(n)\overline{w(n-k)} \\ &= \sum_{n=0}^{N-1} z(n)\tilde{w}(k-n) = \tilde{w} * z(k) = z * \tilde{w}(k).\end{aligned}$$

2.
$$\begin{aligned}\langle z, R_k \tilde{w} \rangle &= \sum_{n=0}^{N-1} z(n)\overline{R_k \tilde{w}(n)} = \sum_{n=0}^{N-1} z(n)\overline{\tilde{w}(n-k)} \\ &= \sum_{n=0}^{N-1} z(n)\widetilde{\tilde{w}}(k-n) = w * z(k) = z * w(k).\end{aligned}$$

□

Lemma 5.5. *Let $w \in \ell^2(\mathbb{Z}_N)$. Then $\{R_k w\}_{k=0}^{N-1}$ is an orthonormal basis for $\ell^2(\mathbb{Z}_N)$ if and only if $|\hat{w}(n)| = 1$ for all $n \in \mathbb{Z}_N$.*

Proof. Let us introduce the Dirac Delta distribution,

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

By a simple calculation, we obtain $\hat{\delta}(n) = 1$ for all $n \in N$:

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

$\{R_k w\}_{k=0}^{N-1}$ is an orthonormal basis if and only if

$$\langle w, R_k w \rangle = \begin{cases} 0 & \forall k = 1, \ldots, N-1 \\ 1 & k = 0 \end{cases}$$

By notation,

$$\begin{aligned} (w * \tilde{w})(k) = \delta(k) &\iff (w * \tilde{w})\hat{}(n) = \hat{\delta}(n) = 1 \ \forall n \\ &\iff \hat{w}(n)(\tilde{w})\hat{}(n) = 1 \ \forall n \iff |\hat{w}(n)| = 1. \end{aligned}$$

□

Definition 5.6. *Suppose N is an even integer, $N = 2M$ for some $M \in \mathbb{N}$. An orthonormal basis for $\ell^2(\mathbb{Z}_N)$ of the form*

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

for some $u, v \in \ell^2(\mathbb{Z}_N)$ is called a first-stage wavelet basis.

Our goal is to determine when a pair u, v generates a *first-stage wavelet*. First we will introduce the necessary background for the theorem that will determine that pair.

Lemma 5.7. *Suppose $M \in \mathbb{N}$, $N = 2M$, $z \in \ell^2(\mathbb{Z}_N)$. Define $z^* \in \ell^2(\mathbb{Z}_N)$ by*

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

Then,

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

Proof. By definition,

$$\begin{aligned}
\widehat{(z^*)}(n) &= \sum_{k=0}^{N-1} z^*(k) e^{-2\pi i k n/N} = \sum_{k=0}^{N-1} (-1)^k z(k) e^{-2\pi i k n/N} \\
&= \sum_{k=0}^{N-1} z(k) e^{-i\pi k} e^{-2\pi i k n/N} \underbrace{=}_{\frac{-2\pi i k n}{N} - \pi i k = \frac{-2\pi i k (n+M)}{N}} \sum_{k=0}^{N-1} z(k) e^{-2\pi i k (n+M)/N} \\
&= \hat{z}(n+M).
\end{aligned}$$

□

Remark 5.8. *For any $z \in \ell^2(\mathbb{Z}_N)$, with N even*

$$(z + z^*)(n) = z(n)(1 + (-1)^n) = \begin{cases} 2z(n) & \text{if } n \text{ is even,} \\ 0 & \text{if } n \text{ is odd.} \end{cases}$$

Lemma 5.9. *Suppose $M \in \mathbb{N}$, $N = 2M$ and $w \in \ell^2(\mathbb{Z}_N)$. Then $\{R_{2k} w\}_{k=0}^{M-1}$ is an orthonormal set with M elements if and only if*

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

Proof. $\{R_{2k} w\}_{k=0}^{M-1}$ is an orthonormal basis if and only if

$$\langle w, R_{2k} w \rangle = \begin{cases} 1 & k = 0 \\ 0 & k = 1, \ldots, M-1\ . \end{cases}$$

$$(w * \tilde{w})(n) + (w * \tilde{w})^*(n) = \begin{cases} 2 & n = 0 \\ 0 & n = 1, \ldots, N-1\ . \end{cases}$$

$$
\begin{aligned}
&(w * \tilde{w}) + (w * \tilde{w})^* = 2\delta \iff \\
&(w * \tilde{w})\hat{}(n) + [(w * \tilde{w})^*]\hat{}(n) = 2\hat{\delta} = 2 \ \forall n \qquad \underbrace{\iff}_{\text{Lemma 5.7, Notation 5.2}} \\
&|\hat{w}(n)|^2 + |\hat{w}(n+M)|^2 = 2 \ \forall n.
\end{aligned}
$$

□

Definition 5.10. *Suppose $M \in \mathbb{N}$, $N = 2M$ and $u, v \in \ell^2(\mathbb{Z}_N)$. For $n \in \mathbb{Z}$, define $A(n)$ the system matrix of u and v by*

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

Now, we can introduce the necessary and sufficient conditions to determine *first-stage wavelet basis* pair.

Theorem 5.11. *Suppose $M \in \mathbb{N}$, $N = 2M$. Let $u, v \in \ell^2(\mathbb{Z}_N)$. Then,*

$$
\begin{aligned}
B &= \{R_{2k}u\}_{k=0}^{M-1} \bigcup \{R_{2k}u\}_{k=0}^{M-1} \\
&= \{u, R_2u, R_4u, \ldots, R_{N-2}u, v, R_2v, R_4v, \ldots, R_{N-2}v\}
\end{aligned}
$$

is an orthonormal basis for $\ell^2(\mathbb{Z}_N)$ if and only if $A(n)$ 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*

1. $|\hat{u}(n)|^2 + |\hat{u}(n+M)|^2 = 2 \ \forall n$,
2. $|\hat{v}(n)|^2 + |\hat{v}(n+M)|^2 = 2 \ \forall n$,
3. $\hat{u}(n)\overline{\hat{v}(n)} + \hat{u}(n+M)\overline{\hat{v}(n+M)} = 0 \ \forall n$.

Proof. We know that $\{R_{2k}u\}_{k=0}^{M-1}$ is an orthonormal set if and only if first condition is satisfied.

The same for $\{R_{2k}v\}_{k=0}^{M-1}$ with the second condition.

So, we only have to proof the last one:

$$\langle R_{2k}u, R_{2k}v \rangle = 0, \ \forall j \neq k \iff (3) \ is \ satisfied.$$

$$(u * \tilde{v})(2k) = \langle u, R_{2k} v \rangle = 0 \iff$$
$$(u * \tilde{v})(2k) = 0, \ \forall k = 0, \ldots, M-1 \iff$$
$$(u * \tilde{v})(n) + (u * \tilde{v})^*(n) = 0 \iff$$
$$(u * \tilde{v})(n) + [(u * \tilde{v})^*(n)] = 0 \iff$$
$$\hat{u}(n)\overline{\hat{v}(n)} + \hat{u}(n+M)\overline{\hat{v}(n+M)} = 0 \ \forall n.$$

Example 5.12. *1. Haar first-stage wavelet basis.* □

$$u = \left(\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, 0, \ldots, 0 \right),$$
$$v = \left(\frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}}, 0, \ldots, 0 \right).$$

In the Figure 5.1 we show two elements of the Haar basis of length 16.

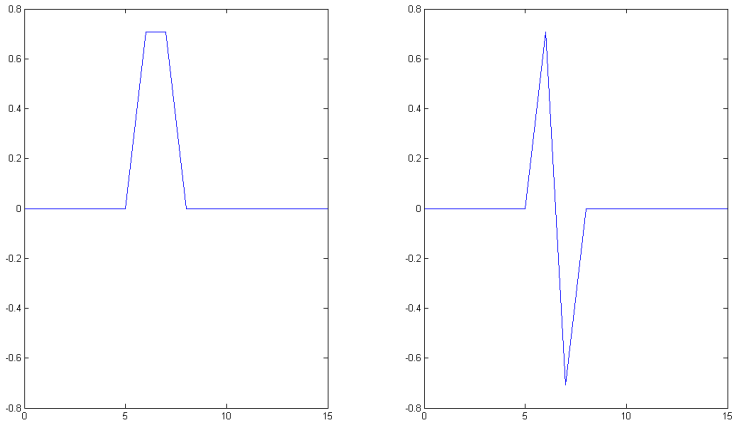


Figure 5.1: $R_8 u$ and $R_8 v$

2. Shannon first-stage wavelet basis.

Suppose N divisible by 4. *We define,*

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

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

We observe that $A(n)$ are unitary.

$$A\left(\frac{N}{4}\right) = \frac{1}{\sqrt{2}} \begin{pmatrix} \sqrt{2} & 0 & \cdots & 0 \\ 0 & \cdots & 0 & \sqrt{2} \end{pmatrix}$$

3. *Real Shannon first-stage wavelet basis.*

Suppose N divisible by 4. We define,

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

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

We observe that $A(n)$ are unitary.

$$A\left(\frac{N}{4}\right) = \frac{1}{\sqrt{2}} \begin{pmatrix} i & 1 \\ -i & 1 \end{pmatrix}$$

In the Figure 5.2 we show two elements of the Shannon real basis of length 64.

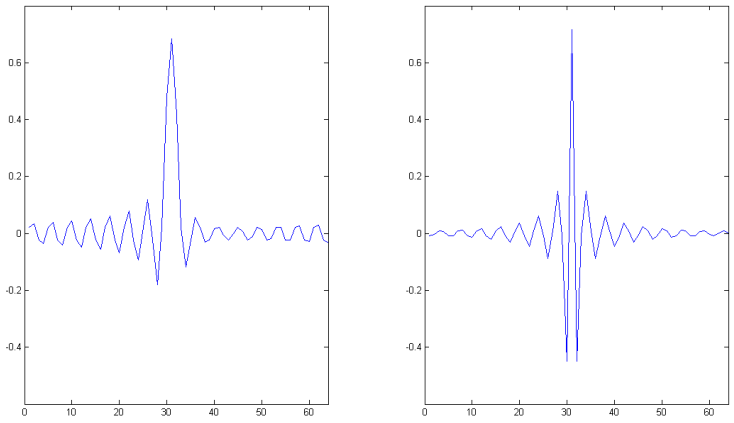


Figure 5.2: $R_{32}u$ and $R_{32}v$

4. Daub6 wavelet basis.

Ingrid Daubechies constructed families of wavelets that are very well localized in space rather than in frequency. We assume that N is divisible by 2^p, for some positive integer p, and that $N/2^p > 6$. Let $M = N/2$. Our goal is to construct $u \in \ell^2(\mathbb{Z}_N)$ such that u has only six nonzero components and satisfies the first equation in Theorem 5.11 and then we apply Lemma 5.13 to find v.

We begin with the trivial identity

$$\left[\cos^2\left(\frac{\pi n}{N}\right) + \sin^2\left(\frac{\pi n}{N}\right)\right]^5 = 1 \text{ for all } n.$$

Expanding this out, we have

$$\begin{aligned}&\cos^{10}\left(\frac{\pi n}{N}\right) + 5\cos^8\left(\frac{\pi n}{N}\right)\sin^2\left(\frac{\pi n}{N}\right) + 10\cos^6\left(\frac{\pi n}{N}\right)\sin^4\left(\frac{\pi n}{N}\right)\\ &+ 10\cos^4\left(\frac{\pi n}{N}\right)\sin^6\left(\frac{\pi n}{N}\right) + 5\cos^2\left(\frac{\pi n}{N}\right)\sin^8\left(\frac{\pi n}{N}\right) + \sin^{10}\left(\frac{\pi n}{N}\right) = 1.\end{aligned} \tag{5.1}$$

Define

$$b(n) = \cos^{10}\left(\frac{\pi n}{N}\right) + 5\cos^8\left(\frac{\pi n}{N}\right)\sin^2\left(\frac{\pi n}{N}\right) + 10\cos^6\left(\frac{\pi n}{N}\right)\sin^4\left(\frac{\pi n}{N}\right).$$

Note that

$$\cos\left(\frac{\pi(n+M)}{N}\right) = \cos\left(\frac{\pi n}{N} + \frac{\pi}{2}\right) = -\sin\left(\frac{\pi n}{N}\right)$$

and similarly

$$\sin\left(\frac{\pi(n+M)}{N}\right) = \cos\left(\frac{\pi n}{N}\right).$$

Hence

$$b(n+M) = 10\cos^4\left(\frac{\pi n}{N}\right)\sin^6\left(\frac{\pi n}{N}\right) + 5\cos^2\left(\frac{\pi n}{N}\right)\sin^8\left(\frac{\pi n}{N}\right) + \sin^{10}\left(\frac{\pi n}{N}\right).$$

Thus by equation 5.1

$$b(n) + b(n+M) = 1 \text{ for all } n.$$

We select $u \in \ell^2(\mathbb{Z}_N)$ so that

$$|\hat{u}(n)|^2 = 2b(n). \tag{5.2}$$

Then we have

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

We could obtain equation 5.2 by setting $u(n) = (\sqrt{2b(n)})\check{}$. *But here we do not have a vector* u *with only six nonzero values. We write*

$$\begin{aligned} b(n) &= \cos^6\left(\frac{\pi n}{N}\right)\left[\cos^4\left(\frac{\pi n}{N}\right) + 5\cos^2\left(\frac{\pi n}{N}\right)\sin^2\left(\frac{\pi n}{N}\right) + 10\sin^4\left(\frac{\pi n}{N}\right)\right] \\ &= \cos^6\left(\frac{\pi n}{N}\right)\left[\left(\cos^2\left(\frac{\pi n}{N}\right) - \sqrt{10}\sin^2\left(\frac{\pi n}{N}\right)\right)^2\right. \\ &\qquad \left. + (5+2\sqrt{10})\cos^2\left(\frac{\pi n}{N}\right)\sin^2\left(\frac{\pi n}{N}\right)\right]. \end{aligned}$$

Define $\hat{u} \in \ell^2(\mathbb{Z}_N)$ *by*

$$\begin{aligned} \hat{u}(n) = \sqrt{2}e^{-2\pi i n/N}\cos^3\left(\frac{\pi n}{N}\right)\left[\cos^2\left(\frac{\pi n}{N}\right) - \sqrt{10}\sin^2\left(\frac{\pi n}{N}\right)\right. \\ \left. + i\sqrt{5+2\sqrt{10}}\cos\left(\frac{\pi n}{N}\right)\sin\left(\frac{\pi n}{N}\right)\right]. \end{aligned}$$

By applying Euler's formula and the double angle identities, we have

$$\begin{aligned} \hat{u}(n) = \sqrt{2}e^{-2\pi i 4/N}e^{3\pi i n/N}\left(\frac{e^{i\pi n/N} + e^{-i\pi n/N}}{2}\right)^3 \\ \left[\frac{1}{2}\left(1+\cos\left(\frac{2\pi n}{N}\right)\right) - \frac{\sqrt{10}}{2}\left(1-\cos\left(\frac{2\pi n}{N}\right)\right)\right. \\ \left. + i\frac{\sqrt{5+2\sqrt{10}}}{2}\sin\left(\frac{2\pi n}{N}\right)\right]. \end{aligned}$$

To simplify, $a = 1-\sqrt{10}$, $b = 1+\sqrt{10}$ *and* $c = \sqrt{5+2\sqrt{10}}$.

$$\begin{aligned} \hat{u}(n) = \frac{\sqrt{2}}{8}e^{-2\pi i 4n/N}\left(e^{2\pi i n/N}+1\right)^3 \\ \left[\frac{a}{2} + \frac{b}{4}\left(e^{2\pi i n/N} + e^{-2\pi i n/N}\right) + \frac{c}{4}\left(e^{2\pi i n/N} - e^{-2\pi i n/N}\right)\right]. \end{aligned}$$

At this point we can see that

$$\hat{u}(n) = \sum_{k=0}^{5} u(k)e^{-2\pi i k n/N},$$

for some numbers $u(0), u(1), u(2), u(3), u(4), u(5)$. *If we calculate them, we obtain*

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

Finally, with Lemma 5.13 we obtain the vector v

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

In the Figure 5.3 we show two elements of the Daubechies6 basis of length 64.

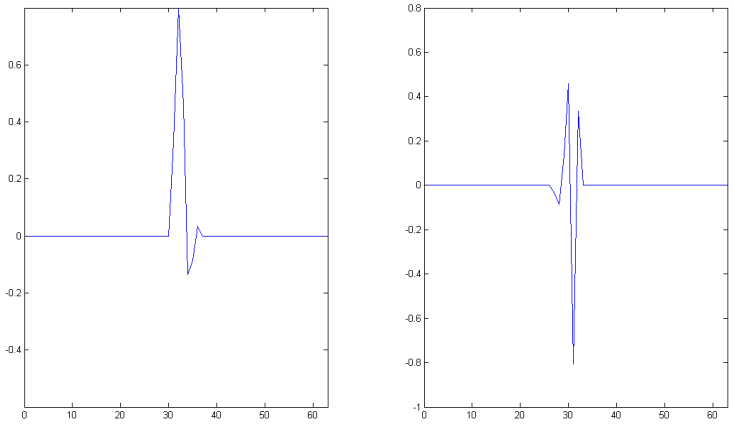


Figure 5.3: $R_{32}u$ and $R_{32}v$

We can compare Figure 5.2 with Figure 5.3 with same length.

Lemma 5.13. *Suppose* $M \in \mathbb{N}$, $N = 2M$ *and* $u \in \ell^2(\mathbb{Z}_N)$ *is such that* $\{R_{2k}u\}_{k=0}^{M-1}$ *is an orthonormal set with* M *elements. Define* $v \in \ell^2(\mathbb{Z}_N)$ *by*

$$v(k) = (-1)^{k-1}\overline{u(1-k)} \ \forall k.$$

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

Proof. We have to prove that the second and third conditions of Theorem 5.11 are satisfied.

First we will define $\hat{v}(m)$ and $\hat{v}(m+M)$:

1.

$$\begin{aligned}\hat{v}(m) &= \sum_{n=0}^{N-1} v(n)e^{-2\pi imn/N} = \sum_{n=0}^{N-1}(-1)^{n-1}\overline{u(1-n)}e^{-2\pi imn/N} \\ &\overset{k=1-n}{=} \sum_{n=0}^{N-1}(-1)^{-k}\overline{u(k)}e^{-2\pi im(1-k)/N} \\ &= e^{-2\pi im/N}\sum_{k=0}^{N-1}\overline{u(k)}(e^{-i\pi})^{-k}e^{2\pi imk/N} = e^{-2\pi im/N}\sum_{k=0}^{N-1}\overline{u(k)}e^{i\pi k+2\pi imk/N} \\ &= e^{-2\pi im/N}\overline{\sum_{k=0}^{N-1}u(k)e^{i\pi k+2\pi imk/N}} = e^{-2\pi im/N}\overline{\hat{u}(m+M)}.\end{aligned}$$

2.

$$\begin{aligned}\hat{v}(m+M) &= e^{-2\pi i(m+M)/N}\overline{\hat{u}(m+2M)} = e^{-2\pi im/N}e^{-2\pi iM/N}\hat{u}(m+N) \\ &= -e^{-2\pi im/N}\hat{u}(m).\end{aligned}$$

Now we can prove the conditions

1. $|\hat{v}(m)|^2 + |\hat{v}(m+M)|^2 = |\hat{v}(m+M)|^2 + |\hat{v}(m)|^2 = 2.$

2.

$$\begin{aligned}&\hat{u}(m)\overline{\hat{v}(m)} + \hat{u}(m+M)\overline{\hat{v}(m+M)} \\ &= \hat{u}(m)e^{2\pi im/N}\hat{u}(m+M) + \hat{u}(m+M)(-1)e^{2\pi im/N}\hat{u}(m) = 0.\end{aligned}$$

□

Suppose that $B = \{R_{2k}u\}_{k=0}^{M-1}\bigcup\{R_{2k}v\}_{k=0}^{M-1}$ is a first-stage wavelet basis. The change of basis of B to the euclidean basis E, is the matrix U, with columns the

vectors $v, R_2v, \ldots, R_{N-2}v, u, R_2u, \ldots, R_{N-2}u$. B is orthonormal, then U is a unitary matrix, so the change of basis from E to B is the matrix $U^{-1} = U^*$. Thus,

$$[z]_B = \begin{bmatrix} z * \tilde{v}(0) \\ \vdots \\ z * \tilde{v}(N-2) \\ z * \tilde{u}(0) \\ \vdots \\ z * \tilde{u}(N-2) \end{bmatrix}$$

where $[z]_B$ is the vector $z \in \ell^2(\mathbb{Z}_N)$ in B basis.

Definition 5.14. *Suppose $M \in \mathbb{N}$, $N = 2M$ and $u \in \ell^2(\mathbb{Z}_N)$. We define the Downsampling operator $D : \ell^2(\mathbb{Z}_N) \longrightarrow \ell^2(\mathbb{Z}_M)$*

$$D(z)(n) = z(2n), \ \ for\ n = 0, \ldots, M-1$$

In other words, if $z = (z(0), z(1), \ldots, z(N-1))$ the downsampling operator is often denoted $\downarrow 2$ in diagrams. The calculation of $[z]_B$ is represented in Figure 5.4.

We have seen that we can compute the E to B change of basis quickly. Now we are going to compute the B to E change of basis. This can be obtained by multiplying by the matrix U, but it takes $N \times N$ multiplications, and this is slow. In Figure 5.4 we see a fast procedure basek on the *filter bank* approach.

Definition 5.15. *A filter bank is an array of band-pass filters that separates the input signal into multiple components, each one carrying a simple frequency subband of the original signal.*

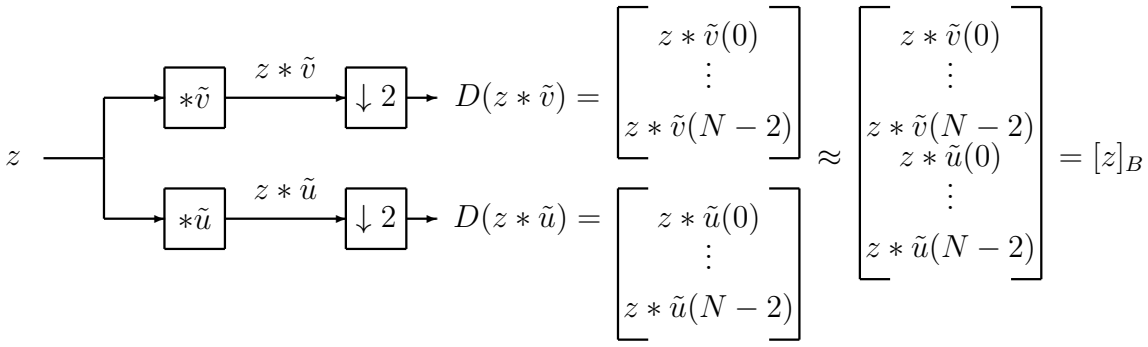


Figure 5.4

Definition 5.16. *Suppose $M \in \mathbb{N}$ and $N = 2M$. Define $U : \ell^2(\mathbb{Z}_M) \longrightarrow \ell^2(\mathbb{Z}_N)$ for $z \in \ell^2(\mathbb{Z}_M)$,*

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

The operator U is called the upsampling operator. It is denoted by $\uparrow 2$.

The *upsampling operator* doubles the size of a vector by inserting a 0 between any two adjacent values.

Remark 5.17.

1.

$$D \circ U(z) = z, \tag{5.3}$$

2.

$$U \circ D(z) = \frac{1}{2}(z + z^*). \tag{5.4}$$

To regain z from the output of the left filter bank in Figure 5.4 ,we follow up with a right filter bank as in the right half of Figure 5.5 Here $s, t \in \ell^2(\mathbb{Z}_N)$ are unknown. The output of the upper branch of Figure 5.5 is $\tilde{t} * U(D(z * \tilde{v}))$ and the output of the lower branch is $\tilde{s} * U(D(z * \tilde{u}))$.

The lemma below gives conditions under which the sum of these outputs is always the original input z. When this happens we say we have *perfect reconstruction* in the filter bank. Note that we do not necessarily assume the conditions of Theorem 5.11.

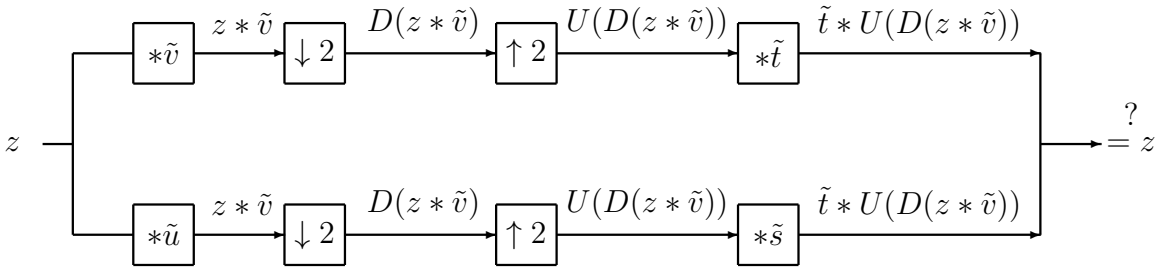


Figure 5.5

Lemma 5.18. *Suppose $M \in \mathbb{N}$, $N = 2M$ and $u, v, s, t \in \ell^2(\mathbb{Z}_N)$. For $n = 0, \ldots, N-1$, let $A(n)$ be the system matrix (def.) for u and v. Then we have perfect reconstruction in Figure 5.5, that is,*

$$\tilde{t} * U(D(z * \tilde{v})) + \tilde{s} * U(D(z * \tilde{u})) = z$$

for all $z \in \ell^2(\mathbb{Z}_N)$, if and only if

$$A(n) \cdot \begin{bmatrix} s(n) \\ t(n) \end{bmatrix} = \begin{bmatrix} \sqrt{2} \\ 0 \end{bmatrix}$$

for all $n = 0, \ldots, N-1$. In the case that $A(n)$ is unitary, this simplifies to $\hat{t}(n) = \overline{\hat{v}(n)}$ and $\hat{s}(n) = \overline{\hat{u}(n)}$. If $A(n)$ is unitary for all n, then $t = \tilde{v}$ and $s = \tilde{u}$.

Proof. By properties of the upsampling and downsampling operator we have that

$$U(D(z * \tilde{v})) = \frac{1}{2}\left((z * \tilde{v}) + (z * \tilde{v})^*\right)$$

and similarly with v replaced by u.

Remark 5.19. $z^*(n) = (-1)^n z(n)\ \forall n \ \Rightarrow (z^*)\hat{}(n) = \hat{z}(n+M)\ \forall n$.

Taking Fourier transform,

$$\begin{aligned} U(D(z*\tilde{v}))\hat{}(n) &= \frac{1}{2}(\hat{z}(n)\overline{\hat{v}(n)} + (z*\tilde{v})\hat{}(n+M)) \\ &= \frac{1}{2}(\hat{z}(n)\overline{\hat{v}(n)} + (\hat{z}(n+M)\overline{\hat{v}(n+M)})\ \forall n \end{aligned}$$

and similarly with v replaced by u.

Let us see that

$$\tilde{t} * U(D(z*\tilde{v})) + \tilde{s} * U(D(z*\tilde{u})) = z,\ \forall z \in \ell^2(\mathbb{Z}_N).$$

We will prove that

$$\left(\tilde{t} * U(D(z*\tilde{v})) + \tilde{s} * U(D(z*\tilde{u}))\right)\hat{}(n) = \hat{z}(n),\ \forall z \in \ell^2(\mathbb{Z}_N).$$

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

This happens if and only if,

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

and

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

and it is equivalent to

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

Notice that

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

In the case that $A(n)$ is unitary, $A(n)$ is invertible and $(A(n))^{-1} = A^*(n)$, so solving

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

From this equality we obtain $\hat{s}(n) = \overline{\hat{u}(n)}$ and $\hat{t}(n) = \overline{\hat{v}(n)}$. Then, if $A(n)$ is unitary for all n, $s(n) = \tilde{u}$ and $t(n) = \tilde{v}$. Notice that $\tilde{s} = u$, $\tilde{t} = v$. □

Remark 5.20. *If u, v are wavelets, then*

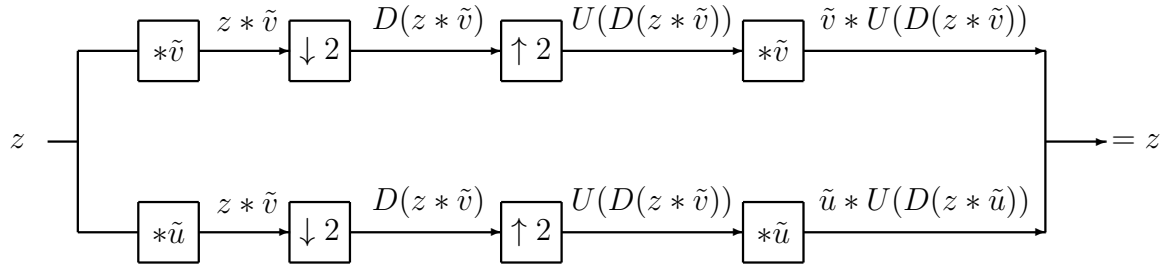


Figure 5.6

If $N = \dot{4}$, we will do 2 steps, if $N = \dot{8}$, we will do 3 steps and so on.

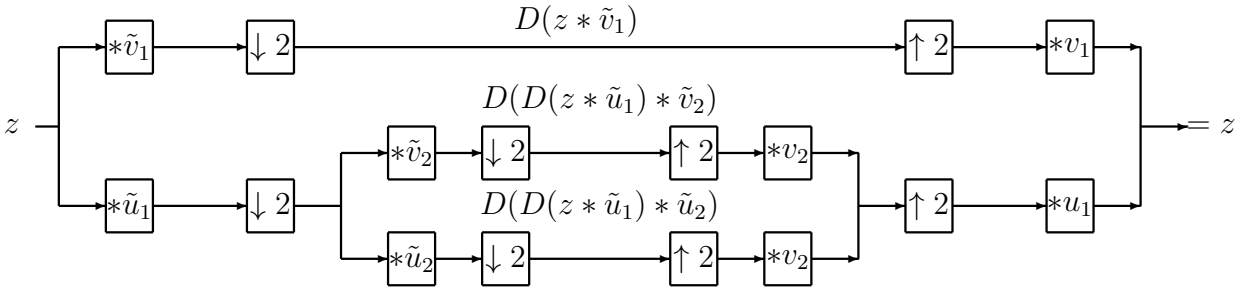


Figure 5.7

Definition 5.21. *If $N = 2^p$, we define a sequence of* wavelet filters *of stage p with a sequence of vectors $(u_1, v_1), \ldots, (u_p, v_p)$ such that*

1. $(u_l, v_l) \in \ell^2(\mathbb{Z}_{N/2^{l-1}})$, $l = 1, \ldots, p$.

2.

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

for $n = 0, \ldots, \frac{N}{2^l} - 1$ *the matrix are unitary.*

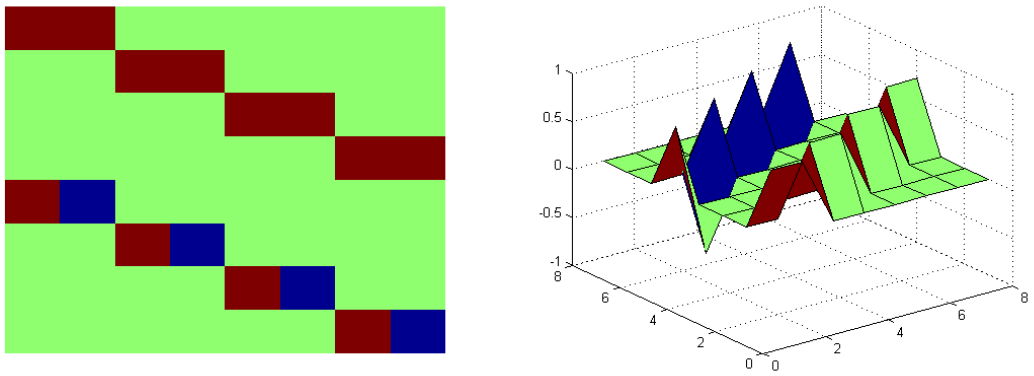
5.3 Matlab Implementation

5.3.1 Haar Wavelets

Code

```
function H=Haar1D(x)
%Supose rows=1 (x is a vector)
[rows,cols] = size(x);

% Check if size = pow(2), otherwise call padding function
[new_size,N] = padding(cols);
x_new = zeros(rows,new_size);
x_new(1:rows,1:cols)=x;

% Calcul Haar transform
v = x_new';
A = CalculTransf(pow2(N-1));
v_new = A*v;
H=[v_new(1:pow2(N-1))' v_new(pow2(N-1)+1:end)'];

%%%%%% Auxiliar function %%%%%%%%%%
%%% Matrix transformation
function A=CalculTransf(dim)

for i=1:dim
    S(i,i*2-1:i*2)=1/sqrt(2);
```

```
end
for i=1:dim
    D(i,i*2-1)=1/sqrt(2);
    D(i,i*2)=-1/sqrt(2);
end
A = [S;D];

%%% Padding
function [m,j]=padding(n)
m=2;
j=1;
while n>m
    m=m*2;
    j=j+1;
end
```



(a) 2D (b) 3D

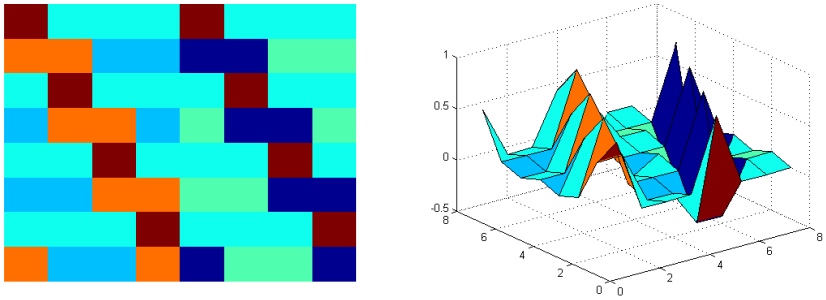
Figure 5.8: U matrix of Haar basis

5.3.2 Shannon Wavelets

Code

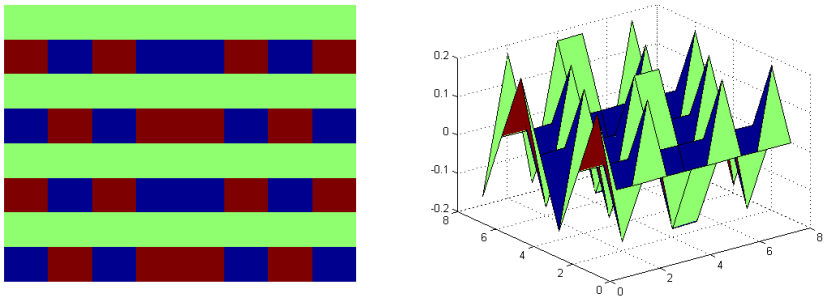
```
function z2=Shannon(z)

[rows,cols] = size(z);
%Check if m is multiple of 4
[m]=multiple4(cols);
v = zeros(1,m-cols);
z = [z v];

U = basisShannon(m);

for i=1:m
    z2(i) = sum(U(:,i)'.*z);
end
%% Auxiliar function
function U=basisShannon(m)
%% Calcule of Shannon basis in dimension m
fu = zeros(1,m);
fv = zeros(1,m);

fu(1:m/4) = sqrt(2);
fu(3*m/4 + 2:end) = sqrt(2);
fu(m/4 + 1)=j;
fu(3*m/4 + 1)=-j;

fv(m/4+2:3*m/4) = sqrt(2);
fv(3*m/4 + 1)=1;
fv(m/4 + 1)=1;

u=ifft(fu);
```

```
v=ifft(fv);
%% Calcule matrix U
for i=1:m/2
    U(:,i) = circshift(u,[1 (i-1)*2])';
end
for i=1:m/2
    U(:,i+m/2) = circshift(v,[1 (i-1)*2])';
end
```



(a) 2D (b) 3D

Figure 5.9: U matrix of Shannon basis, real part



(a) 2D (b) 3D

Figure 5.10: U matrix of Shannon basis, imaginary part

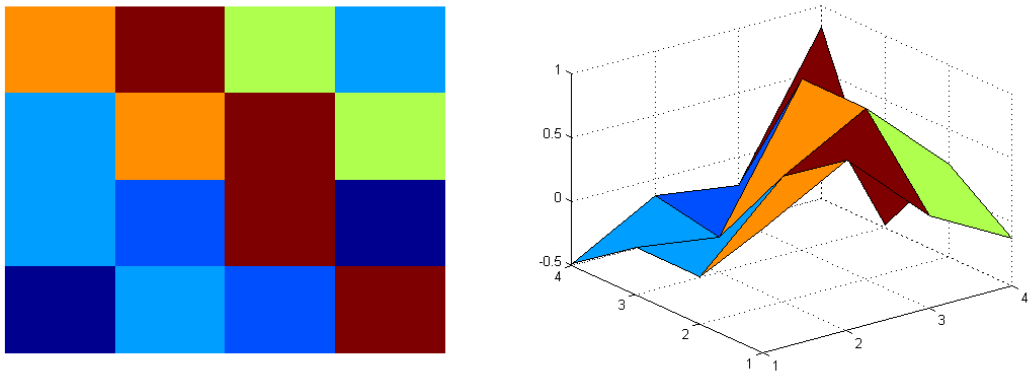
5.3.3 Daubechies Wavelets

We will compute the *Daubechies 4* and *Daubechies 6*.

Code

```
function Daub = Daub4(x)

%% Define alpha & beta
a1 = (1+sqrt(3))/(4*sqrt(2));a2 = (3+sqrt(3))/(4*sqrt(2));
a3 = (3-sqrt(3))/(4*sqrt(2));a4 = (1-sqrt(3))/(4*sqrt(2));
b1 = a4;b2 = -a3;b3 = a2;b4 = -a1;

[rows,cols]=size(x);
% Check if size = pow(2), otherwise call padding function
[new_size,N] = padding(cols);
x_new = zeros(rows,new_size);
x_new(1:rows,1:cols)=x;

a = zeros(1,new_size);
a(1:4)=[a1 a2 a3 a4];
b = zeros(1,new_size);
b(1:4)=[b1 b2 b3 b4];

% Calcule of matrix
m=0;
for i=1:new_size/2
    S(i,:)=circshift(a,[1 m]);
    m=m+1;
end
m=0;
for i=1:new_size/2
    D(i,:)=circshift(b,[1 m]);
```

```
    m=m+1;
end

A=[S;D];
Daub=A*x_new';
%%% Padding
function [m,j]=padding(n)
m=2;
j=1;
while n>m
    m=m*2;
    j=j+1;
end
```



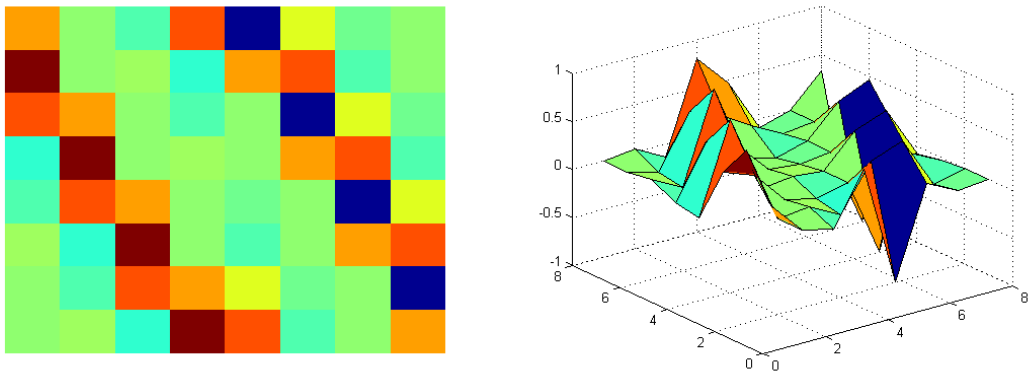
(a) 2D (b) 3D

Figure 5.11: U matrix Daubechies4 basis

Daubechies6 function.

Code

```
function U = Daub6(rows,cols)

a=1-sqrt(10);
b=1+sqrt(10);
c=sqrt(5+2*sqrt(10));

u = zeros(1,cols);
v = zeros(1,cols);

u(1) = b+c;
u(2) = 2*a + 3*b + 3*c;
u(3) = 6*a + 4*b + 2*c;
u(4) = 6*a + 4*b - 2*c;
u(5) = 2*a + 3*b - 3*c;
u(6) = b-c;
u = u.*(sqrt(2)/32);
v(1) = -u(2);
v(2) = u(1);
v(end)=u(3);
v(end-1)=-u(4);
v(end-2)=u(5);
v(end-3)=-u(6);

%% Calcule matrix U
for i=1:cols/2
    U(:,i) = circshift(u,[1 2*(i-1)]).';
    U(:,i+cols/2) = circshift(v,[1 2*(i-1)]).';
end
```



(a) 2D (b) 3D

Figure 5.12: U matrix of Daubechies6 basis

Chapter 6

Applications: Compression and De-noising

In this chapter we will deal with wavelets basis for compression and de-noising signal. The reason why wavelets basis are used for these applications is because they are well localized in frequency, and moreover because Daubechies wavelets are also well localized in space (as we will see in the examples of image compression). For a better comprehension of one dimensional signal processing, to analyse the spectrum of music, see [5].

For each $N \in \mathbb{N}$ one can use *Daubechies's DN wavelets* for $\ell^2(\mathbb{Z})$, for which u has N nonzero components, to construct supported wavelets for $L^2(\mathbb{R})$. As N increases, *Daubechies's DN wavelets* have larger support intervals. However, this is compensated in many applications by the facts that DN tend to have more cancellation and more smoothness.

The smoothness of *Daubechies's wavelets* grows linearly with their support, and linear growth is the best. For both *Daubechies wavelets transforms* and the corresponding scaling function require just a few values from the signal to get a good approximation, and that is *Daubechies* provide us with a powerful tool for signal processing.

6.1 Signal Compression

Wavelet transform produces a large number of values having zero, or near zero, magnitudes. The compression signal is obtained by applying a wavelet transform, and getting only the largest values obtained. After that, we apply the inverse transform and we will get the compression signal. Here, we will not deal with the encoding of those pixels that are set to zero.

To compare the results with the different wavelets basis we introduce the definition of the error that will measure the output signals.

Definition 6.1. *Mean Square Error(MSE) is defined as*

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

where N is the total number of elements, and the sum over i, j is the sum over all the elements in the signal, f is the original signal and g is its compression.

In the examples the relative error is displayed, and it is calculated by dividing the *MSE* with the norm of the signal

$$Error = N\frac{MSE}{norm(f)},$$

where f is the given signal.

Example 6.2. *We introduce two examples of* 1 *dimensional signal compression.*

1.

$$z(n) = \begin{cases} 0 & 0 \le n \le 127 \\ \sin\left(\frac{|n-128|^{1.7}}{128}\right) & 128 \le n \le 255 \\ 0 & 256 \le n \le 383 \\ \sin\left(\frac{|n-128|^{2}}{128}\right) & 384 \le n \le 447 \\ 0 & 448 \le n \le 511 \end{cases}$$

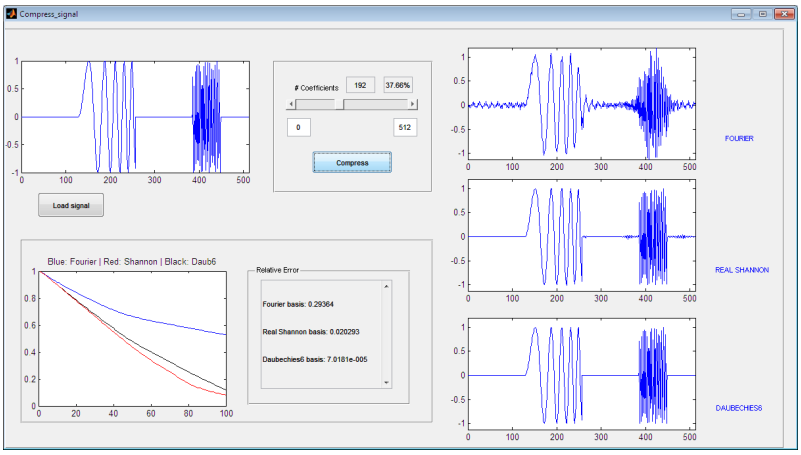


Figure 6.1

2.

$$z(n) = \begin{cases} 1 - \frac{n}{64} & 0 \leq n \leq 63 \\ 0 & 64 \leq n \leq 255 \\ 5 - \frac{n}{64} & 256 \leq n \leq 319 \\ 0 & 320 \leq n \leq 447 \end{cases}$$

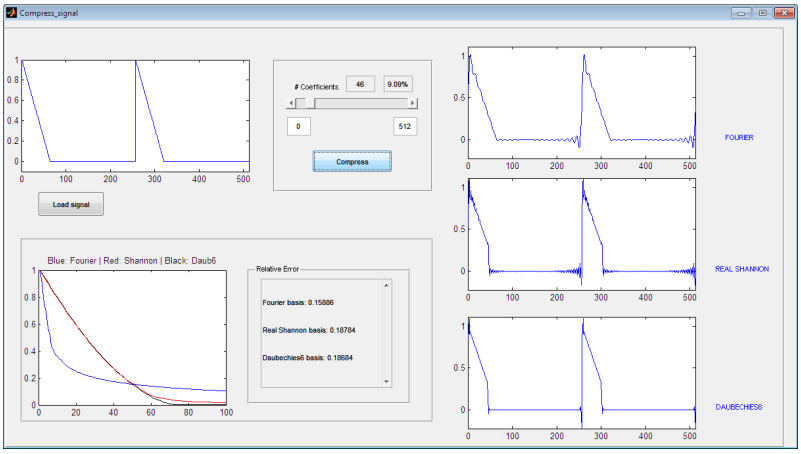


Figure 6.2

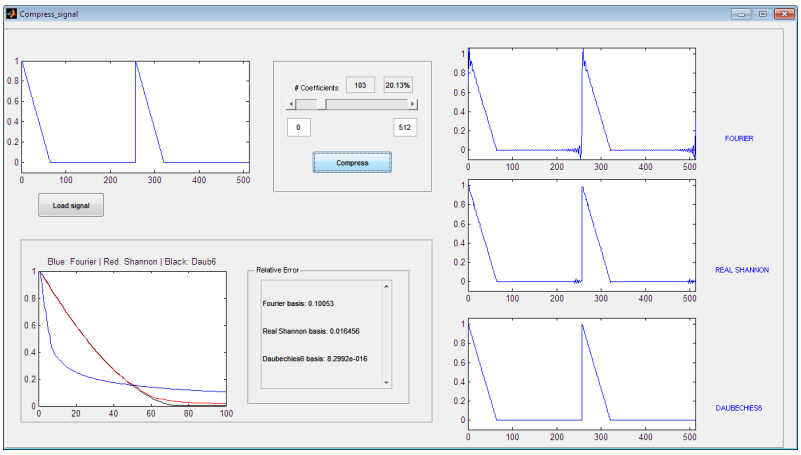


Figure 6.3

In this example we can see that in Figure 6.2 it seems that the Fourier transform compress better the signal, but as we can see in Figure 6.3, where we have taken more coefficients in the transformation, the Fourier compression is still having problems in those parts where have large changes, and not like wavelets transforms. With only the 20% *of the coefficients the Daubechies6 has an error of* 10^{-16}.

Example 6.3. *Here, we have the image of barbara.png. It has been compressed by taking the* 10% *of the largest coefficients of each wavelet and Fourier transform. If we compare the results by using Fourier and wavelets, we see that the outputs in Haar and Daubechies wavelets are better than the first one, as explained in Chapter 5.*

(a) Original image.

(b) Fourier 10% coeff.

(c) Haar 10% coeff.

(d) Daubechies6 10% coeff.

Figure 6.4: Compression *barbara.png* image.

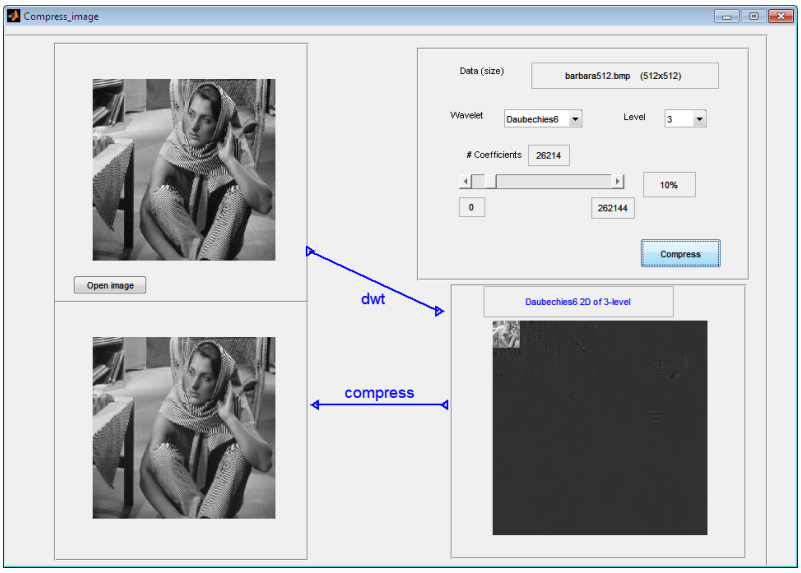


Figure 6.5

6.1.1 Matlab implementation

This function calculates *Fourier*, *Daubechies6* and *Shannon* wavelet transformation.

Code

```
 function compressWavelets(x,K)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%%  Compress signal x taking the higher
%%  values in diferent transform
%%  wavelets
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[rows,cols]=size(x);

v_max = [max(x)];
v_min = [min(x)];
```

```
%   Fast Fourier
F = fft(x);
F_err = F;
[order,v_f]=sort(abs(F),'descend');
F(v_f(K+1:end))=0;
Fourier = ifft(F);
v_max = [v_max max(real(Fourier))];
v_min = [v_min min(real(Fourier))];
error_Fourier = norm(real(Fourier) - x)/norm(x)

%   D6
U = TransfD6(rows,cols);
D = D6(x,U);
D_err = D;

[order_d,v_d] = sort(abs(D),'descend');
D(v_d(K+1:end))=0;
Daub6=ID6(D,U);
v_max = [v_max max(real(Daub6))];
v_min = [v_min min(real(Daub6))];
error_Daub6 = norm(-(Daub6) + x)/norm(x)

%   Shannon transform
Us = TransfSh(rows,cols);
S = Shannon2(x,Us);
S_err = S;

[order_s,v_s]=sort(abs(S),'descend');

S(v_s(K+1:end))=0;
Shannon=IShannon(S,Us);
v_max = [v_max max(real(Shannon))];
```

```
v_min = [v_min min(real(Shannon))];
error_Shannon = norm(real(Shannon) - x')/norm(x)

%%  Relative error for k=0:300
error_f=[];
error_d=[];
error_s=[];
norm_x = norm(x);
for k=100:-1:1
    F_err(v_f(k:end))=0;
    Fourier_err = ifft(F_err);
    error_f = [error_f norm(real(Fourier_err) - x)/norm_x];

      D_err(v_d(k:end))=0;
      Daub6_err = ID6(D_err,U);
      error_d = [error_d norm(real(Daub6_err) - x)/norm_x];

     S_err(v_s(k:end))=0;
     Shannon_err = IShannon(S_err,Us);
     error_s = [error_s norm(real(Shannon_err) - x')/norm_x];
end


%% Plot the results
figure;
subplot(3,2,1);plot(x);axis([0 length(x) min(v_min) max(v_max)]);
title('Signal');
subplot(3,2,2);plot(real(Fourier));
axis([0 length(x) min(v_min) max(v_max)]);
title(['Fourier with ',int2str(K),' higher coeff.']);
subplot(3,2,3);plot(real(Daub6));
axis([0 length(x) min(v_min) max(v_max)]);
```

```
title(['Daub6 with ',int2str(K),' higher coeff.']);
subplot(3,2,4);plot(real(Shannon));
axis([0 length(x) min(v_min) max(v_max)]);
title(['Shannon with ',int2str(K),' higher coeff.']);
subplot(3,2,5);plot(error_f(end:-1:1),'b');hold on;
plot(error_d(end:-1:1),'g');plot(error_s(end:-1:1),'r');hold off;
title('Red: Shannon. Green: Daub6. Blue: Fourier.');
```

Compression image by *Daubechies6*.

Code

```
function [fcompres,coef]=compress_image_d6(signal,level,eps)

[r,c]=size(signal);

%Order
F_abs=abs(signal);
[order,x]=ordenar_matriz(F_abs);
%Be carefull, now order is a vector!!
Fcompres=signal;
Fcompres=reshape(Fcompres,1,[]);
Fcompres(x(eps+1:end))= 0;
Fcompres = reshape(Fcompres,[r c]);
fcompres = InvD6_rkl(Fcompres,level);
```

Compression image by *Haar*.

Code

```
function [fcompres,coef]=compress_image_haar(signal,level,eps)

[r,c]=size(signal);
```

```
%Order
F_abs=abs(signal);
[order,x]=ordenar_matriz(F_abs);
%Be carefull, now order is a vector!!
Fcompres=signal;
Fcompres=reshape(Fcompres,1,[]);
Fcompres(x(eps+1:end))= 0;
Fcompres = reshape(Fcompres,[r c]);
fcompres = Inv_rkl(Fcompres,level);
```

Auxiliar function *order matrix*.

Code

```
function [max_v,x]=ordenar_matriz(A)

[r,c]=size(A);
B = reshape(A,1,[]);
[max_v,x] = sort(B,'descend');
```

6.2 Signal De-noising

In this section we will obtain signal de-noising by observing the wavelet transformation of the signal, and thresholding it to remove the frequencies of noise. We have applied this same reasoning to separate frequencies of a given signal that has distinctives high frequencies and low frequencies.

Example 6.4. *In both Figures 6.6 and 6.7 we have used the Daubechies6 transform of level* 4 *and* 5 *respectively. The processed signal is a sinusodial with Gaussian noise. If we consider the transform signal, we will see that noise is between the*

interval $[-3\ 3]$. *Then, by thresholding the transformation with this interval, we will obtain the desire signal. Notice that the more level we apply in the transform, the best results are obtained.*

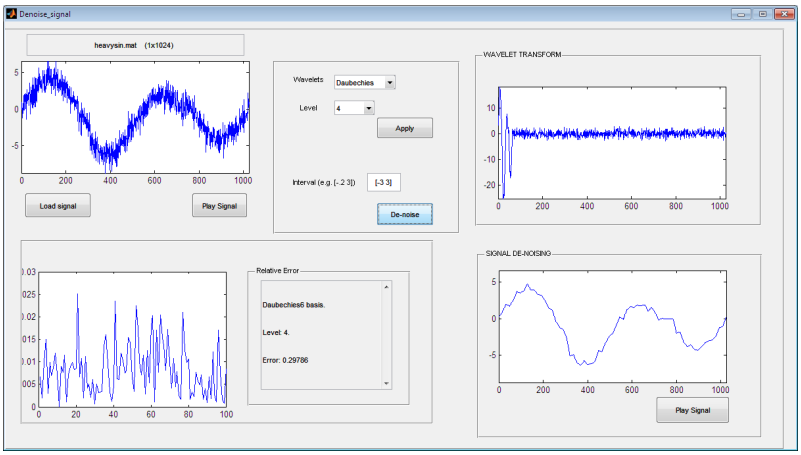


Figure 6.6: Signal De-noising.

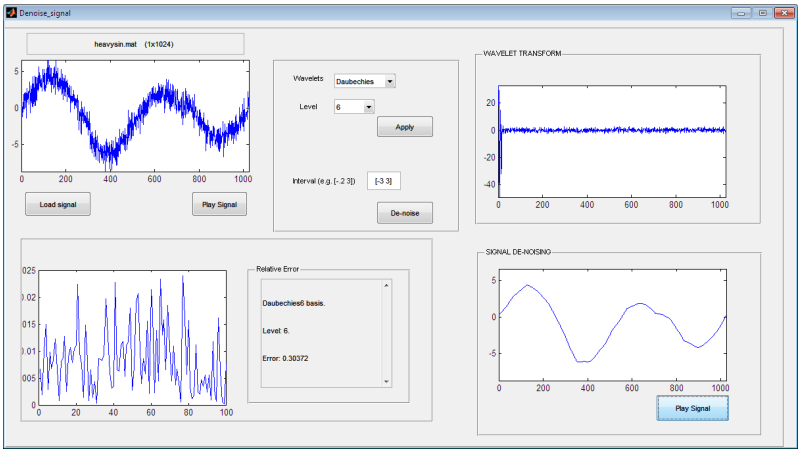


Figure 6.7: Signal De-noising.

Example 6.5. *Given a signal with two different frequencies, we attempt to separate them by thresholding the wavelet transform, obtaining the high frequencies and the low frequencies, and finally applying the inverse transform to each one and getting the two different signals.*

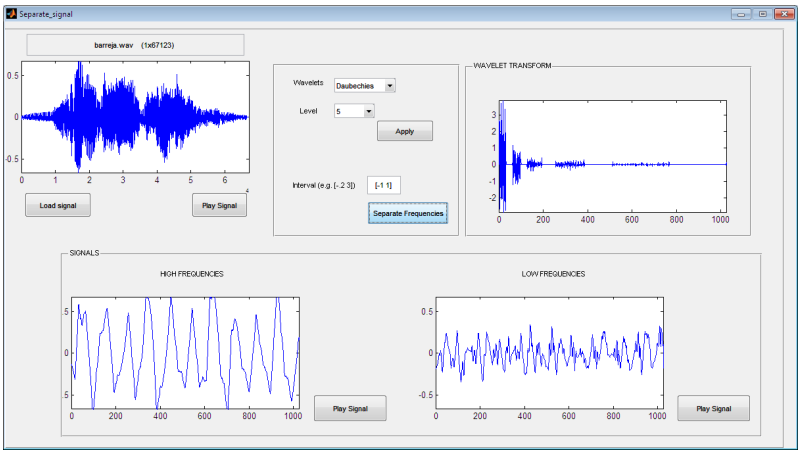


Figure 6.8: Separate frequencies *barreja.wav*.

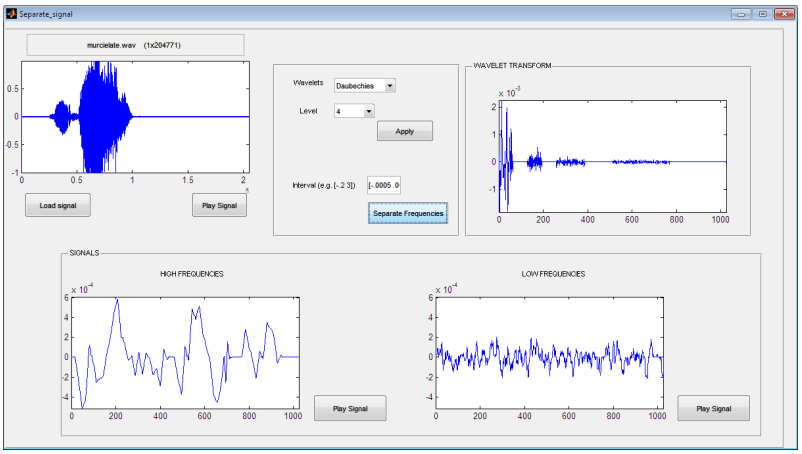


Figure 6.9: Separate frequencies *murcielate.wav*.

In the implementation of the program, we only have selected an array of lenght 1024 *in the signal, because the size of the original signals exceeds the memory of Matlab program. Even so, the frequencies are well separated in this region of the array.*

Example 6.6. *Here we deal with image de-noising. In Figure 6.10 we have the picture of lena.png with Gaussian noise. If we look to Figure 6.10, we see that the histogram of the image has form of a Gaussian function. Then, if we calculate the*

deviation of the original image, when we make the wavelet transform, we can delete the noise by thresholding with the value of its deviation.

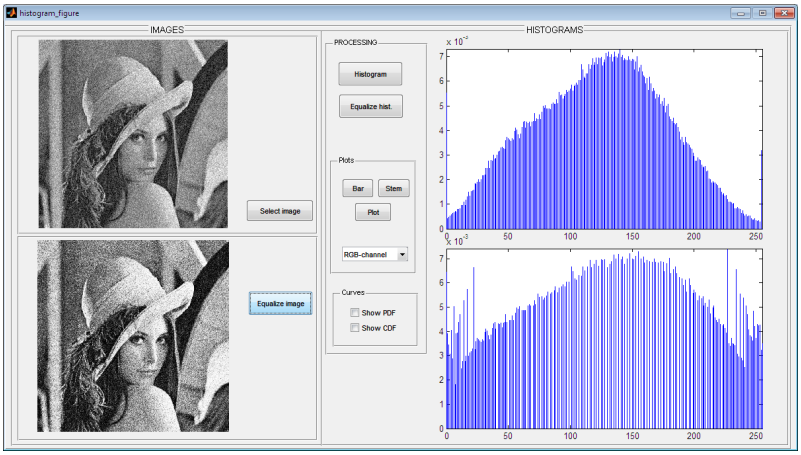


Figure 6.10: Histogram of *lena.png* with Gaussian noise.

In this example we have de-noised the image using two different values of threshold. In Figure 6.11 we have used the deviation of the original image, which is equal to 0.21*. In Figure 6.12 we have used three times the deviation of the image. We observe that in first figure the noise is not completely deleted, while in the second figure there is not noise, but all the image has been blured.*

(a) Original image. (b) Haar (c) Daubechies6

Figure 6.11: De-noising *lena.png* with Gaussian noise. Threshold: 0.21.

(a) Original image. (b) Haar (c) Daubechies6

Figure 6.12: De-noise *lena.png* with Gaussian noise. Threshold: 3*0.21.

6.2.1 Matlab implementation

Daubechies6 de-noising signal.

Code

```
function denoise = denoise_d61d(signal,k,T)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%%  Denoise signal k, T is an interval
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[r c] = size(signal);

denoise = signal.*(abs(signal)>T(2));

denoise = ID61D(denoise,k);
```

Daubechies6 separating frequencies.

Code

```
function [high,low] = separate_d61d(signal,k,T)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%%  Separates signal, level k, T is an interval
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[r c] = size(signal);

high1 = signal.*(abs(signal)>T(2));
low1 = signal.*(abs(signal)<T(2));

high = ID61D(high1,k);
low = ID61D(low1,k);
```

Image de-noising by *Daubechies6*.

Code

```
function denoise=denoise_image_d6(signal,level)

Fdenoise=signal;
dev = std2((signal));
denoise2 = Fdenoise.*(abs(Fdenoise)> 3*dev);
denoise = InvD6_rkl(denoise2,level);
```

Image de-noising by *Haar*.

Code

```
function denoise=denoise_image_haar(signal,level)
```

```
Fdenoise=signal;
dev = std2(abs(signal));
denoise2 = Fdenoise.*(abs(Fdenoise)> 3*dev);
denoise = Inv_rkl(denoise2,level);
```

Bibliography

[1] Michael W. Frazier, *An Introduction to Waveletes Through Linear Algebra*. Springer, 2000.

[2] Rafael C. González, Richard E. Woods and Steven L. Eddins, *Digital Image Processing Using MATLAB*. Pearson Prentice Hall, 2004.

[3] James S. Walker, *Fast Fourier Transforms*, 2nd Edition. CRC Press, 2003.

[4] James S. Walker, *Fourier Analysis and Wavelet Analysis*, Notices of the AMS **44** (1997), 658–668.

[5] James S. Walker and Amanda J. Potts, *Time-Frequency spectra of music*, SIAM Review **44** (2002), 457–475.