Lecture XI: The Fast Fourier Transform (FFT) algorithm

Maxim Raginsky

BME 171: Signals and Systems
Duke University

October 17, 2008

This lecture

Plan for the lecture:

1. Recap: the DTFT
2. Limitations of the DTFT
3. The discrete Fourier transform (DFT)
4. Computational limitations of the DFT
5. The Fast Fourier Transform (FFT) algorithm
   - decimation in time
   - main idea
   - analysis
6. Applications of the FFT

Recap: discrete-time Fourier transform

In the last lecture, we have learned about one way of representing discrete-time signals in the frequency domain: the discrete-time Fourier transform (DTFT).

For a signal $x[n]$, the DTFT $X(\Omega)$ is defined as

$$X(\Omega) = \sum_{n=-\infty}^{\infty} x[n]e^{-jn\Omega}.$$

It is a 2π-periodic function of a continuous variable, the frequency Ω.

Limitations of the DTFT; discrete Fourier transform (DFT)

Most discrete-time signal processing is done on digital computers. Hence, unless $X(\Omega)$ can be computed in closed form, storing it in computer memory and manipulating it is difficult.

To cope with memory limitations, we can use another frequency-domain representation of discrete-time signals: the **discrete Fourier transform** (DFT).

The DFT is characterized by the **memory** (or **record**) **length** N $(= 1, 2, 3, \ldots)$, and represents a discrete-time signal $x[n]$ by a vector with N complex components. It can therefore be stored in memory, retrieved, and processed using digital techniques.

Definition of the DFT

Given N, the N-**point DFT** of $x[n]$ is defined as a vector $(X_0, X_1, \ldots, X_{N-1})$, where

$$X_k = \sum_{n=0}^{N-1} x[n] e^{-j2\pi nk/N}, \qquad k = 0, 1, \ldots, N-1.$$

Note that, to compute the DFT, we use only the values

$$x[0], x[1], \ldots, x[N-1].$$

Hence, if the signal is nonzero outside this range of n, the DFT loses some information. However, if the signal is nonzero only for $n = 0, \ldots, N-1$, we can recover it from $\{X_k\}$ via the formula

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X_k e^{j2\pi nk/N}$$

If $x[n] = 0$ for $n < 0$ and $n \geq N$, then the DFT $\{X_k\}$ can be viewed as a sampled version of the DTFT $X(\Omega)$. Indeed, let $\Omega_k = 2\pi k/N$, for $k = 0, 1, \ldots, N-1$. Then

$$\begin{aligned} X(\Omega_k) &= \sum_{n=0}^{N-1} x[n] e^{-jn\Omega_k} \\ &= \sum_{n=0}^{N-1} x[n] e^{-j2\pi nk/N} \\ &= X_k. \end{aligned}$$

Thus, $(X_0, \ldots, X_{N-1})$ are the values of $X(\Omega)$ sampled at $\Omega_k = 2\pi k/N$, $k = 0, 1, \ldots, N-1$.

Computational complexity of the DFT

Despite the fact that the DFT is a vector with finite size, computing it for large values of N is very intensive. In order to compute each

$$X_k = \sum_{n=0}^{N-1} x[n] e^{-j2\pi nk/N},$$

we need N complex multiplications and $N-1$ additions. A multiplication step is more expensive computationally than an addition step, so we will only count multiplications. (Keep in mind, though, that each complex multiplication requires four real multiplications.)

Overall, to compute the entire N values $X_0, \ldots, X_{N-1}$, we need N^2 complex multiplications, which is computationally expensive. Fortunately, in 1965 James Cooley and John Tukey discovered an efficient way of computing the FFT, requiring on the order of $N \log_2 N$ multiplications.

Note: it should be pointed out that the procedure discovered by Cooley and Tukey goes back at least to Carl F. Gauss (around 1805).

The FFT algorithm

We will assume from now on that N is a power of 2, i.e., $N = 2^r$ for some integer $r = 1, 2, \ldots$.

The FFT algorithm relies on the fact that the task of computing the N-point DFT of a signal can be broken down into two tasks, each involving an $N/2$-point DFT. This process can be continued **recursively**, until we end up with 1-point DFTs.

To describe this method, which is referred to as **FFT by decimation in time**, we introduce the notation

$$W_N = e^{-j2\pi/N}$$

Note that W_N is the N-**th root of unity**, i.e.,

$$W_N^N = (e^{-j2\pi/N})^N = 1.$$

The FFT algorithm

We can write

$$X_k = \sum_{n=0}^{N-1} x[n] e^{-j2\pi nk/N} = \sum_{k=0}^{N-1} W_N^{kn}.$$

Now, given $x[n]$, define two signals $a[n]$ and $b[n]$ as follows:

$$\begin{aligned} a[n] &= x[2n], \qquad n = 0, 1, \ldots, N/2 - 1 \\ b[n] &= x[2n+1], \qquad n = 0, 1, \ldots, N/2 - 1. \end{aligned}$$

We can compute the $N/2$-point DFT's of $a[n]$ and $b[n]$:

$$\begin{aligned} A_k &= \sum_{n=0}^{N/2-1} a[n] e^{-j2\pi nk/(N/2)} = \sum_{n=0}^{N/2-1} , \qquad k = 0, 1, \ldots, N/2 - 1 \\ B_k &= \sum_{n=0}^{N/2-1} b[n] e^{-j2\pi nk/(N/2)} = \sum_{n=0}^{N/2-1} , \qquad k = 0, 1, \ldots, N/2 - 1. \end{aligned}$$

The FFT algorithm

The main idea behind the FFT algorithm lies in the fact that

$$X_k = \begin{cases} A_k + W_N^k B_k, & k = 0, 1, \ldots, N/2 - 1 \\ A_{k-N/2} - W_N^k B_k, & k = N/2, N/2 + 1, \ldots, N - 1 \end{cases}$$

Thus, in order to compute the N-point DFT X_k, we need to compute two $N/2$-point DFTs, A_k and B_k, and then combine the results according to this formula.

Note that we can repeat this procedure separately on $a[n]$ and $b[n]$: split them into even- and odd-numbered components, compute the corresponding DFT's, etc.

In this way, the FFT requires only on the order of

$$\frac{N \log_2 N}{2}$$

operations, rather than the N^2 required by the direct method.

The FFT algorithm

We will prove the FFT formula only for $k = 0, 1, \ldots, N/2 - 1$; the second one is proved in the same way.

$$A_k + W_N^k B_k = \sum_{n=0}^{N/2-1} a[n] e^{-j2\pi nk/(N/2)} + W_N^k \sum_{n=0}^{N/2-1} b[n] e^{-j2\pi nk/(N/2)}$$

$$= \sum_{n=0}^{N/2} a[n] W_{N/2}^{nk} + \sum_{n=0}^{N/2-1} b[n] W_{N/2}^{nk} W_N^k.$$

Now, note that

$$W_{N/2}^{nk} = (e^{-j2\pi/(N/2)})^{nk} = (e^{-j2\pi/N})^{2nk} = W_N^{2nk}.$$

Substituting, we get

$$A_k + W_N^k B_k = \sum_{n=0}^{N/2} a[n] W_N^{2nk} + \sum_{n=0}^{N/2-1} b[n] W_N^{(2n+1)k}$$

$$
\begin{aligned}
&= \sum_{n=0}^{N/2-1} x[2n] W_N^{2nk} + \sum_{n=0}^{N/2-1} x[2n+1] W_N^{(2n+1)k} \\
&= \sum_{\substack{\bar{n}=0 \\ \bar{n}\text{ even}}}^{N-1} x[\bar{n}] W_N^{\bar{n}k} + \sum_{\substack{\bar{n}=1 \\ \bar{n}\text{ odd}}}^{N-1} x[\bar{n}] W_N^{\bar{n}k} \\
&= \sum_{n=0}^{N-1} x[n] W_N^{nk} \\
&= \sum_{n=0}^{N-1} x[n] e^{-j2\pi nk/N} \\
&= X_k.
\end{aligned}
$$

Q.E.D.

The same principles can be used to derive the FFT algorithm for the inverse DFT.

Applications of FFT

The FFT is widely used in signal analysis. Some applications include:

- efficient approximation of continuous-time Fourier transforms
- efficient computation of convolutions using the relation

$$x[n] \star v[n] \leftrightarrow X(\Omega)V(\Omega),$$

  provided $x[n]$ and $v[n]$ are zero outside some common range $n = 0, \ldots, N-1$
- removal of noise from sampled data – the idea is that "white noise" has a very special frequency-domain behavior, which makes it easy to detect and remove.