

Mathematics of Image and Data Analysis
Math 5467

Lecture 10: The Fast Fourier Transform (FFT)

Instructor: Jeff Calder
Email: jcalder@umn.edu

<http://www-users.math.umn.edu/~jwcalder/5467S21>

Last time

- Intro to the DFT

Today

- The Fast Fourier Transform (FFT)

Recall $f \in \mathbb{C}^n \Leftrightarrow f = (f^{(0)}, f^{(1)}, \dots, f^{(n-1)}), f^{(k)} \in \mathbb{C}$

Definition 1. The *Discrete Fourier Transform (DFT)* is the mapping $\mathcal{D} : L^2(\mathbb{Z}_n) \rightarrow L^2(\mathbb{Z}_n)$ defined by

$$\rightarrow \mathcal{D}f(\ell) = \sum_{k=0}^{n-1} f(k)\omega^{-k\ell} = \sum_{k=0}^{n-1} f(k)e^{-2\pi i k\ell/n},$$

$$L^2(\mathbb{Z}_n) \cong \mathbb{C}^n$$

where $\omega = e^{2\pi i/n}$ and $\mathbb{Z}_n = \{0, 1, \dots, n-1\}$ is the cyclic group $\mathbb{Z}_n = \mathbb{Z}/n$.

Freq. of u_ℓ is $\frac{\ell}{n}$ cycles per grid point.

The DFT can be viewed as a change of basis into the orthogonal basis functions

$$\cos(2\pi k\ell/n) \quad \underline{u_\ell(k)} = \omega^{k\ell} = \underline{e^{2\pi i k\ell/n}}$$

for $\ell = 0, 1, \dots, n-1$.

In audio with 44,100 samples per second 

$\Rightarrow u_\ell$ has frequency $\frac{\ell}{n} \cdot 44,100$ Hz

Inverse Fourier Transform

Theorem 2 (Fourier Inversion Theorem). *For any $f \in L^2(\mathbb{Z}_n)$ we have*

$$(1) \quad f(k) = \frac{1}{n} \sum_{\ell=0}^{n-1} \mathcal{D}f(\ell) \omega^{k\ell} = \frac{1}{n} \sum_{\ell=0}^{n-1} \mathcal{D}f(\ell) e^{2\pi i k \ell / n}.$$

Definition 3 (Inverse Discrete Fourier Transform). The *Inverse Discrete Fourier Transform (IDFT)* is the mapping $\mathcal{D}^{-1} : L^2(\mathbb{Z}_n) \rightarrow L^2(\mathbb{Z}_n)$ defined by

$$\mathcal{D}^{-1}f(\ell) = \frac{1}{n} \sum_{k=0}^{n-1} f(k) \omega^{k\ell} = \frac{1}{n} \sum_{k=0}^{n-1} f(k) e^{2\pi i k \ell / n}.$$

Ex 3

$$\mathcal{D}^{-1} = \frac{1}{n} \overline{\mathcal{D} f}$$

Basic properties

Exercise 4. Show that the DFT enjoys the following basic shift properties.

1. Recall that $u_\ell(k) := e^{2\pi i k \ell / n} = \omega^{k\ell}$. Show that

$$\mathcal{D}(f \cdot u_\ell)(k) = \cancel{\mathcal{D}f(k+\ell)} = \mathcal{D}f(k-\ell).$$

2. Let $T_\ell : L^2(\mathbb{Z}_n) \rightarrow L^2(\mathbb{Z}_n)$ be the translation operator $T_\ell f(k) = f(k - \ell)$. Show that

$$\mathcal{D}(T_\ell f)(k) = e^{-2\pi i k \ell / n} \mathcal{D}f(k).$$

[Hint: You can equivalently show that $\mathcal{D}^{-1}(f \cdot u_\ell)(k) = \mathcal{D}^{-1}f(k+\ell)$, using an argument similar to part 1.]

$$\begin{aligned} \textcircled{1} \quad \mathcal{D}(f \cdot u_\ell)(k) &= \sum_{j=0}^{n-1} f(j) u_\ell(j) \omega^{-jk} \\ &= \sum_{j=0}^{n-1} f(j) \omega^{j\ell} \omega^{-jk} \end{aligned}$$

△

$$= \sum_{j=0}^{n-1} f(j) \omega^{-j(k-l)} = Df(k-l)$$

$$\begin{aligned} \textcircled{2} \quad D^{-1}(f \cdot u_l)(k) &= \frac{1}{n} \sum_{j=0}^{n-1} f(j) \omega^{jl} \omega^{lk} \\ &= \frac{1}{n} \sum_{j=0}^{n-1} f(j) \omega^{j(l+k)} \\ &= D^{-1}f(l+k) \\ &= (T_{-l} D^{-1}f)(k). \end{aligned}$$

Take D on both sides

$$f \cdot u_\ell = D(T_{-\ell} D^{-1} f)$$

Write $g = D^{-1} f$, $f = Dg$ so

$$Dg \cdot u_\ell = D(T_{-\ell} g)$$

Swap ℓ for $-\ell$

$$\Rightarrow D(T_\ell g) = Dg \cdot u_{-\ell}$$

Computational Complexity

Question: How many operations (multiplications or additions) does it take to compute $\mathcal{D}f$ for $f \in L^2(\mathbb{Z}_n)$? Recall we have to compute

$$\mathcal{D}f(\ell) = \sum_{k=0}^{n-1} f(k)e^{-2\pi i k \ell / n}$$

for $\ell = 0, 1, \dots, n-1$.

Answer: $\mathcal{D}f(\ell)$ takes n multiplications,
and $n-1$ additions. ($2n-1$)

Then do this n times $\ell=0, \dots, n-1$

$$\Rightarrow n(2n-1) = 2n^2 - n$$

The Fast Fourier Transform (FFT)

The Fast Fourier Transform computes $\mathcal{D}f$ in $O(n \log n)$ operations. It is one of the most important and widely used algorithms in science and engineering.

- One of the top 10 algorithms of 20th Century — IEEE Computing in Science & Engineering magazine.
- “The most important numerical algorithm of our lifetime” — Gilbert Strang, MIT.

The Fast Fourier Transform (FFT)

Notation: Let n be even. For $f \in L^2(\mathbb{Z}_n)$ the even and odd parts of f , denoted f_e and f_o , respectively, are the functions in $L^2(\mathbb{Z}_{\frac{n}{2}})$ defined by

$$f_e(k) = f(2k) \quad \text{and} \quad f_o(k) = f(2k + 1),$$

for $k = 0, 1, \dots, \frac{n}{2} - 1$. We also denote by \mathcal{D}_n the DFT on $L^2(\mathbb{Z}_n)$.

The FFT is based on the following observation.

Lemma 5. *For each $f \in L^2(\mathbb{Z}_n)$ with n even we have*

$$(2) \quad \mathcal{D}_n f(\ell) = \mathcal{D}_{\frac{n}{2}} f_e(\ell) + e^{-2\pi i \ell / n} \mathcal{D}_{\frac{n}{2}} f_o(\ell),$$

for $\ell = 0, 1, \dots, n - 1$.

$$\underline{P_{\text{cont}}} = D_n f(l) = \sum_{k=0}^{n-1} f(k) e^{-2\pi i k l / n}$$

$$= \sum_{k=0}^{\frac{n}{2}-1} f(2k) e^{-2\pi i (2k) l / n}$$

even part

$$+ \sum_{k=0}^{\frac{n}{2}-1} f(2k+1) e^{-2\pi i (2k+1) l / n}$$

odd part.

$$= \sum_{k=0}^{\frac{n}{2}-1} f_e(k) e^{-2\pi i k l / (\frac{n}{2})}$$

$$+ e^{-2\pi i l / n} \sum_{k=0}^{\frac{n}{2}-1} f_o(k) e^{-2\pi i k l / (\frac{n}{2})}$$

$$= D_{\frac{n}{2}} f_e(\ell) + e^{-2\pi i \ell / n} D_{\frac{n}{2}} f_o(\ell) \quad \square$$

Remember \mathbb{Z}_n is cyclic

Remark 6. In the expression

$$\mathcal{D}_n f(\ell) = \mathcal{D}_{\frac{n}{2}} f_e(\ell) + e^{-2\pi i \ell / n} \mathcal{D}_{\frac{n}{2}} f_o(\ell),$$


it is important to point out that $\mathcal{D}_n f \in L^2(\mathbb{Z}_n)$ and $\mathcal{D}_{\frac{n}{2}} f_e, \mathcal{D}_{\frac{n}{2}} f_o \in L^2(\mathbb{Z}_{\frac{n}{2}})$. For $\frac{n}{2} \leq \ell \leq n-1$, the periodicity of $\mathbb{Z}_{\frac{n}{2}}$ gives that

$$\mathcal{D}_n f(\ell) = \mathcal{D}_{\frac{n}{2}} f_e(\ell - \frac{n}{2}) + e^{-2\pi i \ell / n} \mathcal{D}_{\frac{n}{2}} f_o(\ell - \frac{n}{2}).$$

This is important to keep in mind in practical implementations, since indexing arrays in Python or Matlab does not work cyclically.

The FFT Algorithm

The FFT is based on using the expression

$$\mathcal{D}_n f(\ell) = \mathcal{D}_{\frac{n}{2}} f_e(\ell) + e^{-2\pi i \ell / n} \mathcal{D}_{\frac{n}{2}} f_o(\ell),$$


$O(n)$

to recursively reduce to the DFT on shorter signals (by half each time we iterate). We only need to iterate $\log_2 n$ times before reaching D_1 (if $n = 2^k$), and $D_1 f = f$ is the identity.

Rough Computational Complexity: Whenever we use the expression above to combine the DFT on smaller spaces, it costs $O(n)$ operations. The splitting is done $\log_2 n$ times, yielding $O(n \log_2 n)$ operations.

The FFT Algorithm

$$D_n f(l) = \sum_{k=0}^{n-1} f(k) e^{-2\pi i k l / n}$$

= 1 if $k=0$.

The FFT can be implemented with recursive programming.

Algorithm 1 The Fast Fourier Transform (FFT) in Python

```
1 import numpy as np
2
3 def fft(f):
4     n = len(f)
5     k = np.arange(n)    #Array [0,1,2,...,n-1]
6     if n == 1:
7         return f        #D_1 is the identity
8     else:
9         Dfe = fft(f[::2]) #FFT of even samples
10        Dfo = fft(f[1::2]) #FFT of odd samples
11        Dfe = np.hstack((Dfe,Dfe)) #Extend periodically
12        Dfo = np.hstack((Dfo,Dfo)) #Extend periodically
13        return Dfe + np.exp(-2*np.pi*1j*k/n)*Dfo #Combine via Lemma
```

Complexity Analysis

We measure complexity in terms of the number of real operations, which are additions, subtractions, multiplications, or divisions of two real numbers.

~~Applying other functions (like sin and cos) can take longer and might sometimes be counted differently. We will count these as one operation too.~~

$$(a+ib) + (c+id) = (a+b) + i(c+d)$$

Important points:

- Adding two complex numbers requires 2 real operations.
- Multiplying two complex numbers requires 6 real operations.

$$\begin{aligned}(a+ib) \cdot (c+id) &= ac + iad + ibc + i^2bd \\ &= \underline{(ac-bd)} + i \underline{(ad+bc)}\end{aligned}$$

Complexity Analysis

We define

$$A_n = \text{Number of real operations taken by the FFT on } L^2(\mathbb{Z}_n).$$

Since \mathcal{D}_1 is the identity, $A_1 = 0$.

We assume that $n = 2^k$ so we can always split evenly. Using the expression

$$\mathcal{D}_n f(\ell) = \mathcal{D}_{\frac{n}{2}} f_e(\ell) + e^{-2\pi i \ell / n} \mathcal{D}_{\frac{n}{2}} f_o(\ell),$$

we obtain the recursion

$$A_n = 2A_{\frac{n}{2}} + 8n.$$

1 add. →

↑

1 mult. = 6 real op.

↑
1 add + 1 mult. n times.

Complexity Analysis

Lemma 7. Let $n = 2^k$ for a positive integer k , and assume A_n satisfies

$$A_n = 2A_{\frac{n}{2}} + 8n$$

for $n \geq 2$ and $A_1 = 0$. Then we have

$$(3) \quad A_n = 8n \log_2 n.$$

Note: The lemma says that the FFT on a signal of length n , where n is a power of 2, takes exactly $8n \log_2 n$ operations.

Proof:
$$A_n = 2A_{\frac{n}{2}} + 8n$$
$$= 2\left(2A_{\frac{n}{2^2}} + 8\frac{n}{2}\right) + 8n$$

$$= 2^2 A \frac{n}{2^2} + 8n + 8n$$

$$= 2^2 \left(2 A \frac{n}{2^3} + 8 \frac{n}{2^2} \right) + 8n + 8n$$

$$= 2^3 A \frac{n}{2^3} + 8n + 8n + 8n$$

Do k -times $\begin{matrix} \uparrow \\ \uparrow \\ \uparrow \end{matrix}$

Prove by induction $\rightarrow = 2^k A \frac{n}{2^k} + 8nk, \quad k \geq 1$

Take k so that $2^k = n$ to get

$$A_n = n \cancel{A_1} + 8n \log_2 n \quad (k = \log_2 n) \quad A_1 = 0$$

$$= 8n \log_2 n$$

XII

Remark: $A_n = 5n \log_2 n$ by clever reuse
of multiplications.

HW #3: Split-radix FFT $4n \log_2 n$

2007: $\frac{34}{9} n \log_2 n$

The Fast Inverse Fourier Transform

The FFT can be easily extended to compute the inverse DFT, using the analogous identity

$$(4) \quad \mathcal{D}_n^{-1} f(\ell) = \frac{1}{2} \mathcal{D}_{\frac{n}{2}}^{-1} f_e(\ell) + \frac{1}{2} e^{2\pi i \ell / n} \mathcal{D}_{\frac{n}{2}}^{-1} f_o(\ell).$$

The proof of (4) is left to an exercise.

Alternatively, we may use the identity (also left as an exercise)

$$\mathcal{D}_n^{-1} = \frac{1}{n} \overline{\mathcal{D}_n f}$$

to compute the inverse DFT efficiently using a single forward FFT, two complex conjugation operations, and an elementwise division.

FFT in Python ([.ipynb](#))