The Fourier transform and the FFT algorithm

Steffen Haug

Introduction

The *discrete Fourier transform* (DFT)

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} kn} \tag{1}$$

transforms a *sequence* $\{x_n\}$ into another sequence $\{X_n\}$. Sequences are simply *arrays of numbers* for our practical purposes, and in the finite case it might sometimes be sensible to think of them as vectors in $\mathbb{F}^N$. Compare it to the continuous Fourier transform ($\xi \sim k$, $t \sim n$):

$$\hat{f}(\xi) = \int_{\mathbb{R}} f(t) e^{-2\pi i \xi t} dt \tag{2}$$

Calling the DFT an approximation of the continuous Fourier transform might be a stretch, but at least it motivates why it looks like it does. Essentially, you can think of $\{x_n\}$ as sampling f at discrete time points. $\{X_n\}$ gives you a discrete approximation of the frequency distribution of f with the integral approximated by a sort of Riemann-sum.

The aim of these notes is to explain *why the hell this works* in terms of basic linear algebra with most of the dense mathematical detail hand-waved away, and finally to explain the idea behind the FFT algorithm.

Why is this even interesting?

The fourier transformed signal $\hat{f}$ gives the frequency spectrum of f. In most real engineering applications f is unknown, and we only have discretely sampled sensor data. Moreover, analytically evaluating an integral is almost always totally infeasible. A discrete version that is simple to evaluate is therefor useful for many applications:

- Processing signals transmitted via EM-waves (Radio, **Internet**, etc.)
- Maritime industry (Ocean waves, etc.)
- Mechanical vibrations in machines and buildings (Civil engineering)
- Geophysics

And the list goes on! This shows up pretty much in every single branch of real engineering. Moreover, most of the applications are time-critical, and a lot of the systems are hard real-time systems. Speed is of the essence!

Why does this even work?

Or: A crash course in functional analysis.

```
- functions form vector space (Hilbert space) L (integrabl
- integrals of the form int f g dx is an inner product <f,
- family of periodic functions exp(iwx) for all w is a ort
- the projection of v onto u is (<v, u> / <u, u>)*u
- exps are normalized: proj of v onto e is simply <v, e>e
- for any frequency w, the component of exp(w) in f is sim
- || Proj_exp f || = <f, exp> = int f exp dx
```

Back to familiar linear algebra: The inner product in L is actually very reminiscent of the inner product in F^k! (Summing componentwise products).

The key idea: Decomposing the Fourier transform

Rewrite (1) as the sum of a sum over even and a sum over odd indices:

$$\begin{aligned} X_k &= \sum_{n=0}^{N/2-1} x_{2n} e^{-\frac{2\pi i}{N} 2nk} \\ &+ \sum_{n=0}^{N/2-1} x_{2n+1} e^{-\frac{2\pi i}{N}(2n+1)k} \end{aligned} \tag{3}$$

Factor out one of the exponentials in the odd-indexed sum:

$$\begin{aligned} X_k &= \sum_{n=0}^{N/2-1} x_{2n} e^{-\frac{2\pi i}{N} 2nk} \\ &+ e^{-\frac{2\pi i}{N} k} \sum_{n=0}^{N/2-1} x_{2n+1} e^{-\frac{2\pi i}{N} 2nk} \end{aligned} \tag{4}$$

Rewrite using $2 = 1/(1/2)$ to move the 2 into the denominator:

$$\begin{aligned} X_k &= \sum_{n=0}^{N/2-1} x_{2n} e^{-\frac{2\pi i}{N/2} nk} \\ &+ e^{-\frac{2\pi i}{N} k} \sum_{n=0}^{N/2-1} x_{2n+1} e^{-\frac{2\pi i}{N/2} nk} \end{aligned} \tag{5}$$

Notice that this is now simply two DFTs of half the length!

$$X_k = E_k + e^{-\frac{2\pi i}{N} k} O_k \tag{6}$$

This enables decomposing a DFT into two smaller DFTs, but by itself this does not improve the performance of our algorithm.

Gaining speed: Reusing intermediate results

```
- key idea: DFT symmetrical around roots of unity; we only need to calculate the first half
- Even-indexed sum has the same parity, odd-indexed sum has the opposite parity
- X_k = E + O
- X_-k = E - O
```

Scaling factors and symbol conventions

there are some practical things anyone using a DFT is likely to stumble across.

radians/s vs Hz. substitution $\omega = 2\pi\xi$. ω quite normal in textbooks because mathematicians like radians.

when you just have an array of samples, the FFT algorithm doesnt actually care what your sample rate is. what is the relationship between the original sample rate, and the x-axis of the output?

ko