Sparse Fast Fourier Transform

Leo Turi
Registration Number: *1007564*
Supervisor: *Professor Michele Pavon*
Faculty of Engineering
Bachelor Degree in Information Engineering

July 22^{nd}, 2013

"Smoke · Roast = Const."

- ING. F. MARTINOLI -

Last month we had to sit through a presentation on eliminating redundancy, and it was a bunch of Power Point slides, plus a guy reading out what was on the slides, and then he gave us all hard copies. I don't understand these things.

- MAX BARRY, *COMPANY* -

Introduction

Our world is deeply influenced by information technology. It is taught in schools, it is a basic requirement in most companies and industries and, in general, it is and will be one of the leading sectors both for the present time and for the future. This was not the case for the past, because half a century ago information technology was still in its beginnings, and many did not believe in its importance. It is clear that they were mistaken, but their misbelief is comprehensible. Only few people would have been able to foresee the incredible growth this sector was about to experience.

Nowadays, it is still growing further, and many stunning theories are developed every month. Since 2004, a very beautiful theory has raised the interest of the scholars and technicians all around the world. It is known as Compressed Sensing (CS), and proposes to revolutionize the signal processing field. As of now, we are used to sample signals in their entirety and later compress then, either to save space while storing them or to save time while sending them. In this process there is a clear inefficiency: we store data that we will later discard. Compressed Sensing brings the compress process right to the sampling: we sample just the signifying parts of the signal. To do so, it chooses as a quality factor *signal sparsity*. Actually, many signals can be described by a few number of coefficients when represented under a certain basis. By sampling only the correct coefficients, then, we would be able to obtain a correct representation of the signal in *sub-linear* time. This is not just theory: it is possible also in practice.

The main benefit that CS theory would bring to signal processing is a significant speed boost to data acquisition. However, CS paradigms are not confined to just the sampling process, they are absolutely general, and many applications are possible.

What we focus on in this thesis, is a possible application of CS principles not to signal acquisition but to signal *analysis*. Specifically, we know that the most used method of analysis are the Fourier Transforms. This subfield of signal processing has become extremely important since the invention of the Fast Fourier Transform (FFT), an extremely fast algorithm to compute the Discrete Fourier Transform (DFT). This algorithm made practical what once was not, as *naive* DFT algorithms are computationally costly. Thanks to the FFT, this field has grown both in popularity and in utility.

It is common knowledge that there is no known algorithm which can improve over the FFT. There exist algorithms which are faster than FFT for particular cases, of course, but none is totally general. This has been true for a long time, but may not be so since May 2012.

In that period, a research team of the Massachussets Institute of Technology developed an algorithm known as Sparse Fast Fourier Transform (SFFT). This algorithm promises to improve over the FFT for almost every *sparse* signal. The theory applies almost the same principle as CS to Fourier Transform, and computes just the significant coefficients in the frequency domain in *sub-linear* time. This is something that not even the FFT is able to do, and thus this theory could bring a revolution in the signal analysis field.

In the first chapter, we review the basic knowledge necessary to understand the whole topic. Then, in the second chapter we introduce the FFT and some of its implementations, to show how the actual state-of-the-art algorithm is technically made. The third chapter is a brief introduction to CS theory, its principles and its solutions for sparse signal recovery. The chapter ends with some practical implementations of CS. The final and most important chapter deals with the SFFT algorithm developed by the MIT team, which goes under the name of Nearly Optimal Sparse Fast Fourier Transform. It is a very complex algorithm, that we introduce mainly in its theory. We also briefly speak about the probability of success of the algorithm, for it is not 100%. Finally, we show the original algorithm published by the team, and we list some benchmark examples against the best FFT implementation.

Contents

Chapter 1

Useful Concepts on Signals and Fourier Analysis

In this section we introduce the basic concepts required to understand what the Fast Fourier Transform aims to do. As signal transmission nowadays is almost only digital, we review only discrete-time signals, the family to which digital signals belong. We give the most useful definitions and introduce the main properties. Then, we review Hilbert Spaces, in order to give a proper background for the following analysis techniques we introduce. Those techiques, which are nothing but the various Fourier Transforms, form the most important knowledge necessary to understand the following chapters, and thus will be reviewed thoroughly.

1.1 Introduction to Discrete-Time Signals

Definition 1.1 (Discrete-Time Signal)**.** A discrete-time signal $x[n]$ is a real (or complex) valued function defined only for integer values of the indipendent variable n. Formally

$$x : \mathbb{N} \mapsto \mathbb{R} \text{ (or } \mathbb{C}) \tag{1.1}$$

The *Support* of this signal is the smallest set of values $[m_x, M_x]$ for which

$$x[n] = 0 \text{ if } n < m_x \text{ or } n > M_x, \tag{1.2}$$

and we will indicate it as $\text{supp}(x)$. In case this set is finite, we say that the signal has *limited support*.

We define the scalar product $\langle \cdot \rangle$ between two signals $x[n]$ and $y[n]$ as

$$\langle x, y \rangle = \sum_i x[i] \cdot \overline{y[i]} \tag{1.3}$$

and the Energy that every signal carries as the sum series over its support

$$E_x = \sum_{n=m_x}^{M_x} |x[n]|^2. \tag{1.4}$$

A signal is said to be *periodic* with period N if it is unchanged by a time shift of N, where N is a positive integer, i.e. if

$$x[n] = x[n+N], \forall n \in \mathbb{N} \tag{1.5}$$

The smallest positive value of N for which eq.(1.3) holds is the signal's *fundamental period* N_0.

In case a signal x[n] is *aperiodic*, or *non-periodic*, we can repeat $x[n]$ at a fixed rate over time to obtain a periodic signal:

Definition 1.2 (Periodic Repetition)**.** Given an aperiodic discrete time signal $x[n]$, we define its *periodic repetition* of period T as

$$\widehat{x}[n] = \sum_{k=-\infty}^{+\infty} x[t - kT]. \tag{1.6}$$

We note that, in case $T > \text{supp}(x)$, $x[n]$ equals the original signal over the single period.

Fundamental discrete-time signals are the *Complex Exponential Signals*, defined by

$$x[n] = e^{j\omega_0 n} \tag{1.7}$$

and the *Sinusoidal Signals*, defined by

$$x[n] = A\cos(\omega_0 n + \psi). \tag{1.8}$$

These signals are closely related to each other through Euler's Formula,

$$e^{j\omega_0 n} = \cos(\omega_0 n) + j\sin(\omega_0 n). \tag{1.9}$$

We note that the fundamental complex exponential $e^{\frac{2\pi}{N}n}$ is periodic with period N. Furthermore, the set of all discrete-time complex exponential signals periodic with period N is given by

$$\psi_k[n] = e^{jk\omega_0 n} = e^{jk\frac{2\pi}{N}n},\ k = 0, \pm 1, \pm 2, \ldots \tag{1.10}$$

All of these signals have fundamental frequencies that are multiples of $2\pi/N$ and thus they are said to be *harmonically related*. There are only N distinct signals in the set given by eq.(1.10), as discrete time complex exponentials which differ in frequency by a multiple of 2π are identical. Hence, in eq.(1.10) it suffices to tale k = 0, 1,..., N-1.

Moreover, this family of signals has also the particularity of being *orthogonal*. First, though, we need to introduce another concept:

Definition 1.3 (ℓ^2 Space)**.** Discrete-time signals which have finite energy over the whole time domain belong to the signal space called *square summable* or ℓ^2 space, i.e.

$$\ell^2 = \{x \in x[n]_{n\in\mathbb{Z}} : \exists M \in \mathbb{R} : |E_x| < M\} \tag{1.11}$$

Definition 1.4 (Orthogonality and Orthonormality)**.** Two ℓ^2 signals $x[n], y[n] \in H$ are said to be orthogonal when

$$\langle x, y\rangle = \sum_{i=-\infty}^{+\infty} x[i]\overline{y[i]} = 0. \tag{1.12}$$

A family of signals $\{\psi_1 \dots \psi_n\}$ of length n is said to be orthogonal when

$$\langle \psi_i, \psi_j\rangle = 0, \quad \text{iff} \quad i \neq j. \tag{1.13}$$

Last, a family of signals is said to be *orthonormal* if it is orthogonal and each of its vectors has norm equals to 1, i.e. when

$$\begin{cases} \langle \psi_i, \psi_j\rangle = 0 & \text{iff} \quad i \neq j \\ \\ \langle \psi_i, \psi_i\rangle = ||\psi_i||^2 = 1 & \text{iff} \quad i = j \end{cases} \tag{1.14}$$

As an example, the normalized family of complex exponential

$$\psi_k[n] = \frac{1}{\sqrt{N}} e^{jk\frac{2\pi}{N}n}, \; k = 0, \dots, N-1 \tag{1.15}$$

is indeed orthonormal.

Orthogonal and orthonormal families will be thoroughly used in the following sections. As of now, in order to give proper definitions later on, we need to introduce the concept of Hilbert space. The knowledge of vector spaces is given for granted, yet, for any further insight, we suggest reading an equivalent of [1].

1.2 Hilbert Space

The ideal space for Fourier Analysis is a Hilbert Space, in which series converge to a vector. We note that what we are defining is properly a *vector* space, but we can exploit a direct connection between finite signals and vectors directly from the definition we gave of discrete time signal.

Definition 1.5 (Hilbert Space)**.** Be H a complex vector space [1] on which we can define a map:

$$\begin{array}{l} H \times H \longrightarrow \mathbb{C} \\ (x, y) \longrightarrow \langle x, y \rangle_H \end{array} . \tag{1.16}$$

This map is known as *scalar* or *internal product*, and has the following properties:

(i) $\langle y, x \rangle = \langle x, \overline{y} \rangle$, where $\overline{y}$ stands for y's complex conjugate;

(ii) $\langle x + y, z \rangle = \langle x, z \rangle + \langle y, z \rangle$

(iii) $\langle \alpha x, y \rangle = \alpha \langle x, y \rangle$ where $\alpha \in \mathbb{C}$

(iv) $\begin{cases} \langle x, x \rangle = 0 & \text{if } x = 0 \\ \langle x, x \rangle \geq 0 & \text{otherwise} \end{cases}$

(v) (Cauchy-Schwartz inequality)

$$|\langle x, y \rangle| \leq ||x||_H \cdot ||y||_H \tag{1.17}$$

(vi) (Norm)

$$||x||_H = \langle x, x \rangle^{\frac{1}{2}} \tag{1.18}$$

In case every Cauchy sequence, i.e. each sequence $\{x_n\}_{n \in \mathbb{N}}$ with values in H such that

$$\forall \epsilon > 0 \;\; \exists N(\epsilon) = N : \forall m, n \geq N, \;\; ||x_m - x_n|| < \epsilon, \tag{1.19}$$

has limit inside H, then we say that H is *complete* and call it Hilbert Space. An example of an infinite dimensional Hilbert Space is provided by ℓ^2, see 1.3.

1.3 Fourier Analysis

In this section we will review the different kind of Fourier Representations and Transforms, and end up introducing the Discrete Fourier Transform, which is the basis which the second chapter stems from.

1.3.1 Fourier Series Representation of Discrete-Time Signals

What we previously introduced wouldn't be worth of vast consideration without the work of Jean-Baptiste Joseph Fourier, a French mathematician and physicist who lived across the 18th and 19th centuries. He had the idea that any complex *periodic* function could be represented through the superposition of a (possibly infinite) series of periodic functions, namely

simple sines and cosines. His idea was not entirely new, as already Euler, D'Alembert, Bernoulli and Gauss had used similiar trigonometric series in their studies. Yet, contrary to them, Fourier believed that such a series - what we now call *Fourier Series* - could represent *any arbitrary function*. This totally revolutionized both mathematics and physics.

In particular, in the field of signals, the Fourier series representation of a discrete time *periodic* signal is a *finite* sum. Given a family of harmonically related complex exponentials, as defined in eq.(1.10), we can consider the representation of a periodic signal, through the linear combination of the $\Psi_k[n]$. Thus

$$x[n] = \sum_{k=0}^{N-1} a_k e^{jk\omega_0 n}, \quad \text{with} \quad \omega_0 = \frac{2\pi}{N}. \tag{1.20}$$

This equation is referred to as the *discrete-time Fourier series*. Since the exponentials $\Psi_k[n]$ are linearly indipendent, it represents a set of N indipendent linear equations for the N unknown coefficients a_k, and thus the system is determined. Then, we can solve eq.(1.20) backwards to obtain the a_k coefficients, as

$$a_k = \frac{1}{N} \sum_{n=0}^{N-1} x[n] e^{-jk\frac{2\pi}{N}n}. \tag{1.21}$$

We note that even in case we considered more than N sequential values of k, the values a_k would repeat periodically with period N as a consequence of eq.(1.10). The whole information associated to the periodic signal $x[n]$ is then contained in its N Fourier series coefficients. This means that, once we fix the harmonically related family of signals, $\Psi_k[n]$, x[n] is equally described by its discrete-time representation (in the time domain) and by its Fourier coefficients (in the frequency domain). Then we can identify a relationship between these two kind of representations, i.e.

$$x[n] \overset{\mathcal{F}_s}{\longleftrightarrow} a_k \tag{1.22}$$

described by the *Discrete-Time Fourier Series, (DTFS)*. It is an almost isometric map

$$\mathcal{F}_s : \mathbb{C}^N \to \mathbb{C}^N \tag{1.23}$$

enjoying several properties:

- (Linearity)

$$Az[n] + By[n] \overset{\mathcal{F}_s}{\longleftrightarrow} Aa_k + Bb_k; \tag{1.24}$$

- (Time Shifting)

$$x[n - n_0] \overset{\mathcal{F}_s}{\longleftrightarrow} a_k e^{-j\frac{2\pi}{N}n_0}; \tag{1.25}$$

- (Time Reversal)
$$x[-n] \overset{\mathcal{F}_s}{\longleftrightarrow} a_{-k}; \tag{1.26}$$

- (Multiplication)
$$x[n]y[n] \overset{\mathcal{F}_s}{\longleftrightarrow} \sum_{l=0}^{N-1} a_l b_{k-l} = a_k * b_k; \tag{1.27}$$

- (Periodic Convolution)
$$\sum_{r=0}^{N-1} x[r]y(n-r) = x[n] * y[n] \overset{\mathcal{F}_s}{\longleftrightarrow} N a_k b_k; \tag{1.28}$$

- (Conjugation)
$$\overline{x[n]} \overset{\mathcal{F}_s}{\longleftrightarrow} \overline{a}_{-k}; \tag{1.29}$$

- (Energy Conversion)
$$||a||^2 = \frac{1}{N}||x[n]||^2 \tag{1.30}$$

where $x[n]$ and $y[n]$ are periodic signals with period N and a_k and b_k are their Fourier series coefficients.

1.3.2 Discrete-Time Fourier Transform

In the previous section, we showed that the Fourier Series representation applies to periodic signals. In this section we show how, for finite energy signals, we can extend Fourier analysis to *aperiodic* signals.

Given the aperiodic signal $x[n] \in \ell^2$ we first define its periodic repetition $\tilde{x}[n]$ of period T, with $T > \text{supp}(x)$, so that over one period the two signals are equal. Now, as T approaches ∞, it is immediate that $\tilde{x}[n] = x[n]$ for *any* finite value of n. Let us now examine the DTFS of $\tilde{x}[n]$, and in particular its Fourier Series coefficients:

$$\tilde{x}[n] = \sum_{k=0}^{N-1} a_k e^{jk\frac{2\pi}{N}n}, \tag{1.31}$$

$$a_k = \frac{1}{N}\sum_{n=0}^{N-1} \tilde{x}[n] e^{-jk\frac{2\pi}{N}n}. \tag{1.32}$$

Since $\tilde{x}[n] = x[n]$ over a period that includes the support of $x[n]$, we can substitute $x[n]$ in eq.(1.33). Moreover, $x[n]$ is zero outside its support, thus we

can extend the series to the whole discrete-time domain without modifying the result:

$$a_k = \frac{1}{N}\sum_{n=0}^{N-1} x[n]e^{-jk\frac{2\pi}{N}n} = \frac{1}{N}\sum_{n=-\infty}^{+\infty} x[n]e^{-jk\frac{2\pi}{N}n}. \tag{1.33}$$

We now have set all the basis to define the discrete time Fourier transform.

Definition 1.6 (Discrete-Time Fourier Transform)**.** The *Discrete-Time Fourier Transform*, (DTFT), of the aperiodic signal $x[n] \in \ell^2$ is defined as the mean square limit

$$X(e^{j\omega}) = \sum_{n=-\infty}^{+\infty} x[n]e^{-j\omega n}. \tag{1.34}$$

We note that the coefficients a_k are proportional to samples of $X(e^{j\omega})$, i.e.,

$$a_k = \frac{1}{N}X(e^{jk\omega_0}), \tag{1.35}$$

where $\omega_0 = 2\pi/N$ is the spacing of the samples in the frequency domain.

Combining equations eq.(1.31) with eq.(1.35) as well as $1/N = \omega_0/2\pi$ yields

$$\tilde{x}[n] = \frac{1}{2\pi}\sum_{k=0}^{N-1} X(e^{jk\omega_0})e^{jk\omega_0 n}\omega_0. \tag{1.36}$$

Now, we can note two things: first, both $X(e^{j\omega})$ and $e^{j\omega n}$ are periodic of period 2π, and so their product is periodic as well; second, each term of the summation represents the area of a rectangle of height $X(e^{jk\omega_0})e^{jk\omega_0 n}$ and width ω_0, thus, as N approaches to ∞, ω_0 becomes infinitesimal, and so the summation passes to an integral. Therefore, as $N \to \infty$, $\tilde{x}[n] = x[n]$, from equation eq.(1.36) we can obtain the original signal $x[n]$, thus defining the Inverse DTFT:

Definition 1.7 (Inverse Discrete-Time Fourier Transform)**.** Given the complex function $X(e^{jk\omega_0})$ defined in eq.(1.34), the Inverse DTFT grants the possibility to reconstruct the original aperiodic signal $x[n]$ as

$$x[n] = \frac{1}{2\pi}\int_{2\pi} X(e^{j\omega})e^{j\omega n}d\omega, \tag{1.37}$$

where the interval of integration can be taken as *any* interval of length 2π, due to the periodicity. We must point out that the function $X(e^{jk\omega})$ must be integrable over its period, but this is assured by the fact that $x[n] \in \ell^2$.

Theorem 1.3.1 (Placherel's Theorem)**.** *Given two signals $x[n], y[n] \in \ell^2$ and their respective transforms $X(e^{j\omega})$ and $Y(e^{j\omega})$, then*

$$\sum_{n=-\infty}^{+\infty} x[n] \cdot \overline{y[n]} = \frac{1}{2\pi} \int_{2\pi} X(e^{j\omega}) \cdot \overline{Y(e^{j\omega})} d\omega \tag{1.38}$$

In particular $x[n] = y[n]$, we get $||x[n]||^2 = ||X(e^{j\omega}||^2$.

Thus, we now have a way to analyze every aperiodic finite energy discrete time signal in the frequency domain.

1.3.3 Discrete Fourier Transform

The discrete Fourier analysis is one of our matter of interest. Still, it is impossible to apply as it is, as the frequency domain of the tranforms is still infinite and continous. One of the reason for the great use of discrete time methods for the analysis of signals is the development of exceedingly efficient tools for performing Fourier analysis of discrete-time sequences. At the heart of these methods is a technique that is very closely allied with the discrete-time Fourier analysis and that is ideally suited for use on a digital computer due to some fancy properties. This technique is the Discrete Fourier Transform, or DFT, for limited-support signals.

Definition 1.8 (Discrete Fourier Transform)**.** Let $x[n]$ be a limited support signal and $\tilde{x}[n]$ be its periodic repetition of period N. The Fourier series coefficients for $\tilde{x}[n]$ are given by

$$a_k = \frac{1}{N} \sum_{n=0}^{N-1} \tilde{x}[n] e^{-jk\frac{2\pi}{N}n}. \tag{1.39}$$

By choosing the interval $\{N\}$ where $\tilde{x}[n] = x[n]$, the coefficients a_k define the *Discrete Fourier Transform* of $x[n]$, i.e.

$$\widehat{X}[k] = a_k = \frac{1}{N} \sum_{n=0}^{N-1} x[n] e^{-jk\frac{2\pi}{N}n}. \tag{1.40}$$

The importance of the DFT stems from several properties:

- (Completeness) The DFT is an invertible linear transformation

$$\mathcal{F} : \mathbb{C}^N \to \mathbb{C}^N \tag{1.41}$$

and its *inverse transform* is defined as

$$x[n] = \sum_{k=0}^{N-1} \widehat{X}[k] e^{jk\frac{2\pi}{N}n}, \quad \text{with} \quad n = 0, 1, 2, \ldots, N-1. \tag{1.42}$$

- (Orthogonality) The vectors $u_k = [e^{j\frac{2\pi}{N}kn}, n = 0, \ldots, N-1]$ form an orthogonal basis over the set of N-dimensional complex numbers;

- (Parseval's Theorem for the DFT)

$$\sum_{n=0}^{N-1} x[n]\overline{y[n]} = \frac{1}{N}\sum_{k=0}^{N-1} \widehat{X}[k]\overline{\widehat{Y}[k]} \tag{1.43}$$

- (Periodicity)

$$\begin{aligned}
\widehat{X}[k+N] &= \frac{1}{N}\sum_{\{N\}} x[n]e^{-j(k+N)\frac{2\pi}{N}n} = \\
&= \frac{1}{N}\sum_{\{N\}} x[n]e^{-jk\frac{2\pi}{N}n}\underbrace{e^{-j2\pi n}}_{=1} = \\
&= \frac{1}{N}\sum_{\{N\}} x[n]e^{-jk\frac{2\pi}{N}n} = \\
&= \widehat{X}[k]
\end{aligned} \tag{1.44}$$

- (Sampled DTFT) The DFT coefficients of a signal can be seen as the corresponent DTFT values *sampled* with frequency $\omega = 2\pi/N$, i.e.

$$\widehat{X}[k] = \frac{1}{N}X(e^{j\frac{2\pi}{N}k}) \tag{1.45}$$

  where $X(e^{j\omega})$ is the DTFT of x[n].

Chapter 2

The Fast Fourier Transform

In this chapter we introduce the reader to the Fast Fourier Transform, to its history and its development, to prepare the background over which we we will show some of the latest and most used versions of this extremely important algorithm. We will also give a look at the C++ implementation of the Cooley-Tuckey Algorithm, so that comparisons will be easy to do when we will talk about the Sparse FFT.

2.1 What is the FFT?

The Fast Fourier Transform is an algorithm, as we said, to compute the DFT and its inverse. Its importance derives from the fact that it made the Fourier analysis of digital signal an affordable task: an algorithm which naively implements DFT definition takes $\mathrm{O}(N^2)$ arithmetical operations to define the set of N coefficients, while an FFT is much faster, and can compute the same DFT in only $\mathrm{O}(N \log N)$ operations. This time complexity is often called *linearithmic*. The difference in speed can be enormous, especially for long data sets, where N could be $10^{6\sim9}$. Moreover, many FFT have an interesting feature that grants an increase on accuracy at the expense of increase computations. We say "many" due to the fact that there is NOT a single FFT algorithm, instead there are many different algorithms which involve a wide range of mathematics to attain high speed. We cannot see all of them, but the reader is free of reviewing what we left out in [3].

One of the most used is the Cooley-Tukey Algorithm (CT), which was developed in 1965 and which we will review later on in this chapter. It is a divide and conquer algorithm that recursively breaks down a DFT of any composite size $N = N_1 N_2$ into many smaller DFTs of sizes N_1 and N_2, along with $\mathrm{O}(N)$ multiplications by complex root of unity traditionally called *twiddle factors*.

Another wide spread algorithm cares about the case in which N_1 and N_2 are coprime, and is known as Prime-Factor (or Good-Thomas) algorithm.

It is based on the Chinese Remainder Theorem, which is used to factorize the DFT similiarly to CT but without the twiddle factor.

Other algorithms are Bluestein's, Rader's and Bruun's FFT algorithms, which are variants of the CT algorithm based on polynomial factorization.

Eventually, to give the FFT a proper background, it is good to review its history.

2.1.1 History of the FFT

The history of the FFT is particular. No clear traces can be found of the "first" algorithm written that speeds up the DFT - or any equivalent sinusoid expansion - and a long road must be travelled to define its history. Yet, its history, as we'll see, draws a beautiful circle in time.

Modern FFT history starts in 1965 with two American mathematicians, James Cooley and John William Tukey, who developed the omonimous algorithm, *Cooley-Tukey (CT)*. Soon after them, another mathematician who goes by the name of P. Rudnick demostrated a similiar algorithm based on the work of Danielson and Lanczos, which appeared in 1942. The discovery of these early studies prompted an investigation into the history of the FFT [4].

It has been discovered that many scientists had used algorithm with the same aim as the FFT. Among those we can list George Howard Darwin (son of the more famous Charles), Lord Kelvin, Francesco Carlini and Peter Andreas Hansen. But a single footnote on the work *A History of Numerical Analysis from the 16th Through the 19th Century* by H. Goldstine casted light on who could have first worked on a solid FFT algorithm:

> This fascinating work of Gauss was neglected and was rediscovered by Cooley and Tukey in an inportant paper in 1965.

The quotation goes back to the beginning of the 19th Century, to a treatise written by Carl Friedrich Gauss, entitled "Theoria Interpolationis Methodo Nova Tractata". The German scientist extended Euler's and Lagrange's studied on series expansion while considering the problem of determining the orbit of certain asteroids from sample locations. These functions were expressed by a Fourier series, and Gauss showed on his treatize on interpolation that its coefficients were given by the now well-known formulas for the DFT. This set of equations is the earliest explicit formula for the general DFT that has been found.

Gauss used the algorithm to compute the Fourier transform passing by a set of N_1 samples, and then reapplied it to compute another Fourier transform passing by another set of N_1 samples that were the offsets from the positions of the first set. The two sets of coefficients were though completely different, showing that the algorithm could be flawed. With modern termi-

nology, we can instead say that the waveform was simply undersampled, and that therefore the coefficients were in error because of aliasing of the high frequency harmonics of the algorithm.

The solution Gauss gave for this issue was very important. He measured a total of N_2 sets of N_1 equally spaced samples, which together form an overall set of $N = N_1 N_2$ equally spaced samples. Then, the Fourier series for the N samples is computed by first computing the coefficients for each of the N_2 sets of length N_1, and then computing the coefficients of the N_1 series of length N_2. After some algebric steps, what he got was the following equation:

$$C(k_1 + N_1 k_2) = \sum_{n_2=0}^{N_2-1} [\sum_{n_1=0}^{N_1-1} X(N_2 n_1 + n_2) W_{N_1}^{n_1 k_1} W_{N_2}^{n_2 k_2}] W_{N_2}^{n_2 k_2} \qquad (2.1)$$

This is exactly the exponential form of Gauss' algorithm, where the W_N term accounts for the shift from the origin of the N_2 length-N_1 sequences.

This algorithm actually computes the DFT in a surprisingly fast way, a feature that Gauss did not analyse. Yet, there is a very curious fact with the algorithm: it is, *exactly*, the FFT algorithm derived by Cooley and Tukey in 1965, no less than 150 years later. The same W_N factor is called a *twiddle factor*, a factor to correct the DFT of the inner sum for the shifted samples of $X(n)$, exactly what Gauss found.

This states the history of the FFT to be far longer than the trivial 50 years it's been used up to now. Yet, this also states that, in a way or another, the "best" algorithm that could lately be found was the same as two centuries ago. It is from this point of view that we can see the importance of developing an algorithm *faster* than the FFT: to resume the natural course of science evolution.

Let's now focus just on CT Algorithm, and get to know better the beauty of the FFT.

2.2 Cooley and Tukey's FFT

CT algorithm comes in different forms, based on how the N samples are factorized. The simplest and most common form is a *radix-2* decimation-in-time FFT (*Radix-2 DIT*), in which the algorithm divides a DFT of size N into two interleaved DFTs of size $N/2$ with each recursive stage.

Radix-2 DIT first computes the DFTs of the even-indexed inputs $x[2m]$ and of the odd-indexed inputs $x[2m + 1]$, and then combines those two results to produce the DFT of the whole sequence. There are two ways of implementing the algorithm: one is the recursive way, which has lower running times but uses a great amount of memory; the other is the *in-place* way, which is slower but very light in memory consumption. By the way,

there's an implicit assumption: the length of the samples $x[n]$ is a power of two, but usually this is not a hard restriction.

The mathematical form of the reconstructed DFT is

$$X[k] = \sum_{m=0}^{N/2-1} x[2m]e^{-j2km\frac{2\pi}{N}} + e^{-jk\frac{2\pi}{N}} \sum_{m=0}^{N/2-1} x[2m+1]e^{-j2km\frac{2\pi}{N}} \tag{2.2}$$

where the exponential outside the sum is the *twiddle factor*. We can call the even-indexed input DFT $E[k]$ and the odd-indexed input DFT $O[k]$, obtaining the form

$$X[k] = E[k] + e^{-j\frac{2\pi}{N}k}O[k]. \tag{2.3}$$

We need to compute only $N/2$ outputs: thanks to the periodicity of the DFT, the outputs for $N/2 \leq k < N$ from a DFT of length $N/2$ are identical to the outputs for $0 < k \leq N/2$. What changes is the *twiddle factor*, which flippes the sign of the $O[k + N/2]$ terms. Thus the whole DFT can be calculated as follows:

$$X[k] = \begin{cases} E[k] + e^{-j\frac{2\pi}{N}k}O[k] & \text{if} \quad k < N/2 \\ \\ E[k - N/2] - e^{-j\frac{2\pi}{N}(k-N/2)}O[k - N/2] & \text{if} \quad k \geq N/2 \end{cases} \tag{2.4}$$

The above re-expression of an N-size DFT as two N/2-size DFTs is also called the Daniel-Lanczos lemma, since the identity was noted by those two authors in 1942: they followed the recursion backwards until the transform spectrum converged to the original spectrum. By doing so, curiously, they didn't notice that they achieved the asymptotic complexity of $N \log N$, which is now the common standard for an ideal FFT.

We shall now analyse how the algorithm Cooley-Tukey can be implemented and the issues linked to each implementation.

2.3 Implementations

At first, Cooley and Tukey designed the algorithm to be implemented recursively. A very simple code for this implementation is Algorithm 2.3.1:

While this granted the best performance attainable, the memory required for large set of data was improbe. Thus, they investigated the problem of devising an *in-place* algorithm that overwrites the input with its output data using only O(1) auxiliary storage.

Their solution was to apply a reordering technique, called *bit-reversal*, to the Radix-2 DIT. Bit-reversal is the permutation where the data at an index n, written in binary with digits $b_4b_3b_2b_1b_0$ is transferred to the index with reversed digits $b_0b_1b_2b_3b_4$.

```
procedure RCT(x) {
    n = length(x);
    if (n == 1) then
        return x;
    end if
    m = n/2;
    X = (x_{2j})_{j=0}^{m-1};
    Y = (x_{2j+1})_{j=0}^{m-1};
    X = RCT(X);
    Y = RCT(Y);
    U = (X_{kmodm})_{k=0}^{n-1};
    V = (g^{-k} Y_{kmodm})_{k=0}^{n-1};
    return U + V;
end procedure
```

Algorithm 2.3.1: Out-of-place Cooley-Tukey FFT pseudocode

This choice was due to the fact that, when we try to overwrite the input with the output at the last stage of the algorithm given by eq.(2.4), the digits we obtain should go in the first and second *halves* of the output array, which corresponds to the *most* significant bit b_4, whereas the two inputs $E[k]$ and $O[k]$ are interleaved in the *even and odd elements*, corresponding to the *least* significant bit b_0. Thus, if we perform every step of the transform, we obtain that *all* the bits must be switched to perform the in-place FFT, and this requires the use of the bit-reversal technique.

The following C++ algorithm is an in-place implementation, which explicitly applies the bit-reversion to the input array.

```
//in-place Cooley-Tukey FFT algorithm with bit-reversal

void CTFFT(double* data, unsigned long nn)
{
    unsigned long n, mmax, m, j, istep, i;
    double wtemp, wr, wpr, wpi, wi, theta;
    double tempr, tempi;

    // reverse-binary reindexing
    n = nn<<1;
    j=1;
    for (i=1; i<n; i+=2) {
        if (j>i) {
            swap(data[j-1], data[i-1]);
            swap(data[j], data[i]);
        }
        m = nn;
```

```
        while (m>=2 && j>m) {
            j -= m;
            m >>= 1;
        }
        j += m;
    };

    // here begins the Danielson-Lanczos section
    mmax=2;
    while (n>mmax) {
        istep = mmax<<1;
        theta = -(2*M_PI/mmax);
        wtemp = sin(0.5*theta);
        wpr = -2.0*wtemp*wtemp;
        wpi = sin(theta);
        wr = 1.0;
        wi = 0.0;
        for (m=1; m < mmax; m += 2) {
            for (i=m; i <= n; i += istep) {
                j=i+mmax;
                tempr = wr*data[j-1] - wi*data[j];
                tempi = wr * data[j] + wi*data[j-1];

                data[j-1] = data[i-1] - tempr;
                data[j] = data[i] - tempi;
                data[i-1] += tempr;
                data[i] += tempi;
            }
            wtemp=wr;
            wr += wr*wpr - wi*wpi;
            wi += wi*wpr + wtemp*wpi;
        }
        mmax=istep;
    }
}
```

Algorithm 2.3.2: In-place Cooley-Tukey FFT C++ algorithm

Variations of Cooley-Tukey algorithm, as well as many other algorithms, are used in one of the fastest C++ FFT implementations, which is known as the Fastest Fourier Transform in the West (FFTW) [5]. It is one of the best implementations existent, and can compute transforms of real and complex-valued arrays of arbitrary size and dimension in O(N log N) time. The benchmarks against most of the used FFT algorithms [6] attest that it effectively is one of the fastest algorithms developed, and its performances, when tested on Intel® hardware, are almost on par with Intel®'s own highly optimized IPPS routines [7].

2.4 Runtimes

As we previously said, ideal algorithms for the FFT have *linearhithmic* time complexity $O(N \log N)$. As the algorithms are implemented through various programming languages, though, slow downs may occur depending on the different implementations. Usually, recursive implementations requires time to refine memory usage, and iterative finds its bottleneck with 32-bit system and large sets of data.

Figure (2.1) shows the running times of the C++ implementations of the previous in-place algorithm [CTFFT], of a non optimized out-of-place recursive algorithm [RCT (sub-optimal)], and, last, of a very optimized version of the recursive algorithm [RCT (optimized)] [8]. The optimizations done to the recursive algorithm care about complex root of unity recursive calculus, which appears also in the iterative algorithm, and generally specialize the FFT to be short and extremely fast.

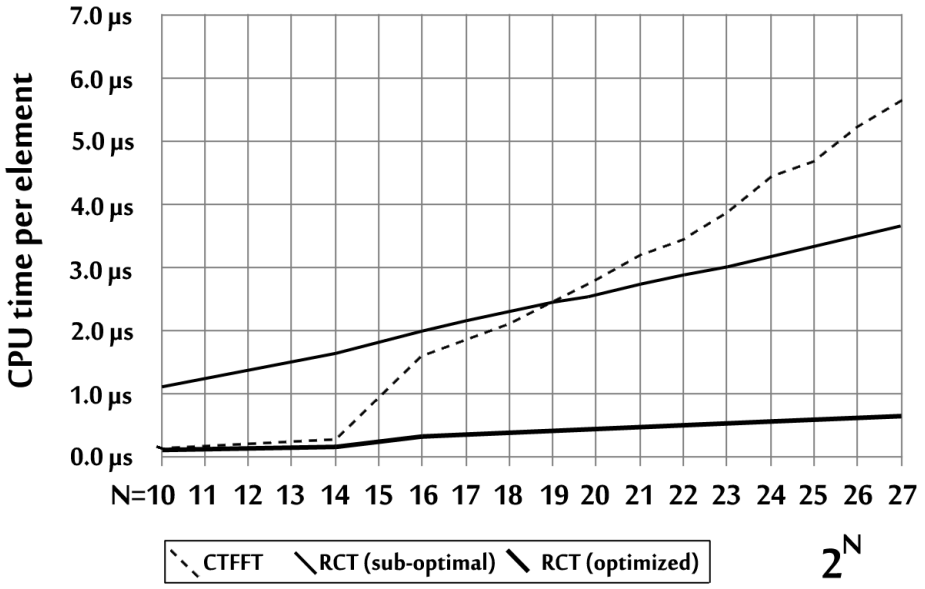

Figure 2.1: FFT C++ algorithms comparison.

The running time of the iterative algorithm results to be low for small values of n, but drastically increases as n grows. This is due to the actual limitations of CPUs. The test was run on a 32-bit system, but a 64-bit system behaved almost the same only for higher values of n. On contrary, the recursive algorithm needs to be optimized in order to obtain very good results, although it is clear that its slow-down ratio is drastically less steep than the iterative algorithm, both before and after optimization. The draw-

back is the load on the main memory, which is drastically higher than in the iterative case, although the high availability of RAM in modern computers renders this drawback less important.

2.5 Example

Let's see now an example of Fourier analysis of a picture.

As the Fourier transform is composed of complex values, it can be visualized as an image as well. In most implementations the image mean, F(0,0), is displayed in the center of the image. The further away from the center an image point is, the higher is its corresponding frequency; the whiter it is, the higher is the coefficient corresponding to that frequency.

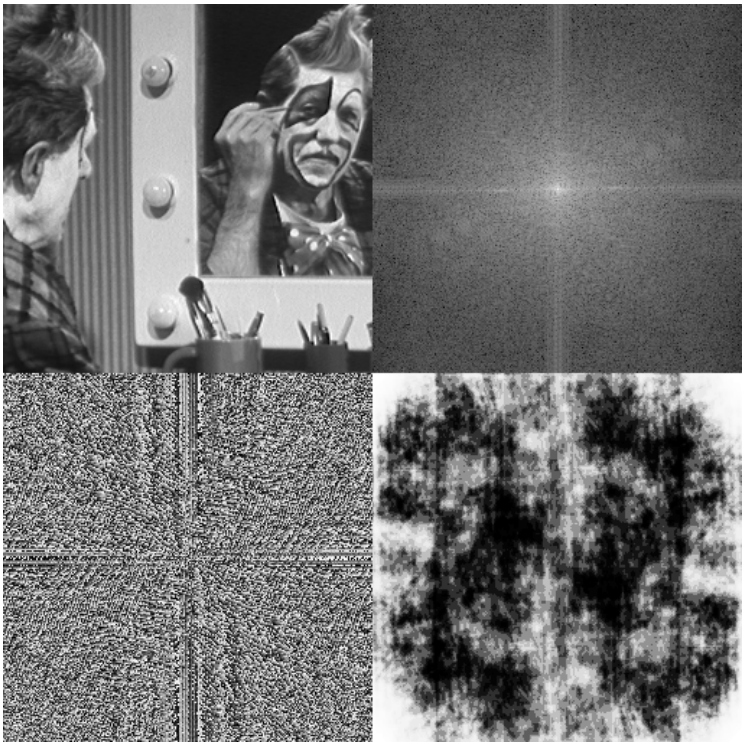

Figure 2.2: Fourier Analysis of an Image

Figure (2.2) is composed of four images. The upper left is our original image. Its transform presents the whole range of frequencies, thus we need to display it in logarithmic form so that the image doesn't appear pitch black. The result is the top right image, and shows that the image contains components of all frequencies, but that their magnitude gets smaller for higher frequencies. Hence, low frequencies contain more image information

than the higher ones. The transform image also tells us that there are two dominating directions in the Fourier image, one passing vertically and one horizontally through the center. These originate from the regular patterns in the background of the original image.

The third image, in the lower left corner, is the phase of the Fourier transform. The value of each point determines the phase of the corresponding frequency. The phase image does not yield much new information about the structure of the spatial domain image; however, note that if we apply the inverse Fourier Transform to the above magnitude image while ignoring the phase we obtain the bottom right image: it carries all the components of the original image, but without phase information it results corrupted beyond recognition.

Chapter 3

Compressed Sensing Theory

In this chapter we introduce the main features of the Compressed Sensing theory, and show some practical applications.

3.1 Sample Less to Compress More

Nowadays we are overwhelmed by information. Impressive amounts of data are constantly stored in our personal computers, notebooks and smartphones, and we are used to access to them frequently and to frequently share them with others. When faced with the huge capacity of modern portable storage devices, we seldom care about the dimension of the files we store and share. But, when it comes to data sharing on an internet connection, we suddenly start to care about how much we receive and how much we transmit, as sharing takes time.

To speed things up, the easiest thing to do is *not* to transmit useless informations. This can be easily done by *compressing* datas before sending them. There are two forms of compression: lossless and lossy. The first grants the possibility to recover the whole set of data by decompression, but compression rates are low. An example are the PNG and BMP image formats. The second allows very high compression rates (90% of the file size can be omitted), yet does not allow complete recovery of the original datas, as the less useful informations have been discarted douring compression. Examples of this are the JPEG, JPG and JPEGmini image formats.

We can observe that, when we transmit lossy-compressed files, then those files will remain that way forever. There is no point in trying to decompress them, as some share of their information was lost. But, this raises a question: in case we always shared lossy-compressed file, wouldn't obtaining the whole original files in many case become obsolete?

This is in part true. Thus, in certain cases, if we could be able to sample *directly* compressed original files, we would never need to compress data again, thus losing redundancy in the whole process.

This is exactly the idea which lies behind the Compressed Sensing (CS) theory. It is a relatively new theory, which has been developed during the last few years. It caused a great interest in the scientific world near 2006, thanks to the work of Emmanuel Candes, Terence Tao, David Donoho and Justin Romberg, who obtained important results towards a *practical* compressed sampling method.

CS theory asserts that one can recover certain signals from far fewer samples or measurements than traditional methods use. To accomplish this, CS relies on two principles: *sparsity* and *incoherence*.

- Sparsity expresses the idea that, with respect to a suitable basis Ψ, most signals of interest have many zero coefficients, and thus contain less information than what we could guess from their size.

- Incoherence says that, the more casual is the set of measurements, the more general can be the solution to the recovering process.

Compressed Sensing also introduces an important innovation for what regards signal reconstruction. While the usual Nyquist-Shannon sampling approach is linear, so that signal reconstruction is certain and easy to do, in CS the approach is the exact opposite: reconstruction problems are *non-linear*, and solutions are both uncertain and hard to find. This could seem meaningless, at first, but when put under practice with the correct set of measurements and the correct algorithms, running times result to be significantly better than by using a linear approach.

Signal reconstruction is basisd on the solution of an ℓ^1-norm minimization problem. This requires high computational cost, but, together with an incoherent set of measurements and the hipothesis that the original signal has a sparse representation, it allows exact recovery from a very shallow set of measurements.

Candes, Tao, Donoho and Romberg discoveries regarded the minimum number of samples needed to perform exact signal reconstruction with very good probability. This, together with the latest discoveries in the field of ℓ^1-norm minimization algorithms, allowed CS to be implemented successfully in many fields.

One of the first projects was the One Pixel Camera [10], developed at Rice University [9]. Using CS, a camera which samples one pixel at a time is able to generate a full resolution image, with good definition, by sampling less than 40% of the image pixels. For instance, a 65536 pixel image (256x256 pixel) was obtained by using just 6600 samples, 10% of its resolution, and by recovering the other 90% through ℓ^1-norm minimization. Figure (3.1) shows the recovered image.

Other fields which would benefit of CS approach are those which require a long time to perform sampling. Magnetic Resonance Imaging, for instance, requires up to 30 minutes to scan the subject, depending on the number of

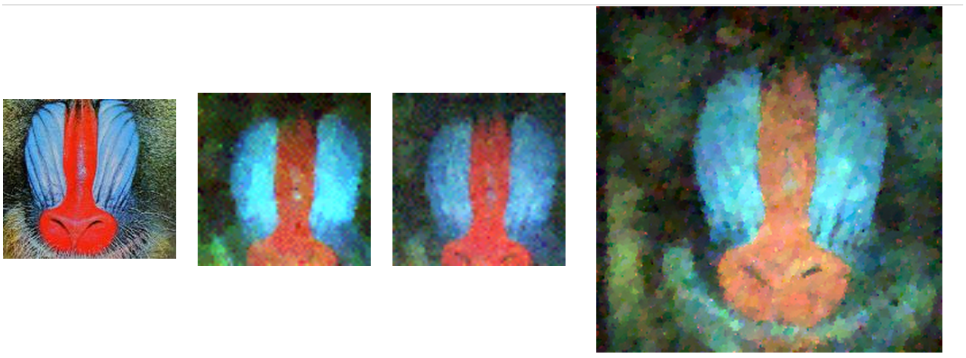

Figure 3.1: Example of CS applied to photography.

body parts to be scanned and the needed resolution. Through CS, though, it would be possible to scan at significant less resolution while keeping image quality inaltered, thus reducing drastically scanning time.

Compressed Sensing is an active field of research, and is day by day been developed further. Rice University keeps an updated list of almost any research and publication regarding CS. For an in-depth analysis of CS, we suggest reading some other publication listed on the online database [12].

Now, we shall try and explain deeply how CS allows to sample less than needed.

3.2 Recovery Approach

Let us suppose that a continuous-time signal $x(t)$, when expressed in a specific basis Ψ, presents coefficients which are either much bigger than zero or near-zero. If we removed those near-zero coefficients, we would obtain a signal $\tilde{x}(t)$ which is the *sparse*, or equivalently the lossy-compressed, representation of $x(t)$.

What we aim to do is sampling *directly* that signal $\tilde{x}(t)$, obtaining the sparse signal $\tilde{x}[n]$. So let us now introduce the specific concept of sparsity.

Definition 3.1 (k-Sparse Signal)**.** A discrete time signal $\tilde{x}[n]$ is said to be k-Sparse if its Support $\{i : x[i] \neq 0\}$ has cardinality less or equal to k. [13]

The first way we can think to obtain the sparse signal is by sampling directly in the Ψ domain. This certainly would allow us to recover the signal $\tilde{x}[n]$. Yet, this approach does not work. First, we would need to know beforehand which is the correct basis to sample in, a feature we do not always possess. Second, this approach is not solid to variations: if the

signal changed, the basis could change as well, and we'd need to readapt to it. But changes aren't immediate, and we could end up sampling in the wrong basis, rendering unuseful all our work.

What CS practically does is looking for a *general* sampling basis, which allows us to obtain *random* components of the signal. This seems meaningless at first, but it is the best approach possible. First, in this way we can develop a general solution which applies to each and every signal we sample, *indipendently on the basis in which they have sparse representation.* Thus our system would be solid to variations, and wouldn't require previous knowledge on the signal, at all.

This approach is known as *Incoherent Sampling.* In fact, the figure of merit on which a basis is chosen is the *Coherence* it has with the basis in which the sampled signal is sparse. Here follows a proper definition of coherence.

Definition 3.2 (Coherence)**.** The coherence between the sensing basis Φ and the representation basis Ψ is the greatest correlation between two of their vectors, i.e.

$$\mu(\Phi, \Psi) = \sqrt{N} \cdot \max_{1 \leq i,j \leq N} |\langle \phi_i, \psi_j \rangle|. \tag{3.1}$$

The less two basis are coherent, the more general will be the samples we obtain. An immediate example is that of spike-basis $\phi_i[n] = \delta[n-k]$ (in time domain) and the normalized Fourier basis of eq.(1.15) (in frequency domain). When we sense with the first base and sample in the second base, we obtain maximum incoherence, i.e. $\mu(\Phi, \Psi) = 1$.

The most stunning results, though, are obtained when we deal with random orthonormal basis. With high probability, the coherence between a random basis Φ and any orthonormal basis Ψ is about $\sqrt{2log n}$ [14]. By extension, random waveforms ϕ_k with independent identically distributed entries, e.g. Gaussian, will also exhibit a very low coherence with *any* fixed representation Ψ, which is exactly what we need.

This has a highly valuable implication: as random basis have low coherence with most basis, then we can always use as sampling base *pure White Noise.* This is stunning, as what we usually aim to remove from most of physical systems is exactly noise.

After selecting the most incoherent base, thus, we need to develop a worthy system to recover our signal. Yet, as what we aim to do is to recover the correct signal from less samples than what is traditionally needed, there are infinite possible reconstructions of the signal. Then, we need to impose specific conditions on the signal we want to retrieve. This was already done, though, when we introduce the concept of sparsity. The recovered signal must be the *sparsest* signal possible.

What would be best to use is ℓ^0-norm minimization, i.e. minimizing a norm which counts the number of non-zero entries in a signal. Unfortunately, solving ℓ^0-norm minimization issues is infeasible, since it is both numerically unstable and NP-complete [11].

Luckily, as we anticipated, it has been demonstrated by David L. Donoho [15] that, for most systems, optimization based on the ℓ_1-norm can exactly recover sparse signals and closely approximate compressible signals with high probability, provided enough samples are accessible.

The optimization problem is stated as follows.

Remark 1 (ℓ^1-Norm Optimization Problem). Suppose we are able to observe only a subset M of length $m < N$ of the a_k coefficients of x when expanded in an orthonormal basis Ψ. The proposed reconstruction $\tilde{x}$ is given by $\tilde{x} = \Psi\tilde{a}$, where Ψ is the representation matrix and $\tilde{a}$ is the solution proposed to the convex optimization

$$\min_{\tilde{a}\in\mathbb{R}^n} ||\tilde{a}||_{\ell_1} \quad \text{subject to} \quad a_k = \langle \psi_k, \Psi\tilde{a}\rangle, \quad \forall k \in M \subset \{1 \dots N\} \tag{3.2}$$

Through this minimization, we are almost sure to recover the correct signal, provided enough samples are available. The inquiry on the minimum number of samples needed to perform exact recovery requires a section of its own.

3.3 Compressed Sampling Theorem

We can also formalize the intuitions about CS we introduced so far by defining a specific sampling theorem.

Let x be a signal with sparse $\tilde{x} = \Psi\tilde{a}$ representation in a certain orthonormal basis Ψ, and let $\{a_k\}$ be its set of coefficients such that $a_k = \langle x, \psi_k\rangle = \langle \Psi\tilde{a}, \psi_k\rangle$. We can recover the $\tilde{x}$ signal as we saw in the previous paragraph. Let us now discuss the the minimum number of samples needed to recover the signal with negligibile error.

Theorem 3.3.1 (Of Exact Signal Recovery)**.** [14] *Fix $x \in \mathbb{R}^n$ and suppose that the coefficient sequence a of x in the basis Ψ is k-sparse. Select m measurements in the Φ domain uniformly at random. Then if*

$$m \geq C \cdot \mu^2(\Phi, \Psi) \cdot k \cdot \log n \tag{3.3}$$

for some small positive constant C, the solution to the optimization problem (3.2) is exact with overwhelming probability. In details, if the constant C is of the form $22(\delta + 1)$ in eq.(3.3), then the probability of success exceeds $1 - O(N^{-\delta})$.

This theorem has some important implications, which emphasize its generality:

1) The role of coherence is clear: the smaller the coherence, the fewer samples are needed, hence CS emphasis on low coherence systems.

2) There is no loss of information even when measuring a set of m coefficients which may be far less than the signal size apparently demands. In case the coherence is close to 1, then $C \cdot k \cdot \log n < n$ samples suffice for exact recovery.

3) The algorithm does not assume any knowledge on the number of nonzero coordinates of a, their location, or their amplitudes, which we assume are all completely unknown a priori. Simply, if the signal happens to be sufficiently sparse, exact recovery occurs.

3.4 Applications

Practical examples of CS cannot be found yet in consumer's products. By the way, its computational approach could one day lead to low-cost equivalents of actual expensive technologies, which would then need just a processor unit to perform what today is made physically.

Let us now give a quick look to some projects which consider CS applications. Most of the projects deal with Imaging, which was the first ideal application of CS even before its theory was totally defined.

3.4.1 Lensless Camera

An immediate and impressive example, on the line of the previously introduced One Pixel Camera [10], is a project which has been developed by the Bell labs in the last months and is called Lensless Camera [16]. It is the latest application of CS to image sampling, and, as of now, promises to grant an improvement over modern image acquisition.

Each and every camera which exists in the modern age is affected by a simple, often overlooked, issue: the lens which allows to obtain images also causes some visible effects of distortion near the margins of the sampled image. Bell's solution would not be affected by this issue: a matrix of pixels acts as a matrix of objectives, which capture light without the need of any lens, and a CS algorithm reconstructs the image from randomized samples. By doing so, first there appear to be no distortions. The light is caught directly, and each sensor allows rough reconstruction of the image, which CS exploits to recover the final image. Second, the sensor definition may simply be increased by joining together matrix units. This is in a way different from modern CMOS technology, which requires the sensors to be implanted onto a chip sensor and requires new printing to increase the sensor resolution.

It must be clear that these features are not yet fully enstablished, as the Lensless Camera is still an experiment. Yet, if developed correctly, they would offer new insights onto image sampling techniques.

3.4.2 Sparse MRI

As stated at the beginning of the chapter, CS finds its natural fit also in MRI [17]. Magnetic Risonance Imaging is a medical imaging technique used in radiology to visualize internal structures of the body in detail. The machinery is composed of a narrow tunnel inside which the patient should lay still. Depending on the level of detail needed, scan times may be quite long, also up to 20-30 minutes, and patients who cannot keep still for all that time or, as example, who suffer from claustrophobia could be unable to undergo a complete scan.

With CS, though, scan times could be lower dramatically, and quality would not degrade at all. There are many types of MRI but most of them satisfy the hypothesis of Compressed Sensing, and thus it may be implemented successfully.

3.4.3 Compressive Radar Imaging

Compressed Sensing would also carry benefits to radar systems [18]. By citing from the cited paper, "CS has the potential to make two signicant improvements to radar systems: (i) eliminating the need for the pulse compression matched filter at the receiver, and (ii) reducing the required receiver analog-to-digital conversion bandwidth. These ideas could enable the design of new, simplied radar systems, shifting the emphasis from expensive receiver hardware to smart signal recovery algorithms".

There are many other possible applications of CS to vast fields of modern technology, but the role of CS in this thesis is introductory to the following chapter, thus we cannot list them all. Every further information can be found on Rice University database [12].

Chapter 4

Nearly Optimal Sparse Fourier Transform

This section is based on the latest instance of sparse Fourier transform [19] developed by a research team of the Massachussetts Institute of Technology, or MIT. The team was composed by four people: Professors Piotr Indyk and Dina Katabi, and PhD Candidates Haitham Hassenieh and Eric Prince.

Just like in Compressed Sensing, suppose that a N-dimensional signal is k-sparse in the *frequency* domain. Then its Fourier transform can be represented succinctly using only k coefficients, and computing an almost exact DFT would be possible in *sub-linearithmic* time. Furthermore, even if the signal is *not* sparse, we can aim to recover the best k-sparse approximation of its Fourier transform to improve over standard DFT running time.

To this account, the team developed two algorithms which are able to *improve* not only over naive DFT algorithms, but also over the Fast Fourier Transform.

The algorithms are designed to handle both cases: the exactly sparse case, and the general case. The first requires the signal to be at most k-sparse, and the algorithm is the fastest; the latter applies to a generic input signal and aims to approximate its sparse representation. This causes the second algorithm to be slightly more complex and to require that a minimum number of samples are available.

For both algorithms the probability of success is not 100%. For the sake of explanation, we shall overlook this at first, but we shall later return on this matter.

4.1 Objectives

Sublinear sparse Fourier analysis is not a brand new concept. This field has witnessed many significant advances in the past two decades, and there exists many algorithms which exploit sparsity to improve over FFT. Yet,

two main issues affect the actual state of art:

1. There exists no algorithm which improves over the FFT's runtime for the *whole* range of sparse signals, i.e. $k = o(N)$.

2. Most algorithms are quite complex, and suffer from large "big-Oh" constants.

The MIT team aimed to design two algorithms which would not suffer from these issues. For what concerns the exactly sparse case, they obtained exactly what they wanted. The key property is the $\mathrm{O}(k \log N)$ complexity, which allows to obtain $o(N \log N)$ complexity for *any* $k = o(N)$. Moreover the algorithm is quite simple, and presents small "big-Oh" constants.

For the general case, which requires a greater degree of approximation, the running time is at most one $\log N/k$ factor away from the oprimal runtime, with $O(k \log N \log(N/k))$, and the "big-Oh" constant is slightly greater than in the exactly sparse case. Moreover, the vector $\widehat{X'}$ computed by the algorithm satisfies the following ℓ_2/ℓ_2 *guarantee*:

$$||\widehat{X} - \widehat{X'}||_2 \leq C \min_{k-sparse\ \widehat{Y}} ||\widehat{X} - \widehat{Y}||_2, \tag{4.1}$$

where C is some approximation factor and the minimization is over k-sparse signals.

Let us see how the algorithms work.

4.2 Basic Understanding

In theory, Sparse Fourier algorithms work by hashing the Fourier coefficients of the input signal into a small number of bins.

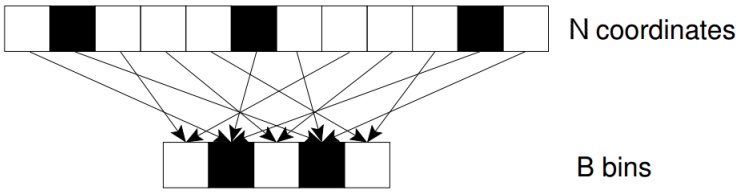

Figure 4.1: Representation of the binning process.

A linear approach would then require that, for each bin, we calculate two values, $\widehat{u_h}$ and $\widehat{u'_h}$, which are respectively the sum and the weighted sum of the Fourier coefficients stored inside the h-th bin

$$\widehat{u_h} = \sum_{n \in \text{h-th bin}} \widehat{X}[N], \quad \widehat{u'_h} = \sum_{n \in \text{h-th bin}} n \cdot \widehat{X}[N]. \tag{4.2}$$

In case in each bin falls just one coefficient, then we can recover the coefficient immediately as $n = \widehat{u'_h}/\widehat{u_h}$. With the hypothesis that the signal is sparse in the frequency domain, and by using a randomized hash function, there is a high probability that each bin contains just one significant coefficient. Thus, $\widehat{u_h}$ would be a good estimation of that coefficient value.

To estimate the index of the coefficient, though, it is not possible to use the weighted sum, as this would require the knowledge of all the n Fourier coefficients. Moreover, there are more than one coefficient per bin, thus the index approximation must be the best possible. To obtain this, an alternative approach is used, which could though return a wrong index. So, the process must be repeated to ensure that each coefficient is correctly identified.

Usually, each iteration the number of coefficients which are not identified correctly is a constant number k'. For the exactly sparse case, $k' = k/2$, while for the general case $k = k/4$. If we remove those $k - k'$ coefficients from the signal, then for the following iteration the signal can be considered k'-sparse. This allows to reduce both the window size and the number of bins, i.e. B, and thus to reduce the total time complexity. We see that the identified coefficients can be removed from the bins rather than from the signal, as the latter is a costly operation.

By correctly recovering all the k coefficients, we obtain a correct Fourier transform of the signal in case it is at most k-sparse in frequency domain, or the best k-sparse approximation in case the signal is not sparse at all.

This is the idea behind how the algorithm works. In practice, though, each operation must meet certain requirements:

1. Storing needs to be done in sublinear time, thus it is performed through a N-dimensional filter vector with particular properties, as later shown.

2. Location and estimation methods need to be as fast as possible, both in the exactly sparse and in the general case.

3. As the algorithm has been implemented iteratively, a fast method for updating the signal is required, so that identified coefficients are not considered in future iterations.

The algorithms were implemented by adapting previously designed techniques and by designing brand new ones, thus we will now introduce them in detail.

4.3 Flat Filtering Windows

The filter used for the hashing is a N-dimensional shifted filter vector G, whose Fourier transform we will call $\widehat{G}$. The filter needs to be concentrated

both in time and in frequency. That is, G must be zero except for a small number of coefficients and also $\widehat{G}$ must be negligible except at a small *fraction* ($\sim 1/k$) of the frequency coordinates, corresponding to the filter's pass band region.

The filter permits to analyse just a fraction of the whole signal at a time. In this way, instead of computing the N-dimensional DFT, we can compute the B-dimensional DFT of this section, where B is the support of the filter, in $O(B \log B)$ time. These B Fourier coefficients result to be a subsampling of the of the N-dimensional DFT applied to the first B therms of the signal, and we can hash these values correctly inside each bin. This is kind of analogous to summing each Fourier coefficient inside the bin, with the difference that we are also adding some noise due to the subsampling. The noise added results to be negligible if the signal is sparse, while it must be taken into account in the general case.

To build a filter which satisfies the first constraint, a *box-car* function would be enough, i.e. an ideal square wave with infinite period. Although, Fourier transform of a box-car is nothing but a sync, which decays to zero at infinity, thus causing what we could call *leakage* in frequency. That is, the spectrum that would go inside a bin may 'leak' in the nearby bins.

What we would like to happen is that our filter behaved as a box-car function *in frequency*, so that there would not be any leakage. The best solution in this direction is to use a Gaussian function.

Lemma 4.3.1. *Let G be a gaussian with $\sigma = B\sqrt{\log N}$, H a box-car filter of length N/B, and F the filter we want to design. Then, $\widehat{F} = \widehat{G} * H$ is a filter which has (pass) band N/B and support $O(B \log N)$. Moreover, leakage can happen at most in one other bin. Figure (4.3.1) shows the time and frequency graphs of this filter.*

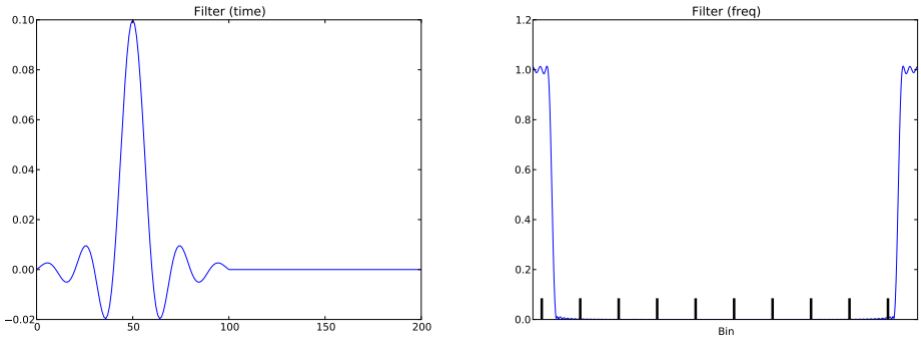

Figure 4.2: Time and frequency plots of the $\widehat{F}$ filter.

The filters used in the algorithm are called Flat Filtering Windows, and were designed together with an earlier version of the sparse Fourier transform. A Flat Filtering Window is a couple $(G, \widehat{G'})$ where both G and $\widehat{G'}$ are

like the previously defined filters, and in addition $\widehat{G'}$ satisfy certain properties [19]. Further insight can be found in the published article [20].

A particularity of these filters is that they present a special region within their pass band, called *super-pass* region, in which $\widehat{G}$ is relatively larger compared to the values assumed in the rest of the pass band. This region is shown in Figure(4.3). We say that a coefficient is *isolated* if it falls into a filter's super-pass region and no other coefficient falls into filters pass region [19]. As the super-pass region is a constant fraction of the pass region, the probability of isolating a coefficient is constant. When a coefficient becomes isolated, then it can be estimated and located.

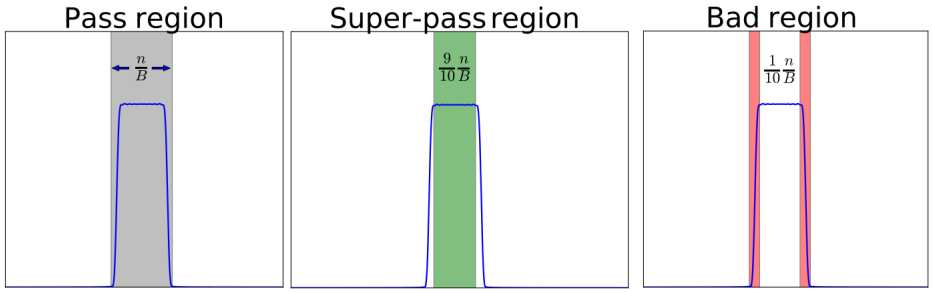

Figure 4.3: Frequency plot of a Flat Filtering Window, with visual subdivision in regions.

4.4 Location and Estimation

We briefly spoke of the estimation method in Section 4.2, and here we just need to formalize it. As there likely falls just one "big" coefficient inside each bin, the value contained inside the bin can be considered a good approximation of the coefficient value. Thus, estimation is straight forward in both cases. The location method, instead, depends on whether the exactly sparse case or the general case is handled.

Before diving into the details, a general observation requires to be done for both of the cases. Note that the value of the identified coefficient is supposed to be the most significant within the bin, some magnitudes larger than the rest of the coefficients. This, though, is true only if the coefficient fell inside the super-pass region of the filter. Otherwise, if the coefficient falls into the bad region of the filter, its value could be softened up to the point of not being relevant anymore, and thus the estimation would be wrong. Moreover, if the coefficient was also *isolated* during filtering, then we can be sure that the index estimation is correct, for none other coefficient falls in the N/B window. Otherwise, index estimation could be flawed by the noise

caused by the other coefficient. Thus a necessary condition to locate and estimate a coefficient, is that it is *isolated*.

4.4.1 Exactly Sparse Case

In the exactly sparse case, location is based on the difference in phase between two samples of the signal, sampled at a time unit of difference. The bins are represented by a vector $\widehat{u}$ of length $B = O(k)$. Another vector, $\widehat{u'}$ is used to store the samples after the time shift.

For each step of the algorithm $\widehat{u}$ and $\widehat{u'}$ are computed by hashing and summing the Fourier coefficients of the input signal. The coefficients stored inside $\widehat{u'}$ differ from the original coefficients by an $e^{-2\pi\frac{n}{N}}$ factor, due to DTFT property of time-shifting (see eq.(1.25)) inherited by DFT. Thus we can obtain the index n of the h-th bin's largest coefficient with good fidelity, by dividing $\widehat{u_h}/\widehat{u'_h}$. This approach was inspired by the one used to estimate frequency offset in orthogonal frequency division multiplexing, a telecomunications coding technique (OFDM) [21].

4.4.2 General Case

For what regards the general case, preliminary experiments showed [19] that the two-samples approach works surprisingly well if more than two samples are used. Still, in theory this should not be possible. When time shifting a signal which is not sparse, the noise caused by the other frequencies cannot be ignored anymore, and thus the phase difference cannot be considered linear anymore in the index of the coefficients. In fact, it results to be $2\pi\frac{n}{N} + \nu$, where ν is a random factor due to the noise [19].

However, the disturbance caused by the noise must remain limited within certain values, that is $E[\nu^2]$ is limited. Then, we need to search for the correct value of n among the N/k elements inside the bucket, and this in $O(\log N/k)$ time.

A first solution is to perform measurements by time shifting the filter randomly, using $\beta \in [N]$. To select the best n candidate, a minimum distance criterion based on the phase is used. $O(\log N/k)$ random measurements would then suffice to distinguish among the N/k coefficients, because each measurement would likely be within $\pi/4$ from its corresponding real value and probably further apart from any other real value. Yet, by doing so, the choice would be done in linear time complexity, as N/k checks should be done.

Instead, the algorithm performs a t-ary search on the location for $t = O(\log N)$. At each $O(\log_t(N/k))$ level, the region is split into t consecutive subregions of size w. The time shift is different this time, and allows the decoding to be done in $O(t \log t)$, with probability of failure t^2. By repeating $log_t(N/k)$ times, we can find n precisely. The total number of measurements

is then $O(\log t \log_t(N/k)) = O(\log(N/k))$, and the decodification (location) time is $O(\log(N/k)\log(n))$.

4.5 Updating the signal

We noted that the aforementioned techniques locate a coefficient correctly only if it has been isolated. During each filtering step, each coefficient becomes isolated only with constant probability. Therefore the filtering process needs to be repeated to ensure that each coefficient is correctly identified.

Repeating the filtering right away $O(\log N)$ times would not be the best idea, as it would cause a total running time of $O(k \log^2 N)$. Instead, we can reduce the filtering time by removing the identified coefficients. Subtracting them from the signal is a computationally costly operation, thus the coefficients are removed from the *bins* obtained by filtering the signal. This can be done in time linear in the number of subtracted coefficient.

For the exactly sparse case, the cost of filtering the signal is then $O(B \log N)$, where B is the number of bins. b is set to $O(k')$, where k' is the number of yet-unidentified coefficients. Thus, initially B is equal to $O(n)$, but its value decreases by a constant factor each iteration. In total, removing the coefficients requires $O(k)$ time.

4.6 Total Time Complexity

We can now show the total time of the algorithms.

For the exactly sparse case:

- Index location and estimation is immediate.
- The removal of the coefficients from the bin takes $O(k)$ time in total.
- Hashing is each time done in $O(B \log N)$ time, where B halves each iteration.
- The first iteration dominates over the others, with $B = k$.

Thus the total time complexity results to be $O(k \log N + k) = O(k \log N)$.

For the general case:

- To identify the coefficients $O(\log(N/B))$ measurements are needed.
- Coefficient value estimation is immediate.
- The removal of the coefficients from the bin takes $O(k)$ time in total.

- Hashing is then done in $O(B \log N + k)$ time, where B halves each iteration.
- The first iteration still dominates over the others, with $B = k$.

Thus the total time complexity results to be $O((B \log N + k)log(N/B)) = O(k \log N \log(N/k))$.

4.7 The Role of Chance

Now that we described the general behaviour of the algorithms, it is time to resume that matter which we have overlooked up to now: the algorithms' probability of success.

We must note that the effective time complexities are asymptotically the same as the ones we showed in the previous section, but the therms on which they are evaluated are different. This is due to the fact that most events, starting from the hashing, have to deal with random probability, which is taken into account inside the algorithms.

We now show both the algorithms and make some considerations on the role that chanche has in the recovery of the correct sparse signals.

4.7.1 Algorithm for the Exactly Sparse Case

The pseudocode for the exactly sparse case algorithm is shown as Algorithm 4.8.1.

Let us analyse NoiselessSparseFFTInner. Each iteration the window size is set at $B = k'/\beta$ for a small β constant; then a variable σ is chosen randomly among the odds numbers in $[N]$, and another variable b is chosen randomly as well. These are the parameters for the randomized hash function. Defined $\pi_{\sigma,b}(n) = \sigma(n - b) \bmod N$, the bin in which each coefficient is mapped to is $h_{\sigma,b}(n) = \text{round}(\pi_{\sigma,b}(n)B/N)$. Set $S = \text{supp}(\widehat{X} - \widehat{Z})$.

On the probability space induced by σ, we define two events which can happen:

- "Collision" event $E_{coll}(n)$ holds iff there is leakage to another bin due to the non ideal flat window function, i.e. iff $h_{\sigma,b}(n) \in h_{\sigma,b}(m)$ with $m \in S, m \neq n$.
- "Large Offset" event $E_{off}(n)$ holds iff the coefficient falls in a bad region (i.e. if the offset from the center frequency of the bin is too large).

It can be proved [19] that:

- $Pr[E_{coll}(n)] \leq 4|S|/B$, where $|S|$ is the number of not-yet identified coefficients.

- $Pr[E_{off}(n)] \leq \alpha$, where $\alpha]$ is the parameter which defines the bad region size in the flat filtering window $G_{B,\alpha,\delta}$.

In case neither of these cases happen, then estimation and location are performed correctly.

The procedure HashToBins cares about the hashing of the coefficients, and returns the vector of bins $\widehat{u}$. The values stored depend from the flat window function $(G, \widehat{G'})$, which are defined at each iteration by the values B, α and δ. δ is a constant which must be chosen to grant good performances. In the case, $\delta = 1/(16n^2L)$, where L is a value such that $\widehat{X}[N] \in \{-L \cdot\}$.

The variable $\widehat{Z}$ stores in every instant the computed sparse Fourier transform of x. Set $\zeta = \{n \in supp(\widehat{Z}) : E_{off}(n) \text{ holds}\}$, then the running time of HashToBins results to be $O(\frac{B}{\alpha}\log(1/\delta) + |supp(\widehat{Z})| + \zeta\log(1/\delta))$.

The total running time and precision of the algorithm are defined from the main loop NoiselessSparseFFT.

Define $S_t = \text{supp}(\widehat{X} - \widehat{Z})$. Each iteration is considered *successful* iff $|S_t|/|S_{t-1}| \leq 1/8$, i.e. if the number of coefficients identified is below a certain threshold. Then, if we consider the events $E(t)$ that occur iff the number of precedent successful iterations is greater than $t/2$, we can prove that $\sum_t Pr[E(t)] \leq 1/3$. Thus

- the probability that the all the coefficients are identified correctly is greater than 2/3.

The running time of NoiselessSparseFFT is dominated by $O(\log N)$ executions of HashToBins. Since large offset events happen in median $\zeta = \alpha|\text{supp}(\widehat{Z})|$ times and $1/\delta \propto N^2$, the expected running time of each execution of HashToBins is $O(\frac{B}{\alpha}\log N + k + \alpha k\log(1/\delta)) = O(\frac{B}{\alpha}\log N + k + \alpha k \log N)$. Setting $\alpha = \Theta(2^{-n/2})$ and $\beta = \Theta(1)$, as in the algorithm, the expected running time round n becomes $O(2^{-n/2}k\log N + k + 2^{-n/2}k\log N)$. Therefore,

- the total expected running time for NoiselessSparseFFT is $O(k\log N)$.

4.7.2 Algorithm for the General Case

The algorithm for the General case is divided into Algorithms 4.8.2 and 4.8.3. This case is not too different from the previous, yet it is more complex.

Let's start from the SparseFFT procedure. As in the noiseless case, we define $\pi_{\sigma,b}(n) = \sigma(n-b) \bmod N$ and $h_{\sigma,b}(n) = \text{round}(\pi_{\sigma,b}(n)B/N)$. We say that $h_{\sigma,b}(n)$ is the bin that frequency n is mapped into. Moreover, we define $h^{-1}_{\sigma,b}(m) = \{m \in [N] | h_{\sigma,b}(n) = m\}$.

Define the approximation error on x as

$$\mathrm{Err}(x,k) = \min_{\text{k-sparse y}} ||x-y||_2. \tag{4.3}$$

In each iteration of SparseFFT, define $\widehat{X'} = \widehat{X} - \widehat{Z}$, and let $S = \{n \in [N] : |\widehat{X'}|^2 \geq \epsilon\rho^2/k\}$, where ϵ is a small number and ρ is a threshold which takes in consideration the approximation error on x and some other parameters.

It can be shown that, with parameters ϵ, $R = O(\log k/\log\log k)$ and $B = \Omega(\frac{Ck}{\alpha^2\epsilon})$, for C larger than some fixed constant,

- SparseFFT recovers $\widehat{Z}^{(R+1)}$ with

$$||\widehat{X} - \widehat{Z}^{(R+1)}||_2 \leq (1+\epsilon)\mathrm{Err}(\widehat{X},k) + \delta||\widehat{X}||_2. \tag{4.4}$$

  in $O(\frac{k}{\epsilon}\log(N/k)\log(N/\delta))$ time.

- The probability that SparseFFT computes the correct values with the above precision is at least 3/4.

4.7.3 Lower Bounds

As we said in the introduction, a minimum number of the signal samples must be available in order for the algorithm to work correctly.

If one assumes that the FFT is optimal and hence the DFT cannot be solved with less than $\mathrm{O}(N\log N)$ samples, this algorithm results optimal as long as $k = N^{\Omega(1)}$. Thus it requires a minimum of $\Omega(k\log k)$ samples in order to have a good approximation.

For the general case, the result the team obtained can be translated into the language of Compressed Sensing:

Theorem 4.7.1. *Let $F \in \mathbb{C}^{N\times N}$ be orthonormal and satisfy $|F_{i,j}| = 1/\sqrt{N}$ for all i,j. Suppose an algorithm takes m adaptive samples of Fx and computes z with*

$$||x-z||_2 \leq 2 \min_{k\text{-sparse } y} ||x-y||_2 \tag{4.5}$$

for any x, with probability at least 3/4. Then $m = \Omega(k\log(N/k)/\log\log N)$.

Corollary 4.7.2. *Any algorithm computing the Fourier transform must access $\Omega(k\log(N/k)\log\log N)$ samples from the time domain.*

Note that, if the samples were chosen adaptively, then $m = \Omega(k\log(N/k))$. As we pointed out in Chapter 3, though, adaptive sampling is not always the right choice, and algorithm versatility would on contrary suffer from this choice.

4.8 Testing the Exactly Sparse Case

We now show some tests done with the discussed algorithm for the exactly sparse case. The implementation source files can be found on the main site [23] under the section *Code*. The algorithm were written in C++ language and tested under Ubuntu 10.04 and 11.10 OS versions. All the relative documentation con be found in the corresponding paper [24].

The code implements a testing environment, in which the SFFT algorithm is compared to the FFTW of the same input. The structure of the benchmark is the following:

```
INPUT: ./experiment -N (N dimension) -K (K dimension) (input)
OUTPUT:
RUNNING EXPERIMENT: n= (N dimension), k= (K dimension)
Simulation:
********************************************************************************
Projected error rate:
Expected running time:
********************************************************************************
sFFT filter parameters for: n= , k= .
********************************************************************************
Comb Filter: (parameters)
Location Filter: (parameters)
Estimation Filter: (parameters)
Window size:         Location Filter                  Estimation Filter
Noise in filter:         Location Filter                 Estimation Filter
********************************************************************************
sFFT Results
********************************************************************************
Total sFFT time:

ERROR:
K= ; MISSED (estimation, result) =                          L1 ERROR=
********************************************************************************
FFTW Results
********************************************************************************
Time to create FFTW plan:
Time to run FFTW :
********************************************************************************
```

We shall display the benchmark for a positive case, in which SFFT bests FFTW, and for a negative case, in which SFFT takes longer than FFTW routines.

4.8.1 Positive Case

```
18. SFFT 1.0 ( N = 4194304, K = 50, SNR = 0dB)

INPUT: ./experiment -K 50 -B 4 -E 0.2 -L 15 -l 3 -r 2 -t 1e-6 -e 1e-8 -S 1
OUPUT:
RUNNING EXPERIMENT: n=4194304, k=50.
```

```
Simulation:
********************************************************************************
Projected error rate: 0.00103322 (5.11769e-06 per large frequency)
Expected running time: 0.0349159
********************************************************************************
SNR = 1.00033 / 0.00 dB
sFFT filter parameters for: n=4194304, k=50.
********************************************************************************
Comb Filter: none
Location Filter: (numlobes=12349.0, tol=1e-06, b=448) B: 100/8192 loops: 2/3
Estimation Filter: (numlobes=617.0, tol=1e-08, b=10468) B: 512 loops: 15
Window size: Location Filter : 114061; Estimation Filter : 7507;
Noise in filter: Location Filter : 6.09389e-08; Estimation Filter 1.70111e-09
********************************************************************************
sFFT Results
********************************************************************************
Total sFFT time: 0.030106
Time distribution:
scoretable Comb perm+?lter grouping estimation stepB+C other total
0.005050 0.000000 0.015866 0.002781 0.004032 0.001417 0000960 0.030106
16.8% 0.0% 52.7% 9.2% 13.4% 4.7% 3.2% 100.0%
ERROR:
K=50; MISSED (estimation, result) = (0, 0); L1 ERROR= 3.41686
(0.0683372 per large frequency)
********************************************************************************
FFTW Results
********************************************************************************
Time to create FFTW plan: 0.000276
Time to run FFTW : 0.296326
********************************************************************************
```

4.8.2 Negative Case

```
1. SFFT 2.0 ( N = 65536, K = 50)
INPUT: ./experiment -N 65536 -K 50 -B 4 -E 2 -M 128 -m 6 -L 10 -l 4 -r 2
-t 1e-8 -e 1e-8
OUTPUT:
RUNNING EXPERIMENT: n=65536, k=50
Simulation:
********************************************************************************
Projected error rate: 0.000181988 (0.000158854 per large frequency)
Expected running time: 0.0125901
********************************************************************************
sFFT filter parameters for: n=65536, k=50.
********************************************************************************
Comb Filter: loops: 6 mod: 100/8192
Location Filter: (numlobes=1810.0, tol=1e-08, b=47) B: 100/1024 loops: 2/4
Estimation Filter: (numlobes=905.0, tol=1e-08, b=111) B: 512 loops: 10
Window size: Location Filter : 22023; Estimation Filter : 11011;
Noise in filter: Location Filter : 6.22353e-10; Estimation Filter 1.60315e-09
********************************************************************************
sFFT Results
********************************************************************************
```

```
Total sFFT time: 0.005218Time distribution:
scoretable Comb perm+filter grouping estimation stepB+C other total
0.000011 0.002111 0.002404 0.000084 0.000203 0.000376 0.000028 0.005218
0.2% 40.5% 46.1% 1.6% 3.9% 7.2% 0.5% 100.0%
ERROR:
K=50; MISSED (estimation, result) = (0, 0); L1 ERROR= 8.84975e-07
(1.76995e-08 per large frequency)
********************************************************************************
FFTW Results
********************************************************************************
Time to create FFTW plan: 0.000397
Time to run FFTW : 0.001631
********************************************************************************
```

4.8.3 Result analysis

The tests on the exactly sparse case have shown that SFFT improves over FFTW for values of $k/N < 3\%$. For values greater than the 3% of N, the algorithms cannot improve over FFTW. Further optimizations would probably allow to attain this feature, but for the time being the algorithms cannot beat the FFTW for *all* k.

It is to note, though, that the SFFT runtime depends almost exclusively on k. In fact, as we see in the first graph of Figure 4.8.3, fixing k and increasing the size N of the signal leaves the SFFT running time almost unchanged, while the FFTW running time grows linearly depending on N. This was to be expected, obviously, as the time complexities of the two algorithms are $O(k \log N)$ for the SFFT and $O(N \log N)$ for the FFTW.

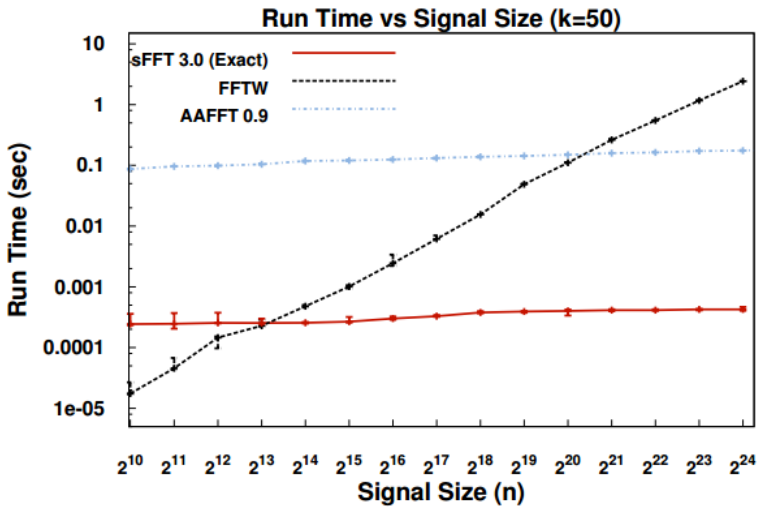

Figure 4.4: C++ implementation of SFFT algorithm compared to FFTW, Run Time vs Signal Size.

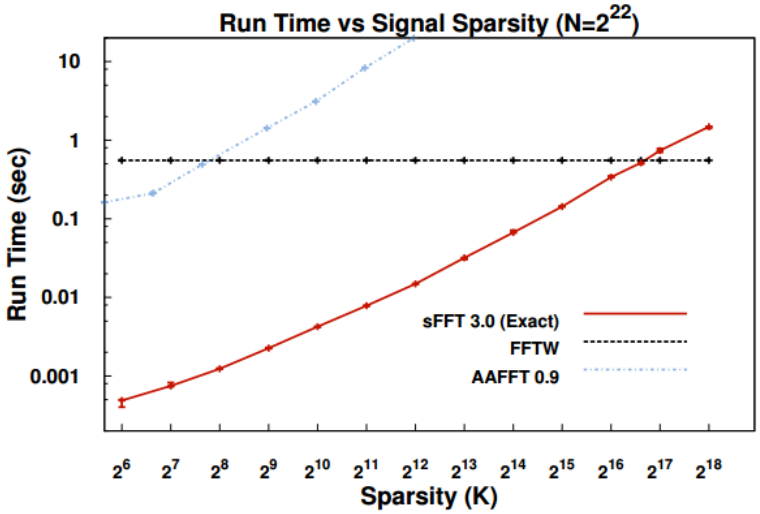

Figure 4.5: C++ implementation of SFFT algorithm compared to FFTW, Run Time vs Signal Sparsity.

procedure HashToBins(x, $\widehat{Z}$, $P_{\sigma,a,b}$, B, δ, α)
    Compute $\widehat{y}_{jN/B}$ for $j \in [B]$, where $y = G_{B,\alpha,\delta} \cdot (P_{\sigma,a,b}x)$
    Compute $\widehat{y'}_{jN/B} = \widehat{y}_{jN/B} - (\widehat{G'_{B,\alpha,\delta}} * \widehat{P_{\sigma,a,b}z})_{jN/B}$ for $j \in [B]$
    return $\widehat{u}$ given by $\widehat{u}_h = \widehat{y'}_{jN/B}$.
end procedure
procedure NoiselessSparseFFTInner(x, k', $\widehat{Z}$, α)
    Let $B = k'/\beta$, for sufficiently small constant β.
    Let $\delta = 1/(4n^2L)$.
    Choose σ uniformly at random from the set of odd numbers in $[N]$.
    Choose b uniformly at random from $[N]$.
    $\widehat{u} \leftarrow$ HashToBins$(x, \widehat{Z}, P_{\sigma,0,b}, B, \delta, \alpha)$.
    $\widehat{u}' \leftarrow$ HashToBins$(x, \widehat{Z}, P_{\sigma,1,b}, B, \delta, \alpha)$.
    $\widehat{w} \leftarrow 0$.
    Compute $J = \{j : |\widehat{u}_h| > 1/2\}$.
    for $j \in J$ **do**
        $a \leftarrow \widehat{u}_j/\widehat{u}'_j$.
        $i \leftarrow \sigma^{-1}(\text{round}(\phi(a)\frac{n}{2\pi})) \bmod n$. ▷ $\phi(a)$ denotes the phase of a.
        $v \leftarrow \text{round}(\widehat{u}_j)$.
        $\widehat{w}_i \leftarrow v$.
    end for
    return $\widehat{w}$
end procedure
procedure NoiselessSparseFFT(x, k)
    $\widehat{Z} \leftarrow 0$
    for $t \in 0, 1, \ldots, \log k$ **do**
        $k_t \leftarrow k/2^t$, $\alpha_t \leftarrow \Theta(2^{-t})$.
        $\widehat{Z} \leftarrow \widehat{Z} +$ NoiselessSparseFFTInner$(x, k_t, \widehat{Z}, \alpha_t)$.
    end for
    return $\widehat{Z}$
end procedure

Algorithm 4.8.1: Exact k-sparse recovery

procedure SPARSEFFT(x, k, ϵ, δ)
  $R \leftarrow O(\log k / \log\log k)$
  $\widehat{Z}^{(1)} \leftarrow 0$
  for $r \in [R]$ **do**
    Choose B_r, k_r, α_r
    $R_{est} \leftarrow O(\log(\frac{B_r}{\alpha_r k_r}))$
    $L_r \leftarrow$ LOCATESIGNAL$(x, \widehat{Z}^{(r)}, B_r, \alpha_r, \delta)$
    $\widehat{Z}^{(r+1)} \leftarrow \widehat{Z}^{(r)} +$ ESTIMATEVALUES$(x, \widehat{Z}^{(r)}, 3k_r, L_r, B_r, \delta, R_{est})$.
  end for
  return $\widehat{Z}^{(R+1)}$
end procedure
procedure ESTIMATEVALUES(x, $\widehat{Z}$, k', L, B, δ, R_{est})
  for $r \in [R_{est}]$ **do**
    Choose $a_r, b_r \in [N]$ uniformly at random.
    Choose σ_r uniformly at random from the set of odd numbers in $[N]$.
    $\widehat{u}^{(r)} \leftarrow$ HASHTOBINS$(x, \widehat{Z}, P_{\sigma,a_r,b}, B, \delta)$.
  end for
  $\widehat{w} \leftarrow 0$
  for $i \in L$ **do**
    $\widehat{w}_i \leftarrow \text{median}_r\, \widehat{u}^{(r)}_{h_{\sigma,b}(i)} \omega^{-a_r \sigma i}$. ▷ Separate median in real and imaginary axes.
  end for
  $J \leftarrow \arg\max_{|J|=k'} \|\widehat{w}_J\|_2$.
  return $\widehat{w}_J$
end procedure

Algorithm 4.8.2: k-sparse recovery for general signals, part 1/2.

procedure LocateSignal(x, $\widehat{Z}$, B, α, δ)
  Choose uniformly at random $\sigma, b \in [N]$ with σ odd.
  Initialize $l_i^{(1)} = (i-1)N/B$ for $i \in [B]$.
  Let $w_0 = N/B, t = \log N, t' = t/4, D_{max} = \log_{t'}(w_0 + 1)$.
  Let $R_{loc} = \Theta(\log_{1/\alpha}(t/\alpha))$ per Lemma **??**.
  for $D \in [D_{max}]$ **do**
    $l^{(D+1)} \leftarrow$ LocateInner$(x, \widehat{Z}, B, \delta, \alpha, \sigma, \beta, l^{(D)}, w_0/(t')^{D-1}, t, R_{loc})$
  end for
  $L \leftarrow \{\pi^{-1}_{\sigma,b}(l_j^{(D_{max}+1)}) \mid j \in [B]\}$
  return L
end procedure

▷ δ, α parameters for G, G'
▷ $(l_1, l_1 + w), \ldots, (l_B, l_B + w)$ the plausible regions.
▷ $B \approx k/\epsilon$ the number of bins
▷ $t \approx \log N$ the number of regions to split into.
▷ $R_{loc} \approx \log t = \log\log N$ the number of rounds to run

procedure LocateInner(x, $\widehat{Z}$, B, δ, α, σ, b, l, w, t, R_{loc})
  Let $s = \Theta(\alpha^{1/3})$.
  Let $v_{j,q} = 0$ for $(j, q) \in [B] \times [t]$.
  for $r \in [R_{loc}]$ **do**
    Choose $a \in [N]$ uniformly at random.
    Choose $\beta \in \{\frac{snt}{4w}, \ldots, \frac{snt}{2w}\}$ uniformly at random.
    $\widehat{u} \leftarrow$ HashToBins$(x, \widehat{Z}, P_{\sigma,a,b}, B, \delta, \alpha)$.
    $\widehat{u}' \leftarrow$ HashToBins$(x, \widehat{Z}, P_{\sigma,a+\beta,b}, B, \delta, \alpha)$.
    for $j \in [B]$ **do**
      $c_j \leftarrow \phi(\widehat{u}_h / \widehat{u}'_j)$
      for $q \in [t]$ **do**
        $m_{j,q} \leftarrow l_j + \frac{q-1/2}{t} w$
        $\theta_{j,q} \leftarrow \frac{2\pi(m_{j,q}+\sigma b)}{n} \bmod 2\pi$
        if $\min(|\beta\theta_{j,q} - c_j|, 2\pi - |\beta\theta_{j,q} - c_j|) < s\pi$ **then**
          $v_{j,q} \leftarrow v_{j,q} + 1$
        end if
      end for
    end for
  end for
  for $j \in [B]$ **do**
    $Q^* \leftarrow \{q \in [t] \mid v_{j,q} > R_{loc}/2\}$
    if $Q^* \neq \emptyset$ **then**
      $l'_j \leftarrow \min_{q \in Q^*} l_j + \frac{q-1}{t} w$
    else
      $l'_j \leftarrow \perp$
    end if
  end for
  return l'
end procedure

Algorithm 4.8.3: k-sparse recovery for general signals, part 2/2.

Conclusions

In this thesis, we introduced a brand new algorithm, known as Sparse Fast Fourier Transform or SFFT, which aims to improve over the actual state-of-the-art algorithm for Fourier analysis, the FFT.

We have seen that the algorithm works best when the signal is exactly sparse, but also that it effectively bests FFT only if the signal is sparse *enough*. This issue is minor, as many signals have high degrees of sparsity. However, the algorithm also supposes that the signal's length is a power of two, which could be instead a serious limitation.

The general case solution is still a bit rough, and there remain some open problems which we haven't yet dealt with. First, it is not known whether the runtime could eventually be reduced to that of the exact sparse case. Second, neither is known whether the precision of the algorithm could be increased further. Third, if we were able to have an ℓ^∞/ℓ^2 guarantee instead of the present ℓ^2/ℓ^2 one, the algorithm would be significantly more solid.

These points altogether indicate that the SFFT has a lot to grow. As a matter of fact, since its release, three versions of the algorithm have been published, and each time there were several improvements. However, it is undeniable that sparsity could be the key we were looking for to obtain extremely fast and efficient data acquisition and analysis algorithms. Many others have tried this road, and SFFT is just the latest - and probably best - representative. We are sure that the whole sector will develop further and further in the following years, also on the wave of CS incredible growth. It is not to exclude that, exactly from this sector, there will eventually rise the next state-of-the-art.

Bibliography

[1] Cristina Ronconi, *"Appunti di Geometria"*, Univer Editrice, Dicembre, 2002.

[2] Alan V. Oppenheim, Alan S. Willsky, and S. Hamid Nawab, *"Signals & Systems"*, Second Edition, Prentice Hall International, Boston, Massachussetts, August 16, 1996.

[3] James S. Walker, *"Fast Fourier transforms"*, CRC press, 1996.

[4] Michael T. Heideman, Don H. Johnson, C. Sidney Burrus, *"Gauss and the History of the Fast Fourier Transform"*, IEEE ASSP Magazine, October, 1984.

[5] Matteo Frigo and Steven G. Johnson, *The Design and Implementation of FFTW3"*, Proceedings of the IEEE 93, pp. 216-231, 2005.

[6] Matteo Frigo and Steven G. Johnson, *"benchFFT*, *www.fftw.org/benchfft/*, 2005.

[7] Intel®Labs, *http://software.intel.com/en-us/intel-ipp*, 2013.

[8] Vlodymyr Myrnyy, *"A Simple and Efficient FFT Implementation in C++"*, *DrDobbs.com*, Dr. Dobb's, May 10, 2007.

[9] Rice University Homepage, *www.rice.edu*, Rice University, Houston, Texas, 2013.

[10] Marco F. Duarte, Mark A. Davenport, Dharmpal Takhar, Jason N. Laska, Ting Sun, Kevin F. Kelly, and Richard G. Baraniuk, *Single Pixel Imaging via Compressive Sampling*, IEEE Signal Processing Magazine, March 2008.

[11] Dongdong Ge, Xiaoje Jiang, and Yinyu Ye, *"A Note on the Complexity of Lp Minimization"*, Stanford University, Stanford, California, September 3, 2009.

[12] Compressed Sensing Resources, *dsp.rice.edu/cs*, Rice University, Houston, Texas, 2013.

[13] Emmanuel J. Candes, *"Compressive Sampling"*, Int. Congress of Mathematics, 3, pp. 1433-1452, Madrid, Spain, 2006.

[14] Emmanuel J. Candes, Michael B. Wakin *"An Introduction To Compressive Sampling"* IEEE Signal Processing Magazine, pp. 21-30, March, 2008.

[15] David L. Donoho, *"For Most Large Underdetermined Systems of Linear Equations the Minimal ℓ^1-norm Solution is also the Sparsest Solution"*, Stanford University, Stanford, California, September 16, 2004.

[16] Gang Huang, Hong Jiang, Kim Matthews, and Paul Wilford, *"Lensless Imaging by Compressive Sensing"*, Bell Labs, Alcatel-Lucent, Murray Hill, New Jersey, USA, May 2013.

[17] Michael Lustig, David Donoho, and John M. Pauly, *"Sparse MRI: The application of compressed sensing for rapid MR imaging"*, Magnetic Resonance in Medicine, pp. 1182 - 1195, December 2007.

[18] Richard Baraniuk, Philippe Steeghs, *"Compressive radar imaging"*, IEEE Radar Conference, Waltham, Massachusetts, April 2007.

[19] Haitham Hassanieh, Piotr Indyk, Dina Katabi, and Eric Price, *"Nearly Optimal Sparse Fourier Transform"*, STOC'12, ACM Symposium on Theory of Computing, New York, USA, May 2012.

[20] Haitham Hassanieh, Piotr Indyk, Dina Katabi, and Eric Price, *"Simple and practical algorithm for sparse Fourier transform"*, SODA, 2012.

[21] Juha Heiskala, Juhn Terry, *"OFDM Wireless LANs: A Theoretical and Practical Guide"*, Sams, Indianapolis, Indiana, USA, 2001.

[22] Anna C. Gilbert, Yi Li, Ely Porat, and Martin J. Strauss, *"Approximate sparse recovery: optimizing time and measurements"*, STOC, pp. 475-484, 2010.

[23] MIT Staff, *"SFFT: Sparse Fast Fourier Transform"*, *groups.csail.mit.edu/netmit/sFFT/index.html*, 2012.

[24] Haitham Hassanieh, Piotr Indyk, Dina Katabi, and Eric Price, *"Sparse Fast Fourier Transform Code Documentation (SFFT 1.0 and 2.0)"*, STOC'12, ACM Symposium on Theory of Computing, New York, USA, May 2012.