Data-sparse matrix computations
Lecture 2: FFT - The Fast Fourier Transform

Lecturer: Anil Damle
Scribers: Mateo Díaz, Mike Sosa, Paul Upchurch

August 24th, 2017

1 Introduction

It is without a doubt, one crucial piece of the toolbox of numerical methods. The FFT or Fast Fourier Transform is a fast algorithm used to compute the Discrete Fourier Transform. It is one of the most common discrete transforms used today.

Note: While many other algorithms used throughout this course can be coded up for use in other projects, you shouldn't attempt to do this with the FFT. The FFT has been fine tuned over the past 20 to 30 years to include various optimizations and to account for different edge cases. Many different libraries for the FFT exist, but a good common library to use would be the FFTW (Fastest Fourier Transform in the West).

1.1 Fourier Transform

Given an integrable complex-valued function $f : \mathbb{R} \to \mathbb{C}$, one can define its Fourier Transform as a function given by

$$\hat{f}(x) = \int_{\infty}^{\infty} f(t)e^{-2\pi ixt}dt.$$

This transformation is a bijection, and we get get his inverse is

$$f(t) = \int_{\infty}^{\infty} \hat{f}(x)e^{2\pi ixt}dx.$$

Intuitively, the transform decomposes the function in terms of its frequency domain. We usually think of it as a signal $f(t)$ as a function that changes over time; however, it is possible to consider any function as a combination of periodic oscillatory fictions (or frequencies). Thus, it is possible to write a function as an infinite sum (integral) of periodic functions. With this in mind, each function evaluation $\hat{f}(x)$, can be seen as the coefficient associated with the oscillatory function $e^{2\pi ixt}$ in the infinite sum.

In many contexts it is easier to consider the function written in the frequency domain. We will see later some operations, such as convolution, can be simpler to handle in this domain, both from a symbolic, and a numerical viewpoint. This transform is interesting on its own, and has been studied in depth from a pure theoretical perspective. However, in these notes we are mainly concern with the numerics involved, that is why we are mainly going to talk about the discrete version of the transform.

1.2 Discrete Fourier Transform

Assume we are given a signal (either real or complex) of length N, encoded in a vector

$$x = \begin{bmatrix} x_0 \\ x_2 \\ \vdots \\ x_{n-1} \end{bmatrix}.$$

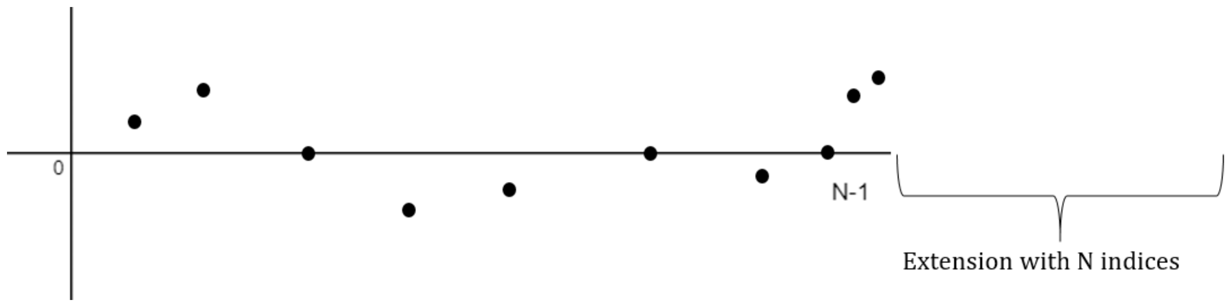


Figure 1: Periodicity of the function described by x.

We think of this vector as a discrete set of evaluations of the function, i.e. $x_k = f(t_k)$ for some fix t_k. Notice that this doesn't define a priori a function with full domain $\mathbb{R}$; however, we are only going to consider periodic functions, therefore, the vector represents only one period of the function, see Figure **??**. We define its Discrete Fourier transform (DFT) as

$$\hat{x}_k = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} x_n e^{-2\pi i k n/N} \quad \text{for } k = 0, 1, \ldots, N-1$$

Notice that this is just a discretization of the transform described in the previous section, where our delta step becomes $1/\sqrt{N}$.

Remark 1. *Here are some caveats to take into account: (1) Since f is periodic you can find in the literature that k is defined to range in $\left[-\frac{N-1}{2}, \frac{N-1}{2}\right]$ when N odd, and $\left[-\frac{N}{2}, \frac{N}{2}\right]$ otherwise. We adopt our notation for the seek of clarity. (2) We might ignore the constant factor, $1/\sqrt{N}$, in some of our calculations. but be warned, it is there.*

Just as in the continuous case, with the DFT, we can also reconstruct the original signal using

$$x_j = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} \hat{x}_k \underbrace{e^{-2\pi i j n/N}}_{\text{functions oscillate a freq at } \approx k}.$$

Thus, it lets us write the function in terms of periodic functions (i.e. the frequency domain). This defines an N periodic extension of $\{x\}_{n=0}^{N-1}$. Formally

$$x_{j+lN} = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} \hat{x}_k e^{-2\pi i (j+lN)n/N} = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} \hat{x}_k e^{-2\pi i j n/N} \underbrace{e^{-2\pi i l N/N}}_{=1} = x_j.$$

In the next section we will describe some applications to see why is this transform useful. But for now, let's get a grasp on some of the properties of this transform. The fist thing to notice is that the DFT is just matrix-vector multiplication. Thus, if we define a $n \times n$ matrix F with

$$F_{jk} = \frac{1}{\sqrt{N}} e^{-2\pi i (j-1)(k-1)/N}$$

where $j, k \in \{1, \ldots, n\}$. Then, it is possible to describe the Discrete Fourier Transform as $\hat{x} = Fx$. In full generality matrix-vector multiplication has complexity $O(n^2)$, for huge signals, this can be too expensive. But this matrix has a very particular structure, can we take advantage of it to improve this complexity? We will see in Section 2 that the answer to this question is yes, we can actually reduce it to $O(n \log(n))$!

The paradigm in which many numerical methods use the DFT is the following, assume that the input to be a signal x,

1. Compute the DFT, $\hat{x} = Fx$.

2. Process your signal in the frequency domain, $\hat{y} = \Phi(\hat{x})$.

3. Pull it back to the discrete, $y = F^{-1}\hat{y}$.

Some methods receive, or return multiple signals, but follow the same idea. Thus, we would like to have a way to very efficiently apply the inverse F^{-1} as well. Luckily, applying F is just as simply as applying F^{-1}, this is formalized in the following Lemma.

Lemma 1. *$F^{-1} = F^*$ (Complex conjugate).*

Proof. Let $F_1, \dots, F_N$ denote the columns of the matrix F. Then, the rows of F^* are exactly the component-wise complex-conjugate of $F_1, \dots, F_N$, denoted by $F_1^*, \dots, F_N^*$. Then,

$$\begin{aligned}
(F^*F)_{jk} = F_j^* F_k &= \frac{1}{N}\sum_{l=1}^{N} e^{2\pi i(l-1)(j-1)/N} e^{-2\pi i(l-1)(k-1)/N} \\
&= \frac{1}{N}\sum_{l=1}^{N} e^{2\pi i(l-1)(j-k)/N} \\
&= \frac{1}{N}\sum_{l=1}^{N} \left(e^{2\pi i(j-k)/N}\right)^{(l-1)} \\
&=: \frac{1}{N}\sum_{l=1}^{N} \omega_{jk}^{l-1} \\
&= \frac{1}{N}\begin{cases} \frac{1-\omega_{jk}^N}{1-\omega_{jk}} & \text{if } j \neq k \\ N & \text{otherwise} \end{cases} \\
&= \delta_{jk}.
\end{aligned}$$

where δ_{jk} is the dirac delta. Note that we can factorize $1 - z^N = (1-z)\sum_{i=0}^{N-1} z^k$, additionally $\omega jk = e^{2\pi i(j-k)/N}$ is a root of unity, and $\omega = 1$ if, only if, $j = k$. By combining these facts we get the penultimate penultimate equality. Thus, $F^*F = I$, and the result follows. $\square$

1.3 Some applications

Before, we continue to the description of a fast algorithm to compute the DFT, let us review some interesting applications. As we will see, one of the main perks of working in the frequency domain is that convolutions become just products, and consequently can be computed fast.

1. **Spectral analysis.** In signal processing, one is interested in the decomposition (or spectrum) given by the DFT. A signal can encode an audio file, an image, an MRI, or a text file. Decomposing signals allows us to understand them as sums of simpler frequency function, and thus their inherit structure.

2. **Data compression.** We live in a world were tons of information is continuously shared, it is of fundamental importance to be able to compress it, but how can we do it? Suppose that you decompose a signal into individual frequencies. Our measurement instruments are always affected by noise, so this decomposition will very likely include very small coefficients that can be ignore to get a compressed version the signal. In a some how similar paradigm, one can discard the frequencies that are not perceptible to the human senses, this is the basic idea of algorithms like JPEG.

3. **Polynomial multiplication.** Assume you are given two polynomials with real coefficients $a(x), b(x) \in \mathbb{R}[x]$ with order m, and n respectively, and you're asked to compute its product. Note that

$$c(x) := a(x)b(x) = \sum_{i=0}^{n+m} \underbrace{\left(\sum_{j=0}^{i} a_i b_{i-j}\right)}_{c_i :=} x^i.$$

which is just a convolution of the signal $(a_0, \dots, a_n)$, and $(b_0, \dots, b_n)$. Hence, it can be efficiently tackle in the frequency domain.

4. **Multiplication of big integers** Recall that any integer with n digits can be written as

$$\sum_{i=0}^{n-1} d_{i+1} 10^i$$

where d_i is the ith digit from right to left. Hence, we can associate a polynomial $p_N(x) = \sum_i d_{i+1} x^i$ to any integer N. Suppose we want to calculate the product of two giant integer N, M, then, it is faster to compute the product $p_N(x)p_M(x)$ using FFT and evaluate it at 10.

5. **Stochastic processes.** In a discrete random walk, one has a lattice in the $\mathbb{R}^n$ (for example $\mathbb{Z}^d$), and a probability distribution Φ that describes the probability of jumping to other points in the lattice. A natural question is what's the distribution of the walk after n steps? it turns out that the distribution is described as the nth convolution of the distribution, Φ^{*n}, which can be computed very efficiently once the distribution is written in the frequency domain.

2 The Fast Fourier Transform

In this section we present the a fast algorithm to compute the DFT in $O(n \log(n))$ time. It is easier to understand this complexity when we consider a power-of-two dimension, we will start by describing this case, and then we will move to the case where the dimension can be decomposed as product of two integers.

2.1 Cooley-Tukey algorithm (Easy case)

Let's consider the specific case where $N = 2^m$. The key idea behind it is to use a recursive algorithm to compute the product. For this we are going to write our length N DFT as two smaller DFTs of length $N/2$, and then repeat this process until we hit the FFT of 1 dimensional signals. Let's start by decomposing the formula

$$\begin{aligned}
\hat{x}_k &= \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} x_n e^{-2\pi i k n/N} \\
&= \frac{1}{\sqrt{N}} \left(\sum_{n=0}^{N/2-1} x_{2n} e^{-2\pi i k 2n/N} + \sum_{n=0}^{N/2-1} x_{2n+1} e^{-2\pi i k(2n+1)/N} \right) \\
&= \frac{1}{\sqrt{N}} \left(\sum_{n=0}^{N/2-1} x_{2n} e^{-2\pi i k n/(N/2)} + e^{-2\pi i k/N} \sum_{n=0}^{N/2-1} x_{2n+1} e^{-2\pi i k n/(N/2)} \right).
\end{aligned}$$

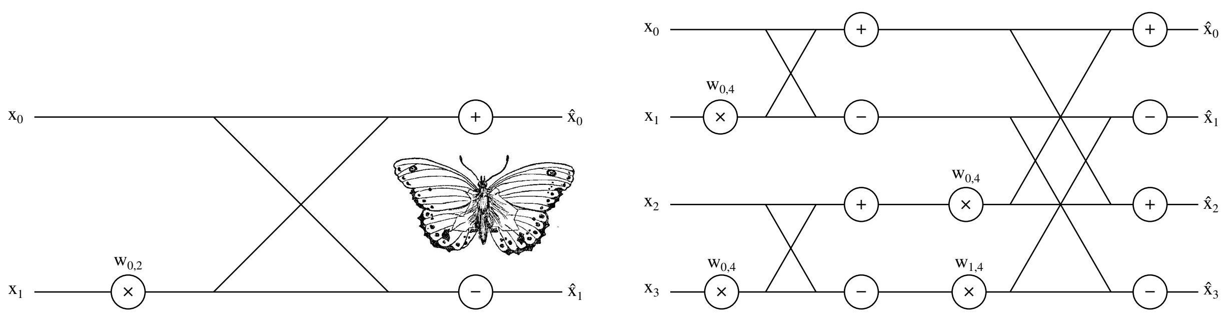
This is almost two DFTs of length $N/2$, one for the even terms, $\hat{x_e}$ and one for the odd terms, $\hat{x_o}$, except $\hat{x}_k$ is length N. By Euler's formula we know that $e^{-2\pi i a} = 1$ (a any integer) thus $(\hat{x_e})_k = (\hat{x_e})_{k \pm N/2}$. Now we can express $\hat{x}_k$ in terms of $\hat{x_e}$ and $\hat{x_o}$.

$$\hat{x}_k = \begin{cases} (\hat{x_e})_k + e^{-2\pi i k/N} (\hat{x_o})_k & \text{for } k = 0, 1, \dots, N/2 - 1 \\ (\hat{x_e})_{k-N/2} + e^{-2\pi i k/N} (\hat{x_o})_{k-N/2} & \text{for } k = N/2, N/2+1, \dots, N-1 \end{cases}$$

Again, with Euler's formula we can simplify this to

$$\begin{aligned}
\hat{x}_k &= (\hat{x_e})_k + e^{-2\pi i k/N} (\hat{x_o})_k && \text{for } k = 0, 1, \dots, N/2 - 1 \\
\hat{x}_{k+N/2} &= (\hat{x_e})_k - e^{-2\pi i k/N} (\hat{x_o})_k && \text{for } k = 0, 1, \dots, N/2 - 1
\end{aligned}$$

Thus, we can express a power-of-two-sized DFT as two power-of-two-sized DFTs of length $N/2$ if $m > 0$.



(a) Length-2 DFT (aka "butterfly"). Image credit: Lord Robert Baden-Powell, 1915.

(b) Length-4 DFT

Figure 2: Computation of power-of-two-sized DFTs

Computation The building block of a power-of-two-sized DFT is a length-2 DFT (aka "butterfly") shown in Figure 2(a). Each butterfly needs two complex add/subs and one complex multiply. The multiplication factor is $w_{k,N} = e^{-2\pi ik/N}$ (aka "twiddle factor"). The next higher-level DFT is length 4 and is composed of two butterflies (see Figure 2(b)). Since the lengths are powers of two we can express any twiddle factor in terms of the next higher level of the computation by $w_{k,N} = w_{2k,2N}$. For a length-N DFT there are $\log N$ levels of $N/2$ butterflies so the total computation is $(N/2)\log N$ complex muls and $N \log N$ add/subs. This is $2N \log N$ real muls and $3N \log N$ real add/subs.

2.2 Non-power-of-two DFT

What if $N \neq 2^m$? Consider the case that the DFT is of composite length $N = N_1 N_2$ and that at least one of the factors is small. The small factor is described in the literature as the *radix.* A composite-length DFT can be expressed in terms of a DFT of length N_1 and a DFT of length N_2. Thus, the key idea for solving this case is that we recursively decompose the DFT until we have only radix DFTs which can be solved with optimized codes (as large as radix-32 is typical in FFTW [2]).

First, rewrite indices k and n as

$$\begin{aligned} k &= N_1 k_1 + k_2 && \text{with } k_1 = 0, 1, \ldots, N_2 - 1 \text{ and } k_2 = 0, 1, \ldots, N_1 - 1 \\ n &= N_2 n_1 + n_2 && \text{with } n_1 = 0, 1, \ldots, N_1 - 1 \text{ and } n_2 = 0, 1, \ldots, N_2 - 1 \end{aligned}$$

Then, the composite-length DFT (excluding the constant factor, for brevity) is

$$\begin{aligned} \hat{x}_{N_1 k_1 + k_2} &= \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x_{N_2 n_1 + n_2} e^{-2\pi i (N_1 k_1 + k_2)(N_2 n_1 + n_2)/N} \\ &= \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x_{N_2 n_1 + n_2} e^{-2\pi i N_2 n_1 k_2/N} \underbrace{e^{-2\pi i N_1 k_1 N_2 n_1/N}}_{=1} e^{-2\pi i k_2 n_2/N} e^{-2\pi i N_1 k_1 n_2/N} \\ &= \underbrace{\sum_{n_1=0}^{N_1-1} \left[\underbrace{\left[\sum_{n_2=0}^{N_2-1} x_{N_2 n_1 + n_2} e^{-2\pi i k_2 n_1/N_1} \right]}_{\text{DFT of length } N_1} e^{-2\pi i k_2 n_2/N} \right] e^{-2\pi i k_1 n_2/N_2}}_{\text{DFT of length } N_2} \end{aligned}$$

So, we can write a DFT of length $N = N_1 N_2$ as N_2 DFTs of length N_1 and 1 of length N_1.

If N has a large prime factor then we can use Rader's or Bluestein's algorithm [3, 1]. Both algorithms rewrite the DFT as a cyclic convolution. The key idea of Bluestein's algorithm is that the prime-length DFT can be replaced by two non-prime-length DFTs. First, we have the fact that

$$2kn = k^2 + n^2 - (k-n)^2$$

Then, rewrite the DFT as

$$\begin{aligned}\hat{x}_k &= \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} x_n e^{-2\pi i k n/N} \\ &= \frac{1}{\sqrt{N}} e^{-\frac{\pi i}{N} k^2} \underbrace{\sum_{n=0}^{N-1} x_n e^{-\frac{\pi i}{N} n^2} e^{\frac{\pi i}{N}(k-n)^2}}_{\text{convolution}}\end{aligned}$$

So, a prime-length DFT can be written as a convolution, which can be performed with a pair of zero-padded DFTs of length at least $2N-1$. Since the length is unbounded from above we can choose a composite or power-of-two length.

References

[1] Leo Bluestein. A linear filtering approach to the computation of discrete fourier transform. *IEEE Transactions on Audio and Electroacoustics*, 18(4):451–455, 1970.

[2] Matteo Frigo and Steven G. Johnson. The design and implementation of FFTW3. *Proceedings of the IEEE*, 93(2):216–231, 2005. Special issue on “Program Generation, Optimization, and Platform Adaptation”.

[3] Charles M Rader. Discrete fourier transforms when the number of data samples is prime. *Proceedings of the IEEE*, 56(6):1107–1108, 1968.