MIT 6.893: Algorithms and Signal Processing Fall 2014

Lecture 4 — September 17, 2014

Prof. Piotr Indyk *Scribe: Chinmay Hegde*

In this lecture, we take some initial steps towards obtaining a $O(k \log n)$ sparse Fourier transform algorithm that works even for worst case inputs.

1 Recap

In the previous lecture, we discussed a sparse FFT algorithm that ran in $O(k \log n)$ time but was accompanied by a few caveats. The algorithm was based on the idea that *aliasing* the frequency spectrum could be efficiently achieved by computing the Fourier transform of a subsampling of the input signal. This aliasing could be achieved by binning the frequency indices using a pair of co-prime aliasing filters. If the nonzero Fourier coefficients are assumed to occur in random locations, then they are likely to be isolated by (at least one of) the filtering steps, and therefore can be estimated correctly and subtracted.

The big issue with this approach is that this only constitutes an average case analysis since it makes a randomness assumption about the input signal. There are other issues: this approach only works for $k < O(\sqrt{n})$, and does not immediately generalize to the case where the signal is not exactly k-sparse.

2 Pseudo-random spectrum permutation

One way to get around the random input assumption is to instead introduce randomness in the *algorithm* in order to somehow isolate the coefficients. This is achieved via the method of *spectrum permutation*. The high level idea is that a (pseudo-)random permuation in the time domain induces a (pseudo-)random permutation in the frequency domain, which in turn results in the isolation of large frequency coefficients. We make use of the following result [1, 2]:

Lemma 1. *Let σ be invertible modulo n, and $\beta \in [n]$. Define $a'_j = a_{\sigma j}\omega^{-\beta j}$. Then, for any u, $\hat{a}'_u = \hat{a}_{\sigma^{-1}(u+\beta)}$.*

Proof. Let $j' = \sigma j$. For any u,

$$\hat{a}'_u = \sum_{j=0}^{n-1} a_{\sigma j}\omega^{-\beta j}\omega^{-uj} = \sum_{j=0}^{n-1} a_{\sigma j}\omega^{-(u+\beta)j} = \sum_{j'=0}^{n-1} a_{j'}\omega^{-(u+\beta)\sigma^{-1}j'} = a_{\sigma^{-1}(u+\beta)}.$$

□

Using this Lemma, we can *simulate* the randomness assumption on the input by choosing β uniformly at random from $\{0, \ldots, n-1\}$ and σ uniformly at random from the invertible numbers in $\{0, \ldots, n-1\}$. Two important special cases arise when n is a prime, and when n is a power of 2. If n is prime, then any σ and β work (and in fact, this scheme is often used for constructing pairwise independent hash functions.) If n is a power of 2, then any odd σ works. We can prove the following simple result when n is a power of 2.

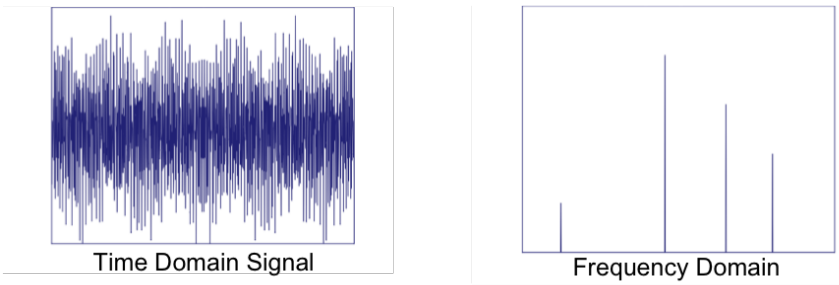

Figure 1: Time and frequency-domain representations of a signal $\mathbf{a}$.

Lemma 2. *Let n be a power of 2, σ is a random uniformly-drawn odd number from $[n]$ and $j \neq 0$, then $Pr[\sigma j \in [-C, C]] \leq O(C/n)$.*

Proof. Let $j = d2^b$ for some odd d. Then, the orbit of σj (or, set of all possible potential values of σj) consists of integers of the form $d'2^b$ for all odd d'. There are at most $O(n/2^b)$ such values; moreover, the number of values within $[-C, C]$ is smaller than $2 \cdot \text{round}(C/2^b)$. Since σ is uniformly chosen, the chance that σj will lie within $[-C, C]$ is at most $O(C/n)$. □

However, there are issues in both these special cases. In the case where n is a prime, we can simulate the randomness in the nonzero locations, but the subsequent aliasing would not be possible (since we cannot evenly divide $[n]$ into a given number of buckets). In the case where n is a power of 2 (say, $n = 2^t$) then aliasing would be possible since we could choose some 2^r, $r < t$ as the number of buckets. However, we have the (unfortunate) property that

$$\begin{aligned} \sigma^{-1}(u + \beta) \equiv \sigma^{-1}(v + \beta) \mod 2^r, \quad \text{iff} \\ u \equiv v \mod 2^r. \end{aligned}$$

In other words, there is no benefit to be gained from this type of random binning! We need to think of an alternative binning approach.

3 Filtering and aliasing

We will adopt a slightly different approach for aliasing the Fourier coefficients. A basic property of the Fourier transform is that *multiplication* of two signals in one domain (either time or frequency) corresponds to *convolution* in the other domain, i.e., for length-n signals $\mathbf{a}$ and $\mathbf{g}$:

$$\begin{aligned} \widehat{\mathbf{a} \times \mathbf{g}} = \hat{\mathbf{a}} * \hat{\mathbf{g}} \quad \text{where} \\ (\hat{\mathbf{a}} * \hat{\mathbf{g}})_j = \sum_t \hat{a}_t \hat{g}_{j-t}. \end{aligned}$$

This property suggests the following scheme. Consider filtering (multiplying) the time domain signal $\mathbf{a}$ using an appropriate function $\mathbf{g}$ that is localized in both time and frequency. In the frequency domain, this corresponds to convolving the frequency domain representations $\hat{\mathbf{a}}$ and $\hat{\mathbf{g}}$. Now, aliasing the *time* domain signal into buckets of size B is equivalent to subsampling the frequency domain (or in other words, computing the B-point DFT). If the filter $\mathbf{g}$ has a small support, then the aliasing step simply consists of keeping the first B coefficients of $\mathbf{g} \times \mathbf{a}$. We could choose the filter $\mathbf{g}$ in different ways:

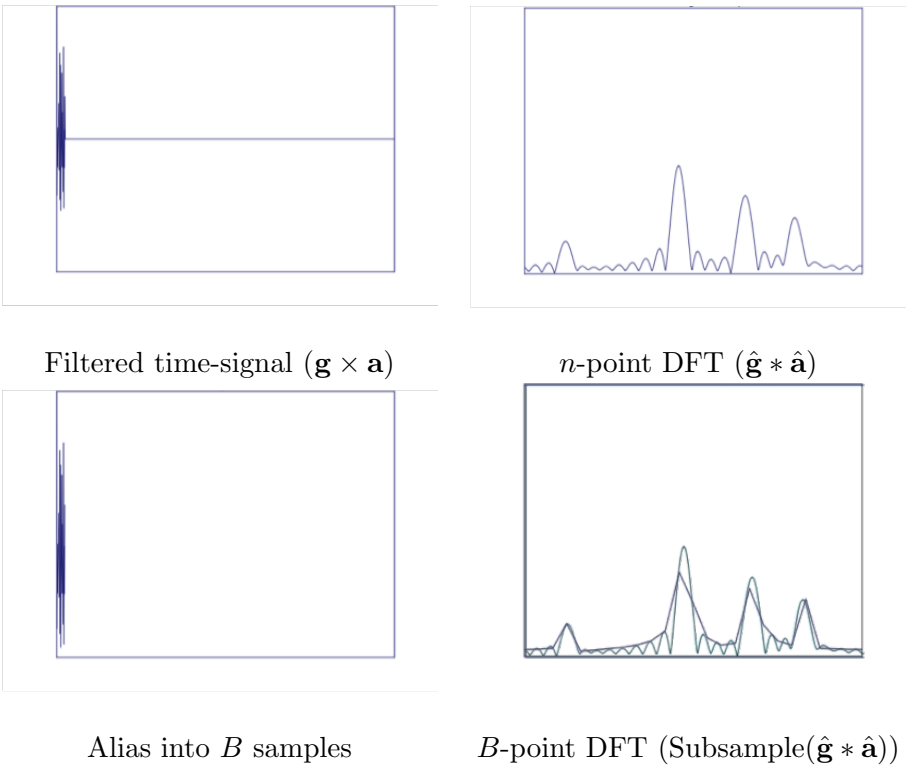

Figure 2: Illustration of the new aliasing approach. Here, $\mathbf{a}$ is the signal; $\mathbf{g}$ is a rectangular (boxcar) filter; $\times$ represents element-wise multiplication and $*$ represents convolution.

1. *Box-car:* This corresponds the simple filter $\mathbf{g}$ with its first B coefficients equal to 1 and all other coefficients equal to zero. From a time-domain perspective, this filter is trivial to implement since it only requires retaining the first few samples of $\mathbf{a}$. However, this filter doesn't localize in the frequency domain very well; a straightforward derivation shows that the magnitude of $\mathbf{g}$ is the *sinc* function; the magnitude of this function is $O(|\frac{\sin \pi j}{\pi j}|)$, which decays as $1/j$ and this is insufficient for our purposes. Moreover, due to the convolution property, a bin containing a large Fourier coefficient can "interfere" with several other bins. Despite these limitations, such filters can and have been used for sparse Fourier applications [1, 2].

2. *Truncated Gaussian:* It is well-known that the (continuous) Gaussian function is an *eigenfunction* of the continuous Fourier transform. Therefore, a discretization, followed by a periodic summation, of the Gaussian function yields an eigenvector of the discrete Fourier transform:

$$a_j = \sum_{k \in \mathbb{Z}} \exp\left(-\frac{\pi \cdot (j + nk)^2}{n}\right),$$

   A closed form expression for this series is not known, but it converges very rapidly and can be numerically approximated by something having a small support (i.e., by truncation). This provides a filter that is perfectly localized and also decays exponentially, but still leaks into $O(\sqrt{\log n})$ adjacent bins.

3. *Sinc* $\times$ *Gaussian*: On the other hand, we could consider convolving the Gaussian with a box-car in the frequency doman, which corresponds to multiplying a Gaussian with a sinc

function in the time domain. This mitigates some of the effects induced by the slow decay of the sinc function, while still keeping the filter (fairly) localized in the frequency domain. This is what we will end up using.

4 Conditions required for "good" filter functions

We define a formal set of conditions on the filter function $\mathbf{g}$ that is needed for our method to work. First, we define the notion of a *standard window function*, and show that such functions are easy to construct. The definitions and implementations are from [3].

Definition 3. *An* (ε, δ, w) standard window function *is a symmetric vector* $\mathbf{g} \in \mathbb{R}^n$ *with supp*$(\mathbf{g}) \subseteq [-w/2, w/2]$ *such that* $\hat{g}_0 = 1$, $\hat{g}_j > 0$ *for all* $j \in [-\varepsilon n, \varepsilon n]$, *and* $|\hat{g}_j| < \delta$ *for all* $j \notin [-\varepsilon n, \varepsilon n]$.

Lemma 4. *For any* ε *and* δ, *there exists an* $(\varepsilon, \delta, O(\frac{1}{\varepsilon}\log(1/\delta)))$ *standard window function.*

Proof. Take a (suitably normalized) Gaussian with standard deviation $w' = \Theta(\sqrt{\log(1/\delta)}/\varepsilon)$ and truncate it at $w = O(\frac{1}{\varepsilon}\log(1/\delta))$ A discrete version of the classical *uncertainty principle* implies that the standard deviation is $n/w' = \Theta(n\varepsilon/\sqrt{\log(1/\delta)})$. The result follows from plugging in this value of standard deviation into the expression for the Gaussian. □

We want something slightly different. Ideally, we would use a filter that is close to a box-car filter (i.e., flat and close to 1 around the origin) in the frequency domain with an exponential fall-off. More precisely, we need a *flat window function*, defined as follows.

Definition 5. *An* $(\varepsilon, \varepsilon', \delta, w)$ flat window function *is a symmetric vector* $\mathbf{g} \in \mathbb{R}^n$ *with supp*$(\mathbf{g}) \subseteq [-w/2, w/2]$ *such that* $\hat{g}_j \in [1-\delta, 1+\delta]$ *for all* $j \in [-\varepsilon' n, \varepsilon' n]$ *and* $|\hat{g}_j| < \delta$ *for all* $j \notin [-\varepsilon n, \varepsilon n]$.

In our method, we will use $\varepsilon = 2\varepsilon'$ and $\delta < 1/n^{O(1)}$. A flat window function can be constructed from any standard window function by convolving it with a box-car window function. We have the following result.

Lemma 6. *For any* $\varepsilon, \varepsilon'$ *with* $\varepsilon > \varepsilon'$, *there exists an* $(\varepsilon, \varepsilon', \delta, O(\frac{1}{\varepsilon - \varepsilon'}\log(1/\delta)))$ *flat window function.*

Proof. Let $\alpha = (\varepsilon - \varepsilon')/2$. Suppose that $\mathbf{g}$ is an $(\alpha, \frac{\delta}{n(\varepsilon+\varepsilon')}, w)$ standard window function with $w = O(\log(n/\delta)/\alpha)$. We can safely assume that $\varepsilon, \varepsilon' > 1/(2n)$ since otherwise $[-\varepsilon n, \varepsilon n] = \{0\}$. Therefore, $\log \frac{(\varepsilon+\varepsilon')n}{\delta} = O(\log n/\delta)$.

Define $\mathbf{g}'$ as the sum of $1 + 2n(\varepsilon' + \alpha)$ shifted copies of $\hat{\mathbf{g}}$, normalized to have $\hat{g}'_0 \approx 1$:

$$\hat{g}'_u = \frac{\sum_{j=-(\varepsilon'+\alpha)n}^{(\varepsilon'+\alpha)n} \hat{g}_{u+j}}{\sum_{j=-\alpha n}^{\alpha n} \hat{g}_j}.$$

The width of supp(g') is the same as the width of supp(g) since shifting in the frequency domain does not change the support in the time domain.

Let N be the numerator and D be the denominator of $\hat{g}'_u$. Since $\hat{g}$ is positive and $\hat{g}_0 = 1$, the value of D is at least 1. For $u \in [-\varepsilon' n, \varepsilon' n]$, N consists of the $2\alpha n$ terms in D, plus $2\varepsilon' n$ additional terms that are no greater than $\delta/(n(\varepsilon + \varepsilon'))$ each. Therefore, $|N - 1| < \delta$ for small u. For large u, i.e., $u \notin [-\varepsilon n, \varepsilon n]$, a similar argument implies that $|N| < \delta$, while $D > 1$. Therefore, $\hat{\mathbf{g}}'$ is an $(\varepsilon, \varepsilon', \delta, w)$ window function with the stated parameters.

□

5 Putting things together

The above ingredients suffice to develop an algorithm that is required to establish Theorem 7. The high level idea is to:

1. Permute the indices of the input
2. Apply a flat window function and alias into bins.
3. Estimate the index and value of the nonzero coefficients in each bin.
4. Repeat.

The first three steps can be achieved in time proportional to $O(k \log n)$. Some care needs to be taken in the last step, and there are lots of parameters to be chosen carefully. We will outline these more precisely in the next lecture. In particular, we will show the following result, which will serve as a basic building block for future algorithms.

Theorem 7. *Assume that n is a power of 2, and $\hat{\mathbf{a}}$ contains at most k nonzero coefficients. Also assume that the coefficients of $\hat{a}$ can be specified with polynomial precision, i.e., $\hat{a}_u \in \{0, 1, \ldots, n^{O(1)}\}$ for $u \in [n]$. Then, there exists an algorithm that, given an input signal $\mathbf{a}$, runs in time $O(k \log n)$ and outputs at most k Fourier coefficients, and each coefficient is correct with constant probability.*

References

[1] A. Gilbert, S. Guha, P. Indyk, M. Muthukrishnan, and M. Strauss. Near-optimal sparse Fourier representations via sampling. *STOC*, 2002.

[2] A. Gilbert, S. Muthukrishnan and M. Strauss. Improved time bounds for near-optimal sparse Fourier representations. *Optics and Photonics. International Society for Optics and Photonics.*, 2005.

[3] H. Hassanieh, P. Indyk, D. Katabi, and E. Price, "Simple and Practical Algorithm for Sparse Fourier Transform", *SODA*, 2012.

[4] H. Hassanieh, P. Indyk, D. Katabi, and E. Price. Near-optimal algorithm for sparse Fourier transform. *STOC*, 2012.