Mathematics of Image and Data Analysis
Math 5467

The Fast Fourier Transform

Instructor: Jeff Calder
Email: jcalder@umn.edu

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

Last time

- Intro to the DFT

Today

- The Fast Fourier Transform (FFT)

Recall $f \in L^2(\mathbb{Z}_n)$, $f = (f(0), f(1), \ldots, f(n-1)) \in \mathbb{C}^n$

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

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

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

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

$$\mathcal{D}f(\ell) = \langle f, u_\ell \rangle, \quad u_\ell(k) = e^{2\pi i k\ell/n}$$

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

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

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

Inverse Fourier Transform

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

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

Definition 3 (Inverse Discrete Fourier Transform)**.** The *Inverse Discrete Fourier Transform (IDFT)* is the mapping $\mathcal{D}^{-1} : L^2(\mathbb{Z}_n) \to L^2(\mathbb{Z}_n)$ 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: $D^{-1}f = \frac{1}{n}\overline{D(\overline{f})}$

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

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

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

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

~~[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.]~~

△

Proof:

① $$D(f u_\ell)(k) = \sum_{q=0}^{n-1} f(q) u_\ell(q) \omega^{-qk}$$

$$= \sum_{q=0}^{n-1} f(q) \omega^{\ell q} \omega^{-qk}$$

$$= \sum_{q=0}^{n-1} f(q)\, w^{-(k-\ell)q}$$

$$= Df(k-\ell).$$

② $$D(T_\ell f)(k) = \sum_{q=0}^{n-1} T_\ell f(q)\, w^{-qk}$$

$$= \sum_{q=0}^{n-1} f(q-\ell)\, w^{-qk}$$

$p = q-\ell$

$q = p+\ell$

$p : -\ell \rightarrow n-1-\ell$

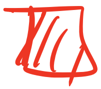
$$= \sum_{p=-\ell}^{n-1-\ell} f(p)\, w^{-(p+\ell)k}$$

$$= w^{-\ell k} \sum_{p=-\ell}^{n-1-\ell} f(p) w^{-pk}$$

$Df(K)$.

$\downarrow$

by periodicity



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

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

$n+n-1$ op.

$2n-1$

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

Takes $n(2n-1) \sim 2n^2 - n = O(n^2)$

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 algorihtms 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 $k = 0, 1, \ldots, \frac{n}{2} - 1$. 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*

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

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

<u>Proof</u>: $D_n f(\ell) = \sum_{k=0}^{n-1} f(k) e^{-2\pi i k\ell/n}$

$$= \underbrace{\sum_{k=0}^{\frac{n}{2}-1} f(2k)\, e^{-2\pi i (2k)\ell/n}}_{\text{even terms}} + \underbrace{\sum_{k=0}^{\frac{n}{2}-1} f(2k+1)\, e^{-2\pi i (2k+1)\ell/n}}_{\text{odd terms}}$$

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

$$= D_{\frac{n}{2}} f_e(k) + e^{-2\pi i (\frac{\ell}{\cancel{2}})/(\frac{n}{\cancel{2}})} \sum_{k=0}^{\frac{n}{2}-1} f_o(k)\, e^{-2\pi i \frac{k\ell}{(\frac{n}{2})}}$$

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

Remember $\mathbb{Z}_n$ is cyclic

$L^2(\mathbb{Z}_n)$ $\quad$ $L^2(\mathbb{Z}_{\frac{n}{2}})$

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

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

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

The FFT Algorithm

$O(n)$

The FFT is based on using the expression

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

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.

$$D_1 f(0) = \sum_{k=0}^{0} f(k) e^{-2\pi i 0 \cdot 0/1} = f(0)$$

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.

$$\frac{n}{2^k} = 1$$

$$n = 2^k$$

$$k = \log_2 n$$

The FFT Algorithm

The FFT can be implemented with recursive programming.

Algorithm 1 The Fast Fourier Transform (FFT) in Python

```
import numpy as np

def fft(f):
    n = len(f)
    k = np.arange(n)   #Array [0,1,2,...,n-1]
    if n == 1:
        return f       #D_1 is the identity
    else:
        Dfe = fft(f[::2])  #FFT of even samples
        Dfo = fft(f[1::2]) #FFT of odd samples
        Dfe = np.hstack((Dfe,Dfe))  #Extend periodically
        Dfo = np.hstack((Dfo,Dfo))  #Extend periodically
        return Dfe + np.exp(-2*np.pi*1j*k/n)*Dfo  #Combine via Lemma
```

$$a + ib \sim (a, b)$$

Complexity Analysis

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

$$(a + bi) + (c + di) = (a + b) + (c + d)i$$

Important points:

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

$$(a+bi)(c+di) = ac + adi + bci + bdi^2$$

$$= (ac - bd) + (ad + bc)i$$

$$= (ac - bd, ad + bc)$$

$\lambda(a+bi) = \lambda a + \lambda b i$

$(\lambda r)e^{i\theta}$

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

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

2 6 8n

we obtain the recusion

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

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) $$A_n = 8\,n\log_2 n.$$

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:

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

$$= 2\left(2A_{\frac{n}{4}} + 8\left(\frac{n}{2}\right)\right) + 8n$$

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

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

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

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

$$\vdots$$

$$= 2^k A_{\frac{n}{2^k}} + k\cdot 8n$$

$n = 2^k$

$k = \log_2(n)$

$$= n A_1 + 8n\log_2 n$$

$$A_1 = 0$$

$\boxed{\text{Th}}$

1. $A_n \sim 5n\log_2 n$

2. Split-Radix FFT: $\boxed{\le 4n\log_2 n}$
   ↳ HW#3

3. 2007 $\quad cn\log_2 n, \quad c < 4.$

The Fast Inverse Fourier Transform

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

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

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

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

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

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

FFT in Python (.ipynb)