Math 994-002: Applied and Computational Harmonic Analysis, MSU, Spring 2020

Lecture 06: Finite Length Signals, DFT, and FFT

January 28, 2019

Lecturer: Matthew Hirn

3.3 Finite Length Signals

In practice we cannot store an infinite number of samples $\{f(n)\}_{n\in\mathbb{Z}}$ of a signal f; instead we can only keep a finite number of samples, say $\{f(n)\}_{0\leq n<N}$. We thus must amend our definition of the Fourier transform as well as convolution, which will lead to the Discrete Fourier Transform (DFT) and circular convolution. One thing that will arise is that regardless of whether the original signal f is periodic, we will be forced to think of the finite sampling $(f(n))_{0\leq n<N}$ as a discrete periodic signal with period N. This will lead to border effects which must be accounted for. However, the circular convolution theorem and Fast Fourier Transform will allow for fast computations of convolution operators.

Let $x, y \in \mathbb{C}^N$, which are vectors of length N, e.g., N samples of a signal f such that $x[n] = f(n)$ for $0 \leq n < N$. The inner product between x and y is:

$$\langle x, y\rangle = \sum_{n=0}^{N-1} x[n]y^*[n]$$

We must replace the sinusoids $e^{it\omega}$ ($t \in \mathbb{R}$) and $e^{in\omega}$ ($n \in \mathbb{Z}$), which are continuous in the frequency variable ω, with discrete counterparts. The variable ω is replaced with an index k with $0 \leq k < N$:

$$e_k[n] = \exp\left(\frac{2\pi ikn}{N}\right), \quad 0 \leq n, k < N \tag{14}$$

The *Discrete Fourier Transform (DFT)* of x is defined as:

$$\widehat{x}[k] = \langle x, e_k\rangle = \sum_{n=0}^{N-1} x[n] \exp\left(-\frac{2\pi ikn}{N}\right), \quad 0 \leq k < N$$

The following theorem shows that the set of vectors $\{e_k\}_{0\leq k<N}$ is an orthogonal basis for $\mathbb{C}^N$.

Theorem 3.6. *The family of vectors $\{e_k\}_{0\leq k<N}$ as defined in (14) is orthogonal basis for $\mathbb{C}^N$.*

Thus the DFT is a bijection and hence invertible. Since $\|e_k\|^2 = N$ for all k, it follows from Theorem 3.6 that x can be represented in the orthogonal basis $\{e_k\}_{0\leq k<N}$ as:

$$x[n] = \sum_{k=0}^{N-1} \frac{\langle x, e_k\rangle}{\|e_k\|^2} e_k[n] = \frac{1}{N}\sum_{k=0}^{N-1} \widehat{x}[k] \exp\left(\frac{2\pi ikn}{N}\right)$$

This gives the inverse DFT.

We would like a convolution theorem for the DFT similar to the convolution theorem for $\mathbf{L}^1(\mathbb{R})$ functions and $\boldsymbol{\ell}^1(\mathbb{R})$ sequences. We define convolution of $x, y \in \mathbb{C}^N$ by first extending them to signals $x_0, y_0 \in \boldsymbol{\ell}^1(\mathbb{R})$ defined as:

$$x_0[n] = \begin{cases} x[n] & 0 \leq n < N \\ 0 & n < 0 \text{ or } n \geq N \end{cases}$$

We then define the convolution of x and y as the convolution of x_0 and y_0, and keep only the values with a chance of being nonzero:

$$x * y[n] = \sum_{m \in \mathbb{Z}} x_0[m] y_0[n-m], \quad 0 \leq n < 2N-1$$

In practice, there will be many times when you want to compute such convolutions. Indeed, if x and y are discrete samplings of non-periodic signals f and g, respectively, computing $x * y$ will give a discrete approximation for $f * g$. However, for computational reasons, we will often not want to compute discrete convolutions directly (more on this in a bit). Indeed, it will be better to compute such convolutions "in frequency," which will require a convolution theorem for the DFT. However, the discrete sinusoids $\{e_k\}_{0 \leq k < N}$ are not eigenvectors of discrete convolution operators $Lx = x * h$. The vectors e_k are periodic, but the standard convolution is not; indeed, it extends the vectors x, y to a twice longer vector $x * y$. We therefore define a periodic version of convolution, which is called *circular convolution*.

To define circular convolution, rather than extending x and y with zeros, we will extend them with a periodization over N samples:

$$x_p[n] = x[n \mod N], \quad n \in \mathbb{Z}$$

The circular convolution is defined as:

$$x \circledast y[n] = \sum_{m=0}^{N-1} x_p[m] y_p[n-m]$$

Note that $x \circledast y \in \mathbb{C}^N$. One then has the following circular convolution theorem:

Theorem 3.7. *If $x, y \in \mathbb{C}^N$, then*

$$\widehat{x \circledast y}[k] = \widehat{x}[k]\widehat{y}[k]$$

The key to this theorem, and the DFT more generally, is that since the discrete sinusoids e_k are periodic vectors with period N, the DFT treats all vectors $x \in \mathbb{C}^N$ as periodic vectors with period N. This manifests in the convolution theorem by requiring us to utilize circular convolutions. However, it also means that when computing DFTs, we always need to think of x as a periodic vector with period N. In particular, seemingly "smooth" vectors such as $x[n] = n$ actually have very sharp transitions once made periodic, since $x[N-1] = N-1$ and $x[N] = x[0] = 0$; see Figure 5.

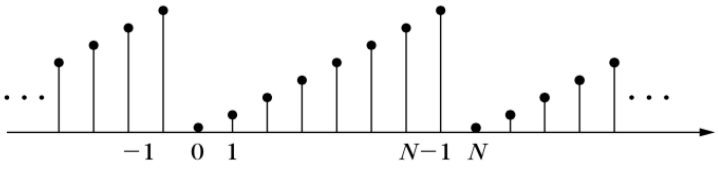

Figure 5: Periodization of the ramp vector $x[n] = n$ on $\mathbb{R}^N$. Taken from Figure 3.3 of *A Wavelet Tour of Signal Processing*.

Remark 3.8. Notice as well, we now have the following correspondences:

- Signals $f \in \mathbf{L}^2(\mathbb{R})$ and $\widehat{f} \in \mathbf{L}^2(\mathbb{R})$, otherwise no restrictions.
- Discrete, but infinite samplings $a \in \boldsymbol{\ell}^2(\mathbb{Z})$ with $a[n] = f(n)$, and $\widehat{a} \in \mathbf{L}^2[-\pi, \pi]$ a 2π periodic Fourier series in which

$$\widehat{a}(\omega) = \sum_{n \in \mathbb{Z}} \widehat{f}(\omega - 2\pi n)$$

- Discrete, finite samplings $x \in \mathbb{C}^N$ which must be considered as N-periodic to do any frequency calculation (e.g., Theorem 3.7). In particular, if

$$x[n] = \sum_{p \in \mathbb{Z}} a[n - pN]$$

then $\widehat{x} \in \mathbb{C}^N$ with

$$\widehat{x}[k] = \widehat{a}(2\pi k/N) \qquad \text{(see Exercise 22)}$$

Thus a discrete, but infinite sampling of f in time/space periodizes its Fourier transform, possibly leading to aliasing. A finite, discrete sampling in frequency also periodizes the signal in time/space, leading to possible border effects. We must account for both of these in practice.

The circular convolution theorem will be very important for opening up fast algorithms for computing $x \circledast y$. This will be made possible by the *Fast Fourier Transform (FFT)*, which we now describe. To motivate the algorithm, recall the DFT:

$$\widehat{x}[k] = \sum_{n=0}^{N-1} x[n] \exp\left(-\frac{2\pi i k n}{N}\right), \quad 0 \le k < N$$

and observe that it requires N^2 (complex) multiplications and additions (N for each k). The FFT algorithm reduces this to $O(N \log_2 N)$.

The FFT algorithm works through a divide and conquer approach; in these notes I will describe the radix-2 decimation in time (DIT) algorithm. This is a recursive algorithm.

Given x we divide the DFT summation into two sums, one for the even indices of x and one for the odd indices of x:

$$\begin{aligned}\widehat{x}[k] &= \sum_{n=0}^{N/2-1} x[2n] \exp\left(\frac{-2\pi i k(2n)}{N}\right) + \sum_{n=0}^{N/2-1} x[2n+1] \exp\left(\frac{-2\pi i k(2n+1)}{N}\right) \\ &= \sum_{n=0}^{N/2-1} x[2n] \exp\left(\frac{-2\pi i k n}{N/2}\right) + e^{-2\pi i k/N} \sum_{n=0}^{N/2-1} x[2n+1] \exp\left(\frac{-2\pi i k n}{N/2}\right)\end{aligned}$$

The second line looks like the sum of two DFTs of length $N/2$ signals. Indeed, define $x_{\mathrm{e}}, x_{\mathrm{o}} \in \mathbb{R}^{N/2}$ as:

$$\begin{aligned}x_{\mathrm{e}}[n] &= x[2n], \quad 0 \le n < N/2 \\ x_{\mathrm{o}}[n] &= x[2n+1], \quad 0 \le n < N/2\end{aligned}$$

and notice that we have

$$\begin{aligned}\widehat{x}_{\mathrm{e}}[k] &= \sum_{n=0}^{N/2-1} x[2n] \exp\left(\frac{-2\pi i k n}{N/2}\right), \quad 0 \le k < N/2 \\ \widehat{x}_{\mathrm{o}}[k] &= \sum_{n=0}^{N/2-1} x[2n+1] \exp\left(\frac{-2\pi i k n}{N/2}\right), \quad 0 \le k < N/2\end{aligned}$$

This allows us to recover $\widehat{x}[k]$ for $0 \le k < N/2$ as:

$$\widehat{x}[k] = \widehat{x}_{\mathrm{e}}[k] + e^{-2\pi i k/N} \widehat{x}_{\mathrm{o}}[k], \quad 0 \le k < N/2 \tag{15}$$

For the frequencies $N/2 \le k < N$, we use the fact that the DFT is periodic and observe that

$$\widehat{x}_{\mathrm{e}}[k + N/2] = \widehat{x}_{\mathrm{e}}[k] \quad \text{and} \quad \widehat{x}_{\mathrm{o}}[k + N/2] = \widehat{x}_{\mathrm{o}}[k]$$

We thus obtain:

$$\begin{aligned}0 \le k < N/2, \quad \widehat{x}[k + N/2] &= \widehat{x}_{\mathrm{e}}[k] + e^{-2\pi i (k+N/2)/N} \widehat{x}_{\mathrm{o}}[k] \\ &= \widehat{x}_{\mathrm{e}}[k] - e^{-2\pi i k/N} \widehat{x}_{\mathrm{o}}[k]\end{aligned} \tag{16}$$

Putting together (15) and (16) we obtain $\widehat{x}[k]$ for all $0 \le k < N$. Notice that already we have reduced computations. Indeed, the one step algorithm proceeds by first dividing x into even and odd indices signals, computing the two length $N/2$ DFTs, and recombining as above. The two length N DFTs cost $2(N/2)^2 = N^2/4$ multiplications and additions, and the combination costs N additions and multiplications. Thus we have replaced N^2 complex multiplications and additions with $N^2/2 + N$ complex multiplications and additions, which is already better for $N \ge 3$.

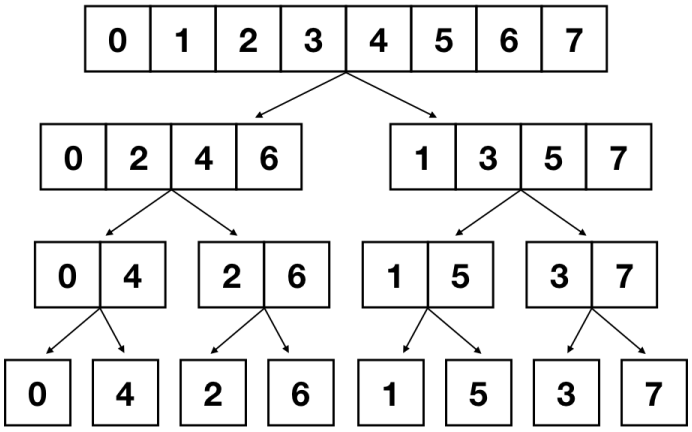

Figure 6: Recursive subdivision scheme of the FFT algorithm for $N = 8$.

Now for simplicity, suppose $N = 2^p$. The $O(N \log_2 N)$ FFT algorithm is obtained by recursively subdividing the original signal x, according to the same procedure as outlined above into "even" and "odd" components, until we have N signals of length one; Figure 6 illustrates the idea for $N = 8$. The algorithm then "computes" N length one DFTs - notice that these just return the value of each length one signal, so no computation is actually performed. The algorithm then forms $N/2$ length two DFTs the next level up by combining the pairs of length one DFTs that have the same parent signal, and multiplying the "odd" length one signal by the appropriate complex value before combination. At each level we incur a cost of $O(N)$, and there are $p = \log_2 N$ levels; thus the total cost of the algorithm is $O(N \log_2 N)$.

The FFT algorithm is remarkable for turning an $O(N^2)$ calculation into an $O(N \log N)$ calculation with no loss of accuracy. For this reason it is a pillar of digital signal processing. However, it is fundamentally an algebraic property of the DFT, based on symmetries. As such, the algorithm is "fragile," and in particular, if you do not uniformly sample your signal f, you cannot apply the FFT algorithm. That however is a discussion for another day (or class).

The FFT algorithm allows us to compute convolutions $x * y$ fast. Suppose the $x, y \in \mathbb{C}^N$ as usual; if we compute $x * y$ directly it will cost us $O(N^2)$ calculations. In order to calculate the non-circular convolution faster, we can use the circular convolution Theorem 3.7, which will allow us to leverage the FFT algorithm. The main idea is that instead of computing $x * y$ directly, we compute $\widehat{x}$ and $\widehat{y}$, each costing $O(N \log N)$ calculations, then we compute the multiplication $\widehat{x}[k]\widehat{y}[k]$ for $0 \leq k < N$, costing $O(N)$ calculations, and then we compute the inverse Fourier transform of $(\widehat{x}[k]\widehat{y}[k])_{0 \leq k < N}$ with another FFT, which costs $O(N \log N)$ calculations; the total run time of the algorithm if $O(N \log N)$, which in practice (depending

upon the exact FFT algorithm you use) will be better for $N \geq 32$. One thing that we have not addressed though, is that the convolution theorem for finite length signal applies to circular convolution. If we do not account for this, we will run into border effects since we will be computing $x \circledast y$ instead of $x * y$. To fix this issue, we zero pad x and y by defining:

$$x_0[n] = \begin{cases} x[n] & 0 \leq n < N \\ 0 & N \leq n < 2N \end{cases}$$

The signal $x_0 \in \mathbb{C}^{2N}$ and is just the signal x but with N zeros appended to the back of it. One can then verify that:

$$x_0 \circledast y_0[n] = x * y[n], \quad 0 \leq n < 2N$$

Thus we apply the fast FFT based algorithm to x_0 and y_0 (rather than x and y) to obtain $x * y$ in $O(N \log N)$ time.

Exercise 19. Read Section 3.3 of *A Wavelet Tour of Signal Processing.*

Exercise 20. Read Section 3.4 of *A Wavelet Tour of Signal Processing.*

Exercise 21. Let $\widehat{x}[k]$ be the DFT of a finite signale $x \in \mathbb{C}^N$. Define a signal $y \in \mathbb{C}^{2N}$ by:

$$\widehat{y}[N/2] = \widehat{y}[3N/2] = \widehat{x}[N/2]$$

and

$$\widehat{y}[k] = \begin{cases} 2\widehat{x}[k] & 0 \leq k < N/2 \\ 0 & N/2 < k < 3N/2 \\ 2\widehat{x}[k-N] & 3N/2 < k < 2N \end{cases}$$

Prove that y is an interpolation of x that satisfies $y[2n] = x[n]$ for all $0 \leq n < N$.

Exercise 22. We want to compute numerically the Fourier transform of $f(t)$. Let $a[n] = f(n)$ for $n \in \mathbb{Z}$ be the countably infinite discrete sampling of f and let $x \in \mathbb{C}^N$ be the periodization of a over the period of length N:

$$x[n] = \sum_{p \in \mathbb{Z}} a[n - pN]$$

(a) Prove that the DFT of x is related to the Fourier series of a and to the Fourier transform of f by the following formula:

$$\widehat{x}[k] = \widehat{a}(2\pi k/N) = \sum_{\ell \in \mathbb{Z}} \widehat{f}\left(\frac{2\pi k}{N} - 2\pi\ell\right)$$

(b) Suppose that $|f(t)|$ and $|\widehat{f}(\omega)|$ are negligible when $t \notin [-t_0, t_0]$ and $\omega \notin [-\omega_0, \omega_0]$. Relate N to t_0 and ω_0 so that one can compute an approximate value of $\widehat{f}(\omega)$ for all $\omega \in \mathbb{R}$ by interpolating the samples $\widehat{x} \in \mathbb{C}^N$. It is possible to compute exactly $\widehat{f}(\omega)$ with such an interpolation formula?

Exercise 23. We are going to implement the FFT and fast convolution algorithms:

(a) Implement the DFT algorithm (programming language of your choice). Record the runtime for many values of N, and plot it as a function of N. Do you see the quadratic scaling? Turn in your code and plot(s).

(b) Make precise the $O(N \log N)$ FFT algorithm described above and implement it on your own for $N = 2^p$. Test the algorithm for accuracy by comparing its outputs to the outputs of your DFT algorithm. Test the algorithm for speed by comparing the runtime for numerous values of N to the runtimes your recorded for the DFT. For which value of N does your FFT algorithm become faster? Turn in your code, at least one output showing that the DFT and FFT codes produce the same results, and a plot of the FFT runtimes as a function of N (you can combine this plot with the DFT plot).

(c) Using either your own FFT and inverse FFT code, or built in code (in Matlab or Python, for example) since you are not required to write your own inverse FFT code, implement an algorithm to compute $x*y$ (for $x, y \in \mathbb{C}^N$) in $O(N \log N)$ time. Verify the accuracy by comparing against convolution code that computes $x * y$ directly (either your own code, or built in code), and compare the runtimes. For which N is your $O(N \log N)$ convolution code faster?

References

[1] Stéphane Mallat. *A Wavelet Tour of Signal Processing, Third Edition: The Sparse Way*. Academic Press, 3rd edition, 2008.

[2] Elias M. Stein and Rami Shakarchi. *Fourier Analysis: An Introduction*. Princeton Lectures in Analysis. Princeton University Press, 2003.

[3] John J. Benedetto and Matthew Dellatorre. Uncertainty principles and weighted norm inequalities. *Contemporary Mathematics*, 693:55–78, 2017.

[4] Yves Meyer. *Wavelets and Operators*, volume 1. Cambridge University Press, 1993.

[5] Karlheinz Gröchenig. *Foundations of Time Frequency Analysis*. Springer Birkhäuser, 2001.