Fast Fourier Transform (FFT)

Friday, August 12, 2022 2:11 PM

Computational Complexity of DFT

$$S(k) = \sum_{n=0}^{N-1} s[n] e^{-j\frac{2\pi k n}{N}}$$

has 2N multiplications and 2N-2 additions. We need N computations of S(k) for equally spaced frequencies.

$N(4N-2) = 4N^2 - 2N$ total operations are needed to compute the DFT. This is an $O(N^2)$ algorithm. Is there a better way?

FFT Algorithm

Suppose a discrete-time signal has length 2^L for some $L \in \mathbb{Z}$. It turns out we can recursively divide finding the DFT to a bunch of simple subproblems.

$$S(k) = s(0) + s(1)e^{-j\frac{2\pi k}{N}} + s(2)e^{-j\frac{2\pi(2k)}{N}} + \cdots + s(N-1)e^{-j\frac{2\pi k(N-1)}{N}}$$

$$= \boxed{s(0) + s(2)e^{-j\frac{2\pi k}{N/2}} + \cdots + s(N-2)e^{-j\frac{2\pi k(N/2-1)}{N/2}}}$$

$$+ \boxed{\left[s(1) + s(3)e^{-j\frac{2\pi k}{N/2}} + \cdots + s(N-1)e^{-j\frac{2\pi k(N/2-1)}{N/2}}\right]} e^{-j\frac{2\pi k}{N}}$$

Note we can solve for S(k) by doing two N/2-length DFTs. We still need to evaluate each for $k = 0, 1, \ldots, N-1$, but note that each N/2-length DFT is periodic with period N/2. The $e^{-j\frac{2\pi k}{N}} = f(k)$ has a period of N still, however.

Let's call $\square$ terms E(k) and $\square$ terms O(k). We can now say $S(k) = O(k) + E(k)f(k)$.

Can now say $S(k) = O(k) + E(k)\, f(k)$.

$$E(k+\tfrac{N}{2}) = E(k)$$
$$O(k+\tfrac{N}{2}) = O(k)$$
$$f(k+\tfrac{N}{2}) = -f(k+\tfrac{N}{2})$$ ← 180° rotation in complex plane: multiply by -1
$$f(k+N) = f(k)$$

For $k = 0, 1, 2, \ldots, \frac{N}{2}-1$, we know $S(k) = O(k) + f(k)E(k)$

For $k = N/2, N/2+1, \ldots, N-1$, we see $S(k) = O(k-N/2) + f(k-N/2)\,E(k-N/2)$

$$= O(k) - f(k)\,E(k)$$

from f(k) periodicity

Let's look at pseudo code:

Algorithm: Cooley-Tukey FFT (fft)

Input:

s[n] — signal

N — signal length

Output:

S(k) for $k = 0, 1, \ldots, N-1$

```
if N = 1 then
  L S(0) ← s[0]; //base case
else
  | even_half ← s[n] where 2|n ; //even indices
  | odd_half ← s[n] where 2∤n+1 ; // odd incides
  |
  |   O(k) ← fft(even_half, N/2); //recursive calls
  |   E(k) ← fft(odd_half, N/2); // there are two
  |
  | //Combine N/2-length FFTs
  | for k ∈ [0, N/2 - 1]
  |   | S(k) ← O(k) + e^(j2πk/N) E(k)
  |   | S(k+N/2) ← O(k) - e^(j2πk/N) E(k)
  L
return S(k);
```

} 4 mult.
2+2+2 = 6 add.

See the MATLAB implementation in the GitHub repo.

Relation to Sampling

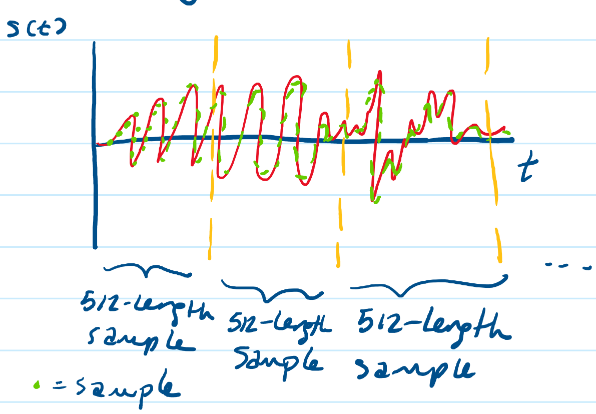
Suppose $s[n]$ is sampled from analog signal $s(t)$. Recall that $f_D = \frac{f_A}{f_s}$. For an N-length signal $s[n]$,

we understand FFT gives N samples of $S(e^{j2\pi f})$, the DTFT. These are equally spaced in digital frequencies $[0, 1)$. Note that $[1/2, 1)$ corresponds to negative frequencies $[-1/2, 0]$.

So, really, we are looking at "real" * frequencies between 0 and $f_s/2$, the Nyquist frequency. Assuming no aliasing, this should capture the signal in its entirity.

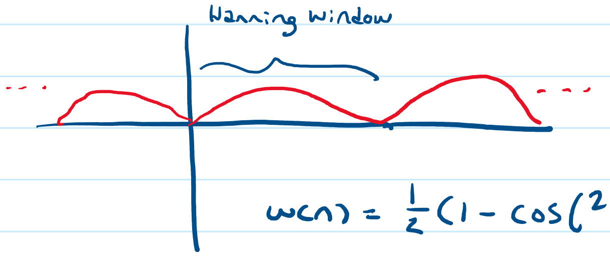
* The meaning of negative frequencies really only makes since for complex exponentials.

Spectrograms



To generate spectrograms of a signal, we sample an analog signal and analyze N-length windows. This is like multiplying by shifted pulse functions, which is abrupt.

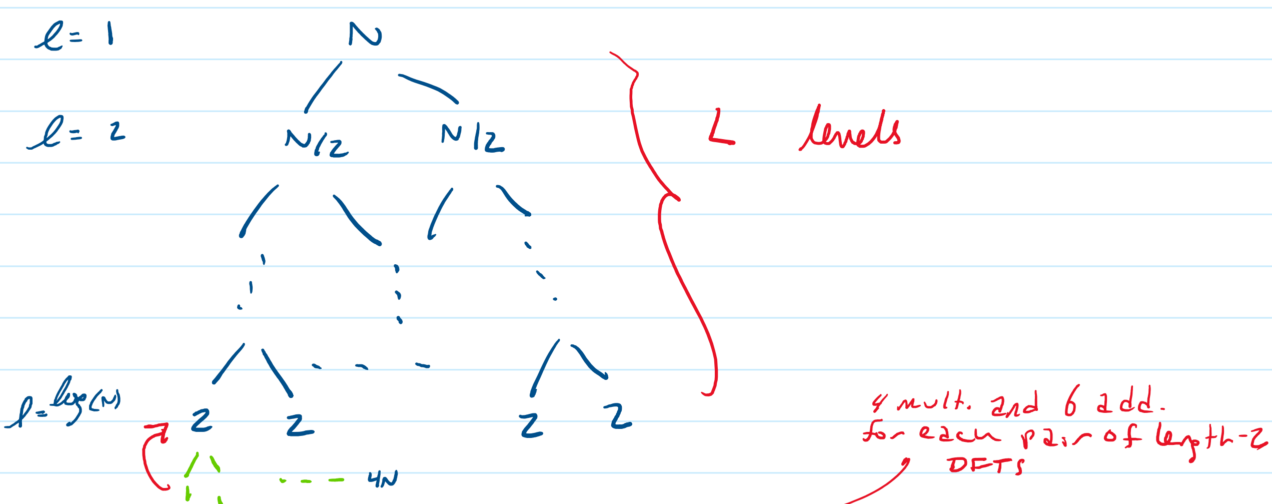
often, we multiply by a windowing function like a Hanning function to reduce high-frequency artifacts stemming from the abrupt nature of a step up or down.



$$w(n) = \frac{1}{2}\left(1 - \cos\left(2\pi n/N\right)\right)$$

Windows are often overlapped b/c using a window periodic in nature sort of modulates the original signal. overlapping windows reduces the effect of windowing on the spectra.

Computational Complexity of FFT



Combining pairs of length-2 DFTs requires $10\left(\frac{N}{4}\right) = \frac{5N}{2}$ computations.

Each Level requires this many computations since $(N/2^l) \cdot 2^l$ is invariant to l.

$\uparrow$ # for loop runs $\quad\quad$ $\uparrow$ # DFTS to find

So $\frac{5N}{2} \underbrace{\log_2(N)}_{\#\text{ levels}}$ is the number of computations needed.

The FFT is $O(N \log N)$ in asymptotic runtime.