Deterministic Sparse FFT Algorithms

Dissertation

zur Erlangung des mathematisch-naturwissenschaftlichen Doktorgrades
„Doctor rerum naturalium“
der Georg-August-Universität Göttingen

im Promotionsprogramm *Mathematical Sciences*
der Georg-August University School of Science (GAUSS)

vorgelegt von
Katrin Ulrike Wannenwetsch
aus Stuttgart

Göttingen, 2016

Betreuungsausschuss

Prof. Dr. Gerlind Plonka-Hoch
Institut für Numerische und Angewandte Mathematik
Georg-August-Universität Göttingen

Prof. Dr. Felix Krahmer
Fakultät für Mathematik
Technische Universität München

Mitglieder der Prüfungskommission

Referentin:

Prof. Dr. Gerlind Plonka-Hoch
Institut für Numerische und Angewandte Mathematik
Georg-August-Universität Göttingen

Korreferent:

Prof. Dr. Daniel Potts
Fakultät für Mathematik
Technische Universität Chemnitz

Weitere Mitglieder der Prüfungskommission:

Jun.-Prof. Dr. Anja Fischer
Institut für Numerische und Angewandte Mathematik
Georg-August-Universität Göttingen

Prof. Dr. Felix Krahmer
Fakultät für Mathematik
Technische Universität München

PD Dr. Hartje Kriete
Mathematisches Institut
Georg-August-Universität Göttingen

Prof. Dr. Tatyana Krivobokova
Institut für Mathematische Stochastik
Georg-August-Universität Göttingen

Prof. Dr. D. Russell Luke
Institut für Numerische und Angewandte Mathematik
Georg-August-Universität Göttingen

Tag der mündlichen Prüfung: 9. August 2016

Acknowledgements

This thesis has been written during the last years when I had the joy to work in a very pleasant atmosphere at the Institute for Numerical and Applied Mathematics at the Georg-August-Universität Göttingen.

First of all, I am much obliged to my advisor Gerlind Plonka-Hoch for her supervision, her encouraging support, many inspiring discussions, new insights and her literally always "open door". I am also very thankful to Felix Krahmer and Daniel Potts for their interest in my work and for being my second advisor respectively for acting as a referee for my thesis.

Furthermore, I want to thank my colleagues of the "Research Group for Mathematical Signal and Image Processing" for providing a comfortable and cooperative working atmosphere, many discussions as well as the continuous coffee and tea supply.

This thesis would not have been possible without the generous financial support of the DFG in the project "Efficient function reconstruction using eigenfunctions of linear operators" and the Research Training Groups 1023 "Identification in Mathematical Models" and 2088 "Discovering Structure in Complex Data" which not only provided financial support but also a great scientific program. I am furthermore very grateful for the numerous possibilities to travel to workshops and conferences.

I am eternally thankful to my parents for their understanding, advice and unconditional support. Additionally, I want to thank Anne and Annekathrin for being part of my life and the best "kleine Anne" and "große Anne" one could wish for.

Finally, special thanks go to Oliver for his never-ending optimism, support, patience and love. Thank you.

Contents

List of Figures

List of Tables

Notation

$\mathbb{N}$	natural numbers (excluding 0)
$\mathbb{Z}$	integers
$\mathbb{R}$	real numbers
$\mathbb{R}_+$	real nonnegative numbers (including 0)
$\mathbb{C}$	complex numbers
e	Euler's number
i	imaginary unit $\sqrt{-1}$
π	constant pi
ω_N	N-th root of unity; $\omega_N = \mathrm{e}^{-\frac{2\pi \mathrm{i}}{N}}$
N	vector length
$\mathbf{e}_k$	k-th unit vector in $\mathbb{R}^N$
$\mathbf{I}_N$	identity matrix of size N
J	dyadic natural number; $N = 2^J$
$\mathbf{x}$	vector of length N
$\mathbf{F}_N$	Fourier matrix of size N
$\widehat{\mathbf{x}}$	Fourier transform of vector $\mathbf{x}$; $\widehat{\mathbf{x}} := \mathbf{F}_N \mathbf{x}$
M	sparsity of vector $\mathbf{x}$; $M = \|\mathbf{x}\|_0 = \|\{k \mid x_k = 0\}\|$
m	support length of vector $\mathbf{x}$
μ	first support index of $\mathbf{x}$
$\mathbf{x}^{(j)}$	j-th periodization (of length 2^j) of vector $\mathbf{x}$
m_j	support length of $\mathbf{x}^{(j)}$
$\lfloor a \rfloor$	largest integer less than or equal to $a \in \mathbb{R}$
$\lceil a \rceil$	smallest integer greater than or equal to $a \in \mathbb{R}$
L_j	$L_j = \lceil \log_2 m_j \rceil$
$\mu^{(j)}$	first support index of periodization $\mathbf{x}^{(j)}$
$\boldsymbol{\varepsilon}$	noise vector
SNR	signal-to-noise-ratio
$\mathbf{A}$	matrix of size $N_1 \times N_2$
$\widehat{\mathbf{A}}$	Fourier transform of matrix $\mathbf{A}$; $\widehat{\mathbf{A}} := \mathbf{F}_{N_1} \mathbf{A} \mathbf{F}_{N_2}$
$m_1 \times m_2$	support size of matrix $\mathbf{A}$
(μ_1, μ_2)	first support index of matrix $\mathbf{A}$
$\mathbf{E}$	noise matrix

1. Introduction

The discrete Fourier transform (DFT) is a common and well-known transform that maps a finite, discrete signal to its spectrum of frequencies. It has various applications in many fields, such as in signal processing or in data compression but it can be applied to compute convolutions or for the solution of partial differential equations as well.

Due to this versatility in practice, it is of high interest to develop fast algorithms that allow efficient computations of discrete Fourier transforms, as the number of arithmetical operations which is needed to compute a DFT of length N by matrix multiplication is proportional to N^2. Reducing computational costs became more and more interesting when machine computing came up, the first algorithm for fast Fourier transform (FFT) was published by Cooley and Tukey in 1965 [9]. Since this time, numerous different versions of FFT algorithms (e.g., also for nonequispaced data [43]) have been developed and broadly applied in many fields. Charles Van Loan [53] states that:

> *The fast Fourier transform (FFT) is one of the truly great computational developments of this century. It has changed the face of science and engineering so much so that it is not an exaggeration to say that life as we know it would be very different without the FFT.*

This emphasizes the great importance of FFTs. Conventional FFT algorithms can be applied to arbitrary vectors of length N and require a number of arithmetical operations which is of the order $N \log N$. It has been shown in [32] that the qualitative bound of $\mathcal{O}(N \log N)$ operations cannot be improved. Hence, one possibility to achieve a lower complexity is to impose restrictions on the vectors such as sparsity or clustered nonzero components.

Therefore, there has been an increasing activity in the field of sparse FFT algorithms in recent years. Here, a signal of length N is assumed to possess only M significant frequency components. Sparse FFT algorithms focus on the problem of computing the M-sparse Fourier transform of these signals. In the following, we provide a short overview on the different approaches and their underlying principles.

The survey [16] describes the general principle of many algorithms that we want to summarize here. The procedure typically consists of three steps. First, frequencies with coefficients of large magnitude in the Fourier spectrum are identified. This is done by binning the frequency band into several subsets so, with high probability, each of the few searched frequencies is assigned to a different subset. For this step, various types of filters can be used. Then, the significant frequencies in each bin are determined applying search techniques as e.g. described in [16]. Subsequently, the coefficients corresponding to the identified frequencies are estimated in a second step. Finally, the signal is updated by subtracting the summands which where found in the preceding steps so the procedure can be repeated for the new signal.

One of the first publications on approximate sparse FFT applying this principle with a sublinear complexity is [15], but the algorithm is still quadratic in sparsity. This initial approach by Gilbert et. al. was later improved in [17], where an algorithm that achieves linear time in sparsity M, that is polynomial in $\log N$, $\log M$ and depends also on the accuracy as well as the probability of a successful reconstruction is presented. In [18], the authors present a tutorial on this algorithm. Both aforementioned algorithms are randomized which means that they only succeed with a certain probability smaller than one. Further randomized sparse FFT algorithms following the above scheme include [21], [22], [34] and [26]. The algorithm proposed in [34] even achieves a $\mathcal{O}(M \log M)$ complexity, whereas [21] needs in $\mathcal{O}(M \log N)$ arithmetical operations.

An overview on runtimes in [16] indicates that for a sparsity of $M = 50$, different implementations of [22] become to be more efficient than a standard FFT algorithm for signal lengths larger than 2^{17} and implementations of [17] for signal lengths $> 2^{21}$. Considering sparsity, for vectors of length $N = 2^{22}$, the algorithms pay off for $M < 100$ [17] resp. $M < 500$ [22]. However, the runtime of [21] was better for all considered vector lengths and sparsities.

A drawback of these randomized algorithms is that they fail with a constant probability greater than zero. The algorithms might return a wrong result, although there is no efficient method to check whether this solution is correct. Additionaly, samples have to be drawn randomly for this methods which is not easily achievable in any case. Therefore, also deterministic sparse FFT algorithms computing (approximate) sparse FFTs without errors in the noiseless case exist. These techniques are in particular advantageous for applications that are sensitive to failure. We mention publications containing deterministic algorithms which nevertheless all follow a similar principle as described above. In [27] and [28], a special subsampling method is used in order to identify fre-

quencies via a combinatorial approach using the Chinese Remainder Theorem. The algorithm uses a large amount of samples for which the signal has to be evaluated at certain points but the computations require only shorter DFTs than the signal length. Its complexity is $\mathcal{O}(M^2 \log^4 N)$. Further deterministic approaches can be found in [1], [2] and [29]. The latter proposes an algorithm with $\mathcal{O}(M \log M)$ runtime, though it only works with the a priori condition that time-shifted samples are available. In [8], this algorithm is generalized for the noisy case.

Sparse FFT algorithms for the multidimensional case have e.g. been proposed in [25] and [28], where the latter achieves an arithmetical complexity $\mathcal{O}(d^4 M^2 \log^4(dN))$, depending on the dimension d.

From another point of view, the reconstruction of sparse vectors from Fourier data might be seen as a parameter estimation problem as well. This means that techniques such as Prony's method can also be applied in this setting. Algorithms combining both approaches can e.g. be found in [42] or [47] and we will discuss the connection between sparse FFT and Prony's method in Section 2.4 in detail.

Furthermore, compressed sensing ([12], [7]) is related to randomized sparse FFT methods. Compressed sensing aims to recover compressible signals from linear measurements, hence it is also applicable to reconstruct a sparse frequency vector from a small number of sampling values. However, reconstruction algorithms usually incorporate iterative schemes to solve corresponding minimization problems with higher computational costs than $N(\log N)$. The relationship between both approaches is described in more detail in [29].

Finally, we discuss applications of sparse FFT algorithms. In [20], the authors consider the problem of GPS synchronization, where a satellite sends signals to a GPS receiver which can then determine its position by analyzing the signal. In order to do so, the signal has to be processed by the receiver where sparse signals occur. The necessary Fourier transforms can be performed by sparse FFT algorithms which accelerates the GPS synchronization. Another application is spectrum sensing where spectra are "scanned" in order to identify frequencies. Here, sparse FFT methods can be used for a faster computation, e.g. in the field of GHz-spectra [23] or cognitive radios which can detect vacant frequencies in order to use them for transmission [55]. Moreover, sparse FFT methods can be applied for 2D correlation spectroscopy of in vivo data, see [51].

In this thesis, we focus on sublinear-time algorithms for sparse FFT which are deterministic, i.e., they produce no error (apart from potential numerical errors) in the case of exact data. The problem considered here is the reconstruction of vectors from Fourier

data using as few Fourier values and arithmetical operations as possible. This means that we actually develop fast algorithms for inverse Fourier transforms. The vectors to be reconstructed are assumed to fulfill some a priori conditions, such as being sparse or having nonzero components only within a small interval. We present a completely new reconstruction approach which is different from the methods described above. Instead of applying the aforementioned binning methods, we employ the concept of periodized vectors, which can in some sense be considered as similar, but opposed to most of the aforementioned methods requires no randomization.

In a first approach, we assume a vector $\mathbf{x} \in \mathbb{C}^N$ to have small support of given length m, i.e., only a small index interval of length m where nonzero components might occur. For these vectors, we develop an iterative reconstruction procedure which achieves a complexity of $\mathcal{O}(m \log m)$ in the case of exact data and $\mathcal{O}(m \log N)$ for noisy data. The proposed algorithm can be stabilized for noisy input data and already pays off for $m < N/4$. The only a priori condition for its application is the knowledge of the support length (or an upper bound) m of the vector and its Fourier transform $\widehat{\mathbf{x}}$, even though we do not need all Fourier values for the reconstruction. This algorithm is also generalized for the two-dimensional case and can therefore be applied to matrices and images with small support.

Furthermore, we develop another sublinear-time algorithm for the reconstruction of real nonnegative vectors $\mathbf{x} \in \mathbb{R}_+^N$ from Fourier data. In this setting, we do not need the a priori condition that the vector to be reconstructed has a small support. The proposed algorithm automatically recognizes if a vector has only a few clustered nonzero components and benefits from this fact. This means that the procedure can be successfully applied to any arbitrary real nonnegative vector, even though the complexity is lower for vectors with small support. The deterministic algorithm requires $\mathcal{O}(m \log m \log(N/m))$ arithmetical operations for vectors of length N with support length m. Numerical results show that the algorithm also works in a very stable way.

The latter algorithm is generalized to vectors with sparsity M without the need for clustering of these nonzero components. We propose a new method for fast reconstruction of general nonnegative M-sparse signals which involves Vandermonde type matrices. These matrices with knots on the unit circle are investigated further and we present bounds on the condition of the Vandermonde type matrices as well as bounds on the minimal distances of knots on the unit circle that can occur in the reconstruction procedure.

This thesis is organized as follows. First, we give an overview on the reconstruction of

signals from Fourier data in Chapter 2. The DFT is introduced as well as its efficient implementation, the fast Fourier transform (FFT) which we analyze in detail. Subsequently, the recovery of vectors with only one nonzero component is considered. We show how these vectors can be easily reconstructed from Fourier values in the case of exact data. In this context, we discuss how the reconstruction problem can be seen as a parameter estimation problem and therefore techniques such as Prony's method can be applied to solve it. We use this approach for vector reconstruction from Fourier data and summarize recent results on this field which combine sparse FFT with Prony's method. In Chapter 3, we provide the underlying principles for all of our sparse FFT algorithms, e.g. the definition of the support interval of a vector or periodizations. Additionally, we indicate how the reconstruction of vectors in the Fourier basis can be transferred to other bases.

Chapter 4 and 5 contain the development and the analysis of the new sublinear sparse FFT algorithms mentioned above. Chapter 4 is devoted to the new deterministic algorithm for the reconstruction of complex vectors with small support from Fourier data. First, we develop an iterative reconstruction procedure for exact data before proposing stabilizations for the case of noisy data. In numerical experiments, the algorithm is applied to perturbed data and evaluated in different settings. In Chapter 5, we present the algorithm for real nonnegative vectors which may or may not have short support. The iterative recovering procedure for this kind of vectors is deduced and we formulate a corresponding algorithm. The chapter closes with a numerical evaluation of this algorithm.

In Chapter 6, the setting of Chapter 5 is generalized to real nonnegative vectors with sparsity M, i.e., we omit the assumption that the nonzero components of the vectors are clustered within a short support interval. For this setting, the proposed adaptive algorithm contains matrices depending on certain parameters. In order to guarantee a stable reconstruction, we prove bounds on the conditions of these matrices as well as worst case estimates.

Finally, in Chapter 7 we show that the one-dimensional algorithm in Chapter 4 can be used to develop a 2D sparse FFT algorithm. We include numerical experiments illustrating the application of the algorithm to matrices and images.

We summarize the thesis in Chapter 8 and give an overview on open problems and further research. In the appendix, some exemplary MATLAB implementations for the developed algorithms are given.

Parts of this thesis have already been published in [40], [39], [38] and [41].

2. Reconstructing vectors from Fourier data

The reconstruction of data from Fourier values or, equivalently, the computation of Fourier transforms has been a widely discussed problem within the last decades.

Before presenting new results on deterministic sparse fast Fourier transform algorithms, this chapter introduces the preliminaries on discrete Fourier transform as well as some approaches to efficient reconstruction of signals from Fourier data. We study the fast Fourier transform for general complex vectors of length N and the Prony method for recovery of vectors with few nonzero components in the following.

2.1. Discrete Fourier Transform

We give a short introduction to the discrete Fourier transform (DFT) and present some of its properties. Our reference for this section is [52, Chapter 2.3].

Let $\mathbf{x} = (x_j)_{j=0}^{N-1} \in \mathbb{C}^N$. The *discrete Fourier transform* $\widehat{\mathbf{x}} \in \mathbb{C}^N$ of $\mathbf{x}$ is defined by

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

where the discrete Fourier matrix is given by

$$\mathbf{F}_N = \left(\omega_N^{jk}\right)_{j,k=0}^{N-1} \in \mathbb{C}^{N\times N}$$

with the N-th root of unity $\omega_N := \mathrm{e}^{-\frac{2\pi \mathrm{i}}{N}}$. This means that the Fourier transform $\widehat{\mathbf{x}}$ is of the form

$$\widehat{x}_k = \sum_{j=0}^{N-1} x_j \omega_N^{jk}, \qquad k = 0, \ldots, N-1.$$

Consequently, we obtain the inverse Fourier transform as

$$\mathbf{x} := \mathbf{F}_N^{-1}\widehat{\mathbf{x}}$$

with

$$\mathbf{F}_N^{-1} := \frac{1}{N}\left(\omega_N^{-jk}\right)_{j,k=0}^{N-1} \in \mathbb{C}^{N\times N}.$$

Hence, the components of $\mathbf{x}$ can be written as

$$x_j = \frac{1}{N}\sum_{k=0}^{N-1} \widehat{x}_k \omega_N^{-kj}, \qquad j = 0, \ldots, N-1.$$

We illustrate this by a small example.

Example 2.1 Let $N = 4$. Then, since $\mathrm{e}^{-\frac{\pi \mathrm{i}}{2}} = -\mathrm{i}$, we have

$$\mathbf{F}_4 = \left(\mathrm{e}^{-\frac{2\pi \mathrm{i} jk}{4}}\right)_{j,k=0}^{3} = \left((-\mathrm{i})^{jk}\right)_{j,k=0}^{3} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -\mathrm{i} & -1 & \mathrm{i} \\ 1 & -1 & 1 & -1 \\ 1 & \mathrm{i} & -1 & -\mathrm{i} \end{pmatrix}.$$

For $\mathbf{x} = \begin{pmatrix} 1 & 0 & 0 & 0 \end{pmatrix}^T$, the discrete Fourier transform is

$$\widehat{\mathbf{x}} = \mathbf{F}_4\mathbf{x} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -\mathrm{i} & -1 & \mathrm{i} \\ 1 & -1 & 1 & -1 \\ 1 & \mathrm{i} & -1 & -\mathrm{i} \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}.$$

Consequently, the inverse Fourier transform is given by

$$\mathbf{x} = \mathbf{F}_4^{-1}\widehat{\mathbf{x}} = \frac{1}{4}\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & \mathrm{i} & -1 & -\mathrm{i} \\ 1 & -1 & 1 & -1 \\ 1 & -\mathrm{i} & -1 & \mathrm{i} \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} = \frac{1}{4}\begin{pmatrix} 4 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}.$$

An important property of the Fourier matrix $\mathbf{F}_N$ is its symmetry, i.e., it holds that

$$\mathbf{F}_N^T = \mathbf{F}_N.$$

Remark 2.2 *We observe that*

$$\mathbf{F}_N^{-1} = \frac{1}{N}\left(\omega_N^{-jk}\right)_{j,k=0}^{N-1} = \frac{1}{N}\overline{\mathbf{F}}_N$$

and therefore

$$\mathbf{F}_N\overline{\mathbf{F}}_N^T = \mathbf{F}_N\overline{\mathbf{F}}_N = N\mathbf{F}_N\mathbf{F}_N^{-1} = N\mathbf{I}_N,$$

where $\mathbf{I}_N$ *is the unity matrix of size* N*. Hence, we can conclude that*

$$\frac{1}{\sqrt{N}}\mathbf{F}_N \quad \textit{and} \quad \sqrt{N}\mathbf{F}_N^{-1}$$

are unitary transforms.

Moreover, we have the following relation for $\mathbf{F}_N$ and its inverse $\mathbf{F}_N^{-1}$:

$$\mathbf{F}_N^{-1} = \frac{1}{N}\mathbf{U}_N\mathbf{F}_N,$$

where we denote by $\mathbf{U}_N$ the "flip matrix" of size $N \times N$

$$\mathbf{U}_N = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 0 & & 1 \\ \vdots & & \iddots & \\ 0 & 1 & & 0 \end{pmatrix} = \left(\delta_{j+k}^{(N)}\right)_{j,k=0}^{N-1}$$

with the N-periodic Kronecker symbol

$$\delta_\ell^{(N)} = \begin{cases} 1 & \ell \equiv 0 \bmod N, \\ 0 & \ell \not\equiv 0 \bmod N, \end{cases}$$

for $\ell \in \mathbb{Z}$.

The above relation is particularly interesting, since it shows that the Fourier transform and its inverse can be computed using the same algorithm. Therefore, we only need to develop algorithms for either one of the transforms.

We summarize some properties of the DFT in the following theorem.

Theorem 2.3 (cf. Satz 2.3.1 in [52]) *Let* $\mathbf{x}, \mathbf{y} \in \mathbb{C}^N$ *and* $\widehat{\mathbf{x}} = \mathbf{F}_N\mathbf{x}$, $\widehat{\mathbf{y}} = \mathbf{F}_N\mathbf{y} \in \mathbb{C}^N$ *be the corresponding discrete Fourier transforms. Then the following properties hold:*

2. Reconstructing vectors from Fourier data

1. Linearity:

$$\begin{aligned} \widehat{\mathbf{x}+\mathbf{y}} &= \widehat{\mathbf{x}}+\widehat{\mathbf{y}}, \\ \widehat{\alpha\mathbf{x}} &= \alpha\widehat{\mathbf{x}} \qquad (\alpha\in\mathbb{C}). \end{aligned}$$

2. Flipping property:

$$\mathbf{x} \;=\; \mathbf{F}_N^{-1}\widehat{\mathbf{x}} \;=\; \frac{1}{N}\mathbf{U}_N\mathbf{F}_N\widehat{\mathbf{x}}.$$

3. Symmetry:

$$\begin{aligned} \widehat{\mathbf{U}_N\mathbf{x}} &= \mathbf{U}_N\widehat{\mathbf{x}}, \\ \widehat{\overline{\mathbf{x}}} &= \mathbf{U}_N\overline{\widehat{\mathbf{x}}}. \end{aligned}$$

4. Time respectively frequency shifting:

$$\begin{aligned} \widehat{\mathbf{P}^n\mathbf{x}} &= \mathbf{M}^n\widehat{\mathbf{x}}, \\ \widehat{\mathbf{M}^{-n}\mathbf{x}} &= \mathbf{P}^n\widehat{\mathbf{x}}, \end{aligned}$$

where $n\in\mathbb{Z}$, $\mathbf{P}=\left(\delta^{(N)}_{j-k-1}\right)_{j,k=0}^{N-1}$ *and* $\mathbf{M}=\operatorname{diag}\left(\omega_N^j\right)_{j=0}^{N-1}$.

5. Parseval's Theorem:

$$\frac{1}{N}(\widehat{\mathbf{x}},\widehat{\mathbf{y}}) \;=\; (\mathbf{x},\mathbf{y}) \;:=\; \sum_{k=0}^{N-1} x_k\overline{y_k}.$$

For $\mathbf{x}=\mathbf{y}$, *we obtain the conservation of energies*

$$\frac{1}{N}\sum_{k=0}^{N-1}|\widehat{x}_k|^2 \;=\; \frac{1}{N}(\widehat{\mathbf{x}},\widehat{\mathbf{x}}) \;=\; (\mathbf{x},\mathbf{x}) \;=\; \sum_{k=0}^{N-1}|x_k|^2 \;\geq\; 0.$$

The computational complexity of the discrete Fourier transform corresponds to the complexity of the multiplication with a matrix of dimension N, i.e., the transform requires $N(N-1)$ complex additions and N^2 complex multiplications.

In the following sections we will show that the complexity of $\mathcal{O}(N^2)$ can be improved by both more efficient algorithms and a priori conditions on the vectors to be computed.

2.2. Fast Fourier Transform

The fast Fourier transform (FFT) is an efficient implementation of the discrete Fourier transform which reduces the computational complexity of the calculation of a complex Fourier transform for a vector $\mathbf{x} \in \mathbb{C}^N$ from $\mathcal{O}(N^2)$ to $\mathcal{O}(N \log N)$.

The underlying principle of all considered algorithms is the *divide-and-conquer* technique which means that the computation can be divided into smaller problems of the same structure that can again be decomposed in the same way.

The procedure which is probably mostly used today has been published in 1965 by Cooley and Tukey [9] and is known as the Cooley-Tukey algorithm. We illustrate the idea of this algorithm as well as examples for small N, following the presentations in [52] and [33].

The algorithm is based on the fact that is it possible to decompose the vector length N into a product of two integers $N_1, N_2 > 1$ such that $N = N_1 N_2$ holds. Let

$$\widehat{x}_k = \sum_{j=0}^{N-1} x_j \omega_N^{jk}, \qquad k = 0, \ldots, N-1,$$

denote the components of the Fourier vector $\widehat{\mathbf{x}}$. By redefining the indices as

$$\begin{aligned} k &= k_1 N_2 + k_2, & k_1 &= 0, \ldots, N_1 - 1, \ k_2 = 0, \ldots, N_2 - 1, \\ j &= j_1 + j_2 N_1, & j_1 &= 0, \ldots, N_1 - 1, \ j_2 = 0, \ldots, N_2 - 1, \end{aligned}$$

we obtain

$$\widehat{x}_{k_1 N_2 + k_2} = \sum_{j_1=0}^{N_1-1} \sum_{j_2=0}^{N_2-1} x_{j_1 + j_2 N_1} \omega_N^{(j_1 + j_2 N_1)(k_1 N_2 + k_2)}, \qquad k_r = 0, \ldots, N_r - 1 \text{ for } r = 1, 2.$$

Evaluating

$$\omega_N^{(j_1 + j_2 N_1)(k_1 N_2 + k_2)} = \omega_{N_1}^{j_1 k_1} \omega_N^{j_1 k_2} \omega_{N_2}^{j_2 k_2}$$

yields

$$\begin{aligned}\widehat{x}_{k_1N_2+k_2} &= \sum_{j_1=0}^{N_1-1} \omega_{N_1}^{j_1k_1}\omega_N^{j_1k_2} \sum_{j_2=0}^{N_2-1} x_{j_1+j_2N_1}\omega_{N_2}^{j_2k_2} \\ &= \sum_{j_1=0}^{N_1-1} \omega_{N_1}^{j_1k_1}\omega_N^{j_1k_2} u_{j_1+k_2N_1}, \qquad k_r = 0,\dots,N_r-1 \text{ for } r=1,2,\end{aligned} \tag{2.1}$$

where the sums

$$u_{j_1+k_2N_1} := \sum_{j_2=0}^{N_2-1} x_{j_1+j_2N_1}\omega_{N_2}^{j_2k_2}, \qquad k_2 = 0,\dots,N_2-1,$$

represent Fourier transforms of length N_2. The remaining sum in (2.1) can be computed by first multiplying the values $u_{j_1+k_2N_1}$ with the *twiddle factors* $\omega_N^{j_1k_2}$ which yields

$$v_{j_1+k_2N_1} := u_{j_1+k_2N_1}\omega_N^{j_1k_2}, \qquad j_1 = 0,\dots,N_1-1, \quad k_2 = 0,\dots,N_2-1.$$

Finally, we compute for $k_2 \in \{0,\dots,N_2-1\}$ the N_2 discrete Fourier transforms of length N_1 by

$$\widehat{x}_{k_1N_2+k_2} = \sum_{j_1=0}^{N_1-1} v_{j_1+k_2N_1}\omega_{N_1}^{j_1k_1}, \qquad k_1 = 0,\dots,N_1-1.$$

This means that we can split the DFT of size $N = N_1N_2$ into N_1 DFTs of size N_2 and N_2 DFTs of size N_1. These problems have the same structure as the original sum but are of smaller dimension.

Compared to $N(N-1)$ complex additions and N^2 complex multiplications for the DFT of size N, the computation now only requires $N_1N_2(N_2-1) + N_2N_1(N_1-1) = N(N_1+N_2-2)$ additions and $N_1N_2^2 + N + N_2N_1^2 = N(N_1+N_2+1)$ multiplications. If the factors N_1 and N_2 can again be factorized, we can further decompose the problem into smaller DFTs.

We consider from now on vectors $\mathbf{x} \in \mathbb{C}^N$ with $N = 2^J$ for some $J \in \mathbb{N}$. By our above considerations, we can decompose the DFT for those vectors into DFTs of size 2. FFT algorithms for vectors of length 2^J are referred to as *radix-2 algorithms*.

Let $N_1 = 2$ and $N_2 = N/2$ and evaluate the above equation (2.1) of the Cooley-Tukey

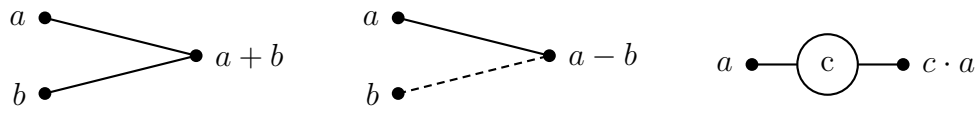


Figure 2.1.: Addition, subtraction and multiplication visualized in butterfly graphs.

algorithm. This yields for the components of $\widehat{\mathbf{x}}$

$$\widehat{x}_{k_1 N/2+k_2} = \sum_{\ell=0}^{N/2-1} x_{2\ell}\, \omega_{N/2}^{\ell k_2} + \omega_N^{k_1 N/2+k_2} \sum_{\ell=0}^{N/2-1} x_{2\ell+1}\, \omega_{N/2}^{\ell k_2}$$

for $k_1 = 0, 1$ and $k_2 = 0, \ldots, N/2-1$. Using $\omega_N^{k_2+N/2} = -\omega_N^{k_2}$, we obtain the following two expressions for the components of $\widehat{\mathbf{x}}$

$$\widehat{x}_k = \sum_{\ell=0}^{N/2-1} x_{2\ell}\, \omega_{N/2}^{\ell k} + \omega_N^k \sum_{\ell=0}^{N/2-1} x_{2\ell+1}\, \omega_{N/2}^{\ell k}, \qquad k = 0, \ldots, N/2-1,$$

and

$$\widehat{x}_{k+N/2} = \sum_{\ell=0}^{N/2-1} x_{2\ell}\, \omega_{N/2}^{\ell k} - \omega_N^k \sum_{\ell=0}^{N/2-1} x_{2\ell+1}\, \omega_{N/2}^{\ell k}, \qquad k = 0, \ldots, N/2-1.$$

Hence, each component of $\widehat{\mathbf{x}}$ can be obtained by computation of two DFTs of size $N/2$. By reiterating this principle, all entries $\widehat{x}_k$, $k = 0, \ldots, N-1$, can be efficiently computed. For the case that $N = 8$, we illustrate the additions and multiplications which are necessary to compute the Fourier components $\widehat{x}_k$ from x_k, $k = 0, \ldots, N-1$, in Figure 2.2, cf. Abb. 7 in [52, Chapter 3]. The so-called butterfly graph visualizes the arithmetical operations of the transform where addition, subtraction and multiplication are given as in Figure 2.1, see also Abb. 1–3 in [52, Chapter 3].

The procedure of the Cooley-Tukey algorithm is referred to as decimation in time. We also present the decimation in frequency. The corresponding algorithm which is named after Sande and Tukey can be obtained by setting $N_1 = N/2$ and $N_2 = 2$. Thus, the components of $\widehat{\mathbf{x}}$ are of the form

$$\widehat{x}_{2k_1+k_2} = \sum_{\ell=0}^{N/2-1} \left(x_\ell + (-1)^{k_2} x_{N/2+\ell}\right) \omega_N^{(2k_1+k_2)\ell}$$

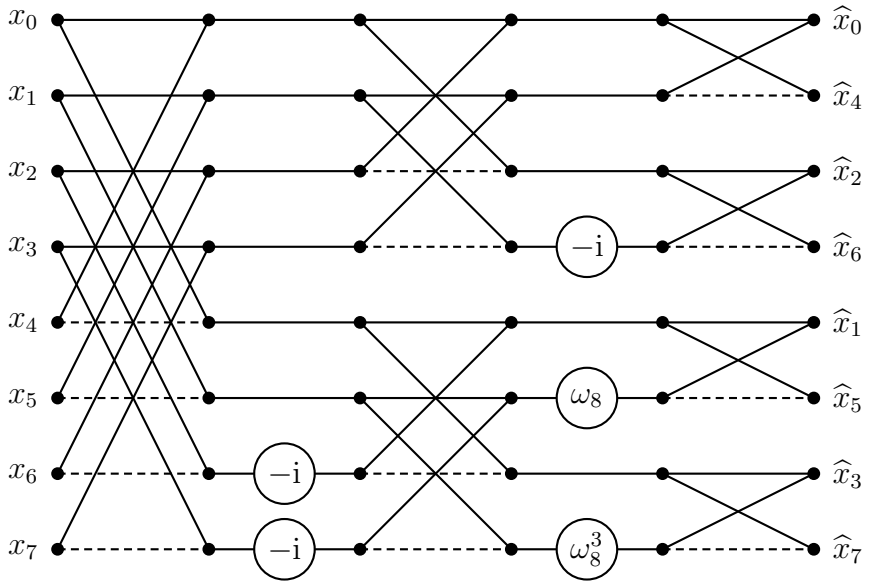


Figure 2.2.: Butterfly graph of the Cooley-Tukey algorithm for a DFT of size $N = 8$.

for $k_1 = 0, \ldots, N/2-1$ and $k_2 = 0, 1$. Hence the even and the odd entries of $\widehat{\mathbf{x}}$ are given by

$$\widehat{x}_{2k} = \sum_{\ell=0}^{N/2-1} \left(x_\ell + x_{N/2+\ell}\right) \omega_{N/2}^{k\ell}, \qquad k = 0, \ldots, N/2-1,$$

resp.

$$\begin{aligned}\widehat{x}_{2k+1} &= \sum_{\ell=0}^{N/2-1} \left(x_\ell - x_{N/2+\ell}\right) \omega_N^{(2k+1)\ell} \\ &= \sum_{\ell=0}^{N/2-1} \left(\left(x_\ell - x_{N/2+\ell}\right) \omega_N^{\ell}\right) \omega_{N/2}^{k\ell}, \qquad k = 0, \ldots, N/2-1.\end{aligned}$$

The Sande-Tukey algorithm for $N = 8$ is illustrated as a butterfly graph in Figure 2.3 (cf. [52, Chapter 3, Abb. 6]).

Let us shortly comment on the computational complexity of this algorithm. The above representation shows that the DFT of size N can be split into $N/2$ DFTs of size 2 of the vectors $(x_\ell, x_{N/2+\ell})^T$, $\ell = 0, \ldots, N/2-1$ as well as $N/2$ multiplications with the twiddle factors ω_N^ℓ and two DFTs of size $N/2$ of the vectors $\left(x_\ell + x_{N/2+\ell}\right)_{\ell=0}^{N/2-1}$ resp. $\left(x_\ell + x_{N/2+\ell}\right)_{\ell=0}^{N/2-1}$. The latter can again be decomposed in the same manner

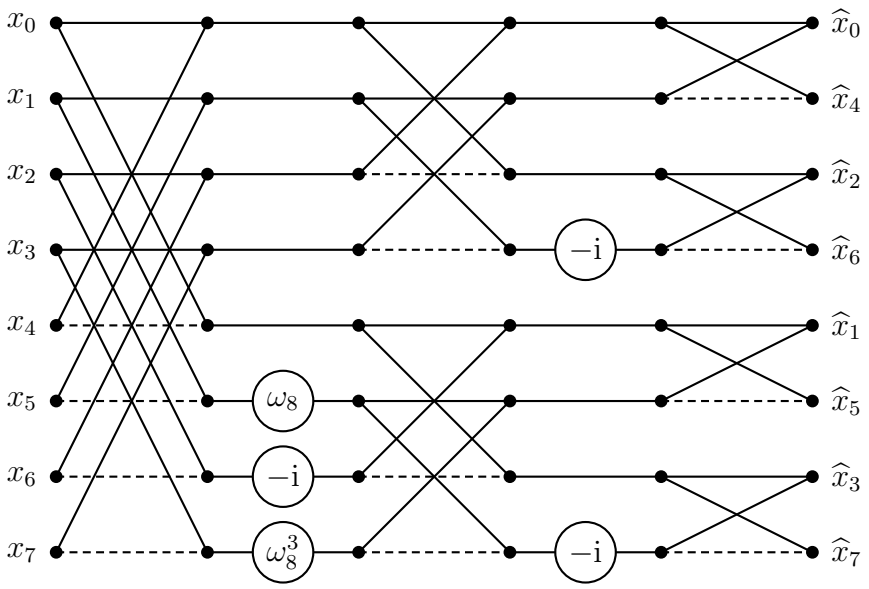


Figure 2.3.: Butterfly graph of the Sande-Tukey algorithm for a DFT of size $N = 8$.

(setting $N_1 = N/4$ and $N_2 = 2$). Thus, we can compute the Fourier values in $J = \log_2 N$ steps if we proceed in this way. At each iteration step, the splitting into smaller DFTs requires N additions and $N/2$ multiplications with twiddle factors which yields altogether $N \log_2 N$ complex additions and $N/2 \log_2 N$ complex multiplications, hence a computational complexity of $\mathcal{O}(N \log N)$.

For the Cooley-Tukey algorithm, we obtain the same results on computational complexity based on similar considerations. The reduction in complexity is illustrated in the following small example.

Example 2.4 Let $N = 2^{10} = 1024$. The computation of a Fourier transform of length N by matrix multiplication would require $N(N-1)$ additions and N^2 multiplications. We compare this to the complexity of a radix-2 FFT algorithm.

$$\frac{N^2 + N(N-1)}{\frac{N}{2}\log_2 N + N \log_2 N} = \frac{2N-1}{\frac{3}{2}\log_2 N} = \frac{2048-1}{\frac{3}{2}\cdot 10} = \frac{2047}{15} \approx 136.47$$

The result shows that the complexity of a matrix multiplication of size $N \times N$ with $N = 2^{10}$ is by a factor 136 greater than for the FFT algorithm.

There exist various other FFT algorithms as e.g. the radix-4 algorithms for $N = 4^J$, where the decomposition can be started setting $N_1 = N/4$ and $N_2 = 4$. The different approaches can also be combined.

In general, the complexity achieved by FFT algorithms is $\mathcal{O}(N \log N)$ although the exact number of operations required by the different approaches varies. However, Morgenstern [32] showed that there is no linear algorithm for computing a DFT of length N which requires less than $\mathcal{O}(N \log N)$ operations.

2.3. Reconstructing vectors with one nonzero component from Fourier data

Let us now assume that for given $\widehat{\mathbf{x}} \in \mathbb{C}^N$ we know a priori that the vector $\mathbf{x} \in \mathbb{C}^N$ to be reconstructed is sparse, i.e., it has only few nonvanishing components. To begin our considerations on reconstructing vectors from Fourier data, we illustrate the reconstruction of vectors with only one nonzero component.

Let $\mathbf{x} \in \mathbb{C}^N$ be a complex vector with one nonzero entry, i.e.,

$$\mathbf{x} := x_k \mathbf{e}_k = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ x_k \\ 0 \\ \vdots \\ 0 \end{pmatrix},$$

where $\mathbf{e}_j = (\delta_{j\ell})_{\ell=0}^{N-1}$, $j = 0, \ldots, N-1$, denote the unit vectors in $\mathbb{C}^N$. Then the Fourier transform $\widehat{\mathbf{x}} = \mathbf{F}_N \mathbf{x}$ of $\mathbf{x}$ is given by

$$\widehat{\mathbf{x}} = x_k \mathbf{F}_N \mathbf{e}_k = x_k \begin{pmatrix} \omega_N^0 \\ \omega_N^k \\ \vdots \\ \omega_N^{(N-2)k} \\ \omega_N^{(N-1)k} \end{pmatrix}.$$

Hence, the first two entries of $\widehat{\mathbf{x}}$ are $\widehat{x}_0 = x_k$ and $\widehat{x}_1 = x_k \omega_N^k$. These two components are

already sufficient to reconstruct $\mathbf{x}$. From

$$x_k = \widehat{x}_0 \qquad \text{and} \qquad \omega_N^k = \frac{\widehat{x}_1}{\widehat{x}_0},$$

we obtain the nonzero component x_k of $\mathbf{x}$ and its index k. Note that the determination of k is only stable in case of exact data.

For vectors with more than one nonzero entry, reconstruction procedures are more involved. However, there are several established methods for the efficient reconstruction of vectors from Fourier data. The fast Fourier transform which was discussed in Section 2.2 is one of them and can be applied to any complex vector.

Moreover, there are further approaches that focus on the reconstruction of sparse vectors which have only few nonzero components. We discuss some of them in the following, such as the Prony method [10] or sparse fast Fourier transforms.

2.4. Reconstructing sparse vectors from Fourier data: Prony's method

In this section, we focus on Prony's method and show how a complex vector with several nonzero entries can be recovered using this method.

Let us first recall the vector reconstruction problem of the preceding section and consider a complex vector $\mathbf{x} \in \mathbb{C}^N$ with a small number $M \in \mathbb{N}$ of nonzero components, i.e., an M-sparse vector where the sparsity of a vector $\mathbf{x}$ is given by $M := \|\mathbf{x}\|_0$. Then the vector $\mathbf{x}$ can be written as

$$\mathbf{x} = \sum_{j=1}^{M} x_{n_j} \mathbf{e}_{n_j}$$

with $0 \leq n_1 \leq \cdots \leq n_M \leq N-1$. We want to reconstruct $\mathbf{x}$ from its Fourier transform

$$\widehat{\mathbf{x}} = \sum_{j=1}^{M} x_{n_j} \mathbf{F}_N \mathbf{e}_{n_j} = \left(\sum_{j=1}^{M} x_{n_j} \omega_N^{\ell n_j} \right)_{\ell=0}^{N-1}.$$

Hence, in order to recover $\mathbf{x}$, we need to find the indices n_j and the coefficients x_{n_j} for $j = 1, \ldots, M$.

Problems of this form can be solved with the help of Prony's method. The first ideas of this method trace back to G. R. de Prony and date from 1795, cf. [10]. We shortly

summarize the reconstruction procedure for exact data as it can e.g. be found in [35].

Prony's method can be applied for the reconstruction of exponential sums of the form

$$f(x) := \sum_{j=1}^{M} c_j \mathrm{e}^{xT_j}$$

where the complex parameters c_j, T_j, $j = 1, \ldots, M$, are unknown with $c_j \neq 0$ and $T_j \in (-\infty, 0] + \mathrm{i}[-\pi, \pi)$. The function f can be reconstructed from $2M$ function values $f(\ell)$, $\ell = 0, \ldots, 2M-1$. We define the so-called Prony polynomial

$$P(z) := \prod_{j=1}^{M} (z - \lambda_j)$$

with $\lambda_j := e^{T_j}$, i.e., the roots of the polynomial are the exponentials that we want to determine. Let further

$$P(z) := \sum_{k=0}^{M} p_k \, z^k$$

be the monomial representation of $P(z)$ such that $p_M = 1$ holds. We observe that

$$\begin{aligned} \sum_{k=0}^{M} p_k f(k+m) = \sum_{k=0}^{M} p_k \sum_{j=1}^{M} c_j \mathrm{e}^{(k+m)T_j} &= \sum_{j=1}^{M} c_j \lambda_j^m \left(\sum_{k=0}^{M} p_k \lambda_j^k \right) \\ &= \sum_{j=1}^{M} c_j \lambda_j^m P(\lambda_j) = 0 \end{aligned} \tag{2.2}$$

for all $m \in \mathbb{N}_0$ since the values λ_j, $j = 1, \ldots, M$, are the roots of the polynomial $P(z)$ and therefore $P(\lambda_j) = 0$. Thus, we obtain a linear Hankel system

$$\sum_{k=0}^{M-1} p_k f(k+m) = -f(M+m), \qquad m = 0, \ldots, M-1,$$

which allows us to determine the coefficients p_k of the Prony polynomial $P(z)$. The zeros of $P(z)$ are the values $\lambda_j = \mathrm{e}^{T_j}$ and we hence also obtain the values T_j, $j = 1, \ldots, M$.

Finally, we complete the reconstruction of f by computing the coefficients c_j from the

overdetermined linear system

$$f(\ell) = \sum_{j=1}^{M} c_j \mathrm{e}^{\ell T_j}, \qquad \ell = 0, \ldots, 2M-1$$

using a least-squares approach. We now want to use Prony's method to reconstruct the vector

$$\mathbf{x} = \sum_{j=1}^{M} x_{n_j} \mathbf{e}_{n_j}$$

of the above example. Let us assume that we know the Fourier values

$$\widehat{x}_\ell = \sum_{j=1}^{M} x_{n_j} \omega_N^{\ell n_j}, \qquad \ell = 0, \ldots, 2M-1.$$

The Prony polynomial is in this case of the form

$$P(z) := \prod_{j=1}^{M} (z - \omega_N^{n_j}) = \sum_{\ell=0}^{M} p_\ell \, z^\ell$$

with unknown parameters n_j and with $p_M = 1$. Then we obtain, in a similar way as in (2.2), the linear Hankel system

$$\sum_{\ell=0}^{M-1} p_\ell \, \widehat{x}_{\ell+m} = -\widehat{x}_{M+m}, \qquad m = 0, \ldots, M-1,$$

from which we recover the coefficients p_k of the Prony polynomial $P(z)$. This allows us to determine the roots $\omega_N^{n_j}$ of $P(z)$ and hence the indices n_j, $j = 1, \ldots, M$. Then the components x_{n_j} are given by the overdetermined linear system

$$\widehat{x}_\ell = \sum_{j=1}^{M} x_{n_j} \omega_N^{\ell n_j}, \qquad \ell = 0, \ldots, 2M-1.$$

We summarize the reconstruction of a vector $\mathbf{x} \in \mathbb{C}^N$ with M nonzero entries from its Fourier transform $\widehat{\mathbf{x}}$ in Algorithm 2.5.

Algorithm 2.5 (Classical Prony method for vector reconstruction from Fourier data)
Input: M and $\widehat{x}_\ell$, $\ell = 0, \ldots, 2M-1$.

1. Solve the Hankel system

$$\begin{pmatrix} \widehat{x}_0 & \widehat{x}_1 & \cdots & \widehat{x}_{M-1} \\ \widehat{x}_1 & \widehat{x}_2 & \cdots & \widehat{x}_M \\ \vdots & \vdots & & \vdots \\ \widehat{x}_{M-1} & \widehat{x}_M & \cdots & \widehat{x}_{2M-2} \end{pmatrix} \begin{pmatrix} p_0 \\ p_1 \\ \vdots \\ p_{M-1} \end{pmatrix} = - \begin{pmatrix} \widehat{x}_M \\ \widehat{x}_{M+1} \\ \vdots \\ \widehat{x}_{2M-1} \end{pmatrix}$$

2. Compute the zeros of the Prony polynomial $P(z) = \sum_{\ell=0}^{M} p_\ell z^\ell$ and extract the parameters n_j from its zeros $z_j = \omega_N^{n_j}$, $j = 1, \ldots, M$.

3. Compute the components x_{n_j} solving the linear system

$$\widehat{x}_\ell = \sum_{j=1}^{M} x_{n_j} \omega_N^{\ell n_j}, \qquad \ell = 0, \ldots, 2M-1.$$

Output: Parameters n_j and x_{n_j}, $j = 1, \ldots, M$.

Unfortunately, Prony's method is in general numerically unstable, see [44]. Therefore, there has been some effort to develop stabilized versions of Prony's method. In [44] and [45], the approximate Prony method is proposed which is based on [5]. Further approaches to methods for parameter identification include MUSIC [49], ESPRIT [48] or the matrix pencil method [24]. Some of these methods have shown to be Prony-like by Potts and Tasche in [46]. Moreover, it has been shown in [37] that Prony's method is equivalent to the annihilating filter method, see e.g. [13], [54].

In contrast to the classical Prony method which was presented above, many of the stabilized versions do not require the a priori knowledge of the number M of active frequencies but detect it automatically if the number of given measurements is sufficiently large. Error estimates for Prony-like methods can e.g. be found in [3], [14], [45].

Let us return to our above example. For many of the stabilized Prony methods, such as e.g. ESPRIT, a singular value decomposition of the Hankel matrix has to be computed and thereafter the eigenvalues of a suitable companion matrix of $P(z)$. In our case, this

means that we have the Hankel matrix

$$\begin{pmatrix} \widehat{x}_0 & \widehat{x}_1 & \cdots & \widehat{x}_{M-1} \\ \widehat{x}_1 & \widehat{x}_2 & \cdots & \widehat{x}_M \\ \vdots & \vdots & & \vdots \\ \widehat{x}_{M-1} & \widehat{x}_M & \cdots & \widehat{x}_{2M-2} \end{pmatrix} = \mathbf{V}\mathbf{D}\mathbf{V}^T$$

with

$$\mathbf{V} = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ \omega_N^{n_1} & \omega_N^{n_2} & \cdots & \omega_N^{n_M} \\ \vdots & \vdots & & \vdots \\ \omega_N^{n_1(M-1)} & \omega_N^{n_2(M-1)} & \cdots & \omega_N^{n_M(M-1)} \end{pmatrix} \quad \text{and} \quad \mathbf{D} = \operatorname{diag}(x_{n_1}, \ldots, x_{n_M}).$$

The computational complexity of this approach is due to the singular value decomposition $\mathcal{O}(M^3)$. The major difficulty arises from the fact that the condition number of the Hankel matrix used in the Prony approach can be arbitrarily large. This can be seen from the factorization above that incorporates a Vandermonde matrix determined by the knots $\omega_N^{n_1}, \ldots, \omega_N^{n_M}$ on the unit circle. The condition number is small if these knots are (almost) equidistantly distributed on the unit circle, see [4]. There exist several ideas to overcome these difficulties. First, one could apply additional Fourier components and hence consider a rectangular Hankel matrix with a better condition. This idea has been used e.g. in [14], [36], [44]. Bounds for the condition number of the corresponding Vandermonde matrix can be found e.g. in [30]. Another possibility would be to include a randomly chosen (odd) "sampling factor" $\sigma \in \mathbb{N}$ and apply the values $\widehat{x}_{\sigma\ell}$ instead of $\widehat{x}_\ell$ for the reconstruction of $\mathbf{x}$ with the hope that the distribution of the new knots $\omega_N^{\sigma n_1}, \ldots, \omega_N^{\sigma n_M}$ is closer to an equidistant distribution on the unit circle. Finally, a splitting approach can be used by examining different bands. This means that we split the set $\{0, \ldots, N-1\}$ into disjoint subsets (by applying a suitable filter) and seek for nonzero components in these subsets. This idea has been pursued in [42], [47].

These new approaches regard the problem of Fourier transforms for sparse vectors as a Prony-like problem. In [42] or [47], the ESPRIT and MUSIC methods are applied for sparse FFT. In order to overcome the problems of a quite large complexity and instability, in [42] quasi-random samples are drawn, in contrast to the procedure in Remark 2.6 below. Additionally, the authors of [42] suggest to split the frequency set into smaller subsets which can then be reconstructed successively.

The recent publication [47] proposes to apply a divide-and-conquer technique and hence to split the reconstruction problem into several smaller problems of lower sparsity with the same structure. They use the technique of shifted sampling which has also been applied in [29], [8].

Remark 2.6 *The classical Prony method has been generalized by Peter and Plonka in [35]. The generalized Prony method can be applied to recover sums of eigenfunctions of linear operators. Let V be a normed vector space over $\mathbb{C}$ and let $\mathcal{A} : V \to V$ be a linear operator which is assumed to have eigenvalues. A set of pairwise distinct eigenvalues of $\mathcal{A}$ is denoted by $\Lambda := \{\lambda_j \mid j \in I\}$. We assign the eigenfunctions v_j to λ_j, $j \in I$, then there is a unique correspondence between eigenvalues and eigenfunctions.*

Let us consider the M-sparse expansion f of eigenfunctions of the operator $\mathcal{A}$,

$$f = \sum_{j \in J} c_j v_j$$

with $J \subset I$, $|J| = M$ and $c_j \neq 0$ for all $j \in J$. Further, we define a linear functional $F : V \to \mathbb{C}$ with the property $Fv_j \neq 0$ for all $j \in J$. Then the expansion f, i.e., the coefficients $c_j \in \mathbb{C}$ and the eigenfunctions v_j, can be uniquely reconstructed from the values $F(\mathcal{A}^k f)$, $k = 0, \ldots, 2M-1$, cf. Theorem 2.1 in [35].

We show that this can be applied to our special situation by choosing a vector space V, a linear operator $\mathbf{D}$ and the functional F in a suitable way, see [35, Chapter 5]. Let $V = \mathbb{C}^N$ and define a linear operator $\mathbf{D} : \mathbb{C}^N \to \mathbb{C}^N$ represented by the diagonal matrix

$$\mathbf{D} := \operatorname{diag}(\omega_N^0, \omega_N^1, \ldots, \omega_N^{N-1}),$$

where $\omega_N = \mathrm{e}^{-\frac{-2\pi \mathrm{i}}{N}}$ is defined as before. Let $F : \mathbb{C}^N \to \mathbb{C}^N$ be a linear functional of the form $F\mathbf{x} = \mathbb{1}^T \mathbf{x} := \sum_{j=0}^{N-1} x_j$. Then Theorem 2.1 in [35] states that we can reconstruct M-sparse vectors of the form

$$\mathbf{x} = \sum_{j=1}^{M} c_{n_j} \mathbf{e}_{n_j}$$

with $0 \leq n_1 \leq \cdots \leq n_M \leq N-1$ from $2M$ values

$$F(\mathbf{D}^k \mathbf{x}) = \mathbb{1}^T \cdot \mathbf{D}^k \mathbf{x} = \omega_N^{0 \cdot k} x_0 + \omega_N^{1 \cdot k} x_1 + \cdots + \omega_N^{(N-1)k} x_{N-1}$$

for $k = 0, \ldots, 2M-1$. This means that the required input values correspond exactly to

the vector $\mathbf{y} = (y_k)_{k=0}^{2M-1}$ *given by*

$$\mathbf{y} = \mathbf{F}_{N,2M}\mathbf{x}$$

where $\mathbf{F}_{N,2M} = \left(\omega_N^{k\ell}\right)_{k,\ell=0}^{2M-1,N-1} \in \mathbb{C}^{2M\times N}$ *contains the first* $2M$ *rows of the Fourier matrix* $\mathbf{F}_N$ *(see also [35, Remark 5.2]). Hence, this approach also allows to reconstruct a vector with few nonzero components from Fourier data* $\widehat{x}_0, \dots, \widehat{x}_{2M-1}$.

3. Preliminaries for sparse FFT algorithms

In this chapter we introduce the preliminaries for the sparse FFT algorithms which are described in the following chapters.

For this general setting we consider complex vectors if not specified otherwise. Let $\mathbf{x} = (x_k)_{k=0}^{N-1} \in \mathbb{C}^N$. As before, the discrete Fourier transform $\widehat{\mathbf{x}} \in \mathbb{C}^N$ of $\mathbf{x}$ is defined by

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

with the Fourier matrix $\mathbf{F}_N = \left(\omega_N^{k\ell}\right)_{k,\ell=0}^{N}$ and $\omega_N := \mathrm{e}^{-\frac{2\pi \mathrm{i}}{N}}$.

In Chapter 2 we gave an overview of conventional FFT algorithms. These algorithms compute the discrete Fourier transform of arbitrary vectors of length $N = 2^J$ and have a complexity of $\mathcal{O}(N \log N)$. One way to improve these bounds is to impose further restrictions on the considered vectors. Therefore, in the following, we develop fast Fourier algorithms for vectors with a special structure such as sparsity or a small support, i.e., vectors that only have few nonzero components. These restrictions allow us to reduce complexity compared to conventional algorithms.

3.1. Vectors with small support and vector periodization

Let us now consider vectors that have a small support. By the support we understand for our purposes the shortest interval of indices of the vector that corresponds to nonzero components.

Definition 3.1 *Let* $\mathbf{x} = (x_k)_{k=0}^{N-1} \in \mathbb{C}^N$. *The* support length $m = |\operatorname{supp} \mathbf{x}|$ *of* $\mathbf{x}$ *is defined as the minimal positive integer* m *for which there exists an index* $\mu \in \{0, \ldots, N-1\}$ *such that* $x_k = 0$ *for all* $k \notin I := \{(\mu + r) \bmod N \mid r = 0, \ldots, m-1\}$. *We call the index set* I *the* support interval *of* $\mathbf{x}$ *and* μ *the* first support index *of* $\mathbf{x}$.

This definition implies that the first component of the support interval needs not necessarily to be the index of the first nonzero entry of the vector and that the support interval is defined modulo the vector length. In other words, the support can begin at the end of the vector and continue at the first entry.

Note that the support interval and the first support index of a vector $\mathbf{x}$ are not necessarily uniquely defined. This is only the case if $m \leq \frac{N}{2}$. Example 3.2 illustrates both cases. However, the support length is unique and the first and the last component of the support, x_μ and $x_{(\mu+m-1) \bmod N}$, are nonzero by definition.

We are especially interested in vectors with $m \ll N$, i.e., vectors being also sparse in a classical sense which means that they have very few nonzero components compared to the vector length. As we allow zero components within the support and hence indices in I which correspond to zero entries, the support length m of a vector $\mathbf{x}$ is an upper bound for its sparsity $\|\mathbf{x}\|_0 = |\{k \,|\, x_k \neq 0\}|$.

The following small examples show different types of support intervals.

Example 3.2

1. $\mathbf{x} = (0,0,0,1,-1,0,0,0)^T \in \mathbb{C}^8$. For this vector, the support interval is uniquely defined: $I = \{3,4\}$. The support length and the first support index of $\mathbf{x}$ are $m = 2$ and $\mu = 3$.

2. $\mathbf{x} = (1,2,0,3,0,0,4,5)^T \in \mathbb{C}^8$. The support interval of $\mathbf{x}$ is unique and given by $I = \{6,7,0,1,2,3\}$, the support length is $m = 6$ and the first support index is $\mu = 6$. The index set I contains one index, 2, which corresponds to a zero component of $\mathbf{x}$.

3. $\mathbf{x} \in \mathbb{C}^{1024}$ with two nonzero entries, $x_0 = 1$ and $x_{512} = 1$. Then $\mathbf{x}$ has support length $m = 513$, but two possible support intervals: $I = \{0,1,2,\ldots,512\}$ with first support index $\mu = 0$ or $I = \{512,513,\ldots,1022,1023,0\}$ with first support index $\mu = 512$. In this example, the support length is much larger than the sparsity of $\mathbf{x}$.

Let from now on $N = 2^J$ for some $J \in \mathbb{N}$, i.e., we consider vectors whose length is a power of two. For this type of vectors, one can define so-called *periodizations*. The principle is as follows: Let $\mathbf{x} \in \mathbb{C}^{2^J}$, then we have periodized vectors $\mathbf{x}^{(j)}$, $j = J,\ldots,0$, where any periodization $\mathbf{x}^{(j)}$ is of length 2^j. Each vector $\mathbf{x}^{(j)}$ is created inductively from the vector of length 2^{j+1}, $\mathbf{x}^{(j+1)}$, by cutting $\mathbf{x}^{(j+1)}$ in the middle, then putting one half on the other and adding up coinciding entries.

Definition 3.3 *Let* $\mathbf{x} \in \mathbb{C}^N$ *with* $N = 2^J$ *for some* $J \in \mathbb{N}$. *Then the* periodizations $\mathbf{x}^{(j)} \in \mathbb{C}^{2^j}$ *of* $\mathbf{x}$ *are given by*

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

for $j = 0, \ldots, J$.

Remark 3.4 *We observe the following special cases of periodizations:*

1. $\mathbf{x}^{(0)} = \sum_{k=0}^{N-1} x_k$ *is the sum of all entries of* $\mathbf{x}$.
2. $\mathbf{x}^{(1)} = \left(\sum_{k=0}^{N/2-1} x_{2k}, \sum_{k=0}^{N/2-1} x_{2k+1} \right)^T$ *has in the first component the sum of all even-indexed entries of* $\mathbf{x}$ *and in the second component the sum of all odd-indexed entries of* $\mathbf{x}$.
3. $\mathbf{x}^{(J)} = \mathbf{x}$ *is the original vector.*

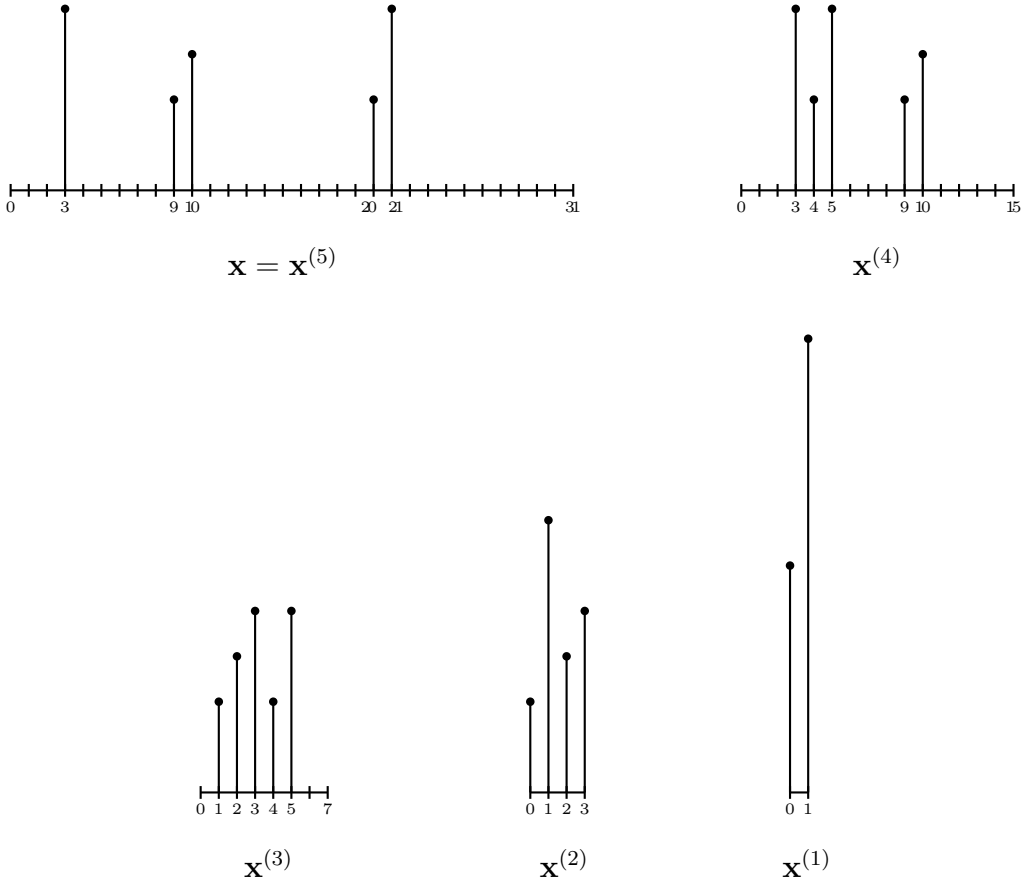
Example 3.5 Let $N = 32 = 2^5$. For simplicity, we consider here a vector with real, nonnegative components. We choose $\mathbf{x}$ to be

$$\mathbf{x} = (0,0,0,4,0,0,0,0,0,2,3,0,0,0,0,0,0,0,0,0,2,4,0,0,0,0,0,0,0,0,0,0)^T.$$

Then $\mathbf{x}$ has the periodizations

$$\begin{aligned}
\mathbf{x}^{(4)} &= (0,0,0,4,2,4,0,0,0,2,3,0,0,0,0,0)^T \\
\mathbf{x}^{(3)} &= (0,2,3,4,2,4,0,0)^T \\
\mathbf{x}^{(2)} &= (2,6,3,4)^T \\
\mathbf{x}^{(1)} &= (5,10)^T \\
\mathbf{x}^{(0)} &= (15)
\end{aligned}$$

In the case of such a vector $\mathbf{x}$ with real components we can easily visualize the periodizations of $\mathbf{x}$:



Remark 3.6 *Consider periodizations $\mathbf{x}^{(j)}$, $j = 0, \ldots, J$, of a vector $\mathbf{x} \in \mathbb{C}^N$, $N = 2^J$, with corresponding support lengths m_j and first support indices $\mu^{(j)}$. Then we observe that the sequence of support lengths*

$$m_0 \leq m_1 \leq \cdots \leq m_{J-1} \leq m_J$$

is always increasing and never decreasing. This can also be observed in the preceding example.

As we consider in the following the reconstruction of vectors from Fourier data, it is of interest how the Fourier transform $\widehat{\mathbf{x}}^{(j)}$ of a periodization $\mathbf{x}^{(j)}$ of $\mathbf{x}$ can be computed. It turns out that the components of the vector $\widehat{\mathbf{x}}^{(j)}$ can already be found as entries of $\widehat{\mathbf{x}}$. This means that if the Fourier transform $\widehat{\mathbf{x}}$ is given, we only have to pick the proper components in order to obtain $\widehat{\mathbf{x}}^{(j)}$.

See also the publications [40], [41] for the following lemma.

Lemma 3.7 *Let* $\mathbf{x} \in \mathbb{C}^N$ *with* $N = 2^J$ *for* $J \in \mathbb{N}$, *and* $\mathbf{x}^{(j)}$, $j = 0, \dots, J$, *be its periodizations, as given in Definition 3.3. Then the discrete Fourier transform* $\widehat{\mathbf{x}}^{(j)}$ *of* $\mathbf{x}^{(j)}$ *is given by*

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

where $\widehat{\mathbf{x}} = \mathbf{F}_N\mathbf{x}$ *is the Fourier transform of* $\mathbf{x}$.

Proof. By Definition 3.3, the components $\widehat{x}_k^{(j)}$ of $\widehat{\mathbf{x}}^{(j)}$ are given by

$$\begin{aligned}\widehat{x}_k^{(j)} &= \sum_{r=0}^{2^j-1} \omega_{2^j}^{kr}\, x_r^{(j)} = \sum_{r=0}^{2^j-1}\sum_{\ell=0}^{2^{J-j}-1} x_{r+2^j\ell}\, \omega_{2^j}^{kr} = \sum_{r=0}^{2^j-1}\sum_{\ell=0}^{2^{J-j}-1} x_{r+2^j\ell}\, \omega_N^{2^{J-j}kr} \\ &= \sum_{n=0}^{N-1} x_n\, \omega_N^{2^{J-j}k(n \bmod 2^j)} = \sum_{n=0}^{N-1} x_n\, \omega_N^{2^{J-j}kn} = \widehat{x}_{2^{J-j}k}.\end{aligned}$$

This proves the assertion. □

The following example illustrates the application of Lemma 3.7 for a vector $\mathbf{x}$ of length $N = 4$.

Example 3.8 Let $N = 4$ and $\mathbf{x} = (1, 1, 1, 0)^T \in \mathbb{R}^4$. Then $\mathbf{x}$ has periodizations $\mathbf{x}^{(2)} = \mathbf{x}$, $\mathbf{x}^{(1)} = (2, 1)^T$ and $\mathbf{x}^{(0)} = 3$.

Further, the Fourier transform $\widehat{\mathbf{x}}$ of $\mathbf{x}$ is

$$\widehat{\mathbf{x}} = \mathbf{F}_4\mathbf{x} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -\mathrm{i} & -1 & \mathrm{i} \\ 1 & -1 & 1 & -1 \\ 1 & \mathrm{i} & -1 & -\mathrm{i} \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 3 \\ -\mathrm{i} \\ 1 \\ \mathrm{i} \end{pmatrix}.$$

For the Fourier transforms of the periodized vectors, we compute

$$\begin{aligned}\widehat{\mathbf{x}}^{(2)} &= \widehat{\mathbf{x}} = \begin{pmatrix} 3 \\ -\mathrm{i} \\ 1 \\ \mathrm{i} \end{pmatrix}, \\ \widehat{\mathbf{x}}^{(1)} &= \mathbf{F}_2\mathbf{x}^{(1)} = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 2 \\ 1 \end{pmatrix} = \begin{pmatrix} 3 \\ 1 \end{pmatrix}, \\ \widehat{\mathbf{x}}^{(0)} &= \mathbf{x}^{(0)} = 3.\end{aligned}$$

3.2. Sparse vector reconstruction for other bases

The above introduced problem of sparse vector reconstruction has a broad application since it can be transferred to other finite bases replacing the discrete Fourier basis. Let us shortly illustrate how this can be done.

One is often interested in a reconstruction of functions that are sparsely represented in a given basis $\mathcal{B} = \{b_0, \ldots, b_{N-1}\}$ that spans an N-dimensional subspace of a Hilbert space H. Let for example $z \in H$ be of the form

$$z = \sum_{j=1}^{M} c_{n_j}\, b_{n_j},$$

where $M \ll N$ and $0 \leq n_1 < n_2 < \ldots < n_M \leq N-1$, $c_{n_j} \in \mathbb{C}$. Further, let $\widetilde{\mathcal{B}} = \{\tilde{b}_0, \ldots, \tilde{b}_{N-1}\}$ be the dual basis to $\mathcal{B}$ in H, i.e., we have

$$\left\langle b_j, \tilde{b}_k \right\rangle = \delta_{j,k},$$

where $\langle \cdot, \cdot \rangle$ denotes the scalar product in H. Choosing now

$$g_\ell := \sum_{k=0}^{N-1} \mathrm{e}^{2\pi \mathrm{i} k\ell/N} \tilde{b}_k = \sum_{k=0}^{N-1} \omega_N^{-k\ell}\, \tilde{b}_k, \qquad \ell = 0, \ldots, N-1,$$

we observe that

$$\begin{aligned} \langle z,\, g_\ell \rangle &= \left\langle \sum_{j=1}^{M} c_{n_j} b_{n_j}, \sum_{k=0}^{N-1} \omega_N^{-\ell k}\, \tilde{b}_k \right\rangle \\ &= \sum_{j=1}^{M} \sum_{k=0}^{N-1} c_{n_j}\, \omega_N^{\ell k} \left\langle b_{n_j},\, \tilde{b}_k \right\rangle = \sum_{j=1}^{M} c_{n_j}\, \omega_N^{\ell n_j} = \widehat{c}_\ell, \end{aligned}$$

where $\widehat{\mathbf{c}} = (\widehat{c}_\ell)_{\ell=0}^{N-1} = \mathbf{F}_N\, \mathbf{c}$ is the discrete Fourier transform of the M-sparse vector $\mathbf{c} \in \mathbb{C}^N$ with $c_j = 0$ for $j \notin \{n_1, \ldots, n_M\}$. Hence, the reconstruction of z from a suitable amount of moments $\langle z,\, g_\ell \rangle$ can be rewritten as the reconstruction of the sparse vector $\mathbf{c}$ from the Fourier data $\widehat{c}_\ell = \langle z, g_\ell \rangle$.

This allows us to apply our developed sparse FFT methods also to reconstruction of sparse vectors represented in a basis $\mathcal{B} = \{b_0, \ldots, b_{N-1}\}$ different from the discrete Fourier basis.

4. A sparse FFT algorithm for vectors with small support

After having introduced algorithms for the fast Fourier transform (FFT) in Chapter 2, we present here a new approach to fast Fourier algorithms with a priori conditions.

Conventional FFT algorithms compute the Fourier transform of a vector of length N with complexity $\mathcal{O}(N \log N)$. As mentioned before, these algorithms are optimal, an improved complexity for such algorithms can only be achieved by assuming a priori conditions on the involved vectors. Sparsity, i.e., the a priori knowledge that the vector merely has few nonzero components, is such a common assumption. Several approaches to FFT for sparse vectors have been described in Chapter 1 and 2. Our setting here is similar: we also assume complex vectors $\mathbf{x} \in \mathbb{C}^N$ with few nonzero entries compared to the vector length N, but we additionally suppose that these components are concentrated in a short interval of the vector $\mathbf{x}$. We refer to vectors of this type as vectors with small support.

We develop a sublinear algorithm that can be applied to all complex vectors $\mathbf{x} \in \mathbb{C}^N$, the only a priori knowledge needed for its application is the given support length m of $\mathbf{x}$ or an upper bound for it, such that $\mathbf{x}$ vanishes outside the support interval. Moreover, we require the knowledge of the Fourier transform $\widehat{\mathbf{x}}$ but no further special values. In particular, we do not have to evaluate additional, very specific data as e.g. in [27] where it is required that the function to be reconstructed can be evaluated at certain points. In contrast, we do not even need all given Fourier values for a successful recovery.

The reconstruction procedure for $\mathbf{x}$ from Fourier data $\widehat{\mathbf{x}}$ is generally divided into two main parts: First, we apply an inverse FFT of relatively short length in order to get a periodization of $\mathbf{x}$ which already contains the full support interval of $\mathbf{x}$. Hence, further reconstruction steps focus on identifying the support within the computed periodized vector and on determining the right support interval of the vector $\mathbf{x}$ in order to shift the support components to the right position.

For exact data, the algorithm achieves a complexity of $\mathcal{O}(m \log m)$ whereas for noisy

measurements, the stabilized algorithm requires $\mathcal{O}(m \log N)$ arithmetical operations.

The results of this chapter have been published in [40].

4.1. Preliminaries

Throughout this chapter, we consider complex vectors $\mathbf{x} \in \mathbb{C}^N$ for $N = 2^J$ with $J \in \mathbb{N}$. The discrete Fourier transform is used as before as well as the periodized vectors $\mathbf{x}^{(j)}$ of length 2^j, $j \in \{0, \ldots, J\}$ (see Definition 3.3).

For further considerations, two facts will be helpful: The Fourier transform $\widehat{\mathbf{x}}^{(j)}$ of a periodization $\mathbf{x}^{(j)}$ as given in Lemma 3.7 can be easily obtained from $\widehat{\mathbf{x}}$ by picking appropriate components. Furthermore, we state here a Lemma on the Fourier transform of vectors with shifted components. The Fourier transform of a vector with cyclically shifted entries only differs from the Fourier transform of the original vector by a multiplication with a root of unity.

Lemma 4.1 *Let $\mathbf{x} = (x_k)_{k=0}^{N-1} \in \mathbb{C}^N$ with $N = 2^J$. Consider for some $j \in \{0, \ldots, J-1\}$ and $\nu \in \{0, \ldots, 2^{J-j} - 1\}$ a shifted version $\mathbf{y} = (y_k)_{k=0}^{N-1} \in \mathbb{C}^N$ with components*

$$y_k := x_{(k+2^j\nu) \bmod N}, \qquad k = 0, \ldots, N-1.$$

Then the components of the Fourier transforms $\widehat{\mathbf{x}} = \mathbf{F}_N \mathbf{x}$ of $\mathbf{x}$ and $\widehat{\mathbf{y}} = \mathbf{F}_N \mathbf{y}$ of the shifted version $\mathbf{y}$ satisfy

$$\widehat{y}_\ell = \omega_{2^{J-j}}^{-\ell\nu}\, \widehat{x}_\ell, \qquad \ell = 0, \ldots, N-1.$$

In particular, for $j = J-1$ and $\nu = 1$ we have $y_k = x_{(k+N/2) \bmod N}$, $k = 0, \ldots, N-1$, with

$$\widehat{x}_{2k} = \widehat{y}_{2k}, \qquad \widehat{x}_{2k+1} = -\widehat{y}_{2k+1}, \qquad k = 0, \ldots, N/2-1.$$

Proof. Using $\widehat{\mathbf{x}} = \mathbf{F}_N \mathbf{x} = (\widehat{x}_\ell)_{\ell=0}^{N-1}$ and $\widehat{\mathbf{y}} = \mathbf{F}_N \mathbf{y} = (\widehat{y}_\ell)_{\ell=0}^{N-1}$ we obtain

$$\widehat{y}_\ell = \sum_{k=0}^{N-1} y_k\, \omega_N^{\ell k} = \sum_{k=0}^{N-1} x_{(k+2^j\nu) \bmod N}\, \omega_N^{\ell k} = \sum_{k=0}^{N-1} x_k\, \omega_N^{\ell(k-2^j\nu)} = \omega_{2^{J-j}}^{-\ell\nu}\, \widehat{x}_\ell.$$

For $j = J-1$ and $\nu = 1$ the assertion follows by

$$\widehat{y}_\ell = \omega_{2^{J-j}}^{-\ell\nu} \widehat{x}_\ell = \omega_2^{-\ell}\, \widehat{x}_\ell = (-1)^\ell\, \widehat{x}_\ell. \qquad \square$$

4.2. Reconstructing vectors with small support from exact Fourier data

We describe the reconstruction of vectors $\mathbf{x} \in \mathbb{C}^N$ from exact Fourier data $\widehat{\mathbf{x}} = \mathbf{F}_N \mathbf{x}$. It is assumed that, in addition to the Fourier data $\widehat{\mathbf{x}}$, the support length $m < N$ of $\mathbf{x}$ or an upper bound for it is given. Then the following procedure can be applied to reconstruct $\mathbf{x}$ from $\widehat{\mathbf{x}}$.

As a first step, we compute $L := \lceil \log_2 m \rceil$, i.e., $L \in \mathbb{N}$ such that $2^{L-1} < m \leq 2^L$. Afterwards, the periodized vector $\mathbf{x}^{(L+1)}$ is determined from its Fourier transform $\widehat{\mathbf{x}}^{(L+1)}$ which, by Lemma 3.7, is a vector containing components of $\widehat{\mathbf{x}}$. Hence, we compose $\widehat{\mathbf{x}}^{(L+1)} = (\widehat{x}_{2^{J-(L+1)}k})_{k=0}^{2^{L+1}-1}$ and compute $\mathbf{x}^{(L+1)} = \mathbf{F}_{2^{L+1}}^{-1} \widehat{\mathbf{x}}^{(L+1)}$ using an inverse FFT algorithm.

The obtained periodization $\mathbf{x}^{(L+1)}$ of length 2^{L+1} already contains all support components of $\mathbf{x}$ in the right order and beyond that no further nonzero entries. This can be seen as follows: The components of $\mathbf{x}^{(L+1)}$ are given by

$$x_k^{(L+1)} = \sum_{\ell=0}^{2^{J-L-1}-1} x_{k+2^{L+1}\ell}, \qquad k = 0, \ldots, 2^{L+1} - 1. \tag{4.1}$$

As $|\operatorname{supp} \mathbf{x}| = m \leq 2^L$ holds for the support length, each of these sums for $k \in \{0, \ldots, 2^{L+1} - 1\}$ contains at most one nonvanishing summand. On the other hand, as the support length can only increase and never decrease with the periodizations $\mathbf{x}^{(0)}, \mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(J)}$, it holds that $|\operatorname{supp} \mathbf{x}^{(L+1)}| = m_{L+1} \leq m = |\operatorname{supp} \mathbf{x}|$. Moreover, due to the fact that $m_{L+1} \leq m \leq 2^L$, the support interval and the first support index $\mu^{(L+1)}$ of $\mathbf{x}^{(L+1)}$ are uniquely defined. This means that the support of $\mathbf{x}$ occurs in full length in the periodization $\mathbf{x}^{(L+1)}$.

Hence, the next reconstruction step consists in the identification of this support interval in $\mathbf{x}^{(L+1)}$. We compute the first support index $\mu^{(L+1)}$ of $\mathbf{x}^{(L+1)}$ with the help of local energies

$$e_k := \sum_{\ell=k}^{m+k-1} \left| x_{\ell \bmod 2^{L+1}}^{(L+1)} \right|^2 \tag{4.2}$$

for $k = 0, \ldots, 2^{L+1} - 1$. For exact data, the index $\mu^{(L+1)}$ is then uniquely defined as the index k for which the local energy e_k is maximal, i.e., $\mu^{(L+1)} = \operatorname{argmax}_k e_k$. In case that m only is an upper bound for the support length of $\mathbf{x}$, we choose one of the indices with

maximal local energy.

As a next step, the first support index $\mu = \mu^{(J)}$ of $\mathbf{x}$ has to be determined in order to place the support at the right position in $\mathbf{x}$. By (4.1) we conclude that $\mu^{(J)}$ is of the form $\mu^{(J)} = \mu^{(L+1)} + 2^{L+1}\nu$ for some $\nu \in \{0, 1, \ldots, 2^{J-L-1} - 1\}$. Thus, the value of the shift ν has to be computed which can be done very efficiently using one additional Fourier value.

Provided that the first support index $\mu^{(J)}$ of $\mathbf{x}$ index is known, $\mathbf{x}$ can be recovered with components

$$x_{(\mu^{(J)}+k) \bmod N} = \begin{cases} x^{(L+1)}_{(\mu^{(L+1)}+k) \bmod 2^{L+1}} & k = 0, \ldots, m-1, \\ 0 & k = m, \ldots, N-1. \end{cases}$$

The following theorem describes the reconstruction of $\mathbf{x}$ from $\mathbf{x}^{(L+1)}$, in particular the efficient determination of the shift ν.

Theorem 4.2 *Let $\mathbf{x} \in \mathbb{C}^N$, $N = 2^J$ for $J \in \mathbb{N}$, have support length $m \leq N/4$ (or a support length bounded by m) with $2^{L-1} < m \leq 2^L$. For $L < J - 1$, let the 2^{L+1}-periodization $\mathbf{x}^{(L+1)}$ of $\mathbf{x}$ be given. Then the vector $\mathbf{x}$ can be uniquely recovered from $\mathbf{x}^{(L+1)}$ and one nonzero component of the vector $(\widehat{x}_{2k+1})_{k=0}^{N/2-1}$.*

Proof. Let the 2^{L+1}-periodization $\mathbf{x}^{(L+1)}$ have the support interval

$$\{(\mu^{(L+1)} + r) \bmod 2^{L+1} \mid r = 0, \ldots, m-1\}.$$

As $m \leq 2^L$ holds by assumption, the support interval of $\mathbf{x}^{(L+1)}$ is uniquely defined. Applying the above considerations, we conclude from (4.1) that the first support index $\mu^{(J)} = \mu$ of $\mathbf{x}$ is of the form $\mu^{(J)} = \mu^{(L+1)} + 2^{L+1}\nu$ for some $\nu \in \{0, \ldots, 2^{J-L-1} - 1\}$. Hence, we aim to determine the shift ν.

First, we choose $\nu = 0$ and consider the vector $\mathbf{u}^0 \in \mathbb{C}^N$ with components

$$u^0_{(\mu^{(L+1)}+k) \bmod N} = \begin{cases} x^{(L+1)}_{(\mu^{(L+1)}+k) \bmod 2^{L+1}} & k = 0, \ldots, m-1, \\ 0 & k = m, \ldots, N-1. \end{cases}$$

The vector $\mathbf{u}^0$ has the first support index $\mu^{(L+1)}$ and is one of the possible solution vectors for $\mathbf{x}$ in $\mathbb{C}^N$ as it has support length m and the corresponding 2^{L+1}-periodization $\mathbf{x}^{(L+1)}$. All further possible vectors $\mathbf{u}^\nu$ in $\mathbb{C}^N$ with 2^{L+1}-periodization $(\mathbf{u}^\nu)^{(L+1)} = \mathbf{x}^{(L+1)}$

but different first support indices $\mu^{(L+1)} + 2^{L+1}\nu$ can be written in the form

$$\mathbf{u}^\nu := (u_k^\nu)_{k=0}^{N-1} \quad \text{with} \quad u_k^0 = u^\nu_{(k+2^{L+1}\nu) \bmod N}, \quad k = 0, \dots, N-1,$$

for some $\nu \in \{1, \dots, 2^{J-L-1} - 1\}$ and thus are shifted versions of $\mathbf{u}^0$. This implies that the vector $\mathbf{x}$, which we want to reconstruct, is contained in the set

$$\{\mathbf{u}^\nu \mid \nu = 0, \dots, 2^{J-L-1} - 1\}.$$

We aim to find ν such that $\mathbf{x} = \mathbf{u}^\nu$ holds. In order to determine the correct ν, we consider the Fourier transforms $\widehat{\mathbf{u}}^\nu = (\widehat{u}_\ell^\nu)_{\ell=0}^{N-1}$ of the shifted vectors $\mathbf{u}^\nu$. By Lemma 4.1, the components of these vectors are given by

$$\widehat{u}_\ell^\nu = \omega_{2^{J-L-1}}^{-\ell\nu} \widehat{u}_\ell^0.$$

Hence, for the correct ν, it holds that

$$\widehat{x}_\ell = \omega_{2^{J-L-1}}^{-\ell\nu} \widehat{u}_\ell^0 \qquad \text{for all } \ell = 0, \dots, N-1.$$

This means that one component $\widehat{x}_\ell$ of the Fourier transform $\widehat{\mathbf{x}}$ suffices to compute ν from this equation if we choose the index ℓ appropriately. For an odd-indexed component $\widehat{x}_{2k_0+1}$ of $\widehat{\mathbf{x}}$, the numbers $(2k_0 + 1)$ and 2^{J-L-1} are mutually prime, i.e., $\gcd(2k_0 + 1, 2^{J-L-1}) = 1$, such that $\nu \in \{0, \dots, 2^{J-L-1} - 1\}$ can be uniquely recovered from $\omega_{2^{J-L-1}}^{-(2k_0+1)\nu}$.

Hence, we choose one nonzero odd-indexed Fourier value $\widehat{x}_{2k_0+1}$ and compare it to the $(2k_0 + 1)$-th component

$$\widehat{u}_{2k_0+1}^0 = \sum_{r=0}^{m-1} x_{(\mu^{(L+1)}+r) \bmod 2^{L+1}} \; \omega_N^{(\mu^{(L+1)}+r)(2k_0+1)}$$

of $\widehat{\mathbf{u}}^0$. Then ν is obtained from

$$\omega_{2^{J-L-1}}^{-(2k_0+1)\nu} = \frac{\widehat{x}_{2k_0+1}}{\widehat{u}_{2k_0+1}^0}$$

and we recover $\mathbf{x}$ as $\mathbf{x} = \mathbf{u}^\nu$. □

Remark 4.3 *The existence of a nonzero component in the vector $(\widehat{x}_{2k+1})_{k=0}^{N/2-1}$ is shown in Lemma 4.5 below.*

We illustrate the reconstruction of a vector $\mathbf{x}$ using the presented techniques in a small example.

Example 4.4 Let $N = 8 = 2^3$, i.e., $J = 3$.
Choose $\widehat{\mathbf{x}} = \left(2, 1+\omega_8, 1+\omega_8^2, \ldots, 1+\omega_8^7\right)^T$.
This given $\widehat{\mathbf{x}}$ corresponds to $\mathbf{x} := \left(1,1,0,0,0,0,0,0\right)^T$ with support length $m = 2$, $L = \lceil \log_2 2 \rceil = 1$ and support interval $I = \{0,1\}$ as we will now see.

We first compose the vector

$$\widehat{\mathbf{x}}^{(L+1)} = \widehat{\mathbf{x}}^{(2)} = \begin{pmatrix} 2 \\ 1+\omega_8^2 \\ 1+\omega_8^4 \\ 1+\omega_8^6 \end{pmatrix} = \begin{pmatrix} 2 \\ 1-\mathrm{i} \\ 0 \\ 1+\mathrm{i} \end{pmatrix}$$

of length $2^2 = 4$ and then compute

$$\mathbf{x}^{(2)} = \mathbf{F}_4^{-1}\widehat{\mathbf{x}}^{(2)} = \frac{1}{4}\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & \mathrm{i} & -1 & -\mathrm{i} \\ 1 & -1 & 1 & -1 \\ 1 & -\mathrm{i} & -1 & \mathrm{i} \end{pmatrix}\begin{pmatrix} 2 \\ 1-\mathrm{i} \\ 0 \\ 1+\mathrm{i} \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \end{pmatrix}.$$

This periodization has first support index $\mu^{(2)} = 0$. This means that the support index μ of $\mathbf{x}$ has to be of the form $\mu = \mu^{(3)} = \mu^{(2)} + 4\nu$ with $\nu \in \{0,1\}$, i.e., $\mu^{(3)} = 0$ or $\mu^{(3)} = 4$. We choose the first odd-indexed Fourier component $\widehat{x}_1 = 1 + \omega_8$ for comparison. Using the notation of the proof of Theorem 4.2, we compute

$$\widehat{u}_1^0 = \sum_{\ell=0}^{m-1} x^{(2)}_{(\mu^{(2)}+\ell) \bmod 2^2}\, \omega_8^{\mu^{(2)}+\ell} = 1 \cdot 1 + 1 \cdot \omega_8 = 1 + \omega_8.$$

Comparing to the Fourier value $\widehat{x}_1$ yields

$$\omega_2^{-\nu} = \frac{\widehat{x}_1}{\widehat{u}_1^0} = \frac{1+\omega_8}{1+\omega_8} = 1$$

with the unique solution $\nu = 0$. Hence, we obtain the first support index $\mu = \mu^{(2)} + 4\nu = 0$ and therefore for the reconstruction of $\mathbf{x}$ the correct solution

$$\mathbf{x} = \left(1,1,0,0,0,0,0,0\right)^T.$$

For the reconstruction, only 5 of 8 Fourier values were used.

It remains to prove that the vector $(\widehat{x}_{2k+1})_{k=0}^{N/2-1}$ has a nonzero component which can be chosen for comparison in Theorem 4.2 in order to obtain the index $\mu^{(J)}$.

Lemma 4.5 *Let $\mathbf{x} \in \mathbb{C}^N \setminus \{\mathbf{0}\}$, $N = 2^J \geq 4$ for $J \in \mathbb{N}$, and let $\mathbf{x}$ have support length $|\operatorname{supp} \mathbf{x}| \leq m \leq N/4$. Then the vector $(\widehat{x}_{2k+1})_{k=0}^{N/2-1}$, which contains all odd-indexed components of $\widehat{\mathbf{x}}$, has at least one nonzero component.*

Proof. Let $\mu^{(J)}$ be the first support index of $\mathbf{x}$ and thus

$$I = \{\mu^{(J)}, (\mu^{(J)} + 1) \bmod N, \ldots, (\mu^{(J)} + m - 1) \bmod N\}$$

is the support interval of $\mathbf{x}$. We define the trigonometric polynomial

$$p(\omega) = \left| \sum_{k=0}^{N-1} x_k \mathrm{e}^{-\mathrm{i}\omega k} \right|^2 = \left| \mathrm{e}^{-\mathrm{i}\omega\mu^{(J)}} \sum_{\ell=0}^{m-1} x_{(\mu^{(J)}+\ell) \bmod N} \mathrm{e}^{-\mathrm{i}\omega\ell} \right|^2 \tag{4.3}$$

which is real, nonnegative and of degree $\leq m - 1$. This implies that $p(\omega)$ has at most $m - 1$ pairwise different roots and all of these roots occur at least twice. We observe that

$$|\widehat{x}_k|^2 = \left| \sum_{\ell=0}^{N-1} x_\ell \mathrm{e}^{-\frac{2\pi\mathrm{i}k\ell}{N}} \right|^2 = p\left(\frac{2\pi k}{N}\right), \quad k = 0, \ldots, N-1.$$

As $p(\omega)$ has at most $m - 1$ roots and $m \leq N/4$, this implies that not all components $\widehat{x}_{2k+1}$, $k = 0, \ldots, N/2 - 1$, can be zero. □

Remark 4.6 *An alternative proof of Lemma 4.5 is the following. Assume that all odd components of $\widehat{\mathbf{x}}$ are zero, i.e., $(\widehat{x}_{2k+1})_{k=0}^{N/2-1} = 0$. Then we obtain for the components x_k of $\mathbf{x}$*

$$\begin{aligned} x_k &= \frac{1}{N} \sum_{\ell=0}^{N-1} \widehat{x}_\ell \, \omega_N^{-k\ell} = \frac{1}{2^J} \sum_{\ell=0}^{2^{J-1}-1} \widehat{x}_{2\ell} \, \omega_{2^J}^{-k2\ell} = \frac{1}{2^J} \sum_{\ell=0}^{2^{J-1}-1} \widehat{x}_\ell^{(J-1)} \, \omega_{2^{J-1}}^{-k\ell} \\ &= \begin{cases} \frac{1}{2} x_k^{(J-1)} & 0 \leq k \leq 2^{J-1} - 1, \\ \frac{1}{2} x_{k-2^{J-1}}^{(J-1)} & 2^{J-1} - 1 \leq k \leq 2^J - 1. \end{cases} \end{aligned}$$

This would imply that all nonzero components of the periodization $\mathbf{x}^{(J-1)}$ split up into two entries of equal value with distance 2^{J-1} when reconstructing $\mathbf{x}$ and therefore $m > N/2$. But this is a contradiction to the assumption $m \leq N/4 = 2^{J-2}$.

We have seen in Lemma 4.5 that the vector $(\widehat{x}_{2k+1})_{k=0}^{N/2-1}$ has at most $m-1$ zero components and it therefore always has several nonzero components of which we can choose one in order to compute the first support index $\mu^{(J)}$ of $\mathbf{x}$ as given in Theorem 4.2. Such a nonzero component can be found by at most $m-1$ comparisons. However, it would be an asset to pick a Fourier value with large modulus for a stable computation.

Remark 4.7 *In order to choose an odd Fourier component $\widehat{x}_{2k_0+1}$ of large modulus in Theorem 4.2, it would be best to determine*

$$k_0 := \operatorname{argmax} \left\{ |\widehat{x}_{2k+1}|^2 \;\middle|\; k = 0, \ldots, N/2-1 \right\}.$$

As this would imply that we have to compare a large number of values and is therefore very costly, we can instead first identify the index of the component of largest modulus in the periodization $\widehat{\mathbf{x}}^{(L+1)}$,

$$k_0^{(L+1)} := \operatorname{argmax} \left\{ |\widehat{x}_{2^{J-(L+1)}k}|^2 \;\middle|\; k = 0, \ldots, 2^{L+1}-1 \right\}.$$

Recalling that $2^{L-1} < m \leq 2^L$ holds, the index $k_0^{(L+1)}$ can be obtained by $\mathcal{O}(m)$ operations. Using the notation of the trigonometric polynomial (4.3), we obtain for this entry

$$\left| \widehat{x}_{2^{J-(L+1)}k_0^{(L+1)}} \right|^2 = p\left(\frac{2\pi k_0^{(L+1)}}{2^{L+1}} \right) = \max_{k=0,\ldots,2^{L+1}-1} p\left(\frac{2\pi k}{2^{L+1}} \right) > 0,$$

and we can assume that $p\left(\frac{2\pi k_0^{(L+1)}}{2^{L+1}}\right)$ is close to the global maximum $\|p\|_\infty$ of $p(\omega)$. As a Fourier value for comparison, we can now choose the odd-indexed Fourier value with largest modulus neighboring to $\widehat{x}_{2^{J-(L+1)}k_0^{(L+1)}}$: either $\widehat{x}_{2^{J-(L+1)}k_0^{(L+1)}+1}$ or $\widehat{x}_{2^{J-(L+1)}k_0^{(L+1)}-1}$.

We give an estimate for the modulus of these two Fourier values. Assume that $p(\omega)$ attains its global maximum at some point

$$\omega_0 \in \left[\frac{2\pi(k_0^{(L+1)} - \frac{1}{2})}{2^{L+1}}, \frac{2\pi(k_0^{(L+1)} + \frac{1}{2})}{2^{L+1}} \right),$$

then for this extremal point it holds that $\left| \omega_0 - \frac{2\pi k_0^{(L+1)}}{2^{L+1}} \right| \leq \frac{\pi}{2^{L+1}}$. If we assume that we

have chosen one of the above values for $\widehat{x}_{2k_0+1}$*, i.e., either* $k_0 = 2^{J-L-2}k_0^{(L+1)} + 1$ *or* $k_0 = 2^{J-L-2}k_0^{(L+1)} - 1$*, it follows that* $\left|\omega_0 - \frac{2\pi(2k_0+1)}{N}\right| \leq \frac{\pi}{2^{L+1}}$*. By the Lemma of Stečkin, which can be found e.g. in [19], we obtain the worst case estimate*

$$|\widehat{x}_{2k_0+1}|^2 = p\left(\frac{2\pi(2k_0+1)}{N}\right) \geq \|p\|_\infty \cos\left(\frac{\pi(m-1)}{2^{L+1}}\right) > 0$$

for the lower bound of the modulus of $\widehat{x}_{2k_0+1}$*.*

The procedure of recovering $\mathbf{x}$ from exact Fourier data $\widehat{\mathbf{x}}$ is summarized in Algorithm 4.8.

Algorithm 4.8 (Sparse FFT for vectors with small support for exact Fourier data)
Input: $\widehat{\mathbf{x}} \in \mathbb{C}^N$, $N = 2^J$, $|\operatorname{supp}\mathbf{x}| \leq m < N$.

- Compute L such that $2^{L-1} < m \leq 2^L$, i.e., $L := \lceil \log_2 m \rceil$.
- If $L = J$ or $L = J-1$, compute $\mathbf{x} = \mathbf{F}_N^{-1}\widehat{\mathbf{x}}$ using an inverse FFT algorithm of length N.
- If $L < J-1$:
  1. Set $\widehat{\mathbf{x}}^{(L+1)} := (\widehat{x}_{2^{J-(L+1)}k})_{k=0}^{2^{L+1}-1}$ and compute
  $$\mathbf{x}^{(L+1)} := \mathbf{F}_{2^{L+1}}^{-1}\widehat{\mathbf{x}}^{(L+1)}$$
  using an FFT algorithm for the inverse discrete Fourier transform of length 2^{L+1}.
  2. Determine the first support index $\mu^{(L+1)} \in \{0, \ldots, 2^{L+1}-1\}$ of $\mathbf{x}^{(L+1)}$ such that $x_{\mu^{(L+1)}}^{(L+1)} \neq 0$ and $x_k^{(L+1)} = 0$ for $k \notin \{(\mu^{(L+1)}+r) \bmod 2^{L+1} \mid r = 0, \ldots, m-1\}$.
  3. Choose a Fourier component $\widehat{x}_{2k_0+1} \neq 0$ of $\widehat{\mathbf{x}}$ and compute the sum
  $$\widehat{u}_{2k_0+1}^0 := \sum_{\ell=0}^{m-1} x_{(\mu^{(L+1)}+\ell) \bmod 2^{L+1}}^{(L+1)} \omega_N^{(2k_0+1)(\mu^{(L+1)}+\ell)}.$$
  4. Compute the quotient $b := \widehat{x}_{2k_0+1}/\widehat{u}_{2k_0+1}^0$ which is by construction of the form $b = \omega_{2^{J-L-1}}^p$ for some $p \in \{0, \ldots, 2^{J-L-1}-1\}$ and find $\nu \in \{0, \ldots, 2^{J-L-1}-1\}$ such that $(2k_0+1)\,\nu = p \bmod 2^{J-L-1}$.

5. Set $\mu^{(J)} := \mu^{(L+1)} + 2^{L+1}\nu$, and $\mathbf{x} := (x_k)_{k=0}^{N-1}$ with entries

$$x_{(\mu^{(J)}+\ell) \bmod N} := \begin{cases} x^{(L+1)}_{(\mu^{(L+1)}+\ell) \bmod 2^{L+1}} & \ell = 0, \ldots, m-1, \\ 0 & \ell = m, \ldots, N-1. \end{cases}$$

Output: $\mathbf{x}$.

The algorithm for the reconstruction of a vector $\mathbf{x} \in \mathbb{C}^N$ from exact data has an overall arithmetical complexity of $\mathcal{O}(m \log m)$ if $m \leq N/4$. Let us consider the complexity of the individual steps. Initially, the algorithm executes an inverse Fourier transform of length $2^{L+1} < 4m$ using an inverse FFT algorithm. This requires $\mathcal{O}(m \log m)$ arithmetical operations. The following steps require $\mathcal{O}(m)$ (or less) arithmetical operations each: Determining the support interval of the periodization $\mathbf{x}^{(L+1)}$ can be efficiently done using the local energies (4.2). Computing the first local energy e_0 requires $\mathcal{O}(m)$ arithmetical operations. Then any other energy can be obtained by the recursion $e_{k+1} = e_k - |x_k|^2 + |x_{(k+m) \bmod 2^{L+1}}|^2$ for $k = 0, \ldots, 2^{L+1} - 2$. Thus, the overall complexity for the identification of $\mu^{(L+1)}$ remains $\mathcal{O}(m)$ since $\mathbf{x}^{(L+1)}$ has at most m nonzero components.

The computation of the shift ν is done using the component $\widehat{u}^0_{2k_0+1}$ which is obtained by $\mathcal{O}(m)$ multiplications and additions. Finally, the vector $\mathbf{x}$ is reconstructed by plugging in the (at most) m support components according to the first support index $\mu^{(J)}$.

The number of Fourier values applied for reconstruction is bounded by $4m$. In particular, the first inverse Fourier transform requires 2^{L+1} values and one additional Fourier value is employed for comparison in order to compute the shift ν.

4.3. Reconstructing vectors with small support from noisy Fourier data

The newly developed algorithm can also be applied to noisy data. For this purpose, several steps of Algorithm 4.8 have to be stabilized. Assume from now on that perturbed Fourier data $\widehat{\mathbf{y}} = (\widehat{y}_k)_{k=0}^{N-1} \in \mathbb{C}^N$

$$\widehat{y}_k = \widehat{x}_k + \boldsymbol{\varepsilon}_k$$

is given where $\boldsymbol{\varepsilon} = (\varepsilon_k)_{k=0}^{N-1} \in \mathbb{C}^N$ denotes a noise vector. We want to reconstruct the vector $\mathbf{x} \in \mathbb{C}^N$ from the perturbed Fourier data $\widehat{\mathbf{y}}$ under the assumption that $\mathbf{x}$ has a small support interval of length $m \leq N/4$.

We recapitulate the steps of Algorithm 4.8 and propose stabilizations for the following crucial reconstruction steps:

1. the correct identification of the support interval of $\mathbf{x}^{(L+1)}$, i.e., determining the first support index $\mu^{(L+1)}$ of $\mathbf{x}^{(L+1)}$,

2. the correct identification of the support interval of $\mathbf{x}$, i.e., determining the first support index $\mu^{(J)}$ of $\mathbf{x}$, and

3. the evaluation of the nonzero components within the support of $\mathbf{x}$.

All these steps can be improved by employing more Fourier values of $\widehat{\mathbf{y}}$ than in the case of exact data.

4.3.1. Stable identification of the support interval of $\mathbf{x}^{(L+1)}$

We stabilize the determination of the support interval of $\mathbf{x}^{(L+1)}$ by evaluating additional Fourier values. Let us first define shifted vectors of Fourier data.

Definition 4.9 *Let $\mathbf{x} \in \mathbb{C}^N$, $N = 2^J$ for $J > 0$, and let $\mathbf{x}$ have support length $|\operatorname{supp} \mathbf{x}| \leq m \leq N/4$ with $2^{L-1} < m \leq 2^L$. For $L < J-1$, let $\mathbf{x}^{(L+1)} = (x_k^{(L+1)})_{k=0}^{2^{L+1}-1}$ be the 2^{L+1}-periodization of $\mathbf{x}$ and $\widehat{\mathbf{x}}^{(L+1)} = (\widehat{x}_{2^{J-L-1}k})_{k=0}^{2^{L+1}-1}$ be its Fourier transform. Then define shifted versions of the vector $\widehat{\mathbf{x}}^{(L+1)}$ by*

$$\widehat{\mathbf{z}}^{(\kappa)} := (\widehat{x}_{2^{J-L-1}k+\kappa})_{k=0}^{2^{L+1}-1}, \qquad \kappa = 0, \dots, 2^{J-L-1} - 1.$$

In particular, this implies that $\mathbf{z}^{(0)} = \mathbf{x}^{(L+1)}$. In case of noisy Fourier data $\widehat{\mathbf{y}} = \widehat{\mathbf{x}} + \boldsymbol{\varepsilon}$, we analogously define the shifted vectors

$$\widehat{\widetilde{\mathbf{z}}}^{(\kappa)} := (\widehat{y}_{2^{J-L-1}k+\kappa})_{k=0}^{2^{L+1}-1}, \qquad \kappa = 0, \dots, 2^{J-L-1} - 1.$$

The following theorem states that, by considering the inverse Fourier transform of the shifted versions $\widehat{\mathbf{z}}^{(\kappa)}$ of $\widehat{\mathbf{x}}^{(L+1)}$, we obtain vectors whose components have the same modulus as the components of $\mathbf{x}^{(L+1)}$.

Theorem 4.10 *Let $\mathbf{x} \in \mathbb{C}^N$, $N = 2^J$ for $J > 0$, and let $\mathbf{x}$ have support length $|\operatorname{supp} \mathbf{x}| \leq m \leq N/4$ with $2^{L-1} < m \leq 2^L$. For $L < J-1$, let $\widehat{\mathbf{z}}^{(\kappa)}$, $\kappa = 0, \dots, 2^{J-L-1} - 1$, be given as in Definition 4.9 and particularly $\widehat{\mathbf{x}}^{(L+1)} = \widehat{\mathbf{z}}^{(0)}$. Then the inverse Fourier transform*

$\mathbf{z}^{(\kappa)} = \left(z_\ell^{(\kappa)}\right)_{\ell=0}^{2^{L+1}-1} = \mathbf{F}_{2^{L+1}}^{-1}\widehat{\mathbf{z}}^{(\kappa)}$ *of the shifted vectors* $\widehat{\mathbf{z}}^{(\kappa)}$, $\kappa = 0, \dots, 2^{J-L-1}-1$, *satisfies*

$$\left|z_\ell^{(\kappa)}\right| = \left|x_\ell^{(L+1)}\right|, \qquad \ell = 0, \dots, 2^{L+1}-1.$$

Proof. By definition, we obtain for the components $z_\ell^{(\kappa)}$ of $\mathbf{z}^{(\kappa)}$

$$\begin{aligned} z_\ell^{(\kappa)} &= \frac{1}{2^{L+1}} \sum_{k=0}^{2^{L+1}-1} \widehat{x}_{2^{J-L-1}k+\kappa}\, \omega_{2^{L+1}}^{-k\ell} \\ &= \frac{1}{2^{L+1}} \sum_{k=0}^{2^{L+1}-1} \sum_{r=0}^{N-1} x_r\, \omega_N^{r(2^{J-L-1}k+\kappa)}\, \omega_{2^{L+1}}^{-k\ell} = \frac{1}{2^{L+1}} \sum_{r=0}^{N-1} x_r\, \omega_N^{r\kappa} \sum_{k=0}^{2^{L+1}-1} \omega_{2^{L+1}}^{k(r-\ell)} \\ &= \sum_{j=0}^{2^{J-L-1}-1} x_{\ell+2^{L+1}j}\, \omega_N^{(\ell+2^{L+1}j)\kappa} = \omega_N^{\ell\kappa} \sum_{j=0}^{2^{J-L-1}-1} x_{\ell+2^{L+1}j}\, \omega_N^{2^{L+1}j\kappa}. \end{aligned}$$

For the second last equation, we have used that

$$\sum_{k=0}^{2^{L+1}-1} \omega_{2^{L+1}}^{k(r-\ell)} = \begin{cases} 2^{L+1} & r \equiv \ell \bmod 2^{L+1}, \\ 0 & r \not\equiv \ell \bmod 2^{L+1}. \end{cases}$$

Since $|\operatorname{supp}\mathbf{x}| \le m < 2^{L+1}$, for each index ℓ the above sum contains at most one summand, thus $\left|z_\ell^{(\kappa)}\right| = \left|x_\ell^{(L+1)}\right|$. □

Considering the vectors $\mathbf{z}^{(\kappa)}$ for $\kappa = 0, \dots, 2^{J-L-1}-1$, we observe that each of these vectors is reconstructed from different Fourier components. We use this fact to stabilize the computation of the values $\left|y_\ell^{(L+1)}\right|^2$ in case of noisy data.

Similarly to the case of exact data, we begin the reconstruction by picking the vector $\widetilde{\widehat{\mathbf{z}}}^{(0)} = (\widehat{y}_{2^{J-L-1}k})_{k=0}^{2^{L+1}-1}$ of noisy measurements and compute $\widetilde{\mathbf{z}}^{(0)} = \mathbf{F}_{2^{L+1}}^{-1}\widetilde{\widehat{\mathbf{z}}}^{(0)} = \mathbf{y}^{(L+1)}$. Then we consider the local energies

$$\widetilde{e}_k^{(0)} := \sum_{\ell=k}^{m+k-1} \left|y_{\ell \bmod 2^{L+1}}^{(L+1)}\right|^2 = \sum_{\ell=k}^{m+k-1} \left|\widetilde{z}_{\ell \bmod 2^{L+1}}^{(0)}\right|^2, \qquad k = 0, \dots, 2^{L+1}-1,$$

of $\widetilde{\mathbf{z}}^{(0)}$. This gives us a first estimate

$$\mu_0^{(L+1)} = \operatorname*{argmax}_k \widetilde{e}_k^{(0)}$$

for the first support index $\mu^{(L+1)}$ of the periodization $\mathbf{x}^{(L+1)}$.

For a further stabilization in case of strongly perturbed data, we now take advantage of additional vectors with shifted Fourier values. We consider in a next step for $\kappa = 2^{J-L-2}$ the vector $\widehat{\widetilde{\mathbf{z}}}^{(2^{J-L-2})} = (\widehat{y}_{2^{J-L-2}(2k+1)})_{k=0}^{2^{L+1}-1}$ and obtain $\widetilde{\mathbf{z}}^{(2^{J-L-2})}$ by inverse FFT. This vector has the local energies

$$\widetilde{e}_k^{(1)} := \sum_{\ell=k}^{m+k-1} \left| \widetilde{z}_{\ell \bmod 2^{L+1}}^{(2^{J-L-2})} \right|^2, \qquad k = 0, \dots, 2^{L+1} - 1.$$

Hence, we can average the results of this vector with the previous ones and obtain

$$\mu_1^{(L+1)} = \operatorname*{argmax}_k \frac{1}{2} \left(\widetilde{e}_k^{(0)} + \widetilde{e}_k^{(1)} \right)$$

as a new estimate for $\mu^{(L+1)}$. If $\mu_1^{(L+1)} = \mu_0^{(L+1)}$, we conclude that the estimates were correct. Hence, we set $\mu^{(L+1)} = \mu_1^{(L+1)} = \mu_0^{(L+1)}$ for the first support index of the periodization $\mathbf{x}^{(L+1)}$ and proceed with further reconstruction steps. Otherwise, we repeat the procedure with an additional vector of shifted Fourier data.

Choose e.g. $\kappa = 2^{J-L-3}$ and compute $\widetilde{\mathbf{z}}^{(2^{J-L-3})} = \mathbf{F}_{2^{L+1}}^{-1}(\widehat{y}_{2^{J-L-3}(4k+1)})_{k=0}^{2^{L+1}-1}$. For this vector, we obtain local energies

$$\widetilde{e}_k^{(2)} := \sum_{\ell=k}^{m+k-1} \left| \widetilde{z}_{\ell \bmod 2^{L+1}}^{(2^{J-L-3})} \right|^2, \qquad k = 0, \dots, 2^{L+1} - 1,$$

and hence a third estimate

$$\mu_2^{(L+1)} = \operatorname*{argmax}_k \frac{1}{3} \left(\widetilde{e}_k^{(0)} + \widetilde{e}_k^{(1)} + \widetilde{e}_k^{(2)} \right)$$

for $\mu^{(L+1)}$. This procedure can be repeated for all possible choices of κ. In practice, we stop the iteration if the same index has been obtained in two consecutive steps.

4.3.2. Stable identification of the support interval of $\mathbf{x}$

After determining the correct support interval of the periodization $\mathbf{x}^{(L+1)}$, the next reconstruction step is to identify the first support index $\mu = \mu^{(J)}$ of the vector $\mathbf{x}$. We have seen in Section 4.2 that $\mu^{(J)}$ is of the form $\mu^{(J)} = \mu^{(L+1)} + 2^{L+1}\nu$ for some $\nu \in \{0, \dots, 2^{J-L-1} - 1\}$.

In contrast to the case of exact data, we proceed here by iteratively computing the indices $\mu^{(j)}$, $j = L+2, \dots, J$, i.e., we use the knowledge of the first support index $\mu^{(j)}$

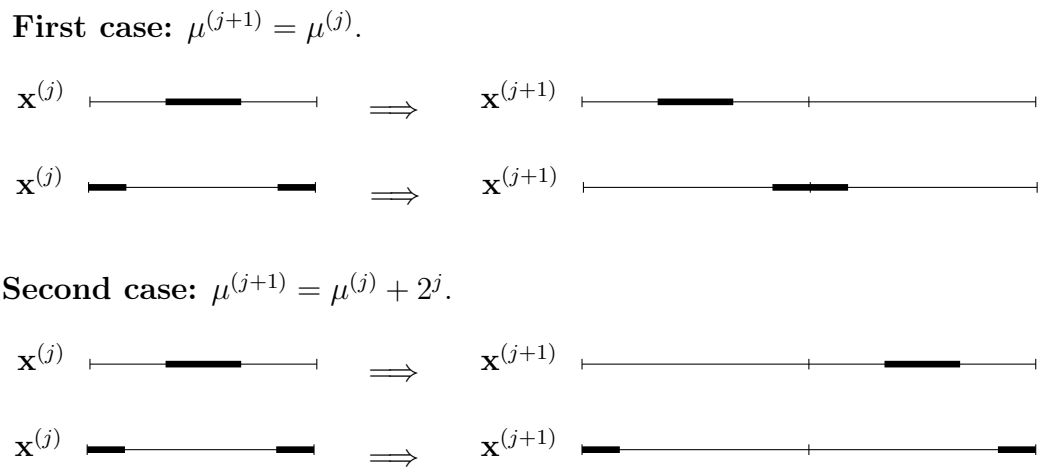


Figure 4.1.: Possible support change in one iteration step.

of $\mathbf{x}^{(j)}$ to compute the first support index $\mu^{(j+1)}$ of the periodization $\mathbf{x}^{(j+1)}$ of double length for $j = L+1, \ldots, J-1$.

As $m \leq 2^{L+1}$, we know that all vectors $\mathbf{x}^{(j)}$, $j = L+1, \ldots, J-1$, have the same support length as $\mathbf{x}^{(L+1)}$. Additionally, for the components of the periodizations $\mathbf{x}^{(j)}$ and $\mathbf{x}^{(j+1)}$, $j = L+1, \ldots, J-1$, it holds that

$$x_k^{(j+1)} + x_{k+2^j}^{(j+1)} = x_k^{(j)}, \qquad k = 0, \ldots, 2^j - 1.$$

This means that we only have two possibilities for the first support index when reconstructing $\mu^{(j+1)}$ from $\mu^{(j)}$. Either the index stays the same or it is shifted by 2^j, i.e., we obtain either $\mu^{(j+1)} = \mu^{(j)}$ or $\mu^{(j+1)} = \mu^{(j)} + 2^j$. Both possible cases are illustrated in Figure 4.1, depending on the position of the support interval in $\mathbf{x}^{(j)}$.

The procedure of reconstructing $\mathbf{x}^{(j+1)}$ from $\mathbf{x}^{(j)}$ is described in the following theorem.

Theorem 4.11 *Let $\mathbf{x} \in \mathbb{C}^N$, $N = 2^J$, have support of length $|\operatorname{supp} \mathbf{x}| \leq m \leq N/4$ with $2^{L-1} < m \leq 2^L$. Further, denote by $\mathbf{x}^{(j)}$, $L+1 \leq j \leq J-1$, the periodizations of $\mathbf{x} = \mathbf{x}^{(J)}$ as given in Definition 3.3. Then, for each $j = L+1, \ldots, J-1$, the vector $\mathbf{x}^{(j+1)}$ can be uniquely recovered from $\mathbf{x}^{(j)}$ and one additional nonzero component of the vector of Fourier values $(\widehat{x}_{2^{J-(j+1)}(2k+1)})_{k=0}^{2^j-1}$.*

Proof. Let $\mathbf{x}^{(j)}$ be the given periodization of length 2^j. Its first support index is denoted by $\mu^{(j)}$ and hence the vector has the support $\operatorname{supp} \mathbf{x}^{(j)} = \{(\mu^{(j)}+r) \bmod 2^j \mid r = 0, \ldots, m-1\}$.

In order to recover $\mathbf{x}^{(j+1)}$ we need to detect if $\mu^{(j+1)} = \mu^{(j)}$ or $\mu^{(j+1)} = \mu^{(j)} + 2^j$. We denote the two possible vectors for $\mathbf{x}^{(j+1)}$ by $\mathbf{u}^{(0)} = \left(u_\ell^{(0)}\right)_{\ell=0}^{2^{j+1}-1}$ with components

$$u_{\mu^{(j)}+\ell}^{(0)} = \begin{cases} x_{(\mu^{(j)}+\ell) \bmod 2^j} & \ell = 0, \dots, m-1, \\ 0 & \text{else}, \end{cases}$$

resp. $\mathbf{u}^{(1)} = \left(u_\ell^{(1)}\right)_{\ell=0}^{2^{j+1}-1}$ with components

$$u_{(\mu^{(j)}+2^j+\ell) \bmod 2^{j+1}}^{(1)} = \begin{cases} x_{(\mu^{(j)}+\ell) \bmod 2^j} & \ell = 0, \dots, m-1, \\ 0 & \text{else}. \end{cases}$$

Thus, it holds that $\mathbf{x}^{(j+1)} = \mathbf{u}^{(0)}$ if $\mu^{(j+1)} = \mu^{(j)}$ and $\mathbf{x}^{(j+1)} = \mathbf{u}^{(1)}$ if $\mu^{(j+1)} = \mu^{(j)} + 2^j$, i.e., if the support interval is shifted by 2^j.

We observe that both vectors $\mathbf{u}^{(0)}$ and $\mathbf{u}^{(1)}$ only differ by a shift of all components by 2^j. Hence, we can apply Lemma 4.1 and obtain for the Fourier transforms $\widehat{\mathbf{u}}^{(0)} = \left(\widehat{u}_\ell^{(0)}\right)_{\ell=0}^{2^{j+1}-1}$ and $\widehat{\mathbf{u}}^{(1)} = \left(\widehat{u}_\ell^{(1)}\right)_{\ell=0}^{2^{j+1}-1}$ the relationship

$$\widehat{u}_{2k+1}^{(0)} = -\widehat{u}_{2k+1}^{(1)}, \qquad k = 0, \dots, 2^j - 1.$$

This implies that the Fourier transforms of both possible vectors only differ in the sign of all odd-indexed components. By comparison with one additional odd-indexed nonzero Fourier value $\widehat{x}_{2^{J-(j+1)}(2k_0+1)} = \widehat{x}_{2k_0+1}^{(j+1)}$, we can determine the correct solution for $\mathbf{x}^{(j+1)}$. For that purpose, compute one nonzero entry

$$\widehat{u}_{2k_0+1}^{(0)} = \sum_{\ell=0}^{m-1} x_{(\mu^{(j)}+\ell) \bmod 2^j}^{(j)} \omega_{2^{j+1}}^{(2k_0+1)(\ell+\mu^{(j)})}$$

of $\widehat{\mathbf{u}}^{(0)}$ and compare it to the Fourier value $\widehat{x}_{2^{J-(j+1)}(2k_0+1)}$. If

$$\left|\widehat{u}_{2k_0+1}^{(0)} - \widehat{x}_{2^{J-(j+1)}(2k_0+1)}\right| < \left|\widehat{u}_{2k_0+1}^{(0)} + \widehat{x}_{2^{J-(j+1)}(2k_0+1)}\right|,$$

then we set $\mathbf{x}^{(j+1)} = \mathbf{u}^{(0)}$ and $\mu^{(j+1)} = \mu^{(j)}$. Otherwise, we set $\mathbf{x}^{(j+1)} = \mathbf{u}^{(1)}$ with shifted index $\mu^{(j+1)} = \mu^{(j)} + 2^j$. □

4.3.3. Evaluation of the nonzero components of x

The reconstruction of the vector $\mathbf{x}$ is now basically completed as the first support index $\mu^{(L+1)}$ of $\mathbf{x}^{(L+1)}$ has been stably determined and the first support indices of the periodizations have been identified step-by-step. However, we can still improve the accuracy of the nonzero components by taking advantage of the vectors $\widetilde{\mathbf{z}}^{(\kappa)} \in \mathbb{C}^{2^{L+1}}$ that were already computed in Subsection 4.3.1 for the determination of the support of $\mathbf{x}^{(L+1)}$. From the proof of Theorem 4.10, we can conclude that

$$\widetilde{z}^{(\kappa)}_{\ell \bmod 2^{L+1}} = \omega_N^{\ell\kappa} \left(x_{(\ell+2^{L+1}\nu) \bmod N}\, \omega_N^{2^{L+1}\nu\kappa}\right), \qquad \ell = \mu^{(L+1)}, \ldots, \mu^{(L+1)} + m - 1,$$

holds for the components of $\widetilde{\mathbf{z}}^{(\kappa)}$. The shift ν has been iteratively computed in the preceding subsection, such that the components x_k of $\mathbf{x}$ are implicitly given by the entries of the vectors $\widetilde{\mathbf{z}}^{(\kappa_r)}$ for each κ_r which was considered in Subsection 4.3.1.

Using the relation $\mu^{(J)} = \mu^{(L+1)} + 2^{L+1}\nu$ we can reformulate the above equation to

$$\widetilde{z}^{(\kappa)}_{(\mu^{(L+1)}+k) \bmod 2^{L+1}} = x_{(\mu^{(J)}+k) \bmod N}\, \omega_N^{\kappa(\mu^{(J)}+k)}, \qquad k = 0, \ldots, m-1.$$

We average over all measurements in order to compute the support entries $x_{(\mu^{(J)}+k) \bmod N}$, $k = 0, \ldots, m-1$, of $\widehat{\mathbf{x}}$ and obtain an improved estimate

$$x_{(\mu^{(J)}+k) \bmod N} = \frac{1}{B+1} \sum_{r=0}^{B} \widetilde{z}^{(\kappa_r)}_{(\mu^{(L+1)}+k) \bmod 2^{L+1}}\, \omega_N^{-\kappa_r(\mu^{(J)}+k)}, \qquad k = 0, \ldots, m-1,$$

where $B+1$ is the number of vectors

$$\widetilde{\mathbf{z}}^{(\kappa_r)} = \mathbf{F}^{-1}_{2^{L+1}} \left(\widehat{y}_{2^{J-L-1}k+\kappa_r}\right)_{k=0}^{2^{L+1}-1}, \qquad \kappa_r \in \{0, \ldots, 2^{J-L-1}-1\}$$

that are involved in the computation.

4.3.4. Algorithm

We summarize the reconstruction of a vector $\mathbf{x}$ from noisy Fourier data $\widehat{\mathbf{y}}$ in Algorithm 4.12. It includes all improvements that were proposed in Subsection 4.3.1 – 4.3.3.

Algorithm 4.12 (Sparse FFT for vectors with small support for noisy Fourier data)
Input: Noisy measurement vector $\widehat{\mathbf{y}} \in \mathbb{C}^N$, $N = 2^J$, $|\operatorname{supp} \mathbf{x}| \leq m < N$.

- Compute L such that $2^{L-1} < m \leq 2^L$, i.e., $L := \lceil \log_2 m \rceil$.

- If $L = J$ or $L = J - 1$, compute $\mathbf{x} = \mathbf{F}_N^{-1} \widehat{\mathbf{y}}$ by inverse FFT.

- If $L < J - 1$:

  1. Choose $\widehat{\widetilde{\mathbf{z}}}^{(0)} := \widehat{\mathbf{y}}^{(L+1)} = (\widehat{y}_{2^{J-(L+1)}k})_{k=0}^{2^{L+1}-1}$ and compute $\widetilde{\mathbf{z}}^{(0)} = \mathbf{y}^{(L+1)} := \mathbf{F}_{2^{L+1}}^{-1} \widehat{\mathbf{y}}^{(L+1)}$ using an FFT algorithm for the inverse discrete Fourier transform (IFFT).

  2. Determine the first support index $\mu^{(L+1)} \in \{0, \ldots, 2^{L+1} - 1\}$ of $\mathbf{x}^{(L+1)}$ using the following iteration:

     - Compute the local energies

     $$\widetilde{e}_k^{(0)} := \sum_{\ell=k}^{m+k-1} \left| \widetilde{z}^{(0)}_{\ell \bmod 2^{L+1}} \right|^2, \qquad k = 0, \ldots, 2^{L+1} - 1,$$

     and compute $\mu_0^{(L+1)} := \operatorname{argmax}_k \widetilde{e}_k^{(0)}$.

     - Compute $\widetilde{\mathbf{z}}^{(J-L-2)}$ by IFFT from $(\widehat{y}_{2^{J-L-2}(2k+1)})_{k=0}^{2^{L+1}-1}$, determine

     $$\widetilde{e}_k^{(1)} := \sum_{\ell=k}^{m+k-1} \left| \widetilde{z}^{(2^{J-L-2})}_{\ell \bmod 2^{L+1}} \right|^2, \qquad k = 0, \ldots, 2^{L+1} - 1,$$

     and take $\mu_1^{(L+1)} := \operatorname{argmax}_k \frac{1}{2}(\widetilde{e}_k^{(0)} + \widetilde{e}_k^{(1)})$.

     - Set $j := 0$.
     While $\mu_j^{(L+1)} \neq \mu_{j+1}^{(L+1)}$

       proceed by computing for a further $\kappa \in \{1, \ldots, 2^{J-L-1} - 1\}$ the vector $\widetilde{\mathbf{z}}^{(\kappa)}$ by IFFT from $(\widehat{y}_{2^{J-L-1}k+\kappa)})_{k=0}^{2^{L+1}-1}$, the energies

       $$\widetilde{e}_k^{(j+2)} := \sum_{\ell=k}^{m+k-1} \left| \widetilde{z}^{(\kappa)}_{\ell \bmod 2^{L+1}} \right|^2, \qquad k = 0, \ldots, 2^{L+1} - 1,$$

       and take $\mu_{j+2}^{(L+1)} := \operatorname{argmax}_k \frac{1}{j+3} \sum_{r=0}^{j+2} \widetilde{e}_k^{(r)}$.
       Set $\mu^{(L+1)} := \mu_{j+2}^{(L+1)}$ and $j := j + 1$.

     End (while).

- Set $\mathbf{x}^{(L+1)} = (x_k^{(L+1)})_{k=0}^{2^{L+1}-1}$ with

$$x^{(L+1)}_{(k+\mu^{(L+1)}) \bmod 2^{L+1}} := \begin{cases} \widetilde{z}^{(0)}_{(k+\mu^{(L+1)}) \bmod 2^{L+1}} & k = 0, \ldots, m-1, \\ 0 & k = m, \ldots, 2^{L+1}-1. \end{cases}$$

3. For $j = L+1, \ldots, J-1$

   Choose a Fourier component $\widehat{y}_{2^{J-(j+1)}(2k_0+1)} = \widehat{y}^{(j+1)}_{2k_0+1} \neq 0$ and compute the following value for comparison:

$$A := \sum_{\ell=0}^{m-1} x^{(L+1)}_{(\mu^{(L+1)}+\ell) \bmod 2^{L+1}} \, \omega_{2^{j+1}}^{(2k_0+1)(\mu^{(j)}+\ell)}.$$

   If $|A - \widehat{y}^{(j+1)}_{2k_0+1}| < |A + \widehat{y}^{(j+1)}_{2k_0+1}|$, then set $\mu^{(j+1)} := \mu^{(j)}$ and $\mathbf{x}^{(j+1)} := (x_k^{(j+1)})_{k=0}^{2^{j+1}-1}$ with entries

$$x^{(j+1)}_{\mu^{(j)}+\ell} = \begin{cases} x^{(j)}_{(\mu^{(j)}+\ell) \bmod 2^j} & \ell = 0, \ldots, m-1, \\ 0 & \text{else.} \end{cases}$$

   If $|A - \widehat{y}^{(j+1)}_{2k_0+1}| \geq |A + \widehat{y}^{(j+1)}_{2k_0+1}|$, then set $\mu^{(j+1)} := \mu^{(j)} + 2^j$ and $\mathbf{x}^{(j+1)} := (x_k^{(j+1)})_{k=0}^{2^{j+1}-1}$ with entries

$$x^{(j+1)}_{(\mu^{(j)}+2^j+\ell) \bmod 2^{j+1}} = \begin{cases} x^{(j)}_{(\mu^{(j)}+\ell) \bmod 2^j} & \ell = 0, \ldots, m-1, \\ 0 & \text{else.} \end{cases}$$

   End (for).

4. Assuming that $\widetilde{\mathbf{z}}^{(\kappa_r)} \in \mathbb{C}^{2^{L+1}}$, $r = 0, \ldots, B$, have already been evaluated in step 2, we compute

$$x_{(\mu^{(J)}+k) \bmod N} = \frac{1}{B+1} \sum_{r=0}^{B} \widetilde{z}^{(\kappa_r)}_{(\mu^{(L+1)}+k) \bmod 2^{L+1}} \, \omega_N^{-\kappa_r(\mu^{(J)}+k)},$$

   $k = 0, \ldots, m-1$.

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

An exemplary MATLAB implementation of this algorithm is included in Appendix A.1.

The complexity of the algorithm mainly depends on the support length m of $\mathbf{x}$. We summarize the number of necessary arithmetical operations for each step. In step 1 and 2, the periodization $\mathbf{x}^{(L+1)}$ has to be determined as well as further vectors $\widetilde{\mathbf{z}}^{(\kappa)}$ for a more stable identification of the support interval of $\mathbf{x}^{(L+1)}$. This includes two or more inverse FFTs of length $2^{L+1} < 4m$ and therefore $\mathcal{O}(m \log m)$ arithmetical operations. In order to determine the first support index $\mu^{(L+1)}$ of $\mathbf{x}^{(L+1)}$, we compute the local energies defined in Section 4.3.1. This requires $\mathcal{O}(m)$ operations. Hence, altogether steps 1 and 2 have an arithmetical complexity of $\mathcal{O}(m \log m)$.

In step 3 of the algorithm, the first support index $\mu^{(J)}$ is computed iteratively from $\mu^{(L+1)}$. This requires $J-(L+1) = \log_2 N - \lceil \log_2 m \rceil - 1 = \lfloor \log_2(N/m) \rfloor - 1$ reconstruction steps where at each level a scalar product of length m has to be computed and then to be compared to a suitable component of $\widehat{\mathbf{x}}$. Finding such a nonzero Fourier value requires less than m comparisons (as there are at most $m-1$ nonzero components to choose from, see Lemma 4.5). Hence, step 3 requires $\mathcal{O}(m \log(N/m))$ arithmetical operations.

Finally, the improvement of the (at most) m nonzero support components of $\mathbf{x}$ using vectors from step 2 needs a few additional multiplications and additions, depending on the number of vectors used in step 2.

To conclude, the overall complexity of the algorithm for reconstructing a vector $\mathbf{x}$ from noisy Fourier data is $\mathcal{O}(m \log m + m \log(N/m)) = \mathcal{O}(m \log N)$.

The number of Fourier values required for reconstruction of a vector $\mathbf{x}$ of length N with support length m highly depends on the number of vectors used in step 2 of the algorithm. Step 1 requires 2^{L+1} Fourier values and each additional vector in step 2 again 2^{L+1} values. In step 3, we need one Fourier value for comparison at each iteration level, hence $J-(L+1) = \lfloor \log_2(N/m) \rfloor - 1$ values for all iteration levels.

Remark 4.13 *For the computation of* $\mathbf{x}$ *by the algorithm it is not necessary to reconstruct all periodizations* $\mathbf{x}^{(j)}$*,* $j = L+2, \ldots, J-1$*, in step 3 in full length. After computing the periodization* $\mathbf{x}^{(L+1)}$ *by an inverse FFT algorithm, it suffices for all further periodizations* $\mathbf{x}^{(j)}$ *to determine their first support index* $\mu^{(j)}$*. This is recursively done from* $\mu^{(j-1)}$ *using the values A. The computation of any A only involves* $\mu^{(j)}$ *and the periodization* $\mathbf{x}^{(L+1)}$*.*

To finally reconstruct $\mathbf{x}$*, we only need the support, which was already contained in* $\mathbf{x}^{(L+1)}$*, and the first support index* $\mu^{(J)}$ *that was reconstructed iteratively.*

The intermediate periodizations $\mathbf{x}^{(j)}$*,* $j = L+2, \ldots, J-1$*, in step 3 of Algorithm 4.12 are only given for clarity but are not necessary for the reconstruction of* $\mathbf{x}$*.*

4.4. Numerical results

In this chapter, we evaluate the numerical stability of Algorithm 4.12 for the reconstruction of vectors with small support from noisy data. In particular, we apply the algorithm to noisy Fourier data $\widehat{\mathbf{y}} := (\widehat{y}_k)_{k=0}^{N-1} \in \mathbb{C}^N$ where

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

Here, the exact Fourier data $\widehat{\mathbf{x}} \in \mathbb{C}^N$ is perturbed by addition of a noise vector $\boldsymbol{\varepsilon} \in \mathbb{C}^N$.

In numerical experiments, we want to compare the implementation of Algorithm 4.12 (see Appendix A.1) to a regular inverse FFT algorithm, where we use `ifft` in MATLAB 2013a.

The algorithms are applied in different settings. First, we give an example for a vector of length 256 and illustrate the results of the reconstruction in detail. Afterwards, the algorithm is applied to larger numbers of randomly generated vectors of length $2^{20} = 1048576$. These vectors are either perturbed by uniformly distributed noise or by normally distributed noise, where we use the standard normal distribution. As a measure for the noise level, we introduce an SNR value.

Definition 4.14 *The signal-to-noise-ratio (SNR) for the noisy input data is given by*

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

Using the ℓ_2-norm, we quantify the reconstruction error by

$$\frac{\|\mathbf{x} - \mathbf{x}'\|_2}{N}$$

where $\mathbf{x}'$ denotes the reconstruction of $\mathbf{x}$ by the proposed algorithm. Similarly, the ℓ_2-error of a vector reconstruction from $\widehat{\mathbf{y}}$ by an inverse FFT algorithm is given by $\|\mathbf{x} - \mathbf{F}_N^{-1}\widehat{\mathbf{y}}\|_2/N$. As a second measure for the reconstruction error, we introduce SNR values

$$\mathrm{SNR}_{\mathrm{alg}} = 20 \cdot \log_{10} \frac{\|\mathbf{x}\|_2}{\|\mathbf{x} - \mathbf{x}'\|_2} \qquad \text{resp.} \qquad \mathrm{SNR}_{\mathrm{IFFT}} = 20 \cdot \log_{10} \frac{\|\mathbf{x}\|_2}{\|\mathbf{x} - \mathbf{F}_N^{-1}\widehat{\mathbf{y}}\|_2}.$$

Let us first illustrate the reconstruction by our deterministic algorithm for a small example. Let $\mathbf{x} \in \mathbb{R}^N$ with $N = 2^8 = 256$ and support length $m = 6$, i.e., $J = 8$ and $L = 3$. The vector $\mathbf{x}$ has the following nonzero components: $x_{105} = 8$, $x_{107} = -3$, $x_{108} = -5$

and $x_{110} = 2$. We perturb the Fourier data $\widehat{\mathbf{x}}$ by a noise vector $\boldsymbol{\varepsilon}$ and reconstruct $\mathbf{x}$ from $\widehat{\mathbf{y}} = \widehat{\mathbf{x}} + \boldsymbol{\varepsilon}$. The randomly chosen vector $\boldsymbol{\varepsilon}$ contains uniformly distributed noise with SNR = 20 and has in this example norms $\|\boldsymbol{\varepsilon}\|_\infty = 1.666$ and $\|\boldsymbol{\varepsilon}\|_1/N = 0.939$.

The reconstruction $\mathbf{x}'$ by the proposed algorithm has only six nonzero components:

$$x'_{105} = 7.944 - 0.090\mathrm{i}, \quad x'_{106} = 0.032 - 0.220\mathrm{i}, \quad x'_{107} = -2.936 - 0.061\mathrm{i},$$
$$x'_{108} = -5.007 + 0.089\mathrm{i}, \quad x'_{109} = -0.073 - 0.005\mathrm{i} \quad \text{and } x'_{110} = 2.129 - 0.141\mathrm{i}.$$

We observe that our algorithm detects the correct support interval of $\mathbf{x}$ and only reconstructs nonzero components within this interval. In contrast, the inverse Fourier transform applied to the noisy vector $\widehat{\mathbf{y}}$ returns a resulting vector with no zero entries. However, the components outside of the support are very small.

The results of the reconstructions by the proposed deterministic algorithm and by an inverse FFT applied to $\widehat{\mathbf{y}}$ yield errors $\|\mathbf{x}-\mathbf{x}'\|_2/256 = 0.00134$ resp. $\|\mathbf{x}-\mathbf{F}_{256}^{-1}\widehat{\mathbf{y}}\|_2/256 = 0.00395$. Moreover, the reconstructions have SNR values $\mathrm{SNR}_{\mathrm{alg}} = 29.364$ and $\mathrm{SNR}_{\mathrm{IFFT}} = 20$.

In this example, the algorithm requires no additional vectors for the identification of the first support index $\mu^{(L+1)}$ of the periodization $\mathbf{x}^{(L+1)}$ in step 2. This means that only two vectors $\widetilde{\mathbf{z}}^{(0)}$ and $\widetilde{\mathbf{z}}^{(J-L-2)} = \widetilde{\mathbf{z}}^{(3)}$ of length $2^{L+1} = 16$ are evaluated. Hence, $32 + 4 = 36$ of 256 Fourier values were used to recover $\mathbf{x}$ by our algorithm whereas the reconstruction by an inverse Fourier transform requires all 256 Fourier values.

The results for both vector reconstructions are illustrated in Figure 4.2. It is obvious to see that the algorithm only reconstructs the support of $\mathbf{x}$ and sets all remaining components to zero whereas the reconstruction by an inverse FFT algorithm does not recognize the relevant parts of $\mathbf{x}$ and therefore also recovers noisy components outside of the support.

Remark 4.15 *The ℓ_2-error of the reconstruction by an inverse Fourier transform in this example does not depend on the instance of the noise vector $\boldsymbol{\varepsilon}$ if the vector $\widehat{\mathbf{x}}$ and the SNR value are fixed. This can be seen by the following computation:*

$$\begin{aligned}\frac{1}{N}\left\|\mathbf{x} - \mathbf{F}_N^{-1}\widehat{\mathbf{y}}\right\|_2 &= \frac{1}{N}\left\|\mathbf{x} - \mathbf{F}_N^{-1}\widehat{\mathbf{x}} - \mathbf{F}_N^{-1}\boldsymbol{\varepsilon}\right\|_2 = \frac{1}{N}\left\|\mathbf{F}_N^{-1}\boldsymbol{\varepsilon}\right\|_2 \\ &= \frac{1}{N}\left\|\frac{1}{\sqrt{N}}\boldsymbol{\varepsilon}\right\|_2 = \frac{1}{N\sqrt{N}}\|\boldsymbol{\varepsilon}\|_2,\end{aligned}$$

where the second last equation holds as $\sqrt{N}\mathbf{F}_N^{-1}$ is a unitary matrix (see Remark 2.2).

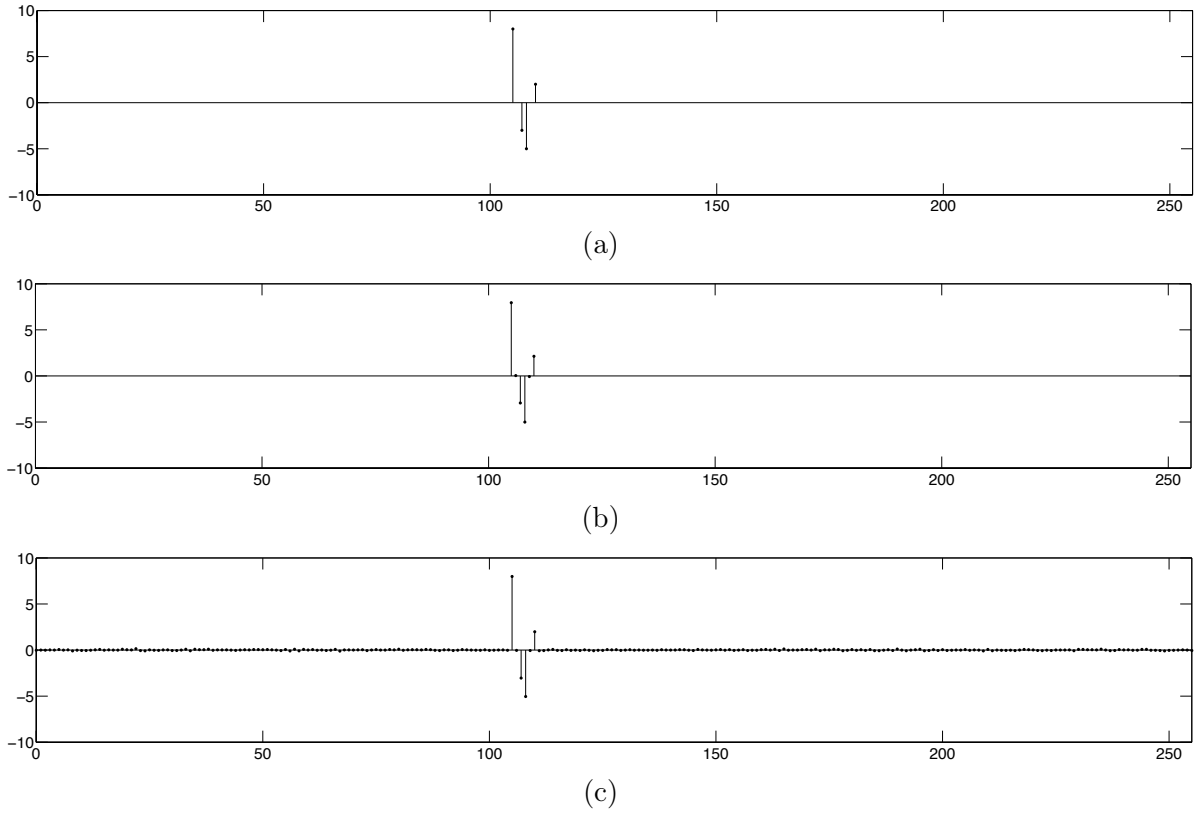


Figure 4.2.: (a) Original vector $\mathbf{x} \in \mathbb{R}^{256}$; (b) Reconstruction of $\mathbf{x}$ (real part) using the sparse FFT Algorithm 4.12; (c) Reconstruction of $\mathbf{x}$ (real part) using an inverse FFT algorithm.

The norm $\|\boldsymbol{\varepsilon}\|_2$ and hence the reconstruction error only depends on $\widehat{\mathbf{x}}$ (by definition of the SNR) resp. on $\mathbf{x}$.

However, the error of the reconstruction by our algorithm directly depends on the components of $\boldsymbol{\varepsilon}$ that are actually involved in the evaluation as not all of the entries of the noisy vector $\widehat{\mathbf{y}}$ are included in the reconstruction.

In further experiments we investigate the average reconstruction results for a large number of randomly generated vectors. The setting is as follows: We reconstruct vectors $\mathbf{x} \in \mathbb{C}^N$ with $N = 2^{20}$ for which the Fourier data $\widehat{\mathbf{x}}$ is perturbed either by uniformly or normally distributed noise $\boldsymbol{\varepsilon}$. The considered noise levels are SNR values between 0 and 50. Additionally, we vary the support length and set either a rather short support with $m = 20$ or a quite long support with $m = 2^{16}$. For each possible setting, 100 randomly chosen vectors $\mathbf{x}$ are recovered from $\widehat{\mathbf{y}} = \widehat{\mathbf{x}} + \boldsymbol{\varepsilon}$ by the deterministic algorithm and by an inverse FFT algorithm. The vectors $\mathbf{x}$ are chosen such that $|\mathrm{Re}(x_k)| \leq 10$, $|\mathrm{Im}(x_k)| \leq 10$

holds for all indices k within the support interval.

The obtained results are shown in several figures. In Figure 4.3, we compare the average error in 100 reconstructions by both methods, by our new sparse FFT algorithm as well as by an inverse FFT algorithm applied to $\widehat{\mathbf{y}}$. The vectors $\mathbf{x}$ have a support length of $m = 20$ and the data $\widehat{\mathbf{y}}$ is perturbed by uniform noise. The error is measured in two different ways: either by the ℓ_2-norm $\|\mathbf{x} - \mathbf{x}'\|_2/N$ or by the SNR values $\mathrm{SNR}_{\mathrm{alg}}$ and $\mathrm{SNR}_{\mathrm{IFFT}}$. Figure 4.4 illustrates the results for an equivalent setting where only the support length was changed to $m = 2^{16} = 65536$.

The reconstruction errors by both methods for data perturbed by normally distributed noise are compared in Figure 4.5 and 4.6. In Figure 4.5, the ℓ_2-norm as well as the SNR values are compared for the reconstructions of vectors $\mathbf{x}$ with support length $m = 20$ whereas in Figure 4.6, we do the same for vectors $\mathbf{x}$ with support length $m = 2^{16}$.

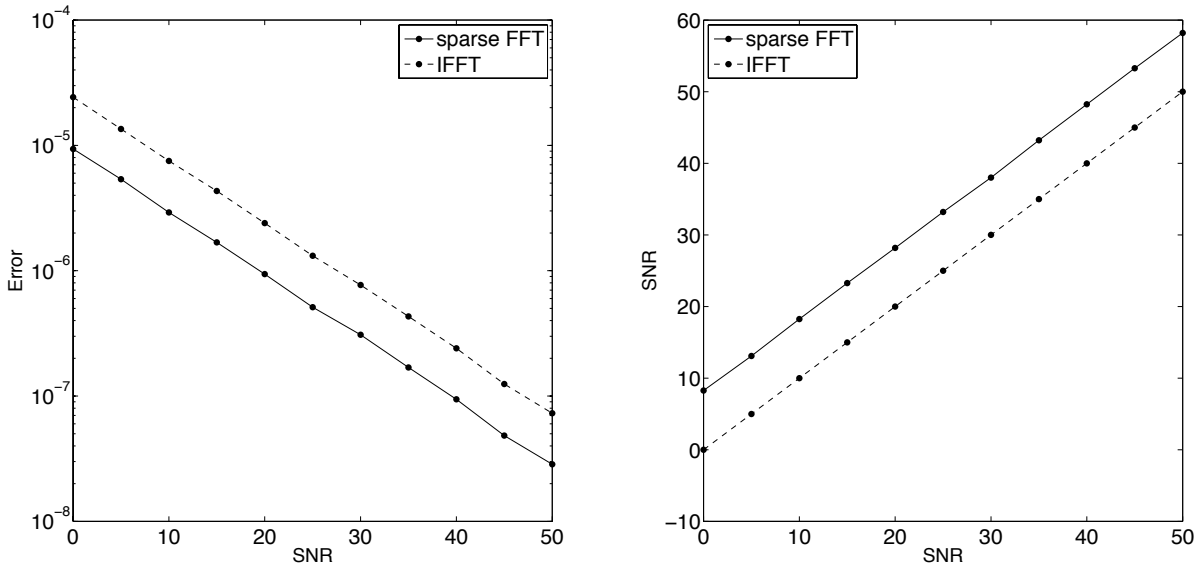
When looking at the figures, it is apparent that the error $\mathrm{SNR}_{\mathrm{IFFT}}$ does not depend on the noise but seems to be equal to the SNR value of the added noise vector $\boldsymbol{\varepsilon}$. This is indeed the case.

Remark 4.16 *Similar to the ℓ_2-error above, the error* $\mathrm{SNR}_{\mathrm{IFFT}}$ *for reconstruction of* $\mathbf{x}$ *by an inverse FFT algorithm does not depend on* $\mathbf{x}$ *or the instance of the noise vector* $\boldsymbol{\varepsilon}$ *itself, but only on its* SNR *value and both values even coincide:*

$$\begin{aligned}\mathrm{SNR}_{\mathrm{IFFT}} &= 20 \cdot \log_{10} \frac{\|\mathbf{x}\|_2}{\|\mathbf{x} - \mathbf{F}_N^{-1}\widehat{\mathbf{y}}\|_2} = 20 \cdot \log_{10} \frac{\|\mathbf{x}\|_2}{\|\mathbf{F}_N^{-1}\boldsymbol{\varepsilon}\|_2} = 20 \cdot \log_{10} \frac{\sqrt{N}\|\mathbf{F}_N^{-1}\widehat{\mathbf{x}}\|_2}{\|\boldsymbol{\varepsilon}\|_2} \\ &= 20 \cdot \log_{10} \frac{\sqrt{N}\frac{1}{\sqrt{N}}\|\widehat{\mathbf{x}}\|_2}{\|\boldsymbol{\varepsilon}\|_2} = 20 \cdot \log_{10} \frac{\|\widehat{\mathbf{x}}\|_2}{\|\boldsymbol{\varepsilon}\|_2} = \mathrm{SNR}.\end{aligned}$$

Again, the error $\mathrm{SNR}_{\mathrm{alg}}$ *of our proposed algorithm directly depends on the used components of* $\widehat{\mathbf{y}}$ *and therefore differs for various noise vectors.*

A crucial point in the reconstruction process by our algorithm is the correct identification of the support interval of the vector $\mathbf{x}$. For this, the correct determination of the first support index $\mu^{(L+1)}$ of the periodization $\mathbf{x}^{(L+1)}$ in step 2 of Algorithm 4.12 is of particular importance. If $\mu^{(L+1)}$ cannot be correctly identified, it is no longer possible to achieve the correct support for $\mathbf{x}$ in further steps of the algorithm. However, the determination of $\mu^{(L+1)}$ by our algorithm works in a very stable way. In order to see this, we evaluated further parameters in the above experiments which can be found in Table 4.1 – 4.4. The tables display the number of cases in which the first support index μ of $\mathbf{x}$ has been determined correctly in the reconstruction of 100 randomly chosen vectors

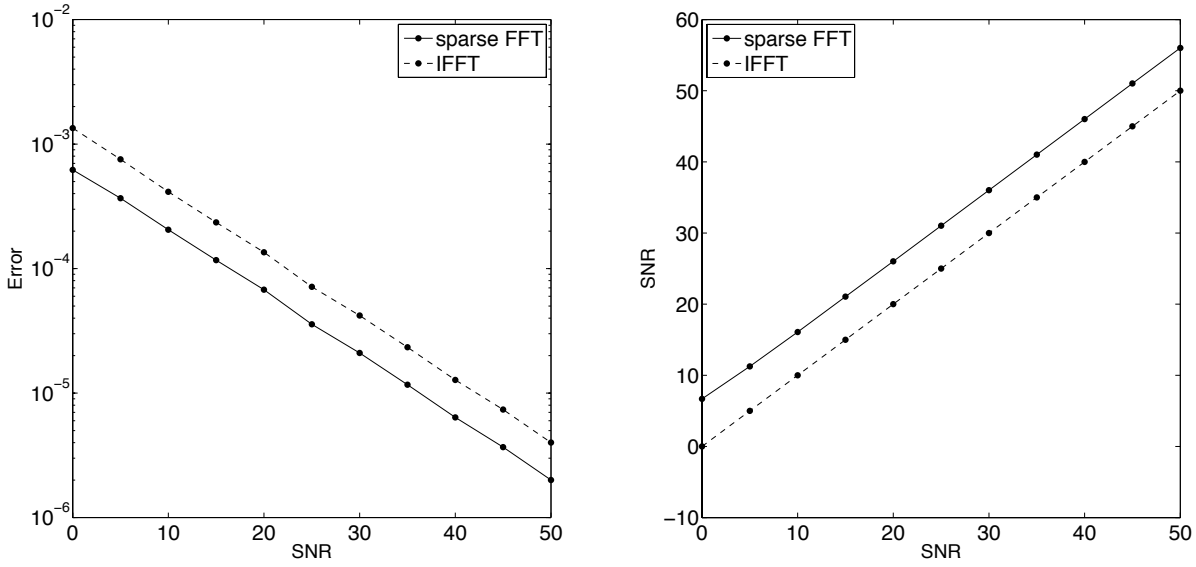


(a) (b)

Figure 4.3.: Uniformly distributed noise, $N = 2^{20}$, $m = 20$: (a) Average reconstruction error $\|\mathbf{x} - \mathbf{x}'\|_2/N$ and (b) average $\mathrm{SNR}_{\mathrm{alg}}$ resp. $\mathrm{SNR}_{\mathrm{IFFT}}$ for different noise levels, comparing the sparse FFT Algorithm 4.12 and regular inverse FFT.

SNR	correctly identified μ	largest error in μ	average number of vectors in step 2	$\|\varepsilon\|_\infty$	$\|\varepsilon\|_1/N$
0	84%	13	2.16	43.986	23.817
5	95%	2	2.03	24.556	13.296
10	99%	2	2.05	13.651	7.391
15	100%	0	2	7.853	4.252
20	100%	0	2	4.351	2.356
25	100%	0	2	2.389	1.294
30	100%	0	2	1.395	0.755
35	100%	0	2	0.783	0.424
40	100%	0	2	0.436	0.236
45	100%	0	2	0.226	0.123
50	100%	0	2	0.132	0.072

Table 4.1.: Uniformly distributed noise, $N = 2^{20}$, $m = 20$: Percentage of correctly identified μ, largest error in μ, average number of used vectors in step 2 of the Algorithm 4.12 and average norm of noise in 100 randomly chosen vectors for different noise levels.

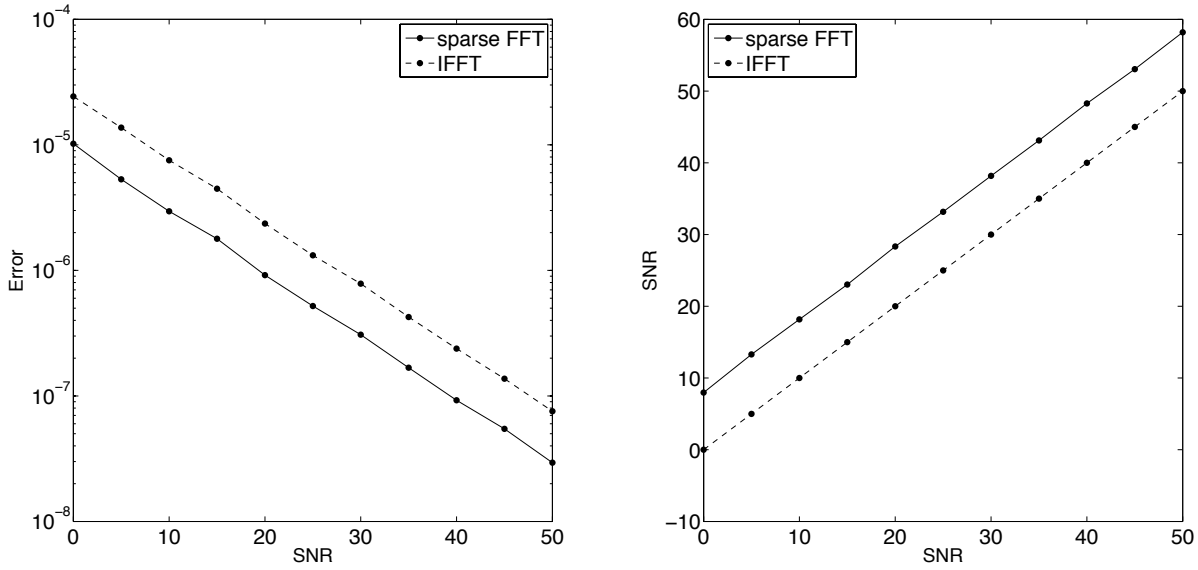


(a) (b)

Figure 4.4.: Uniformly distributed noise, $N = 2^{20}$, $m = 2^{16}$: (a) Average reconstruction error $\|\mathbf{x} - \mathbf{x}'\|_2/N$ and (b) average $\mathrm{SNR}_{\mathrm{alg}}$ resp. $\mathrm{SNR}_{\mathrm{IFFT}}$ for different noise levels, comparing the sparse FFT Algorithm 4.12 and regular inverse FFT.

SNR	correctly identified μ	largest error in μ	average number of vectors in step 2	$\|\varepsilon\|_\infty$	$\|\varepsilon\|_1/N$
0	82%	7	2.43	2441.806	1321.939
5	94%	7	2.14	1368.389	740.833
10	99%	1	2.04	751.915	407.071
15	100%	0	2.02	426.447	230.883
20	100%	0	2	245.649	133.006
25	100%	0	2	129.539	70.130
30	100%	0	2	76.200	41.256
35	100%	0	2	42.371	22.939
40	100%	0	2	23.163	12.540
45	100%	0	2	13.374	7.241
50	100%	0	2	7.275	3.939

Table 4.2.: Uniformly distributed noise, $N = 2^{20}$, $m = 2^{16}$: Percentage of correctly identified μ, largest error in μ, average number of used vectors in step 2 of the Algorithm 4.12 and average norm of noise in 100 randomly chosen vectors for different noise levels.

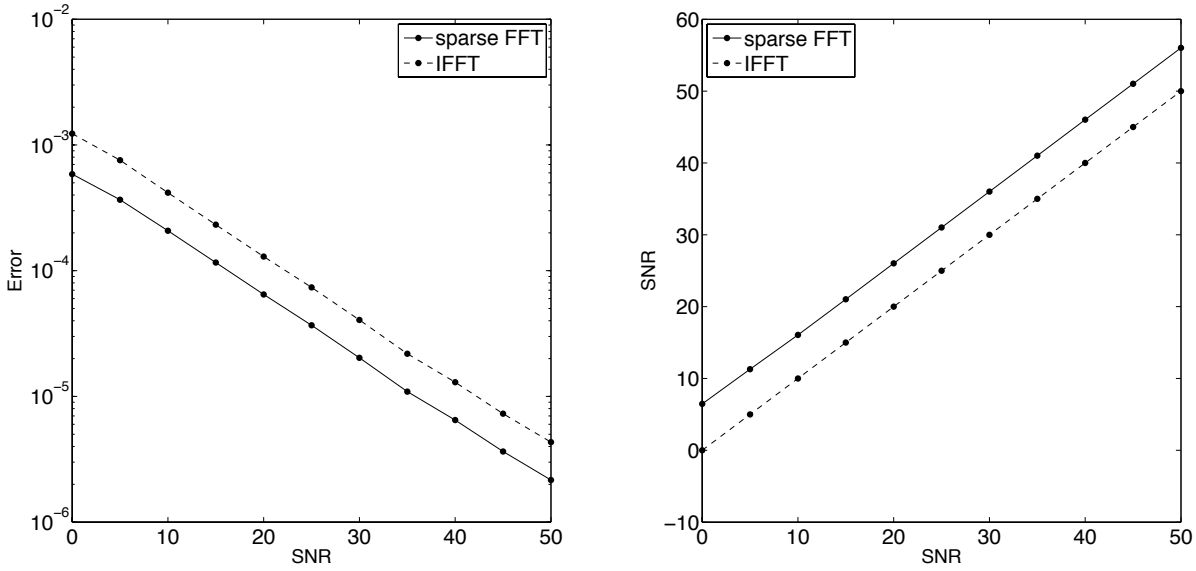


(a) (b)

Figure 4.5.: Normally distributed noise, $N = 2^{20}$, $m = 20$: (a) Average reconstruction error $\|\mathbf{x} - \mathbf{x}'\|_2/N$ and (b) average $\mathrm{SNR}_{\mathrm{alg}}$ resp. $\mathrm{SNR}_{\mathrm{IFFT}}$ for different noise levels, comparing the sparse FFT Algorithm 4.12 and regular inverse FFT.

SNR	correctly identified μ	largest error in μ	average number of vectors in step 2	$\|\varepsilon\|_\infty$	$\|\varepsilon\|_1/N$
0	84%	746498	2.25	96.024	22.604
5	97%	2	2.01	54.515	12.743
10	99%	1	2.02	30.107	7.007
15	100%	0	2	17.735	4.160
20	100%	0	2	9.436	2.196
25	100%	0	2	5.220	1.226
30	100%	0	2	3.122	0.729
35	100%	0	2	1.696	0.395
40	100%	0	2	0.947	0.221
45	100%	0	2	0.543	0.128
50	100%	0	2	0.302	0.070

Table 4.3.: Normally distributed noise, $N = 2^{20}$, $m = 20$: Percentage of correctly identified μ, largest error in μ, average number of used vectors in step 2 of the Algorithm 4.12 and average norm of noise in 100 randomly chosen vectors for different noise levels.



(a) (b)

Figure 4.6.: Normally distributed noise, $N = 2^{20}$, $m = 2^{16}$: (a) Average reconstruction error $\|\mathbf{x} - \mathbf{x}'\|_2/N$ and (b) average $\mathrm{SNR}_{\mathrm{alg}}$ resp. $\mathrm{SNR}_{\mathrm{IFFT}}$ for different noise levels, comparing the sparse FFT Algorithm 4.12 and regular inverse FFT.

SNR	correctly identified μ	largest error in μ	average number of vectors in step 2	$\|\boldsymbol{\varepsilon}\|_\infty$	$\|\boldsymbol{\varepsilon}\|_1/N$
0	84%	9	2.27	4917.601	1144.560
5	87%	5	2.17	3011.634	703.557
10	97%	1	2.02	1660.230	388.014
15	100%	0	2	924.087	216.056
20	100%	0	2	516.591	120.399
25	100%	0	2	292.855	68.423
30	100%	0	2	161.257	37.720
35	100%	0	2	87.879	20.313
40	100%	0	2	51.197	12.062
45	100%	0	2	28.865	6.777
50	100%	0	2	17.276	4.020

Table 4.4.: Normally distributed noise, $N = 2^{20}$, $m = 2^{16}$: Percentage of correctly identified μ, largest error in μ, average number of used vectors in step 2 of the Algorithm 4.12 and average norm of noise in 100 randomly chosen vectors for different noise levels.

at each noise level. Additionally, we indicate the largest difference in μ as well as the average number of vectors required by the algorithm in step 2. Furthermore, the tables give average norms for the noise vectors $\boldsymbol{\varepsilon}$ at each noise level.

Altogether, the findings of the numerical experiments show that the proposed deterministic algorithm for the reconstruction of vectors with small support is stable, even for high noise levels. Moreover, it works almost equally well for uniformly and for normally distributed noise. The results additionally show that the algorithm succeeds for vectors with short support as well as for vectors with long support compared to the full length of $\mathbf{x}$. However, one can observe that the average reconstruction errors are in general slightly larger for vectors $\mathbf{x}$ with support length $m = 2^{16}$. For those vectors with a comparatively long support, the norm of $\mathbf{x}$ is essentially larger than for vectors $\mathbf{x}$ with support length $m = 20$. Hence, also the signal energy and therefore $\|\mathbf{x}\|_2$ are larger. By the definition of the SNR values, this subsequently causes larger norms of the error vectors $\|\boldsymbol{\varepsilon}\|_2$.

For all possible settings, the measured ℓ_2-error is mostly about half of the error of the reconstruction by a direct inverse FFT algorithm, whereas the complexity of our algorithm is significantly lower. The results in Table 4.1 – 4.4 show that the first support index μ of $\mathbf{x}$ can be determined very reliably, only for strong normally distributed noise, outliers occur more frequently. Here, we have to keep in mind that according to our settings, the nonzero components of test vectors satisfy $|\mathrm{Re}(x_k)| \leq 10$, $|\mathrm{Im}(x_k)| \leq 10$ while we did not fix a lower bound for $|\mathrm{Re}(x_k)|$ and $|\mathrm{Im}(x_k)|$. Thus, it may happen that by chance the component corresponding to the first support index has very small modulus and can no longer be recognized if it is smaller than noise components.

Therefore, in particular for normally distributed noise, some very rare large outliers can occur as we can especially see in Table 4.3. This is because, in contrast to uniform noise, the noise vector $\boldsymbol{\varepsilon}$ might have some very large entries in this case. Our findings show that it then can happen that not only $\mu^{(L+1)}$ cannot be correctly determined but also the shifts are incorrect and hence the error of μ is even larger than that of $\mu^{(L+1)}$. In our experiments, this only occurs for normal noise, support length $m = 20$ and $\mathrm{SNR} = 0$. For all other cases, the error of μ only arises from a wrongly estimated $\mu^{(L+1)}$. Nevertheless, the algorithm generally also works stably for normally generated noise. The tables show that the support can be correctly identified in all cases for $\mathrm{SNR} \geq 15$.

The average number of vectors $\widetilde{\mathbf{z}}^{(\kappa)}$ required in step 2 of Algorithm 4.12 for the determination of the first support index $\mu^{(L+1)}$ of the periodization $\mathbf{x}^{(L+1)}$ (and hence μ) has also been evaluated. Mostly, the algorithm uses the (minimal) number of two vectors $\widetilde{\mathbf{z}}^{(0)}$

and $\widetilde{\mathbf{z}}^{(J-L-2)}$, only for high noise levels, it applies additional vectors $\widetilde{\mathbf{z}}^{(\kappa)}$. For SNR $= 0$, the maximal number of vectors $\widetilde{\mathbf{z}}^{(\kappa)}$ required for reconstruction in our experiments was six, for uniformly distributed noise and a support length of $m = 2^{16}$.

Table 4.1 – 4.4 also indicate average numbers for the infinity norm as well as for the ℓ_1-norm of the noise vectors $\boldsymbol{\varepsilon}$. We observe that the absolute noise ε_k can be considerably large for a support length of $m = 2^{16}$. This can be explained by the larger norm of these vectors, as seen above. In general, the maximum norm $\|\boldsymbol{\varepsilon}\|_\infty$ is larger for normally distributed noise which is because of the normal distribution where outliers can occur.

Remark 4.17 *A comparison of runtimes of our algorithm, the* `ifft` *command of* MATLAB *2013a and a naive implementation of the Sande-Tukey algorithm shows that for vectors of length $N = 2^{20}$, the proposed algorithm runs faster for support lengths $\leq 10^3$. For these vectors, the runtime of the sparse FFT algorithm is approximately $4 \cdot 10^{-2}$ seconds where we use as hardware base an Apple MacBook Pro 9,2 with 8GB RAM and an Intel i5-3210M running OSX 10.10.5. The runtimes for the* `ifft` *command and the Sande-Tukey algorithm are approximately $6 \cdot 10^{-2}$ seconds resp.* 10 *seconds, for all support lengths.*

5. A sparse FFT algorithm for real nonnegative vectors

As mentioned in previous chapters, the fast Fourier transform of a vector can be improved in complexity if we assume further constraints on the vectors. One possibility for the structure of vectors was proposed in Chapter 4 where an algorithm for the reconstruction of vectors with small support of known length was developed.

However, in many applications there occur vectors with small support although we do not have any information about their support length or whether they really have small support at all. Therefore, in this chapter we want to develop a deterministic FFT algorithm for vectors with real nonnegative components which does not require a priori information on the support length of the vector to be reconstructed. Our algorithm automatically detects a small support and benefits from it and falls back on a regular FFT algorithm if this is not the case. This means that the algorithm is suited for the reconstruction of any real nonnegative vector, whether it has small support or not. However, for a vector of length N with a support interval of length m, we can achieve a computational complexity of $\mathcal{O}(m \log m \log(N/m))$. The number of required Fourier values in this case is bounded by $\mathcal{O}(m \log(N/m))$.

For the reconstruction of a real nonnegative vector $\mathbf{x} \in \mathbb{R}_+^N$ from Fourier values we choose a similar approach as in the case of a known support length. The vector $\mathbf{x}$ is recovered iteratively by computing its periodizations step-by-step. At each iteration level, the support interval of the periodization is determined which decides about the further proceeding. In case that the support length is smaller than half of the vector length, an optimized reconstruction method can be applied, whereas we fall back on a conventional FFT algorithm otherwise.

The results presented in this chapter have been published in [41].

5.1. Reconstructing real nonnegative vectors from Fourier data

Let $\mathbf{x} = (x_k)_{k=0}^{N-1} \in \mathbb{R}_+^N$ be a vector with real nonnegative components where $N = 2^J$ for some $J > 0$. We keep the definitions of the discrete Fourier transform, the support interval and the periodized vectors as before.

In the following we develop an algorithm that recovers the vector $\mathbf{x}$ from its given Fourier values $\widehat{\mathbf{x}}$. The algorithm is again based on the so-called divide-and-conquer technique and additionally detects vectors with small support length, i.e., vectors that have a support that is significantly shorter than the vector itself. In this case, it has a considerably lower complexity and requires a smaller amount of Fourier samples for reconstruction.

From now on, let $\mathbf{x} \in \mathbb{R}_+^N$ be the vector to be reconstructed and $\widehat{\mathbf{x}}$ be the vector of given Fourier samples. The support length of $\mathbf{x}$ is denoted by m and $L := \lceil \log_2 m \rceil$. The reconstruction procedure is iterative, beginning with one Fourier sample $\widehat{x}_0 = \mathbf{x}^{(0)} \in \mathbb{R}_+$. Each periodization $\mathbf{x}^{(j+1)}$ will be computed from $\mathbf{x}^{(j)}$ for $j = 0, \ldots, J-1$, depending on the respective support length $m_j = |\operatorname{supp} \mathbf{x}^{(j)}|$ of $\mathbf{x}^{(j)}$. Therefore, we compute m_j at each reconstruction step and then distinguish two cases: if $m_j > 2^{j-1}$, i.e., the support of $\mathbf{x}^{(j)}$ is longer than half of the vector length of $\mathbf{x}^{(j)}$, the next periodization $\mathbf{x}^{(j+1)}$ is computed by an inverse FFT of length 2^j and hence with the complexity of a regular FFT algorithm, $\mathcal{O}(2^j \log 2^j)$.

If on the other hand $m_j \leq 2^{j-1}$, we apply a new procedure with a reduced complexity to recover $\mathbf{x}^{(j+1)}$ from $\mathbf{x}^{(j)}$. In this case, the support of $\mathbf{x}^{(j)}$ is shorter than half of the vector length 2^j and hence uniquely defined in any case. The basic idea of the reconstruction procedure is to restrict the calculations to this support interval and to only execute an inverse FFT of length 2^{L_j} where $L_j = \lceil \log_2 m_j \rceil$.

In both cases, the underlying principle is to recover $\mathbf{x}^{(j+1)}$ by an inverse Fourier transform of a certain length taking advantage of the knowledge of the periodization $\mathbf{x}^{(j)}$. Before analyzing the computations in detail in both cases, we have a closer look at the Fourier transform $\widehat{\mathbf{x}}^{(j+1)}$ of $\mathbf{x}^{(j+1)}$ and its connection to the vector $\mathbf{x}^{(j)}$. We introduce the notation of *partial vectors*

$$\mathbf{x}_0^{(j+1)} = \left(x_k^{(j+1)}\right)_{k=0}^{2^j-1} \qquad \text{and} \qquad \mathbf{x}_1^{(j+1)} = \left(x_k^{(j+1)}\right)_{k=2^j}^{2^{j+1}-1}$$

which denote the first resp. the second half of the vector $\mathbf{x}^{(j+1)}$. Hence, by Definition 3.3,

the relation

$$\mathbf{x}^{(j)} = \mathbf{x}_0^{(j+1)} + \mathbf{x}_1^{(j+1)} \tag{5.1}$$

holds.

Remark 5.1 *At this point, we can explain why the application of the algorithm is restricted to vectors with real nonnegative components.*

Equation (5.1) describes the connection between the components of $\mathbf{x}^{(j)}$ *and* $\mathbf{x}^{(j+1)}$*. Supposing that the entries of* $\mathbf{x}^{(j)}$ *are known, the assumption that* $\mathbf{x}$ *only has real nonnegative entries now facilitates the computation of the components of* $\mathbf{x}^{(j+1)}$*: Because of the positivity, components can not "vanish" while adding them up in periodizations such that the vector* $\mathbf{x}^{(j+1)}$ *can only have nonzero entries within the same support interval as* $\mathbf{x}^{(j)}$ *or within this interval shifted by* 2^j*.*

Let us now deduce the reconstruction of $\mathbf{x}^{(j+1)}$ from $\mathbf{x}^{(j)}$ in detail. By the definition of the discrete Fourier transform, Lemma 3.7 and (5.1) we obtain

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

From

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

by Lemma 3.7 and the above equation, we conclude that the even-indexed Fourier components of $\widehat{\mathbf{x}}^{(j+1)}$ only contain information on $\mathbf{x}^{(j)}$ whereas the odd-indexed components contribute new information on the vector $\mathbf{x}^{(j+1)}$. Hence, we can restrict the system of equations to the ones with odd indices without losing information and obtain

$$\begin{aligned}
\left(\widehat{x}_{2k+1}^{(j+1)}\right)_{k=0}^{2^j-1} &= (\widehat{x}_{2^{J-j-1}(2k+1)})_{k=0}^{2^j-1} \\
&= \left(\omega_{2^{j+1}}^{(2k+1)\ell}\right)_{k,\ell=0}^{2^j-1} \mathbf{x}_0^{(j+1)} - \left(\omega_{2^{j+1}}^{(2k+1)\ell}\right)_{k,\ell=0}^{2^j-1} (\mathbf{x}^{(j)} - \mathbf{x}_0^{(j+1)}) \\
&= \left(\omega_{2^{j+1}}^{(2k+1)\ell}\right)_{k,\ell=0}^{2^j-1} (2\mathbf{x}_0^{(j+1)} - \mathbf{x}^{(j)}) \\
&= \left(\omega_{2^j}^{k\ell}\right)_{k,\ell=0}^{2^j-1} \operatorname{diag}\left(\omega_{2^{j+1}}^{\ell}\right)_{\ell=0}^{2^j-1} (2\mathbf{x}_0^{(j+1)} - \mathbf{x}^{(j)})
\end{aligned}$$

$$= \mathbf{F}_{2^j} \operatorname{diag}\left(\omega_{2^{j+1}}^{\ell}\right)_{\ell=0}^{2^j-1} \left(2\mathbf{x}_0^{(j+1)} - \mathbf{x}^{(j)}\right). \tag{5.2}$$

This relation allows us to compute $\mathbf{x}^{(j+1)} = \begin{pmatrix} \mathbf{x}_0^{(j+1)} \\ \mathbf{x}_1^{(j+1)} \end{pmatrix}$ directly from $\mathbf{x}^{(j)}$. In the first case where the support length m_j of $\mathbf{x}^{(j)}$ exceeds half of the vector length, the periodized vector $\mathbf{x}^{(j+1)}$ is obtained in this way. For the second case we need further considerations in order to improve the complexity of the reconstruction.

(1) First case: $m_j > 2^{j-1}$

In the case of a rather long support of $\mathbf{x}^{(j)}$ the partial vectors of $\mathbf{x}^{(j+1)}$ are obtained directly from equation (5.2) by

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

Then the second partial vector $\mathbf{x}_1^{(j+1)}$ is given by $\mathbf{x}^{(j)} = \mathbf{x}_0^{(j+1)} + \mathbf{x}_1^{(j+1)}$ or, in an explicit formulation, by

$$\mathbf{x}_1^{(j+1)} = \frac{1}{2}\left(-\operatorname{diag}\left(\omega_{2^{j+1}}^{-\ell}\right)_{\ell=0}^{2^j-1} \mathbf{F}_{2^j}^{-1} \left(\widehat{x}_{2^{J-j-1}(2k+1)}\right)_{k=0}^{2^j-1} + \mathbf{x}^{(j)}\right).$$

Finally, compose $\mathbf{x}^{(j+1)}$ as $\mathbf{x}^{(j+1)} = \begin{pmatrix} \mathbf{x}_0^{(j+1)} \\ \mathbf{x}_1^{(j+1)} \end{pmatrix}$. The reconstruction of $\mathbf{x}_0^{(L+1)}$ is mainly via a Fourier transform of length 2^j. Additionally, the multiplication with a diagonal matrix, i.e., 2^j complex multiplications, 2^j additions and one dyadic shift by 2 are required. The vector $\mathbf{x}_1^{(L+1)}$ is then obtained by $\mathcal{O}(2^j)$ arithmetical operations which yields an overall complexity of $\mathcal{O}(2^j \log 2^j)$ for this reconstruction step.

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

If the support length is shorter than or equal to half of the length of $\mathbf{x}^{(j)}$, we first compute $L_j = \lceil \log_2 m_j \rceil$. The first support index of the vector $\mathbf{x}^{(j)}$ is denoted by $\mu^{(j)}$. Then the support of $\mathbf{x}^{(j)}$ lies within an interval of length 2^{L_j} which begins at the first support index $\mu^{(j)}$.

In contrast to the first case we now consider not just partial vectors but only the parts of $\mathbf{x}^{(j)}$ and $\mathbf{x}^{(j+1)}$ which contain the nonzero components. Therefore, we introduce the notation of $\widetilde{\mathbf{x}}^{(j)}$, $\widetilde{\mathbf{x}}_0^{(j+1)}$ and $\widetilde{\mathbf{x}}_1^{(j+1)}$ that denote the vectors of length 2^{L_j} containing the

support of $\mathbf{x}^{(j)}$, $\mathbf{x}_0^{(j+1)}$ and $\mathbf{x}_1^{(j+1)}$ respectively:

$$\begin{aligned}\widetilde{\mathbf{x}}^{(j)} &= \left(x^{(j)}_{(\mu^{(j)}+r) \bmod 2^j}\right)_{r=0}^{2^{L_j}-1},\\ \widetilde{\mathbf{x}}_0^{(j+1)} &= \left(x^{(j+1)}_{(\mu^{(j)}+r) \bmod 2^j}\right)_{r=0}^{2^{L_j}-1} \quad \text{and}\\ \widetilde{\mathbf{x}}_1^{(j+1)} &= \left(x^{(j+1)}_{2^j+(\mu^{(j)}+r) \bmod 2^j}\right)_{r=0}^{2^{L_j}-1}.\end{aligned}$$

Using these vectors, the system of equations (5.2) can be restricted to the relevant part

$$\begin{aligned}\left(\widehat{x}_{2^{J-j-1}(2k+1)}\right)_{k=0}^{2^j-1} &= \left(\omega_{2^{j+1}}^{(2k+1)\ell}\right)_{k,\ell=0}^{2^j-1} \left(2\mathbf{x}_0^{(j+1)} - \mathbf{x}^{(j)}\right)\\ &= \left(\omega_{2^{j+1}}^{(2k+1)((\mu^{(j)}+r) \bmod 2^j)}\right)_{k,r=0}^{2^j-1,2^{L_j}-1} \left(2\widetilde{\mathbf{x}}_0^{(j+1)} - \widetilde{\mathbf{x}}^{(j)}\right).\end{aligned} \tag{5.3}$$

This system (5.3) still consists of 2^j equations. But by the definition of the periodized vectors, the relation

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

holds which already gives 2^{L_j} conditions on the components of $\widetilde{\mathbf{x}}_0^{(j+1)}$ (or $\widetilde{\mathbf{x}}_1^{(j+1)}$). Thus, another 2^{L_j} linearly independent equations would be sufficient to recover $\widetilde{\mathbf{x}}_0^{(j+1)}$ and hence $\widetilde{\mathbf{x}}^{(j+1)}$. Therefore, we can reduce the number of equations in (5.3) to 2^{L_j}. We do this in a way that will be convenient for a further matrix decomposition and choose 2^{L_j} indices in $\{0,\dots,2^j-1\}$ by $k_p = 2^{j-L_j}p$ for $p = 0,\dots,2^{L_j}-1$. This yields

$$\begin{aligned}\left(\widehat{x}_{2^{J-j-1}(2^{j+1-L_j}p+1)}\right)_{p=0}^{2^{L_j}-1} &= \left(\widehat{x}_{2^{J-L_j}p+2^{J-j-1}}\right)_{p=0}^{2^{L_j}-1}\\ &= \left(\omega_{2^{j+1}}^{(2^{j+1-L_j}p+1)((\mu^{(j)}+r) \bmod 2^j)}\right)_{p,r=0}^{2^{L_j}-1} \left(2\widetilde{\mathbf{x}}_0^{(j+1)} - \widetilde{\mathbf{x}}^{(j)}\right).\end{aligned} \tag{5.4}$$

The structure of the matrix $\left(\omega_{2^{j+1}}^{(2^{j+1-L_j}p+1)((\mu^{(j)}+r) \bmod 2^j)}\right)_{p,r=0}^{2^{L_j}-1}$ depends on the first support index $\mu^{(j)}$ of $\mathbf{x}^{(j)}$. Hence, we obtain a factorization relative to the choice of $\mu^{(j)}$ for this matrix.

Lemma 5.2 *Let $j \in \{0,\dots,J-1\}$ and $\mathbf{x}^{(j)}$ be a corresponding periodized vector of length 2^j. The support length of $\mathbf{x}^{(j)}$ is denoted by m_j and $L_j := \lceil \log_2 m_j \rceil$. Assume that $m_j \le 2^{j-1}$. Then the matrix $(\omega_{2^{j+1}}^{(2k+1)((\mu^{(j)}+r) \bmod 2^j)})_{k,r=0}^{2^j-1,2^{L_j}-1}$ can be decomposed as*

follows, depending on the first support index $\mu^{(j)}$ *of* $\mathbf{x}^{(j)}$*:*

$$\begin{aligned}
&\left(\omega_{2^{j+1}}^{(2^{j+1-L_j}p+1)((\mu^{(j)}+r)\bmod 2^j)}\right)_{p,r=0}^{2^{L_j}-1} \\
&= \begin{cases}
\omega_{2^{j+1}}^{\mu^{(j)}}\ \operatorname{diag}\left(\omega_{2^{L_j}}^{\mu^{(j)}p}\right)_{p=0}^{2^{L_j}-1}\ \mathbf{F}_{2^{L_j}}\ \operatorname{diag}\left(\omega_{2^{j+1}}^{r}\right)_{r=0}^{2^{L_j}-1} & \text{if } \mu^{(j)} \le 2^j - 2^{L_j}, \\
\omega_{2^{j+1}}^{\mu^{(j)}}\ \operatorname{diag}\left(\omega_{2^{L_j}}^{\mu^{(j)}p}\right)_{p=0}^{2^{L_j}-1}\ \mathbf{F}_{2^{L_j}}\ \operatorname{diag}\left(\omega_{2^{j+1}}^{r}\right)_{r=0}^{2^{L_j}-1}\ \mathbf{D}_{2^{L_j}}^{2^j-\mu^{(j)}} & \text{if } \mu^{(j)} > 2^j - 2^{L_j},
\end{cases}
\end{aligned}$$

where

$$\mathbf{D}_{2^{L_j}}^{2^j-\mu^{(j)}} := \begin{pmatrix} \mathbf{I}_{2^j-\mu^{(j)}} & 0 \\ 0 & -\mathbf{I}_{2^{L_j}-2^j+\mu^{(j)}} \end{pmatrix}$$

and $\mathbf{I}_d$ *denotes the identity matrix of size* d*.*

Proof. First, we factorize $\left(\omega_{2^{j+1}}^{(2k+1)((\mu^{(j)}+r)\bmod 2^j)}\right)_{k,r=0}^{2^j-1,2^{L_j}-1}$ as a product of Fourier and diagonal matrices,

$$\begin{aligned}
&\left(\omega_{2^{j+1}}{}^{(2^{j+1-L_j}p+1)((\mu^{(j)}+r)\bmod 2^j)}\right)_{p,r=0}^{2^{L_j}-1} \\
&\quad= \left(\omega_{2^{j+1}}^{2^{j+1-L_j}p((\mu^{(j)}+r)\bmod 2^j)}\ \omega_{2^{j+1}}^{(\mu^{(j)}+r)\bmod 2^j}\right)_{p,r=0}^{2^{L_j}-1} \\
&\quad= \left(\omega_{2^{L_j}}^{p((\mu^{(j)}+r)\bmod 2^j)}\right)_{p,r=0}^{2^{L_j}-1}\ \operatorname{diag}\left(\omega_{2^{j+1}}^{(\mu^{(j)}+r)\bmod 2^j}\right)_{r=0}^{2^{L_j}-1} \\
&\quad= \left(\omega_{2^{L_j}}^{p(\mu^{(j)}+r)}\right)_{p,r=0}^{2^{L_j}-1}\ \operatorname{diag}\left(\omega_{2^{j+1}}^{(\mu^{(j)}+r)\bmod 2^j}\right)_{r=0}^{2^{L_j}-1} \\
&\quad= \operatorname{diag}\left(\omega_{2^{L_j}}^{\mu^{(j)}p}\right)_{p=0}^{2^{L_j}-1}\ \left(\omega_{2^{L_j}}^{pr}\right)_{p,r=0}^{2^{L_j}-1}\ \operatorname{diag}\left(\omega_{2^{j+1}}^{(\mu^{(j)}+r)\bmod 2^j}\right)_{r=0}^{2^{L_j}-1} \\
&\quad= \operatorname{diag}\left(\omega_{2^{L_j}}^{\mu^{(j)}p}\right)_{p=0}^{2^{L_j}-1}\ \mathbf{F}_{2^{L_j}}\ \operatorname{diag}\left(\omega_{2^{j+1}}^{(\mu^{(j)}+r)\bmod 2^j}\right)_{r=0}^{2^{L_j}-1}.
\end{aligned} \tag{5.5}$$

The equation $\left(\omega_{2^{L_j}}^{p((\mu^{(j)}+r)\bmod 2^j)}\right)_{p,r=0}^{2^{L_j}-1} = \left(\omega_{2^{L_j}}^{p(\mu^{(j)}+r)}\right)_{p,r=0}^{2^{L_j}-1}$ holds because $m \le 2^{j-1}$ and hence $j > L_j$ and therefore 2^j is a multiple of 2^{L_j}.

For the diagonal matrix $\operatorname{diag}\left(\omega_{2^{j+1}}^{(\mu^{(j)}+r)\bmod 2^j}\right)_{r=0}^{2^{L_j}-1}$, we have to distinguish two cases. First, consider $\mu^{(j)} \le 2^j - 2^{L_j}$. Then

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

holds since $\mu^{(j)} + r \le 2^j - 2^{L_j} + r \le 2^j - 1$ for any $r \in \{0, \ldots, 2^{L_j} - 1\}$.

On the other hand, for $\mu^{(j)} > 2^j - 2^{L_j}$, the matrix can be written as a product of two diagonal matrices,

$$\begin{aligned}
&\operatorname{diag}\Big(\omega_{2^{j+1}}{}^{(\mu^{(j)}+r) \bmod 2^j}\Big)_{r=0}^{2^{L_j}-1} \\
&\qquad = \begin{pmatrix} \operatorname{diag}\left(\omega_{2^{j+1}}^{\mu^{(j)}+r}\right)_{r=0}^{2^j-\mu^{(j)}-1} & 0 \\ 0 & \operatorname{diag}\left(\omega_{2^{j+1}}^{\mu^{(j)}+r-2^j}\right)_{r=2^j-\mu^{(j)}}^{2^{L_j}-1} \end{pmatrix} \\
&\qquad = \begin{pmatrix} \operatorname{diag}\left(\omega_{2^{j+1}}^{\mu^{(j)}+r}\right)_{r=0}^{2^j-\mu^{(j)}-1} & 0 \\ 0 & \operatorname{diag}\left(-\omega_{2^{j+1}}^{\mu^{(j)}+r}\right)_{r=2^j-\mu^{(j)}}^{2^{L_j}-1} \end{pmatrix} \\
&\qquad = \operatorname{diag}\left(\omega_{2^{j+1}}^{\mu^{(j)}+r}\right)_{r=0}^{2^{L_j}-1} \ \operatorname{diag}(\underbrace{1,\dots,1}_{2^j-\mu^{(j)}},\underbrace{-1,\dots,-1}_{2^{L_j}-2^j+\mu^{(j)}}) \\
&\qquad = \omega_{2^{j+1}}^{\mu^{(j)}} \ \operatorname{diag}\left(\omega_{2^{j+1}}^{r}\right)_{r=0}^{2^{L_j}-1} \ \mathbf{D}_{2^{L_j}}^{2^j-\mu^{(j)}}
\end{aligned}$$

where

$$\mathbf{D}_{2^{L_j}}^{2^j-\mu^{(j)}} := \operatorname{diag}(\underbrace{1,\dots,1}_{2^j-\mu^{(j)}},\underbrace{-1,\dots,-1}_{2^{L_j}-2^j+\mu^{(j)}}) = \begin{pmatrix} \mathbf{I}_{2^j-\mu^{(j)}} & 0 \\ 0 & -\mathbf{I}_{2^{L_j}-2^j+\mu^{(j)}} \end{pmatrix}.$$

Here, $\mathbf{I}_d$ denotes the identity matrix of size d. Altogether, we have

$$\operatorname{diag}\Big(\omega_{2^{j+1}}{}^{(\mu^{(j)}+r) \bmod 2^j}\Big)_{r=0}^{2^{L_j}-1} = \begin{cases} \omega_{2^{j+1}}^{\mu^{(j)}} \ \operatorname{diag}\left(\omega_{2^{j+1}}^{r}\right)_{r=0}^{2^{L_j}-1} & \text{if } \mu^{(j)} \le 2^j - 2^{L_j}, \\ \omega_{2^{j+1}}^{\mu^{(j)}} \ \operatorname{diag}\left(\omega_{2^{j+1}}^{r}\right)_{r=0}^{2^{L_j}-1} \ \mathbf{D}_{2^{L_j}}^{2^j-\mu^{(j)}} & \text{if } \mu^{(j)} > 2^j - 2^{L_j}. \end{cases}$$

Plugging this into (5.5), we obtain

$$\begin{aligned}
&\left(\omega_{2^{j+1}}^{(2^{j+1-L_j}p+1)((\mu^{(j)}+r) \bmod 2^j)}\right)_{p,r=0}^{2^{L_j}-1} \\
&= \operatorname{diag}\left(\omega_{2^{L_j}}^{\mu^{(j)}p}\right)_{p=0}^{2^{L_j}-1} \ \mathbf{F}_{2^{L_j}} \ \operatorname{diag}\left(\omega_{2^{j+1}}^{(\mu^{(j)}+r) \bmod 2^j}\right)_{r=0}^{2^{L_j}-1} \\
&= \begin{cases} \omega_{2^{j+1}}^{\mu^{(j)}} \ \operatorname{diag}\left(\omega_{2^{L_j}}^{\mu^{(j)}p}\right)_{p=0}^{2^{L_j}-1} \ \mathbf{F}_{2^{L_j}} \ \operatorname{diag}(\omega_{2^{j+1}}^{r})_{r=0}^{2^{L_j}-1} & \text{if } \mu^{(j)} \le 2^j - 2^{L_j}, \\ \omega_{2^{j+1}}^{\mu^{(j)}} \ \operatorname{diag}\left(\omega_{2^{L_j}}^{\mu^{(j)}p}\right)_{p=0}^{2^{L_j}-1} \ \mathbf{F}_{2^{L_j}} \ \operatorname{diag}(\omega_{2^{j+1}}^{r})_{r=0}^{2^{L_j}-1} \ \mathbf{D}_{2^{L_j}}^{2^j-\mu^{(j)}} & \text{if } \mu^{(j)} > 2^j - 2^{L_j}. \end{cases}
\end{aligned}$$

This proves the asserted matrix decomposition. □

Using Lemma 5.2, we can decompose the matrix in equation (5.4) for an efficient reconstruction of $\mathbf{x}_0^{(j+1)}$ and hence of $\mathbf{x}^{(j+1)}$ from $\mathbf{x}^{(j)}$.

If $\mu^{(j)} \leq 2^j - 2^{L_j}$ holds for the first support index $\mu^{(j)}$ of $\mathbf{x}^{(j)}$, plugging in the decomposition from Lemma 5.2 yields

$$\begin{aligned}\left(\widehat{x}_{2^{J-L_j}p+2^{J-j-1}}\right)_{p=0}^{2^{L_j}-1} &= \left(\omega_{2^{j+1}}^{(2^{j+1-L_j}p+1)((\mu^{(j)}+r)\bmod 2^j)}\right)_{p,r=0}^{2^{L_j}-1} \left(2\widetilde{\mathbf{x}}_0^{(j+1)} - \widetilde{\mathbf{x}}^{(j)}\right) \\ &= \omega_{2^{j+1}}^{\mu^{(j)}} \operatorname{diag}\left(\omega_{2^{L_j}}^{\mu^{(j)}p}\right)_{p=0}^{2^{L_j}-1} \mathbf{F}_{2^{L_j}} \operatorname{diag}\left(\omega_{2^{j+1}}^{r}\right)_{r=0}^{2^{L_j}-1} \left(2\widetilde{\mathbf{x}}_0^{(j+1)} - \widetilde{\mathbf{x}}^{(j)}\right).\end{aligned}$$

We are thus able to compute

$$\begin{aligned}\widetilde{\mathbf{x}}_0^{(j+1)} = \frac{1}{2}\Bigg(&\omega_{2^{j+1}}^{-\mu^{(j)}} \operatorname{diag}\left(\omega_{2^{j+1}}^{-r}\right)_{r=0}^{2^{L_j}-1} \mathbf{F}_{2^{L_j}}^{-1} \\ &\cdot \operatorname{diag}\left(\omega_{2^{L_j}}^{-\mu^{(j)}p}\right)_{p=0}^{2^{L_j}-1} \left(\widehat{x}_{2^{J-L_j}p+2^{J-j-1}}\right)_{p=0}^{2^{L_j}-1} + \widetilde{\mathbf{x}}^{(j)}\Bigg)\end{aligned}$$

Similar to the first case, $\widetilde{\mathbf{x}}_1^{(j+1)}$ can be computed using $\widetilde{\mathbf{x}}^{(j+1)} = \widetilde{\mathbf{x}}_0^{(j+1)} + \widetilde{\mathbf{x}}_1^{(j+1)}$ or directly as

$$\begin{aligned}\widetilde{\mathbf{x}}_1^{(j+1)} = \frac{1}{2}\Bigg(&-\omega_{2^{j+1}}^{-\mu^{(j)}} \operatorname{diag}\left(\omega_{2^{j+1}}^{-r}\right)_{r=0}^{2^{L_j}-1} \mathbf{F}_{2^{L_j}}^{-1} \\ &\cdot \operatorname{diag}\left(\omega_{2^{L_j}}^{-\mu^{(j)}p}\right)_{p=0}^{2^{L_j}-1} \left(\widehat{x}_{2^{J-L_j}p+2^{J-j-1}}\right)_{p=0}^{2^{L_j}-1} + \widetilde{\mathbf{x}}^{(j)}\Bigg).\end{aligned}$$

For the case that $\mu^{(j)} > 2^j - 2^{L_j}$, the system (5.4) factorizes in a similar way,

$$\begin{aligned}\left(\widehat{x}_{2^{J-L_j}p+2^{J-j-1}}\right)_{p=0}^{2^{L_j}-1} &= \left(\omega_{2^{j+1}}^{(2^{j+1-L_j}p+1)((\mu^{(j)}+r)\bmod 2^j)}\right)_{p,r=0}^{2^{L_j}-1} \left(2\widetilde{\mathbf{x}}_0^{(j+1)} - \widetilde{\mathbf{x}}^{(j)}\right) \\ = \omega_{2^{j+1}}^{\mu^{(j)}} \operatorname{diag}\left(\omega_{2^{L_j}}^{\mu^{(j)}p}\right)_{p=0}^{2^{L_j}-1} &\mathbf{F}_{2^{L_j}} \operatorname{diag}\left(\omega_{2^{j+1}}^{r}\right)_{r=0}^{2^{L_j}-1} \mathbf{D}_{2^{L_j}}^{2^j-\mu^{(j)}} \left(2\widetilde{\mathbf{x}}_0^{(j+1)} - \widetilde{\mathbf{x}}^{(j)}\right).\end{aligned}$$

Hence, we get the vectors

$$\begin{aligned}\widetilde{\mathbf{x}}_0^{(j+1)} = \frac{1}{2}\Bigg(&\omega_{2^{j+1}}^{-\mu^{(j)}} \mathbf{D}_{2^{L_j}}^{2^j-\mu^{(j)}} \operatorname{diag}\left(\omega_{2^{j+1}}^{-r}\right)_{r=0}^{2^{L_j}-1} \mathbf{F}_{2^{L_j}}^{-1} \\ &\cdot \operatorname{diag}\left(\omega_{2^{L_j}}^{-\mu^{(j)}p}\right)_{p=0}^{2^{L_j}-1} \left(\widehat{x}_{2^{J-L_j}p+2^{J-j-1}}\right)_{p=0}^{2^{L_j}-1} + \widetilde{\mathbf{x}}^{(j)}\Bigg)\end{aligned}$$

and $\widetilde{\mathbf{x}}_1^{(j+1)} = \widetilde{\mathbf{x}}^{(j)} - \widetilde{\mathbf{x}}_0^{(j+1)}$ or, in an explicit formulation,

$$\widetilde{\mathbf{x}}_1^{(j+1)} = \frac{1}{2}\left(-\omega_{2^{j+1}}^{-\mu^{(j)}}\ \mathbf{D}_{2^{L_j}}^{2^j-\mu^{(j)}}\ \mathrm{diag}\left(\omega_{2^{j+1}}^{-r}\right)_{r=0}^{2^{L_j}-1}\ \mathbf{F}_{2^{L_j}}^{-1}\right.$$
$$\left.\cdot\ \mathrm{diag}\left(\omega_{2^{L_j}}^{-\mu^{(j)}p}\right)_{p=0}^{2^{L_j}-1}\ \left(\widehat{x}_{2^{J-L_j}p+2^{J-j-1}}\right)_{p=0}^{2^{L_j}-1} + \widetilde{\mathbf{x}}^{(j)}\right).$$

We obtain the vectors $\mathbf{x}_0^{(j+1)}$ and $\mathbf{x}_1^{(j+1)}$ by inserting the support vectors $\widetilde{\mathbf{x}}_0^{(j+1)}$ and $\widetilde{\mathbf{x}}_1^{(j+1)}$ according to the first support index $\mu^{(j)}$. Then we set in both cases

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

The reconstruction of the vector $\widetilde{\mathbf{x}}_0^{(j+1)}$, proceeding as described above, only requires $\mathcal{O}(2^{L_j}\log 2^{L_j})$ arithmetical operations. The second support vector $\widetilde{\mathbf{x}}_1^{(j+1)}$ can then be obtained by 2^{L_j} further additions.

Example 5.3 Let $N = 2^3 = 8$ and $\mathbf{x} \in \mathbb{R}_+^N$ be given by

$$\mathbf{x} = \Big(1,1,0,0,0,0,0,0\Big)^T.$$

Assuming that its Fourier transform

$$\widehat{\mathbf{x}} = \Big(2, 1+\omega_8, 1+\omega_8^2, 1+\omega_8^3, 1+\omega_8^4, 1+\omega_8^5, 1+\omega_8^6, 1+\omega_8^7\Big)^T$$

is given, we want to reconstruct $\mathbf{x}$ step-by-step, using either the method of case 1 or 2.

Let us begin with one Fourier value $\mathbf{x}^{(0)} = \widehat{x}_0 = 2$. Applying the first case, we obtain

$$\mathbf{x}_0^{(1)} = \frac{1}{2}\left(\omega_2^0\,\mathbf{F}_1^{-1}\,\widehat{x}_4 + \mathbf{x}^{(0)}\right) = \frac{1}{2}(1+\omega_8^4+2) = 1$$

and

$$\mathbf{x}_1^{(1)} = \mathbf{x}^{(0)} - \mathbf{x}_0^{(1)} = 2-1 = 1$$

and therefore the next periodization

$$\mathbf{x}^{(1)} = \Big(1,1\Big)^T.$$

It holds that $m_1 > 2^0 = 1$ so we proceed computing $\mathbf{x}^{(2)}$ applying case 1. This yields

$$\begin{aligned}\mathbf{x}_0^{(2)} &= \frac{1}{2}\left(\operatorname{diag}\left(\omega_4^{-\ell}\right)_{\ell=0}^{1}\ \mathbf{F}_2^{-1}\ \left(\widehat{x}_{2(2k+1)}\right)_{k=0}^{1} + \mathbf{x}^{(1)}\right)\\ &= \frac{1}{2}\left(\begin{pmatrix}1 & 0\\ 0 & \mathrm{i}\end{pmatrix}\frac{1}{2}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}\begin{pmatrix}1-\mathrm{i}\\ 1+\mathrm{i}\end{pmatrix} + \begin{pmatrix}1\\ 1\end{pmatrix}\right)\\ &= \frac{1}{2}\left(\begin{pmatrix}1\\ 1\end{pmatrix} + \begin{pmatrix}1\\ 1\end{pmatrix}\right) = \begin{pmatrix}1\\ 1\end{pmatrix}\end{aligned}$$

and the second partial vector

$$\mathbf{x}_1^{(2)} = \mathbf{x}^{(1)} - \mathbf{x}_0^{(2)} = \begin{pmatrix}1\\ 1\end{pmatrix} - \begin{pmatrix}1\\ 1\end{pmatrix} = \begin{pmatrix}0\\ 0\end{pmatrix}$$

such that

$$\mathbf{x}^{(2)} = \Big(1, 1, 0, 0\Big)^T.$$

We have $m_2 \leq 2^1 = 2$ for the periodization $\mathbf{x}^{(2)}$ and hence can apply the second case for the reconstruction of $\mathbf{x}^{(3)}$. We set $L_2 = 1$ and $\mu^{(2)} = 0$ for the first support index of $\mathbf{x}^{(2)}$. The vector

$$\widetilde{\mathbf{x}}^{(2)} = \begin{pmatrix}1\\ 1\end{pmatrix}.$$

contains the support of $\mathbf{x}^{(2)}$. Applying the second case, we obtain

$$\begin{aligned}\widetilde{\mathbf{x}}_0^{(3)} &= \frac{1}{2}\left(\omega_8^{-0}\begin{pmatrix}\omega_8^{-0} & 0\\ 0 & \omega_8^{-1}\end{pmatrix}\mathbf{F}_2^{-1}\begin{pmatrix}\omega_2^{-0} & 0\\ 0 & \omega_2^{-0}\end{pmatrix}\begin{pmatrix}\widehat{x}_1\\ \widehat{x}_5\end{pmatrix} + \begin{pmatrix}1\\ 1\end{pmatrix}\right)\\ &= \frac{1}{2}\left(\begin{pmatrix}1 & 0\\ 0 & \omega_8^{-1}\end{pmatrix}\frac{1}{2}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}\begin{pmatrix}1+\omega_8\\ 1+\omega_8^5\end{pmatrix} + \begin{pmatrix}1\\ 1\end{pmatrix}\right)\\ &= \frac{1}{2}\left(\frac{1}{2}\begin{pmatrix}1 & 0\\ 0 & \omega_8^{-1}\end{pmatrix}\begin{pmatrix}2+\omega_8+\omega_8^5\\ \omega_8-\omega_8^5\end{pmatrix} + \begin{pmatrix}1\\ 1\end{pmatrix}\right)\\ &= \frac{1}{2}\left(\begin{pmatrix}1\\ 1\end{pmatrix} + \begin{pmatrix}1\\ 1\end{pmatrix}\right) = \begin{pmatrix}1\\ 1\end{pmatrix}\end{aligned}$$

and therefore

$$\widetilde{\mathbf{x}}_1^{(3)} = \widetilde{\mathbf{x}}^{(2)} - \widetilde{\mathbf{x}}_0^{(3)} = \begin{pmatrix}1\\ 1\end{pmatrix} - \begin{pmatrix}1\\ 1\end{pmatrix} = \begin{pmatrix}0\\ 0\end{pmatrix}.$$

Plugging in the support vectors $\widetilde{\mathbf{x}}_0^{(3)}$ and $\widetilde{\mathbf{x}}_1^{(3)}$ yields the correct reconstruction

$$\mathbf{x}^{(3)} = \Big(1,1,0,0,0,0,0,0\Big)^T = \mathbf{x}.$$

The recovery of $\mathbf{x}$ required 6 of 8 Fourier values and 21 arithmetical operations (14 additions, 2 multiplications and 5 dyadic shifts), where we do not count a multiplication by -1.

5.2. Algorithm

The procedure of reconstructing a vector $\mathbf{x} \in \mathbb{R}_+^N$ from as few Fourier values as possible is summarized in the following algorithm. The algorithm iteratively computes $\mathbf{x}$ by applying either the method of case 1 or case 2 in each reconstruction step.

If some a priori information on the vector $\mathbf{x}$, such as a lower bound 2^{s-1} for the support length m, is given, then the iteration procedure may be started with the vector $\mathbf{x}^{(s)}$ of length 2^s. In this case, the periodized vector $\mathbf{x}^{(s)}$ first has to be computed using an inverse FFT algorithm. Once the support has been completely detected, the algorithm can use case 2 for reconstruction as the support of $\mathbf{x}$ already occurs in the periodization $\mathbf{x}^{(s)}$ and only has to be shifted to its right position in $\mathbf{x}$ in order to reconstruct the full vector. See also Chapter 4 for this, where the procedure is a main element of vector reconstruction.

Note that this merely holds for exact data. For noisy data, the first case still might apply even after the full support length is achieved at a certain periodization level.

Usually, we just start the algorithm setting $s = 0$.

Algorithm 5.4 (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 $m = |\text{supp } \mathbf{x}|$;
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 $m_j := \big|\text{supp}\, \mathbf{x}^{(j)}\big|$ and find the first support index $\mu^{(j)}$ of $\mathbf{x}^{(j)}$.

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

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

  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

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

  End (if).

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

$$\widetilde{\mathbf{x}}^{(j)} := \left(x^{(j)}_{(\mu^{(j)}+r) \bmod 2^j}\right)_{r=0}^{2^{L_j}-1} \qquad \text{and} \qquad \mathbf{y}^{(j)} := \left(\widehat{x}_{2^{J-L_j}p+2^{J-j-1}}\right)_{p=0}^{2^{L_j}-1}.$$

  Compute

$$\mathbf{z}^{(j)} := \operatorname{diag}\left(\omega_{2^{L_j}}^{-\mu^{(j)}p}\right)_{p=0}^{2^{L_j}-1} \ \mathbf{F}_{2^{L_j}}^{-1} \operatorname{diag}\left(\omega_{2^{j+1}}^{-(\mu^{(j)}+r) \bmod 2^j}\right)_{r=0}^{2^{L_j}-1} \ \mathbf{y}^{(j)}$$

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

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

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

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

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

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

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

End (if).

End (for).

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

An example for the implementation of this algorithm in MATLAB can be found in Section A.2.

In the algorithm, we have introduced a threshold parameter T which makes sure that at each iteration level, the periodized vector $\mathbf{x}^{(j)}$ only has positive entries. Even in the case of exact data, there might arise some numerical error. When processing noisy data, the threshold parameter T plays an important role and has to be chosen suitably, see Section 5.3.

Remark 5.5 *At every iteration level the algorithm automatically decides whether the first or the second case applies to reconstruct $\mathbf{x}^{(j+1)}$ from $\mathbf{x}^{(j)}$. For this decision, the computation of the support length of $\mathbf{x}^{(j)}$ is required. This can be efficiently done including the known support indices of the preceding periodization $\mathbf{x}^{(j-1)}$. This information is useful as 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}. This implies that only $2m$ components have to be considered in order to find the support length m_j and the first support index $\mu^{(j)}$ of $\mathbf{x}^{(j)}$. As a result, this requires an effort of $\mathcal{O}(m_j)$ arithmetical operations.*

Let us give an outline of the arithmetical complexity of the complete algorithm. If we choose $s = 0$, i.e., we start the reconstruction with $\widehat{x}_0 = \mathbf{x}^{(0)}$, the complexity of the algorithm is at most $\mathcal{O}(m \log m \log(N/m))$. This can be seen as follows: From one

iteration step to the next, the support length m_j of the periodized vectors $\mathbf{x}^{(j)}$ only can increase, but never decrease. Hence, it holds that $m_0 \leq m_1 \leq \cdots \leq m_{J-1} \leq m_J = m$.

In general, we can divide the recovery process into two general parts: After the final support length has been achieved for the first time, say in the periodization $\mathbf{x}^{(L)}$ after the iteration step $j = L-1$, then $2^{L-1} < m_L = m \leq 2^L$ holds. This means that from iteration step $j = L+1$ on, the second case can be applied as we have $m_j \leq 2^L \leq 2^{j-1}$ for $j = L+1, \ldots, J-1$ then. This requires $\mathcal{O}(m \log m)$ arithmetical operations at each level, including three multiplications with diagonal matrices of size $2^L \times 2^L$, an inverse Fourier transform of length 2^L and 2^{L+1} additions.

The first $L+1$ reconstruction steps for $j = 0, \ldots, L$ require either the application of the first or of the second case, depending on the distribution of the nonzero components of the periodization $\mathbf{x}^{(j)}$. Altogether, these steps require at most $\mathcal{O}(2^L L) = \mathcal{O}(m \log m)$ arithmetical operations, caused by the inverse FFT of size 2^j, 2^j complex multiplications, 2^{j+1} additions and a dyadic shift by 2 computing the periodization $\mathbf{x}^{(j+1)}$ at each iteration level which is in complexity comparable to a conventional FFT algorithm of length 2^L.

To sum up, the algorithm to recover $\mathbf{x}$ from Fourier values $\widehat{\mathbf{x}}$ has an overall effort of $\mathcal{O}((J-L)m \log m) = \mathcal{O}(m \log m \log(N/m))$ arithmetical operations.

Remark 5.6 *It is merely possible to achieve the sublinear complexity of the algorithm in case that less than N Fourier samples of the vector $\widehat{\mathbf{x}}$ are employed. The number of required samples depends on the reconstruction case that is applied at the j-th iteration level. For a reconstruction using the first case, 2^j Fourier samples are needed whereas recovering by the second case requires only 2^{L_j} Fourier samples that are contained in the vector $\mathbf{y}^{(j)}$. Let us summarize the overall number of samples required for reconstruction. We assume that we have recovered the periodization $\mathbf{x}^{(L)}$ in the reconstruction step $L-1$ where $L := \lceil \log_2 m \rceil$. Then $2^{L-1} < m \leq 2^L$ holds and at the remaining iteration levels $j = L+1, \ldots, J-1$ the second case can be applied, as seen above. This procedure needs for each step 2^L additional Fourier samples, i.e., $(J-L-1)2^L$ in total. The first case of the method for $j = 0, \ldots, L$ uses at most 2^{L+1} samples for reconstruction. To conclude, the number of required Fourier samples is bounded by $\mathcal{O}(m \log(N/m))$.*

The newly developed algorithm for the reconstruction of real nonnegative vectors from Fourier values may be applied to any vector $\mathbf{x} \in \mathbb{R}_+^N$, whether or not it has small support. In any case, it works efficiently: If the support length of $\mathbf{x}$ is comparatively long and the algorithm cannot take advantage of the vector's structure, it applies the first case, that is more elaborate, at most of the reconstruction levels. However, the effort of

the reconstruction does not exceed $\mathcal{O}(N \log N)$ operations which is the complexity of a regular FFT algorithm.

In case that the support length of $\mathbf{x}$ is relatively small compared to its full vector length, the algorithm automatically detects this fact and benefits from it with regard to computational complexity. Then, under most circumstances, the second case is applied in the algorithm.

One can even benefit from our proposed algorithm if the vector to be recovered does not have small support but actually almost full support length instead. It may happen that the periodizations of a sparse vector occur to have small support such that the second case of the algorithm can be applied in intermediate steps of the reconstruction. This occurs e.g. if the vector $\mathbf{x} \in \mathbb{R}_+^N$ contains several small support pieces which are distributed equidistantly and which coincide over the course of periodizations. Then these support pieces possibly add up to one small support interval which allows to apply case 2 for recovery and therefore implies an enhancement concerning computational complexity.

Example 5.7 Let $N = 2^{10} = 1024$ and choose $\mathbf{x} = \mathbf{x}^{(10)} \in \mathbb{R}_+^{1024}$ with positive entries $x_0 = 1$, $x_{256} = 1$, $x_{512} = 1$ and $x_{768} = 1$. Then $\mathbf{x}^{(9)}$ has two positive entries: $x_0 = 2$ and $x_{256} = 2$. All further periodizations $\mathbf{x}^{(8)}, \ldots, \mathbf{x}^{(0)}$ only have one positive entry: $x_0 = 4$. For this vector, the algorithm applies the second case for $j = 0, \ldots, 8$ with $L_j = 0$.

5.3. Numerical results

In this section, the numerical stability of the proposed algorithm for the reconstruction of a real nonnegative vector $\mathbf{x} \in \mathbb{R}_+^N$ from Fourier data is considered. For this purpose, we assume that perturbed Fourier data $\widehat{\mathbf{y}} = (y_k)_{k=0}^{N-1} \in \mathbb{C}^N$ is given with entries

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

where $\boldsymbol{\varepsilon} = (\varepsilon_k)_{k=0}^{N-1} \in \mathbb{C}^N$ is a noise vector. The vector $\mathbf{x}$ will be recovered from $\widehat{\mathbf{y}}$ using as few as possible Fourier values and operations.

In the case of noisy data, it is of particular importance to identify the support interval correctly in each reconstruction step. This assures that we are able to determine the support interval in all further steps. As above, the identification of the support interval of a periodized vector $\mathbf{x}^{(j)}$ can be achieved by only considering the relevant components

which are given by the support of the preceding periodization $\mathbf{x}^{(j-1)}$. The appropriate choice of the threshold parameter T enables us to distinguish between relevant entries of the periodized vectors and noise. Note that in the case of noisy data the probability is higher that the support interval gets longer and shorter again in the course of the algorithm. We will focus on the particular choice of the parameter T below as this strongly influences the number of events in which each of case 1 oder case 2 are applied.

In the following experiments, we will compare the implementation of our newly developed algorithm (see Appendix A.2) to the inverse Fourier transform `ifft` of MATLAB 2013a. Before we give some numerical examples, we repeat the definition of the SNR value used as a measure for the noise vector $\boldsymbol{\varepsilon} = (\varepsilon_k)_{k=0}^{N-1}$ by which the Fourier data $\widehat{\mathbf{x}}$ is perturbed. As introduced in Definition 4.14, the signal-to-noise-ratio (SNR) is given by

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

Similarly to Section 4.4, we quantify the reconstruction error by an ℓ_2-norm $\|\mathbf{x}-\mathbf{x}'\|_2/N$ where $\mathbf{x}'$ denotes the reconstruction of $\mathbf{x}$ by the proposed algorithm. As we compare the results of the algorithm to an inverse Fourier transform applied to $\widehat{\mathbf{y}}$, we also compute its error $\|\mathbf{x}-\mathbf{F}_N^{-1}\widehat{\mathbf{y}}\|_2/N$. Moreover, the errors of the recovery by the sparse FFT algorithm and an inverse Fourier transform are compared using the SNR values

$$\mathrm{SNR}_{\mathrm{alg}} = 20 \cdot \log_{10} \frac{\|\mathbf{x}\|_2}{\|\mathbf{x}-\mathbf{x}'\|_2} \qquad \text{resp.} \qquad \mathrm{SNR}_{\mathrm{IFFT}} = 20 \cdot \log_{10} \frac{\|\mathbf{x}\|_2}{\|\mathbf{x}-\mathbf{F}_N^{-1}\widehat{\mathbf{y}}\|_2}.$$

Let us first give an example for the reconstruction of a comparatively small vector. We choose $N = 2^8 = 256$ and consider the vector $\mathbf{x} \in \mathbb{R}_+^{256}$ with nonzero entries $x_{50} = 5$, $x_{53} = 8$, $x_{54} = 1$, $x_{179} = 2$, $x_{180} = 7$ and $x_{181} = 4$. The Fourier data $\widehat{\mathbf{x}}$ is perturbed by a uniform noise vector $\boldsymbol{\varepsilon}$ with $\mathrm{SNR} = 20$ and $\mathbf{x}$ is reconstructed from $\widehat{\mathbf{y}} = \widehat{\mathbf{x}} + \boldsymbol{\varepsilon}$ using our algorithm. For the randomly chosen noise vector $\boldsymbol{\varepsilon}$, we have $\|\boldsymbol{\varepsilon}\|_\infty = 2.148$ and $\|\boldsymbol{\varepsilon}\|_1/N = 1.183$.

The algorithm returns a reconstruction $\mathbf{x}'$ with nonzero entries

$$\begin{aligned} x'_{50} &= 5.115, \quad x'_{53} = 7.973, \quad x'_{54} = 0.966, \\ x'_{179} &= 2.032, \quad x'_{180} = 7.044 \quad \text{and} \quad x'_{181} = 3.925. \end{aligned}$$

In particular, the support of $\mathbf{x}$ is correctly identified by the algorithm. Hence, the error of the reconstruction $\mathbf{x}'$ is $\|\mathbf{x}-\mathbf{x}'\|_2/N = 6.014 \cdot 10^{-4}$ whereas the error of the inverse

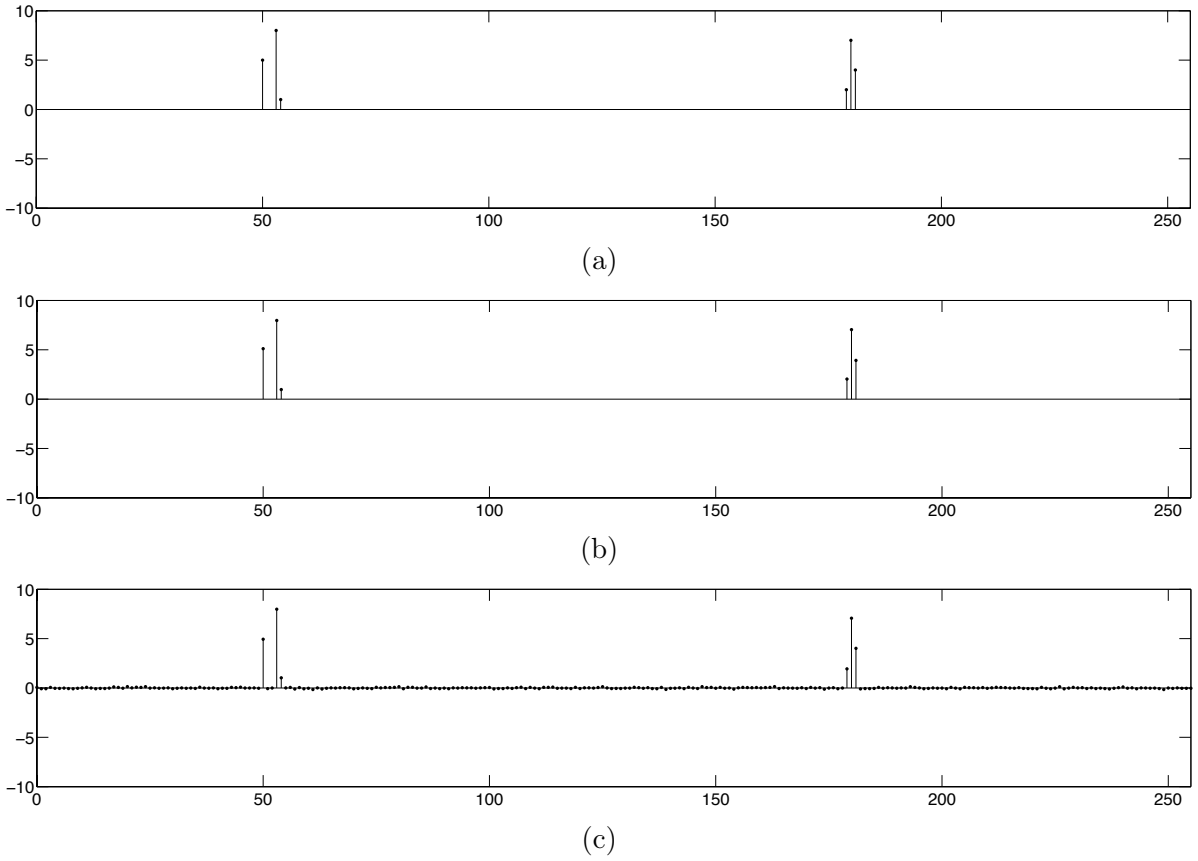


Figure 5.1.: (a) Original vector $\mathbf{x} \in \mathbb{R}_+^{256}$; (b) Reconstruction of $\mathbf{x}$ using the sparse FFT Algorithm 5.4; (c) Reconstruction of $\mathbf{x}$ using an inverse FFT algorithm.

Fourier transform is $\|\mathbf{x} - \mathbf{F}_N^{-1}\widehat{\mathbf{y}}\|_2/N = 4.9 \cdot 10^{-3}$. The SNR errors are $\mathrm{SNR}_{\mathrm{alg}} = 38.2659$ and $\mathrm{SNR}_{\mathrm{IFFT}} = 20$.

In this example, the threshold parameter $T = 0.5$ was chosen. This choice implies an application of the first case in the first four reconstruction steps and the second case in the remaining four steps. The computation of the reconstruction $\mathbf{x}'$ required 48 Fourier values. The results of the reconstruction by our algorithm and an inverse FFT algorithm are illustrated in Figure 5.1.

Remark 5.8 *Remark 4.15 in Section 4.4 on sparse FFT for vectors with small support holds for this example as well. If the vector $\widehat{\mathbf{x}}$ and the SNR value are fixed, the error of the reconstruction by an inverse Fourier transform does not depend on the instance of the noise vector $\boldsymbol{\varepsilon}$. Solely the error of the recovery by the sparse FFT algorithm varies for different noise vectors since not all components of the noisy data vector $\widehat{\mathbf{y}}$ (and hence also of the noise vector) may be involved in the evaluation.*

As a next step, we compare the reconstruction by the proposed algorithm to a reconstruction of $\mathbf{x}$ by an inverse FFT algorithm for a large number of vectors. The vectors $\mathbf{x} \in \mathbb{R}_+^N$ to be reconstructed are randomly chosen for $N = 2^{15} = 32768$ and a support length of $m = 15$ with real nonnegative components $0 \leq x_k \leq 10$. These vectors are perturbed by noise vectors $\boldsymbol{\varepsilon}$ for SNR values between 10 and 50. For each noise level, we consider 100 randomly chosen vectors $\mathbf{x}$ and reconstruct them from the noisy Fourier data $\widehat{\mathbf{y}} = \widehat{\mathbf{x}} + \boldsymbol{\varepsilon}$.

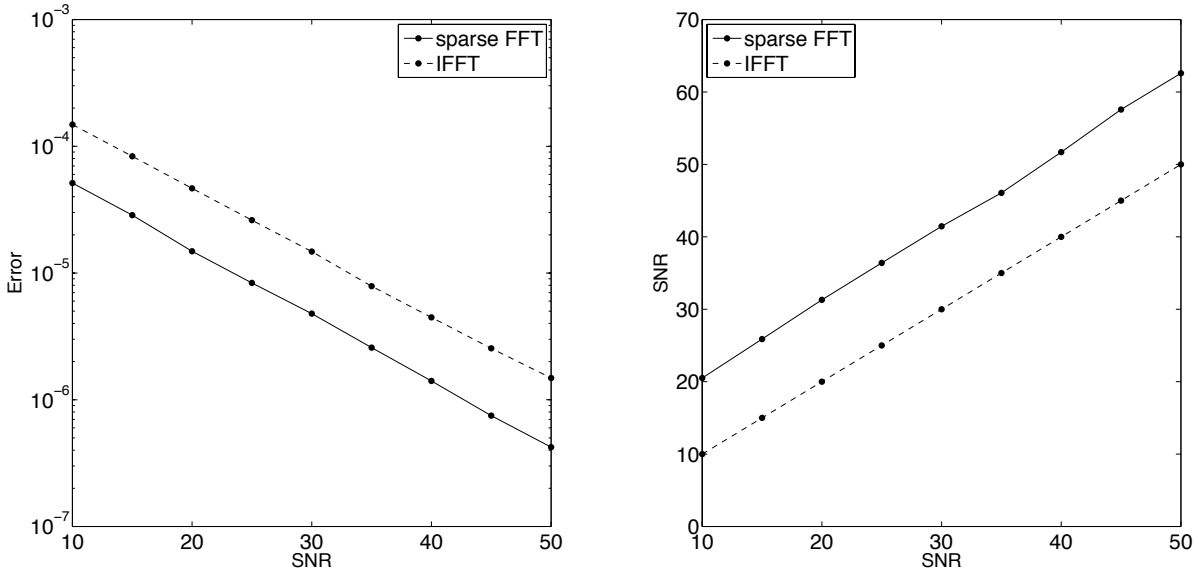
First, we add uniform noise to the Fourier data, then we repeat the experiment employing normally distributed noise (where we use the standard normal distribution). For both kinds of noise, the reconstruction error $\|\mathbf{x} - \mathbf{x}'\|_2/N$ for the reconstruction $\mathbf{x}'$ by our algorithm is compared to the error $\|\mathbf{x} - \mathbf{F}_N^{-1}\widehat{\mathbf{y}}\|_2/N$ of an inverse FFT algorithm. Additionally, we measure the quality of the reconstructions by the SNR values $\mathrm{SNR}_{\mathrm{alg}}$ and $\mathrm{SNR}_{\mathrm{IFFT}}$.

Remark 5.9 *Remark 4.16 holds here similarly. The SNR value of the reconstruction by an inverse FFT algorithm, SNR_{IFFT}, does not depend on the instance of the noise vector $\boldsymbol{\varepsilon}$, but only on its SNR and it even holds that $SNR_{IFFT} = SNR$.*

The results of the numerical experiments can be found in Figure 5.2 and Figure 5.3. The findings show that the algorithm is numerically stable for both uniform and normally distributed noise and that the results are comparable. This is especially remarkable in the sense that for normally distributed noise, single components of the noise vector might be very large. For all settings, the measured reconstruction error is significantly smaller for our deterministic algorithm while it also has a lower complexity, especially in case of a small support compared to the length of $\mathbf{x}$.

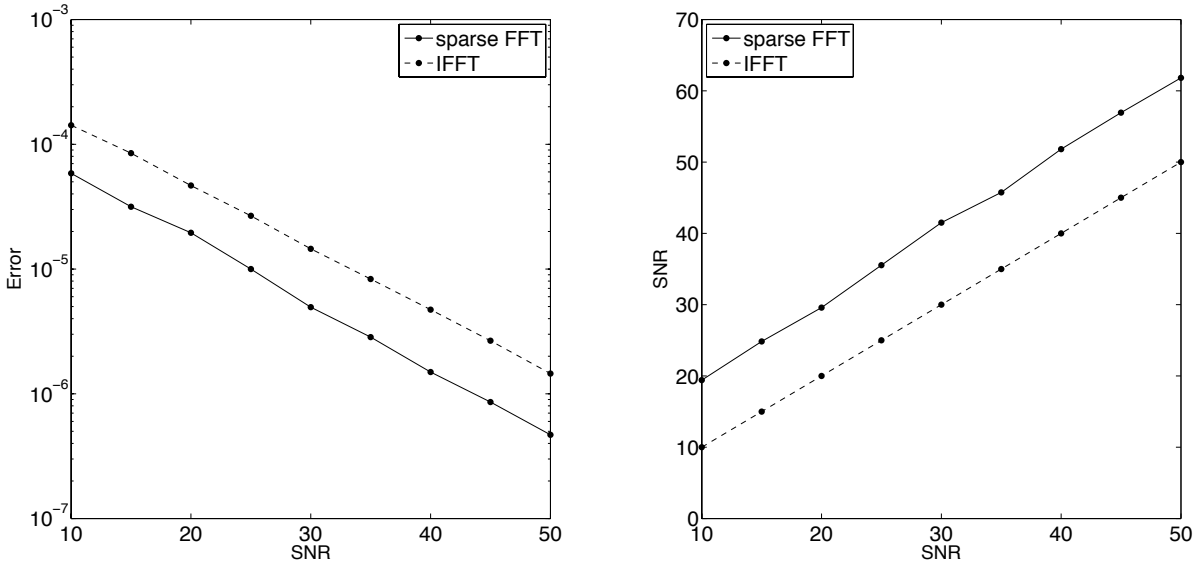
As already stated above, the appropriate choice of the threshold parameter T is of particular importance for an exact and efficient reconstruction in the case of noisy Fourier data. It decides about which entries of a periodization are relevant and which are too small and regarded as noise. This influences the measured support length in the reconstruction steps and hence the decision if case 1 or case 2 is applied at a certain iteration level. The larger T is chosen, the larger is the number of cases in which case 2 can be applied. On the other side, this can be disadvantageous for the exactness of the reconstruction. Hence, the choice of T is a trade-off between the achieved complexity of the reconstruction and the exactness of the result.

We give the values of T used in the experiment as well as the number of reconstruction steps in which the first resp. second case was used for reconstruction in Table 5.1 and 5.2.



(a) (b)

Figure 5.2.: Uniformly distributed noise, $N = 2^{15}$, $m = 15$: (a) Average reconstruction error $\|\mathbf{x} - \mathbf{x}'\|_2/N$ and (b) average $\mathrm{SNR}_{\mathrm{alg}}$ resp. $\mathrm{SNR}_{\mathrm{IFFT}}$ for different noise levels, comparing the sparse FFT Algorithm 5.4 and regular inverse FFT.



(a) (b)

Figure 5.3.: Normally distributed noise, $N = 2^{15}$, $m = 15$: (a) Average reconstruction error $\|\mathbf{x} - \mathbf{x}'\|_2/N$ and (b) average $\mathrm{SNR}_{\mathrm{alg}}$ resp. $\mathrm{SNR}_{\mathrm{IFFT}}$ for different noise levels, comparing the sparse FFT Algorithm 5.4 and regular inverse FFT.

SNR	T	average number first case	average number second case
10	1.4	5.09	9.91
15	0.9	4.74	10.26
20	0.6	4.53	10.47
25	0.4	4.43	10.57
30	0.3	4.60	10.40
35	0.2	4.46	10.54
40	0.1	4.34	10.66
45	0.1	4.50	10.50
50	0.05	4.48	10.52

Table 5.1.: Uniformly distributed noise, $N = 2^{15}$, $m = 15$: Threshold parameter T and number of first/second case applied in the reconstruction of 100 perturbed randomly generated vectors, depending on SNR value.

SNR	T	average number first case	average number second case
10	1.5	4.75	10.25
15	1.0	4.57	10.43
20	0.8	4.41	10.59
25	0.4	4.42	10.58
30	0.3	4.47	10.53
35	0.2	4.45	10.55
40	0.1	4.46	10.54
45	0.1	4.51	10.49
50	0.05	4.49	10.51

Table 5.2.: Normally distributed noise, $N = 2^{15}$, $m = 15$: Threshold parameter T and number of first/second case applied in the reconstruction of 100 perturbed randomly generated vectors, depending on SNR value.

6. An adaptive sparse FFT algorithm

In this chapter, we want to present another approach to a deterministic algorithm for the recovery of sparse vectors from a small amount of Fourier data. As in Chapter 5, we assume the vectors $\mathbf{x} = (x_k)_{k=0}^{N-1} \in \mathbb{R}_+^N$, which we want to reconstruct, to be real and nonnegative. In contrast to the prior setting, we now suppose that the vectors to be reconstructed are M-sparse without necessarily having a small support. The support length does not need to be known nor will it influence the complexity of the algorithm.

The only precondition for the application of the algorithm is the knowledge of the Fourier transform $\widehat{\mathbf{x}} \in \mathbb{C}^N$ of $\mathbf{x}$, some of which will be used for reconstruction. The (possible) sparsity M of $\mathbf{x}$ does not need to be known in advance. We apply an iterative procedure for the reconstruction of $\mathbf{x}$ which adaptively determines the components of $\widehat{\mathbf{x}}$ needed for recovery at each level. As in previous chapters, we use the concept of periodized vectors for the iterative reconstruction. The number of Fourier values that are actually necessary for a stable reconstruction at an iteration level depends on the sparsity of the respective periodization and the distribution of the nonzero components within this vector. To ensure stability of the reconstruction, parameters have to be chosen suitably at each iteration level. In the second part of the chapter, we give estimates for the condition of the occuring matrices depending on these parameters.

The reconstruction procedure for a vector $\mathbf{x} \in \mathbb{R}_+^N$ by the proposed algorithm will require at most $M \log N$ Fourier values and its complexity will not exceed the complexity of an FFT algorithm in case that $\mathbf{x}$ is not sparse. Disregarding possible costs for reconstruction parameters for stabilization, the computational complexity of the algorithm is $\mathcal{O}(M^2 \log(N/M))$.

6.1. An adaptive approach for stable reconstruction from Fourier data

Let us first fix the notation for this chapter. The vector $\mathbf{x} = (x_k)_{k=0}^{N-1} \in \mathbb{R}_+^N$ to be reconstructed is supposed to be of length $N = 2^J$ for some $J \in \mathbb{N}$. Its discrete Fourier transform is given by $\widehat{\mathbf{x}} := \mathbf{F}_N \mathbf{x} \in \mathbb{C}^N$ where $\mathbf{F}_N := \left(\omega_N^{jk}\right)_{j,k=0}^{N-1}$ with $\omega_N^{jk} := \mathrm{e}^{-\frac{2\pi \mathrm{i}}{N}}$ is the Fourier matrix of order N (see also Chapter 2). We keep the definitions of periodized vectors as introduced in Chapter 3.

Moreover, we assume $\mathbf{x}$ to be M-sparse, where $0 \leq M \leq N$, i.e., $\mathbf{x}$ possesses M positive components. However, the sparsity M of $\mathbf{x}$ is not known in advance.

Using these assumptions we develop an adaptive reconstruction procedure. We start by considering the periodization of length one, given by one Fourier value

$$\mathbf{x}^{(0)} = \sum_{k=0}^{N-1} x_k = \widehat{x}_0.$$

Since $\mathbf{x} \in \mathbb{R}_+^N$, we can conclude from $\widehat{x}_0 = 0$ that the vector $\mathbf{x}$ is the zero vector, i.e., it is 0-sparse. In case that $\mathbf{x}^{(0)} = \widehat{x}_0 > 0$, we proceed and consider $\mathbf{x}^{(1)}$. Obviously we have $\mathbf{x}^{(1)} = (x_0^{(1)}, x_1^{(1)})^T$, where $x_0^{(1)} + x_1^{(1)} = \mathbf{x}^{(0)} = \widehat{x}_0$ is already known. Choosing one additional Fourier component $\widehat{x}_1^{(1)} = \widehat{x}_{N/2} = x_0^{(1)} - x_1^{(1)}$ and using $x_1^{(1)} = \mathbf{x}^{(0)} - x_0^{(1)}$, we find that $\widehat{x}_{N/2} = 2x_0^{(1)} - \mathbf{x}^{(0)}$ and hence the two components of the periodization $\mathbf{x}^{(1)}$ are given by

$$x_0^{(1)} = \frac{1}{2}\left(\widehat{x}_{N/2} + \mathbf{x}^{(0)}\right) \qquad \text{resp.} \qquad x_1^{(1)} = \mathbf{x}^{(0)} - x_0^{(1)} = \frac{1}{2}\left(\mathbf{x}^{(0)} - \widehat{x}_{N/2}\right).$$

As $\mathbf{x}^{(1)} = \left(\sum_{k=0}^{N/2-1} x_{2k}, \sum_{k=0}^{N/2-1} x_{2k+1}\right)^T$, we can conclude that all even components of $\mathbf{x}$ vanish if $x_0^{(1)} = 0$ and we do not need to consider them further. In case that $x_1^{(1)} = 0$, it follows analogously that all odd components of $\mathbf{x}$ are zero.

Assume now that we have computed the periodization $\mathbf{x}^{(j)} \in \mathbb{R}_+^{2^j}$ at the j-th iteration level and let $M_j \leq 2^j$ be the determined sparsity of $\mathbf{x}^{(j)}$. Obviously, we have $M_j \leq M$. Further, let

$$0 \leq n_1^{(j)} < n_2^{(j)} < \cdots < n_{M_j}^{(j)} \leq 2^j - 1$$

denote the indices of the corresponding nonzero components of $\mathbf{x}^{(j)}$. By definition of the

periodization, the components of $\mathbf{x}^{(j+1)} = \left(x_k^{(j+1)}\right)_{k=0}^{2^{j+1}-1}$ satisfy the equations

$$x_k^{(j+1)} + x_{k+2^j}^{(j+1)} = x_k^{(j)}, \qquad k = 0, \dots, 2^j - 1. \tag{6.1}$$

In particular, the following holds: If $x_k^{(j)} = 0$ for some $k \in \{0, 1, \dots, 2^j - 1\}$, then it directly follows that $x_k^{(j+1)} = x_{k+2^j}^{(j+1)} = 0$. Hence, in order to compute $\mathbf{x}^{(j+1)}$ we only need to consider the $2M_j$ components $x_{n_k}^{(j+1)}$ and $x_{n_k+2^j}^{(j+1)}$ for $k = 1, \dots, M_j$ as candidates for nonzero entries in $\mathbf{x}^{(j+1)}$ while all other components of $\mathbf{x}^{(j+1)}$ can already be assumed to be zero. Moreover, (6.1) provides M_j conditions on these values, so that we only need M_j suitably chosen further Fourier values to recover $\mathbf{x}^{(j+1)}$.

These considerations lead to the following theorem where we state that $\mathbf{x}^{(j+1)}$ can be uniquely recovered from $\mathbf{x}^{(j)}$ for $j = 0, \dots, J-1$ using M_j additional Fourier components.

Theorem 6.1 *Let $\mathbf{x} \in \mathbb{R}_+^N$, $N = 2^J$ with $J \in \mathbb{N}$, and let $\mathbf{x}^{(j)}$, $j = 0, \dots, J$, be the periodized vectors as given in Definition 3.3.*

Then, for each $j = 0, \dots, J-1$, we have: If $\mathbf{x}^{(j)} \in \mathbb{R}_+^{2^j}$ is M_j-sparse with support indices $0 \le n_1^{(j)} < n_2^{(j)} < \cdots < n_{M_j}^{(j)} \le 2^j - 1$, then the vector $\mathbf{x}^{(j+1)}$ can be uniquely recovered from $\mathbf{x}^{(j)}$ and M_j components $\widehat{x}_{k_1}, \dots, \widehat{x}_{k_{M_j}}$ of $\widehat{\mathbf{x}} = \mathbf{F}_N \mathbf{x}$, where the indices $k_1, \dots, k_{M_j}$ are taken from the set $\left\{2^{J-j-1}(2\ell+1) \,\middle|\, \ell = 0, \dots 2^j - 1\right\}$ such that the matrix

$$\left(\omega_N^{k_p n_r^{(j)}}\right)_{p,r=1}^{M_j} = \left(\mathrm{e}^{-2\pi \mathrm{i} k_p n_r^{(j)}/N}\right)_{p,r=1}^{M_j} \in \mathbb{C}^{M_j \times M_j}$$

is invertible.

Proof. Using the partial vectors $\mathbf{x}_0^{(j+1)} := \left(x_k^{(j+1)}\right)_{k=0}^{2^j-1}$ and $\mathbf{x}_1^{(j+1)} := \left(x_k^{(j+1)}\right)_{k=2^j}^{2^{j+1}-1}$, we have

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

such that it suffices to compute $\mathbf{x}_0^{(j+1)}$ in order to recover $\mathbf{x}^{(j+1)}$. By Lemma 3.7 we find

$$\begin{aligned} \left(\widehat{x}_{2^{J-j-1}k}\right)_{k=0}^{2^{j+1}-1} = \widehat{\mathbf{x}}^{(j+1)} &= \mathbf{F}_{2^{j+1}} \begin{pmatrix} \mathbf{x}_0^{(j+1)} \\ \mathbf{x}_1^{(j+1)} \end{pmatrix} = \mathbf{F}_{2^{j+1}} \begin{pmatrix} \mathbf{x}_0^{(j+1)} \\ \mathbf{x}^{(j)} - \mathbf{x}_0^{(j+1)} \end{pmatrix} \\ &= \left(\omega_{2^{j+1}}^{k\ell}\right)_{k=0,\ell=0}^{2^{j+1}-1,2^j-1} \mathbf{x}_0^{(j+1)} + \left((-1)^k \omega_{2^{j+1}}^{k\ell}\right)_{k=0,\ell=0}^{2^{j+1}-1,2^j-1} \left(\mathbf{x}^{(j)} - \mathbf{x}_0^{(j+1)}\right). \end{aligned} \tag{6.2}$$

We simply observe that the even indexed entries $\widehat{x}_{2\ell}^{(j+1)} = \widehat{x}_\ell^{(j)} = \widehat{x}_{2^{J-j}\ell}$ do not further

contribute to the recovery of the vector $\mathbf{x}_0^{(j+1)}$ but are already determined from $\mathbf{x}^{(j)}$ which is known from the previous step. Let $0 \le n_1^{(j)} < n_2^{(j)} < \cdots < n_{M_j}^{(j)} \le 2^j - 1$ denote the indices of the nonzero entries of $\mathbf{x}^{(j)}$. Then also $\mathbf{x}_0^{(j+1)}$ can only have nonzero entries at these indices. According to this observation, we restrict the vectors to

$$\widetilde{\mathbf{x}}_0^{(j+1)} := \left(x_{n_r}^{(j+1)}\right)_{r=1}^{M_j} \in \mathbb{R}_+^{M_j}, \qquad \widetilde{\mathbf{x}}^{(j)} := \left(x_{n_r}^{(j)}\right)_{r=1}^{M_j} \in \mathbb{R}_+^{M_j}.$$

Further, let $k_1, \ldots, k_{M_j}$ be pairwise different indices from

$$\left\{2^{J-j-1}(2\ell+1) \,\middle|\, \ell = 0, \ldots 2^j - 1\right\},$$

i.e., we have $k_p := 2^{J-j-1}(2\kappa_p + 1)$ with $\kappa_p \in \{0, \ldots, 2^j - 1\}$ for $p = 1, \ldots, M_j$. We now restrict the system in (6.2) to the M_j equations corresponding to these indices $k_1, \ldots, k_{M_j}$ and find

$$\widehat{\mathbf{z}}^{(j+1)} := \begin{pmatrix} \widehat{x}_{k_1} \\ \vdots \\ \widehat{x}_{k_{M_j}} \end{pmatrix} = \begin{pmatrix} \widehat{x}_{2\kappa_1+1}^{(j+1)} \\ \vdots \\ \widehat{x}_{2\kappa_{M_j}+1}^{(j+1)} \end{pmatrix} = \mathbf{A}^{(j+1)}\, \widetilde{\mathbf{x}}_0^{(j+1)} - \mathbf{A}^{(j+1)} \left(\widetilde{\mathbf{x}}^{(j)} - \widetilde{\mathbf{x}}_0^{(j+1)}\right), \tag{6.3}$$

where

$$\mathbf{A}^{(j+1)} = \left(\omega_N^{k_p n_r^{(j)}}\right)_{p,r=1}^{M_j} = \left(\omega_{2^j}^{\kappa_p n_r^{(j)}}\right)_{p,r=1}^{M_j} \cdot \operatorname{diag}\left(\omega_{2^{j+1}}^{n_1^{(j)}}, \ldots, \omega_{2^{j+1}}^{n_{M_j}^{(j)}}\right). \tag{6.4}$$

If $\mathbf{A}^{(j+1)}$ resp. $\left(\omega_{2^j}^{\kappa_p n_r^{(j)}}\right)_{p,r=1}^{M_j}$ is invertible, it follows from (6.3) that

$$\mathbf{A}^{(j+1)}\, \widetilde{\mathbf{x}}_0^{(j+1)} = \frac{1}{2}\left(\widehat{\mathbf{z}}^{(j+1)} + \mathbf{A}^{(j+1)}\, \widetilde{\mathbf{x}}^{(j)}\right),$$

and we can recover $\widetilde{\mathbf{x}}_0^{(j+1)}$ by solving this $M_j \times M_j$ system of equations. Hence, the components of $\mathbf{x}^{(j+1)} \in \mathbb{R}_+^{2^{j+1}}$ are given by

$$x_\ell^{(j+1)} = \begin{cases} \left(\widetilde{\mathbf{x}}_0^{(j+1)}\right)_\ell & \text{for } \ell = n_\ell^{(j)}, \\ \left(\widetilde{\mathbf{x}}^{(j)}\right)_{\ell-2^j} - \left(\widetilde{\mathbf{x}}_0^{(j+1)}\right)_{\ell-2^j} & \text{for } \ell = n_k^{(j)} + 2^j, \\ 0 & \text{else,} \end{cases}$$

which yields the reconstruction of $\mathbf{x}^{(j+1)}$. □

We summarize the reconstruction procedure for M-sparse vectors from Fourier data in the following algorithm.

Algorithm 6.2 (Reconstruction of a vector from Fourier measurements)
Input: $N = 2^J$ (length of the vector $\mathbf{x}$);
Fourier values $\widehat{x}_k$, $k \in K$, where the index set K is chosen adaptively during the algorithm;
threshold parameter T.

1. Set $M := 0$ and $K := \{0\}$. Choose the Fourier value $\widehat{x}_0$.
   If $\widehat{x}_0 < T$, then $\mathbf{x} = \mathbf{0}$ and $I^{(J)} := \emptyset$.
   If $\widehat{x}_0 \geq T$, then set $M := 1$, $I^{(0)} := \{0\}$, and $\widetilde{\mathbf{x}}^{(0)} = \widehat{x}_0$.

2. If $\widehat{x}_0 \geq T$ then
   For $j = 0, \ldots, J-1$

   - Choose M indices $k_p = 2^{J-j-1}(2\kappa_p + 1)$ with $\kappa_p \in \{0, \ldots, 2^j - 1\}$ for $p = 1, \ldots, M$ such that

   $$\mathbf{A}^{(j+1)} := \left(\omega_N^{k_p \ell}\right)_{p=1,\ldots,M;\, \ell \in I^{(j)}}$$

   is well-conditioned and set $K := K \cup \{k_1, \ldots, k_M\}$.

   - Choose the vector of Fourier values $\widehat{\mathbf{z}}^{(j+1)} := (\widehat{x}_{k_p})_{p=1}^M \in \mathbb{C}^M$ and solve the linear system

   $$\mathbf{A}^{(j+1)}\, \widetilde{\mathbf{x}}_0^{(j+1)} = \frac{1}{2}(\widehat{\mathbf{z}}^{(j+1)} + \mathbf{A}^{(j+1)}\, \widetilde{\mathbf{x}}^{(j)}).$$

   - Set $\widetilde{\mathbf{x}}_1^{(j+1)} := \widetilde{\mathbf{x}}^{(j)} - \widetilde{\mathbf{x}}_0^{(j+1)}$ and $\widetilde{\mathbf{x}}^{(j+1)} := ((\widetilde{\mathbf{x}}_0^{(j+1)})^T, (\widetilde{\mathbf{x}}_1^{(j+1)})^T)^T$.
   - Determine the set of active indices $I^{(j+1)} \subset \left(I^{(j)} \cup (I^{(j)} + 2^j)\right)$ by removing all indices in $I^{(j)} \cup (I^{(j)} + 2^j)$ that correspond to entries in $\widetilde{\mathbf{x}}^{(j+1)}$ being smaller than T. Set $M := \#I^{(j+1)}$ being the number of positive entries of $\mathbf{x}^{(j+1)}$.

   End (for).

Output: $I^{(J)}$, the set of active indices in $\mathbf{x}$ with $M = \#I^{(J)}$;
$\widetilde{\mathbf{x}} = \widetilde{\mathbf{x}}^{(J)} = (x_k)_{k \in I^{(J)}}$, the vector restricted to nonzero entries;
K, the index set of used Fourier values from $\widehat{\mathbf{x}}$.

It remains to answer the question how the $M = M_j$ indices $k_1, \ldots, k_{M_j}$ should be chosen depending on the determined set $I^{(j)}$ of indices which corresponds to the nonzero entries of $\mathbf{x}^{(j)}$ at the j-th iteration step. In order to obtain a numerically stable algorithm, these indices have to be chosen in a way such that the matrix $\mathbf{A}^{(j+1)}$ is well-conditioned.

Remark 6.3 *Note that the situation in Chapter 5 is actually a special case of the setting here. There, the vector* $\mathbf{x}$ *is assumed to have short support. Hence, we can restrict our computations to a relatively short interval of consecutive active indices in each periodization* $\mathbf{x}^{(j)}$*.*

This makes the choice of the indices $k_1, \ldots, k_{M_j}$ *comparatively easy. We set*

$$k_p = 2^{j-L_j} p, \qquad p = 0, \ldots, 2^{L_j} - 1,$$

where $L_j = \lceil \log_2 M_j \rceil$ *and* M_j *denotes the support length of the* 2^j*-periodization* $\mathbf{x}^{(j)}$*. Then the matrix* $\mathbf{A}^{(j+1)}$ *can be split into a product of diagonal matrices and a Fourier matrix which ensures a stable reconstruction (for details, see Section 5.1).*

In contrast, the active components of the periodizations $\mathbf{x}^{(j)}$ *in the present setting are not necessarily clustered such that the indices* $k_1, \ldots, k_{M_j}$ *cannot be appropriately chosen in this way.*

We simplify the problem by restricting the adaptive search for suitable indices $k_1, \ldots, k_{M_j}$ in a special way such that the obtained coefficient matrices $\mathbf{A}^{(j+1)}$ in (6.4) are given by Vandermonde matrices with knots on the unit circle. More precisely, we consider $k_p = 2^{J-j-1}(2\kappa_p + 1)$ with $\kappa_p := \sigma(p-1) \bmod 2^j$ for $p = 1, \ldots, M_j$, and we will choose $\sigma = \sigma_j \in \{1, \ldots, 2^j - 1\}$ such that $\mathbf{A}^{(j+1)}$ in (6.4) is of the form

$$\mathbf{A}^{(j+1)} = \mathbf{V}_{M_j} \operatorname{diag}\left(\omega_{2^{j+1}}^{n_1^{(j)}}, \ldots, \omega_{2^{j+1}}^{n_{M_j}^{(j)}}\right),$$

where the Vandermonde matrix

$$\mathbf{V}_{M_j} := \mathbf{V}_{M_j}\left(\omega_{2^j}^{\sigma n_1^{(j)}}, \ldots, \omega_{2^j}^{\sigma n_{M_j}^{(j)}}\right) := \left(\omega_{2^j}^{\sigma(p-1)n_r^{(j)}}\right)_{p,r=1}^{M_j} = \left(\omega_{2^j}^{\kappa_p n_r^{(j)}}\right)_{p,r=1}^{M_j},$$

being determined by the knots $\omega_{2^j}^{\sigma n_1^{(j)}}, \ldots, \omega_{2^j}^{\sigma n_{M_j}^{(j)}}$, is well-conditioned.

We postpone the problem of an efficient computation of a suitable parameter σ, which ensures a well-conditioned Vandermonde matrix, to the next section. In the last part of

this section, we generalize Theorem 6.1 in order to extend a known periodization $\mathbf{x}^{(j)}$ to $\mathbf{x}^{(j+s)}$ with $j+s \leq J$ in one step.

Let us again assume that we have already computed the periodization $\mathbf{x}^{(j)} \in \mathbb{R}_+^{2^j}$ and that $\mathbf{x}^{(j)}$ possesses M_j nonzero entries with indices $n_k^{(j)}$, $k = 1, \ldots, M_j$. This means that the nonzero components of $\mathbf{x} = \mathbf{x}^{(J)}$ must be contained in the set

$$\left\{ x_{n_k^{(j)}+2^j\ell} \,\middle|\, k = 1, \ldots, M_j,\ \ell = 0, \ldots, 2^{J-j}-1 \right\}. \tag{6.5}$$

We consider the partial vectors

$$\mathbf{y}^{(k)} = \left(y_\ell^{(k)}\right)_{\ell=0}^{2^{J-j}-1} := \left(x_{n_k^{(j)}+2^j\ell}\right)_{\ell=0}^{2^{J-j}-1} \in \mathbb{R}_+^{2^{J-j}}, \qquad k = 1, \ldots, M_j,$$

which contain the possibly nonzero components of $\mathbf{x}$ given in the set (6.5). For these partial vectors, it holds that the sum of their entries equals the corresponding component in $\mathbf{x}^{(j)}$, i.e.,

$$\sum_{\ell=0}^{2^{J-j}-1} y_\ell^{(k)} = \sum_{\ell=0}^{2^{J-j}-1} x_{n_k^{(j)}+2^j\ell} = x_{n_k^{(j)}}^{(j)}.$$

Obviously, we only need to determine these vectors $\mathbf{y}^{(k)}$ in order to compute the complete vector $\mathbf{x} = \mathbf{x}^{(J)}$. With the objective to recover $\mathbf{x}^{(j+s)}$ from $\mathbf{x}^{(j)}$, we similarly consider the vectors

$$\mathbf{y}^{(k,s)} = \left(y_r^{(k,s)}\right)_{r=0}^{2^s-1} := \left(x_{n_k^{(j)}+2^j\ell}^{(j+s)}\right)_{\ell=0}^{2^s-1} \in \mathbb{R}_+^{2^s-1}, \qquad k = 1, \ldots, M_j,$$

that determine $\mathbf{x}^{(j+s)}$ completely. These vectors contain the potentially nonzero components of $\mathbf{x}^{(j+s)}$ where each $\mathbf{y}^{(k,s)}$ again arises from one nonzero entry $x_{n_k^{(j)}}^{(j)}$ in $\mathbf{x}^{(j)}$ such that $\sum_{r=0}^{2^s-1} y_r^{(k,s)} = x_{n_k^{(j)}}^{(j)}$. Note that $\mathbf{y}^{(k,s)} \in \mathbb{R}_+^{2^s}$ is the 2^s-periodization of $\mathbf{y}^{(k)} \in \mathbb{R}_+^{2^{J-j}}$.

Theorem 6.4 *Let $\mathbf{x} \in \mathbb{R}_+^N$, $N = 2^J$ with $J \in \mathbb{N}$, and let $\mathbf{x}^{(j)}$, $j = 0, \ldots, J$, be the periodizations as in Definition 3.3. For a given j with $1 \leq j < J$, assume that $\mathbf{x}^{(j)}$ is known. Suppose that $\mathbf{x}^{(j)} \in \mathbb{R}_+^{2^j}$ is M_j-sparse with support indices $0 \leq n_1^{(j)} < n_2^{(j)} < \ldots < n_{M_j}^{(j)} \leq 2^j - 1$. Then the vector $\mathbf{x}^{(j+s)}$ (with $j+s \leq J$) can be uniquely recovered from $\mathbf{x}^{(j)} \in \mathbb{R}_+^{2^j}$ and $M_j 2^s$ components of the vector $\widehat{\mathbf{x}} = \mathbf{F}_N \mathbf{x}$ of the form*

$$\widehat{x}_{2^{J-j-s}r+2^{J-j}k\sigma}, \qquad r = 0, \ldots 2^s - 1,\ k = 1, \ldots M_j,$$

where σ is chosen such that the Vandermonde matrix $\mathbf{V}_{M_j}\left(\omega_{2^j}^{\sigma n_1^{(j)}},\dots,\omega_{2^j}^{\sigma n_{M_j}^{(j)}}\right)$ is invertible and well-conditioned.

Proof. Since $\mathbf{x}^{(j)}$ is already known, the indices of the nonzero components of $\mathbf{x}^{(j+s)}$ have to be contained in the set $\left\{n_k^{(j)}+2^j\ell \,\middle|\, k=1,\dots,M_j,\ \ell=0,\dots,2^s-1\right\}$ and hence the possibly nonzero components of $\mathbf{x}^{(j+s)}$ are given by

$$\left\{x^{(j+s)}_{n_k^{(j)}+2^j\ell} \,\middle|\, k=1,\dots,M_j,\ \ell=0,\dots,2^s-1\right\}.$$

Using the definition of the vectors $\mathbf{y}^{(k)}$ above, we find for $r=0,\dots,N-1$,

$$\begin{aligned}\widehat{x}_r &= \sum_{k=1}^{M_j}\sum_{\ell=0}^{2^{J-j}-1} x_{n_k^{(j)}+2^j\ell}\,\omega_N^{(n_k^{(j)}+2^j\ell)r}\\ &= \sum_{k=1}^{M_j}\left(\sum_{\ell=0}^{2^{J-j}-1} y_\ell^{(k)}\omega_N^{2^j\ell r}\right)\omega_N^{n_k^{(j)}r} = \sum_{k=1}^{M_j}\widehat{y}^{(k)}_{r \bmod 2^{J-j}}\,\omega_N^{n_k^{(j)}r},\end{aligned}$$

where we use the notation $\widehat{\mathbf{y}}^{(k)}=\left(\widehat{y}_r^{(k)}\right)_{r=0}^{2^{J-j}-1}=\mathbf{F}_{2^{J-j}}\mathbf{y}^{(k)}$. As known from Lemma 3.7, the 2^s-periodization $\mathbf{y}^{(k,s)}$ of $\mathbf{y}^{(k)}$ has the Fourier transform $\widehat{\mathbf{y}}^{(k,s)}=\left(\widehat{y}^{(k)}_{2^{J-j-s}r}\right)_{r=0}^{2^s-1}$.

We consider the Fourier values $\widehat{x}_{2^{J-j-s}r+2^{J-j}\nu\sigma}$ for $r=0,\dots,2^s-1$, $\nu=1,\dots,M_j$ and some fixed $\sigma\in\{0,\dots,2^j-1\}$ and find

$$\widehat{x}_{2^{J-j-s}r+2^{J-j}\nu\sigma}=\sum_{k=1}^{M_j}\widehat{y}^{(k)}_{2^{J-j-s}r}\,\omega_{2^{j+s}}^{n_k^{(j)}r}\,\omega_{2^j}^{\nu\sigma n_k^{(j)}}. \tag{6.6}$$

Thus,

$$(\widehat{x}_{2^{J-j-s}r+2^{J-j}\nu\sigma})_{\nu=1,r=0}^{M_j,2^s-1}=\mathbf{V}_{M_j}(\sigma)\,\left(\omega_{2^{j+s}}^{n_k^{(j)}r}\,\widehat{y}^{(k)}_{2^{J-j-s}r}\right)_{k=1,r=0}^{M_j,2^s-1}, \tag{6.7}$$

where the Vandermonde matrix $\mathbf{V}_{M_j}(\sigma):=\mathbf{V}_{M_j}(\omega_{2^j}^{\sigma n_1^{(j)}},\dots,\omega_{2^j}^{\sigma n_{M_j}^{(j)}})$ of size $M_j\times M_j$ is determined by the knots $\omega_{2^j}^{\sigma n_1^{(j)}},\dots,\omega_{2^j}^{\sigma n_{M_j}^{(j)}}$ on the unit circle. For determining $\mathbf{y}^{(k,s)}$, we first need to solve the system (6.7) to obtain $\left(\omega_{2^{j+s}}^{n_k^{(j)}r}\,\widehat{y}^{(k)}_{2^{J-j-s}r}\right)_{k=1,r=0}^{M_j,2^s-1}$. This can be done in $\mathcal{O}(M_j^2 2^s)$ operations when we apply a QR decomposition to the Vandermonde matrix whose computation requires $\mathcal{O}(M_j^2)$ operations, see [11]. Afterwards, we compute the M_j

vectors $\mathbf{y}^{(k,s)}$ using a fast inverse Fourier transform of length 2^s requiring $\mathcal{O}(M_j 2^s s)$ arithmetical operations. Altogether, the computation of $\mathbf{x}^{(j+s)}$ requires $\mathcal{O}(M_j 2^s \max\{M_j, s\})$ operations. □

Remark 6.5 *The choice of Fourier values in (6.6) can be understood as shifted sampling. In [47], the used function values are similar to the ones used above. In contrast to our approach, Potts, Tasche and Volkmer use the procedure to split polynomials of large sparsity into several polynomials of lower sparsity which can then be recovered by applying Prony's method. The idea of shifted sampling has originally been proposed in [29].*

Remark 6.6 *Observe that for $s = 1$, Theorem 6.1 is a special case of Theorem 6.4 where we only need to fix one parameter σ in order to determine the coefficient matrices $\mathbf{A}^{(j+1)}$.*

Assuming that $\mathbf{x}$ has the sparsity $M = M_J$ with $2^{L-1} < M \le 2^L$, and that we can find a σ at each iteration level such that the matrix $\mathbf{V}_{M_j}(\omega_{2^j}^{\sigma n_1^{(j)}}, \ldots, \omega_{2^j}^{\sigma n_{M_j}^{(j)}})$ is well-conditioned, then we need at most $\mathcal{O}(2^L L + M^2 \log 2^{J-L}) = \mathcal{O}(M^2 \log(N/M))$ arithmetical operations to recover $\mathbf{x}$ not counting the effort to find σ at each iteration step.

If the condition number of the quadratic matrix $\mathbf{V}_{M_j}(\omega_{2^j}^{\sigma n_1^{(j)}}, \ldots, \omega_{2^j}^{\sigma n_{M_j}^{(j)}})$ is not small enough, we may add further lines and use a rectangular Fourier matrix. In this case, we need to enlarge the system of equations (6.7) and consider

$$(\widehat{x}_{2^{J-j-s}r+2^{J-j}k\sigma})_{k=1,r=0}^{M_j',2^s-1} = \mathbf{V}_{M_j',M_j}^{(j)}(\sigma) \left(\omega_{2^{j+s}}^{n_p^{(j)} r}\, \widehat{y}_{2^{J-j-s}r}^{(p)}\right)_{p=1,r=0}^{M_j,2^s-1}$$

with $M_j' \ge M_j$, where we need now $2^s M_j'$ Fourier values and apply the rectangular Vandermonde matrix $\mathbf{V}_{M_j',M_j}^{(j)}(\sigma) = \left(\omega_{2^j}^{\sigma k n_p^{(j)}}\right)_{k=0,p=1}^{M_j'-1,M_j}$.

6.2. Vandermonde matrices with knots on the unit circle

We now investigate the question how to find an optimal $\sigma = \sigma_j$ at each iteration step in order to ensure a well-conditioned Vandermonde system and hence a stable reconstruction in Algorithm 6.2. In this section, we present results on bounds for the condition of the Vandermonde matrices as well as some considerations on the possible distribution of knots on the unit circle.

For simplicity, we neglect the subscripts j in this section. Let $0 \leq n_1 < n_2 < \ldots < n_M < N$ be a known set of indices, i.e., in our case the indices of the support components of a vector. We want to find an optimal parameter $\sigma \in \{1, \ldots, N-1\}$ such that the Vandermonde matrix $\mathbf{V}_{M',M}$ (with $N > M' \geq M$) of size $M' \times M$ determined by the knots $\omega_N^{\sigma n_k}$, $k = 1, \ldots, M$, has a suitably bounded condition number. At the same time, M' should stay in the range of M in order to reduce the costs for solving a corresponding Vandermonde system.

In order to efficiently find an optimal parameter σ at each iteration step, we investigate Vandermonde matrices in more detail. It is well-known that a quadratic Vandermonde matrix $\mathbf{V}_{M,M}$ is invertible (i.e., $\det(\mathbf{V}_{M,M}) \neq 0$) if and only if the support indices $(\sigma n_k \bmod N)$ are pairwise distinct for $k = 1, \ldots, M$, see e.g. [50, Example 2.18]. Thus, we can choose $\sigma = 1$ to ensure invertibility of $\mathbf{V}_{M,M}$. This choice is non-adaptive and not related to the knowledge of the index set $\{n_1, \ldots, n_M\}$. Therefore, it can lead to large condition numbers.

Indeed, the condition of the rectangular matrix $\mathbf{V}_{M',M}$ strongly depends on the distribution of the M support indices or, equivalently, on the distribution of the values $\omega_N^{\sigma n_k}$, $k = 1, \ldots, M$, on the unit circle. The condition number of $\mathbf{V}_{M,M}$ can even be one if and only if the values $\omega_N^{\sigma n_k}$ are equidistantly distributed on the unit circle, i.e., if M is a divisor of N and

$$\{\omega_N^{\sigma n_k} \,|\, k = 1, \ldots, M\} = \{c\, \omega_M^r \,|\, k = 1, \ldots, M\},$$

where c is a unitary constant, see [4].

Recall that the condition number of an $(M' \times M)$-matrix $\mathbf{V}_{M',M}(\sigma)$ based on the spectral norm is determined by

$$\kappa_2\left(\mathbf{V}_{M',M}(\sigma)\right) := \frac{\max\limits_{\mathbf{u} \in \mathbb{C}^M, \|\mathbf{u}\|_2 = 1} \|\mathbf{V}_{M',M}(\sigma)\, \mathbf{u}\|_2}{\min\limits_{\mathbf{u} \in \mathbb{C}^M, \|\mathbf{u}\|_2 = 1} \|\mathbf{V}_{M',M}(\sigma)\, \mathbf{u}\|_2}.$$

In order to bound the condition number of $\mathbf{V}_{M',M}$, an observation by Moitra [30] will be helpful. We slightly modify his result and give a different proof that directly adapts Hilbert's inequality from [31]. The condition number of the Vandermonde matrix $\mathbf{V}_{M',M}$ depends on the minimal distance d_σ between two knots on the unit circle.

Theorem 6.7 *Let $0 \leq n_1 < n_2 < \ldots < n_M < N$ be a given set of indices. For a given*

parameter $\sigma \in \{1, \dots, N\}$ *let*

$$d_\sigma := \min_{1 \le k < \ell \le M} (\pm\sigma(n_\ell - n_k)) \bmod N \tag{6.8}$$

be the smallest (periodic) distance between two indices σn_ℓ *and* σn_k*, and assume that* $d_\sigma > 0$*. Then the condition number* $\kappa_2(\mathbf{V}_{M',M}(\sigma))$ *of the Vandermonde matrix*

$$V_{M',M}(\sigma) := \left(\omega_N^{\sigma n_k \ell}\right)_{\ell=0,k=1}^{M'-1,M}$$

satisfies

$$\kappa_2\left(\mathbf{V}_{M',M}(\sigma)\right)^2 \le \frac{M' + \frac{N}{d_\sigma}}{M' - \frac{N}{d_\sigma}}, \tag{6.9}$$

provided that $M' > \frac{N}{d_\sigma}$*.*

Proof. 1. Assume that $\tilde{n}_k := \frac{\sigma n_k \bmod N}{N}$ for $k = 1, \dots, M$. By assumption, the values $\widetilde{n}_k$ are distinct numbers in $[0, 1)$ and the minimal (cyclic) distance between two of these values is d_σ/N. Considering the matrix $(\mathbf{V}_{M',M}(\sigma))^* \mathbf{V}_{M',M}(\sigma) = (c_{k\ell})_{k,\ell=1}^{M}$, we find

$$\begin{aligned} c_{k\ell} &= \sum_{r=0}^{M'-1} \mathrm{e}^{\frac{2\pi\mathrm{i}\sigma n_k r}{N}} \mathrm{e}^{-\frac{2\pi\mathrm{i}\sigma n_\ell r}{N}} = \sum_{r=0}^{M'-1} \mathrm{e}^{-2\pi\mathrm{i}(\tilde{n}_\ell - \tilde{n}_k) r} \\ &= \begin{cases} \frac{1-\mathrm{e}^{-2\pi\mathrm{i}(\tilde{n}_\ell - \tilde{n}_k)M}}{1-\mathrm{e}^{-2\pi\mathrm{i}(\tilde{n}_\ell - \tilde{n}_k)}} & \text{if } \tilde{n}_k \ne \tilde{n}_\ell, \\ M' & \text{if } \tilde{n}_k = \tilde{n}_\ell, \end{cases} \\ &= \begin{cases} \frac{\mathrm{e}^{-2\pi\mathrm{i}(\tilde{n}_\ell - \tilde{n}_k)M'/2} \sin(2\pi(\tilde{n}_\ell - \tilde{n}_k)M'/2)}{\mathrm{e}^{-2\pi\mathrm{i}(\tilde{n}_\ell - \tilde{n}_k)/2} \sin(2\pi(\tilde{n}_\ell - \tilde{n}_k)/2)} & \text{if } \tilde{n}_k \ne \tilde{n}_\ell, \\ M' & \text{if } \tilde{n}_k = \tilde{n}_\ell, \end{cases} \end{aligned}$$

i.e., we have

$$c_{k\ell} = \mathrm{e}^{-2\pi\mathrm{i}(\tilde{n}_\ell - \tilde{n}_k)(M'-1)/2} D_{M'}\left(2\pi(\tilde{n}_\ell - \tilde{n}_k)\right), \tag{6.10}$$

where

$$D_{M'}(x) = \begin{cases} \frac{\sin(M'x/2)}{\sin(x/2)} & \text{if } x \ne 0, \\ M' & \text{if } x = 0, \end{cases}$$

denotes the Dirichlet kernel. Hence, the symmetric and positive semidefinite ma-

trix

$$\mathbf{C}_M = (D_{M'}(2\pi(\tilde{n}_\ell - \tilde{n}_k)))_{k,\ell=1}^M$$

possesses the same eigenvalues as $(\mathbf{V}_{M',M}(\sigma))^*\mathbf{V}_{M',M}(\sigma)$ since (6.10) yields

$$(\mathbf{V}_{M',M}(\sigma))^*\mathbf{V}_{M',M}(\sigma) = \operatorname{diag}\left(\mathrm{e}^{2\pi\mathrm{i}\tilde{n}_k(M'-1)/2}\right)_{k=1}^M \mathbf{C}_M \operatorname{diag}\left(\mathrm{e}^{-2\pi\mathrm{i}\tilde{n}_\ell(M'-1)/2}\right)_{\ell=1}^M.$$

Let us first consider the Frobenius norm $\|\mathbf{V}_{M',M}(\sigma)\|_F$. Since $\mathbf{C}_M(\ell,\ell) = M'$ for all $\ell = 1,\dots,M$ it follows that

$$\|\mathbf{V}_{M',M}(\sigma)\|_F^2 = \operatorname{tr}(\mathbf{V}_{M',M}(\sigma))^*\mathbf{V}_{M',M}(\sigma)) = \operatorname{tr}\mathbf{C}_M = MM',$$

such that the spectral norm is bounded by

$$\|\mathbf{V}_{M',M}(\sigma)\|_2 \le \|\mathbf{V}_{M',M}(\sigma)\|_F = \sqrt{M'M}.$$

2. We consider for arbitrary $\mathbf{u} \in \mathbb{C}^M$

$$\begin{aligned}\mathbf{u}^T\mathbf{C}_M\overline{\mathbf{u}} &= \sum_{k=1}^M\sum_{\ell=1}^M u_k\overline{u}_\ell\, D_{M'}(2\pi(\tilde{n}_\ell - \tilde{n}_k)) \\ &= M'\sum_{k=1}^M |u_k|^2 + \sum_{\substack{k,\ell=1\\k\neq\ell}}^M u_k\overline{u}_\ell\,\frac{\sin(M'\pi(\tilde{n}_\ell - \tilde{n}_k))}{\sin(\pi(\tilde{n}_\ell - \tilde{n}_k))}.\end{aligned}$$

We recall the following result by Montgomery and Vaughan, see [31, Theorem 1]. Let $0 \le x_1 < x_2 < \ldots < x_R < 1$ and

$$\delta = \min\left\{|(x_k - x_\ell) \bmod 1| \;|\; k,\ell = 1,\dots,R,\ k \neq \ell\right\}.$$

Then

$$\left|\sum_{\substack{k,\ell=1\\k\neq\ell}}^R \frac{u_k\overline{u}_\ell}{\sin(\pi(x_k - x_\ell))}\right| \le \frac{1}{\delta}\sum_{k=1}^R |u_k|^2. \tag{6.11}$$

Using

$$\sin(M'\pi(\tilde{n}_\ell - \tilde{n}_k)) = \frac{1}{2\mathrm{i}}\left(\mathrm{e}^{M'\pi\mathrm{i}(\tilde{n}_\ell-\tilde{n}_k)} - \mathrm{e}^{-M'\pi\mathrm{i}(\tilde{n}_\ell-\tilde{n}_k)}\right)$$

we now divide our sum into two parts and apply equation (6.11) to both of them, with u_k replaced by $u_k\mathrm{e}^{-M'_j\pi\mathrm{i}\tilde{n}_k}$ and $u_k\mathrm{e}^{M'_j\pi\mathrm{i}\tilde{n}_k}$ and $\overline{u}_\ell$ replaced by $\overline{u}_\ell\mathrm{e}^{M'_j\pi\mathrm{i}\tilde{n}_\ell}$ and $\overline{u}_\ell\mathrm{e}^{-M'_j\pi\mathrm{i}\tilde{n}_\ell}$, respectively. Thus, we obtain

$$\left|\sum_{\substack{k,\ell=1\\k\neq\ell}}^{M} u_k\overline{u}_\ell\,\frac{\sin(M'\pi(\tilde{n}_\ell-\tilde{n}_k))}{\sin(\pi(\tilde{n}_\ell-\tilde{n}_k))}\right| \leq \frac{N}{d_\sigma}\|\mathbf{u}\|_2.$$

This observation yields

$$\begin{aligned}\|V_{M',M}(\sigma)\|_2^2 &= \max_{\mathbf{u}\in\mathbb{C}^M,\|\mathbf{u}\|_2=1} \mathbf{u}^T\mathbf{C}_M\overline{\mathbf{u}} \;\leq\; M' + \frac{N}{d_\sigma},\\ \|(V_{M',M}(\sigma))^{-1}\|_2^2 &= \left(\min_{\mathbf{u}\in\mathbb{C}^M,\|\mathbf{u}\|_2=1} \mathbf{u}^T\mathbf{C}_M\overline{\mathbf{u}}\right)^{-1} \leq \left(M' - \frac{N}{d_\sigma}\right)^{-1}.\end{aligned}$$

Hence, we find the condition of $V_{M',M}(\sigma)$

$$\kappa_2\left(\mathbf{V}_{M',M}(\sigma)\right)^2 = \frac{\|V_{M',M}(\sigma)\|_2^2}{\|(V_{M',M}(\sigma))^{-1}\|_2^2} \leq \frac{M' + \frac{N}{d_\sigma}}{M' - \frac{N}{d_\sigma}}$$

which proves the assertion (6.9). □

Remark 6.8 *Note that the assumption of the preceding theorem is $M' > \frac{N}{d_\sigma}$. This means that the bounds for the condition number can only be achieved if we employ a rectangular Vandermonde matrix $\mathbf{V}_{M',M}$ with additional lines.*

The above observations lead us to the problem to optimize the parameter σ defining the matrix $\mathbf{V}_{M',M}(\sigma)$ with knots $\omega_N^{\sigma n_1},\ldots,\omega_N^{\sigma n_M}$ for given indices $0 \leq n_1 < n_2 < \ldots < n_M < N$. We denote by $\widetilde{\sigma}$ the optimal choice for σ that satisfies

$$d_{\widetilde{\sigma}} := \max_{\sigma\in\{1,\ldots,N\}} d_\sigma$$

with d_σ defined in (6.8). In the following we investigate lower bounds for d_σ.

Theorem 6.9 *Let N be of the form $N = 2^J$, $J \in \mathbb{N}$, and let $d = d_{\widetilde{\sigma}} := \max_{\sigma \in \{1,\ldots,N\}} d_\sigma$ with d_σ defined in (6.8) be the distance obtained for the optimally chosen parameter $\widetilde{\sigma}$. Then we have*

$$\frac{N}{M^2} \leq d \leq \frac{N}{M}. \tag{6.12}$$

Proof. 1. Considering the m indices $0 \leq n_1 < n_2 < \ldots < n_M < N$ and the corresponding knots $\tilde{\tilde{n}}_k = \widetilde{\sigma} n_k \bmod N$, the distance d is obviously maximal if the knots $\tilde{\tilde{n}}_k$ are equidistantly distributed on the interval of length N, i.e., if $d = N/M$. This proves the upper bound $d \leq N/M$.

2. Let us now prove a lower bound for d. We observe that $d \geq 1$ holds for any chosen index set $\{n_j\}_{j=1}^M$. Hence, we only have to prove the assertion for cases where $M^2 < N$.

   Let us first consider all $\binom{M}{2}$ distances $d_{\ell,k} := |n_\ell - n_k|$ for $\ell, k = 1, \ldots, M$ and $\ell < k$. Assume that ν indices n_j are odd and $M - \nu$ indices are even. This yields $\nu(M-\nu)$ odd distances $d_{\ell,k}$ and $\frac{M(M-1)}{2} - \nu(M-\nu)$ even distances. Since

$$\pm(N - \sigma) d_{\ell,k} \bmod N = (\mp \sigma d_{\ell,k}) \bmod N,$$

   we obtain the same sets of distances for σ and $\sigma + N/2$. Hence, we can restrict the range of the parameter σ to $\{1, \ldots, N/2\}$ and assume that $\widetilde{\sigma} \in \{1, \ldots, N/2\}$ holds for the optimal $\widetilde{\sigma}$.

   We now assume to the contrary that $d = d_{\widetilde{\sigma}} < N/M^2$. Thus, $d_\sigma < N/M^2$ for all $\sigma \in \{1, \ldots, N-1\}$, i.e., for each σ there exists a distance $d_{\ell,k}^\sigma$ with $(\pm \sigma d_{\ell,k}^\sigma) \bmod N < N/M^2$. We will show that this assumption leads to a contradiction by proving that this cannot hold for all $\sigma \in \{1, \ldots, N-1\}$. We restrict the choice of σ to odd values in $\{1, \ldots, N/2\}$ and show that the assertion does not even hold for these values.

   For fixed distances $d_{\ell,k}$ we now determine the largest possible number of odd values σ such that $(\pm \sigma d_{\ell,k}) \bmod N < N/M^2$. We distinguish between odd and even distances $d_{\ell,k}$ and consider these two cases.

   Case 1: If the fixed distance $d_{\ell,k}$ is odd, then $\pm \sigma d_{\ell,k} \bmod N$ is again odd, and for two pairwise different odd values $\sigma_1, \sigma_2 \in \{1, \ldots, N/2 - 1\}$ the corresponding values $\pm \sigma_1 d_{\ell,k} \bmod N$ and $\pm \sigma_2 d_{\ell,k} \bmod N$ are different. This holds since $\sigma_1 d_{\ell,k} =$

$\sigma_2 d_{\ell,k} \bmod N$ yields $(\sigma_1 - \sigma_2)d_{\ell,k} = 0 \bmod N$ with the only solution $\sigma_1 = \sigma_2$ and analogously, $\sigma_1 d_{\ell,k} = -\sigma_2 d_{\ell,k} \bmod N$ yields $(\sigma_1 + \sigma_2)d_{\ell,k} = 0 \bmod N$ having no solution σ_1, σ_2.

Observe that there are $\lceil N/(2M^2)+1/2 \rceil - 1$ odd integers in the interval $[0, N/M^2]$. Hence, there exist at most $\lceil N/(2M^2) + 1/2 \rceil - 1$ pairwise different odd values σ in $\{1, \ldots, N/2\}$ such that $(\pm\sigma d_{\ell,k}) \bmod N < N/M^2$.

Since we have $\nu(M - \nu)$ odd distances, there can be at most

$$\nu(M-\nu)\left(\lceil N/(2M^2) + 1/2 \rceil - 1\right)$$

pairwise different odd values σ in $\{1, \ldots, N/2\}$ such that the condition

$$(\pm\sigma d_{\ell,k}) \bmod N < \frac{N}{M^2} \tag{6.13}$$

is satisfied with an odd distance $d_{\ell,k}$. Note that this upper bound can only be achieved if all occurring odd distances $d_{\ell,k}$ are pairwise different.

Case 2: Let $d_{\ell,k}$ be a fixed even distance. Then there exists a positive integer μ such that $d_{\ell,k} = 2^\mu \tilde{d}_{\ell,k}$ and $\tilde{d}_{\ell,k}$ is odd. Thus, the condition $\pm\sigma d_{\ell,k} \bmod N < N/M^2$ can be simplified to

$$\pm\sigma \tilde{d}_{\ell,k} \bmod \frac{N}{2^\mu} < \frac{N}{2^\mu M^2}.$$

Hence, at most $\lceil N/(2^{\mu+1}M^2)+1/2 \rceil - 1$ pairwise different odd values σ in $\{1, \ldots, N/2\}$ exist such that (6.13) is satisfied.

Since we have $\frac{M(M-1)}{2} - \nu(M-\nu)$ even distances, it follows that at most

$$\left(\frac{M(M-1)}{2} - \nu(M-\nu)\right)\left(\lceil N/(4M^2) + 1/2 \rceil - 1\right)$$

odd values σ in $\{1, \ldots, N/2\}$ can exist such that the condition (6.13) is satisfied with an even distance $d_{\ell,k}$. Again, note that this upper bound can be only achieved if all occurring even distances $d_{\ell,k}$ are pairwise different and of the form $d_{\ell,k} = 2\tilde{d}_{\ell,k}$ with some odd $\tilde{d}_{\ell,k}$.

3. We now consider the following cases and add up for each case the maximal number of pairwise different σ such that (6.13) holds for odd or even distances.

(i) For $N > 4M^2$ the number of odd σ satisfying (6.13) for at least one distance $d_{\ell,k}$ is bounded by

$$\begin{aligned}
&\nu(M-\nu)\left(\frac{N}{2M^2}+\frac{1}{2}\right)+\left(\frac{M(M-1)}{2}-\nu(M-\nu)\right)\left(\frac{N}{4M^2}+\frac{1}{2}\right)\\
&\quad=\nu(M-\nu)\frac{N}{4M^2}+\frac{M(M-1)}{2}\left(\frac{N}{4M^2}+\frac{1}{2}\right)\\
&\quad\leq\frac{M^2}{4}\frac{N}{4M^2}+\frac{N}{8}-\frac{N}{8M}+\frac{M^2}{4}-\frac{M}{4}\\
&\quad<\frac{N}{16}+\frac{N}{8}+\frac{N}{16}-\frac{N}{8M}-\frac{M}{4}\\
&\quad<\frac{N}{4}.
\end{aligned}$$

Here, we have used that $\nu(M-\nu)\leq\frac{M^2}{4}$ for all $\nu\in\{0,\dots,M\}$. The inequality shows that not all values σ satisfy the condition (6.13) in the case $N > 4M^2$.

(ii) For $3M^2 < N \leq 4M^2$ we have

$$\left\lceil\frac{N}{2M^2}+\frac{1}{2}\right\rceil-1=2,\qquad\left\lceil\frac{N}{4M^2}+\frac{1}{2}\right\rceil-1=1.$$

Hence, the number of odd values σ satisfying (6.13) is bounded by

$$\begin{aligned}
&2\nu(M-\nu)+\frac{M(M-1)}{2}-\nu(M-\nu)=\nu(M-\nu)+\frac{M(M-1)}{2}\\
&\quad\leq\frac{M^2}{4}+\frac{M^2}{2}-\frac{M}{2}=\frac{3M^2}{4}-\frac{M}{2}\\
&\quad<\frac{N}{4},
\end{aligned}$$

as $3M^2 < N$. Thus, also in this case, not all values σ satisfy (6.13).

(iii) For $2M^2 < N \leq 3M^2$ it holds that

$$\left\lceil\frac{N}{2M^2}+\frac{1}{2}\right\rceil-1=1,\qquad\left\lceil\frac{N}{4M^2}+\frac{1}{2}\right\rceil-1=1.$$

Thus, the number of odd values σ satisfying (6.13) is bounded by

$$\begin{aligned}
&\nu(M-\nu)+\frac{M(M-1)}{2}-\nu(M-\nu)=\frac{M(M-1)}{2}\\
&\quad<\frac{M^2}{2}<\frac{N}{4}.
\end{aligned}$$

Hence, also in the case that $2M^2 < N \leq 3M^2$, not all values σ satisfy (6.13).

(iv) For $M^2 < N \leq 2M^2$ we have

$$\left\lceil \frac{N}{2M^2} + \frac{1}{2} \right\rceil - 1 = 1, \qquad \left\lceil \frac{N}{4M^2} + \frac{1}{2} \right\rceil - 1 = 0.$$

Thus, the number of odd values σ satisfying (6.13) is bounded by

$$\nu(M - \nu) \leq \frac{M^2}{4} < \frac{N}{4},$$

i.e., also in this case, not all values σ satisfy (6.13).

Thus, the number of odd values $\sigma \in$ for which there exists a $d_{\ell,k}^{\sigma}$ such that

$$(\pm \sigma d_{\ell,k}^{\sigma}) \bmod N < \frac{N}{M^2}$$

holds is strictly smaller than $N/4$ in any case. This means that the assumption does not hold for all σ and is therefore wrong. Hence, $d \geq \frac{N}{M^2}$. □

Remark 6.10 *1. The lower bound $d = N/M^2$ can be indeed achieved if $N = 2^J = 2^{\alpha} M^2$ for some $\alpha \in \mathbb{N}_0$ (which assures that d is an integer) and if all distances of the form*

$$d_{\ell,k} = 2^{\alpha}(2r + 1), \qquad r = 0, \ldots, \frac{N}{2^{\alpha+2}} - 1,$$

occur. This follows from the above proof. Choosing e.g. $N = 16$, $M = 4$, $\alpha = 0$, and the four indices $n_1 = 0$, $n_2 = 1$, $n_3 = 3$, $n_4 = 8$, then

$$D := \{d_{\ell,k} \,|\, \ell, k = 1, \ldots, 3;\ \ell < k\} = \{1, 2, 3, 5, 7, 8\}$$

contains all odd numbers in $\{0, \ldots, N/2\}$, and we find $d = N/M^2 = 1$.

2. Note that the case $d = N/M^2$ is actually very rare. It occurs only for very special choices of indices $\{n_k\}_{k=1}^{M}$ (as well as its shifts $\{n_k + \ell\}_{k=1}^{M}$, $\ell = 0, \ldots, N-1$, and shifted reflections $\{(N - n_k) + \ell\}_{k=1}^{M}$, $\ell = 0, \ldots, N-1$). In the above case $N = 16$, $M = 4$, there are $\binom{N}{4} = 1820$ possibilities to fix four (ordered) indices, where $d = N/M^2 = 1$ only occurs in 128 cases.

> *In numerical experiments, one can observe that the number of cases with ill-conditioned matrices (i.e., very small d) decreases very rapidly with increasing vector length N. One reason for this is the fact that we have more possibilities to choose M knots on the unit circle, but we also have more possible choices for σ.*

Although we established some bounds for the condition of the occurring Vandermonde matrices and a result on the distance of knots on the unit circle which are determined by indices of a vectors, it remains to develop an efficient algorithm to compute the optimal parameter σ. In practice, choosing $\sigma = \widetilde{\sigma}$ with

$$d_{\widetilde{\sigma}} := \max_{\sigma \in \{1,\dots,N\}} d_\sigma$$

in Algorithm 6.2 already assures a good conditioned matrix in most cases.

Very small distances between knots and therefore very ill-conditioned Vandermonde matrices rarely occur in practice as we pointed out in the above remark. This means that in many cases, reconstruction using a Vandermonde matrix with acceptable condition is possible.

7. A two-dimensional sparse FFT algorithm

In preceding chapters we discussed sparse FFT algorithms for vectors. However, the discrete Fourier transform is defined for matrices as well and there also exist two-dimensional FFT algorithms. This raises the question whether our approaches can be transferred to matrices and therefore also to discrete images.

7.1. Two-dimensional FFT

Let us first introduce the preliminaries on the two-dimensional discrete Fourier transform. We follow here the notation of [52, Section 4.4].

Definition 7.1 *The* two-dimensional discrete Fourier transform $\widehat{\mathbf{A}} \in \mathbb{C}^{N_1 \times N_2}$ *of a matrix* $\mathbf{A} \in \mathbb{C}^{N_1 \times N_2}$ *is given by*

$$\widehat{\mathbf{A}} := \mathbf{F}_{N_1} \mathbf{A} \, \mathbf{F}_{N_2}$$

where

$$\mathbf{F}_{N_i} = \left(\omega_{N_i}^{k\ell}\right)_{k,\ell=0}^{N_i-1} \in \mathbb{C}^{N_i \times N_i}$$

denotes the discrete Fourier matrix with the N_i*-th root of unity* $\omega_{N_i} := \mathrm{e}^{-\frac{2\pi \mathrm{i}}{N_i}}$ *for* $i = 1, 2$.

Hence, the entries of the Fourier transformed matrix $\widehat{\mathbf{A}} = (\widehat{a}_{k_1 k_2})_{k_1,k_2=0}^{N_1-1,N_2-1} \in \mathbb{C}^{N_1 \times N_2}$ are given by

$$\widehat{a}_{k_1 k_2} = \sum_{j_1=0}^{N_1-1} \sum_{j_2=0}^{N_2-1} a_{j_1 j_2} \, \omega_{N_1}^{j_1 k_1} \, \omega_{N_2}^{j_2 k_2}, \qquad k_r = 0, \ldots, N_r - 1; \; r = 1, 2. \tag{7.1}$$

The two-dimensional inverse discrete Fourier transform is thus defined by

$$\mathbf{A} = \mathbf{F}_{N_1}^{-1} \widehat{\mathbf{A}}\, \mathbf{F}_{N_2}^{-1}$$

where

$$\mathbf{F}_{N_i}^{-1} = \frac{1}{N_i} \left(\omega_{N_i}^{-k\ell}\right)_{k,\ell=0}^{N_i-1} \in \mathbb{C}^{N_i \times N_i}$$

is the inverse Fourier matrix for $i = 1, 2$. Hence, the entries of $\mathbf{A}$ can be written as

$$a_{j_1 j_2} = \frac{1}{N_1 N_2} \sum_{k_1=0}^{N_1-1} \sum_{k_2=0}^{N_2-1} \widehat{a}_{k_1 k_2}\, \omega_{N_1}^{-j_1 k_1}\, \omega_{N_2}^{-j_2 k_2}, \qquad j_r = 0, \ldots, N_r - 1;\ r = 1, 2.$$

We illustrate the Fourier transform of a matrix by a small example. We choose a matrix with very few nonzero entries which will be useful in the following.

Example 7.2 Let $\mathbf{A} \in \mathbb{C}^{4\times 4}$ be given by

$$\mathbf{A} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

Then we obtain the Fourier transform

$$\begin{aligned} \widehat{\mathbf{A}} = \mathbf{F}_4 \mathbf{A} \mathbf{F}_4 &= \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -\mathrm{i} & -1 & \mathrm{i} \\ 1 & -1 & 1 & -1 \\ 1 & \mathrm{i} & -1 & -\mathrm{i} \end{pmatrix} \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -\mathrm{i} & -1 & \mathrm{i} \\ 1 & -1 & 1 & -1 \\ 1 & \mathrm{i} & -1 & -\mathrm{i} \end{pmatrix} \\ &= \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -\mathrm{i} & -1 & \mathrm{i} \\ 1 & -1 & 1 & -1 \\ 1 & \mathrm{i} & -1 & -\mathrm{i} \end{pmatrix} \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & -\mathrm{i} & -1 & \mathrm{i} \\ 1 & -1 & 1 & -1 \\ 0 & 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 2 & -1-\mathrm{i} & 0 & -1+\mathrm{i} \\ -1-\mathrm{i} & 0 & -1+\mathrm{i} & 2 \\ 0 & -1+\mathrm{i} & 2 & -1-\mathrm{i} \\ -1+\mathrm{i} & 2 & -1-\mathrm{i} & 0 \end{pmatrix}. \end{aligned}$$

By usual matrix multiplication, the computational complexity of the two-dimensional DFT is $\mathcal{O}(N_1 N_2 (N_1 + N_2))$, caused by $N_2 N_1 (N_1 - 1) + N_1 N_2 (N_2 - 1) = N_1 N_2 (N_1 + N_2 - 2)$ complex additions and $N_2 N_1^2 + N_1 N_2^2 = N_1 N_2 (N_1 + N_2)$ complex multiplications.

We follow the presentation in [33, Section 4.4] and transfer the ideas for fast algorithms from the one-dimensional discrete Fourier transform to the computation of two-dimensional discrete Fourier transforms in order to improve the complexity. For this purpose, we rewrite the entries of $\widehat{\mathbf{A}}$ given in (7.1) as follows

$$\widehat{a}_{k_1k_2} = \sum_{j_1=0}^{N_1-1} \omega_{N_1}^{j_1k_1} \sum_{j_2=0}^{N_2-1} a_{j_1j_2}\, \omega_{N_2}^{j_2k_2}, \qquad k_r = 0, \ldots, N_r - 1;\ r = 1, 2.$$

We observe that the vectors $\bar{\mathbf{a}}_{k_2} = (\bar{a}_{j_1k_2})_{j_1=0}^{N_1-1}$, $k_2 = 1, \ldots, N_2 - 1$, with components

$$\bar{a}_{j_1k_2} = \sum_{j_2=0}^{N_2-1} a_{j_1j_2}\, \omega_{N_2}^{j_2k_2}, \qquad j_1 = 1, \ldots, M_1 - 1,$$

denote one-dimensional discrete Fourier transforms of the rows in $\mathbf{A}$. Hence, this yields a matrix $\bar{\mathbf{A}} = (\bar{\mathbf{a}}_0, \ldots, \bar{\mathbf{a}}_{N_2-1})$ containing the Fourier transformed rows of $\mathbf{A}$.

Then $\widehat{\mathbf{A}}$ can be obtained by applying another one-dimensional Fourier transform to each of the columns of the new matrix,

$$\widehat{a}_{k_1k_2} = \sum_{j_1=0}^{N_1-1} \omega_{N_1}^{j_1k_1}\, \bar{a}_{j_1k_2}, \qquad k_r = 0, \ldots, N_r - 1;\ r = 1, 2.$$

This procedure for the computation of the components of $\widehat{\mathbf{A}}$ is the so-called *row-column* method and it allows us to reduce the two-dimensional discrete Fourier transform to one-dimensional transforms. In particular, for the DFT of an $(N_1 \times N_2)$-matrix $\mathbf{A}$, we first apply N_1 DFTs of length N_2 to the rows of $\mathbf{A}$ and afterwards N_2 columnwise DFTs of length N_1. This approach is also applicable to discrete Fourier transforms of higher dimensions which are defined in a similar way. In consequence, it is possible to apply algorithms developed for the one-dimensional Fourier transform also to higher dimensional Fourier transforms, see [33] or [52].

As there exist several fast algorithms for the discrete Fourier transform of vectors (see also Section 2.2), the two-dimensional DFT can be computed efficiently. Assuming that $N_1 = 2^{J_1}$ and $N_2 = 2^{J_2}$, $J_1, J_2 \in \mathbb{N}$ and applying *radix-2* algorithms for both row and column transforms, we achieve a computational complexity of $\mathcal{O}(N_1N_2 \log(N_1N_2))$. This can be seen as follows: The N_1 FFTs of length N_2 require $N_1N_2 \log_2 N_2$ complex additions and $N_1 \frac{N_2}{2} \log_2 N_2$ complex multiplications and the N_2 FFTs of length N_1 equivalently require $N_2N_1 \log_2 N_1$ complex additions and $N_2 \frac{N_1}{2} \log_2 N_1$ complex multiplications. Hence,

altogether the computation is done in $N_1 N_2 \log_2 N_2 + N_2 N_1 \log_2 N_1 = N_1 N_2 \log_2(N_1 N_2)$ complex additions and $N_1 \frac{N_2}{2} \log_2 N_2 + N_2 \frac{N_1}{2} \log_2 N_1 = \frac{N_1 N_2}{2} \log_2(N_1 N_2)$ complex multiplications.

7.2. A sparse FFT algorithm for matrices with small support

We aim to apply the proposed algorithms of previous chapters to matrices. Therefore, we first fix what we understand by a matrix with small support. These are matrices which have nonzero entries only within a small (not necessarily quadratic) submatrix. Hence, in the easiest case, the support of a matrix $\mathbf{A} \in \mathbb{C}^{N_1 \times N_2}$ can be sketched as follows

$$\mathbf{A} = \left(\begin{array}{ccc} & & \\ & \blacksquare & \\ & & \end{array} \right), \tag{7.2}$$

where the grey submatrix of dimension $m_1 \times m_2$ symbolizes the support. We give a definition for the support size of a matrix which is equivalent to the support length of a vector (cf. Definition 3.1).

Definition 7.3 *Let* $\mathbf{A} = (a_{j_1 j_2})_{j_1, j_2 = 0}^{N_1 - 1, N_2 - 1} \in \mathbb{C}^{N_1 \times N_2}$. *The* support size $(m_1 \times m_2) = |\operatorname{supp} \mathbf{A}|$ *of* $\mathbf{A}$ *is defined as the pair of minimal positive integers* (m_1, m_2) *for which there exists an index* $(\mu_1, \mu_2) \in \{0, \ldots, N_1 - 1\} \times \{0, \ldots, N_2 - 1\}$ *such that* $a_{j_1 j_2} = 0$ *for all* $(j_1, j_2) \notin I = \{((\mu_1 + r_1) \bmod N_1, (\mu_2 + r_2) \bmod N_2) \mid r_1 = 0, \ldots, m_1 - 1,\ r_2 = 0, \ldots, m_2 - 1\}$. *We call the index set* I *the* support set *of* $\mathbf{A}$ *and* (μ_1, μ_2) *the* first support index *of* $\mathbf{A}$.

Note that similarly to the definition of the support of a vector in Chapter 3, the support of matrices is to be understood modulo the number of rows N_1 resp. the number of columns N_2, i.e., it can for instance begin in the last row of a matrix and continue in its first row. In particular, the index of the leftmost uppermost nonzero component is not necessarily the first support index (as it is defined above).

Example 7.4 1. Let $\mathbf{A} \in \mathbb{C}^{4\times 4}$ be the matrix in Example 7.2 given by

$$\mathbf{A} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

Then $\mathbf{A}$ has a support size of 2×2.

2. Choose another matrix $\mathbf{A} \in \mathbb{C}^{4\times 4}$ which is of the form

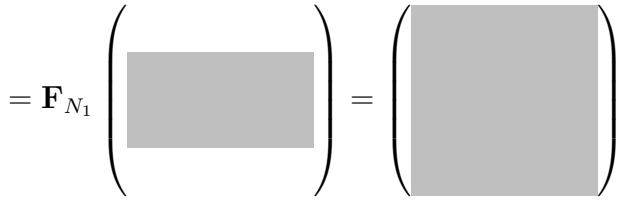
$$\mathbf{A} = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}.$$

Then $\mathbf{A}$ also has a support size of 2×2. This example shows that the support of $\mathbf{A}$ lies not necessarily in the middle of the matrix and may consist of multiple parts. The first support index of $\mathbf{A}$ is $(\mu_1, \mu_2) = (3, 3)$.

In the following, let us consider matrices of size $N_1 \times N_2$ with $N_1 = 2^{J_1}$ and $N_2 = 2^{J_2}$ for $N_1, N_2 \in \mathbb{N}$. We now want to extend Algorithm 4.12 to the recovery of matrices $\mathbf{A} \in \mathbb{C}^{N_1 \times N_2}$ with small support of size $m_1 \times m_2$ from (noisy) Fourier data $\widehat{\mathbf{A}}$. Recall that Algorithm 4.12 is applicable to the reconstruction of vectors with small support from noisy Fourier data. Hence, we have to ensure that all inverse Fourier transforms of row or column vectors, which are necessary for the reconstruction of $\mathbf{A}$, result in vectors with small support.

Let us have a closer look at the two steps of the two-dimensional DFT, first row transforms and afterwards column transforms. We illustrate the support sets of all occurring matrices assuming that the support of $\mathbf{A}$ can be sketched as in (7.2).

$$\widehat{\mathbf{A}} = \mathbf{F}_{N_1} \mathbf{A} \mathbf{F}_{N_2} = \mathbf{F}_{N_1} \begin{pmatrix} \\ \\ \end{pmatrix} \mathbf{F}_{N_2}$$



Observe that the support of the intermediate matrix $\bar{\mathbf{A}}$ after the first transform is of size $m_1 \times N_2$. After the second transform, the support of $\widehat{\mathbf{A}}$ will in general be of size $N_1 \times N_2$, as indicated in the above illustration. This matrix structure can be nicely recognized in Example 7.2.

In the course of reconstruction of $\mathbf{A}$ from its Fourier transform $\widehat{\mathbf{A}}$, we first compute the inverse Fourier transform of all columns of $\widehat{\mathbf{A}}$. We observe that the column vectors in the resulting matrix all have short support of (maximal) length m_1. Hence, Algorithm 4.12 can be applied to each of these transforms. In the second reconstruction step, the inverse Fourier transforms of the rows have to be computed. Note that we have $N_1 - m_1$ rows containing zero vectors for which no computations at all are needed. The inverse transforms of the remaining m_1 rows can again be computed efficiently using Algorithm 4.12 as the resulting rows in $\mathbf{A}$ have small support of maximal length m_2.

Remark 7.5 *Note that Algorithm 5.4 for the reconstruction of vectors with real nonnegative entries cannot be transferred to matrices in the same way. That is because we cannot ensure that the intermediate matrix, which results from the first reconstruction step, only contains real nonnegative entries. Indeed, this will never be the case in practice when we work with noisy data.*

We summarize the reconstruction of a matrix $\mathbf{A} \in \mathbb{C}^{N_1 \times N_2}$ from Fourier data in Algorithm 7.6. For the efficient reconstruction of vectors with short support, we use Algorithm 4.12. The only additional a priori information that is required by the algorithm is the support size of the matrix $\mathbf{A}$ or an upper bound for it (i.e., the size of a larger submatrix which contains the support).

Algorithm 7.6 (Sparse FFT for matrices with small support)
Input: Noisy measurement matrix $\widehat{\mathbf{A}} = (\widehat{A}_{k_1 k_2})_{k_1,k_2=0}^{N_1-1,N_2-1} \in \mathbb{C}^{N_1 \times N_2}$ where $N_1 = 2^{J_1}$ and $N_2 = 2^{J_2}$; $|\operatorname{supp} \mathbf{A}| \leq (m_1 \times m_2)$ with $m_1 < N_1$, $m_2 < N_2$.

1. For $k_2 = 0, \ldots, N_2 - 1$
   Compute the inverse Fourier transform $\widetilde{\mathbf{a}}_{k_2}$ of the vector $(\widehat{\mathbf{a}}_{k_1 k_2})_{k_1=0}^{N_1-1}$ with support

length m_1 using Algorithm 4.12.
End (for).

2. Set $\widetilde{\mathbf{A}} = (\widetilde{\mathbf{a}}_0, \ldots, \widetilde{\mathbf{a}}_{N_2-1})$.

3. For $k_1 = 0, \ldots, N_1 - 1$
Compute the inverse Fourier transform $\mathbf{a}_{k_1}$ of the vector $(\widetilde{\mathbf{a}}_{k_1 j_2})_{j_2=0}^{N_2-1}$ with support length m_2 using Algorithm 4.12.
End (for).

4. Set $\mathbf{A} = (\mathbf{a}_0, \ldots, \mathbf{a}_{N_1-1})^T$.

Output: $\mathbf{A} \in \mathbb{C}^{N_1 \times N_2}$.

We include an exemplary MATLAB implementation in Section A.3.

From Section 4.3.4 we know that the computational complexity of the numerically stable Algorithm 4.12 for a vector of length N with support length m is $\mathcal{O}(m \log N)$. For the two-dimensional reconstruction of a matrix $\mathbf{A} \in \mathbb{C}^{N_1 \times N_2}$ with support size $m_1 \times m_2$, in the first step N_2 applications of the algorithm to vectors of length N_1 with support length m_1 are needed, hence $\mathcal{O}(N_2 m_1 \log N_1)$ arithmetical operations. The second step requires m_1 applications of Algorithm 4.12 to vectors of length N_2 with support length m_2 causing $\mathcal{O}(m_1 m_2 \log N_2)$ operations.

This results in a computational complexity of $\mathcal{O}(m_1(N_2 \log N_1 + m_2 \log N_2))$ for the reconstruction of a matrix $\mathbf{A} \in \mathbb{C}^{N_1 \times N_2}$ with support size $m_1 \times m_2$ from Fourier data $\widehat{\mathbf{A}}$.

Remark 7.7 *For exact input data, we can even achieve a lower complexity. In this case, Algorithm 4.8 can be applied which has a complexity of $\mathcal{O}(m \log m)$ for the reconstruction of a vector with support length m from Fourier data. For the application to matrices, this yields a computational complexity of $\mathcal{O}(m_1(N_2 \log m_1 + m_2 \log m_2))$, caused by N_2 column transforms each requiring $\mathcal{O}(m_1 \log m_1)$ operations and m_1 row transforms of complexity $\mathcal{O}(m_2 \log m_2)$.*

In the exact case, the number of Fourier values used for each column transform is $\mathcal{O}(m_1)$ and $\mathcal{O}(m_2)$ for the row transforms. Hence, $\mathcal{O}(N_2 m_1 + m_1 m_2)$ values are needed for the complete reconstruction. The number of required Fourier values for reconstruction from noisy data depends, as in the one-dimensional case, on the number of additional values used for stabilization.

7.3. Numerical results

In this section we analyze the numerical performance of the two-dimensional sparse FFT Algorithm 7.6. We apply the algorithm to recover a matrix $\mathbf{A} \in \mathbb{C}^{N_1 \times N_2}$ from perturbed Fourier data $\widehat{\mathbf{B}} = (\widehat{b}_{j_1 j_2})_{j_1 j_2 = 0}^{N_1 - 1, N_2 - 1} \in \mathbb{C}^{N_1 \times N_2}$ with components

$$\widehat{b}_{j_1 j_2} = \widehat{a}_{j_1 j_2} + \varepsilon_{j_1 j_2}, \qquad j_1 = 0, \ldots, N_1, \quad j_2 = 0, \ldots, N_2,$$

where $\mathbf{E} = (\varepsilon_{j_1 j_2})_{j_1 j_2 = 0}^{N_1 - 1, N_2 - 1} \in \mathbb{C}^{N_1 \times N_2}$ denotes a noise matrix.

In order to measure the intensity of the noise $\mathbf{E}$, we introduce an SNR value which is similar to the SNR of vectors.

Definition 7.8 *The signal-to-noise-ratio (SNR) for the noisy input data $\widehat{\mathbf{B}} \in \mathbb{C}^{N_1 \times N_2}$ is given by*

$$\begin{aligned} \mathrm{SNR} &= 20 \cdot \log_{10} \frac{\left\| \widehat{\mathbf{A}} \right\|_F}{\|\mathbf{E}\|_F} = 20 \cdot \log_{10} \frac{\left\| \widehat{\mathbf{A}} \right\|_F}{\left\| \widehat{\mathbf{B}} - \widehat{\mathbf{A}} \right\|_F} \\ &= 20 \cdot \log_{10} \frac{\sqrt{\sum_{j_2=0}^{N_2-1} \sum_{j_1=0}^{N_1-1} |\widehat{a}_{j_1 j_2}|^2}}{\sqrt{\sum_{j_2=0}^{N_2-1} \sum_{j_1=0}^{N_1-1} \left| \widehat{b}_{j_1 j_2} - \widehat{a}_{j_1 j_2} \right|^2}}. \end{aligned}$$

The error of the reconstructed matrix $\mathbf{A}'$ will be measured by the Frobenius norm

$$\frac{\|\mathbf{A} - \mathbf{A}'\|_F}{N_1 N_2}.$$

We compare the reconstructions by the proposed algorithm to the result of an inverse Fourier transform directly applied to the noisy data. The error of the reconstruction by regular inverse FFT is given by

$$\frac{\|\mathbf{A} - \mathbf{F}_{N_1}^{-1} \widehat{\mathbf{B}} \mathbf{F}_{N_2}^{-1}\|_F}{N_1 N_2}.$$

Additionally, the errors can be quantified by the SNR values

$$\mathrm{SNR}_{\mathrm{alg}} = 20 \cdot \log_{10} \frac{\|\mathbf{A}\|_F}{\|\mathbf{A} - \mathbf{A}'\|_F} \qquad \text{resp.} \qquad \mathrm{SNR}_{\mathrm{IFFT}} = 20 \cdot \log_{10} \frac{\|\mathbf{A}\|_F}{\|\mathbf{A} - \mathbf{F}_{N_1}^{-1} \widehat{\mathbf{B}} \mathbf{F}_{N_2}^{-1}\|_F}.$$

Let us begin with a small example. We consider the matrix $\mathbf{A} = (a_{j_1 j_2})_{j_1, j_2 = 0}^{15} \in \mathbb{C}^{16 \times 16}$

with nonzero components $a_{21} = 8$, $a_{22} = -3$, $a_{32} = -5$, $a_{33} = 2$, $a_{41} = -1$ and $a_{43} = 4$. Hence, all nonzero entries of $\mathbf{A}$ are contained in a submatrix of size 3×3, i.e., the support size of $\mathbf{A}$ is $m_1 \times m_2 = 3 \times 3$.

In this example, the Fourier data $\widehat{\mathbf{A}}$ is perturbed by a noise matrix $\mathbf{E}$ with uniformly generated noise of SNR = 20. The noise matrix has the largest entry $\max_{j_1,j_2} |\varepsilon_{j_1 j_2}| = 1.817$ and the mean modulus of its components is $\sum_{j_1,j_2} |\varepsilon_{j_1 j_2}|/N_1 N_2 = 1.011$.

The nonzero components of the matrix $\mathbf{A}'$, which was reconstructed by the proposed algorithm, are

$$\begin{aligned}
&a_{21} = 8.001 + 0.052\mathrm{i}, \quad a_{22} = -3.045 - 0.076\mathrm{i}, \quad a_{23} = 0.050 + 0.029\mathrm{i},\\
&a_{31} = 0.042 + 0.062\mathrm{i}, \quad a_{32} = -5.053 - 0.045\mathrm{i}, \quad a_{33} = 1.99 - 0.042\mathrm{i},\\
&a_{41} = -1.069 - 0.010\mathrm{i}, \quad a_{42} = -0.004 - 0.004\mathrm{i}, \quad \text{and } a_{43} = 3.933 - 0.007\mathrm{i}.
\end{aligned}$$

In contrast, there are no zero components in the reconstruction $\mathbf{F}_{N_1}^{-1} \widehat{\mathbf{B}} \mathbf{F}_{N_2}^{-1}$ by the inverse FFT algorithm. However, the components outside the support are very small.

The results yield reconstruction errors of $\|\mathbf{A} - \mathbf{A}'\|_F/256 = 7.360 \cdot 10^{-4}$ for our algorithm resp. $\|\mathbf{A} - \mathbf{F}_{N_1}^{-1} \widehat{\mathbf{B}} \mathbf{F}_{N_2}^{-1}\|_F/256 = 4.261 \cdot 10^{-3}$ for an inverse FFT algorithm. The SNR values of the reconstructed matrices are $\mathrm{SNR}_{\mathrm{alg}} = 35.253$ and $\mathrm{SNR}_{\mathrm{IFFT}} = 20$.

The next example illustrates the application of the algorithm to an image with small support. We choose the "cameraman" image [6] and consider a modified image with small support of size 50×60 (i.e., we set all components to zero which are outside a small support set). Then the Fourier transform of the image is perturbed by either uniformly or normally distributed noise of SNR = 20 and we reconstruct the image from the noisy Fourier data using our algorithm resp. an inverse FFT algorithm. The results can be found in Figure 7.1 where we illustrate the real parts of the reconstructions.

There are no obvious differences which can be recognized in the images. However, the obtained errors for the reconstruction methods are different. In particular, our algorithm recognizes the support and sets all entries outside the support to zero. The errors are given in detail in Table 7.1.

Remark 7.9 *Similarly to the one-dimensional case (cf. Remarks 4.15 and 4.16), we observe that for the reconstruction by an inverse FFT algorithm, the reconstruction error resp. SNR_{IFFT} does not depend on the chosen instance of the noise matrix* $\mathbf{E}$*, but only on its SNR value. This holds because the Frobenius norm is invariant under unitary transforms, similarly to the ℓ_2-norm in the vector case. Hence, the computations in Remarks 4.15 and 4.16 can be done analogously for the matrix case.*

	uniformly distributed noise	normally distributed noise
$\|\mathbf{A} - \mathbf{A}'\|_F/256^2$	0.00213	0.00209
$\|\mathbf{A} - \mathbf{F}_{N_1}^{-1}\widehat{\mathbf{B}}\mathbf{F}_{N_2}^{-1}\|_F/256^2$	0.00973	0.00973
$\mathrm{SNR}_{\mathrm{alg}}$	33.20	33.36
$\mathrm{SNR}_{\mathrm{IFFT}}$	20	20

Table 7.1.: Errors for reconstruction of "cameraman" image with small support.

As a last experiment, we want to investigate the behavior of the algorithm for larger matrices. To do so, we choose the following setting. Let $N = N_1 = N_2 = 2^{10} = 1024$. We consider matrices $\mathbf{A} \in \mathbb{C}^{1024\times 1024}$ with support size $m_1 \times m_2 = 10\times 10$ and complex entries such that $|\mathrm{Re}(a_{j_1 j_2})| \leq 10$, $|\mathrm{Im}(a_{j_1 j_2})| \leq 10$ holds for all (j_1, j_2) within the support. For the numerical experiments the Fourier data is perturbed either by uniformly or normally distributed noise. At each noise level between SNR = 0 and SNR = 50, we evaluate 100 randomly generated matrices $\mathbf{A}$. These are reconstructed from noisy Fourier data by the proposed algorithm and by an inverse FFT algorithm.

In order to give an idea of the applied noise level, we list the component

$$\max_{j_1,j_2\in\{0,\ldots,N-1\}} |\varepsilon_{j_1 j_2}|$$

of maximal modulus in $\mathbf{E}$ as well as the average modulus

$$\sum_{j_1,j_2=0}^{N-1} |\varepsilon_{j_1 j_2}|/N^2$$

of the components of $\mathbf{E}$ for all noise levels and different kinds of noise in Table 7.2.

The reconstruction errors for uniformly distributed noise resp. for normally distributed noise are illustrated in Figure 7.2 resp. Figure 7.3.

To summarize our findings, we can state that the deterministic algorithm for reconstruction of matrices with small support is stable and works equally well for uniformly and normally distributed noise. Moreover, the results show that our algorithm returns good results for high noise levels whereas it always has a smaller complexity than the regular inverse FFT algorithm.

SNR	uniformly distributed noise		normally distributed noise	
	$\max_{j_1,j_2} \lvert\varepsilon_{j_1 j_2}\rvert$	$\sum_{j_1,j_2} \lvert\varepsilon_{j_1 j_2}\rvert/N^2$	$\max_{j_1,j_2} \lvert\varepsilon_{j_1 j_2}\rvert$	$\sum_{j_1,j_2} \lvert\varepsilon_{j_1 j_2}\rvert/N^2$
0	141.899	76.821	310.068	72.236
5	79.928	43.274	173.118	40.488
10	44.874	24.295	99.280	23.018
15	25.148	13.615	55.457	12.903
20	14.079	7.622	31.005	7.231
25	7.920	4.288	17.475	4.064
30	4.485	2.428	9.818	2.293
35	2.517	1.363	5.455	1.278
40	1.419	0.768	3.086	0.721
45	0.792	0.429	1.745	0.406
50	0.449	0.243	0.983	0.229

Table 7.2.: Component of maximal modulus in $\mathbf{E}$ and the average modulus of the components of $\mathbf{E}$ for all noise levels and different kinds of noise.

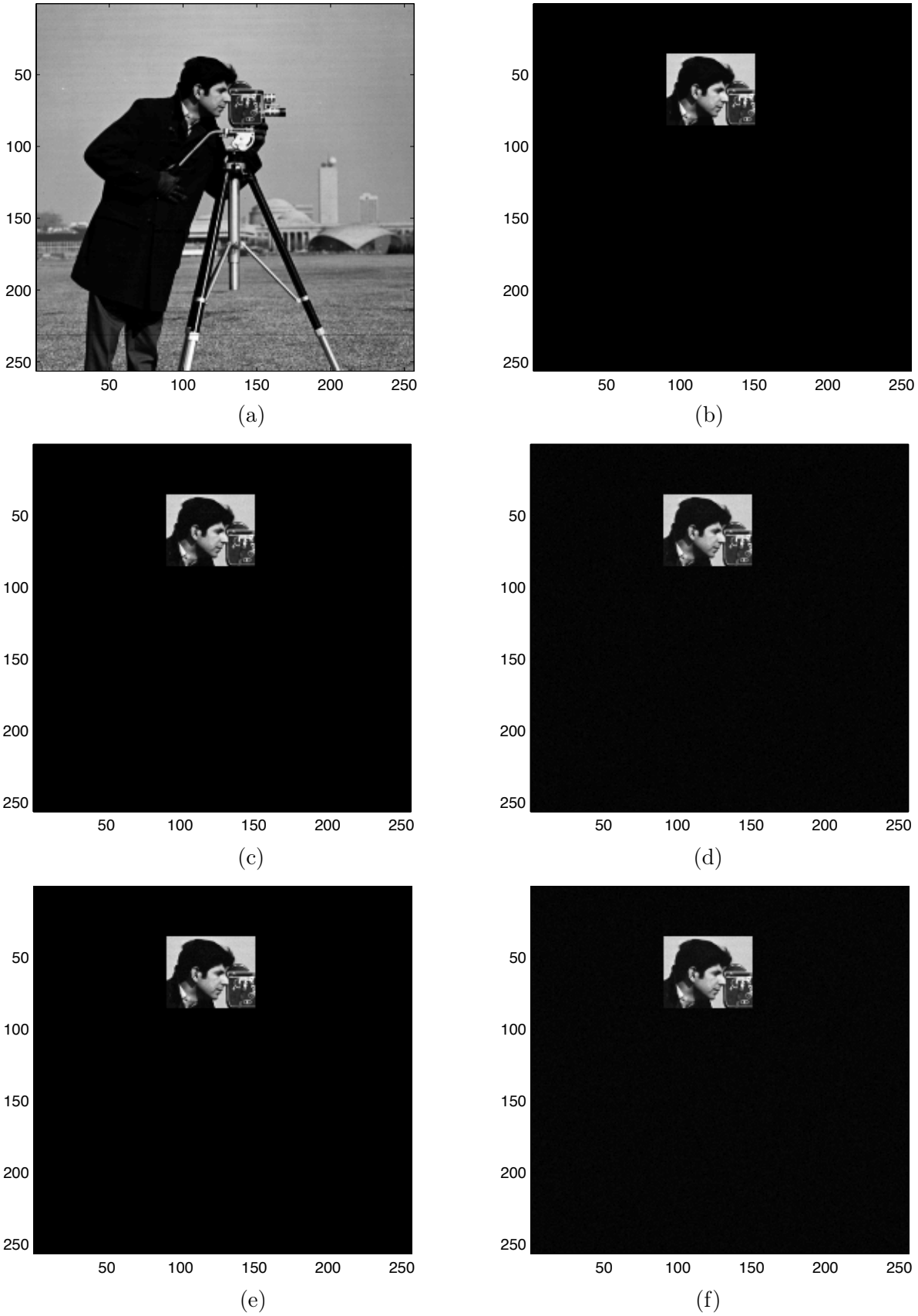
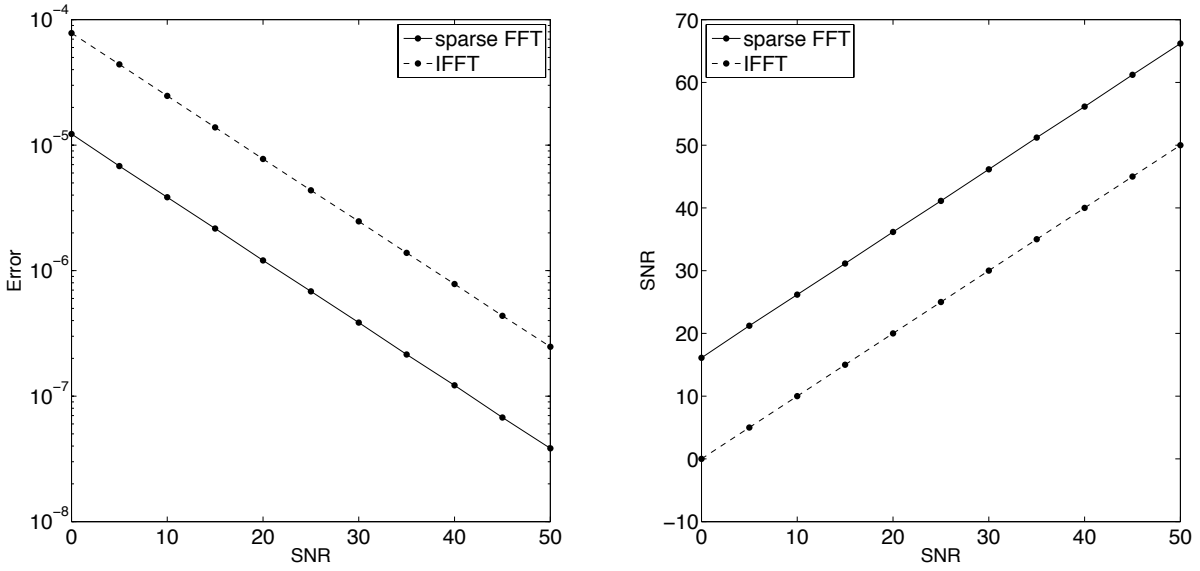
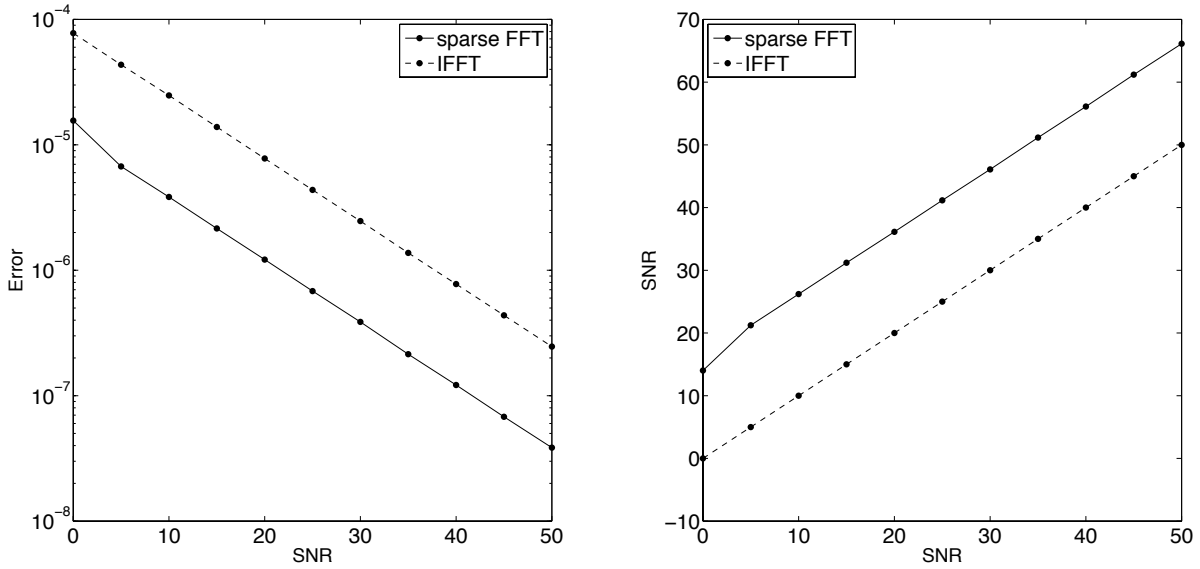


Figure 7.1.: (a) Original image $\mathbf{A} \in \mathbb{R}^{256\times256}$; (b) Image with small support of size 50×60; (c) Reconstruction by Algorithm 7.6 for uniform noise; (d) Reconstruction by IFFT for uniform noise; (e) Reconstruction by Algorithm 7.6 for normally distributed noise; (f) Reconstruction by IFFT for normally distributed noise.



(a) (b)

Figure 7.2.: Uniformly distributed noise, $N_1 = N_2 = 2^{10}$, $m_1 \times m_2 = 10 \times 10$: (a) Average reconstruction error $\|\mathbf{A} - \mathbf{A}'\|_2/1024^2$ and (b) average $\mathrm{SNR}_{\mathrm{alg}}$ resp. $\mathrm{SNR}_{\mathrm{IFFT}}$ for different noise levels, comparing the sparse FFT Algorithm 7.6 and regular inverse FFT.



(a) (b)

Figure 7.3.: Normally distributed noise, $N_1 = N_2 = 2^{10}$, $m_1 \times m_2 = 10 \times 10$: (a) Average reconstruction error $\|\mathbf{A} - \mathbf{A}'\|_2/1024^2$ and (b) average $\mathrm{SNR}_{\mathrm{alg}}$ resp. $\mathrm{SNR}_{\mathrm{IFFT}}$ for different noise levels, comparing the sparse FFT Algorithm 7.6 and regular inverse FFT.

8. Conclusion

This thesis focusses on the reconstruction of vectors or matrices that have small support or that are sparse from Fourier data. For these types of objects, we developed new deterministic sparse FFT algorithms for efficient reconstruction.

After presenting an overview on fast Fourier transform algorithms and the reconstruction from Fourier data in Chapter 2 and introducing the concepts of a small support and vector periodization in Chapter 3, we developed algorithms for different types of a priori assumptions in detail.

Chapter 4 was devoted to a new sparse FFT algorithm for vectors $\mathbf{x} \in \mathbb{C}^N$, $N = 2^J$ for $J \in \mathbb{N}$, with small support. This algorithm requires the knowledge of the support length (or an upper bound) m of $\mathbf{x}$ and $\mathcal{O}(m)$ Fourier values $\widehat{x}_k$ (in the exact case; for noisy data, additional values might be chosen for stabilization) from the set of entries of $\widehat{\mathbf{x}} \in \mathbb{C}^N$, where the components $\widehat{x}_k$ are adaptively chosen in course of the reconstruction. We developed an iterative procedure which first computes an inverse FFT of length 2^{L+1} with $2^L \geq m$ in order to obtain the periodization $\mathbf{x}^{(L+1)}$. This vector $\mathbf{x}^{(L+1)}$ is also a vector with short support length m and additionally all nonzero components of the complete vector $\mathbf{x}$ are already contained in $\mathbf{x}^{(L+1)}$ such that we can reconstruct $\mathbf{x}$ from $\mathbf{x}^{(L+1)}$ step-by-step using $J - (L+1)$ additional Fourier values. In case of exact data, we can even directly reconstruct $\mathbf{x}$ from $\mathbf{x}^{(L+1)}$ using only one additional Fourier value. The computational complexity of the algorithm for exact data is $\mathcal{O}(m \log m)$. For noisy data, we stabilized the algorithm which then leads to a complexity of $\mathcal{O}(m \log N)$.

In numerical experiments, we applied uniformly and normally distributed noise and considered vectors with very short support as well as vectors with comparatively large support. Our newly developed algorithm turned out to be stable for very different settings and yields better results than an inverse Fourier transform directly applied to the noisy data.

The aforementioned algorithm can also be transferred to the two-dimensional case. In Chapter 7, we first showed how the two-dimensional DFT can be realized efficiently by row- resp. columnwise application of one-dimensional FFTs such that the new algo-

rithm developed in Chapter 4 can be used for the reconstruction of matrices with small support. We implemented the matrix reconstruction algorithm and obtained very good reconstruction results for different types of noise. Moreover, the algorithm is stable and outperforms conventional inverse FFT algorithms, similar to the one-dimensional case.

In Chapter 5, we introduced a different new deterministic sparse FFT algorithm. In this case, we restricted ourselves to the reconstruction of real nonnegative vectors $\mathbf{x} \in \mathbb{R}_+^N$. The iterative reconstruction principle, which uses periodizations, is similar to the aforementioned algorithm but we have weaker a priori conditions here. The algorithm works for all real nonnegative vectors but it automatically detects a small support and then employs a more efficient reconstruction method. Hence, we only require the knowledge of the Fourier values but no bound for the support length.

The principle of this algorithm is again an iterative procedure recovering periodizations step-by-step from $\mathbf{x}^{(0)}$ to $\mathbf{x}^{(J)} = \mathbf{x}$. Assuming a support length of m, the complexity of the algorithm is $\mathcal{O}(m \log m \log(N/m))$. In numerical experiments with noisy data, we have shown that it returns better results than the regular inverse FFT, for uniformly distributed noise as well as for normally distributed noise whereas it merely uses $\mathcal{O}(m \log(N/m))$ Fourier values. The algorithm was stable in all settings, the only crucial point is to choose the threshold parameter T in a suitable way such that it balances complexity and accuracy of the algorithm.

The approach was further generalized in Chapter 6 which covers a generalization of the algorithm for the reconstruction of real nonnegative M-sparse vectors $\mathbf{x} \in \mathbb{R}_+^N$ as in contrast to Chapter 5, we do not assume the components of $\mathbf{x}$ to be clustered. The iterative algorithm includes the solution of a linear system with a matrix that depends on the distribution of the nonzero entries of the respective periodization at each reconstruction step. These matrices have to be chosen in a way, that they are well-conditioned in order to assure stable reconstruction.

We simplified the problem using Vandermonde matrices with knots on the unit circle that depend on the indices of nonzero entries which have already been found and on one further parameter σ. This σ can be chosen suitably to improve the condition number of the Vandermonde matrices. For this setting, bounds on the condition of the occurring rectangular Vandermonde matrices have been proven. Further, we showed bounds on the minimal distance of knots on the unit circle, which highly influences the condition of these matrices.

Although this is a very promising approach, an efficient algorithm for the determination of the parameter σ is yet to be developed. Moreover, Vandermonde matrices on the

unit circle are a field of research on their own, where new results on their condition can still be obtained. It is an open problem if there are other possibilities of choosing the reconstruction matrices or even different simplifications than Vandermonde matrices.

The two algorithms for real nonnegative vectors presented in Chapter 5 and 6 might also be generalized for the two-dimensional case to some extent, although they solely could be applied for the second reconstruction step. Only here, the recovered vectors are real and nonnegative (see also Remark 7.5). For the first step, a conventional inverse FFT algorithm has to be used.

Furthermore, it could be investigated if the reconstruction concept introduced for real nonnegative vectors can be generalized to complex vectors where the knowledge of the energy of the Fourier transform could be exploited in order to decide in course of the reconstruction whether components of $\mathbf{x}$ are still "hidden" in zeros.

As a conclusion, our algorithms can be used for applications in different fields. The sparse FFT algorithms are applicable for all purposes where vectors or images with small support occur and where the support size might be estimated in advance. In practice, deterministic algorithms have the advantage that they succeed in any case, which is of particular importance for applications that are sensitive to failure.

A. Exemplary implementations of deterministic sparse FFT algorithms

A.1. Implementation of Algorithm 4.12 for vectors with small support

A.1.1. Algorithm

We show an example for a fast MATLAB implementation of Algorithm 4.12 for vectors with small support. The function `xrec = detSmallSuppSparseFFT(data,m)` returns the inverse Fourier Transform `xrec` for given Fourier data `data` with noise. The reconstruction is assumed to have small support where `m` is an upper bound.

```
function xrec=detSmallSuppSparseFFT(data,m)

% compute N, L and J
N=length(data);
J=log2(N);
L=ceil(log2(m));

if L==J || L==J-1
    % inverse FFT
    disp('Support to large! Using standard ifft.')
    xrec=ifft(data);
else
    % calculate absolute value of Fourier data (for faster maximum
    % calculation later on)
    absdata = abs(data);

    % compute x^{(L+1)} by inverse FFT
    xa=ifft(data((2^(J-L-1)*(0:2^(L+1)-1))+1));

    % compute energies in xa
    b=zeros(2^(L+1),1);
```

```
b(1)=sum(abs(xa(1:m)).^2);
for k=2:2^(L+1)
    b(k)=b(k-1)+abs(xa(mod(m+k-2,2^(L+1))+1))^2-abs(xa(k-1))^2;
end

% compute argmax e_k
muvector=zeros(2,1);
muvector(1)=find(b==max(b),1)-1;
% "shifted" Fourier data
yL2=zeros(2^(L+1),1);
yL2(1:2^(L+1))=data((2^(J-L-2)*(2*(1:2^(L+1))-1))+1);
z2=ifft(yL2);
% matrix with xa, z2 as columns
zvectors=[xa,z2];
% compute energies in z2
c=zeros(2^(L+1),1);
c(1)=sum(abs(z2(1:m)).^2);
for k=2:2^(L+1)
    c(k)=c(k-1)+abs(z2(mod(m+k-2,2^(L+1))+1))^2-abs(z2(k-1))^2;
end
c=b+c;
% compute \mu_1^{(L+1)}
muvector(2)=find(c==max(c),1)-1;

t=1;
while muvector(t) ~= muvector(t+1)
    yLk=zeros(2^(L+1),1);
    yLk(1:2^(L+1))=data(mod(2^(J-L-1)*(0:2^(L+1)-1)+t,N)+1);
    zk=ifft(yLk);
    zvectors(:,t+2)=zk;
    d=zeros(2^(L+1),1);
    d(1)=sum(abs(zk(1:m)).^2);
    for k=2:2^(L+1)
        d(k)=d(k-1)+abs(zk(mod(m+k-2,2^(L+1))+1))^2-abs(zk(k-1))^2;
    end
    c=c+d;
    muvector(t+2)=find(c==max(c),1)-1;
    t=t+1;
end

mu=muvector(end);
muL=mu;

% vector xs is the support of x
xs=xa(mod(mu+(0:m-1),2^(L+1))+1);
for j=L+1:J-1
    % find maximal (odd-indexed) component of \hat{x}^{(j+1)}
    ind = (2^(J-j-1)+1):(2^(J-j)):(2^J-2^(J-j-1)+1);
    [~,s]=max(absdata(ind));
    xhatmax=data(ind(s));
    % compute a
```

```
        a=exp(-pi*1i*(2*s-1)*(mu+(0:m-1))/2^j)*xs;
        if abs(a-xhatmax)>abs(a+xhatmax)
            % if Fourier value is closer to -a, then shift mu
            mu=mu+2^j;
        end
    end

    % improve values of support vector using additional vectors from step 2
    zvectors=[zvectors;zvectors];
    zvectors=zvectors((muL+1):(muL+m),:);
    omegamatrix=zeros(m,t+1);
    omegamatrix(:,1)=ones;
    omegamatrix(:,2)=exp(2*pi*1i/2^(L+2)*(mu+(0:m-1)));
    for k=3:t+1
        omegamatrix(:,k)=exp(2*pi*1i/N*(k-2)*(mu+(0:m-1)));
    end
    zvectors=zvectors.*omegamatrix;
    % improved support vector
    xs=sum(zvectors,2)/(t+1);

    % build reconstruction xres of x
    xrec=zeros(N,1);
    xrec(mod(mu+(0:m-1),2^J)+1)=xs(1:m);
end
end
```

A.1.2. Reconstruction of deterministic sampling vectors

The following MATLAB code generates a deterministic sampling vector and applies Algorithm 4.12 to its perturbed Fourier transform.

```
% set parameters
N = 2^10;          % vector length (must be power of 2)
m = 6;             % support length
                   % (should be at most N/4, otherwise use ifft)

% set vector entries
x=zeros(N,1);
x(1)=1;
x(2)=2;
x(1021)=3;
x(1023)=4;
x(1024)=5;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% create noisy Fourier data

SNR = 20;            % set desired SNR
```

```
xhat=fft(x);          % Fourier transform

noise=rand(N,1)-0.5+1i*(rand(N,1)-0.5);          % create random uniform noise
noise=norm(xhat)*10^(-SNR/20)/norm(noise)*noise; % scale according to SNR
xhatnoise=xhat+noise;                            % noisy input data

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% apply algorithm to noisy data

xrec=detSmallSuppSparseFFT(xhatnoise,m);
```

A.1.3. Reconstruction of random sampling vectors

The following MATLAB code generates a random sampling vector and applies Algorithm 4.12 to its perturbed Fourier transform.

```
% set parameters
N = 2^20;             % vector length (must be power of 2)
m = 20;               % support length
                      % (should be at most N/4, otherwise use ifft)

% calculate random sampling vector
x=zeros(N,1);                          % allocate space
L=ceil(log2(m));                       % calculate L
firstsuppind=randi(N);                 % random first support index
                                       % calculate support
supp=[firstsuppind:min(firstsuppind+m-1,N),1:firstsuppind+m-1-N];
suppzeros=randi(m-2);                  % random number of zeros inside support
supp(randperm(m-2,suppzeros)+1)=[];    % delete number of suppzeros support
                                       % indices (not first or last)
x(supp)=20*(rand(m-suppzeros,1)-0.5+1i*(rand(m-suppzeros,1)-0.5));
                                       % set random complex values


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% create noisy Fourier data

SNR = 20;             % set desired SNR

xhat=fft(x);          % Fourier transform

noise=rand(N,1)-0.5+1i*(rand(N,1)-0.5);          % create random uniform noise
noise=norm(xhat)*10^(-SNR/20)/norm(noise)*noise; % scale according to SNR
xhatnoise=xhat+noise;                            % noisy input data

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% apply algorithm to noisy data

xrec=detSmallSuppSparseFFT(xhatnoise,m);
```

A.2. Implementation of Algorithm 5.4 for real nonnegative vectors

A.2.1. Algorithm

In this section, we give an example for a MATLAB implementation of Algorithm 5.4 for real nonnegative vectors. The given Fourier data might be noisy and the support length of $\mathbf{x}$ needs not to be given. The function `xrec = detRealNonnegSparseFFT(data)` returns the inverse Fourier Transform `xrec` for a given noisy Fourier vector `data`. The reconstruction $\mathbf{x}$ is assumed to be real and nonnegative.

```
function xrec=detRealNonnegSparseFFT(data)

% compute N and J
N=length(data);
J=log2(N);

% fix parameter for threshold
T=0.1;

% x^{(0)}=xhat(0)
xj=data(1);

% if xhat(0)=0, then x=0
if xj<T
    xrec=zeros(N,1);
else
    % set muj and support length for x^{(0)}
    muj=0;
    supplengthj=1;
    % compute first x^{(1)} (via case 1)
    xj=0.5*[xj+data(N/2+1);xj-data(N/2+1)];

    for j=1:J-1
        % compute the support length of the vector xj
        % first determine the support (xj from preceding reconstruction step
        % has zeros outside the possible support interval)

        %%%%%%%%%%%%%%%%%%%%%%%%%%
%       a = find(xj>=T);     % uncomment this for using "find" command to
                             % identify the indices where xj>=T
        %%%%%%%%%%%%%%%%%%%%%%%%%%

        % determine all indices where xj>=T in case that support lies
        % "modulo vector length" of the preceding periodization
        if muj>2^(j-1)-supplengthj
            s11=xj(1:muj-2^(j-1)+supplengthj);
```

```
    s12=xj(muj+1:2^(j-1));
    a11=find(s11>=T);
    a12=find(s12>=T)+muj;
    a1=[a11;a12];

    s21=xj(2^(j-1)+1:muj+supplengthj);
    s22=xj(2^(j-1)+muj+1:2^j);
    a21=find(s21>=T)+2^(j-1);
    a22=find(s22>=T)+2^(j-1)+muj;
    a2=[a21;a22];

    a=[a1;a2];      % contains indices where xj>=T

% determine all indices where xj>=T in case that support lies
% "in the middle" of the preceding periodization
else
    supp1=xj(muj+1:muj+supplengthj);
    supp2=xj(muj+1+2^(j-1):muj+supplengthj+2^(j-1));
    a1=find(supp1>=T)+muj;
    a2=find(supp2>=T)+muj+2^(j-1);

    a=[a1;a2];      % contains indices where xj>=T
end
%%%%%%%%%%%%%%%%%%%%%%%%%%

if isempty(a)
    xrec=zeros(N,1);
    return;
end

b=[a(end)-2^j;a];
% compute the differences between the indices in a
c(1:length(a))=b(2:length(b))-b(1:length(a));
% the support interval is the "complement" of the longest interval
% of zeros
supplengthj=2^j-max(c)+1;

% find last index of the longest interval of zeros; the subsequent
% index is the first support index
d=find(c==max(c),1);
muj=b(d+1)-1;

if supplengthj>2^(j-1)      % case 1
    % set entries outside the support of xj to zero and apply
    % threshold within support
    xj(mod(muj+supplengthj:muj+2^j-1,2^j)+1)=0;
    xj=real(xj);
    xj(mod(muj+(0:supplengthj-1),2^j)+1)=...
            xj(mod(muj+(0:supplengthj-1),2^j)+1).*...
            (xj(mod(muj+(0:supplengthj-1),2^j)+1)>=T);
```

```
            y(1:2^j,1)=data(2^(J-j-1)*(2*(1:2^j)-1)+1);
            z=exp(pi*1i*(0:2^j-1)/2^j).'.*ifft(y);
            % (only compute the parts where support of the next periodization
            % can be located)
            z(mod(muj+supplengthj:muj+2^j-1,2^j)+1)=0;
            xj=0.5*[xj+z;xj-z];

        else                            % case 2
            Lj=ceil(log2(supplengthj));       % compute L_j
            % pick interval of length 2^Lj from xj which contains support
            u=xj(mod(muj+(0:(supplengthj-1)),2^j)+1);
            u=real(u);
            u(u<T)=0;
            u(supplengthj+1:2^Lj)=0;
            y=data(2^(J-Lj)*(0:2^Lj-1)+2^(J-j-1)+1);

            if muj>2^j-2^Lj
                z=  exp(pi*1i*muj/2^j)*...
                    [ones(2^j-muj,1);-ones(2^Lj-2^j+muj,1)].*...
                    exp(pi*1i*(0:(2^Lj-1)).'/2^j).*...
                    ifft(exp(2*pi*1i*muj*(0:2^Lj-1).'/2^Lj).*y);
            else
                z=  exp(pi*1i*muj/2^j)*...
                    exp(pi*1i*(0:(2^Lj-1)).'/2^j).*...
                    ifft(exp(2*pi*1i*muj*(0:2^Lj-1).'/2^Lj).*y);
            end

            z(supplengthj+1:2^Lj)=0; % (only compute the parts where support of
                                     % the next periodization can be located)
            % two parts of support of new xj
            v0=0.5*(u+z);
            v1=0.5*(u-z);

            % "insert" support into vectors of length 2^j
            w0=zeros(2^j,1);
            w0(mod(muj+(0:(2^Lj-1)),2^j)+1)=v0(1:2^Lj);
            w1=zeros(2^j,1);
            w1(mod(muj+(0:(2^Lj-1)),2^j)+1)=v1(1:2^Lj);

            % build xj
            xj=[w0;w1];
        end
    end

    % determine the support of x, as in each iteration step above

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%    a=find(xj>=T);  % uncomment this for using "find" command to identify
                     % the indices where xj>=T
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
    if muj>2^(J-1)-supplengthj
        s11=xj(1:muj-2^(J-1)+supplengthj);
        s12=xj(muj+1:2^(J-1));
        a11=find(s11>=T);
        a12=find(s12>=T)+muj;
        a1=[a11;a12];

        s21=xj(2^(J-1)+1:muj+supplengthj);
        s22=xj(2^(J-1)+muj+1:2^J);
        a21=find(s21>=T)+2^(J-1);
        a22=find(s22>=T)+2^(J-1)+muj;
        a2=[a21;a22];

        a=[a1;a2];
    else
        supp1=xj(muj+1:muj+supplengthj);
        supp2=xj(muj+1+2^(J-1):muj+supplengthj+2^(J-1));
        a1=find(supp1>=T)+muj;
        a2=find(supp2>=T)+muj+2^(J-1);

        a=[a1;a2];
    end
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    if isempty(a)
        xrec=zeros(N,1);
        return;
    end

    b=[a(end)-2^J;a];
    c(1:length(a))=b(2:length(b))-b(1:length(a));
    supplengthj=2^J-max(c)+1;
    d=find(c==max(c),1);
    muj=b(d+1)-1;

    % set entries outside the support of xj to zero
    xj(mod(muj+supplengthj:muj+2^J-1,2^J)+1)=0;
    xj=real(xj);
    xj(mod(muj+(0:supplengthj-1),2^J)+1)=...
              xj(mod(muj+(0:supplengthj-1),2^J)+1).*...
              (xj(mod(muj+(0:supplengthj-1),2^J)+1)>=T);

    xrec=xj;

end
end
```

A.2.2. Reconstruction of deterministic sampling vectors

The following MATLAB code generates a deterministic sampling vector and applies Algorithm 5.4 to its perturbed Fourier transform.

```
% set parameters
N = 2^10;          % vector length (must be power of 2)

% set vector entries
x=zeros(N,1);
x(1)=1;
x(2)=2;
x(3)=3;
x(4)=4;
x(5)=5;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% create noisy Fourier data

SNR = 20;            % set desired SNR

xhat=fft(x);         % Fourier transform

noise=rand(N,1)-0.5+1i*(rand(N,1)-0.5);           % create random uniform noise
noise=norm(xhat)*10^(-SNR/20)/norm(noise)*noise;  % scale according to SNR
xhatnoise=xhat+noise;                             % noisy input data

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% apply algorithm to noisy data

xrec=detRealNonnegSparseFFT(xhatnoise);
```

A.2.3. Reconstruction of random sampling vectors

The following MATLAB code generates a random sampling vector and applies Algorithm 4.12 to its perturbed Fourier transform.

```
%set parameters
N = 2^15;            % vector length (must be power of 2)
m = 15;              % support length

% calculate random sampling vector
x=zeros(N,1);                          % allocate space
L=ceil(log2(m));                       % calculate L
firstsuppind=randi(N);                 % random first support index
                                       % calculate support
supp=[firstsuppind:min(firstsuppind+m-1,N),1:firstsuppind+m-1-N];
suppzeros=randi(m-2);                  % random number of zeros inside support
supp(randperm(m-2,suppzeros)+1)=[];    % delete number of suppzeros support
```

```
                                    % indices (not first or last)
x(supp)=10*(rand(m-suppzeros,1));   % set random complex values

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% create noisy Fourier data

SNR = 20;            % set desired SNR

xhat=fft(x);         % Fourier transform

noise=rand(N,1)-0.5+1i*(rand(N,1)-0.5);          % create random uniform noise
noise=norm(xhat)*10^(-SNR/20)/norm(noise)*noise; % scale according to SNR
xhatnoise=xhat+noise;                            % noisy input data

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% apply algorithm to noisy data

xrec=detRealNonnegSparseFFT(xhatnoise);
```

A.3. Implementation of Algorithm 7.6 for matrices with small support

A.3.1. Algorithm

The following MATLAB code gives an exemplary implementation of Algorithm 7.6. The function `detSmallSuppSparseFFT` used below is Algorithm 4.12 for the reconstruction of vectors with small support whose implementation is given in Section A.1. The function `xrec = det2DSmallSuppSparseFFT(data,m)` returns the inverse Fourier Transform `xrec` for given Fourier data `data` with noise. The reconstruction is assumed to have short support where `m1` $\times$ `m2` is an upper bound.

```
function xrec = det2DSmallSuppSparseFFT(data,m1,m2)

[N1,N2]=size(data);

% apply column transforms
for r=1:N2
    Xa(:,r)=detSmallSuppSparseFFT(data(:,r),m1);
end

% apply row transforms
for s=1:N1
    Xrec(s,:)=detSmallSuppSparseFFT(Xa(s,:).',m2).';
end
end
```

A.3.2. Reconstruction of deterministic sampling matrices

The following MATLAB code generates a deterministic sampling matrix and applies Algorithm 7.6 to its perturbed Fourier transform.

```
% set parameters
N1 = 2^4;            % number of rows (must be power of 2)
N2 = 2^4;            % number of columns (must be power of 2)
m1 = 3;              % support length (should be at most N1/4)
m2 = 3;              % support length (should be at most N2/4)

% set matrix entries
X=zeros(N1,N2);
X(3,2)=8;
X(3,3)=-3;
X(4,3)=-5;
X(4,4)=2;
X(5,2)=-1;
X(5,4)=4;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% create noisy Fourier data

SNR = 20;             % set desired SNR

Xhat=fft2(X);         % Fourier transform

Noise=rand(N1,N2)-0.5+1i*(rand(N1,N2)-0.5);      % create random uniform noise
                                                 % scale according to SNR
Noise=norm(Xhat,'fro')*10^(-SNR/20)/norm(Noise,'fro')*Noise;
Xhatnoise=Xhat+Noise;                            % noisy input data

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% apply algorithm to noisy data

xrec = det2DSmallSuppSparseFFT(data,m1,m2);
```

A.3.3. Reconstruction of random sampling matrices

The following MATLAB code generates a random sampling matrix and applies Algorithm 7.6 to its perturbed Fourier transform.

```
% set parameters
N1 = 2^10;           % number of rows (must be power of 2)
N2 = 2^10;           % number of columns (must be power of 2)
```

```
m1 = 10;            % support length (should be at most N1/4)
m2 = 10;            % support length (should be at most N2/4)

% calculate random sampling matrix
X=zeros(N1,N2);                          % allocate space
fsind=[randi(N1-m1+1),randi(N2-m2+1)];   % random first support index

X(fsind(1):fsind(1)+m1-1,fsind(2):fsind(2)+m2-1)=...
                                20*(rand(m1,m2)-0.5+1i*(rand(m1,m2)-0.5));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% create noisy Fourier data

SNR = 20;             % set desired SNR

Xhat=fft2(X);         % Fourier transform

Noise=rand(N1,N2)-0.5+1i*(rand(N1,N2)-0.5);      % create random uniform noise
                                                 % scale according to SNR
Noise=norm(Xhat,'fro')*10^(-SNR/20)/norm(Noise,'fro')*Noise;
Xhatnoise=Xhat+Noise;                            % noisy input data

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% apply algorithm to noisy data

xrec = det2DSmallSuppSparseFFT(data,m1,m2);
```

Bibliography

[1] A. Akavia. Deterministic sparse Fourier approximation via fooling arithmetic progressions. In *Proceedings of the 23rd International Conference on Learning Theory*, pages 381–393, 2010.

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

[3] D. Batenkov and Y. Yomdin. On the accuracy of solving confluent Prony systems. *SIAM Journal on Applied Mathematics*, 73(1):134–154, 2013.

[4] L. Berman and A. Feuer. On perfect conditioning of Vandermonde matrices on the unit circle. *Electronic Journal of Linear Algebra*, 16:157–161, 2007.

[5] G. Beylkin and L. Monzón. On approximation of functions by exponential sums. *Applied and Computational Harmonic Analysis*, 19(1):17–48, 2005.

[6] cameraman image. cameraman.tif, available in Matlab Image Processing Toolbox, see http://www.mathworks.com/help/pdf_doc/images/images_tb.pdf.

[7] E. J. Candes, J. K. Romberg, and T. Tao. Stable signal recovery from incomplete and inaccurate measurements. *Communications on Pure and Applied Mathematics*, 59(8):1207–1223, 2006.

[8] A. Christlieb, D. Lawlor, and Y. Wang. A multiscale sub-linear time Fourier algorithm for noisy data. *Applied and Computational Harmonic Analysis*, 40(3):553–574, 2016.

[9] J. W. Cooley and J. W. Tukey. An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computation*, 19:297–301, 1965.

[10] B. G. R. de Prony. Essai éxperimental et analytique: sur les lois de la Dilatabilité de fluides élastique et sur celles de la Force expansive de la vapeur de l'alkool, a différentes températures. *Journal de l'école polytechnique*, 1(22):24–76, 1795.

[11] C. J. Demeure. Fast QR factorization of Vandermonde matrices. *Linear Algebra and its Applications*, 122/123/124:165–194, 1989.

[12] D. L. Donoho. Compressed sensing. *IEEE Transactions on Information Theory*, 52(4):1289–1306, 2006.

[13] P. L. Dragotti, M. Vetterli, and T. Blu. Sampling moments and reconstructing signals of finite rate of innovation: Shannon meets Strang-Fix. *IEEE Transactions on Signal Processing*, 55(5, part 1):1741–1757, 2007.

[14] F. Filbir, H. N. Mhaskar, and J. Prestin. On the problem of parameter estimation in exponential sums. *Constructive Approximation*, 35(3):323–343, 2012.

[15] A. C. Gilbert, S. Guha, P. Indyk, S. Muthukrishnan, and M. Strauss. Near-optimal sparse Fourier representations via sampling. In *Proceedings of the 34th annual ACM Symposium on Theory of Computing*, pages 152–161. ACM, 2002.

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

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

[18] A. C. Gilbert, M. J. Strauss, and J. A. Tropp. A tutorial on fast fourier sampling. *IEEE Signal Processing Magazine*, 25(2):57–66, 2008.

[19] J. J. Green. Calculating the maximum modulus of a polynomial using Stečkin's lemma. *SIAM Journal on Numerical Analysis*, 36(4):1022–1029, 1999.

[20] H. Hassanieh, F. Adib, D. Katabi, and P. Indyk. Faster GPS via the sparse Fourier transform. In *Proceedings of the 18th Annual International Conference on Mobile Computing and Networking*, pages 353–364. ACM, 2012.

[21] H. Hassanieh, P. Indyk, D. Katabi, and E. Price. Nearly optimal sparse Fourier transform. In *Proceedings of the 44th Annual ACM Symposium on Theory of Computing*, pages 563–578. ACM, 2012.

[22] H. Hassanieh, P. Indyk, D. Katabi, and E. Price. Simple and practical algorithm for sparse Fourier transform. In *Proceedings of the 23rd annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1183–1194. SIAM, 2012.

[23] H. Hassanieh, L. Shi, O. Abari, E. Hamed, and D. Katabi. GHz-wide sensing and decoding using the sparse Fourier transform. In *IEEE Conference on Computer Communications*, pages 2256–2264. IEEE, 2014.

[24] Y. Hua and T. K. Sarkar. Matrix pencil method for estimating parameters of exponentially damped/undamped sinusoids in noise. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 38(5):814–824, 1990.

[25] P. Indyk and M. Kapralov. Sample-optimal fourier sampling in any constant dimension. In *IEEE 55th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 514–523. IEEE, 2014.

[26] P. Indyk, M. Kapralov, and E. Price. (Nearly) sample-optimal sparse Fourier transform. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 480–499. SIAM, 2014.

[27] M. A. Iwen. Combinatorial sublinear-time Fourier algorithms. *Foundations of Computational Mathematics*, 10(3):303–338, 2010.

[28] M. A. Iwen. Improved approximation guarantees for sublinear-time Fourier algorithms. *Applied and Computational Harmonic Analysis*, 34(1):57–82, 2013.

[29] D. Lawlor, Y. Wang, and A. Christlieb. Adaptive sub-linear time Fourier algorithms. *Advances in Adaptive Data Analysis*, 5(01):1350003, 2013.

[30] A. Moitra. Super-resolution, extremal functions and the condition number of Vandermonde matrices. In *Proceedings of the 2015 ACM Symposium on Theory of Computing*, pages 821–830. ACM, 2015.

[31] H. L. Montgomery and R. C. Vaughan. Hilbert's inequality. *Journal of the London Mathematical Society (2)*, 8:73–82, 1974.

[32] J. Morgenstern. Note on a lower bound of the linear complexity of the fast Fourier transform. *Journal of the Association for Computing Machinery*, 20:305–306, 1973.

[33] H. J. Nussbaumer. *Fast Fourier transform and convolution algorithms*, volume 2 of *Springer Series in Information Sciences*. Springer-Verlag, 1981.

[34] 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. In *IEEE International Symposium on Information Theory Proceedings (ISIT)*, pages 464–468. IEEE, 2013.

[35] T. Peter and G. Plonka. A generalized Prony method for reconstruction of sparse sums of eigenfunctions of linear operators. *Inverse Problems*, 29(2):025001, 21, 2013.

[36] T. Peter, D. Potts, and M. Tasche. Nonlinear approximation by sums of exponentials and translates. *SIAM Journal on Scientific Computing*, 33(4):1920–1947, 2011.

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

[38] G. Plonka and K. Wannenwetsch. A deterministic sparse FFT algorithm for vectors with short support. *Oberwolfach Reports*, 38:41–44, 2015.

[39] G. Plonka and K. Wannenwetsch. Deterministic sparse FFT algorithms. *PAMM*, 15(1):667–668, 2015.

[40] G. Plonka and K. Wannenwetsch. A deterministic sparse FFT algorithm for vectors with small support. *Numerical Algorithms*, 71(4):889–905, 2016.

[41] G. Plonka and K. Wannenwetsch. A sparse fast Fourier algorithm for real nonnegative vectors. *arXiv preprint, arXiv:1602.05444*, 2016.

[42] D. Potts, S. Kunis, S. Heider, and M. Veit. A sparse Prony FFT. In *10th International Conference on Sampling Theory and Applications (SampTA)*, pages 572–575, 2013.

[43] D. Potts, G. Steidl, and M. Tasche. Fast fourier transforms for nonequispaced data: A tutorial. In *Modern sampling theory*, pages 247–270. Springer, 2001.

[44] D. Potts and M. Tasche. Parameter estimation for exponential sums by approximate Prony method. *Signal Processing*, 90(5):1631–1642, 2010.

[45] D. Potts and M. Tasche. Nonlinear approximation by sums of nonincreasing exponentials. *Applicable Analysis*, 90(3-4):609–626, 2011.

[46] D. Potts and M. Tasche. Parameter estimation for nonincreasing exponential sums by Prony-like methods. *Linear Algebra and its Applications*, 439(4):1024–1039, 2013.

[47] D. Potts, M. Tasche, and T. Volkmer. Efficient spectral estimation by MUSIC and ESPRIT with application to sparse FFT. *Frontiers in Applied Mathematics and Statistics*, 2:1, 2016.

[48] R. Roy and T. Kailath. ESPRIT-estimation of signal parameters via rotational invariance techniques. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 37(7):984–995, 1989.

[49] R. Schmidt. Multiple emitter location and signal parameter estimation. *IEEE Transactions on Antennas and Propagation*, 34(3):276–280, 1986.

[50] I. R. Shafarevich and A. Remizov. *Linear Algebra and Geometry*. Springer, 2012.

[51] L. Shi, O. Andronesi, H. Hassanieh, B. Ghazi, D. Katabi, and E. Adalsteinsson. MRS Sparse-FFT: Reducing Acquisition Time And Artifacts For In Vivo 2D Correlation Spectroscopy. In *ISMRM'13, Int. Society for Magnetic Resonance in Medicine Annual Meeting and Exhibition*, 2013.

[52] G. Steidl and M. Tasche. *Schnelle Fourier-Transformationen - Theorie und Anwendungen*. Lehrbriefe der FernUniversität Hagen. 1998.

[53] C. Van Loan. *Computational frameworks for the fast Fourier transform*, volume 10. SIAM, 1992.

[54] M. Vetterli, P. Marziliano, and T. Blu. Sampling signals with finite rate of innovation. *IEEE Transactions on Signal Processing*, 50(6):1417–1428, 2002.

[55] P. K. Yenduri and A. C. Gilbert. Compressive, collaborative spectrum sensing for wideband cognitive radios. In *2012 International Symposium on Wireless Communication Systems (ISWCS)*, pages 531–535. IEEE, 2012.

Curriculum vitae