MATHEMATICS OF COMPUTATION
Volume 83, Number 289, September 2014, Pages 2347–2384
S 0025-5718(2014)02785-3
Article electronically published on February 11, 2014

FAST COMPUTATION OF THE MULTIDIMENSIONAL DISCRETE FOURIER TRANSFORM AND DISCRETE BACKWARD FOURIER TRANSFORM ON SPARSE GRIDS

YING JIANG AND YUESHENG XU

Abstract. We propose a fast discrete Fourier transform for a given data set which may be generated from sampling a function of d-variables on a sparse grid and a fast discrete backward Fourier transform on a hyperbolic cross index set. Computation of these transforms can be formulated as evaluation of *dimension-reducible* sums on sparse grids. We introduce a fast algorithm for evaluating such sums and prove that the total number of operations needed in the algorithm is $\mathcal{O}(n \log^d n)$, where n is the number of components along each coordinate direction of the data set. We then use it to develop fast algorithms for computing the discrete Fourier transform on the sparse grid and the discrete backward Fourier transform on the hyperbolic cross index set. We also show that if the given data set is sampled from a function having regularity of order s, then its discrete Fourier transform has the optimal approximation order $\mathcal{O}(n^{-s})$. Numerical examples are presented to demonstrate the approximation accuracy and computational efficiency of the proposed algorithms.

1. Introduction

A given data set on a sparse grid in $\mathbb{R}^d$ with d being a fixed positive integer may be viewed as a set of functional values sampled from a function of d-variables on the sparse grid. From the given data set we may reconstruct the function by using the hyperbolic cross approximation of d-dimensional multiscale piecewise interpolating polynomials. The discrete Fourier transform of the data set (which we also view as a vector) considered in this paper is the set of the Fourier coefficients of the resulting reconstructed function, and its outcome is a data set indexed with a hyperbolic cross index set. While the discrete backward Fourier transform of a data set on a hyperbolic cross index set is the set of values of the trigonometric polynomial, having the given data set as its coefficients, evaluated at the given sparse grid. The outcome of the discrete backward Fourier transform is the values of the linear

Received by the editor May 9, 2012 and, in revised form, January 6, 2013.

2010 *Mathematics Subject Classification.* Primary 65T50.

Key words and phrases. Multidimensional Fourier transform, multidimensional backward Fourier transform.

This work was supported in part by the Guangdong Provincial Government of China through the "Computational Science Innovative Research Team" program.

The authors were supported in part by the Young Scientist Fund of the National Natural Science Foundation of China under grant 11101439 and the Doctor Program Foundation of Ministry of Education of China under grant 20100171120038.

The second author (corresponding author) was supported in part by the US Air Force Office of Scientific Research under grant FA9550-09-1-0511, by the US National Science Foundation under grant DMS-1115523, and by the Natural Science Foundation of China under grants 11071286 and 91130009.

combination of the Fourier basis functions, with the coefficients being the given data set, evaluated at the sparse grid points. We use here the terminology "discrete backward Fourier transform" instead of "discrete inverse Fourier transform" since an application of the backward transform does not recover exactly the original data set which has been transformed by the discrete Fourier transform. It gives only an approximation of the original data set.

The main purpose of this paper is to develop a fast algorithm for computing the discrete Fourier transform of a given data set which may be generated from sampling a function of d-variables on a sparse grid and for computing the discrete backward Fourier transform on a hyperbolic cross index set. We unify the computation problems of these transforms as an evaluation problem of dimension-reducible sums on sparse grids in $\mathbb{R}^d$ and introduce a fast algorithm for this evaluation. We then specialize the general methodology to computing the discrete Fourier transform on the sparse grid and the discrete backward Fourier transform on the hyperbolic cross index set. When there is no ambiguity, we shall simply call the discrete Fourier transform on the sparse grid *the discrete Fourier transform* and the discrete backward Fourier transform on the hyperbolic cross index set *the discrete backward Fourier transform.* This paper is a continuation of two recent papers: [15], where a fast discrete algorithm for the sparse Fourier transform of a function of d-variables was developed, and [17], where a fast algorithm for evaluating B-spline quasi-interpolants on sparse grids was proposed. The hyperbolic cross approximation plays an important role in this development. Fast algorithms for evaluating sums on sparse grids were presented in [1,3,26], where [3] employed the algorithm in [1] to construct a multigrid algorithm for higher order finite elements on sparse grids, and [26] extended the algorithm in [1] to more general sparse grids.

The hyperbolic cross approximation is an important tool for efficiently representing a function of high dimensions. It achieves an optimal approximation order for a function with bounded mixed derivatives by using the quasi-linear number of Fourier basis functions [19,22,23]. This nice feature leads to a wide use of the hyperbolic cross approximation in solving partial differential equations and high-dimensional integral equations (see, for example, [5,8,10,12,18,20,25]). In the numerical solutions of partial differential equations and integral equations, by using the hyperbolic cross approximation, there are two critical computational issues. The first one is computing the Fourier coefficients used in the hyperbolic cross approximation, and the second is evaluating the trigonometric polynomial obtained from the hyperbolic cross approximation at given points. These issues were addressed in the literature [11,12,14,15,17,21]. The algorithms proposed in [14] for computing the discrete Fourier transform of a data set on sparse grids at hyperbolic cross domain and discrete inverse Fourier transform of a data set on a hyperbolic cross index set at sparse grids requires $\mathcal{O}(n \log^d n)$ number of operations, where n is the order of the univariate trigonometric polynomial used in constructing the sparse multivariate approximation by the tensor product. The function constructed by this algorithm has the approximation order $\mathcal{O}((\log n + 1)^{d-1} n^{-(s-1)})$, where $s > 0$ is the order of the Sobolev regularity of the function from which the data on sparse grids are sampled [12]. The approximation order of the algorithms is not optimal. In particular, for functions of low regularity such as $s < 1$, the algorithms have no approximation order. Most high dimensional data of practical importance have low order of regularity. Therefore, there is a need to develop efficient algorithms suitable for a large class of functions including those having low regularity (for example, $s < 1$). Moreover, the employed sparse grids in [14] must be generated from the zero points of the hierarchical bases defined in [14], which may limit the range of applicability of algorithms presented in [14].

The algorithm introduced in [15] by the same authors of this paper can be used to compute the Fourier transform of the data having a low order of regularity. It has the optimal approximation order $\mathcal{O}(n^{-s})$ for functions whose ℓ-th order mixed partial derivatives are bounded and $\ell > s$ and it requests $\mathcal{O}(n \log^{2d-1} n)$ number of operations. While this algorithm enjoys the optimal approximation order, its computational complexity has an extra $\log^{d-1} n$ factor. This algorithm was applied to constructing fast algorithms for solving singular boundary integral equations [16] and first-kind logarithmic-kernel integral equations on open arcs [24].

The main purpose of this paper is to develop a fast algorithm for evaluating a sum, taken over a *source* sparse grid, of tensor product functions at points of a *target* sparse grid. The proposed algorithm is designed based on the property that both the source and the target sparse grids are *dimension-reducible* in the following sense. A set of d dimensions is called dimension-reducible if it can be represented as a union of sets, each of which is the tensor product (which we call a cell) of a one-dimensional set and a $(d-1)$-dimensional set that has the same structure as the original d-dimensional set, with the one dimensional sets being disjoint and the $(d-1)$-dimensional sets being a nested sequence. The related one-dimensional set is called the projection of the cell on one dimension and the $(d-1)$-dimensional set is called the projection of the cell on the $d-1$ dimensions. Therefore, evaluating such a sum (which we call a dimension-reducible sum), taken over a source sparse grid, of tensor product functions at points of a target sparse grid reduces to evaluating sums, taken over cells of the source dimension-reducible set of d dimensions, of the tensor product functions at points of cells of the target dimension-reducible set of d dimensions.

A dimension-reducible set shares the same structure as a tensor product set. Hence, a dimension-reducible sum can be evaluated in the same way as a tensor product sum (a sum of tensor product of univariate functions over a tensor product index set). Evaluating a tensor product sum at points on a tensor product set can be reduced to evaluating the univariate functions by using a formula recursive in dimensions. Likewise, evaluation of a sum of tensor product functions over a dimension-reducible source sparse grid at points of a dimension-reducible target sparse grid can be treated in a way similar to evaluating a tensor product sum. This is the key point used in this paper to develop a fast algorithm for evaluating a dimension-reducible sum.

Fast algorithms for computing the discrete Fourier transform and discrete backward Fourier transform are obtained by applying the fast algorithm for evaluating dimension-reducible sums. Let us elaborate the relations of these algorithms with those presented in [11, 14]. The algorithm for discrete Fourier transform developed in this paper produces more accurate results than those of [11, 14] for data sets having a low order of smoothness, while the algorithms in [11, 14] might be more efficient for data sets having a higher order of smoothness. The discrete inverse Fourier transform presented in [11, 14] is only applicable to evaluating the values of the sum at points on a sparse grid that is generated by employing the idea of [21] from the zero points of the hierarchical bases. The discrete backward Fourier transform developed in this paper is not restricted to sparse grids of this type. In particular, when the discrete backward Fourier transform developed in this paper is applied to sparse grids of this type, its output is algebraically identical to that produced by the discrete inverse Fourier transform presented in [11,14].

Evaluating a dimension-reducible sum may also be reformulated as a matrix-vector multiplication. Fast matrix-vector multiplications in the context of hyperbolic cross approximation were considered in [21,26]. The algorithm in [21] requests that the matrices involved should be block upper (or lower) triangular. This requirement was removed in the algorithm presented in [26]. We were not aware of this paper (which was published around the same time when the first version of our paper was submitted) until we received a referee's report. We would like to compare our proposed algorithm with that of [26]. The algorithm in [26] was described in two dimensions with an indication that it may be extended to general d dimensions. We feel that the extension can be technically nontrivial. Considering the connection between evaluation of a sum and matrix-vector multiplication, our fast algorithm described in this paper for evaluating a dimension-reducible sum is a d-dimensional version of the algorithm of [26]. In the algorithm we use recursive structures to address challenges brought by higher dimensions and memory management strategies in the algorithmic implementation to control the increase of the memory spaces required by the algorithm. In addition to this, there are other differences between the two algorithms. The idea used in this paper to develop the algorithm was obtained from the papers [2,17]. Moreover, our algorithm is for a more general setting—dimension-reducible sets—which include isotropic sparse grids, anisotropic sparse grids, full grids and the optimized grids in [13] as special examples, while the algorithm in [26] is for a specific sparse grid.

We organize this paper in seven sections. In section 2, we define the discrete Fourier transform on a sparse grid and the discrete backward Fourier transform on a hyperbolic cross index set, and prove that if the given data set is sampled from a function having regularity of order s, then its discrete Fourier transform has approximation order $\mathcal{O}(n^{-s})$. We then reformulate in section 3 the discrete Fourier transform on a sparse grid as a dimension-reducible sum. In section 4 we develop a fast algorithm for evaluating a dimension-reducible sum, and show that it requires only $\mathcal{O}(n\log^d n)$ number of operations and $\mathcal{O}(n\log^{d-1} n)$ number of memory usages. In sections 5 and 6, we specialize the general algorithm developed in section 4 to compute the discrete Fourier transform on a sparse grid and the discrete backward Fourier transform on a hyperbolic cross index set. We present four numerical examples in section 7 which confirm the theoretical estimates with a comparison to a known algorithm proposed in [11,12,14].

2. Discrete Fourier transform on sparse grids

In this section, we define the *discrete Fourier transform on a sparse grid* and the *discrete backward Fourier transform on a hyperbolic cross index set*.

We begin with describing the data set to which the discrete Fourier transform is applied. Let $d \in \mathbb{N}$ be a *fixed* integer. The data set considered in this paper may be generated from sampling a function of d-variables on a sparse grid. By $\mathbb{N}$ we denote the set of natural numbers and let $\mathbb{N}_0 := \{0\} \cup \mathbb{N}$. For each $k \in \mathbb{N}$, we set $\mathbb{Z}_k := \{0, 1, \ldots, k-1\}$, and for each $m \in \mathbb{N}$, we let $\mathbb{Z}_{0,m} := \mathbb{Z}_m$ and $\mathbb{Z}_{j,m} := \mathbb{Z}_{2^{j-1}m}$, $j \in \mathbb{N}$. For a given object $\mathbb{A}$, we let $\mathbb{A}^d := \mathbb{A} \otimes \mathbb{A} \otimes \cdots \otimes \mathbb{A}$ (d-folds), and in general, for each $\mathbf{j} := [j_k : k \in \mathbb{Z}_d] \in \mathbb{N}_0^d$ and given objects $\mathbb{A}_{j_k}$, we define

$$\mathbb{A}_{\mathbf{j}} := \mathbb{A}_{j_0} \otimes \mathbb{A}_{j_1} \otimes \cdots \otimes \mathbb{A}_{j_{d-1}},$$

where $\mathbb{A} \otimes \mathbb{B}$ denotes the tensor product of the objects $\mathbb{A}$ and $\mathbb{B}$. For each $\mathbf{j} := [j_k : k \in \mathbb{Z}_d] \in \mathbb{N}_0^d$, $\mathbf{r} := [r_k : k \in \mathbb{Z}_d] \in \mathbb{N}_0^d$ and given objects $\mathbb{A}_{j_k, r_k}$, we also define

$$\mathbb{A}_{\mathbf{j},\mathbf{r}} := \mathbb{A}_{j_0, r_0} \otimes \mathbb{A}_{j_1, r_1} \otimes \cdots \otimes \mathbb{A}_{j_{d-1}, r_{d-1}}.$$

These objects could be sets, operators, functionals, or functions which will become clear later in a specific context. For each $N \in \mathbb{N}_0$, we define a sparse grid by setting

$$\mathbb{S}_{N,d} := \left\{ \mathbf{j} \in \mathbb{Z}_{N+1}^d : |\mathbf{j}| \le N \right\}, \quad \text{where} \quad |\mathbf{j}| := \sum_{k \in \mathbb{Z}_d} j_k.$$

In this paper, we shall consider the discrete Fourier transform of a data set having the form

$$\mathbf{f}_N^d := [f_{\mathbf{j},\boldsymbol{\tau}} \in \mathbb{R} : \mathbf{j} \in \mathbb{S}_{N,d} \text{ and } \boldsymbol{\tau} \in \mathbb{Z}_{\mathbf{j},m}^d].$$

Such a data set is sometimes viewed as a vector. The domain of the data set $\mathbf{f}_N^d$ is illustrated by examples $\mathbf{f}_6^2$ and $\mathbf{f}_6^3$, respectively, in images (a) and (b) of Figure 1. The discrete Fourier transform of a data set $\mathbf{f}_N^d$ is the set of the Fourier coefficients of the function of d-variables which interpolates the data set by using hyperbolic cross approximation of d one-dimensional multiscale piecewise interpolating polynomials.

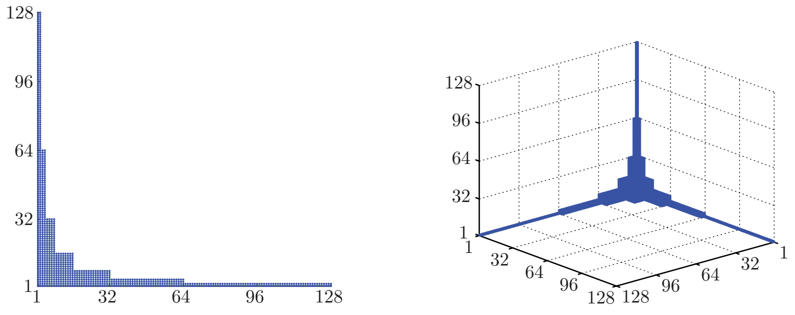


FIGURE 1. Domains of $\mathbf{f}_6^2$ (left) and $\mathbf{f}_6^3$ (right)

We now construct the function that interpolates the data set $\mathbf{f}_N^d$. In this paper, we assume that the data set $\mathbf{f}_N^d$ is obtained from sampling a function f of d variables on a sparse grid. Thus, to construct the function that interpolates the data set $\mathbf{f}_N^d$, we first review a piecewise polynomial introduced in [15] that interpolates on a sparse grid. For this purpose, we recall the one-dimensional multiscale piecewise polynomial interpolation, introduced in [6]. As in [6], we use the refinable set as a tool to describe the set of the interpolation points. We choose a nested set sequence of interpolation points on $I := [0, 2\pi]$. Note that the interval I is the invariant set with respect to the two contractive mappings

$$\Psi := \{\psi_0, \psi_1\}, \quad \text{where} \quad \psi_0(x) := \frac{x}{2} \quad \text{and} \quad \psi_1(x) := \frac{x + 2\pi}{2}, \quad x \in I.$$

A subset V of I is said to be refinable relative to the mappings Ψ if $V \subseteq \Psi(V) := \psi_0(V) \cup \psi_1(V)$. For $m \in \mathbb{N}$, we choose a refinable set

$$V := \{v_r : 0 \le v_0 < v_1 < \cdots < v_{m-1} < 2\pi, r \in \mathbb{Z}_m\}$$

relative to the mappings Ψ, and let $V_{-1} := \emptyset$, $V_0 := \{v_{0,r} := v_r : r \in \mathbb{Z}_m\}$. For each $j \in \mathbb{N}$ and $\mathbf{p}_j := [\varrho_\gamma : \gamma \in \mathbb{Z}_j] \in \mathbb{Z}_2^j$, we define $\psi_{\mathbf{p}_j} := \psi_{\varrho_{j-1}} \circ \psi_{\varrho_{j-2}} \circ \cdots \circ \psi_{\varrho_0}$ and the dyadic expansion

$$\mu(\mathbf{p}_j) := \sum_{\gamma \in \mathbb{Z}_j} \varrho_\gamma 2^\gamma.$$

For $\mathbf{p}_j \in \mathbb{Z}_2^j$ and $r' \in \mathbb{Z}_m$, we write $r = m\mu(\mathbf{p}_j) + r'$ and let $v_{j,r} := \psi_{\mathbf{p}_j}(v_{r'})$. We then define $V_j := \{v_{j,r} : r \in \mathbb{Z}_{2^j m}\}$, for $j \in \mathbb{N}_0$. Note that V_j is refinable relative to the mappings Ψ.

We now describe the Lagrange polynomials of one-variable to be used in the construction of the one-dimensional multiscale piecewise polynomial interpolation. For $x \in \mathbb{R}$, we denote by $\lfloor x \rfloor$ the largest integer not greater than x. Associated with the sets V_j, $j \in \mathbb{N}_0$, we define the Lagrange polynomials $\ell_{0,r}$ of degree $m-1$ on I by requiring $\ell_{0,r}(v_{r'}) = \delta_{r,r'}$ for $r, r' \in \mathbb{Z}_m$, where $\delta_{r,r'} = 1$ if $r = r'$, or 0 if $r \neq r'$, and the piecewise polynomials $\ell_{j,r}$ by requiring that $\ell_{j,r}(x) = 0$, for $x \in I \setminus \psi_{\mathbf{p}_j}(I)$ where r and $\mathbf{p}_j$ satisfy $\lfloor r/m \rfloor = \mu(\mathbf{p}_j)$, and for $j \geq 1$ and for each $r \in \mathbb{Z}_{2^j m}$, $\ell_{j,r}$ is the Lagrange polynomial of degree $m-1$ on $\psi_{\mathbf{p}_j}(I)$ with the property that $\ell_{j,r}(v_{j,r'}) = \delta_{r,r'}$, for $r, r' \in \mathbb{Z}_{2^j m}$. With the above notation, for $j \in \mathbb{N}_0$ we introduce the piecewise polynomial interpolation $\mathcal{P}_j g$ for a continuous univariate function g defined on I by

$$\mathcal{P}_j g := \sum_{r \in \mathbb{Z}_{2^j m}} g(v_{j,r}) \ell_{j,r}.$$

Let $\mathcal{Q}_0 := \mathcal{P}_0$ and $\mathcal{Q}_j := \mathcal{P}_j - \mathcal{P}_{j-1}$ for all $j \in \mathbb{N}$. Hence, for each $N \in \mathbb{N}$, we re-express $\mathcal{P}_N$ as

$$\mathcal{P}_N = \sum_{j \in \mathbb{Z}_{N+1}} \mathcal{Q}_j. \tag{2.1}$$

Formula (2.1) allows us to develop the piecewise polynomial that interpolates on a sparse grid. For $f \in C(I^d)$, the space of all continuous functions defined on I^d, and $N \in \mathbb{N}$, we define

$$\mathcal{S}_N f := \sum_{\mathbf{j} \in \mathbb{S}_{N,d}} \mathcal{Q}_{\mathbf{j}} f. \tag{2.2}$$

When $\mathbf{f}_N^d$ is the values of f on the sparse grid, we let $\mathcal{S}_N(\mathbf{f}_N^d) := \mathcal{S}_N(f)$.

We are now ready to describe the discrete Fourier transform of $\mathbf{f}_N^d$ on the hyperbolic cross index set. For a fixed $\tilde{m} \in \mathbb{N}$, we introduce the index sets

$$\mathbb{I}_j := \left\{ l \in \mathbb{Z} : \lfloor 2^{j-1} \rfloor \tilde{m} \leq |l| < 2^j \tilde{m} \right\}, \quad j \in \mathbb{N}_0.$$

For each $N \in \mathbb{N}$, we denote by $\mathbb{J}_{N,d}$ the union of $\mathbb{I}_{\mathbf{j}}$ for all $\mathbf{j} \in \mathbb{S}_{N,d}$ and call it the *hyperbolic cross index set*. It was shown in [22] that the cardinality of $\mathbb{J}_{N,d}$ is given by $\mathcal{C}(\mathbb{J}_{N,d}) = \mathcal{O}(2^N N^{d-1})$. We illustrate the sparsity of $\mathbb{J}_{N,d}$ with two examples: $\mathbb{J}_{5,2}$ and $\mathbb{J}_{5,3}$ with $\tilde{m} = 1$ in Figure 2. We use $L^2(I^d)$ for the standard Hilbert space of the square integrable functions on I^d, with the usual inner product $\langle \cdot, \cdot \rangle$ and the norm $\|\cdot\| := \langle \cdot, \cdot \rangle^{\frac{1}{2}}$. Let the Fourier basis e_l, $l \in \mathbb{Z}$, be defined by $e_l(x) := \frac{1}{(2\pi)^{1/2}} e^{ilx}$, for $x \in I$, where i denotes the imaginary unit. It is well known that the Fourier basis $e_{\mathbf{l}}$, $\mathbf{l} \in \mathbb{Z}^d$, constitutes an orthonormal basis for the space $L^2(I^d)$. The discrete Fourier transform $\hat{\mathbf{f}}_N^d$ of $\mathbf{f}_N^d$ on $\mathbb{J}_{N,d}$ is defined as the coefficients having indices in

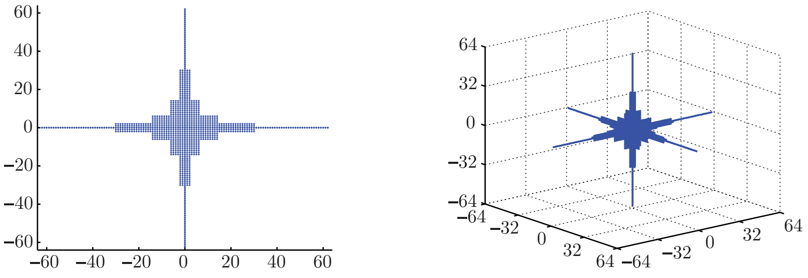


FIGURE 2. Sets $\mathbb{J}_{5,2}$ (left) and $\mathbb{J}_{5,3}$ (right)

the index set $\mathbb{J}_{N,d}$ of the Fourier expansion of function $\mathcal{S}_N(\mathbf{f}_N^d)$. That is,

$$\hat{\mathbf{f}}_N^d := \left[\left\langle \mathcal{S}_N(\mathbf{f}_N^d), e_{\mathbf{l}} \right\rangle : \mathbf{l} \in \mathbb{J}_{N,d}\right]. \tag{2.3}$$

For a given $\mathbf{c}_N^d := [c_{\mathbf{l}} : \mathbf{l} \in \mathbb{J}_{N,d}]$ with $c_{\mathbf{l}} \in \mathbb{R}$, we define the trigonometric function by

$$F_{\mathbf{c}_N^d} := \sum_{\mathbf{l} \in \mathbb{J}_{N,d}} c_{\mathbf{l}} e_{\mathbf{l}}. \tag{2.4}$$

In particular, if in (2.4) we choose $\mathbf{c}_N^d := \hat{\mathbf{f}}_N^d$, we obtain that

$$F_{\hat{\mathbf{f}}_N^d} := \sum_{\mathbf{l} \in \mathbb{J}_{N,d}} \hat{\mathrm{f}}_N^d(\mathbf{l}) e_{\mathbf{l}}.$$

Clearly, $F_{\hat{\mathbf{f}}_N^d}$ is the Fourier expansion of the function $\mathcal{S}_N(\mathbf{f}_N^d)$ with the hyperbolic cross index set $\mathbb{J}_{N,d}$. When the data set $\mathbf{f}_N^d$ is sampled from a given function $f \in C(I^d)$, we expect that the corresponding Fourier expansion $F_{\hat{\mathbf{f}}_N^d}$ would give a good approximation to f. We shall address this issue in the next section.

We now turn to defining the discrete backward Fourier transform of a data set on the hyperbolic cross index set $\mathbb{J}_{N,d}$ at a sparse grid. Such a transform is the set of values of the trigonometric polynomial indexed with set $\mathbb{J}_{N,d}$, having the given data set as its coefficients, evaluated at a given sparse grid which we describe next. Given a fixed $m \in \mathbb{N}$, we define a sequence of refinable sets by setting $\widetilde{V}_j := \left\{\frac{2\pi r}{2^j m} : r \in \mathbb{Z}_{2^j m}\right\}$, for $j \in \mathbb{N}_0$. Clearly, $\widetilde{V}_j$, $j \in \mathbb{N}_0$, have the properties that $\widetilde{V}_j \subset \widetilde{V}_{j+1}$ and $I = \overline{\bigcup_{j \in \mathbb{N}_0} \widetilde{V}_j}$. We let $\widetilde{G}_0 := \widetilde{V}_0$, and for each $j \in \mathbb{N}$, we define $\widetilde{G}_j := \widetilde{V}_j \setminus \widetilde{V}_{j-1}$. The sparse grid is then defined by

$$\widetilde{S}_{N,d} := \bigcup_{\mathbf{j} \in \mathbb{S}_{N,d}} \widetilde{G}_{\mathbf{j}}, \quad N \in \mathbb{N}.$$

The sparse grid $\widetilde{S}_{N,d}$, depending on the integer m, naturally associates with a piecewise polynomial of order m. In other words, it can be used as interpolation nodes for the multiscale interpolating piecewise polynomial of order m. The sparse grid $\widetilde{S}_{N,d}$ with $m = 1$ is the one considered in [11, 12, 14], which associates with the multiscale piecewise constant interpolation. The sparse grids with $m = 2$ and $m = 3$ are shown in (a) and (b) of Figure 3, respectively.

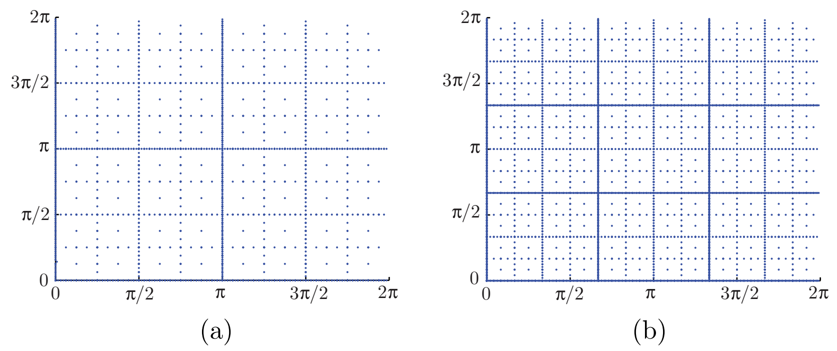


FIGURE 3. Image (a): $\widetilde{S}_{6,2}$ with $m = 2$. Image (b): $\widetilde{S}_{6,2}$ with $m = 3$

For a given data set $\mathbf{c}_N^d := [c_{\mathbf{l}} : \mathbf{l} \in \mathbb{J}_{N,d}]$, we define its discrete backward Fourier transform by setting

$$F_{\mathbf{c}_N^d}(\mathbf{v}) := \sum_{\mathbf{l}\in\mathbb{J}_{N,d}} c_{\mathbf{l}} e_{\mathbf{l}}(\mathbf{v}), \qquad \mathbf{v} \in \widetilde{S}_{N,d}. \tag{2.5}$$

We call $\check{\mathbf{c}}_N^d := [F_{\mathbf{c}_N^d}(\mathbf{v}) : \mathbf{v} \in \widetilde{S}_N^d]$ the discrete backward Fourier transform of $\mathbf{c}_N^d$ on $\widetilde{S}_{N,d}$.

To close this section, we remark on evaluation of the Fourier expansion on a hyperbolic cross index set at points in a *nonuniform* grid. Such an evaluation is crucial in high-dimensional data analysis. However, direct computing the function values is costly due to the noncompact support of the Fourier basis functions. We may first evaluate the Fourier expansion at points in a sparse grid, through which we can reconstruct a multiscale piecewise polynomial that interpolates the Fourier expansion. We then evaluate the piecewise polynomial at the points in the nonuniform grid. Due to the compact support property of the piecewise polynomial basis functions, evaluating the piecewise polynomial requires a significantly smaller amount of computational costs than directly computing the Fourier expansion. For more information on using the hyperbolic cross approximation in nonuniform Fourier transform, the readers are referred to [9]. The sparse grids considered here allow us to conveniently construct the interpolating piecewise polynomial of order m. The proposed methods with an arbitrary integer m provide flexibility in choosing an appropriate order of the piecewise polynomial that approximates the Fourier expansion to be evaluated.

3. A DIMENSION-REDUCIBLE SUM FOR THE DISCRETE FOURIER TRANSFORM

Computing the discrete Fourier transform on a sparse grid requires a convenient formula for the transform. For this purpose, we rewrite the discrete Fourier transform as a dimension-reducible sum so that both its source sparse grid and target sparse grid are dimension-reducible sets. We can evaluate the original d-dimensional dimension-reducible sum by computing $(d-1)$-dimensional dimension-reducible sums and one-dimensional sums. This leads us to a fast recursive algorithm for computing the discrete Fourier transform on a sparse grid.

The development of the dimension-reducible sum for computing the discrete Fourier transform $\hat{\mathbf{f}}_N^d$ of $\mathbf{f}_N^d$ demands the availability of a convenient formula for computing the function $\mathcal{S}_N(\mathbf{f}_N^d)$. Designing such a formula is the main task of this section. The dimension-reducible sum for computing the discrete Fourier transform $\hat{\mathbf{f}}_N^d$ will result from substituting the formula for $\mathcal{S}_N(\mathbf{f}_N^d)$ into the definition of the discrete Fourier transform (2.3). We take two steps to develop the formula for computing $\mathcal{S}_N(\mathbf{f}_N^d)$. In the first step, we recall the interpolation operator $\mathcal{S}_N$ presented originally in [15]. In the second step, by introducing a mapping which locates the function value $f(v_{j,r})$ in f_N^1, we derive the formula for computing $\mathcal{S}_N(\mathbf{f}_N^1)$. We then obtain the formula for computing $\mathcal{S}_N(\mathbf{f}_N^d)$ by using the tensor product.

We now review the interpolation operator $\mathcal{S}_N$. To this end, we first recall a formula for computing $\mathcal{Q}_j g$ for $j \in \mathbb{N}$ for a univariate function g. We need the following index sets:

$$\mathbb{W}_j := \{r \in \mathbb{Z}_{2^j m} : v_{j,r} \in V_j \setminus V_{j-1}\}, \quad \text{for } j \in \mathbb{N}.$$

Note that $\mathcal{Q}_j g$ is a interpolation projection of the remainder $g - \mathcal{P}_{j-1}g$, that is, $\mathcal{Q}_j g = \mathcal{P}_j(g - \mathcal{P}_{j-1}g)$, and for all $r \in \mathbb{Z}_{2^j m} \setminus \mathbb{W}_j$, we have that $(g - \mathcal{P}_{j-1}g)(v_{j,r}) = 0$. Thus, we know that

$$\mathcal{Q}_j g = \sum_{r \in \mathbb{W}_j} (g - \mathcal{P}_{j-1}g)(v_{j,r})\ell_{j,r}. \tag{3.1}$$

To illustrate the projection $\mathcal{Q}_j$, we plot in Figure 4 the basis employed in $\mathcal{Q}_j$, for $j = 0, 1$ and 2, with $m = 2$ and $V_0 = \left\{\frac{2\pi}{3}, \frac{4\pi}{3}\right\}$.

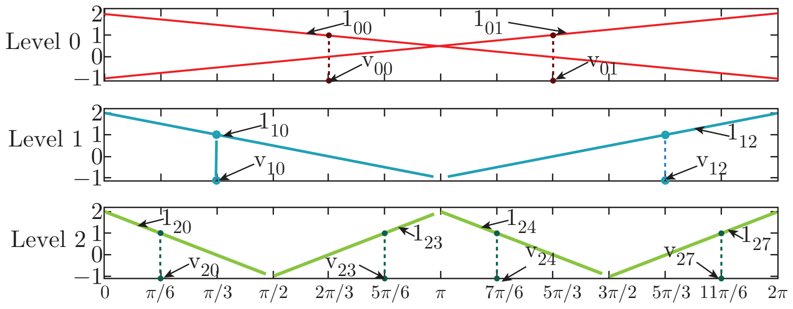


FIGURE 4. The basis employed in $\mathcal{Q}_j$, $j = 0, 1$ and 2

Note that formula (3.1) requires the availability of the values $(\mathcal{P}_{j-1}g)(v_{j,r})$. We next derive a formula for computing these values. Since $\mathcal{P}_{j-1}g$ is a piecewise polynomial, developing such a formula requires us to determine which basis functions $\ell_{j-1,\tilde{r}}$, $\tilde{r} \in \mathbb{W}_{j-1}$, are not vanished at $v_{j,r}$. For $r \in \mathbb{N}_0$, we let $\vartheta(r) := m\left\lfloor \frac{r}{2m} \right\rfloor$. It can be seen that for a fixed j, the index $\tilde{r}$ of the basis functions $\ell_{j-1,\tilde{r}}$ not vanished at $v_{j,r}$ must be $\tilde{r} = \vartheta(r) + q$, for $q \in \mathbb{Z}_m$. Thus, for all $j \in \mathbb{N}$ and $r \in \mathbb{W}_j$ we have that

$$(\mathcal{P}_{j-1}g)(v_{j,r}) = \sum_{q \in \mathbb{Z}_m} g(v_{j-1,\vartheta(r)+q})\ell_{j-1,\vartheta(r)+q}(v_{j,r}). \tag{3.2}$$

We illustrate in Figure 5 the basis functions $\ell_{3,\tilde{r}}$, constructed from the refinable set $V_0 = \left\{\frac{2\pi}{3}, \frac{4\pi}{3}\right\}$ with $m = 2$, which do not vanish at the point $v_{4,27}$. Since $\vartheta(27) = 12$, the only basis functions that do not vanish at point $v_{4,27}$ are $\ell_{3,12}$ and $\ell_{3,13}$.

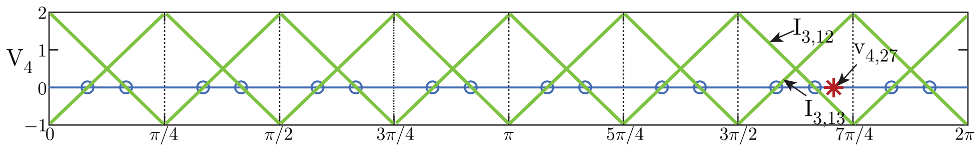


FIGURE 5. Illustrate ϑ

We next present the formula for computing $\mathcal{Q}_j g$ by combining (3.1) and (3.2). To this end, for $q \in \mathbb{Z}_m$ and $\kappa \in \mathbb{Z}_{2m}$, we let $a_{q,\kappa} := \ell_{0,q}(v_{1,\kappa})$. By the scaling and translating property of $\ell_{j,r}$, $j \in \mathbb{N}_0$ and $r \in \mathbb{Z}_{2^j m}$, we obtain for all $j \in \mathbb{N}$ and $r \in \mathbb{W}_j$ that

$$\ell_{j-1,\vartheta(r)+q}(v_{j,r}) = a_{q,r \mod 2m}. \tag{3.3}$$

By introducing the functionals

$$\eta_{j,r}(g) := g(v_{j,r}) - \sum_{q\in\mathbb{Z}_m} a_{q,r \mod 2m}\, g(v_{j-1,\vartheta(r)+q})$$

and substituting them with (3.2) and (3.3) into (3.1), we have that

$$\mathcal{Q}_j g = \sum_{r\in\mathbb{W}_j} \eta_{j,r}(g)\ell_{j,r}, \quad \text{for all} \quad j \in \mathbb{N}. \tag{3.4}$$

Details of the proof for (3.4) can be found in [15]. By letting $\mathbb{W}_0 := \mathbb{Z}_m$ and $\eta_{0,r}(g) := g(v_{0,r})$ for all $r \in \mathbb{Z}_m$, formula (3.4) is extended to the case for $j = 0$.

Formula (3.4) leads to the formula for computing the piecewise polynomial interpolation $\mathcal{S}_N f$ on a sparse grid where f is defined on I^d. From the definition of $\mathcal{Q}_{\mathbf{j}}$ and (3.4), we see for all $\mathbf{j} \in \mathbb{N}_0^d$ that

$$\mathcal{Q}_{\mathbf{j}} f = \sum_{\mathbf{r}\in\mathbb{W}_{\mathbf{j}}} \eta_{\mathbf{j},\mathbf{r}}(f)\ell_{\mathbf{j},\mathbf{r}}. \tag{3.5}$$

Substituting (3.5) into (2.2) leads to the formula

$$\mathcal{S}_N f = \sum_{\mathbf{j}\in\mathbb{S}_{N,d}} \sum_{\mathbf{r}\in\mathbb{W}_{\mathbf{j}}} \eta_{\mathbf{j},\mathbf{r}}(f)\ell_{\mathbf{j},\mathbf{r}}. \tag{3.6}$$

We now turn to developing the formula of $\mathcal{S}_N(\mathbf{f}_N^d)$. We first show the formula for the one-dimensional case with the vector $\mathbf{f}_N^1 := [f_{j,\tau} : j \in \mathbb{Z}_{N+1},\ \tau \in \mathbb{Z}_{j,m}]$. We consider the components $f_{j,\tau}$ of this vector as the functional values sampled from a function f at certain grid points. Note that in this paragraph f is a function of one variable. We next identify the corresponding grid points. Let $j \in \mathbb{N}$ be fixed. For each $\tau \in \mathbb{Z}_{j,m}$, we let $r_{j,\tau} \in \mathbb{W}_j$ be the number such that there are exactly τ elements in $\mathbb{W}_j$ less than $r_{j,\tau}$, and identify $f_{j,\tau}$ with the value of f at the point $v_{j,r_{j,\tau}}$; that is, $f_{j,\tau} = f(v_{j,r_{j,\tau}})$. On the other hand, we need to identify $f(v_{j,r})$ with a component of the vector $\mathbf{f}_N^1$. To this end, we define the mapping ζ for each $j \in \mathbb{N}_0$ and $r \in \mathbb{Z}_{2^j m}$ by $\zeta(j,r) := (j',\tau)$, where $j' \leq j$ and $v_{j,r} = v_{j',r_{j',\tau}}$ and find that for each $j \in \mathbb{Z}_{N+1}$ and $r \in \mathbb{Z}_{2^j m}$, $f(v_{j,r}) = f_{\zeta(j,r)}$. We now re-express $\eta_{j,r}(f)$ as a linear combination of the components of the vector $\mathbf{f}_N^1$. To do this, we set for all $j \in \mathbb{N}$ and $\tau \in \mathbb{Z}_{j,m}$, $b_{j,\tau,q} := \zeta(j-1, \vartheta(r_{j,\tau}) + q)$, for $q \in \mathbb{Z}_m$, and

$b_{j,\tau,m} := \zeta(j, r_{j,\tau})$. We can observe that $f_{b_{j,\tau,q}} = f(v_{j-1,\vartheta(r_{j,\tau})+q})$, $q \in \mathbb{Z}_m$. We also define $d_{j,\tau,q} := -a_{q,r_{j,\tau} \mod 2m}$. With the notation introduced above, we find that

$$\eta_{j,\tau}(f) = f_{b_{j,\tau,m}} + \sum_{q\in\mathbb{Z}_m} d_{j,\tau,q} f_{b_{j,\tau,q}}. \tag{3.7}$$

To unify the notation, we let $b_{0,\tau,0} := (0,\tau)$ and $d_{0,\tau,0} := 1$ for each $\tau \in \mathbb{Z}_{0,m}$, and $b_{j,\tau,m} := (j,\tau)$ and $d_{j,\tau,m} := 1$ for each $j \in \mathbb{N}$ and $\tau \in \mathbb{Z}_{j,m}$. We also let $\mathbb{X}_0 := \mathbb{Z}_1$ and $\mathbb{X}_j := \mathbb{Z}_{m+1}$ for $j \in \mathbb{N}$. For $\mathbf{f}_N^1$, $j \in \mathbb{Z}_{N+1}$ and $\tau \in \mathbb{Z}_{j,m}$, we define

$$\tilde{\eta}_{j,\tau}(\mathbf{f}_N^1) := \sum_{q\in\mathbb{X}_j} d_{j,\tau,q} f_{b_{j,\tau,q}}.$$

From the definition of $\tilde{\eta}_{j,\tau}$ and (3.7), it follows for each $j \in \mathbb{Z}_{N+1}$ and $\tau \in \mathbb{Z}_{j,m}$ that $\eta_{j,\tau}(f) = \tilde{\eta}_{j,\tau}(\mathbf{f}_N^1)$. Thus, the piecewise polynomial $\mathcal{S}_N(\mathbf{f}_N^1)$ that interpolates the data set $\mathbf{f}_N^1$ has the formula

$$\mathcal{S}_N(\mathbf{f}_N^1) = \sum_{j\in\mathbb{Z}_{N+1}} \sum_{\tau\in\mathbb{Z}_{j,m}} \tilde{\eta}_{j,\tau}(\mathbf{f}_N^1) \ell_{j,r_{j,\tau}}. \tag{3.8}$$

We are now ready to present the formula for the function $\mathcal{S}_N(\mathbf{f}_N^d)$ that interpolates the data set $\mathbf{f}_N^d$ for $d > 1$. For each $\mathbf{j} := [j_k : k \in \mathbb{Z}_d] \in \mathbb{N}_0^d$ and $\boldsymbol{\tau} := [\tau_k : k \in \mathbb{Z}_d] \in \mathbb{Z}_{\mathbf{j},m}$, we set $\mathbf{r}_{\mathbf{j},\boldsymbol{\tau}} := [r_{j_k,\tau_k} : k \in \mathbb{Z}_d]$. For $\mathbf{j} \in \mathbb{N}_0^d$, $\boldsymbol{\tau} \in \mathbb{Z}_{\mathbf{j},m}$ and $\mathbf{q} \in \mathbb{X}_{\mathbf{j}}$, we define

$$d_{\mathbf{j},\boldsymbol{\tau},\mathbf{q}} := \prod_{k\in\mathbb{Z}_d} d_{j_k,\tau_k,q_k}.$$

For any two pairs $(j,\tau), (j',\tau') \in \mathbb{N}_0^2$, we define $(j,\tau) \odot (j',\tau') := ([j,j'],[\tau,\tau'])$. For each $\mathbf{j} \in \mathbb{N}_0^d$, $\boldsymbol{\tau} \in \mathbb{Z}_{\mathbf{j},m}$ and $\mathbf{q} \in \mathbb{X}_{\mathbf{j}}$, we let

$$\mathbf{b}_{\mathbf{j},\boldsymbol{\tau},\mathbf{q}} := b_{j_0,\tau_0,q_0} \odot b_{j_1,\tau_1,q_1} \odot \cdots \odot b_{j_{d-1},\tau_{d-1},q_{d-1}}.$$

For $\mathbf{j}, \mathbf{j}' \in \mathbb{N}_0^d$, we say $\mathbf{j}' \le \mathbf{j}$ if $j'_k \le j_k$ for all $k \in \mathbb{Z}_d$. For each $\mathbf{j} \in \mathbb{N}_0^d$, we let $\mathbb{H}_{\mathbf{j},d}$ be the set of all vectors that have the form $\mathbf{h} := [h_{\mathbf{j}',\boldsymbol{\tau}} \in \mathbb{R} : \mathbf{j}' \in \mathbb{N}_0^d, \mathbf{j}' \le \mathbf{j}, \boldsymbol{\tau} \in \mathbb{Z}_{\mathbf{j}',m}]$. For $\mathbf{j} \in \mathbb{N}_0^d$ and $\boldsymbol{\tau} \in \mathbb{Z}_{\mathbf{j},m}$, we define the functionals $\tilde{\eta}_{\mathbf{j},\boldsymbol{\tau}}$ for $\mathbf{h} \in \mathbb{H}_{\mathbf{j},d}$ by

$$\tilde{\eta}_{\mathbf{j},\boldsymbol{\tau}}(\mathbf{h}) := \sum_{\mathbf{q}\in\mathbb{X}_{\mathbf{j}}} d_{\mathbf{j},\boldsymbol{\tau},\mathbf{q}} h_{\mathbf{b}_{\mathbf{j},\boldsymbol{\tau},\mathbf{q}}}.$$

Associated with the given data set $\mathbf{f}_N^d$, for each $\mathbf{j} \in \mathbb{S}_{N,d}$ we define $\mathbf{f}_{\mathbf{j}} := [f_{\mathbf{j}',\boldsymbol{\tau}} : \mathbf{j}' \in \mathbb{N}_0^d,\ \mathbf{j}' \le \mathbf{j}, \boldsymbol{\tau} \in \mathbb{Z}_{\mathbf{j}',m}]$. It is clear that $\mathbf{f}_{\mathbf{j}} \in \mathbb{H}_{\mathbf{j},d}$. We rewrite $\mathcal{S}_N(\mathbf{f}_N^d)$ in the following form

$$\mathcal{S}_N(\mathbf{f}_N^d) = \sum_{\mathbf{j}\in\mathbb{S}_{N,d}} \sum_{\boldsymbol{\tau}\in\mathbb{Z}_{\mathbf{j},m}} \tilde{\eta}_{\mathbf{j},\boldsymbol{\tau}}(\mathbf{f}_{\mathbf{j}}) \ell_{\mathbf{j},\mathbf{r}_{\mathbf{j},\boldsymbol{\tau}}}. \tag{3.9}$$

Formula (3.9) allows us to derive a dimension-reducible sum for $\hat{\mathbf{f}}_N^d(\mathbf{l}) := (\hat{\mathbf{f}}_N^d)_{\mathbf{l}}$. Specifically, for each $j \in \mathbb{N}_0$ and $\tau \in \mathbb{Z}_{j,m}$, we define the function

$$A_{j,\tau}(\omega) := \frac{1}{(2\pi)^{1/2}} \int_I \ell_{j,r_{j,\tau}}(x) e^{i\omega x} dx, \qquad \omega \in \mathbb{R},$$

and for each $\mathbf{l} \in \mathbb{J}_{N,d}$, by using (3.9) we have that

$$\hat{\mathbf{f}}_N^d(\mathbf{l}) = \sum_{\mathbf{j}\in\mathbb{S}_{N,d}} \sum_{\boldsymbol{\tau}\in\mathbb{Z}_{\mathbf{j},m}} \tilde{\eta}_{\mathbf{j},\boldsymbol{\tau}}(\mathbf{f}_{\mathbf{j}}) A_{\mathbf{j},\boldsymbol{\tau}}(\mathbf{l}). \tag{3.10}$$

For efficient computation, we rewrite formula (3.10) in a dimension-reducible sum. For each $j \in \mathbb{N}_0$, we introduce the index set $\mathbb{E}_j := \{r \in \mathbb{N}_0 : \lfloor 2^{j-1} \rfloor m \leq r < 2^j m\}$. For each $N \in \mathbb{N}$, we define

$$\mathbb{F}_{N,d} := \bigcup_{\mathbf{j} \in \mathbb{S}_{N,d}} \mathbb{E}_{\mathbf{j}},$$

and for $\mathbf{j} \in \mathbb{S}_{N,d}$ and $\mathbf{r} \in \mathbb{E}_{\mathbf{j}}$, we identify $\boldsymbol{\tau} = [r_k - \lfloor 2^{j_k - 1} \rfloor m : k \in \mathbb{Z}_d]$ and let $\tilde{\eta}_{\mathbf{r}}(\mathbf{f}_N^d) := \tilde{\eta}_{\mathbf{j},\boldsymbol{\tau}}(\mathbf{f}_{\mathbf{j}})$ and $A_{\mathbf{r}} := A_{\mathbf{j},\boldsymbol{\tau}}$. With the notation introduced above, for each $\mathbf{l} \in \mathbb{J}_{N,d}$ equation (3.10) is rewritten as

$$\hat{\mathrm{f}}_N^d(\mathbf{l}) = \sum_{\mathbf{r} \in \mathbb{F}_{N,d}} \tilde{\eta}_{\mathbf{r}}(\mathbf{f}_N^d) A_{\mathbf{r}}(\mathbf{l}). \tag{3.11}$$

We shall show in section 5 that formula (3.11) is a dimension-reducible sum, which can be recursively computed.

We close this section by discussing the approximation property of $F_{\hat{\mathbf{f}}_N^d}$ to f where the data set $\mathbf{f}_N^d$ is sampled from function $f \in C(I^d)$. For $s \geq 0$ and for $\phi, \psi \in L^2(I^d)$, we define the inner product

$$\langle \phi, \psi \rangle_s := \sum_{\mathbf{l} \in \mathbb{Z}^d} \langle \phi, e_{\mathbf{l}} \rangle \overline{\langle \psi, e_{\mathbf{l}} \rangle} \prod_{k \in \mathbb{Z}_d} (1 + l_k^2)^s. \tag{3.12}$$

All functions $\phi \in L^2(I^d)$ with the property that $\|\phi\|_{H^s_{\text{mix}}(I^d)} := \langle \phi, \phi \rangle_s^{\frac{1}{2}} < \infty$ form a subspace $H^s_{\text{mix}}(I^d)$ of $L^2(I^d)$. It can be verified that $H^s_{\text{mix}}(I^d)$ is a Krobov space endowed with the inner product defined by (3.12). We consider the space of all functions having bounded mixed derivatives up to order m. Suppose that $\Omega \subset \mathbb{R}^d$. For $f \in C^m(\Omega)$ and for $\alpha \in \mathbb{N}_0^d$ with $|\alpha| \leq m$, we define

$$f^{(\alpha)}(\mathbf{x}) := \left(\frac{\partial^{|\alpha|}}{\partial x_0^{\alpha_0} \cdots \partial x_{d-1}^{\alpha_{d-1}}} f \right)(\mathbf{x}), \qquad \mathbf{x} \in \Omega.$$

For an $\alpha \in \mathbb{N}_0^d$, let $|\alpha|_\infty := \max\{\alpha_k : k \in \mathbb{Z}_d\}$. We introduce the space

$$X^m(I^d) := \left\{ f : f^{(\alpha)} \in C(I^d) \text{ and } |\alpha|_\infty \leq m \right\}$$

with the norm

$$\|f\|_{X^m(I^d)} := \max \left\{ \|f^{(\alpha)}\|_\infty : \alpha \in \mathbb{N}_0^d \text{ and } |\alpha|_\infty \leq m \right\}.$$

For $f \in C(I^d)$, $N \in \mathbb{N}$ and $\mathbf{l} \in \mathbb{Z}^d$, we let $\mathbf{Q}_N(f, \mathbf{l})$ denote the approximation of $\langle f, e_{\mathbf{l}} \rangle$ by using the quadrature formula presented in [15] and let $\mathbf{Q}_N(f) := [\mathbf{Q}_N(f, \mathbf{l}) : \mathbf{l} \in \mathbb{J}_{N,d}]$. According to Theorem 3.1 in [15], if $m \geq s + \epsilon$ for an $\epsilon > 0$, then there exists a positive constant c such that for all $f \in H^s_{\text{mix}}(I^d) \cap X^m(I^d)$,

$$\|f - F_{\mathbf{Q}_N(f)}\| \leq c 2^{-sN} (\|f\|_{H^s_{\text{mix}}(I^d)} + \|f\|_{X^m(I^d)}). \tag{3.13}$$

The following theorem gives an estimate of the difference between f and $F_{\hat{\mathbf{f}}_N^d}$.

Theorem 3.1. *Let $s \geq 0$ and $\mathbf{f}_N^d := [\mathrm{f}_{\mathbf{j},\boldsymbol{\tau}} : \mathbf{j} \in \mathbb{S}_{N,d}$ and $\boldsymbol{\tau} \in \mathbb{Z}_{\mathbf{j},m}]$. If $m \geq s + \epsilon$ for $\epsilon > 0$, then there exist a positive constant c and a positive integer N_0 such that for all $f \in H^s_{\text{mix}}(I^d) \cap X^m(I^d)$ with $f(\mathbf{v}_{\mathbf{j},\mathbf{r}_{\mathbf{j},\boldsymbol{\tau}}}) = f_{\mathbf{j},\boldsymbol{\tau}}$, $\mathbf{j} \in \mathbb{S}_{N,d}$, $\boldsymbol{\tau} \in \mathbb{Z}^d_{\mathbf{j},m}$, and for all $N \in \mathbb{N}$ with $N \geq N_0$,*

$$\|f - F_{\hat{\mathbf{f}}_N^d}\| \leq c 2^{-sN} (\|f\|_{H^s_{\text{mix}}(I^d)} + \|f\|_{X^m(I^d)}). \tag{3.14}$$

Proof. From the definitions of $\mathcal{S}_N(\mathbf{f}_N^d)$ and $\mathcal{S}_N(f)$ defined in [15], we have that

$$\mathcal{S}_N(\mathbf{f}_N^d) = \mathcal{S}_N(f). \tag{3.15}$$

Since $(\hat{\mathbf{f}}_N^d)_{\mathbf{l}} = \langle \mathcal{S}_N(\mathbf{f}_N^d), e_{\mathbf{l}} \rangle$, for all $\mathbf{l} \in \mathbb{J}_{N,d}$, from (3.15) and the definition of $\mathbf{Q}_N(f,\mathbf{l})$, we observe that $(\hat{\mathbf{f}}_N^d)_{\mathbf{l}} = \mathbf{Q}_N(f,\mathbf{l})$. Thus, we have that

$$F_{\hat{\mathbf{f}}_N^d} = F_{\mathbf{Q}_N(f)}. \tag{3.16}$$

Substituting (3.16) into (3.13) yields estimate (3.14). □

4. A fast evaluation scheme for dimension-reducible sums

Computing the discrete Fourier transform based on (3.11) and the discrete backward Fourier transform based on (2.5) can be formulated as the problem of evaluating dimension-reducible sums. We introduce in this section a fast algorithm for evaluating such sums. We first define the dimension-reducible set. We then rewrite a d-dimensional dimension-reducible sum as the summation of the tensor products of the $(d-1)$-dimensional dimension-reducible sums with a one-dimensional formula. In this way, evaluating the d-dimensional dimension-reducible sum is reduced to computing the $(d-1)$-dimensional dimension-reducible sums and the one-dimensional formula. Moreover, due to the nestedness of the dimension-reducible sets, the values of the $(d-1)$-dimensional dimension-reducible sums and the one-dimensional formula can be reused. This leads to a fast algorithm for evaluating dimension-reducible sums.

We now define the dimension-reducible set. A mapping Γ from $\mathbb{N}_0^2$ to $\mathbb{N}_0$ is called a double monotonic mapping if for all $j, j' \in \mathbb{N}_0$ $\Gamma(j,j') \geq \Gamma(j,j'+1)$ and $\Gamma(j,j') \leq \Gamma(j+1,j')$. A set $\mathbb{K}_{N,d} \subset \mathbb{R}^d$, for some $N \in \mathbb{N}$, is said to be a dimension-reducible set generated from $\mathbb{T}_j$, $j \in \mathbb{Z}_{N+1}$ and Γ if there exist a sequence of disjoint finite sets $\mathbb{T}_j \subset \mathbb{R}$, $j \in \mathbb{Z}_{N+1}$, and a double monotonic mapping $\Gamma : \mathbb{N}_0^2 \to \mathbb{N}_0$ such that

$$\mathbb{K}_{N,1} = \bigcup_{j \in \mathbb{Z}_{N+1}} \mathbb{T}_j, \quad \mathbb{K}_{N,k+1} = \bigcup_{j \in \mathbb{Z}_{N+1}} \mathbb{T}_j \otimes \mathbb{K}_{\Gamma(N,j),k}, \quad \text{for all} \quad k \in \mathbb{Z}_d \setminus \{0\}. \tag{4.1}$$

We shall prove in sections 5 and 6 that the sets $\mathbb{J}_{N,d}$, $\mathbb{F}_{N,d}$ and $\widetilde{S}_{N,d}$ are dimension-reducible with $\Gamma(N,j) = N - j$ and appropriate sets $\mathbb{T}_j$.

We next present the definition of the dimension-reducible sum that contains (3.11) and (2.5) as special examples. For given sets $\mathbb{L}, \mathbb{U} \subset \mathbb{Z}^d$ and a given vector $\boldsymbol{\xi} := [c_{\mathbf{r}} : \mathbf{r} \in \mathbb{L}]$ of real numbers $c_{\mathbf{r}}$, we consider the evaluation of the sum

$$\mathcal{E}_{\boldsymbol{\xi}} := \sum_{\mathbf{r} \in \mathbb{L}} c_{\mathbf{r}} \varphi_{\mathbf{r}}, \quad \text{on} \quad \mathbb{U},$$

where $\varphi_{\mathbf{r}}$, $\mathbf{r} \in \mathbb{L}$, are tensor products of given univariate functions φ_r, $r \in \mathbb{Z}$. We call $\mathbb{L}$ and $\mathbb{U}$, respectively, the source set and the target set of the sum $\mathcal{E}_{\boldsymbol{\xi}}$. If both $\mathbb{L}$ and $\mathbb{U}$ are dimension-reducible sets, we call the sum $\mathcal{E}_{\boldsymbol{\xi}}$ a *dimension-reducible sum.* More specifically, for fixed $d, N \in \mathbb{N}$, given dimension-reducible sets $\mathbb{L}_{N,d}$ and $\mathbb{U}_{N,d}$, a vector $\boldsymbol{\xi}_N^d := [\xi_{\mathbf{r}} : \mathbf{r} \in \mathbb{L}_{N,d}]$ of real numbers $\xi_{\mathbf{r}}$ and tensor product functions $\varphi_{\mathbf{r}}$, we wish to compute the values of the dimension-reducible sum

$$\mathcal{E}_{\boldsymbol{\xi}_N^d}(\boldsymbol{\nu}) = \sum_{\mathbf{r} \in \mathbb{L}_{N,d}} \xi_{\mathbf{r}} \varphi_{\mathbf{r}}(\boldsymbol{\nu}), \qquad \boldsymbol{\nu} \in \mathbb{U}_{N,d}. \tag{4.2}$$

This formulation covers many examples of sums of practical importance. The discrete Fourier transform and inverse Fourier transform on sparse grids defined in [14] are dimension-reducible sums. The linear combination of B-splines on sparse grids considered in [17] is also a sum having the form (4.2). We shall show later that the sums (3.11) and (2.5) fall into the same format. The main purpose of this section is to design a fast algorithm for computing the values of a dimension-reducible sum having the form (4.2).

We rewrite the dimension-reducible sum (4.2) according to the dimension-reducibility of the source set. To this end, we suppose that the set $\mathbb{L}_{N,d}$ is generated from disjoint finite sets $\mathbb{Y}_j \subset \mathbb{R}$, $j \in \mathbb{Z}_{N+1}$, and a double monotonic mapping Γ. By taking advantage of its dimension-reducibility, we re-express the source index set $\mathbb{L}_{N,d}$ as

$$\mathbb{L}_{N,d} = \bigcup_{j\in\mathbb{Z}_{N+1}} \{[r,\mathbf{r}] \in \mathbb{Z}^d : r \in \mathbb{Y}_j \text{ and } \mathbf{r} \in \mathbb{L}_{\Gamma(N,j),d-1}\}.$$

Thus, for $\mathbf{r} \in \mathbb{L}_{N,d}$, there exists $j \in \mathbb{Z}_{N+1}$ such that $\mathbf{r} = [r, \mathbf{r}']$, where $r \in \mathbb{Y}_j$ and $\mathbf{r}' \in \mathbb{L}_{\Gamma(N,j),d-1}$. Correspondingly, for each $j \in \mathbb{Z}_{N+1}$ and $r \in \mathbb{Y}_j$, the given vector $\boldsymbol{\xi}_N^d$ has the sub-vector $\boldsymbol{\xi}_{N,j,r}^d := [\xi_{[r,\mathbf{r}']} : \mathbf{r}' \in \mathbb{L}_{\Gamma(N,j),d-1}]$, which together with the source set $\mathbb{L}_{\Gamma(N,j),d-1}$ defines a $(d-1)$-dimensional sum $\mathcal{E}_{\boldsymbol{\xi}_{N,j,r}^d}$; that is,

$$\mathcal{E}_{\boldsymbol{\xi}_{N,j,r}^d} := \sum_{\mathbf{r}'\in\mathbb{L}_{\Gamma(N,j),d-1}} \xi_{[r,\mathbf{r}']}\varphi_{\mathbf{r}'}. \tag{4.3}$$

For each $j \in \mathbb{Z}_{N+1}$ and $\mathbf{r} \in \mathbb{L}_{\Gamma(N,j),d-1}$, we define the one-dimensional sum by

$$\widetilde{\mathcal{E}}_{\boldsymbol{\xi}_N^d,j,\mathbf{r}} := \sum_{r\in\mathbb{Y}_j} \xi_{[r,\mathbf{r}]}\varphi_r. \tag{4.4}$$

The above summations allow us to rewrite the d-dimensional dimension-reducible sum $\mathcal{E}_{\boldsymbol{\xi}_N^d}$ as the sum of the following two functions defined for the given vector $\boldsymbol{\xi}_N^d$ and $j \in \mathbb{Z}_{N+1}$ by

$$\mathcal{V}_{\boldsymbol{\xi}_N^d,j} := \sum_{j'\in\mathbb{Z}_{j+1}} \sum_{r\in\mathbb{Y}_{j'}} \varphi_r \otimes \mathcal{E}_{\boldsymbol{\xi}_{N,j',r}^d} \tag{4.5}$$

and

$$\widetilde{\mathcal{V}}_{\boldsymbol{\xi}_N^d,j} := \sum_{j'\in\mathbb{Z}_{N+1}\setminus\mathbb{Z}_{j+1}} \sum_{\mathbf{r}\in\mathbb{L}_{\Gamma(N,j'),d-1}} \widetilde{\mathcal{E}}_{\boldsymbol{\xi}_N^d,j',\mathbf{r}} \otimes \varphi_{\mathbf{r}}, \quad j \in \mathbb{Z}_N, \quad \widetilde{\mathcal{V}}_{\boldsymbol{\xi}_N^d,N} := 0. \tag{4.6}$$

This result is presented in the next lemma.

Lemma 4.1. *If $\mathbb{L}_{N,d}$ is a dimension-reducible set generated from the disjoint finite sets $\mathbb{Y}_j$, $j \in \mathbb{N}_{N+1}$ and the double monotonic mapping Γ, then for any fixed $j_0 \in \mathbb{N}_{N+1}$, we have*

$$\mathcal{E}_{\boldsymbol{\xi}_N^d} = \mathcal{V}_{\boldsymbol{\xi}_N^d,j_0} + \widetilde{\mathcal{V}}_{\boldsymbol{\xi}_N^d,j_0}. \tag{4.7}$$

Proof. By the hypothesis of this lemma, from (4.2) we have that

$$\mathcal{E}_{\boldsymbol{\xi}_N^d} = \sum_{j\in\mathbb{Z}_{N+1}} \sum_{r\in\mathbb{Y}_j} \sum_{\mathbf{r}'\in\mathbb{L}_{\Gamma(N,j),d-1}} \xi_{[r,\mathbf{r}']}\varphi_{[r,\mathbf{r}']}. \tag{4.8}$$

Noting that $\varphi_{[r,\mathbf{r}']} = \varphi_r \otimes \varphi_{\mathbf{r}'}$, by exchanging the order of summations in (4.8) we have for any fixed $j_0 \in \mathbb{N}_{N+1}$ that

$$\mathcal{E}_{\boldsymbol{\xi}_N^d} = \sum_{j\in\mathbb{Z}_{j_0+1}} \sum_{r\in\mathbb{Y}_j} \varphi_r \otimes \left(\sum_{\mathbf{r}\in\mathbb{L}_{\Gamma(N,j),d-1}} \xi_{[r,\mathbf{r}]}\varphi_{\mathbf{r}} \right)$$
$$+ \sum_{j\in\mathbb{Z}_{N+1}\setminus\mathbb{Z}_{j_0+1}} \sum_{\mathbf{r}\in\mathbb{L}_{\Gamma(N,j),d-1}} \left(\sum_{r\in\mathbb{Y}_j} \xi_{[r,\mathbf{r}]}\varphi_r \right) \otimes \varphi_{\mathbf{r}}.$$

Substituting (4.3) and (4.4) into the right-hand side of the equation above, and employing the definition of $\mathcal{V}_{\boldsymbol{\xi}_N^d,j}$ and $\widetilde{\mathcal{V}}_{\boldsymbol{\xi}_N^d,j}$, we obtain the desired formula (4.7). □

Lemma 4.1 and the special structure of the functions $\mathcal{V}_{\boldsymbol{\xi}_N^d,j}$ and $\widetilde{\mathcal{V}}_{\boldsymbol{\xi}_N^d,j}$ allow us to compute the sum $\mathcal{E}_{\boldsymbol{\xi}_N^d}$ recursively in dimensions. To design an efficient algorithm for the computation, we also need to make use of the dimension-reducibility of the target set $\mathbb{U}_{N,d}$ which is assumed to be generated from disjoint finite sets $\mathbb{V}_j \subset \mathbb{R}$, $j \in \mathbb{Z}_{N+1}$, and a double monotonic mapping Γ'. By Lemma 4.1 and the definition of the dimension-reducible set we see that computing (4.2) may be reduced to evaluating $\mathcal{V}_{\boldsymbol{\xi}_N^d,j}$ and $\widetilde{\mathcal{V}}_{\boldsymbol{\xi}_N^d,j}$ on $\mathbb{V}_j \otimes \mathbb{U}_{\Gamma'(N,j),d-1}$ for each $j \in \mathbb{Z}_{N+1}$. We next present formulas for such computation. In preparation, we first show that dimension-reducible sets $\mathbb{K}_{N,d}$ are a nested sequence in $N \in \mathbb{N}$.

Lemma 4.2. *If $\mathbb{K}_{N,d} \subset \mathbb{R}^d$, for $N \in \mathbb{N}$, are dimension-reducible sets, then $\mathbb{K}_{N,d} \subset \mathbb{K}_{N+1,d}$.*

Proof. This result may be proved by induction on d in conjunction with the definition of the dimension-reducible set. □

It follows from Lemma 4.2 that the sets $\mathbb{K}_{\Gamma(N,j),d-1}$, $j \in \mathbb{Z}_{N+1}$, are nested; that is,

$$\mathbb{K}_{\Gamma(N,N),d-1} \subset \mathbb{K}_{\Gamma(N,N-1),d-1} \subset \cdots \subset \mathbb{K}_{\Gamma(N,0),d-1}.$$

By employing the nestedness of the set sequence $\mathbb{U}_{N,d}$ which is ensured by Lemma 4.2, we next present a formula for evaluating $\mathcal{V}_{\boldsymbol{\xi}_N^d,j}$ on $\mathbb{V}_j \otimes \mathbb{U}_{\Gamma'(N,j),d-1}$. For notational convenience, for all $j \in \mathbb{Z}_{N+1}$, $r \in \mathbb{Y}_j$ and $\boldsymbol{\nu} \in \mathbb{U}_{\Gamma'(N,j),d-1}$, we let $\mathrm{E}_{\boldsymbol{\xi}_{N,j,r}^d,\boldsymbol{\nu}} := \mathcal{E}_{\boldsymbol{\xi}_{N,j,r}^d}(\boldsymbol{\nu})$.

Lemma 4.3. *If $\mathbb{L}_{N,d}$ and $\mathbb{U}_{N,d}$ are dimension-reducible sets generated respectively from disjoint finite sets $\mathbb{Y}_j$, $j \in \mathbb{Z}_{N+1}$, and a double monotonic mapping Γ, and $\mathbb{V}_j$, $j \in \mathbb{Z}_{N+1}$, and a double monotonic mapping Γ', then for each fixed $j \in \mathbb{Z}_{N+1}$ and for $\boldsymbol{\nu} := [\nu, \boldsymbol{\nu}'] \in \mathbb{V}_j \otimes \mathbb{U}_{\Gamma'(N,j),d-1}$,*

$$\mathcal{V}_{\boldsymbol{\xi}_N^d,j}(\boldsymbol{\nu}) = \sum_{j'\in\mathbb{Z}_{j+1}} \sum_{r\in\mathbb{Y}_{j'}} \mathrm{E}_{\boldsymbol{\xi}_{N,j'}^d}, r, \boldsymbol{\nu}'\varphi_r(\nu). \tag{4.9}$$

Proof. Since $\mathbb{L}_{N,d}$ is a dimension-reducible set generated from $\mathbb{Y}_{j'}$, $j' \in \mathbb{Z}_{N+1}$, we know for all $j' \in \mathbb{Z}_{N+1}$, $r \in \mathbb{Y}_{j'}$ and given $\boldsymbol{\xi}_N^d$ that $\boldsymbol{\xi}_{N,j',r}^d$ is well defined and so is the function $\mathcal{E}_{\boldsymbol{\xi}_{N,j',r}^d}$. Since $\mathbb{U}_{N,d}$ is a dimension-reducible set generated from $\mathbb{V}_j$, $j \in \mathbb{Z}_{N+1}$, by using Lemma 4.2, we have that

$$\mathbb{U}_{\Gamma'(N,j),d-1} \subset \mathbb{U}_{\Gamma'(N,j'),d-1}, \quad \text{for all } j',\ j \in \mathbb{Z}_{N+1} \text{ with } j' \le j.$$

For $j \in \mathbb{Z}_{N+1}$ and $\boldsymbol{\nu} := [\nu, \boldsymbol{\nu}'] \in \mathbb{V}_j \otimes \mathbb{U}_{\Gamma'(N,j),d-1}$, because $\boldsymbol{\nu}' \in \mathbb{U}_{\Gamma'(N,j),d-1}$, we know for all $j' \in \mathbb{Z}_{j+1}$ that $\boldsymbol{\nu}' \in \mathbb{U}_{\Gamma'(N,j'),d-1}$. This ensures for all $j' \in \mathbb{Z}_{j+1}$ that $\mathrm{E}_{\boldsymbol{\xi}^d_{N,j'}}, r, \boldsymbol{\nu}'$ is well defined. Thus, from the definitions of $\mathrm{E}_{\boldsymbol{\xi}^d_{N,j'}}, r, \boldsymbol{\nu}'$ and $\mathcal{V}_{\boldsymbol{\xi}^d_N, j'}$, we obtain the desired formula (4.9). □

We present in the next lemma a formula for evaluating $\widetilde{\mathcal{V}}_{\boldsymbol{\xi}^d_N,j}$ on $\mathbb{V}_j \otimes \mathbb{U}_{\Gamma'(N,j),d-1}$, for each $j \in \mathbb{Z}_{N+1}$. For $j \in \mathbb{Z}_{N+1}$, $\mathbf{r} \in \mathbb{L}_{\Gamma(N,j),d-1}$ and $\nu \in \mathbb{U}_{j,1}$, we let

(4.10)
$$\tilde{\mathrm{h}}_{\boldsymbol{\xi}^d_N,j,\mathbf{r},\nu} := \begin{cases} \widetilde{\mathcal{E}}_{\boldsymbol{\xi}^d_N,j+1,\mathbf{r}}(\nu) + \tilde{\mathrm{h}}_{\boldsymbol{\xi}^d_N,j+1,\mathbf{r},\nu}, & j \le N-1 \text{ and } \mathbf{r} \in \mathbb{L}_{\Gamma(N,j+1),d-1}, \\ 0, & j = N \text{ or } \mathbf{r} \in \mathbb{L}_{\Gamma(N,j),d-1} \setminus \mathbb{L}_{\Gamma(N,j+1),d-1}. \end{cases}$$

Lemma 4.4. *If $\mathbb{L}_{N,d}$ and $\mathbb{U}_{N,d}$ are dimension-reducible sets generated respectively from disjoint finite sets $\mathbb{Y}_j$, $j \in \mathbb{Z}_{N+1}$, and a double monotonic mapping Γ, and $\mathbb{V}_j$, $j \in \mathbb{Z}_{N+1}$, and a double monotonic mapping Γ', then for each $j \in \mathbb{Z}_{N+1}$ and for all $\boldsymbol{\nu} := [\nu, \boldsymbol{\nu}'] \in \mathbb{V}_j \otimes \mathbb{U}_{\Gamma'(N,j),d-1}$, we have*

$$\widetilde{\mathcal{V}}_{\boldsymbol{\xi}^d_N,j}(\boldsymbol{\nu}) = \sum_{\mathbf{r} \in \mathbb{L}_{\Gamma(N,j),d-1}} \tilde{\mathrm{h}}_{\boldsymbol{\xi}^d_N,j,\mathbf{r},\nu} \varphi_{\mathbf{r}}(\boldsymbol{\nu}'). \tag{4.11}$$

Proof. We prove a somewhat more general result, that is, (4.11) for all $\boldsymbol{\nu} := [\nu, \boldsymbol{\nu}'] \in \mathbb{V}_j \otimes \mathbb{R}^{d-1}$. This is done by a backward induction on $j \in \mathbb{Z}_{N+1}$. When $j = N$, from the definitions of $\widetilde{\mathcal{V}}_{\boldsymbol{\xi}^d_N,N}$ and $\tilde{\mathrm{h}}_{\boldsymbol{\xi}^d_N,j,\mathbf{r},\nu}$, we know that (4.11) holds. We assume that (4.11) holds when $j = j'+1$, $j' \in \mathbb{Z}_N$ and show that (4.11) holds when $j = j'$.

Let $\boldsymbol{\nu} := [\nu, \boldsymbol{\nu}']$ with $\nu \in \mathbb{V}_{j'}$, $\boldsymbol{\nu}' \in \mathbb{R}^{d-1}$. From the definitions of $\widetilde{\mathcal{V}}_{\boldsymbol{\xi}^d_N,j'}$ and $\widetilde{\mathcal{V}}_{\boldsymbol{\xi}^d_N,j'+1}$ we obtain that

$$\widetilde{\mathcal{V}}_{\boldsymbol{\xi}^d_N,j'}(\boldsymbol{\nu}) = \sum_{\mathbf{r} \in \mathbb{L}_{\Gamma(N,j'+1),d-1}} \widetilde{\mathcal{E}}_{\boldsymbol{\xi}^d_N,j'+1,\mathbf{r}}(\nu) \varphi_{\mathbf{r}}(\boldsymbol{\nu}') + \widetilde{\mathcal{V}}_{\boldsymbol{\xi}^d_N,j'+1}(\boldsymbol{\nu}). \tag{4.12}$$

It follows from (4.1) and $\nu \in \mathbb{U}_{j',1}$ that $\nu \in \mathbb{U}_{j'+1,1}$. This guarantees for all $\mathbf{r} \in \mathbb{L}_{\Gamma(N,j'+1),d-1}$ that $\tilde{\mathrm{h}}_{\boldsymbol{\xi}^d_N,j'+1,\mathbf{r},\nu}$ is well defined. By using the induction hypothesis, (4.12) and the definition of $\tilde{\mathrm{h}}_{\boldsymbol{\xi}^d_N,j',\mathbf{r}}$, we have that

$$\widetilde{\mathcal{V}}_{\boldsymbol{\xi}^d_N,j'}(\boldsymbol{\nu}) = \sum_{\mathbf{r} \in \mathbb{L}_{\Gamma(N,j'+1),d-1}} \tilde{\mathrm{h}}_{\boldsymbol{\xi}^d_N,j',\mathbf{j},\mathbf{r},\nu} \varphi_{\mathbf{r}}(\boldsymbol{\nu}'). \tag{4.13}$$

By using (4.13), $\mathbb{L}_{\Gamma(N,j'+1),d-1} \subset \mathbb{L}_{\Gamma(N,j'),d-1}$ and $\tilde{\mathrm{h}}_{\boldsymbol{\xi}^d_N,j',\mathbf{r},\nu} = 0$ when

$$\mathbf{r} \in \mathbb{L}_{\Gamma(N,j'),d-1} \setminus \mathbb{L}_{\Gamma(N,j'+1),d-1},$$

we conclude that (4.11) holds for the case $j = j'$. By the induction principle, (4.11) holds for all $j \in \mathbb{Z}_N$. □

We next describe a fast algorithm for computing the values of the dimension-reducible sum (4.2) based upon Lemmas 4.1, 4.3 and 4.4.

We first discuss the issue of reusing certain computed intermediate values in the computation. When evaluating $\mathcal{V}_{\boldsymbol{\xi}^d_N,j}$ and $\widetilde{\mathcal{V}}_{\boldsymbol{\xi}^d_N,j}$ on $\mathbb{V}_j \otimes \mathbb{U}_{\Gamma'(N,j),d-1}$, we should avoid redundant arithmetic operations by reusing certain intermediate results. Lemma 4.3 shows that for each fixed $j \in \mathbb{Z}_{N+1}$, evaluating $\mathcal{V}_{\boldsymbol{\xi}^d_N,j}$ on $\mathbb{V}_j \otimes \mathbb{U}_{\Gamma'(N,j),d-1}$ requires the values of $\mathcal{E}_{\boldsymbol{\xi}^d_{N,j'},r}$ on $\mathbb{U}_{\Gamma'(N,j),d-1}$ for all $j' \in \mathbb{Z}_{j+1}$ and $r \in \mathbb{Y}_{j'}$. Lemma 4.2 ensures that $\mathbb{U}_{\Gamma'(N,j),d-1} \subset \mathbb{U}_{\Gamma'(N,j'),d-1}$ for $j' \le j$. Thus, for a fixed j the values of $\mathcal{E}_{\boldsymbol{\xi}^d_{N,j',r}}$ on $\mathbb{U}_{\Gamma'(N,j'),d-1}$ for all j' with $j' \le j$ provide sufficient information for evaluating $\mathcal{V}_{\boldsymbol{\xi}^d_N,j}$ on $\mathbb{V}_j \otimes \mathbb{U}_{\Gamma'(N,j),d-1}$. These values can be reused when we evaluate $\mathcal{V}_{\boldsymbol{\xi}^d_N,\tilde{j}}$ on $\mathbb{V}_{\tilde{j}} \otimes \mathbb{U}_{\Gamma'(N,\tilde{j}),d-1}$ for all $\tilde{j}$ with $j \le \tilde{j}$. On the other hand, from formula (4.10) and Lemma 4.4, we see that for each fixed $j \in \mathbb{Z}_{N+1}$, evaluating $\widetilde{\mathcal{V}}_{\boldsymbol{\xi}^d_N,j}$ on $\mathbb{V}_j \otimes \mathbb{U}_{\Gamma'(N,j),d-1}$ by using formula (4.11) requires the values of $\widetilde{\mathcal{E}}_{\boldsymbol{\xi}^d_N,j',\mathbf{r}}$ on $\mathbb{V}_j$ for all $j' > j$ and $\mathbf{r} \in \mathbb{L}_{\Gamma(N,j'),d-1}$. Because of the inclusion relations $\mathbb{V}_j \subset \mathbb{U}_{j,1} \subset \mathbb{U}_{j',1}$, for $j' > j$, the values of $\widetilde{\mathcal{E}}_{\boldsymbol{\xi}^d_N,j',\mathbf{r}}$ on $\mathbb{U}_{j',1}$ for all $j' > j$ and $\mathbf{r} \in \mathbb{L}_{\Gamma(N,j'),d-1}$ provide sufficient information for evaluating $\widetilde{\mathcal{V}}_{\boldsymbol{\xi}^d_N,j}$ on $\mathbb{V}_j \otimes \mathbb{U}_{\Gamma'(N,j),d-1}$. These values can also be reused when we evaluate $\widetilde{\mathcal{V}}_{\boldsymbol{\xi}^d_N,\tilde{j}}$ on $\mathbb{V}_{\tilde{j}} \otimes \mathbb{U}_{\Gamma'(N,\tilde{j}),d-1}$ for all $\tilde{j}$ with $\tilde{j} < j$.

We need an algorithm for evaluating (4.2) in the case when $d = 1$. Such an algorithm depends on the nature of the functions φ_r, $r \in \mathbb{Y}_j$. For example, when the functions φ_r are trigonometric functions, we may employ the fast Fourier transform in designing the algorithm, and when the functions φ_r are wavelets, we may use fast wavelet transforms in developing the algorithm. Thus, we shall not specify the algorithm for evaluating (4.2) in the case with $d = 1$ until we enter a specific context. Instead, we assume that there exists an algorithm "FE1d($N, \boldsymbol{\xi}^1_N$)" for evaluating (4.2) in the case when $d = 1$, for a given $\boldsymbol{\xi}^1_N := [\xi_r : j \in \mathbb{Z}_{N+1}, r \in \mathbb{Y}_j]$. Specific algorithms "FE1d($N, \boldsymbol{\xi}^1_N$)" for the Fourier transform and inverse Fourier transform will be given in sections 4 and 5, respectively.

It remains to discuss the computation of the values $\tilde{\mathrm{h}}_{\boldsymbol{\xi}^d_N,j,\mathbf{r},\nu}$, for all $j \in \mathbb{Z}_{N+1}$, $\nu \in \mathbb{U}_{j,1}$, $\mathbf{r} \in \mathbb{L}_{\Gamma(N,j),d-1}$. From (4.4), we can also employ algorithm "FE1d($N, \boldsymbol{\xi}^1_j$)" to compute $[\widetilde{\mathcal{E}}_{\boldsymbol{\xi}^d_N,j,\mathbf{r}}(\nu) : \nu \in \mathbb{U}_{j,1}]$ by setting $\boldsymbol{\xi}^1_j := [\xi_r : j' \in \mathbb{Z}_{j+1}, r \in \mathbb{Y}_{j'}]$ where $\xi_r = 0$ if $r \in \mathbb{Y}_{j'}$ with $j' \in \mathbb{Z}_j$ and $\xi_r = \xi_{[r,\mathbf{r}]}$, if $r \in \mathbb{Y}_j$. Then, by employing (4.10), we obtain the values $\tilde{\mathrm{h}}_{\boldsymbol{\xi}^d_N,j,\mathbf{r},\nu}$.

We now present the algorithm "FEd" for evaluating $\mathcal{E}_{\boldsymbol{\xi}^d_N}$ on $\mathbb{U}_{N,d}$. The algorithm is written in a recursive form. That is, during the execution of the algorithm, we are allowed to call the algorithm itself with appropriate parameters. For each $\boldsymbol{\nu} \in \mathbb{U}_{N,d}$ and a given $\boldsymbol{\xi}^d_N$, we define $\mathrm{E}_{\boldsymbol{\xi}^d_N,\boldsymbol{\nu}} := \mathcal{E}_{\boldsymbol{\xi}^d_N}(\boldsymbol{\nu})$ and $\mathbf{E}_{\boldsymbol{\xi}^d_N} := [\mathrm{E}_{\boldsymbol{\xi}^d_N,\boldsymbol{\nu}} : \boldsymbol{\nu} \in \mathbb{U}_{N,d}]$. We define the vector $\mathbf{E}_{\boldsymbol{\xi}^d_{N,j,r}}$ in a similar manner. For each $j \in \mathbb{Z}_{N+1}$ and $\mathbf{r} \in \mathbb{L}_{\Gamma(N,j),d-1}$, we let $\mathbf{Y}_{\boldsymbol{\xi}^d_N,j,\mathbf{r}} := [y_r : r \in \mathbb{Y}_{j'}, j' \in \mathbb{Z}_{j+1}]$, where $y_r := \xi_{[r,\mathbf{r}]}$ for $r \in \mathbb{Y}_j$, and $y_r := 0$ for $r \in \mathbb{Y}_{j'}$ with $j' \ne j$. For each $j \in \mathbb{Z}_{N+1}$, $\boldsymbol{\nu} \in \mathbb{U}_{\Gamma'(N,j),d-1}$ and a given $\boldsymbol{\xi}^d_N$, we also let $\mathbf{C}_{\boldsymbol{\xi}^d_N,j,\boldsymbol{\nu}} := [\mathrm{E}_{\boldsymbol{\xi}^d_{N,j',r},\boldsymbol{\nu}} : j' \in \mathbb{Z}_{j+1}, r \in \mathbb{Y}_{j'}]$. For each $j \in \mathbb{Z}_{N+1}$ and $\boldsymbol{\nu} \in \mathbb{U}_{\Gamma'(N,j),d-1}$, we define

$$\mathbf{V}_{\boldsymbol{\xi}^d_N,j,\boldsymbol{\nu}} := [\mathcal{V}_{\boldsymbol{\xi}^d_N,j}(\boldsymbol{\nu}') : \boldsymbol{\nu}' = [\nu, \boldsymbol{\nu}] \text{ with } \nu \in \mathbb{V}_j].$$

For each $j \in \mathbb{Z}_{N+1}$ and $\nu \in \mathbb{V}_{j,1}$, we define

$$\widetilde{\mathbf{V}}_{\boldsymbol{\xi}^d_N,j,\nu} := [\widetilde{\mathcal{V}}_{\boldsymbol{\xi}^d_N,j}(\boldsymbol{\nu}') : \boldsymbol{\nu}' = [\nu, \boldsymbol{\nu}] \text{ with } \boldsymbol{\nu} \in \mathbb{U}_{\Gamma'(N,j),d-1}],$$
$$\widetilde{\mathbf{H}}_{\boldsymbol{\xi}^d_N,j,\nu} := [\tilde{\mathrm{h}}_{\boldsymbol{\xi}^d_N,j,\mathbf{r},\nu} : \mathbf{r} \in \mathbb{L}_{\Gamma(N,j),d-1}].$$

Algorithm 4.5. FEd(d, N, $\boldsymbol{\xi}^d_N$).
Input: $d \in \mathbb{N}$, $N \in \mathbb{N}$ *and* $\boldsymbol{\xi}^d_N := [\xi_{\mathbf{j},\mathbf{r}} : \mathbf{r} \in \mathbb{L}_{N,d}]$.
Output: $\mathbf{E}_{\boldsymbol{\xi}^d_N}$.

Step 1: *If $d = 1$, then compute $\mathbf{E}_{\boldsymbol{\xi}_N^1}$ by* $\mathrm{FE1d}(N, \boldsymbol{\xi}_N^1)$. *Return $\mathbf{E}_{\boldsymbol{\xi}_N^1}$.*

Step 2: *If $d > 1$, then for $j \in \mathbb{Z}_{N+1}$, $r \in \mathbb{Y}_j$, compute $\mathbf{E}_{\boldsymbol{\xi}^d_{N,j,r}}$ by using* $\mathrm{FEd}(d-1, \Gamma(N,j), (\boldsymbol{\xi}_N^d)_{j,r})$.

Step 3: *For $j \in \mathbb{Z}_{N+1}$, $\boldsymbol{\nu} \in \mathbb{U}_{\Gamma'(N,j),d-1}$, compute $\mathbf{V}_{\boldsymbol{\xi}_N^d,j,\boldsymbol{\nu}}$ by using* $\mathrm{FE1d}(j, \mathbf{C}_{\boldsymbol{\xi}_N^d,j,\boldsymbol{\nu}})$.

Step 4: *For $j \in \mathbb{Z}_{N+1}$, $\mathbf{r} \in \mathbb{L}_{\Gamma(N,j),d-1}$, compute $[\widetilde{\mathcal{E}}_{\boldsymbol{\xi}_N^d,j,\mathbf{r}}(\nu) : \nu \in \mathbb{U}_{j,1}]$ by using* $\mathrm{FE1d}(j, \mathbf{Y}_{\boldsymbol{\xi}_N^d,j,\mathbf{r}})$.

Step 5: *Compute $[\widetilde{\mathrm{h}}_{\boldsymbol{\xi}_N^d,j,\mathbf{r},\nu} : j \in \mathbb{Z}_{N+1}, \nu \in \mathbb{U}_{j,1}, \mathbf{r} \in \mathbb{L}_{\Gamma(N,j),d-1}]$ by using* (4.10).

Step 6: *For $j \in \mathbb{Z}_{N+1}$, $\nu \in \mathbb{V}_j$, compute $\widetilde{\mathbf{V}}_{\boldsymbol{\xi}_N^d,j,\nu}$ by using* $\mathrm{FEd}(d-1, \Gamma(N,j), \widetilde{\mathbf{H}}_{\boldsymbol{\xi}_N^d,j,\nu})$.

Step 7: *Compute $\mathbf{E}_{\boldsymbol{\xi}_N^d}$ according to* (4.7). *Return $\mathbf{E}_{\boldsymbol{\xi}_N^d}$.*

Algorithm 4.5 outputs the values of the dimension-reducible sum (4.2). We now turn to estimating its computational cost. For given $c_1 \in \mathbb{R}$ with $c_1 > 0$, for each $d \geq 2$ we define a sequence c_d by

$$c_d := \frac{2c_{d-1}c_1}{d} + \frac{3c_1^2}{d-1} + c_1. \tag{4.14}$$

Lemma 4.6. *If $c_1 > 0$ is fixed, then there exists a positive constant c such that $c_d < c$, for all $d \in \mathbb{N}$.*

Proof. It suffices to show that there exists a positive constant c such that $c_d < c$, for all $d \in \mathbb{N}$ with $d > 2c_1 + 2$. We prove this by contradiction. Assume to the contrary that for all $M > 0$, there exists $d \in \mathbb{N}$ with $d > 2c_1 + 2$ such that $c_d > M$. In particular, there exists a $d^* \in \mathbb{N}$ with $d^* > 2c_1 + 2$ such that $c_{d^*} > M := 5c_1^2 + c_1$ and $c_{d^*} > c_{d^*-1}$. Otherwise, the sequence $\{c_d : d \in \mathbb{N} \text{ with } d > 2c_1 + 2\}$ is bounded. Noting that $c_{d^*} > c_{d^*-1}$, it follows from (4.14) that

$$(c_{d^*} - c_1) \leq \frac{2c_1c_{d^*}}{d^*} + \frac{3c_1^2}{d^*-1}. \tag{4.15}$$

Since $\frac{1}{d^*} < \frac{1}{d^*-1}$, from (4.15) we have that

$$(d^* - 1 - 2c_1)(c_{d^*} - c_1) \leq 5c_1^2. \tag{4.16}$$

By virtue of (4.16) and the inequality $d^* - 1 - 2c_1 > 1$, we obtain that $c_{d^*} < 5c_1^2 + c_1$, which contradicts the fact that $c_{d^*} > 5c_1^2 + c_1$. This ensures that the sequence c_d, $d \in \mathbb{N}$, is bounded. ☐

For each $N \in \mathbb{N}_0$, we denote by $\mathcal{N}_N^d$ the number of operations used in evaluating all elements of $\mathbf{E}_{\boldsymbol{\xi}_N^d}$ by Algorithm 4.5, for a given $\boldsymbol{\xi}_N^d$. By $\mathcal{C}(\mathbb{A})$ we denote the cardinality of the set $\mathbb{A}$.

Proposition 4.7. *If for all $N \in \mathbb{N}_0$ and $j \in \mathbb{N}_0$, $\Gamma(N,j) = \Gamma'(N,j) = N - j$, and there exists a positive constant $\tilde{c}$ such that for some $m \in \mathbb{N}$, for all $d \in \mathbb{N}$ and $N \in \mathbb{N}_0$, both $\mathcal{C}(\mathbb{L}_{N,d})$ and $\mathcal{C}(\mathbb{U}_{N,d})$ are bounded by $\tilde{c}m^d n(\log n + 1)^{d-1}$, and the number of operations used in Algorithm* $\mathrm{FE1d}(N, \boldsymbol{\xi}_N^1)$ *is bounded by $\tilde{c}mn\log n$, then there exists a positive constant c independent of m or d such that for all $N \in \mathbb{N}_0$,*

$$\mathcal{N}_N^d \leq cm^d n(\log n + d)^d, \tag{4.17}$$

where $n := 2^N$.

Proof. Choose $c_1 = \tilde{c}$ and for $d \geq 2$, define the sequence c_d by (4.14). We first establish that

$$\mathcal{N}_N^d \leq c_d m^d 2^N (N+d)^d, \quad \text{for all } \ N \in \mathbb{N}_0. \tag{4.18}$$

By the definition of c_d and Lemma 4.6, we see that the sequence c_d is bounded by a positive constant c, and thus this proposition follows from (4.18).

We prove (4.18) by induction on d. By the hypothesis of this proposition, we see that (4.18) holds for $d = 1$. Let $d' \geq 2$ and assume that (4.18) holds for all $d \in \mathbb{Z}_{d'} \setminus \{0\}$. We next show that (4.18) holds for $d = d'$. For simplicity, we use $\mathcal{N}^{(i)}$ to denote the number of multiplications used in the ith step of Algorithm 4.5, $i = 1, 2, \ldots, 8$, for $d = d'$.

Since (4.18) holds for $d = d' - 1$, from the description of Step 2 in Algorithm 4.5 we have for all $N \in \mathbb{N}_0$ that

$$\mathcal{N}^{(2)} \leq c_{d'-1} m^{d'-1} \sum_{j \in \mathbb{Z}_{N+1}} \sum_{r \in \mathbb{Y}_j} 2^{N-j}(N - j + d' - 1)^{d'-1}. \tag{4.19}$$

Note that $\mathcal{C}(\mathbb{Y}_j) \leq \tilde{c} m 2^j$ and $\sum_{j \in \mathbb{Z}_{N+1}} (N - j + d' - 1)^{d'-1} \leq \frac{(N+d')^{d'}}{d'}$. Combining these inequalities with (4.19), we observe for all $N \in \mathbb{N}_0$ that

$$\mathcal{N}^{(2)} \leq \frac{c_{d'-1} \tilde{c} m^{d'} 2^N (N+d')^{d'}}{d'}. \tag{4.20}$$

Since (4.18) holds for the case $d = 1$, from the definition of $\mathbf{C}_{\boldsymbol{\xi}_N^{d'}, j, \boldsymbol{\nu}}$ we have for all $N \in \mathbb{N}_0$ that

$$\mathcal{N}^{(3)} \leq \tilde{c} m \sum_{j \in \mathbb{Z}_{N+1}} \sum_{\boldsymbol{\nu} \in \mathbb{U}_{\Gamma'(N,j), d'-1}} j 2^j. \tag{4.21}$$

Using the inequalities $\mathcal{C}(\mathbb{U}_{\Gamma'(N,j),d'-1}) \leq \tilde{c} m^{d'-1} 2^{N-j} (N - j + 1)^{d'-2}$ and $\sum_{j \in \mathbb{Z}_{N+1}} j(N - j + 1)^{d'-2} \leq \frac{(N+1)^{d'}}{d'-1}$ in (4.21), we obtain for all $N \in \mathbb{N}_0$ that

$$\mathcal{N}^{(3)} \leq \frac{\tilde{c}^2 m^{d'} 2^N (N+1)^{d'}}{d' - 1}. \tag{4.22}$$

Note that $\mathbf{C}_{\boldsymbol{\xi}_N^{d'}, j, \boldsymbol{\nu}}$ has the same structure with $\mathbf{Y}_{\boldsymbol{\xi}_N^{d'}, j, \mathbf{r}}$. A similar estimate for $\mathcal{N}^{(3)}$ leads to the result for all $N \in \mathbb{N}_0$,

$$\mathcal{N}^{(4)} \leq \frac{\tilde{c}^2 m^{d'} 2^N (N+1)^{d'}}{d' - 1}. \tag{4.23}$$

For each $j \in \mathbb{Z}_{N+1}$, $\nu \in \mathbb{U}_{j,1}$ and $\mathbf{r} \in \mathbb{L}_{\Gamma(N,j),d'-1}$, computing $\tilde{\mathrm{h}}_{\boldsymbol{\xi}_N^{d'}, j, \mathbf{r}, \nu}$ by using (4.10) requires an operation of addition. Thus, by using $\mathcal{C}(\mathbb{L}_{N,d}) \leq \tilde{c} m^d n (\log n + 1)^{d-1}$ we have for all $N \in \mathbb{N}_0$ that

$$\mathcal{N}^{(5)} \leq \tilde{c} m^{d'-1} \sum_{j \in \mathbb{Z}_{N+1}} \sum_{\nu \in \mathbb{U}_{j,1}} 2^{N-j} (N - j + 1)^{d'-2}. \tag{4.24}$$

Since $\mathcal{C}(\mathbb{U}_{j,1}) \leq \tilde{c} m 2^j$ for all $j \in \mathbb{N}_0$, from (4.24) we have for all $N \in \mathbb{N}_0$ that

$$\mathcal{N}^{(5)} \leq \frac{\tilde{c}^2 m^{d'} 2^N (N+2)^{d'-1}}{d' - 1}. \tag{4.25}$$

By replacing $\mathbb{Y}_j$ and $\mathcal{N}^{(2)}$, respectively, by $\mathbb{V}_j$ and $\mathcal{N}^{(6)}$ in the estimate for $\mathcal{N}^{(2)}$, we obtain the estimate

$$\mathcal{N}^{(6)} \leq \frac{c_{d'-1}\tilde{c}m^{d'}2^N(N+d')^{d'}}{d'}. \tag{4.26}$$

For each $\boldsymbol{\nu} \in \mathbb{U}_{N,d'}$, computing $\mathrm{E}_{\boldsymbol{\xi}_N^{d'},\boldsymbol{\nu}}$ using (4.7) requires an operation of addition. Thus, from hypothesis $\mathcal{C}(\mathbb{U}_{N,d}) \leq \tilde{c}m^d n(\log n + 1)^{d-1}$, we have for all $N \in \mathbb{N}_0$ that

$$\mathcal{N}^{(7)} \leq \tilde{c}m^{d'}2^N(N+1)^{d'-1}. \tag{4.27}$$

By summing up (4.20), (4.22), (4.23), (4.25), (4.26) and (4.27), we obtain for all $N \in \mathbb{N}_0$ that

$$\begin{aligned}\mathcal{N}_N^{d'} \leq m^{d'}2^N(N+d')^{d'}\Bigg(&\frac{2c_{d'-1}\tilde{c}}{d'} + \frac{2\tilde{c}^2(N+1)^{d'}}{(d'-1)(N+d')^{d'}}\\ &+\frac{\tilde{c}^2(N+2)^{d'-1}}{(d'-1)(N+d')^{d'}} + \frac{\tilde{c}(N+1)^{d'-1}}{(N+d')^{d'}}\Bigg).\end{aligned} \tag{4.28}$$

Noting that $\frac{(N+1)^{d'-1}}{(N+d')^{d'}} < \frac{(N+2)^{d'-1}}{(N+d')^{d'}} < \frac{(N+1)^{d'}}{(N+d')^{d'}} < 1$, from (4.28) and the definition of $c_{d'}$, we obtain estimate (4.18) for $d = d'$. □

In the remaining part of this section, we estimate the memory space used in Algorithm 4.5. The memory space used in Algorithm 4.5 depends on the selected program design language and the methods for implementing the algorithm. Thus, to estimate it, due to the recursive nature of the algorithm we require that the selected program design language supports recursive programs and pointers of floating-point values. With these requirements, when a vector is called by the program, we send only the address of the memory space used to store the vector to the program. It is well known that program design languages C and C++ meet these requirements.

We also suggest that the following memory management rules be used when implementing Algorithm 4.5. The principle in designing the memory management rules is to release idle memory spaces. Specifically, when data stored in some memory space will not be used in the rest of execution of the algorithm, the memory space will be released (which means that Algorithm 4.5 will no longer use the memory space).

Rule 1: When executing Step 2 or 6 of Algorithm 4.5 with d dimensions, we reuse the memory space which was used in executing “FEd” with $d-1$ dimensions. When executing Steps 3 or 4, we reuse the memory space which was used in executing “FE1d”.

Rule 2: After obtaining the output of Steps 2 and 4, we release the memory space which was used in these two steps except the memory space for storing the input and output of Steps 2 and 4.

Rule 3: For each $i \in \{3, 6, 7\}$, reuse the memory space which was used for storing the input data of Step 1 to store the output data of this step.

Rule 4: For each $i \in \{3, 5, 6, 7\}$, after obtaining the output of Step 1, we release the memory space which was used by Step 1 except the memory space which is used for storing the output of Step 1.

Following Rule 2, we do not release the memory space which was used for storing the input data of Steps 2 and 4 in Algorithm 4.5. This is because the input data of Steps 2 and 4 are the input of Algorithm 4.5 which has to be saved during the execution of the entire algorithm.

By $\mathcal{M}_N^d$ we denote the size of the memory space required by Algorithm 4.5 which includes the size of memory space for storing the input and output data. For fixed $d \in \mathbb{N}$ and each $i \in \{2,3,4,5,6,7\}$, let $\mathcal{M}_i$ denote the size of the memory space required, by Algorithm 4.5 with d dimensions, for executing Step 1 and storing the data which are required by the remaining steps of the algorithm. Then, the memory space $\mathcal{M}_N^d$ required by Algorithm 4.5 can be measured by

$$\mathcal{M}_N^d = \max\{\mathcal{M}_i : i = 2,3,4,5,6,7\}.$$

For a given data set $\mathbb{A}$, we use $\widetilde{\mathcal{M}}(\mathbb{A})$ to denote the size of the memory space used to store data set $\mathbb{A}$. Let $\widetilde{\mathcal{M}}_N^d := \max\{\widetilde{\mathcal{M}}(\boldsymbol{\xi}_N^d), \widetilde{\mathcal{M}}(\mathbf{E}_{\boldsymbol{\xi}_N^d})\}$ and $\widehat{\mathcal{M}}_N^d := \sum_{k\in\mathbb{Z}_d} \widetilde{\mathcal{M}}_N^{k+1}$.

Proposition 4.8. *Suppose that Algorithm* 4.5 *is implemented following Rules 1, 2, 3 and 4. If*

$$\mathcal{M}_N^1 \le 3\widetilde{\mathcal{M}}_N^1, \quad \textit{for all} \quad N \in \mathbb{N}_0, \tag{4.29}$$

and one of the following two conditions is satisfied,

(i) for all $j \in \mathbb{N}_0$, $\mathcal{C}(\mathbb{V}_j) \le \mathcal{C}(\mathbb{Y}_j)$, and for all j, $N \in \mathbb{N}_0$, $\Gamma'(N,j) \le \Gamma(N,j)$,
(ii) for all $j \in \mathbb{N}_0$, $\mathcal{C}(\mathbb{V}_j) \ge \mathcal{C}(\mathbb{Y}_j)$, and for all j, $N \in \mathbb{N}_0$, $\Gamma'(N,j) \ge \Gamma(N,j)$,

then

$$\mathcal{M}_N^d \le 4\widehat{\mathcal{M}}_N^d, \quad \textit{for all} \quad d \in \mathbb{N},\ N \in \mathbb{N}_0. \tag{4.30}$$

Proof. We assume that **(i)** is satisfied. In this case, it can be shown that for all $d \in \mathbb{N}$ and $N \in \mathbb{N}_0$, $\mathcal{C}(\mathbb{U}_{N,d}) \le \mathcal{C}(\mathbb{L}_{N,d})$. We prove estimate (4.30) by induction on d. Hypothesis (4.29) ensures that (4.30) holds for $d = 1$. Let $d' \in \mathbb{N}$. We assume that (4.30) holds for $d = d'$ and show that (4.30) holds for $d = d' + 1$. For $i \in \{2,4,\ldots,7\}$, let $\widetilde{\mathcal{M}}_i$ denote the size of memory space required for executing Step 1 of Algorithm 4.5 for $d = d' + 1$. Since (4.30) holds for $d = d'$, it follows from the description of Step 2 and Rule 1 that

$$\widetilde{\mathcal{M}}_2 \le \sum_{j\in\mathbb{Z}_{N+1}} \sum_{r\in\mathbb{Y}_j} \widetilde{\mathcal{M}}_{\Gamma(N,j)}^{d'} + 4\widehat{\mathcal{M}}_N^{d'} + \widetilde{\mathcal{M}}_N^{d'+1}. \tag{4.31}$$

Because $\mathcal{C}(\mathbb{U}_{j,d'}) \le \mathcal{C}(\mathbb{L}_{j,d'})$ for all $j \in \mathbb{N}_0$, it holds that $\widetilde{\mathcal{M}}_{\Gamma(N,j)}^{d'} = \widetilde{\mathcal{M}}((\boldsymbol{\xi}_N^{d'+1})_{j,r})$ for all $j \in \mathbb{Z}_{N+1}$ and $r \in \mathbb{Y}_j$. Accordingly, by using

$$\mathcal{M}_2 = \widetilde{\mathcal{M}}_2 \quad \text{and} \quad \sum_{j\in\mathbb{Z}_{N+1}} \sum_{r\in\mathbb{Y}_j} \widetilde{\mathcal{M}}((\boldsymbol{\xi}_N^{d'+1})_{j,r}) = \widetilde{\mathcal{M}}(\boldsymbol{\xi}_N^{d'+1}),$$

from (4.31) we know that

$$\mathcal{M}_2 \le 2\widetilde{\mathcal{M}}_N^{d'+1} + 4\widehat{\mathcal{M}}_N^{d'}. \tag{4.32}$$

By Rule 2, before executing Step 3, Algorithm 4.5 occupies a memory space of size $2\widetilde{\mathcal{M}}_N^{d'+1}$ for storing the input and output data of Step 2. On the other hand, by Rule 3, the memory space used for storing the input of Step 3 is reused for storing the output of Step 3. Thus, from the description of Step 3 and hypothesis (4.29), by applying Rules 1 and 3 we have that

$$\widetilde{\mathcal{M}}_3 \le \widetilde{\mathcal{M}}_N^{d'+1} + 3\widetilde{\mathcal{M}}_N^1. \tag{4.33}$$

Since the output of Step 2 is the input of Step 3 and a memory space of size $\widetilde{\mathcal{M}}_N^{d'+1}$ is needed for storing the input of Step 2 when executing Step 3, we have that $\mathcal{M}_3 \leq \widetilde{\mathcal{M}}_3 + \widetilde{\mathcal{M}}_N^{d'+1}$. From (4.33) we obtain that $\mathcal{M}_3 \leq 2\widetilde{\mathcal{M}}_N^{d'+1} + 3\widetilde{\mathcal{M}}_N^1$. Note that storing the input data of Step 4 does not require additional memory space, since the input data of Step 4 is the same as that of Step 2. Thus, by Rule 4 before executing Step 4, we need a memory space of size $2\widetilde{\mathcal{M}}_N^{d'+1}$ to store the output data of Step 3 and the input data of Step 2. By applying Rules 1 and 2 we have that $\widetilde{\mathcal{M}}_4 \leq 2\widetilde{\mathcal{M}}_N^{d'+1} + 3\widetilde{\mathcal{M}}_N^1$, and $\mathcal{M}_4 \leq \widetilde{\mathcal{M}}_4 + \widetilde{\mathcal{M}}_N^{d'+1} \leq 3\widetilde{\mathcal{M}}_N^{d'+1} + 3\widetilde{\mathcal{M}}_N^1$. Before executing Step 5, we need a memory space of size $3\widetilde{\mathcal{M}}_N^{d'+1}$ for storing the output data of Steps 3, 4 and the input data of Step 2. From (4.10) and the description of Step 5, we see that $\widetilde{\mathcal{M}}_5 \leq 2\widetilde{\mathcal{M}}_N^{d'+1}$. Noticing that the output data of Step 4 is the input of Step 5, we have that $\mathcal{M}_5 \leq \widetilde{\mathcal{M}}_5 + 2\widetilde{\mathcal{M}}_N^{d'+1}$. Therefore, we know that $\mathcal{M}_5 \leq 4\widetilde{\mathcal{M}}_N^{d'+1}$. By Rule 4, before executing Step 6, we need a memory space of size $3\widetilde{\mathcal{M}}_N^{d'+1}$ to store the output data of Steps 3, 5 and the input data of Step 2. By using Rule 3 and a similar discussion for Step 2, we have that $\widetilde{\mathcal{M}}_6 \leq \widetilde{\mathcal{M}}_N^{d'+1} + 4\widehat{\mathcal{M}}_N^{d'}$. Noting that the output data of Step 5 is the input of Step 6, we have that $\mathcal{M}_6 \leq \widetilde{\mathcal{M}}_6 + 2\widetilde{\mathcal{M}}_N^{d'+1}$. Then, it holds that $\mathcal{M}_6 \leq 3\widetilde{\mathcal{M}}_N^{d'+1} + 4\widehat{\mathcal{M}}_N^{d'}$. Due to Rule 4, before executing Step 7, a memory space of size $3\widetilde{\mathcal{M}}_N^{d'+1}$ is used for storing the output data of Steps 3, 6 and the input data of Step 2. From (4.7) and Rule 3, we see that $\widetilde{\mathcal{M}}_7 \leq 2\widetilde{\mathcal{M}}_N^{d'+1}$. Note the size of the memory space used for storing the output data of Steps 3 and 6 is counted in $\widetilde{\mathcal{M}}_7$. Thus, we have that $\mathcal{M}_7 \leq \widetilde{\mathcal{M}}_7 + \widetilde{\mathcal{M}}_N^{d'+1}$. This yields that $\mathcal{M}_7 \leq 3\widetilde{\mathcal{M}}_N^{d'+1}$. By the definition of $\mathcal{M}_N^d$, from the estimates of $\mathcal{M}_i$, $i \in \{2,3,4,5,6,7\}$, we conclude that (4.30) holds for $d = d'+1$. By the induction principle, we obtain (4.30) for all $d \in \mathbb{N}$ and $N \in \mathbb{N}_0$.

The case **(ii)** can be similarly handled. □

The next result follows immediately from Proposition 4.8.

Corollary 4.9. *If there exists a positive constant* $\tilde{c}$ *such that for all* $d,\ N \in \mathbb{N}$*, both* $\mathcal{C}(\mathbb{L}_{N,d})$ *and* $\mathcal{C}(\mathbb{U}_{N,d})$ *are bounded by* $\tilde{c}m^d n(\log n + 1)^{d-1}$ *for some positive integer* m*, and the hypotheses of Proposition* 4.8 *hold, then there exists a positive constant* c *such that for all* $m,\ d,\ N \in \mathbb{N}$*,*

$$\mathcal{M}_N^d \leq cn\frac{m^d(\log n + 1)^d - 1}{\log n}, \tag{4.34}$$

where $n := 2^N$*.*

Proof. Note that there exists a positive constant $\bar{c}$ such that for all $d,\ N \in \mathbb{N}$,

$$\widetilde{\mathcal{M}}_N^d \leq \bar{c}m^d n(\log n + 1)^{d-1}.$$

Thus, from the definition of $\widehat{\mathcal{M}}_N^d$ and Proposition 4.8 we conclude the validity of this corollary. □

5. Fast discrete Fourier transform on sparse grids

We present in this section a fast algorithm for computing the discrete Fourier transform of $\mathbf{f}_N^d$ at $\mathbb{J}_{N,d}$ by employing Algorithm 4.5.

We first show that formula (3.11) is a dimension-reducible sum. To this end, we show that $\hat{\mathbf{f}}_N^d$ can be reformulated as a sum in the form of (4.2). We also show that

$\mathbb{J}_{N,d}$ is a dimension-reducible set generated from disjoint finite sets $\mathbb{I}_j$, $j \in \mathbb{Z}_{N+1}$, and $\Gamma'(N,j) = N - j$, for all $j \in \mathbb{Z}_{N+1}$, and $\mathbb{F}_{N,d}$ is a dimension-reducible set generated from $\mathbb{E}_j$, $j \in \mathbb{Z}_{N+1}$, and $\Gamma(N,j) = N - j$ for all $j \in \mathbb{Z}_{N+1}$. We then specify $\mathbb{L}_{N,d}$ with $\mathbb{F}_{N,d}$, $\mathbb{U}_{N,d}$ with $\mathbb{J}_{N,d}$, $\mathbb{Y}_j$ with $\mathbb{E}_j$, $\mathbb{V}_j$ with $\mathbb{I}_j$, and φ_r with A_r for $r \in \mathbb{Z}$. For $\mathbf{f}_N^d$, we define $\boldsymbol{\eta}_{\mathbf{f}_N^d} := [\tilde{\eta}_{\mathbf{r}}(\mathbf{f}_N^d) : \mathbf{r} \in \mathbb{F}_{N,d}]$.

Lemma 5.1. *It holds that*

$$\hat{\mathrm{f}}_N^d(\mathbf{l}) = \mathcal{E}_{\boldsymbol{\eta}_{\mathbf{f}_N^d}}(\mathbf{l}), \qquad \mathbf{l} \in \mathbb{J}_{N,d}. \tag{5.1}$$

Proof. Equation (5.1) follows directly from equation (3.11) and the definition of $\mathcal{E}_{\boldsymbol{\eta}_{\mathbf{f}_N^d}}$. □

We next show that the set $\mathbb{J}_{N,d}$ is dimension-reducible.

Lemma 5.2. *The set $\mathbb{J}_{N,d}$ is a dimension-reducible set generated from disjoint finite sets $\mathbb{I}_j$, $j \in \mathbb{Z}_{N+1}$, and $\Gamma'(N,j) = N - j$ for all $j \in \mathbb{Z}_{N+1}$; that is,*

$$\mathbb{I}_{j_1} \cap \mathbb{I}_{j_2} = \emptyset \textit{ with } j_1 \neq j_2, \quad \mathbb{J}_{N,1} = \bigcup_{j \in \mathbb{Z}_{N+1}} \mathbb{I}_j, \quad \textit{and } \mathbb{J}_{N,d} = \bigcup_{j \in \mathbb{Z}_{N+1}} \mathbb{I}_j \otimes \mathbb{J}_{N-j,d-1}. \tag{5.2}$$

Proof. From the definition of $\mathbb{I}_j$, we know for all j_1, $j_2 \in \mathbb{N}_0$ with $j_1 \neq j_2$ that $\mathbb{I}_{j_1} \cap \mathbb{I}_{j_2} = \emptyset$. The definition of $\mathbb{J}_{N,1}$ ensures the validity of the second equation in (5.2). By writing

$$\mathbb{S}_{N,d} = \{[j, \mathbf{j}] : j \in \mathbb{Z}_{N+1}, \mathbf{j} \in \mathbb{S}_{N-j,d-1}\}$$

and noting for $j \in \mathbb{Z}_{N+1}$, $\mathbf{j} \in \mathbb{S}_{N-j,d-1}$ that $\mathbb{I}_{[j,\mathbf{j}]} = \mathbb{I}_j \otimes \mathbb{I}_{\mathbf{j}}$, we obtain the formula

$$\mathbb{J}_{N,d} = \bigcup_{j \in \mathbb{Z}_{N+1}} \mathbb{I}_j \otimes \left(\bigcup_{\mathbf{j} \in \mathbb{S}_{N-j,d-1}} \mathbb{I}_{\mathbf{j}} \right). \tag{5.3}$$

From (5.3) and the definition of $\mathbb{J}_{N-j,d-1}$, we obtain the third equation in (5.2). □

Likewise, we show in the next lemma that the set $\mathbb{F}_{N,d}$ is dimension-reducible.

Lemma 5.3. *The set $\mathbb{F}_{N,d}$ is a dimension-reducible set generated from disjoint finite sets $\mathbb{E}_j$, $j \in \mathbb{Z}_{N+1}$, and $\Gamma(N,j) = N - j$ for all $j \in \mathbb{Z}_{N+1}$; that is,*

$$\mathbb{E}_{j_1} \cap \mathbb{E}_{j_2} = \emptyset \textit{ with } j_1 \neq j_2, \quad \mathbb{F}_{N,1} = \bigcup_{j \in \mathbb{Z}_{N+1}} \mathbb{E}_j, \quad \textit{and } \mathbb{F}_{N,d} = \bigcup_{j \in \mathbb{Z}_{N+1}} \mathbb{E}_j \otimes \mathbb{F}_{N-j,d-1}. \tag{5.4}$$

Proof. Replacing $\mathbb{I}_j$ and $\mathbb{J}_{N,d}$ in the proof of Lemma 5.2 by $\mathbb{E}_j$ and $\mathbb{F}_{N,d}$, respectively, we obtain (5.4). □

Combining Lemmas 5.1, 5.2 and 5.3, we obtain the following proposition.

Proposition 5.4. *The partial sum* (3.11) *is a dimension-reducible sum.*

We now describe Algorithm "FE1d" in the context of this section. Algorithm "FE1d" to be employed in evaluating (4.2) is the discrete form of Algorithm 3.5 in [15] with the case $d = 1$. To make this article self-contained, we review the algorithm below. We let $\boldsymbol{\xi}_N := [\xi_r : r \in \mathbb{F}_{N,1}]$ and $\widetilde{\boldsymbol{\xi}}_N := \left[\left(\widetilde{\boldsymbol{\xi}}_N\right)_l : l \in \mathbb{J}_{N,1}\right]$, where

$\left(\widetilde{\boldsymbol{\xi}}_N\right)_l := \sum_{r\in\mathbb{F}_{N,1}} \xi_r A_r(l)$. Algorithm "FE1d" is designed to evaluate $\widetilde{\boldsymbol{\xi}}_N$ for a given $\boldsymbol{\xi}_N$. For $q \in \mathbb{Z}_m$ and $l \in \mathbb{Z}$, we let

$$t_{0,q}(l) := \frac{1}{\sqrt{2\pi}} \int_I \ell_{0,r_{0,q}}(x) e^{-ilx} dx$$

and

$$t_{j,q}(l) := \frac{1}{2^{j-1}\sqrt{2\pi}} \int_I \ell_{1,r_{1,q}}(x) e^{-ilx/2^{j-1}} dx, \quad j \geq 1.$$

For $j \in \mathbb{N}_0$, $q \in \mathbb{Z}_m$, we set $\boldsymbol{\xi}_{j,q} := [\xi_{rm+q} : \lfloor 2^{j-1} \rfloor \leq r < 2^j]$ and let $\widehat{\boldsymbol{\xi}}_{j,q} := \left[\left(\widehat{\boldsymbol{\xi}}_{j,q}\right)_l : l \in \mathbb{J}_{N,1}\right]$ be the discrete Fourier transform of $\boldsymbol{\xi}_{j,q}$, where

$$\left(\widehat{\boldsymbol{\xi}}_{j,q}\right)_l := \sum_{r=\lfloor 2^{j-1}\rfloor}^{2^j-1} \xi_{j,rm+q} e^{-i2\pi(r-\lfloor 2^{j-1}\rfloor)l/2^{j-1}}.$$

Proposition 3.3 in [15] shows for each $l \in \mathbb{J}_{N,1}$ that

$$\left(\widetilde{\boldsymbol{\xi}}_N\right)_l = \sum_{j\in\mathbb{Z}_{N+1}} \sum_{q\in\mathbb{Z}_m} t_{j,q}(l) \left(\widehat{\boldsymbol{\xi}}_{j,q}\right)_l. \tag{5.5}$$

Using formula (5.5) to compute $\left(\widetilde{\boldsymbol{\xi}}_N\right)_l$ requires the availability of the values of $\left(\widehat{\boldsymbol{\xi}}_{j,q}\right)_l$ and $t_{j,q}(l)$ for all $j \in \mathbb{Z}_{N+1}$, $q \in \mathbb{Z}_m$ and $l \in \mathbb{J}_{N,1}$. We first show how the values of $\left(\widehat{\boldsymbol{\xi}}_{j,q}\right)_l$ are computed. For $l \in \mathbb{Z}_{2^{j-1}}$, we compute $\left(\widehat{\boldsymbol{\xi}}_{j,q}\right)_l$ by applying the fast Fourier transform to $\boldsymbol{\xi}_{j,q}$ directly, and for $l \in \mathbb{J}_{N,1} \setminus \mathbb{Z}_{2^{j-1}}$, we obtain the value $\left(\widehat{\boldsymbol{\xi}}_{j,q}\right)_l$ by employing the periodicity of the discrete Fourier transform. Specifically, for each $j \in \mathbb{N}_0$, we define $L_j : \mathbb{Z} \to \mathbb{Z}_{2^{j-1}}$ for $l \in \mathbb{Z}$ by

$$L_j(l) := \begin{cases} l \bmod 2^{j-1}, & \text{if } l \bmod 2^{j-1} \geq 0, \\ 2^{j-1} + l \bmod 2^{j-1}, & \text{if } l \bmod 2^{j-1} < 0. \end{cases}$$

By periodicity, we have for all $j \in \mathbb{Z}_{N+1}$, $q \in \mathbb{Z}_m$ and $l \in \mathbb{J}_{N,1}$ that

$$\left(\widehat{\boldsymbol{\xi}}_{N,q}\right)_l = \left(\widehat{\boldsymbol{\xi}}_{j,q}\right)_{L_j(l)}. \tag{5.6}$$

Combining equations (5.5) and (5.6) yields the alterative formula

$$\left(\widetilde{\boldsymbol{\xi}}_N\right)_l = \sum_{j\in\mathbb{Z}_{N+1}} \sum_{q\in\mathbb{Z}_m} t_{j,q}(l) \left(\widehat{\boldsymbol{\xi}}_{j,q}\right)_{L_j(l)}, \qquad l \in \mathbb{J}_{N,1}. \tag{5.7}$$

For $N \in \mathbb{N}_0$, we let $\mathbf{t}_N := \{t_{j,q}(l) : l \in \mathbb{J}_{N,1}, j \in \mathbb{Z}_{N+1}, q \in \mathbb{Z}_m\}$. Since $\mathbf{t}_N$ is determined by N, from (5.5), we see that for a fixed $N \in \mathbb{N}_0$, the values in $\mathbf{t}_N$ can be reused. We describe below Algorithm "FE1d" for computing $\widetilde{\boldsymbol{\xi}}_N$.

Algorithm 5.5. FE1d(N, $\boldsymbol{\xi}_N$)

Input: $N \in \mathbb{N}_0$ *and* $\boldsymbol{\xi}_N := [\xi_r : r \in \mathbb{F}_{N,d}]$.

Output: $\widetilde{\boldsymbol{\xi}}_N$.

Step 1: *Compute* $\left[\left(\widehat{\boldsymbol{\xi}}_{j,q}\right)_l : l \in \mathbb{Z}_{2^{j-1}}\right]$ *by applying the fast Fourier transform* [7] *to* $\boldsymbol{\xi}_{j,q}$*, for all* $j \in \mathbb{Z}_{N+1}$ *and* $q \in \mathbb{Z}_m$.

Step 2: *Compute* $\widetilde{\boldsymbol{\xi}}_N$ *according to formula* (5.7).

The number $\mathcal{M}^1(N)$ of operations used in Algorithm 5.5 can be easily estimated by using Theorem 3.6 in [15]. In fact, there exits a positive constant c such that for all $m \in \mathbb{N}$ and $N \in \mathbb{N}_0$,

$$\mathcal{M}^1(N) \le cm2^N N. \tag{5.8}$$

We next present a formula for computing $t_{j,q}(l)$. For $q \in \mathbb{Z}_m$, $\theta \in \mathbb{Z}_{m-1}$, we define the index set

$$\mathbb{S}_{q,\theta} := \left\{ \mathbf{s} \in \mathbb{Z}_m^{m-\theta-1} : s_k < s_{k+1} \text{ for } k \in \mathbb{Z}_{m-\theta-2}, s_k \ne q \text{ for } k \in \mathbb{Z}_{m-\theta-1} \right\}.$$

For each $q \in \mathbb{Z}_m$, we define

$$c_{q,\theta} := \begin{cases} (-1)^{m-\theta-1} \prod\limits_{s\in\mathbb{Z}_m, s\ne q} (v_q - v_s)^{-1} \sum\limits_{\mathbf{s}\in\mathbb{S}_{q,\theta}} \prod\limits_{k\in\mathbb{Z}_{m-\theta-1}} v_{s_k}, & \theta \in \mathbb{Z}_{m-1}, \\ \prod\limits_{s\in\mathbb{Z}_m, s\ne q} (v_q - v_s)^{-1}, & \theta = m-1. \end{cases} \tag{5.9}$$

For $\theta \in \mathbb{Z}_m$ and $\omega \in \mathbb{R}$, we let $\tilde{t}_\theta(\omega) := \int_I x^\theta e^{-i\omega x} dx$. According to Lemma 4.8 in [15], the values of $\tilde{t}_\theta(\omega)$ can be computed exactly by the formula

$$\tilde{t}_\theta(0) = \frac{(2\pi)^{\theta+1}}{\theta+1}, \quad \tilde{t}_\theta(\omega) = -\sum_{\iota\in\mathbb{Z}_{\theta+1}} \frac{\theta!(2\pi)^{\theta-\iota} e^{-i2\pi\omega}}{(\theta-\iota)!(i\omega)^{\iota+1}} + \frac{\theta!}{(i\omega)^{\theta+1}}, \quad \text{for all } \omega \ne 0.$$

It is proved in [15] for all $j \in \mathbb{Z}_{N+1}$, $q \in \mathbb{Z}_m$ and $l \in \mathbb{J}_{N,1}$ that

$$t_{j,q}(l) = \frac{e^{-i2\pi l \lfloor r_{j',q}/m \rfloor / 2^j}}{2^j\sqrt{2\pi}} \sum_{\theta\in\mathbb{Z}_m} c_{r_{j',q} \bmod m, \theta} \tilde{t}_\theta(l/2^j), \tag{5.10}$$

where $j' = 0$ if $j = 0$, and $j' = 1$ if $j > 0$.

Algorithm 5.6. FFE(N, m, $\mathbf{f}_N^d$)
Input: $d \in \mathbb{N}$, $N \in \mathbb{N}$, $m \in \mathbb{N}$, $\mathbf{f}_N^d := [\mathrm{f}_\mathbf{r} : \mathbf{r} \in \mathbb{F}_{N,d}]$.
Output: $\hat{\mathbf{f}}_N^d$.

Step 1: *Compute* $\boldsymbol{\eta}_{\mathbf{f}_N^d} := \left[\tilde{\eta}_\mathbf{r}(\mathbf{f}_N^d) : \mathbf{r} \in \mathbb{F}_{N,d}\right]$.
Step 2: *Compute* $\mathbf{t}_N := [t_{j,q}(l) : l \in \mathbb{J}_{N,1}, j \in \mathbb{Z}_{N+1}$ and $q \in \mathbb{Z}_m]$, *according to formula* (5.10).
Step 3: *Compute* $\hat{\mathbf{f}}_N^d :=$ FEd$(d, N, \boldsymbol{\eta}_{\mathbf{f}_N^d})$ *by calling Algorithm* 4.5 *which uses Algorithm* 5.5.

For each $N \in \mathbb{N}$, we let $\mathcal{Z}_N^d$ denote the number of operations used in computing $\hat{\mathbf{f}}_N^d$ by using Algorithm 5.6 for a given $\mathbf{f}_N^d$. We next estimate $\mathcal{Z}_N^d$. As a preparation, we first estimate the cardinality of $\mathbb{J}_{N,d}$ and $\mathbb{F}_{N,d}$.

Lemma 5.7. *It holds that for all* $N \in \mathbb{N}$ *and* $d \in \mathbb{N}$,

$$\mathcal{C}(\mathbb{J}_{N,d}) \le \tilde{m}^d n(\log n + 1)^{d-1} \tag{5.11}$$

and

$$\mathcal{C}(\mathbb{F}_{N,d}) \le m^d n(\log n + 1)^{d-1}. \tag{5.12}$$

Proof. We first prove (5.11). It is clear that (5.11) holds for $d = 1$. We consider the cases $d \in \mathbb{N}$ and $d \geq 2$. By writing $\mathbb{S}_{N,d} = \{[\mathbf{j}, j] : \mathbf{j} \in \mathbb{S}_{N,d-1}, j \in \mathbb{Z}_{N-|\mathbf{j}|+1}\}$ and noting for $\mathbf{j} \in \mathbb{S}_{N,d-1}$, $j \in \mathbb{Z}_{N-|\mathbf{j}|+1}$ that $\mathbb{I}_{[\mathbf{j},j]} = \mathbb{I}_{\mathbf{j}} \otimes \mathbb{I}_j$, we obtain the formula

$$\mathbb{J}_{N,d} = \bigcup_{\mathbf{j} \in \mathbb{S}_{N,d-1}} \mathbb{I}_{\mathbf{j}} \otimes \left(\bigcup_{j \in \mathbb{Z}_{N-|\mathbf{j}|+1}} \mathbb{I}_j \right). \tag{5.13}$$

Note that $\bigcup_{j \in \mathbb{Z}_{N-|\mathbf{j}|+1}} \mathbb{I}_j = \mathbb{J}_{N-|\mathbf{j}|,1}$. Thus, from (5.13) we have that

$$\mathbb{J}_{N,d} = \bigcup_{\mathbf{j} \in \mathbb{S}_{N,d-1}} \mathbb{I}_{\mathbf{j}} \otimes \mathbb{J}_{N-|\mathbf{j}|,1}. \tag{5.14}$$

Since $\mathcal{C}(\mathbb{I}_{\mathbf{j}}) \leq \tilde{m}^{d-1} 2^{|\mathbf{j}|}$ and $\mathcal{C}(\mathbb{J}_{N-|\mathbf{j}|,1}) = \tilde{m} 2^{N-|\mathbf{j}|}$, equality (5.14) leads to

$$\mathcal{C}(\mathbb{J}_{N,d}) \leq \tilde{m}^d 2^N \sum_{\mathbf{j} \in \mathbb{S}_{N,d-1}} 1. \tag{5.15}$$

It is easy to shows that $\mathcal{C}(\mathbb{S}_{N,d-1}) = \frac{(N+d-1)!}{N!(d-1)!}$. Thus, inequality (5.15) can be written as

$$\mathcal{C}(\mathbb{J}_{N,d}) \leq \tilde{m}^d 2^N \frac{(N+d-1)!}{N!(d-1)!}. \tag{5.16}$$

Since

$$\frac{(N+d-1)!}{N!(d-1)!(N+1)^{d-1}} = \frac{1}{(d-1)!} \prod_{i=1}^{d-2} (1 + \frac{i}{N+1}) \leq \frac{1}{d-1} \prod_{i=1}^{d-2} \frac{i+1}{i} = 1,$$

we obtain (5.11) from (5.16).

In the above proof, by replacing $\tilde{m}$ by m, $\mathbb{J}_{N,d}$ by $\mathbb{F}_{N,d}$, and $\mathbb{I}_{\mathbf{j}}$ by $\mathbb{E}_{\mathbf{j}}$, we obtain (5.12). □

For each $N \in \mathbb{N}_0$, we let $\mathcal{Z}_N^d$ denote the number of operations used in computing $\hat{\mathbf{f}}_N^d$ by using Algorithm 5.6 for a given $\mathbf{f}_N^d$. In the next theorem we give an estimate of $\mathcal{Z}_N^d$.

Theorem 5.8. *If $n := 2^N$ and $m = \tilde{m}$, then there exists a positive constant c such that for all $N \in \mathbb{N}$, we have*

$$\mathcal{Z}_N^d \leq c m^d n (\log n + d)^d, \tag{5.17}$$

where c does not depend on m and d.

Proof. We let $\mathcal{Z}^{(i)}$ denote the number of operations used in Step 1 of Algorithm 5.6. In Step 1, for each $\mathbf{r} \in \mathbb{F}_{N,d}$, the number of operations used in computing $\tilde{\eta}_{\mathbf{r}}(\mathbf{f}_N^d)$ is $\mathcal{O}(m^d)$. By Proposition 2.13 of [15], the number of components in $\boldsymbol{\eta}_{\mathbf{f}_N^d}$ is $\mathcal{O}(m^d 2^N N^{d-1})$. Thus, $\mathcal{Z}^{(1)} = \mathcal{O}(m^d 2^N N^{d-1})$. From equation (5.10), for each $l \in \mathbb{J}_{N,1}$, $j \in \mathbb{Z}_{N+1}$ and $q \in \mathbb{Z}_m$, the number of operations used in Step 2 for computing $t_{j,q}(l)$ is $\mathcal{O}(m^2)$. Since there are $2(2^{N+1} - 1) - N$ such $t_{j,q}(l)$, $\mathcal{Z}^{(2)} = \mathcal{O}(m^2 2^N)$. From Lemma 5.7 and (5.8), we know that the assumptions of Proposition 4.7 are satisfied. Thus, it follows from Proposition 4.7 that there exists a positive constant c_1 such that for all $N \in \mathbb{N}$, we have

$$\mathcal{Z}^{(3)} \leq c_1 m^d 2^N (N + d)^d.$$

The total number of operations used in Algorithm 5.6 is equal to the number of operations used in all steps. Thus, we conclude that there exists a positive constant c such that for all $N \in \mathbb{N}$, $\mathcal{Z}_N^d \le cm^d 2^N (N+d)^d$. Noting that $n = 2^N$, we obtain estimate (5.17). □

We close this section with a remark on an improvement of Algorithm 5.6. The computational speed of the algorithm can be increased by employing the refinement equation of the Lagrange basis functions $\ell_{j,r}$, $j \in \mathbb{N}_0$, and $r \in \mathbb{Z}_{j,m}$. Specifically, the computational complexity for computing $\{\mathcal{V}^d_{N,\xi^d_N,j,\mathbf{l}}(l) : l \in \mathbb{J}_{j,1}\}$ for all $j \in \mathbb{Z}_{N+1}$ and $\mathbf{l} \in \mathbb{J}_{N-j,d-1}$ can be reduced by making use of the refinement equation. We explain this below. From Lemma 2.1 in [15], the Lagrange basis functions $\ell_{j,r}$, $j \in \mathbb{N}_0$, and $r \in \mathbb{Z}_{j,m}$ satisfy the following refinement equation:

$$\ell_{j,r} = \sum_{\kappa \in \mathbb{Z}_{2m}} a_{r \mod m,\kappa} \ell_{j+1,\vartheta(r)+\kappa}.$$

Thus, from their definition, we know that $A_{j,r}$, $j \in \mathbb{N}_0$, and $r \in \mathbb{Z}_{j,m}$ satisfy

$$A_{j,r} = \sum_{\kappa \in \mathbb{Z}_{2m}} a_{r \mod m,\kappa} A_{j+1,\vartheta(r)+\kappa}. \tag{5.18}$$

By using equality (5.18), for all $j \in \mathbb{Z}_{N+1}$, $\mathbf{l} \in \mathbb{J}_{N-j,d-1}$, we rewrite $\mathcal{V}^d_{N,\boldsymbol{\xi}^d_N,j,\mathbf{l}}$ as

$$\mathcal{V}^d_{N,\boldsymbol{\xi}^d_N,j,\mathbf{l}} = \sum_{r \in \mathbb{Z}_{2^j m}} b_{\boldsymbol{\xi}^d_N,j,r}(\mathbf{l}) A_{j,r}, \tag{5.19}$$

where $b_{\boldsymbol{\xi}^d_N,j,r}(\mathbf{l})$, $r \in \mathbb{Z}_{2^j m}$, are defined by

$$b_{\xi^d_N,j,r}(\mathbf{l}) := \begin{cases} \mathcal{E}^{d-1}_{N,(\xi^d_N)_{0,r}}(\mathbf{l}), & j = 0, \\ \mathcal{E}^{d-1}_{N-j,(\xi^d_N)_{j,r}}(\mathbf{l}) + \sum_{q \in \mathbb{Z}_m} a_{q,r \bmod 2m} b_{\xi^d_N,j-1,\vartheta(r)+q}(\mathbf{l}), & \\ \qquad\qquad j > 0 \text{ and } r \in \mathbb{Z}_{j,m}, & \\ \sum_{q \in \mathbb{Z}_m} a_{q,r \mod 2m} b_{\xi^d_N,j-1,\vartheta(r)+q}(\mathbf{l}), & j > 0 \text{ and } r \notin \mathbb{Z}_{j,m}. \end{cases} \tag{5.20}$$

From (5.19), we see that $\mathcal{V}^d_{N,\xi^d_N,j,\mathbf{l}}$ can be presented as a linear combination of $A_{j,r}$, $r \in \mathbb{Z}_{2^j m}$, which share the same scale factor j. This leads us to an algorithm for evaluating $\mathcal{V}^d_{N,\xi^d_N,j,\mathbf{l}}$ on $l \in \mathbb{J}_{j,1}$ by (5.19), faster than Algorithm 5.5. By using this technique, we can develop an algorithm for computing the Fourier transform on sparse grids which uses fewer operations than Algorithm 5.6.

6. Fast discrete backward Fourier transform on sparse grids

In this section, we develop a fast algorithm for computing the discrete backward Fourier transform of $\mathbf{c}^d_N$ on $\widetilde{S}_{N,d}$ by using Algorithm 4.5.

We show that formula (2.5) is a dimension-reducible sum. To this end, we prove that $F_{\mathbf{c}^d_N}$ is a sum in the form of (4.2) and that $\widetilde{S}_{N,d}$ is a dimension-reducible set generated from disjoint finite sets $\widetilde{G}_j$, $j \in \mathbb{Z}_{N+1}$, and $\Gamma'(N,j) = N - j$ for all $j \in \mathbb{Z}_{N+1}$. We specify in this section $\mathbb{L}_{N,d}$ with $\mathbb{J}_{N,d}$, $\mathbb{U}_{N,d}$ with $\widetilde{S}_{N,d}$, $\mathbb{Y}_j$ with $\mathbb{I}_j$, $\mathbb{V}_j$ with $\widetilde{G}_j$, and φ_r with e_r for $r \in \mathbb{Z}$.

Lemma 6.1. *There holds the relation*

$$F_{\mathbf{c}_N^d} = \mathcal{E}_{\mathbf{c}_N^d}. \tag{6.1}$$

Proof. Equation (6.1) follows directly from equation (2.5) and (4.2). □

We now prove that the set $\widetilde{S}_{N,d}$ is dimension-reducible.

Lemma 6.2. *The set $\widetilde{S}_{N,d}$ is a dimension-reducible set generated from disjoint finite sets $\widetilde{G}_j$, $j \in \mathbb{Z}_{N+1}$, and $\Gamma(N,j) = N - j$ for all $j \in \mathbb{Z}_{N+1}$; that is,*

$$\widetilde{G}_{j_1} \cap \widetilde{G}_{j_2} = \emptyset \text{ with } j_1 \neq j_2, \quad \widetilde{S}_{N,1} = \bigcup_{j\in\mathbb{Z}_{N+1}} \widetilde{G}_j, \quad \text{and } \widetilde{S}_{N,d} = \bigcup_{j\in\mathbb{Z}_{N+1}} \widetilde{G}_j \otimes \widetilde{S}_{N-j,d-1}. \tag{6.2}$$

Proof. By replacing $\mathbb{I}_j$ and $\mathbb{J}_{N,d}$ in the proof of Lemma 5.2 by $\widetilde{G}_j$ and $\widetilde{S}_{N,d}$, respectively, we obtain equation (6.2). □

Combining Lemmas 5.2, 6.1, and 6.2, we obtain the following proposition.

Proposition 6.3. *The sum* (2.5) *is a dimension-reducible sum.*

Proposition 6.3 ensures that Algorithm 4.5 can be used to compute the discrete backward Fourier transform of $\mathbf{c}_N^d$ on $\widetilde{S}_{N,d}$. The use of Algorithm 4.5 requires the availability of Algorithm "FE1d" for computing $F_{\mathbf{c}_N^1}(v)$ given $\mathbf{c}_N^1$, for $v \in \widetilde{S}_{N,1}$. From the definition of $\widetilde{S}_{N,1}$, we know that $\widetilde{S}_{N,1} = \widetilde{V}_N$. Thus, for given $\mathbf{c}_N^1 := [c_l : j \in \mathbb{J}_{N,1}]$, we compute

$$F_{\mathbf{c}_N^1}(v) = \sum_{l\in\mathbb{J}_{N,1}} c_l e^{ilv}, \qquad v \in \widetilde{V}_N. \tag{6.3}$$

Recalling $\widetilde{V}_N = \{\frac{2\pi r}{2^N m} : r \in \mathbb{Z}_{2^N m}\}$, from (6.3) we obtain that

$$F_{\mathbf{c}_N^1}(v_{N,r}) = \sum_{l\in\mathbb{J}_{N,1}} c_l e^{\frac{i2\pi lr}{2^j m}}, \qquad r \in \mathbb{Z}_{2^N m}. \tag{6.4}$$

For $x \in \mathbb{R}$, we denote by $\lceil x \rceil$ the smallest integer not less than x. We let $\widetilde{N}$ be the smallest integer such that $2^{\widetilde{N}} m \geq 2^N \tilde{m}$ and $\widetilde{N} \geq N$, where $\tilde{m}$ appears in the definition of $\mathbb{I}_j$. This is $\widetilde{N} = N + \lceil \max\{0, \log(\frac{\tilde{m}}{m})\} \rceil$. We define $\tilde{c}_l := c_l$ for all $l \in \mathbb{J}_{N,1}$, with $l \geq 0$, $\tilde{c}_l := c_{l-2^{\widetilde{N}}m}$ for all $l - 2^{\widetilde{N}} m \in \mathbb{J}_{N,1}$ with $l - 2^{\widetilde{N}} m < 0$ and $\tilde{c}_l := 0$ otherwise, and let $\tilde{\mathbf{c}}_N := [\tilde{c}_l : l \in \mathbb{Z}_{2^{\widetilde{N}}m}]$. By $\check{\tilde{\mathbf{c}}}_N$ we denote the discrete backward Fourier transform of $\tilde{\mathbf{c}}_N$. That is,

$$\left(\check{\tilde{\mathbf{c}}}_N\right)_r := \sum_{l\in\mathbb{Z}_{2^{\widetilde{N}}m}} \tilde{c}_l e^{i2\pi lr/(2^{\widetilde{N}}m)}, \qquad r \in \mathbb{Z}_{2^{\widetilde{N}}m}, \tag{6.5}$$

which can be computed by the fast Fourier transform. Substituting (6.5) into (6.3) yields that

$$F_{\mathbf{c}_N^1}\left(\frac{2\pi r}{2^N m}\right) = \left(\check{\tilde{\mathbf{c}}}_N\right)_{2^{\widetilde{N}-N}r}, \quad \text{for } r \in \mathbb{Z}_{2^N m}. \tag{6.6}$$

For $\mathbf{c}_N^1$, we let $\check{\mathbf{c}}_N^1 := [F_{\mathbf{c}_N^1}(v) : v \in \bar{S}_{N,1}]$.

Algorithm 6.4. FE1d(N, $\mathbf{c}_N$)
Input: $N \in \mathbb{N}_0$ *and* $\mathbf{c}_N^1 := [c_l : l \in \mathbb{J}_{N,1}]$.
Output: $\check{\mathbf{c}}_N^1$.

Step 1: *Select* $\widetilde{N} = N + \max\left\{0, \log(\frac{\tilde{m}}{m})\right\}$.
Step 2: *Compute the inverse Fourier transform* $\check{\tilde{\mathbf{c}}}_N$ *of* $\tilde{\mathbf{c}}_N$ *by using the fast Fourier transform.*
Step 3: *Compute* $\check{\mathbf{c}}_N^1$ *by using formula* (6.6).

Combining Algorithms 4.5 and 6.4 we have the following fast algorithm for computing the discrete backward Fourier transform of $\mathbf{c}_N^d$ on $\widetilde{S}_{N,d}$.

Algorithm 6.5. IFFE(N, $\mathbf{c}_N^d$)
Input: $N \in \mathbb{N}$, $\mathbf{c}_N^d := [c_{\mathbf{l}} : \mathbf{l} \in \mathbb{J}_{N,d}]$.
Output: $\check{\mathbf{c}}_N^d$.
Step 1: *Compute* $\check{\mathbf{c}}_N^d := \mathrm{FEd}(d, N, \mathbf{c}_N^d)$ *by recalling Algorithm* 4.5*, which uses Algorithm* 6.4.

We next estimate the number $\mathcal{N}_N^d$ of operations used in Algorithm 6.5 for computing the discrete backward Fourier transform of $\mathbf{c}_N^d$ on $\widetilde{S}_{N,d}$.

Theorem 6.6. *If* $n := 2^N$ *and* $m = \tilde{m}$*, then there exists a positive constant* c *such that for all* $N \in \mathbb{N}$*, we have*

$$\mathcal{N}_N^d \leq cm^d n(\log n + d)^d, \tag{6.7}$$

where c *does not depend on* m *and* d.

Proof. To prove (6.7), it suffices to verify the hypotheses of Theorem 4.7. Noting that $m = \tilde{m}$, from Lemma 5.7 we know that $\mathcal{C}(\mathbb{J}_{N,d}) \leq m^d n(\log n + 1)^{d-1}$. By replacing $\mathbb{J}_{N,d}$ and $\mathbb{I}_{\mathbf{j}}$ by $\widetilde{S}_{N,d}$ and $\widetilde{G}_{\mathbf{j}}$, respectively, in the proof of Lemma 5.7, we can see that $\mathcal{C}(\widetilde{S}_{N,d}) \leq m^d n(\log n + 1)^{d-1}$. It remains to prove that the number of operations used in Algorithm 6.4 is $\mathcal{O}(m2^N N)$. From the definition of $\widetilde{N}$ we know that there exists a positive constant c_0 such that for all $N \in \mathbb{N}_0$, $2^{\widetilde{N}} \leq c_0 2^N$. Hence, the number of operations used in computing the fast inverse Fourier transform of $\tilde{\mathbf{c}}_N$ is $\mathcal{O}(m2^N N)$. Noting that for each $r \in \mathbb{Z}_{2^N m}$, the number of operations used in computing $F_{\mathbf{c}_N^1}\left(\frac{2\pi r}{2^N m}\right)$ by (6.6) is $\mathcal{O}(1)$, we conclude that the number of operations used in Algorithm 6.4 is $\mathcal{O}(m2^N N)$. That is, the hypotheses of Theorem 4.7 hold. □

7. Numerical examples

We present in this section numerical examples to demonstrate the performance of the fast algorithms developed in the last two sections. We compare approximation accuracy and (or) computational efficiency of the proposed algorithms with those of the known algorithms for Fourier transform on sparse grids (Algorithm HFFT) and the inverse Fourier transform on sparse grids (Algorithm IHFFT), which were originally introduced in [14] and implemented in [11, 12]. All computer programs for the numerical examples presented in this section are run on a workstation with a 4 core Intel CPU (each of which has 2.8GHz) and 4GB memory.

The first three numerical examples test Algorithm 5.6 for computing the discrete Fourier transform of data sets on a sparse grid at a hyperbolic cross domain. The algorithm used in these computations is the one improved by the idea outlined at the end of section 5. The data sets in these examples are sampled from functions in spaces $H_{\mathrm{mix}}^{0.5-\varepsilon}(I^d)$, where ε is an arbitrarily small positive number. Therefore, according to our theory, the theoretical approximation order for the functions is

$0.5 - \varepsilon$. In Examples 1, 2 and 3, we choose $s = 0.49$ when we compute the Sobolev norm. We compute the relative error by the formula

$$\mathrm{Err} := \|f - F^d_{N,\hat{\mathbf{f}}^d_N}\| / \|f\|_{H^s_{\mathrm{mix}}(I^d)}$$

and the approximation order by the formula

$$\mathrm{AO} := \log_2 \|f - F^d_{N,\hat{\mathbf{f}}^d_N}\| / \|f - F^d_{N+1,\hat{\mathbf{f}}^d_{N+1}}\|.$$

In the presented numerical results, we use "Time" for the computing time spent in calculating the Fourier transform, use "Memory" for the memory used, and use "Num" for the number of the elements in the output set of an algorithm. Here, Time is the average of the times spent in three different runs for the same experiment. We define

$$\mathrm{TC} := \frac{\mathrm{Time}}{n \log^d n}, \quad \text{where} \quad n := 2^N,$$

from which we can observe the constants c in (5.17) and (6.7).

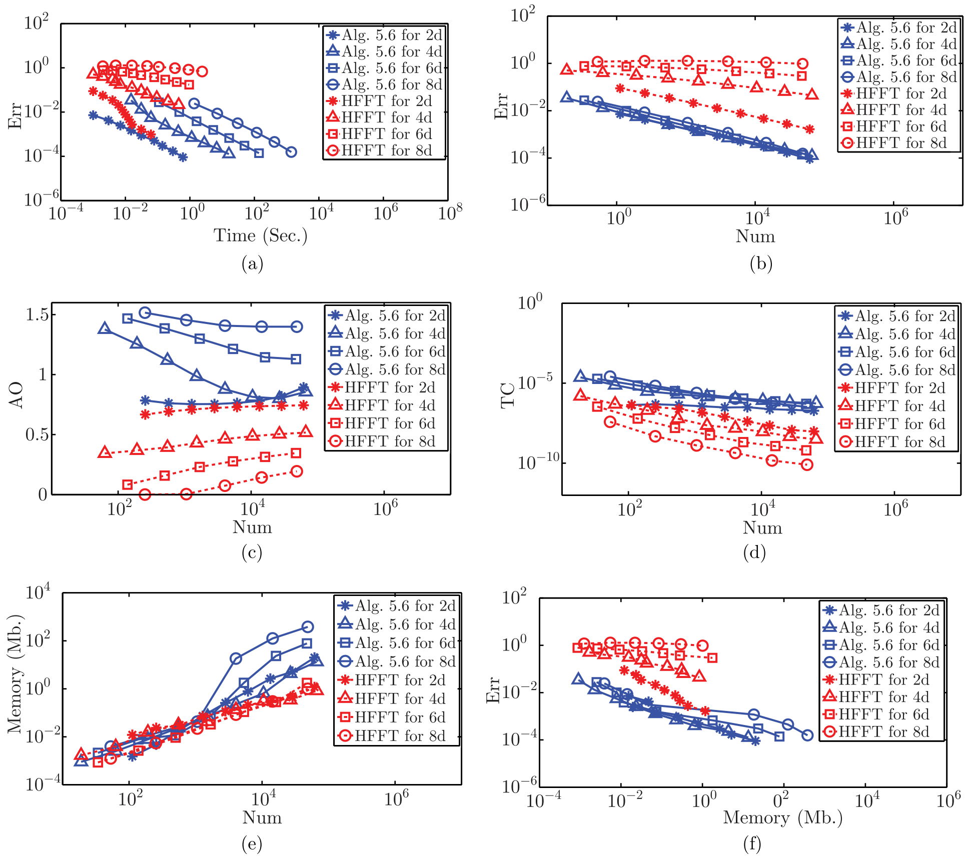


FIGURE 6. Example 1: (a) Err via Time, (b) Err via Num, (c) AO via Num, (d) TC via Num, (e) Memory via Num, (f) Err via Memory.

For the results of Algorithm 5.6 in Example 1, we select $N = 6, 7, \ldots, 14$ for the case $d = 2$, $N = 3, 4, \ldots, 11$ for the case $d = 4$, $N = 3, 4, \ldots, 9$ for the case $d = 6$, and $N = 3, 4, \ldots, 8$ for the case $d = 8$. For the results of Algorithm HFFT in Example 1, we select $N = 6, 7, \ldots, 15$ for the case $d = 2$, $N = 3, 4, \ldots, 13$ for the case $d = 4$, $N = 3, 4, \ldots, 11$ for the case $d = 6$, and $N = 3, 4, \ldots, 10$ for the case $d = 8$. In Examples 2 and 3 for both Algorithms 5.6 and HFFT, we select $N = 6, 7, \ldots, 14$ for the case $d = 2$, $N = 3, 4, \ldots, 11$ for the case $d = 4$, $N = 3, 4, \ldots, 9$ for the case $d = 6$, and $N = 3, 4, \ldots, 8$ for the case $d = 8$. In computing Err and AO in these examples, we replace f in the formulas for these quantities by $F^d_{N,\hat{\mathbf{f}}^d_N}$ with $N = 16$ for $d = 2$ and $d = 4$, $N = 14$ for $d = 6$ and 8. In all figures shown in this section, we use solid blue lines (resp. broken red lines) with $*$, $\triangle$, $\square$ and $\bigcirc$ to represent the values obtained by using Algorithm 5.6 (resp. Algorithm HFFT) for the cases $d = 2, 4, 6, 8$, respectively. For Example 1, we use tables to present the numerical results and figures to demonstrate them. To save space, for Examples 2–4, we omit the tables, presenting only figures to demonstrate the numerical results.

Example 1. We compute the discrete Fourier transform of the data $\mathbf{f}^d_N$ at $\mathbb{J}_{N,d}$ for $d = 2, 4, 6, 8$ with $\tilde{m} = 1$ by using Algorithm 5.6 and compare the numerical results obtained from this algorithm with those from the classical algorithm HFFT. The data set $\mathbf{f}^d_N$ is obtained from sampling the functions

$$f_d(\mathbf{x}) := \sqrt{d\pi^2 - \sum_{k\in\mathbb{Z}_d} (x_k - \pi)^2}, \quad \mathbf{x} \in I^d.$$

Functions f_d are periodic and have no partial derivatives at the vertexes of I^d. The sampling points $v_{\mathbf{j},\mathbf{r}}$, for $\mathbf{j} \in \mathbb{S}_{N,d}$ and $\mathbf{r} \in \mathbb{Z}^d_{\mathbf{j},2}$, are generated by the refinable sets V_j with $V_0 := \{\frac{2\pi}{3}, \frac{4\pi}{3}\}$.

We show in Tables 1-4 the numerical results for Err, Time, Num, AO, Memory and TC for the functions f_d, $d = 2, 4, 6, 8$, obtained from Algorithm 5.6 and Algorithm HFFT. Tables 1, 2, 3, and 4 list the values for functions f_2, f_4, f_4 and f_8, respectively. These numerical results are shown in Figure 6.

TABLE 1. Example 1: Numerical results for function f_2 of Algorithm 5.5 with a comparison to Algorithm HFFT.

		Alg 5.6					HFFT				
N	Num	Err	Time (Sec.)	AO	Memory	TC	Err	Time (Sec.)	AO	Memory	TC
6	112	$7.21e-3$	$1e-3$		$1.5e-2$	$4.3e-7$	$8.82e-2$	$1e-3$		$1.2e-2$	$4.3e-7$
7	256	$4.19e-3$	$3e-3$	0.78	$4.8e-2$	$4.8e-7$	$5.55e-2$	$2e-3$	0.67	$2.3e-2$	$3.2e-7$
8	576	$2.47e-3$	$7e-3$	0.76	$1.9e-2$	$4.3e-7$	$3.44e-2$	$4e-3$	0.69	$3.1e-2$	$2.4e-7$
9	1280	$1.47e-3$	$1.5e-2$	0.75	$7.4e-2$	$3.6e-7$	$2.11e-2$	$6e-3$	0.71	$7.0e-2$	$1.5e-7$
10	2816	$8.69e-4$	$3.1e-2$	0.75	$2.6e-1$	$3.1e-7$	$1.27e-2$	$8e-3$	0.72	$1.2e-1$	$7.8e-8$
11	6144	$5.12e-4$	$7.8e-2$	0.76	$7.9e-1$	$3.2e-7$	$7.68e-3$	$1e-2$	0.73	$2.2e-1$	$4.1e-8$
12	13312	$2.98e-4$	$1.4e-1$	0.78	$2.7e-0$	$2.4e-7$	$4.61e-3$	$1.3e-2$	0.74	$2.8e-1$	$2.2e-8$
13	28672	$1.69e-4$	$2.9e-1$	0.82	$5.2e-0$	$2.2e-7$	$2.76e-3$	$1.6e-2$	0.74	$4.4e-1$	$1.2e-8$
14	61440	$9.10e-5$	$6.1e-1$	0.89	$2.0e+1$	$1.9e-7$	$1.65e-3$	$3.1e-2$	0.74	$1.2e-0$	$9.6e-9$
15	131072						$9.82e-4$	$6.2e-2$	0.75	$2.2e-0$	$8.4e-9$

TABLE 2. Example 1: Numerical results for function f_4 of Algorithm 5.5 with a comparison to Algorithm HFFT.

		Alg 5.6					HFFT				
N	Num	Err	Time (Sec.)	AO	Memory	TC	Err	Time (Sec.)	AO	Memory	TC
3	19	$3.41e-2$	$1.5e-2$		$9.1e-4$	$2.3e-5$	$5.06e-1$	$1e-3$		$1.7e-3$	$1.5e-6$
4	63	$1.31e-2$	$3.1e-2$	1.38	$2.3e-3$	$7.6e-6$	$3.99e-1$	$2e-3$	0.34	$3.9e-3$	$4.9e-7$
5	192	$5.51e-3$	$6.2e-2$	1.25	$7.8e-3$	$3.1e-6$	$3.09e-1$	$4e-3$	0.37	$1.6e-2$	$2.1e-7$
6	552	$2.54e-3$	$1.7e-1$	1.12	$2.4e-2$	$2.1e-6$	$2.35e-1$	$5e-3$	0.39	$3.2e-2$	$6.1e-8$
7	1520	$1.28e-3$	$4.4e-1$	0.98	$6.9e-2$	$1.4e-6$	$1.75e-1$	$7e-3$	0.43	$4.3e-2$	$2.3e-8$
8	4048	$6.98e-4$	$1.1e-0$	0.88	$2.3e-1$	$1.1e-6$	$1.27e-1$	$1.6e-2$	0.46	$1.4e-1$	$1.5e-8$
9	10496	$3.97e-4$	$2.8e-0$	0.81	$6.4e-1$	$8.4e-7$	$9.05e-2$	$3.1e-2$	0.49	$3.1e-1$	$9.2e-9$
10	26624	$2.28e-4$	$6.8e-0$	0.81	$4.3e-0$	$6.6e-7$	$6.38e-2$	$4.7e-2$	0.51	$3.4e-1$	$4.6e-9$
11	66304	$1.26e-4$	$1.6e+1$	0.86	$1.3e+1$	$5.4e-7$	$4.46e-2$	$9.4e-2$	0.52	$8.4e-1$	$3.1e-9$
12	162560						$3.09e-2$	$2.1e-1$	0.53	$1.9e-0$	$2.2e-9$
13	393216						$2.14e-2$	$4.5e-1$	0.53	$4.1e-0$	$1.9e-9$

TABLE 3. Example 1: Numerical results for function f_6 of Algorithm 5.6 with a comparison to Algorithm HFFT.

		Alg 5.6					HFFT				
N	Num	Err	Time (Sec.)	AO	Memory	TC	Err	Time (Sec.)	AO	Memory	TC
3	34	$2.77e-2$	$1.1e-1$		$2.5e-3$	$1.9e-5$	$7.81e-1$	$2e-3$		$8.6e-4$	$3.4e-7$
4	138	$1.01e-2$	$4.7e-1$	1.47	$7.8e-3$	$7.1e-6$	$7.38e-1$	$4e-3$	0.08	$2.7e-3$	$6.1e-8$
5	501	$3.84e-3$	$1.6e-0$	1.39	$1.2e-2$	$3.2e-6$	$6.61e-1$	$8e-3$	0.16	$9.3e-3$	$1.6e-8$
6	1683	$1.56e-3$	$5.3e-0$	1.29	$7.4e-2$	$1.8e-6$	$5.64e-1$	$1.7e-2$	0.23	$3.3e-2$	$5.7e-9$
7	5336	$6.72e-4$	$1.7e+1$	1.21	$1.8e-0$	$1.1e-6$	$4.65e-1$	$3.1e-2$	0.28	$1.1e-1$	$2.7e-9$
8	16172	$3.04e-4$	$4.9e+1$	1.14	$2.3e+1$	$7.3e-7$	$3.74e-1$	$7.8e-2$	0.31	$2.8e-1$	$1.2e-9$
9	47264	$1.39e-4$	$1.4e+2$	1.13	$7.8e+1$	$5.1e-7$	$2.94e-1$	$1.7e-1$	0.35	$1.8e-0$	$6.3e-10$
10							$2.27e-1$	$3.9e-1$	0.37	$3.3e-0$	$3.8e-10$
11							$1.73e-1$	$9.5e-1$	0.39	$6.5e-0$	$2.6e-10$

TABLE 4. Example 1: Numerical results for function f_8 of Algorithm 5.6 with a comparison to Algorithm HFFT.

		Alg 5.6					HFFT				
N	Num	Err	Time (Sec.)	AO	Memory	TC	Err	Time (Sec.)	AO	Memory	TC
3	53	$2.45e-2$	$1.4e-0$		$3.9e-3$	$2.6e-5$	$1.18e-0$	$2e-3$		$1.3e-3$	$3.8e-8$
4	253	$8.57e-3$	$7.1e-0$	1.52	$1.4e-2$	$6.7e-6$	$1.29e-0$	$5e-3$	0	$5.5e-3$	$4.8e-8$
5	1059	$3.13e-3$	$3.0e+1$	1.45	$4.3e-2$	$2.4e-6$	$1.28e-0$	$1.6e-2$	0	$2.2e-2$	$1.3e-9$
6	4043	$1.18e-3$	$1.2e+2$	1.41	$1.8e+1$	$1.1e-6$	$1.22e-0$	$4.7e-2$	0.07	$8.4e-2$	$4.4e-10$
7	14407	$4.47e-4$	$4.2e+2$	1.39	$1.3e+2$	$5.6e-7$	$1.11e-0$	$1.1e-1$	0.14	$3.0e-1$	$1.5e-10$
8	48639	$1.56e-4$	$1.4e+3$	1.39	$3.8e+2$	$3.3e-7$	$9.68e-1$	$3.4e-1$	0.19	$1.0e-0$	$8.0e-10$
9							$8.23e-1$	$9.0e-1$	0.23	$9.1e-0$	$4.1e-11$
10							$6.83e-1$	$2.4e-0$	0.27	$1.9e+1$	$2.6e-11$

Example 2. We compare in this example Algorithm 5.6 and Algorithm HFFT when we use the same sparse grids and the same hyperbolic cross index sets in both of these algorithms. Specifically, we select the sparse grid to consist of the points $v_{\mathbf{j},\mathbf{r}}$, $\mathbf{j} \in \mathbb{S}_{N,d}$ and $\mathbf{r} \in \mathbb{Z}^d_{\mathbf{j},2}$, which are generated by the refinable sets V_j with the initial set $V_0 := \{0, \pi\}$, and select the hyperbolic cross index set as $\mathbb{J}_{N,d}$ with $\tilde{m} = 2$. We compute the discrete Fourier transform of the data set $\mathbf{f}^d_N$ obtained by sampling the same functions f_d, $d = 2, 4, 6, 8$, as in Example 1. The numerical results are shown in Figure 7.

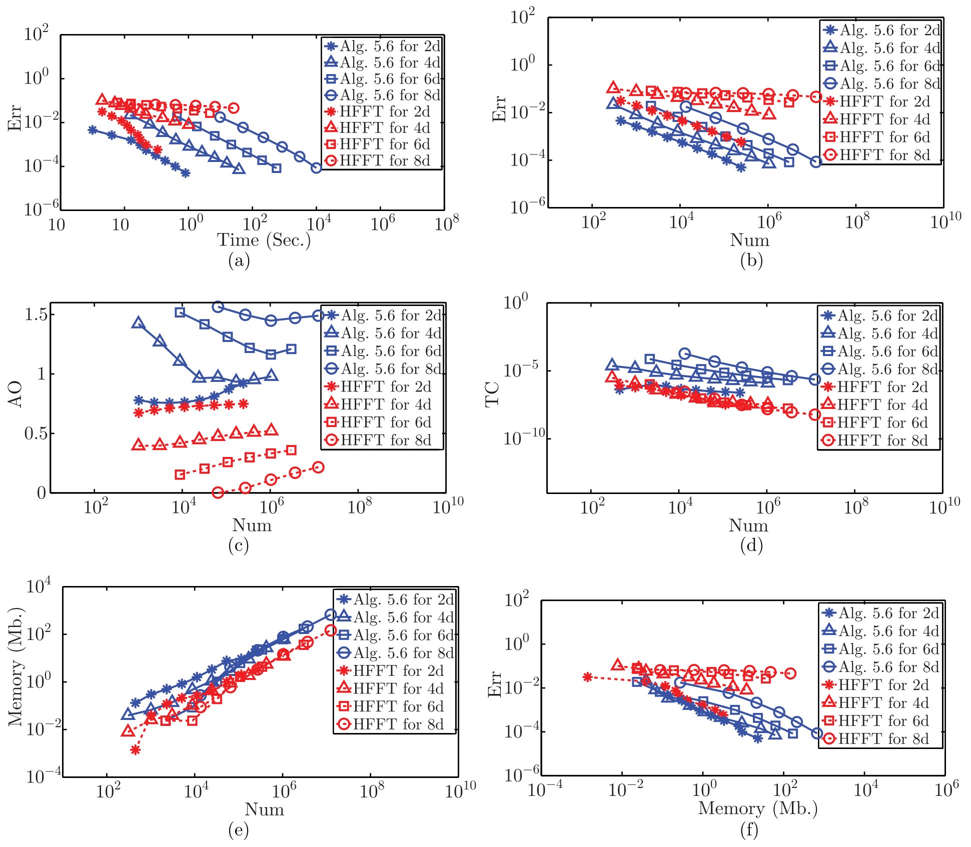


FIGURE 7. Example 2: (a) Err via Time, (b) Err via Num, (c) AO via Num, (d) TC via Num, (e) Memory via Num, (f) Err via Memory.

Example 3. We compute the discrete Fourier transform of the data set $\mathbf{g}_N^d$ at $\mathbb{J}_{N,d}$ for $d = 2, 4, 6, 8$, with $\tilde{m} = 1$ by using Algorithm 5.6. This data set is obtained from sampling the functions

$$g_d(\mathbf{x}) := \cos\left(\sqrt[d]{\prod_{k\in\mathbb{Z}_d} x_k}\right), \quad \mathbf{x} \in I^d.$$

Functions g_d are nonperiodic continuous functions. The sampling points $v_{\mathbf{j},\mathbf{r}}$, for $\mathbf{j} \in \mathbb{S}_{N,d}$ and $\mathbf{r} \in \mathbb{Z}_{\mathbf{j},2}^d$, are generated from the refinable sets V_j with the initial set $V_0 := \{\frac{2\pi}{3}, \frac{4\pi}{3}\}$. We compare the results obtained from Algorithm 5.6 with those from Algorithm HFFT. The numerical results are shown in Figure 8.

From Figures 6, 7 and 8, we see that for a data set sampled from a function with a low regularity, Algorithm 5.6 performs better than Algorithm HFFT. In particular, from Figures 6, 7 and 8, we observe that when we use the same amount of computing time or the same number of the Fourier basis functions, Algorithm 5.6 achieves an accuracy of an order higher than that of Algorithm HFFT.

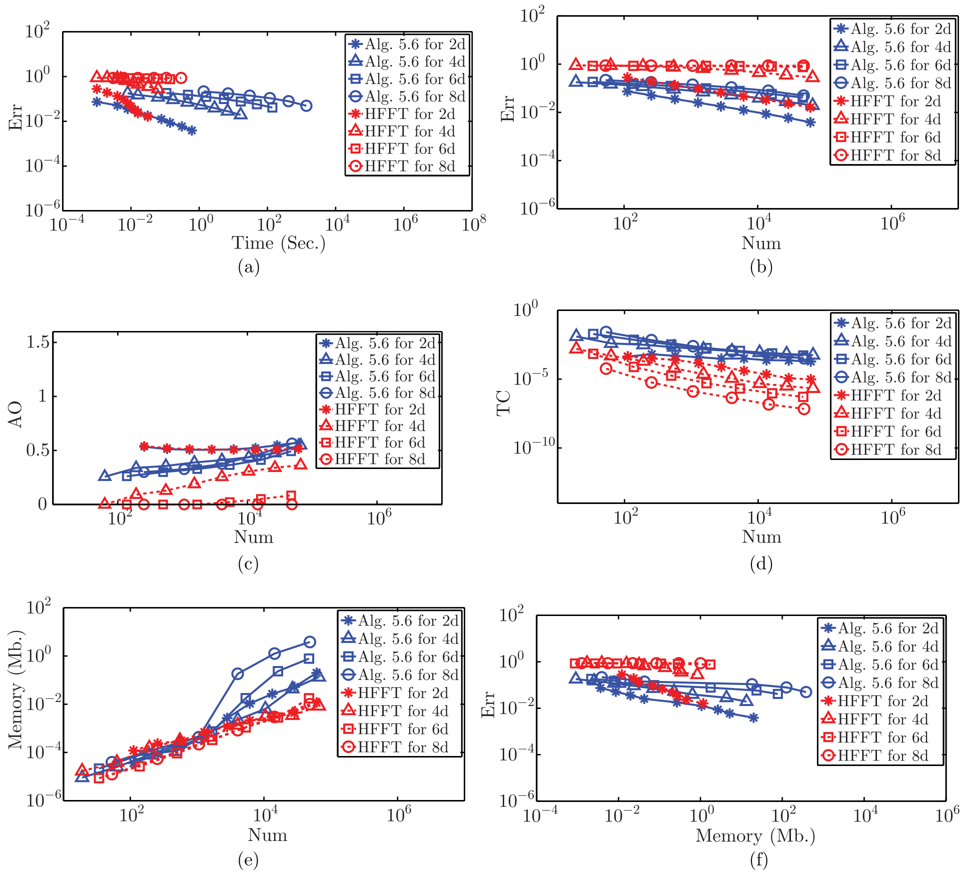


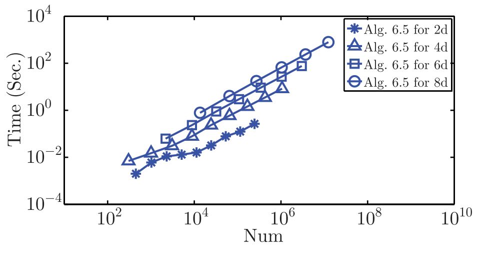
FIGURE 8. Example 3: (a) Err via Time, (b) Err via Num, (c) AO via Num, (d) TC via Num, (e) Memory via Num, (f) Err via Memory.

In the next example we demonstrate the computational efficiency of Algorithm 6.5 for computing the discrete backward Fourier transform.

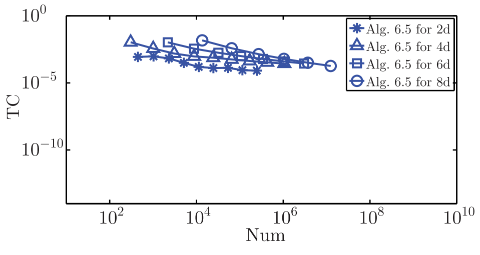
Example 4. We compute the discrete backward Fourier transform of the data set $\mathbf{c}_N^d$, for $d = 2, 4, 6, 8$, by using Algorithm 6.5. The data sets are obtained from sampling the Fourier coefficients of functions f_d considered in Example 1 at points $\mathbf{l} \in \mathbb{J}_{N,d}$ with $\tilde{m} = 1$. We consider the discrete backward Fourier transform on the grid set $\widetilde{S}_{N,d}$ that consists of the sampling points $v_{\mathbf{j},\mathbf{r}}$, for $\mathbf{j} \in \mathbb{S}_{N,d}$ and $\mathbf{r} \in \mathbb{Z}_{\mathbf{j},2}^d$, generated from the refinable sets $\widetilde{V}_j := \left\{\frac{2\pi r}{2^j 3} : r \in \mathbb{Z}_{2^j 3}\right\}$ which has $m = 3$.

The numerical results are shown in Figure 9. In (a) of Figure 9, we plot the values of Time for the cases $d = 2, 4, 6, 8$, where the vertical coordinate represents the value of Time. In (b) of Figure 9, we plot the values of TC for the cases $d = 2, 4, 6, 8$, where the vertical coordinate represents the value of TC. In (c) of Figure 9, we plot the values of Memory for the cases $d = 2, 4, 6, 8$, where the vertical coordinate represents the values of Memory. The horizontal coordinates in all of the images of Figure 9 represent the value of Num.

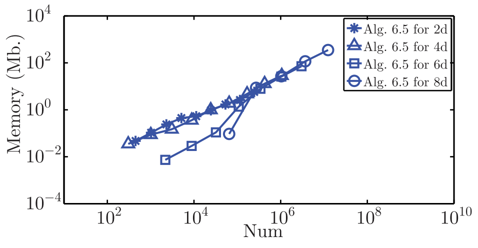
In (a), (b), (c), (d) of Figure 10, we plot the cardinality of the sparse grids $\mathbb{F}_{N,d}$ with $m = 1, 2, 3, 4$, respectively, which are used in Algorithm 5.6, for $d = 2, 4, 6, 8$. The set $\mathbb{F}_{N,d}$ with $m = 1$ corresponds to the data points used in Algorithm HFFT.



(a)



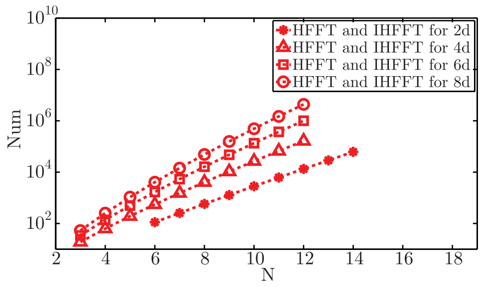
(b)



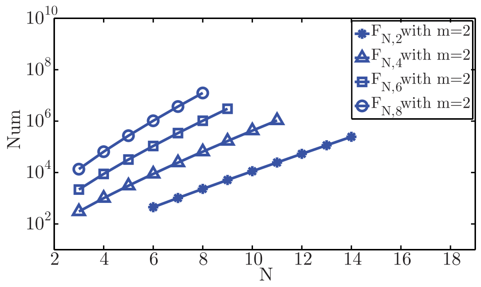
(c)

FIGURE 9. Example 4: (a) Time via Num, (b) TC via Num, (c) Memory via Num.

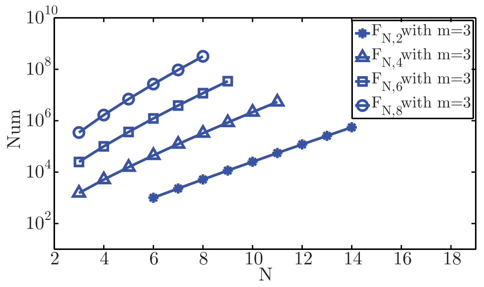
Finally, several comments on the comparison of Algorithm 6.5 to Algorithm IHFFT are in place. Algorithm 6.5 can be used to compute the discrete backward Fourier transform on sparse grids generated from the refinable sets $\widetilde{V}_j := \left\{\frac{2\pi r}{2^j m} : r \in \mathbb{Z}_{2^j m}\right\}$ for any fixed $m \in \mathbb{N}$. While Algorithm IHFFT in the form described in [11, 12, 14] can only deal with the case $m = 1$, although the algorithm may be modified to fit the cases $m = 2^k$ for a positive integer k. In the special cases $m = 2^k$ for a nonnegative integer k in which both algorithms are applicable, Algorithm 6.5 coincides with Algorithm IHFFT (in the case $m = 1$) and with its modified version (in the cases $m = 2^k$). When m is not a power of 2, it is shown in [21] that there exists a hierarchical basis whose zero set is the refinable set $\widetilde{V}_j$. Therefore, Algorithm IHFFT may be extended to the case when m is not a power of 2. To do this, we need to represent the Fourier expansion in the hierarchical basis. A fast algorithm that transforms between the Fourier basis and the hierarchical basis is needed. It may be nontrivial to construct such a fast algorithm. On the other hand, Algorithm 6.5 is applicable and convenient for implementation for the case when m is not a power of 2.



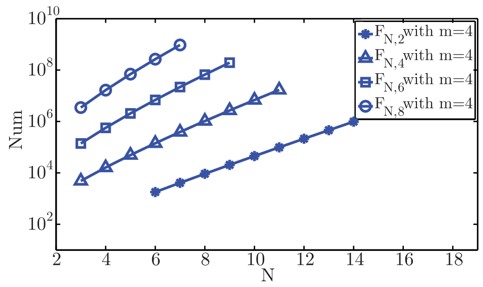
(a) $C(\mathbb{F}_{N,d})$ with $m = 1$



(b) $C(\mathbb{F}_{N,d})$ with $m = 2$



(c) $C(\mathbb{F}_{N,d})$ with $m = 3$



(d) $C(\mathbb{F}_{N,d})$ with $m = 4$

FIGURE 10. The cardinality of $\mathbb{F}_{N,d}$: image (a) the case $m = 1$, image (b) the case $m = 2$, image (c) the case $m = 3$, image (d) the case $m = 4$

ACKNOWLEDGMENT

The authors of this paper are grateful to the referee who brought [26] to their attention.

REFERENCES

[1] Robert Balder and Christoph Zenger, *The solution of multidimensional real Helmholtz equations on sparse grids*, SIAM J. Sci. Comput. **17** (1996), no. 3, 631–646, DOI 10.1137/S1064827593247035. MR1384255 (97h:65132)

[2] Kai Bittner, *Fast algorithms for periodic spline wavelets on sparse grids*, SIAM J. Sci. Comput. **20** (1999), no. 4, 1192–1213 (electronic), DOI 10.1137/S1064827596309098. MR1675469 (2000b:65018)

[3] Hans-Joachim Bungartz, *A multigrid algorithm for higher order finite elements on sparse grids*, Electron. Trans. Numer. Anal. **6** (1997), no. Dec., 63–77 (electronic). Special issue on multilevel methods (Copper Mountain, CO, 1997). MR1615156 (99a:65174)

[4] Hans-Joachim Bungartz and Michael Griebel, *Sparse grids*, Acta Numer. **13** (2004), 147–269, DOI 10.1017/S0962492904000182. MR2249147 (2007e:65102)

[5] Haotao Cai and Yuesheng Xu, *A fast Fourier-Galerkin method for solving singular boundary integral equations*, SIAM J. Numer. Anal. **46** (2008), no. 4, 1965–1984, DOI 10.1137/070703478. MR2399403 (2009c:65356)

[6] Zhongying Chen, Charles A. Micchelli, and Yuesheng Xu, *A construction of interpolating wavelets on invariant sets*, Math. Comp. **68** (1999), no. 228, 1569–1587, DOI 10.1090/S0025-5718-99-01110-2. MR1651746 (99m:65017)

[7] James W. Cooley and John W. Tukey, *An algorithm for the machine calculation of complex Fourier series*, Math. Comp. **19** (1965), 297–301. MR0178586 (31 #2843)

[8] R. A. Devor, P. P. Petrushev, and V. N. Temlyakov, *Multidimensional approximations by trigonometric polynomials with harmonics of a hyperbolic cross* (Russian, with Russian summary), Mat. Zametki **56** (1994), no. 3, 36–63, 158; English transl., Math. Notes **56** (1994), no. 3-4, 900–918 (1995). MR1309839 (96b:42001)

[9] Michael Döhler, Stefan Kunis, and Daniel Potts, *Nonequispaced hyperbolic cross fast Fourier transform*, SIAM J. Numer. Anal. **47** (2010), no. 6, 4415–4428, DOI 10.1137/090754947. MR2585193 (2011a:65466)

[10] Karin Frank, Stefan Heinrich, and Sergei Pereverzev, *Information complexity of multivariate Fredholm integral equations in Sobolev classes*, J. Complexity **12** (1996), no. 1, 17–34, DOI 10.1006/jcom.1996.0004. MR1386591 (97h:65167)

[11] V. Gradinaru, *Fourier transform on sparse grids: code design and the time dependent Schrödinger equation*, Computing **80** (2007), no. 1, 1–22, DOI 10.1007/s00607-007-0225-3. MR2308834 (2008k:65212)

[12] V. Gradinaru, *Strang splitting for the time-dependent Schrödinger equation on sparse grids*, SIAM J. Numer. Anal. **46** (2007/08), no. 1, 103–123, DOI 10.1137/050629823. MR2377257 (2009c:65249)

[13] M. Griebel and S. Knapek, *Optimized general sparse grid approximation spaces for operator equations*, Math. Comp. **78** (2009), no. 268, 2223–2257, DOI 10.1090/S0025-5718-09-02248-0. MR2521287 (2010f:65255)

[14] Klaus Hallatschek, *Fouriertransformation auf dünnen Gittern mit hierarchischen Basen* (German, with English and German summaries), Numer. Math. **63** (1992), no. 1, 83–97, DOI 10.1007/BF01385849. MR1182513 (93f:65014)

[15] Ying Jiang and Yuesheng Xu, *Fast discrete algorithms for sparse Fourier expansions of high dimensional functions*, J. Complexity **26** (2010), no. 1, 51–81, DOI 10.1016/j.jco.2009.10.001. MR2574572 (2011a:65467)

[16] Ying Jiang and Yuesheng Xu, *Fast Fourier-Galerkin methods for solving singular boundary integral equations: numerical integration and precondition*, J. Comput. Appl. Math. **234** (2010), no. 9, 2792–2807, DOI 10.1016/j.cam.2010.01.022. MR2652126 (2011f:65309)

[17] Ying Jiang and Yuesheng Xu, *B-spline quasi-interpolation on sparse grids*, J. Complexity **27** (2011), no. 5, 466–488, DOI 10.1016/j.jco.2011.03.003. MR2805532 (2012i:65014)

[18] S. Knapek, *Hyperbolic cross approximation of integral operators with smooth kernel*, Technical Report 665, SFB 256, Univ. Bonn, 2000.

[19] N. S. Nikol′skaja, *Approximation of differentiable functions of several variables by Fourier sums in the L_p metric* (Russian), Sibirsk. Mat. Ž. **15** (1974), 395–412, 461. MR0336221 (49 #997)

[20] S. V. Pereverzev, *Hyperbolic cross and complexity of an approximate solution of Fredholm integral equations of the second kind with differentiable kernels* (Russian), Sibirsk. Mat. Zh. **32** (1991), no. 1, 107–115, 221, DOI 10.1007/BF00970164; English transl., Siberian Math. J. **32** (1991), no. 1, 85–92. MR1112086 (92g:45014)

[21] Jie Shen and Haijun Yu, *Efficient spectral sparse grid methods and applications to high-dimensional elliptic problems*, SIAM J. Sci. Comput. **32** (2010), no. 6, 3228–3250, DOI 10.1137/100787842. MR2746619 (2012a:65354)

[22] V. Temlyakov, *Approximation of functions with bounded mixed derivative*, Trudy Mat. Inst. Steklov., 178 (1986), 1-112. MR0847439 (87j:42006)

[23] V. N. Temlyakov, *Approximation of periodic functions*, Computational Mathematics and Analysis Series, Nova Science Publishers Inc., Commack, NY, 1993. MR1373654 (96j:41001)

[24] Bo Wang, Rui Wang, and Yuesheng Xu, *Fast Fourier-Galerkin methods for first-kind logarithmic-kernel integral equations on open arcs*, Sci. China Math. **53** (2010), no. 1, 1–22, DOI 10.1007/s11425-010-0014-x. MR2594743 (2011d:65424)

[25] Yuesheng Xu and Aihui Zhou, *Fast Boolean approximation methods for solving integral equations in high dimensions*, J. Integral Equations Appl. **16** (2004), no. 1, 83–110, DOI 10.1216/jiea/1181075260. MR2093500 (2005e:65219)

[26] Andreas Zeiser, *Fast matrix-vector multiplication in the sparse-grid Galerkin method*, J. Sci. Comput. **47** (2011), no. 3, 328–346, DOI 10.1007/s10915-010-9438-2. MR2793587 (2012j:65435)

Guangdong Province Key Lab of Computational Science, School of Mathematics and Computational Science, Sun Yat-sen University, Guangzhou 510275, P. R. China
E-mail address: yjiang80@gmail.com

Guangdong Province Key Lab of Computational Science, School of Mathematics and Computational Science, Sun Yat-sen University, Guangzhou 510275, P. R. China
Current address: Department of Mathematics, Syracuse University, Syracuse, New York 13244
E-mail address: yxu06@syr.edu