A sparse Fast Fourier Algorithm for Real Nonnegative Vectors

Gerlind Plonka[*]   Katrin Wannenwetsch[†]

February 17, 2016

Abstract

In this paper we propose a new fast Fourier transform to recover a real nonnegative signal $\mathbf{x} \in \mathbb{R}_+^N$ from its discrete Fourier transform $\widehat{\mathbf{x}} = \mathbf{F}_N \mathbf{x} \in \mathbb{C}^N$. If the signal $\mathbf{x}$ appears to have a short support, i.e., vanishes outside a support interval of length $m < N$, then the algorithm has an arithmetical complexity of only $\mathcal{O}(m \log m \log(N/m))$ and requires $\mathcal{O}(m \log(N/m))$ Fourier samples for this computation. In contrast to other approaches there is no a priori knowledge needed about sparsity or support bounds for the vector $\mathbf{x}$. The algorithm automatically recognizes and exploits a possible short support of the vector and falls back to a usual radix-2 FFT algorithm if $\mathbf{x}$ has (almost) full support. The numerical stability of the proposed algorithm ist shown by numerical examples.

Key words. discrete Fourier transform, sparse Fourier reconstruction, sublinear sparse FFT

AMS Subject classifications. 65T50, 42A38

1   Introduction

Algorithms for fast Fourier transform play a fundamental role in many areas in numerical analysis, particularly in signal and image processing. It is well-known that FFT algorithms for general vectors $\mathbf{x} \in \mathbb{C}^N$ require $\mathcal{O}(N \log N)$ arithmetical operations and that this qualitative bound cannot be improved, see [11]. However, if the discrete Fourier transform is applied to recover vectors with special properties, there is the hope for even faster algorithms.

In recent years, there has been some effort to derive new so-called "sparse FFT" algorithms that exploit the a priori knowledge that the vector to be recovered is sparse or only has a small amount of significant frequencies. Often, further assumptions on the vector appear, as e.g. that the components to be recovered are from a certain quantized range consisting of a finite set of real entries, see e.g. [5]. Most

---

[*]University of Göttingen, Institute for Numerical and Applied Mathematics, Lotzestr. 16-18, 37083 Göttingen, Germany. Email: plonka@math.uni-goettingen.de

[†]University of Göttingen, Institute for Numerical and Applied Mathematics, Lotzestr. 16-18, 37083 Göttingen, Germany. Email: k.wannenwetsch@math.uni-goettingen.de

of the proposed algorithms are based on randomization [4, 5, 6, 12] and achieve e.g. a complexity of $\mathcal{O}(k \log N)$ [5] for k-sparse signals, or even $\mathcal{O}(k \log k)$, see e.g. [12]. An obvious drawback of randomized approaches is that the algorithms do not always achieve the correct result (or an approximation of it) but only with a certain probability. Another problem is that there exists no sublinear algorithm to check the correctness of the result.

Completely deterministic sparse FFT algorithms for k-sparse signals have been proposed e.g. in [1, 2, 8, 9, 10]. The underlying ideas are based on combinatorial approaches employing FFTs of different prime length and the Chinese remainder theorem. These algorithms usually have polynomial costs in $\log N$ and k and only pay off for very large N and strong sparsity.

Sparse FFT algorithms based on Prony's method, see [7, 13, 15], for k-sparse signals are usually based on singular value decompositions of structured matrices of size k with a complexity of $\mathcal{O}(k^3)$ and are therefore only efficient for small k. This complexity can be reduced in special cases using a splitting approach [15].

In a recent paper [14], the authors proposed a deterministic sparse FFT algorithm for vectors with short support that is based on usual FFT and is numerically stable with a complexity of $\mathcal{O}(m \log N)$ operations, where here m denotes the support length of the signal.

All algorithms mentioned above require a priori information as e.g. the exact sparsity or the support length of the vector to be recovered, or at least a suitable upper bound for it. They are just not applicable without this information or do not always achieve the correct recovery result by using only a guess for this bound.

However, in practice, while certain structures of the vector (as short support or sparsity) often appear, we do not always have the a priori knowledge on a good upper bound. Therefore it is of high interest to develop deterministic FFT algorithms that are able to automatically recognize certain structures of the vector during the algorithm and to exploit it suitably to reduce complexity and run time.

In this paper, we firstly propose an algorithm that meets this requirement in the way that no a priori knowledge about the support length of the vector to be recovered is needed beforehand. We present a new deterministic algorithm to recover a real non-negative vector $\mathbf{x} \in \mathbb{R}_+^N$ from its discrete Fourier transform $\widehat{\mathbf{x}}$. If $\mathbf{x}$ has a support with support length m being significantly smaller than N, then the algorithms automatically recognizes this structure and provides the resulting vector with an arithmetical complexity of $\mathcal{O}(m \log m \, \log \frac{N}{m})$ requiring at most $\mathcal{O}(m \log(N/m))$ Fourier data. The idea of the algorithm is based on divide-and-conquer techniques.

Direct applications for the reconstruction of sparse vectors from Fourier data, both with known or unknown support, appear for instance solving phase retrieval problems, where short support and positivity of the resulting vectors or images is a frequently used precondition in iterated projection algorithms, see e.g. [3].

This paper is structured as follows: After fixing the notations, we introduce the new fast algorithm in Section 2, together with detailed explanations of its structure and complexity. In Section 3, we apply the algorithm to noisy Fourier data and present some numerical results showing the numerical stability of the proposed algorithm in practice.

2

1.1 Notations

Let $\mathbf{x} \in \mathbb{R}_+^N$ with $N = 2^J$ for some $J > 0$ be a real vector with nonnegative entries. We denote the discrete Fourier transform $\widehat{\mathbf{x}}$ of $\mathbf{x}$ by

$$\widehat{\mathbf{x}} = \mathbf{F}_N \mathbf{x},$$

where $\mathbf{F}_N := \left(\omega_N^{jk} \right)_{j,k=0}^{N-1} \in \mathbb{C}^{N \times N}$ is the Fourier matrix and $\omega_N := \mathrm{e}^{\frac{-2\pi \mathrm{i}}{N}}$.

The *support length* $m = |\mathrm{supp}\,\mathbf{x}|$ of $\mathbf{x} \in \mathbb{R}_+^N$ is defined as the minimal positive integer such that $x_k = 0$ for $k \notin I := \{(\mu + \ell) \mathrm{mod}\, N \mid \ell = 0, \ldots, m-1\}$. We call this index set I the *support interval of* $\mathbf{x}$. The *first support index of* $\mathbf{x}$ is denoted by μ. Note that the first support index of $\mathbf{x}$ needs not to be the index of the first nonzero entry in $\mathbf{x}$. Considering for example the vector $\mathbf{x} := (1, 2, 0, 0, 0, 5, 3, 0) \in \mathbb{R}_+^8$, we obtain a support length $m = 5$, the index set $I = \{5, 6, 7, 0, 1\}$ and $\mu = 5$. Note further that the support interval I may contain indices corresponding to zero components of $\mathbf{x}$, as e.g. the index 7 in the small example above. Therefore the support length m is an upper bound of the sparsity, the number of nonzero entries of $\mathbf{x}$. But in any case, it holds that $x_\mu \neq 0$ and $x_{(\mu+m-1) \mathrm{mod}\, N} \neq 0$.

The support length of a vector $\mathbf{x}$ is always uniquely defined. However, the support interval and the first support index μ are not necessarily unique. Consider e.g. the vector $\mathbf{x} \in \mathbb{R}_+^N$ with $x_0 = x_{N/2} = 1$ and $x_\ell = 0$ for $\ell \in \{0, \ldots, N-1\} \backslash \{0, N/2\}$. For this vector it is possible to choose either $\mu = 0$ or $\mu = N/2$ whereas the support length is $N/2 + 1$ in both cases.

The *periodized vectors* $\mathbf{x}^{(j)} \in \mathbb{R}_+^{2^j}$ of $\mathbf{x}$ are defined by

$$\mathbf{x}^{(j)} = (x_k^{(j)})_{k=0}^{2^j - 1} := \left(\sum_{\ell=0}^{2^{J-j}-1} x_{k+2^j \ell} \right)_{k=0}^{2^j - 1} \tag{1.1}$$

for $j = 0, \ldots, J$. In particular, $\mathbf{x}^{(0)} = \sum_{k=0}^{N-1} x_k$ is the sum of all components of $\mathbf{x}$, $\mathbf{x}^{(1)} = \left(\sum_{k=0}^{N/2-1} x_{2k}, \sum_{k=0}^{N/2-1} x_{2k+1} \right)^T$ and $\mathbf{x}^{(J)} = \mathbf{x}$.

The next lemma that has been proved already in [14], shows that components of the discrete Fourier transforms $\widehat{\mathbf{x}}^{(j)}$ need not to be computed but are already given by components of $\widehat{\mathbf{x}} = \mathbf{F}_N \mathbf{x}$.

Lemma 1.1 *For the vectors* $\mathbf{x}^{(j)} \in \mathbb{R}_+^{2^j}$, $j = 0, \ldots, J$, *in (1.1), we have the discrete Fourier transform*

$$\widehat{\mathbf{x}}^{(j)} := \mathbf{F}_{2^j} \mathbf{x}^{(j)} = (\widehat{x}_{2^{J-j}k})_{k=0}^{2^j - 1},$$

where $\widehat{\mathbf{x}} = (\widehat{\mathbf{x}}_k)_{k=0}^{N-1} = \mathbf{F}_N \mathbf{x}$ *is the Fourier transform of* $\mathbf{x} \in \mathbb{R}_+^N$.

2 Sparse FFT algorithm for nonnegative vectors

Let us assume that the Fourier transform $\widehat{\mathbf{x}} = \mathbf{F}_N \mathbf{x}$ of $\mathbf{x} \in \mathbb{R}^N$ is given and that $\mathbf{x}$ has only nonnegative entries. We want to derive an algorithm that automatically recognizes a possible shorter support of $\mathbf{x}$ and applies in this case a faster reconstruction

algorithm while falling back to the usual FFT with $\mathcal{O}(N \log_2 N)$ complexity if $\mathbf{x}$ has full support or does not possess a support with a length being significantly smaller than N.

The main idea to reconstruct $\mathbf{x}$ from $\widehat{\mathbf{x}}$ is to employ the divide-and-conquer technique similarly as for usual radix-2 FFT. Starting with $\mathbf{x}^{(0)} = \widehat{x}_0 \in \mathbb{R}_+$ we recover $\mathbf{x}$ iteratively by reconstructing $\mathbf{x}^{(j+1)}$ from $\mathbf{x}^{(j)}$ for $j = 0, \ldots, J-1$. At each level, we check the support length $m_j := |\operatorname{supp} \mathbf{x}^{(j)}|$ of $\mathbf{x}^{(j)}$ and distinguish two cases: $m_j > 2^{j-1}$ and $m_j \leq 2^{j-1}$. In the first case, i.e., if the support length of $\mathbf{x}^{(j)}$ is greater than half of the vector length, we cannot benefit from a short support and therefore compute $\mathbf{x}^{(j+1)}$ using $\mathbf{x}^{(j)}$ and employing an FFT algorithm of length 2^j. In the second case, if the support of $\mathbf{x}^{(j)}$ is shorter than half of the vector length, we apply a modified reconstruction algorithm that only requires $\mathcal{O}(m_j \log m_j)$ floating point operations.

In both cases, we aim at computing $\mathbf{x}^{(j+1)}$ by a simplified inverse fast Fourier transform that exploits the values of $\mathbf{x}^{(j)}$ being known already from the previous iteration step.

Splitting the vector $\mathbf{x}^{(j+1)}$ into the two partial vectors $\mathbf{x}_0^{(j+1)} = (x_k^{(j+1)})_{k=0}^{2^j-1}$ and $\mathbf{x}_1^{(j+1)} = (x_k^{(j+1)})_{k=2^j}^{2^{j+1}-1}$ of length 2^j, we recall that by definition

$$\mathbf{x}^{(j)} = \mathbf{x}_0^{(j+1)} + \mathbf{x}_1^{(j+1)}. \tag{2.1}$$

Using Lemma 1.1 we observe the following relation for $\widehat{\mathbf{x}}^{(j+1)}$,

$$\widehat{\mathbf{x}}^{(j+1)} = (\widehat{x}_{2^{J-j-1}k})_{k=0}^{2^{j+1}-1} = \mathbf{F}_{2^{j+1}} \mathbf{x}^{(j+1)} = (\omega_{2^{j+1}}^{k\ell})_{k,\ell=0}^{2^{j+1}-1} \begin{pmatrix} \mathbf{x}_0^{(j+1)} \\ \mathbf{x}_1^{(j+1)} \end{pmatrix}$$

$$= (\omega_{2^{j+1}}^{k\ell})_{k,\ell=0}^{2^{j+1}-1,2^j-1} \mathbf{x}_0^{(j+1)} + (\omega_{2^{j+1}}^{k\ell})_{k=0,\ell=2^j}^{2^{j+1}-1,2^{j+1}-1} (\mathbf{x}^{(j)} - \mathbf{x}_0^{(j+1)})$$

$$= (\omega_{2^{j+1}}^{k\ell})_{k,\ell=0}^{2^{j+1}-1,2^j-1} \mathbf{x}_0^{(j+1)} + ((-1)^k \omega_{2^{j+1}}^{k\ell})_{k,\ell=0}^{2^{j+1}-1,2^j-1} (\mathbf{x}^{(j)} - \mathbf{x}_0^{(j+1)}).$$

While, by Lemma 1.1, the even-indexed Fourier components of $\widehat{\mathbf{x}}^{(j+1)}$ only contain information on $\mathbf{x}^{(j)}$, the odd components give new information on the vector $\mathbf{x}^{(j+1)}$. Restricting the equation system to the odd components yields

$$(\widehat{x}_{2k+1}^{(j+1)})_{k=0}^{2^j-1} = (\widehat{x}_{2^{J-j-1}(2k+1)})_{k=0}^{2^j-1}$$

$$= (\omega_{2^{j+1}}^{(2k+1)\ell})_{k,\ell=0}^{2^j-1} \mathbf{x}_0^{(j+1)} - (\omega_{2^{j+1}}^{(2k+1)\ell})_{k,\ell=0}^{2^j-1} (\mathbf{x}^{(j)} - \mathbf{x}_0^{(j+1)})$$

$$= (\omega_{2^{j+1}}^{(2k+1)\ell})_{k,\ell=0}^{2^j-1} (2\mathbf{x}_0^{(j+1)} - \mathbf{x}^{(j)}) \tag{2.2}$$

$$= \mathbf{F}_{2^j} \cdot \operatorname{diag}(\omega_{2^{j+1}}^{\ell})_{\ell=0}^{2^j-1} (2\mathbf{x}_0^{(j+1)} - \mathbf{x}^{(j)}).$$

(1) First case: $m_j > 2^{j-1}$
In this case we just exploit the observations in (2.2) and obtain

$$\mathbf{x}_0^{(j+1)} = \frac{1}{2} \left(\operatorname{diag}(\omega_{2^{j+1}}^{-\ell})_{\ell=0}^{2^j-1} \cdot \mathbf{F}_{2^j}^{-1} \cdot (\widehat{x}_{2^{J-j-1}(2k+1)})_{k=0}^{2^j-1} + \mathbf{x}^{(j)} \right).$$

Thus, $\mathbf{x}_0^{(j+1)}$ can be computed via an inverse FFT of length 2^j. Further we require 2^j complex multiplications, 2^j additions and one dyadic shift by 2. Finally, $\mathbf{x}_1^{(j+1)} = \mathbf{x}^{(j)} - \mathbf{x}_0^{(j+1)}$ is obtained by $\mathcal{O}(2^j)$ flops.

4

(2) Second case: $m_j \leq 2^{j-1}$

For $m_j \leq 2^{j-1}$, we first compute $L_j := \lceil \log_2 m_j \rceil \leq j - 1$. We denote the first support index of $\mathbf{x}^{(j)}$ by $\mu^{(j)}$. Then the support of $\mathbf{x}^{(j)}$ lies within an interval of length 2^{L_j}, beginning at the index $\mu^{(j)}$. Instead of $\mathbf{x}^{(j)}$, $\mathbf{x}_0^{(j+1)}$, and $\mathbf{x}_1^{(j+1)}$ of length 2^j as above, we consider now the partial vectors of length 2^{L_j} containing the relevant support

$$\widetilde{\mathbf{x}}^{(j)} = (x^{(j)}_{(\mu^{(j)}+r)\bmod 2^j})_{r=0}^{2^{L_j}-1},$$

$$\widetilde{\mathbf{x}}_0^{(j+1)} = (x^{(j+1)}_{(\mu^{(j)}+r)\bmod 2^j})_{r=0}^{2^{L_j}-1}, \quad \widetilde{\mathbf{x}}_1^{(j+1)} = (x^{(j+1)}_{2^j+(\mu^{(j)}+r)\bmod 2^j})_{r=0}^{2^{L_j}-1}.$$

Indeed, by (2.1) the vector $\mathbf{x}^{(j+1)}$ that we want to reconstruct cannot have more than 2^{L_j+1} positive entries, and these relevant entries are contained in the restricted vectors $\widetilde{\mathbf{x}}_0^{(j+1)}$ and $\widetilde{\mathbf{x}}_1^{(j+1)}$. Since the condition

$$\widetilde{\mathbf{x}}_0^{(j+1)} + \widetilde{\mathbf{x}}_1^{(j+1)} = \widetilde{\mathbf{x}}^{(j)}$$

is still satisfied, we only need 2^{L_j} further linearly independent conditions to recover $\mathbf{x}^{(j+1)}$ completely. Employing the equation system (2.2) and using the shorter support of the partial vectors, we find

$$(\widehat{x}_{2^{J-j-1}(2k+1)})_{k=0}^{2^j-1} = (\omega_{2^{j+1}}^{(2k+1)\ell})_{k,\ell=0}^{2^j-1}(2\mathbf{x}_0^{(j+1)} - \mathbf{x}^{(j)})$$

$$= (\omega_{2^{j+1}}^{(2k+1)((\mu^{(j)}+r)\bmod 2^j)})_{k,r=0}^{2^j-1,2^{L_j}-1}(2\widetilde{\mathbf{x}}_0^{(j+1)} - \widetilde{\mathbf{x}}^{(j)}).$$

Instead of considering these 2^j equations for $k = 0,\ldots,2^j-1$, we employ now only the 2^{L_j} equations for $k = 2^{j-L_j}p$, $p = 0,\ldots,2^{L_j}-1$, and obtain

$$(\widehat{x}_{2^{J-j-1}(2^{j+1-L_j}p+1)})_{p=0}^{L_j-1} = (\widehat{x}_{2^{J-L_j}p+2^{J-j-1}})_{p=0}^{L_j-1} \tag{2.3}$$

$$= (\omega_{2^{j+1}}^{(2^{j+1-L_j}p+1)((\mu^{(j)}+r)\bmod 2^j)})_{p,r=0}^{2^{L_j}-1,2^{L_j}-1}(2\widetilde{\mathbf{x}}_0^{(j+1)} - \widetilde{\mathbf{x}}^{(j)}),$$

where

$$(\omega_{2^{j+1}}^{(2^{j+1-L_j}p+1)((\mu^{(j)}+r)\bmod 2^j)})_{p,r=0}^{2^{L_j}-1,2^{L_j}-1}$$

$$= (\omega_{2^{L_j}}^{p(\mu^{(j)}+r)})_{p,r=0}^{2^{L_j}-1}\,\mathrm{diag}(\omega_{2^{j+1}}^{(\mu^{(j)}+r)\bmod 2^j})_{r=0}^{2^{L_j}-1}$$

$$= \mathrm{diag}(\omega_{2^{L_j}}^{\mu^{(j)}p})_{p=0}^{2^{L_j}-1}\,\mathbf{F}_{2^{L_j}}\,\mathrm{diag}(\omega_{2^{j+1}}^{(\mu^{(j)}+r)\bmod 2^j})_{r=0}^{2^{L_j}-1}.$$

We finally conclude from (2.3)

$$\widetilde{\mathbf{x}}_0^{(j+1)} = \frac{1}{2}\mathrm{diag}(\omega_{2^{j+1}}^{-(\mu^{(j)}+r)\bmod 2^j})_{r=0}^{2^{L_j}-1}\,\mathbf{F}_{2^{L_j}}^{-1}\,\mathrm{diag}(\omega_{2^{L_j}}^{-\mu^{(j)}p})_{p=0}^{2^{L_j}-1}\,(\widehat{x}_{2^{J-L_j}p+2^{J-j-1}})_{p=0}^{L_j-1}$$

$$+ \frac{1}{2}\widetilde{\mathbf{x}}^{(j)}.$$

Thus, in this case the recovery of $\widetilde{\mathbf{x}}_0^{(j+1)}$ requires only $\mathcal{O}(L_j \log L_j)$ flops, while $\widetilde{\mathbf{x}}_1^{(j+1)}$ is obtained from $\widetilde{\mathbf{x}}_1^{(j+1)} = \widetilde{\mathbf{x}}^{(j)} - \widetilde{\mathbf{x}}_0^{(j+1)}$.

The new fast algorithm to compute the vector $\mathbf{x} \in \mathbb{R}_+^N$ with possible short support from its Fourier transform $\widehat{\mathbf{x}}$ is summarized in the following algorithm, where we iteratively compute the periodized vectors $\mathbf{x}^{(j)} \in \mathbb{R}_+^{2^j}$ by applying either the method of case 1 or case 2 at each iteration level. If there is some a priori information available on a lower bound 2^{s-1} for the support length m of $\mathbf{x}$, then we may start the iteration by computing the periodized vector $\mathbf{x}^{(s)}$ of length 2^s, otherwise we just start with $s = 0$ in the algorithm.

5

Algorithm 2.1 *(Sparse FFT for real nonnegative vectors)*
Input: $\widehat{\mathbf{x}} = (\widehat{x}_k)_{k=0}^{N-1} \in \mathbb{C}^N$, $N = 2^J$;
   $s = 0$ or s such that 2^{s-1} is a lower bound for $m = |\mathrm{supp}\,\mathbf{x}|$;
   threshold parameter T.

1. *Generate* $\widehat{\mathbf{x}}^{(s)} := (\widehat{x}_{2^{J-s}k})_{k=0}^{2^s-1}$ *by extracting suitable components from* $\widehat{\mathbf{x}}$.

2. *Compute the periodized vector* $\mathbf{x}^{(s)} := \mathbf{F}_{2^s}^{-1}\widehat{\mathbf{x}}^{(s)}$ *by inverse FFT of length* 2^s.

3. *For* $j = s, \ldots, J-1$ *do*
  Compute $m_j := |\mathrm{supp}\,\mathbf{x}^{(j)}|$ *and find the first support index* $\mu^{(j)}$ *of* $\mathbf{x}^{(j)}$.

  • Case 1*: If* $m_j > 2^{j-1}$*, then*
   Build $\mathbf{y}^{(j)} := (\widehat{x}_{2^{J-j-1}(2k+1)})_{k=0}^{2^j-1}$ *and compute*

$$\mathbf{z}^{(j)} := \mathrm{diag}(\omega_{2^{j+1}}^{-\ell})_{\ell=0}^{2^j-1} \cdot \mathbf{F}_{2^j}^{-1} \cdot \mathbf{y}^{(j)}$$

   using an inverse FFT of length 2^j.
   Compute

$$\mathbf{x}^{(j+1)} := \frac{1}{2}\begin{pmatrix} \mathbf{x}^{(j)} + \mathbf{z}^{(j)} \\ \mathbf{x}^{(j)} - \mathbf{z}^{(j)} \end{pmatrix}.$$

   For $k = 0, \ldots, 2^{j+1}-1$*, apply a threshold procedure*

$$x_k^{(j+1)} := \begin{cases} \mathrm{Re}\,x_k^{(j+1)} & \textit{if } \mathrm{Re}\,x_k^{(j+1)} \geq T, \\ 0 & \textit{else.} \end{cases}$$

   end (if).

  • Case 2*: If* $m_j \leq 2^{j-1}$*, then*
   Compute $L_j := \lceil \log_2 m_j \rceil$*. Build the vectors*

$$\widetilde{\mathbf{x}}^{(j)} := (x^{(j)}_{(\mu^{(j)}+r)\,\mathrm{mod}\,2^j})_{r=0}^{2^{L_j}-1}, \quad \mathbf{y}^{(j)} := (\widehat{x}_{2^{J-L_j}p+2^{J-j-1}})_{p=0}^{2^{L_j}-1}.$$

   Compute

$$\mathbf{z}^{(j)} := \mathrm{diag}(\omega_{2^{L_j}}^{-\mu^{(j)}p})_{p=0}^{2^{L_j}-1}\, \mathbf{F}_{2^{L_j}}^{-1}\, \mathrm{diag}(\omega_{2^{j+1}}^{-(\mu^{(j)}+r)\,\mathrm{mod}\,2^j})_{r=0}^{2^{L_j}-1}\, \mathbf{y}^{(j)}$$

   using an inverse FFT of length 2^{L_j}.
   Compute $\widetilde{\mathbf{x}}_0^{(j+1)} := \frac{1}{2}(\widetilde{\mathbf{x}}^{(j)} + \mathbf{z}^{(j)})$ *and* $\widetilde{\mathbf{x}}_1^{(j+1)} := \frac{1}{2}(\widetilde{\mathbf{x}}^{(j)} - \mathbf{z}^{(j)})$.
   For $k = 0, \ldots, 2^{L_j}-1$*, apply a threshold procedure*

$$(\widetilde{\mathbf{x}}_0^{(j+1)})_k := \begin{cases} \mathrm{Re}\,(\widetilde{\mathbf{x}}_0^{(j+1)})_k & \textit{if } \mathrm{Re}\,(\widetilde{\mathbf{x}}_0^{(j+1)})_k \geq T, \\ 0 & \textit{else,} \end{cases}$$

$$(\widetilde{\mathbf{x}}_1^{(j+1)})_k := \begin{cases} \mathrm{Re}\,(\widetilde{\mathbf{x}}_1^{(j+1)})_k & \textit{if } \mathrm{Re}\,(\widetilde{\mathbf{x}}_1^{(j+1)})_k \geq T, \\ 0 & \textit{else.} \end{cases}$$

   Determine $\mathbf{x}_0^{(j+1)}$ *and* $\mathbf{x}_1^{(j+1)}$ *by*

$$(\mathbf{x}_0^{(j+1)})_{(\mu^{(j)}+k)\,\mathrm{mod}\,2^j} := \begin{cases} (\widetilde{\mathbf{x}}_0^{(j+1)})_k & k = 0, \ldots, 2^{L_j}-1, \\ 0 & k = 2^{L_j}, \ldots, 2^j, \end{cases}$$

$$(\mathbf{x}_1^{(j+1)})_{(\mu^{(j)}+k)\bmod 2^j} := \begin{cases} (\widetilde{\mathbf{x}}_1^{(j+1)})_k & k = 0, \ldots, 2^{L_j}-1, \\ 0 & k = 2^{L_j}, \ldots, 2^j. \end{cases}$$

$$\text{Set } \mathbf{x}^{(j+1)} := \begin{pmatrix} \mathbf{x}_0^{(j+1)} \\ \mathbf{x}_1^{(j+1)} \end{pmatrix}.$$

$$\text{end (if)}$$

$$\text{end (for)}$$

Output: $\mathbf{x}^{(J)} = \mathbf{x}$.

The threshold parameter T in the algorithm ensures that we obtain a real nonnegative vector as a result regardless of small numerical errors that may arise. In case of noisy Fourier data, the threshold parameter needs to be chosen suitably to suppress errors in the solution vector, see Section 3.

Remark 2.2 *1. At every reconstruction step the algorithm automatically decides whether the first or the second case applies. For this purpose, the support length of* $\mathbf{x}^{(j)}$ *has to be computed. This can be efficiently done by using the known support indices of the preceding periodization* $\mathbf{x}^{(j-1)}$. *By definition of the periodization, the vector* $\mathbf{x}^{(j)}$ *can only have positive entries at the support indices of* $\mathbf{x}^{(j-1)}$ *and at these indices shifted by* 2^{j-1}. *Hence, only* $2m_{j-1}$ *entries have to be considered in order to find the support length* m_j *and the first support index* $\mu^{(j)}$ *of* $\mathbf{x}^{(j)}$ *causing an effort of* $\mathcal{O}(m_j)$ *flops.*

2. We want to emphasize the importance of the modulo operation in the diagonal matrix $\mathrm{diag}(\omega_{2^{j+1}}^{-(\mu^{(j)}+r)\bmod 2^j})_{r=0}^{2^{L_j}-1}$ *in case 2 of the algorithm. While for* $\mu^{(j)} \leq 2^j - 2^{L_j}$ *this matrix can be simplified to*

$$\mathrm{diag}(\omega_{2^{j+1}}^{-(\mu^{(j)}+r)\bmod 2^j})_{r=0}^{2^{L_j}-1} = \mathrm{diag}(\omega_{2^{j+1}}^{-(\mu^{(j)}+r)})_{r=0}^{2^{L_j}-1} = \omega_{2^{j+1}}^{-\mu^{(j)}}\,\mathrm{diag}(\omega_{2^{j+1}}^{-r})_{r=0}^{2^{L_j}-1},$$

it follows for $\mu^{(j)} > 2^j - 2^{L_j}$ *that*

$$\mathrm{diag}(\omega_{2^{j+1}}^{-(\mu^{(j)}+r)\bmod 2^j})_{r=0}^{2^{L_j}-1} = \omega_{2^{j+1}}^{-\mu^{(j)}} \begin{pmatrix} \mathbf{I}_{2^j-\mu^{(j)}} & \\ & -\mathbf{I}_{2^{L_j}-2^j+\mu^{(j)}} \end{pmatrix} \mathrm{diag}(\omega_{2^{j+1}}^{-r})_{r=0}^{2^{L_j}-1},$$

where $\mathbf{I}_{2^j-\mu^{(j)}}$ *and* $\mathbf{I}_{2^{L_j}-2^j+\mu^{(j)}}$ *denote identity matrices of the given size.*

Let us summarize the numerical effort of the complete algorithm. If we choose $s = 0$, the complexity of the algorithm is at most $\mathcal{O}(m \log_2 m \log_2 \frac{N}{m})$. At each iteration step, the support length m_j of the periodized vectors $\mathbf{x}^{(j)}$ can only increase, i.e., we have $m_0 \leq m_1 \leq \ldots \leq m_{J-1} \leq m_J = m$. Once the the final support length has been achieved, say at the iteration step L, and $2^{L-1} < m_L = m \leq 2^L$, we will always employ the second case in the further iteration steps $j = L+1, \ldots, J-1$ that requires $\mathcal{O}(m \log_2 m)$ flops at each step.

The first L reconstructing steps $j = 1, \ldots, L$ may require either the first or the second case (depending on the distribution of nonzero values of $\mathbf{x}$) and require all together at most $\mathcal{O}(2^L L) = \mathcal{O}(m \log m)$ flops, caused by the inverse FFT of size 2^j, a multiplication with a diagonal matrix of size $2^j \times 2^j$ and 2^{j+2} additions and multiplications computing the periodization $\mathbf{x}^{(j+1)}$ at each iteration step, similarly as a usual FFT algorithm of length 2^L.

Together, we have an effort of $\mathcal{O}((J-L)m \log_2 m) = \mathcal{O}(m \log_2 m (\log_2 N/m))$ flops to compute $\mathbf{x}$.

Remark 2.3 *1. The sublinear complexity of the algorithm can only be achieved by employing less than the given N Fourier samples in the vector $\widehat{\mathbf{x}}$. Indeed at the j-th iteration step, we employ either 2^j new Fourier samples in the first case or only 2^{L_j} Fourier samples in the second case, collected in the vector $\mathbf{y}^{(j)}$. Assuming as before that $2^{L-1} \leq m \leq 2^L$, we apply at steps $L+1, \ldots, J-1$ the second case requiring only $(J-L-1)2^L$ Fourier samples while we need at most 2^{L+1} Fourier samples at the first steps $1, \ldots, L$. Altogether, the number of applied Fourier samples is bounded by $\mathcal{O}(m(\log_2 N/m))$.*

2. The proposed algorithm is efficient for any vector $\mathbf{x} \in \mathbb{R}_+^N$, whether or not it has short support. The complexity does never exceed $\mathcal{O}(N \log N)$ of usual FFT algorithms. In case that the support of the vector is quite short compared to the full vector length, we benefit from the algorithm concerning the computational complexity. We may even benefit from the algorithm if the vector has (almost) full support length but is sparse, such that the second case of the algorithms applies in intermediate steps. For example, a vector $\mathbf{x} \in \mathbb{R}_+^N$ containing several equidistantly distributed short support pieces, these support pieces may add up to one short support interval for smaller periodized vectors $\mathbf{x}^{(j)}$ such that we can take advantage of the algorithm.

Let us give an example: Choose $\mathbf{x} \in \mathbb{R}_+^{1028}$ with positive entries $x_0 = 1$, $x_{256} = 1$, $x_{512} = 1$ and $x_{768} = 1$. Then $\mathbf{x}^{(7)}$ has two positive entries, $x_0 = 2$ and $x_{256} = 2$. The periodizations $\mathbf{x}^{(6)}, \ldots, \mathbf{x}^{(0)}$ only have one positive entry, $x_0 = 4$, such that case 2 applies with $L_j = 0$ for $j=1, \ldots, 6$.

3 Numerical Results

We consider the numerical stability of the proposed algorithm. For that purpose, we apply the algorithm to Fourier data being perturbed by uniform noise $\boldsymbol{\varepsilon} = (\varepsilon_k)_{k=0}^{N-1}$, i.e., we have given data

$$\widehat{y}_k = \widehat{x}_k + \varepsilon_k$$

with $|\varepsilon_k| \leq \delta$. The above algorithm also applies to noisy data where we have to choose the threshold parameter T suitably.

In the noisy case, it is of particular importance to determine the support interval correctly at each iteration step. As before, this is achieved by only considering the relevant entries given by the support interval of the preceding periodization. Additionally, the threshold parameter T has to be set in order to distinguish between relevant components of $\mathbf{x}$ and noise.

Let us now give some numerical examples. We measure the noise level for the Fourier data using the signal-to-noise-ratio

$$\mathrm{SNR} = 20 \cdot \log_{10} \frac{\|\widehat{\mathbf{x}}\|_2}{\|\boldsymbol{\varepsilon}\|_2}.$$

The mean error of the reconstruction is given by $\|\mathbf{x} - \mathbf{x}'\|_2/N$, where $\mathbf{x}'$ denotes the reconstruction of $\mathbf{x}$ by our algorithm.

Consider first a vector $\mathbf{x}$ of length $N = 2^8 = 256$, with nonzero entries $x_{50} = 5$, $x_{53} = 8$, $x_{54} = 1$, $x_{179} = 2$, $x_{180} = 7$ and $x_{181} = 4$. We disturb the Fourier data $\widehat{\mathbf{x}}$ by uniform noise $\boldsymbol{\varepsilon}$ with SNR $= 20$ and reconstruct $\mathbf{x}$ from $\widehat{\mathbf{y}} = \widehat{\mathbf{x}} + \boldsymbol{\varepsilon}$ using our algorithm. In this example, we have $\|\boldsymbol{\varepsilon}\|_\infty = 2.149$ and $\|\boldsymbol{\varepsilon}\|_1/N = 1.100$. Choosing $T = 0.9$, the
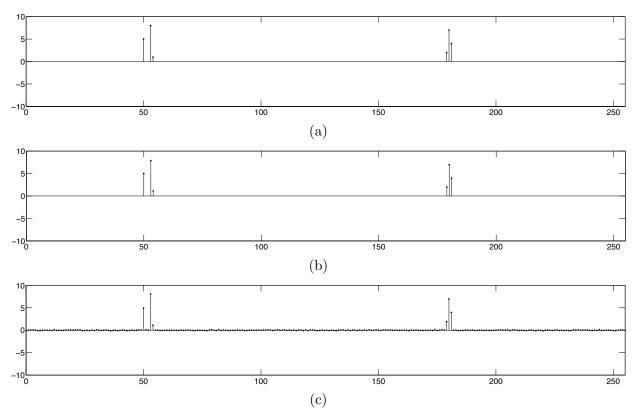
Figure 1:
(a) Original vector $\mathbf{x}$ of length $N = 256$; (b) Reconstruction of $\mathbf{x}$ using the sparse FFT Algorithm 2.1; (c) Reconstruction of $\mathbf{x}$ using the inverse FFT.
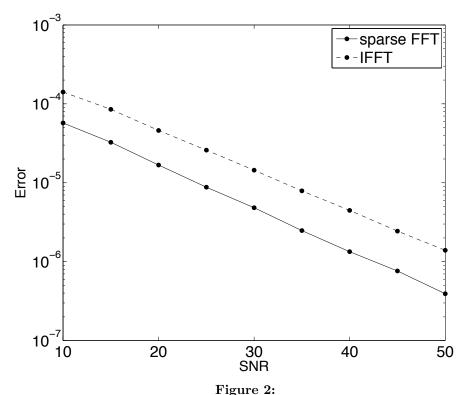
algorithm performs seven reconstruction steps in order to recover $\mathbf{x}$, where in the first three steps, case 1 is used and in the last four steps, case 2 applies.

The reconstructed vector $\mathbf{x}'$ has nonzero entries $x_{50} = 5.005$, $x_{53} = 7.868$, $x_{54} = 1.111$, $x_{179} = 1.972$, $x_{180} = 6.980$, and $x_{181} = 3.964$ yielding an error $\|\mathbf{x} - \mathbf{x}'\|_2/N = 7.003 \cdot 10^{-4}$. In particular, the support of $\mathbf{x}$ is correctly found. Compared to this, the inverse Fourier transform provides an error $\|\mathbf{x} - \mathbf{F}_{256}^{-1}\widehat{\mathbf{y}}\|_2/N = 0.0049$.

Figure 1 illustrates the vector $\mathbf{x}$ as well as both reconstructions from $\widehat{\mathbf{y}}$, by our algorithm and by an inverse FFT.

In a second example, we apply the algorithm to reconstruct randomly chosen real vectors $\mathbf{x} \in \mathbb{R}_+^N$ for $N = 2^{15}$ with entries $0 \leq x_k \leq 10$ from their Fourier data and analyze the reconstruction error. The vectors in our experiment have a support length of $m = 15$. For each noise level between SNR = 10 and SNR = 50, we consider 100 randomly chosen vectors and apply the algorithm to the noisy Fourier data $\widehat{\mathbf{y}}$. The reconstruction error $\|\mathbf{x} - \mathbf{x}'\|_2/N$ for a reconstruction $\mathbf{x}'$ by our algorithm is computed as well as the error $\|\mathbf{x} - \mathbf{F}_N^{-1}\widehat{\mathbf{y}}\|_2/N$ of an inverse FFT reconstruction. The parameter T is chosen appropriately for each noise level. The results of the experiment are shown in Figure 2.

The findings show that the proposed algorithm is numerically stable and returns a good reconstruction with a small error compared to usual inverse FFT. Additionally, it has a lower complexity: in our example with $N = 2^{15}$, our Algorithm 2.1 applies in average 4.5 times case 1 and 10.5 times case 2.

9

Figure 2:
Average reconstruction error $\|\mathbf{x} - \mathbf{x}'\|_2/N$ for different levels of uniform noise, comparing our deterministic sparse FFT algorithm and usual inverse FFT.

Acknowledgement

References

[1] A. Akavia, Deterministic sparse Fourier approximation via fooling arithmetic progressions, in Proc. 23rd COLT, 2010, pp. 381–393.

[2] A. Akavia, Deterministic sparse Fourier approximation via approximating arithmetic progressions, IEEE Trans. Inform. Theory **60**(3) (2014), 1733–1741.

[3] J. R. Fienup, Phase retrieval algorithms: A comparison, Appl. Opt. **21**, 2758–2769 (1982).

[4] A. Gilbert, P. Indyk, M.A. Iwen, and L. Schmidt, Recent developments in the sparse Fourier transform, IEEE Signal Processing Magazine **31**(5) (2014), 91–100.

[5] H. Hassanieh, P. Indyk, D. Katabi, and E. Price, Near-optimal algorithm for sparse Fourier transform, Proc. 44th annual ACM symposium on Theory of Computing, 2012, pp. 563–578.

[6] H. Hassanieh, P. Indyk, D. Katabi, and E. Price, Simple and practical algorithm for sparse Fourier transform, Proc. 23th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '12), 2012, pp. 1183–1194.

[7] S. Heider, S. Kunis, D. Potts, and M. Veit, A sparse Prony FFT, Proc. 10th International Conference on Sampling Theory and Applications (SAMPTA), 2013, pp. 572–575.

[8] M.A. Iwen, Combinatorial sublinear-time Fourier algorithms, Found. Comput. Math. **10** (2010), 303–338.

[9] M.A. Iwen, Improved approximation guarantees for sublinear-time Fourier algorithms, Appl. Comput. Harmon. Anal. **34** (2013), 57–82.

[10] D. Lawlor, Y. Wang, and A. Christlieb, Adaptive sub-linear time Fourier algorithms, Advances in Adaptive Data Analysis **5**(1) (2013), 1350003 (25 pages).

[11] J. Morgenstern, Note on a lower bound on the linear complexity of the fast Fourier transform, J. Assoc. Comput. Mach. **20**(2) (1973), 305–306.

[12] S. Pawar and K. Ramchandran, Computing a k-sparse n-length discrete Fourier transform using at most $4k$ samples and $\mathcal{O}(k \log k)$ complexity, IEEE International Symposium on Information Theory (2013), pp. 464–468.

[13] G. Plonka and M. Tasche, Prony methods for recovery of structured functions, GAMM-Mitt. **37**(2) (2014), 239–258.

[14] G. Plonka and K. Wannenwetsch, A deterministic sparse FFT algorithm for vectors with small support, Numer. Algor., online first, DOI 10.1007/s11075-015-0028-0.

[15] D. Potts, M. Tasche, T. Volkmer, Efficient spectral estimation by MUSIC and ESPRIT with application to sparse FFT, Front. Appl. Math. Stat., 2016, to appear.