1 Introduction

In the field of digital signal processing, one speaks about different signal domains. The two important domains for this paper is the time domain, and the frequency domain.

The key idea of these domains is that the same signal can be visualized and interpreted in different ways. As an example, let us consider the following discrete signal:

$$x[n] = \cos(2\pi \frac{440n}{F_s})$$  \hspace{1cm} (1.1)

This signal can be viewed as a vector of real values between −1 and 1, where the i’th element of vector is $\cos(2\pi \frac{440i}{F_s})$. Another way of interpreting this signal is as a pure digital tone of 440 Hz, sampled at $F_s$ Hz. These two interpretations are illustrated in Figure 1.1.

Mathematically, this corresponds to expressing a vector in different bases. If our signal $x$ is of length $N$, it can be viewed as a vector in $\mathbb{R}^N$, and the time representation discussed above is simply $x$ expressed in the standard basis for $\mathbb{R}^N$. In order to understand the frequency representation, we must first introduce the discrete Fourier basis for $\mathbb{R}^N$:

**Definition 1.1.** The normalized complex exponentials (of order $N$) is defined as follows:

$$\phi_n = \frac{1}{N} \begin{bmatrix} 1 \\ e^{2\pi in/N} \\ \vdots \\ e^{2\pi i(N-1)n/N} \end{bmatrix}$$

The set $\mathcal{F}_N = \{\phi_n\}_{n=0}^{N-1}$ is called the $N$-point (discrete) Fourier basis.

In Definition 1.1 we loosely referred to the set $\mathcal{F}_N$ as a basis. The following lemma shows that this actually is the case:

**Lemma 1.2.** The normalized complex exponentials $\{\phi_n\}_{n=0}^{N-1}$ of order $N$ forms an orthonormal basis in $\mathbb{R}^N$.

We will not prove this lemma in this paper, but a formal proof can be found in [Rya16, Lemma 2.10].

We are now ready to understand the mathematical interpretation of the frequency domain: The frequency representation of $x$ is $x$ expressed in the
Figure 1.1: Time (left) and frequency (right) representation of the signal in (1.1). In this example we have chosen a sampling frequency of $F_s = 4400$ Hz and a signal length of $N = 20$.

The transition from the standard basis to $\mathcal{F}_N$ is called the discrete Fourier transform (DFT). Conversely, the transition from $\mathcal{F}_N$ to the standard basis is called the inverse discrete Fourier transform (IDFT).

We will denote the change of coordinates matrix from the standard basis of $\mathbb{R}^N$ to the Fourier basis $\mathcal{F}_N$ by $F_N$. Finding the Fourier coefficients $X \in \mathbb{R}^N$ of a signal $x \in \mathbb{R}^N$ then amounts to doing the following matrix-vector multiplication:

$$X = F_N x$$

Before we can start addressing the FFT algorithm we must introduce the notion of Big-O notation:

**Definition 1.3.** Let $f, g$ be two functions $f, g : \mathbb{N} \to \mathbb{R}$. Then $f(n) \in O(g(n))$ if it exists a $c \in \mathbb{R}$ and a $N \in \mathbb{N}$ such that $f(n) \leq cg(n)$ for all $n > N$.

This will help us analyze how the runtime of algorithms change depending on the size of the input.

## 2 The Fast Fourier Transform

One great breakthrough in algorithm design during the 20th century was the introduction of the fast Fourier transform (FFT). IEEE included FFT on their list of the 10 most important algorithms discovered in the last century [DS00]. To make things simpler we will assume that the order $N$ is a power of two.

The idea of the FFT is to break up the $N$-dimensional product in (1.2) to four $N/2$-dimensional products like this:

$$F_N x = \begin{bmatrix} F_{N/2} & D_N F_{N/2} \\ F_{N/2} & -D_N F_{N/2} \end{bmatrix} \begin{bmatrix} x_e \\ x_o \end{bmatrix}$$

(2.1)

Here, $x_e, x_o \in \mathbb{R}^{N/2}$ represents the even- and odd entries of $x$, respectively. The matrix $D_N \in \mathbb{R}^{N/2 \times N/2}$ is the diagonal matrix with entries $e^{-2\pi in/N}$ for $0 \leq n < N/2$.

We will not prove that this algorithm in fact implements the DFT, but take it for granted. A more in depth introduction to the FFT is found in Chapter 8 of [MI11], or in Section 2.4 of [Rya16].
3 Analyzing the Complexity

We will now analyze the number of operations (ie: additions and multiplications) needed to compute the DFT using the old method of direct implementation and using the FFT. We first state our main result:

**Theorem 3.1.** The direct implementation of the DFT using Equation (1.2) requires $O(N^2)$ operations. Using the FFT this is reduced to $O(N \log(N))$.

**Proof.** If we were to implement the DFT directly, that is to calculate Equation (1.2), we would end up doing one $N$-dimensional inner product for each element in $x$. Each inner product consists of $N$ multiplications and $N - 1$ additions. Hence it is clear that one inner product takes $O(N)$ operations. We have to do $N$ of them, meaning that the total number of operations needed for the naive approach is $O(N^2)$.

Using the FFT we first observe that the terms $F_{N/2}x_e$ and $F_{N/2}x_o$ each will appear two times in the calculation of $F_Nx$:

\[
\begin{bmatrix}
    F_{N/2} & D_N F_{N/2} \\
    F_{N/2} & -D_N F_{N/2}
\end{bmatrix}
\begin{bmatrix}
    x_e \\
    x_o
\end{bmatrix} = \begin{bmatrix}
    F_{N/2}x_e + D_N F_{N/2}x_o \\
    F_{N/2}x_e - D_N F_{N/2}x_o
\end{bmatrix}
\]

We will only have to calculate them once, and cache the result for the next computation. The only difference between the two block elements of the result vector is the sign of $D_N$. The calculation of $F_{N/2}x_e$ will demand half the number of operations that the calculation of $F_Nx$ uses. The multiplication of a $N/2 \times N/2$ diagonal matrix and a $N/2$-dimensional vector requires $N/2$ multiplications. This gives us the following recurrence relation (here, $T(N)$ represents the number of operations for a $N$-point FFT):

\[
T(N) = 2T\left(\frac{N}{2}\right) + 2\frac{N}{2} = 2T\left(\frac{N}{2}\right) + N
\]

We observe that the term $T(N/2)$ requires $\log_2(N)$ recursions before reaching the base case of $N = 1$, for each recursion we are doing a $O(N)$ operation, giving us that the complexity for the FFT is $O(N \log(N))$.

Going from a $O(N^2)$ to a $O(N \log(N))$ algorithm makes a quite substantial difference on the tractability of the DFT-problem. The fact that the transition between the frequency domain and time domain can be computed quite efficiently allows for more efficient implementations of filters, which plays a great part in digital signal processing.

**References**

