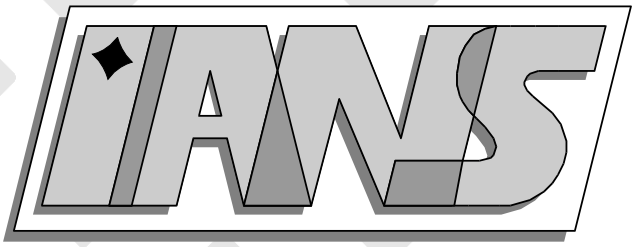
Universität Stuttgart



Efficient Construction of Hierarchical Polynomial Sparse Grid Interpolants using the Fast Discrete Cosine Transform

Andreas Klimke

Berichte aus dem Institut für Angewandte Analysis und Numerische Simulation

Preprint 2006/007

Universität Stuttgart

Efficient Construction of Hierarchical Polynomial Sparse Grid Interpolants using the Fast Discrete Cosine Transform

Andreas Klimke

Berichte aus dem Institut für Angewandte Analysis und Numerische Simulation

Preprint 2006/007

Institut für Angewandte Analysis und Numerische Simulation (IANS)
Fakultät Mathematik und Physik
Fachbereich Mathematik
Pfaffenwaldring 57
D-70 569 Stuttgart

E-Mail: ians-preprints@mathematik.uni-stuttgart.de
WWW: http://preprints.ians.uni-stuttgart.de

ISSN **1611-4176**

Efficient Construction of Hierarchical Polynomial Sparse Grid Interpolants using the Fast Discrete Cosine Transform

Andreas Klimke

13th June 2006

Abstract

To approximate smooth multivariate functions, sparse grid interpolation is superior to full tensor-product grid interpolation. The order of the convergence rate in the maximum norm is preserved up to a logarithmic factor although the number of nodes is reduced significantly. A particularly powerful method is achieved by employing Lagrange polynomials at the Chebyshev Gauss-Lobatto nodes as kernel functions of the multivariate interpolant. In this paper, we describe how the fast discrete cosine transform (or the FFT) can be used in the hierarchical construction process to significantly improve the computational complexity of the standard algorithm using the barycentric form.

1 Introduction

We consider the problem of obtaining an interpolant (and an approximation) of an expensive objective model in order to efficiently apply other procedures, such as an optimization, a Monte Carlo analysis, a fuzzy uncertainty analysis, or an active control algorithm. Sparse grid interpolation[1] offers a suitable tool to obtain such surrogate functions, especially in case of higher-dimensional problems [8, ch. 3], [10].

Standard sparse grid interpolants are constructed from model evaluations at grid nodes that are selected a priori. By employing a hierarchical construction scheme [9], it becomes possible to successively refine the interpolant until a given error tolerance is reached. The most important property of the sparse grid method is the fact that the asymptotic error decay of full grid interpolation with increasing grid resolution is preserved up to a logarithmic factor in case of sufficiently smooth objective functions. Furthermore, the construction as well as the evaluation of the interpolant can be done very efficiently even in the high-dimensional case by employing suitable dimension-adaptive algorithms [6] and a data structure exploiting sparsity [8, ch. 3].

Traditionally, sparse grids use piecewise linear basis functions as kernels. However, as demonstrated by numerous publications (see, e.g., [1, 4, 12, 14]), improved accuracies can be obtained when using polynomial, spline, or trigonometric basis functions provided that the objective model is smooth enough. In this paper, we focus on the so called Chebyshev-Gauss-Lobatto

[1] The sparse grid concept is not limited to the problem of interpolation and approximation. A recent survey article [4] provides a detailed review of the fundamentals and application areas.

sparse grid employing arbitrary-degree polynomials with global support, as thoroughly discussed in [1] and [8, ch. 3]. A novel approach to the construction of the interpolant is presented that achieves a significant reduction of the computational complexity particularily in models where very high-degree polynomials are needed to approximate the objective model. In this case, the computational complexity of the hierarchical construction algorithm is not optimal, since computing the hierarchical surpluses for newly added grid points is computed by subtracting the value of the interpolant at the previous refinement step from the function value at the new grid point. The overal computational complexity may get as high as $\mathcal{O}(N^2)$ when mainly a single dimension is involved in the refinement process. Here, we illustrate how the fast discrete cosine transform (fast DCT) can be used to overcome this limitation due to the well-known properties of these transforms that reduce the cost of computing the spectral coefficients from $\mathcal{O}(n^2)$ to $\mathcal{O}(n \log n)$.

The rest of the paper is organized as follows. In Section 2, we illustrate the main ideas of hierarchical Lagrange polynomial interpolation scheme based on the DCT in 1D. In Section 3, we describe how to generalize the approach to the multivariate sparse grid case consisting of a sum of tensor products. Thereafter, we provide a complexity estimation of our new algorithm. Section 4 presents numerical results obtained with our reference implementation, the Sparse Grid Interpolation Toolbox available for MATLAB.

2 Hierarchical Lagrange polynomial interpolation in 1D

2.1 Nodal representation and the barycentric formula

In this section, we prepare the basic tools needed for the construction of the hierarchical polynomial sparse grid interpolants discussed in Section 3. We start with the one-dimensional case, since sparse grid interpolation uses univariate interpolation formulas

$$U^i(f) = \sum_{x^i \in X^i} a_{x^i} \cdot f(x^i) \tag{1}$$

as building blocks that are extended to the multivariate case via tensor products. In Eq. (1), the set X^i shall denote the set of support nodes $X^i = \{x^i \mid x^i \in [-1, 1]\}$. The a_{x^i} are the basis functions satisfying $a_{x^i}(x^i) = 1$, $a_{x^i}(y^i) = 0$ for all $y^i \in X^i, x^i \neq y^i$. The above formula also contains an index $i \in \mathbb{N}$ that will be used below to construct a nested sequence of such interpolation formulas. Furthermore, we denote the number of support nodes by m_i, i.e., the cardinality of the set X^i is $\#X^i = m_i$.

In this paper, we are concerned with real-valued objective functions f that are at least continuously differentiable. Furthermore, we restrict the basis functions to the case of Lagrange polynomials at the Chebyshev Gauss-Lobatto (CGL) nodes (the extrema of the Chebyshev polynomials). We thus define the sets $X^i = \{x^i_j \mid j = 1, \ldots, m_i\}$ by

$$m_i = \begin{cases} 1 & \text{if } \; i = 1, \\ 2^{i-1} + 1 & \text{if } \; i > 1. \end{cases} \tag{2}$$

$$x^i_j = \begin{cases} (-\cos(\pi(j-1)/(m_i - 1)) & \text{for } j = 1, \ldots, m_i \;\text{ if }\; m_i > 1, \\ 0 & \text{for } j = 1 \;\text{ if }\; m_i = 1, \end{cases} \tag{3}$$

and the basis functions are the Lagrange characteristic polynomials (illustrated for the case $i = 3$ in Fig. 2(a)):

$$a_{x_1^1}(x) = 1 \quad \text{for } i = 1, \text{ and}$$

$$a_{x_j^i}(x) = \prod_{\substack{k=1 \\ k \neq j}}^{m_i} \frac{x - x_k^i}{x_j^i - x_k^i} \quad \text{for } i > 1 \text{ and } j = 1, \ldots, m_i. \tag{4}$$

If suitable sets of support nodes are used, there exists a very efficient and stable method to evaluate polynomial interpolants called barycentric Lagrange interpolation. A thorough review of the method and its properties can be found here [3]. A stability analysis of this interpolation formula was presented by Higham [7]. By employing the barycentric formula (of the second form), and with the a_{x^i} chosen as Lagrange polynomials according to Eq. (4), Eq. (1) can be expressed as

$$U_{\mathrm{B}}^i(f)(x) = \frac{\displaystyle\sum_{j=1}^{m_i} \frac{b_j^i}{x - x_j} f(x_j^i)}{\displaystyle\sum_{j=1}^{m_i} \frac{b_j^i}{x - x_j^i}}, \tag{5}$$

where the capital letter B denotes the barycentric representation of U^i. Most remarkably, the method is of $\mathcal{O}(m_i)$ complexity once the barycentric weights b_j^i appearing in both in the numerator and the denominator are known. While these weights must be computed in case of arbitrary nodes requiring algorithms of quadratic complexity, the weights for the CGL nodes are simply given by

$$b_1^i = \frac{1}{2}; \quad b_j^i = -(-1)^j, \ j = 2, \ldots, m_i - 1; \quad b_{m_i}^i = -\frac{1}{2}(-1)^{m_i}, \tag{6}$$

and thus, do not need to be pre-computed and stored regardless of the number of nodes m_i.

2.2 Hierarchical decomposition

Another important property of the CGL nodes is the subset property (nesting) $X^i \subset X^{i+1}$. This allows us to transform the nodal univariate interpolation formula $U^i(f)$ from Eq. (1) into an equivalent hierarchical decomposition, since U^i can exactly represent U^{i-1}, i.e., $U^i(U^{i-1}(f)) = U^{i-1}(f)$, as shown in [9, Section 2.5]:

$$U_{\mathrm{H}}^i(f) = \sum_{k=1}^{i} \Delta^k(f), \tag{7}$$

where

$$\Delta^k(f) = \sum_{j=1}^{m_k^\Delta} a_j^k \cdot \underbrace{\left(f({}^\Delta x_j^k) - U^{k-1}(f)({}^\Delta x_j^k) \right)}_{:= w_j^k(f)}, \tag{8}$$

In Eqs. (7) and (8), the index k denotes the *hierarchical level*. With increasing level, new support nodes are added to the interpolant. These new support nodes are given by ${}^{\Delta}x_j^k \in X_\Delta^k$, $X_\Delta^k = X^k \setminus X^{k-1}$, with $X^0 = \emptyset$. Consequently, the number of support nodes in X_Δ^k is denoted by $m_k^\Delta = \# X_\Delta^k$. Only the basis functions $a_{{}^{\Delta}x_j^k}$ where ${}^{\Delta}x_j^k$ is included in X_Δ^k are used by Eq. (8). For simplicity, the basis functions $a_{{}^{\Delta}x_j^k}$ with $a_{{}^{\Delta}x_j^k}({}^{\Delta}x_j^k) = 1$ are denoted by a_j^k, $j = 1, \ldots, m_k^\Delta$. See Fig. 2(b) for an illustration of the hierarchical basis functions a_j^k.

We can still apply the efficient barycentric formula to the hierarchical form, obtaining the *barycentric-hierarchical* form

$$U_{\mathrm{B,H}}^i(f)(x) = \sum_{k=1}^{i} \left[\frac{\displaystyle\sum_{j=1}^{m_k^\Delta} \frac{{}^{*}b_j^k}{x - {}^{\Delta}x_j^k} w_j^k(f)}{\displaystyle\sum_{j=1}^{m_k} \frac{b_j^k}{x - x_j^k}} \right], \tag{9}$$

where

$${}^{*}b_j^k = \begin{cases} b_{2j-1}^k, & k = 1, 2,\ j = 1, \ldots, m_k^\Delta, \\ b_{2j}^k, & k > 2,\ j = 1, \ldots, m_k^\Delta. \end{cases}$$

Note that while the sum in the nominator runs over the set of support nodes ${}^{\Delta}x_j^k \in X_\Delta^k$ where the nodes that occurred in a previous nesting level are omitted, the sum in the denominator runs over the full set of support nodes $x_j^k \in X^k$. It is trivial to show that the complexity of computing interpolated values with Eq. (9) remains of order $\mathcal{O}(m^i)$ with a factor of about 2 compared to the standard barycentric form given by Eq. (5).

What have we gained by introducing this hierarchical scheme of Eqs. (7) and (8) compared to Eq. (1) that appears to be much simpler? The first advantage is that no longer function values are used directly in the interpolation formula, but rather the *hierarchical surpluses* w_j^k that tend to zero with increasing level k (as the approximation approaches the objective function), thus providing a natural stopping criteria when an interpolant of a user-specified accuracy is to be constructed by an automatic algorithm. The second advantage is the fact that we can use this decomposition to form a sparse grid in the multivariate case, where one-dimensional difference formulas Δ^k of different hierarchical level are combined suitably via tensor products (see Section 3). This is due to the fact that a hierachical interpolant U^i has *invariant* weights $w_j^k, k = 1, \ldots, i-1, j = 1, \ldots, m_k$ (i.e., the interpolation formula just adds some additional weights $w_j^i, j = 1, \ldots, m_i$ to the existing ones that stay the same).

Unfortunately, there is also a serious disadvantage: Straightforward computation of the hierarchical surpluses yields work of order $\mathcal{O}(m_i^2)$, since the evaluation of the polynomials U^{k-1} in Eq. (8) requires a linear-complexity algorithm (e.g., if the barycentric formula is used), and this has to be done for each hierarchical surplus w_j^k. In the following, we will review the Chebyshev expansion, which will help to overcome this limitation.

Example 1 Consider the univariate function : $[-1, 1] \to \mathbb{R}$, $(x) = \sin(5x + 1/2) + \exp(x)$. In Fig. 1, four subsequent approximations U^i, $i = 1, \ldots, 4$ are shown, using 1, 3, 5, and 9 support nodes, respectively. The figure illustrates the way the hierarchical surpluses are obtained by computing the difference of the function value $f({}^{\Delta}x_j^i)$ at a node ${}^{\Delta}x_j^i$ from the interpolated value $U^{i-1}({}^{\Delta}x_j^i)$.

2.3 Chebysev expansion

We can also expand the polynomial into a Chebyshev series, thus arriving at a spectral decomposition of the following form:

$$U_{\mathrm{T}}^i(f) = \sum_{j=1}^{m_i}{}'' T_{j-1} \cdot c_j^i(f), \tag{10}$$

where $T_0, \ldots, T_{m_i-1}$ are the Chebyshev polynomials of the first kind (the Chebyshev basis polynomials are illustrated in Fig. 2(c)). The spectral coefficients $c_j^i(f)$ are given by

$$c_j^i(f) = \frac{2}{m_i - 1} \sum_{l=1}^{m_i}{}'' T_{j-1}(x_l^i) f(x_l^i), \quad j = 1, \ldots, m_i. \tag{11}$$

The $(''$) on the summation means that the first and last terms are halved. Note that Eqs. (10) and (11) are not defined for $i = 1$. For $i = 1$, we just have $U_T^1(f) = T_0 \cdot c_1^1$ with $c_1^1(f) = f(x_1^1)$.

Properties of the Chebyshev expansion This representation shares the advantage of the hierarchical decomposition in providing a way to monitor the convergence towards the objective function, since the spectral coefficients $c_j^i(f)$ decay with increasing number of support nodes (provided that f is sufficiently smooth and m_i is large enough to recover the oscillations of f). But while a hierachical interpolant U_{H}^i has invariant weights $w_j^{i-1}(f), j = 1, \ldots, m_{i-1}$, the Chebyshev spectral coefficients $c_j^i(f)$ of an interpolant U_{T}^i may change for all indices $j = 1, \ldots, m_{i-1}$ compared to the coefficients $c_j^{i-1}(f)$ of the preceding interpolant U_{T}^{i-1}. This results from the different structure of the Chebyshev basis polynomials, which do no longer satisfy equality to 0 or 1 at all support nodes, cf. Fig. 2(c). Thus, the Chebyshev expansion is not directly applicable in a hierarchical sparse grid setting, where the difference formulas $\Delta(f) = U^i(f) - U^{i-1}(f)$ are needed; these cannot be immediately derived from the spectral coefficients.

Another very important property of the Chebyshev expansion lies in the possibility of applying the fast DCT to very efficiently compute the coefficients in $\mathcal{O}(m_i \log(m_i))$ complexity. This is of course much better than the quadratic complexity required to compute the hierarchical surpluses.

Example 2 Consider the same function as in Ex. 1. Table 1 compares the hierarchical surpluses to the spectral coefficients of the interpolants $U_{\{\mathrm{H,T}\}}^i$ of level $i = 1, \ldots, 3$. Note that the surpluses of previously computed interpolants do not change, but the spectral coefficients do change. To illustrate the decay of the surpluses and the coefficients with increasing number of nodes, we have included Table 2.

Computing the Chebyshev coefficients via FFT or fast DCT By definition, the Chebyshev polynomials satisfy

$$T_j(x) = \cos(j \arccos x), \quad j = 0, \ldots, m_i - 1, \tag{12}$$

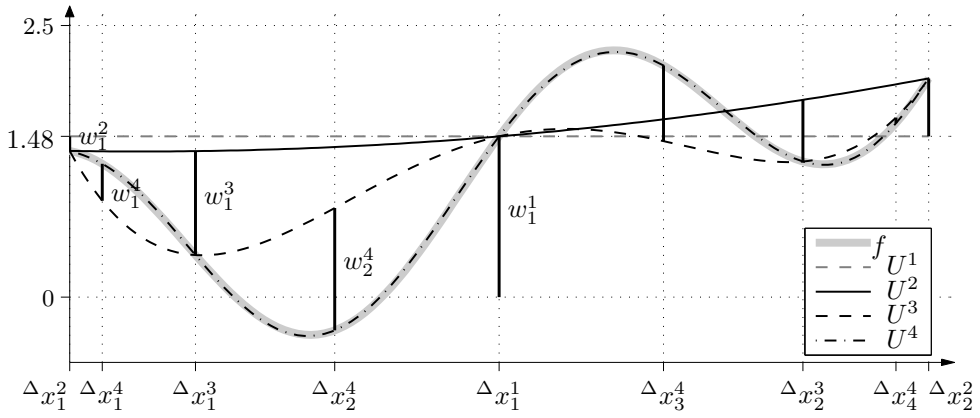


Figure 1: Example 1: Hierarchical construction of interpolants U^i of f.

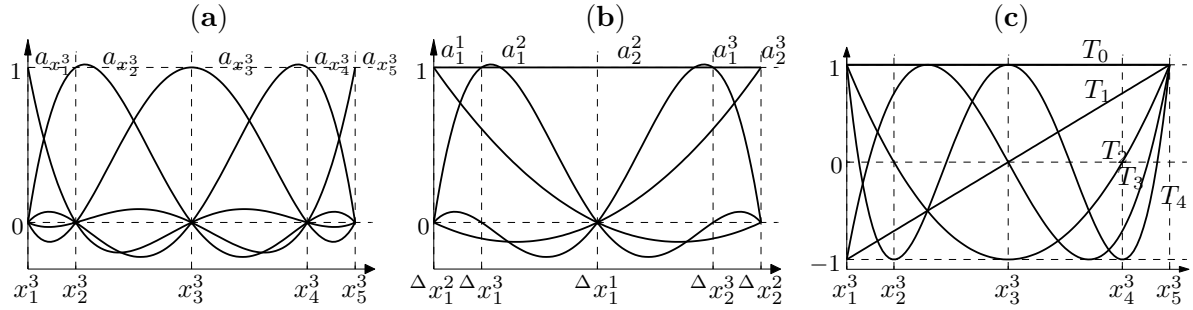


Figure 2: (**a**): Nodal basis functions $a_{x_j^3}$, $x_j^3 \in X^3$; (**b**): hierarchical basis functions a_j^i, ${}^\Delta x_j^i \in X_\Delta^i$, $i = 1, \ldots, 3$; (**c**) Chebyshev polynomials T_{j-1}, $x_j^3 \in X^3$.

Table 1: Ex. 2, U_{H}^i vs. U_{T}^i: Comparison of surpluses and coefficients.

U_{H}^1	U_{H}^2	U_{H}^3	U_{T}^1	U_{T}^2	U_{T}^3
$w_1^1 = 1.48$	$w_1^1 = 1.48$	$w_1^1 = 1.48$	$c_1^1 = 1.48$	$c_1^2 = 3.16$	$c_1^3 = 2.40$
	$w_1^2 = -0.13$	$w_1^2 = -0.13$		$c_2^2 = 0.33$	$c_2^3 = 0.47$
	$w_2^2 = 0.53$	$w_2^2 = 0.53$		$c_3^2 = 0.20$	$c_3^3 = 0.10$
		$w_1^3 = -0.96$			$c_4^3 = -0.14$
		$w_2^3 = -0.57$			$c_5^3 = 0.76$

Table 2: Ex. 2, U_{H}^i vs. U_{T}^i: Comparison of decay.

i	2	3	4	5	6	7	8
$\max_{j=1,\ldots,m_i^\Delta} \lvert w_j^i(f_1)\rvert$	0.53	0.96	1.13	0.021	4.2e–8	8.9e–16	6.7e–16
$\max_{j=m_{i-1}+1,\ldots,m_i} \lvert c_j^i(f_1)\rvert$	0.33	0.76	0.46	0.010	2.0e–8	1.4e–16	1.1e–16

and thus, at the support nodes x^i_{l+1}, $l = 0, \ldots, m_i - 1$, we have

$$T_j(x^i_{l+1}) = \cos(\pi j(1 - l/(m_i - 1))). \tag{13}$$

Substituting Eq. (13) into Eq. (11), we obtain the expression

$$c^i_{j+1}(f) = \frac{2}{m_i - 1} \sum_{l=0}^{m_i-1}{}'' f(x^i_{l+1}) \cos\left(\frac{\pi j(m_i - (l+1))}{(m_i - 1)}\right).$$

By reversing the order that the elements of the sum are processed, this can be simplified to

$$c^i_{j+1}(f) = \frac{2}{m_i - 1} \sum_{l=0}^{m_i-1}{}'' f(x^i_{m_i-l}) \cos(\pi jl/(m_i - 1)) \quad := \frac{2}{m_i - 1} \mathrm{DCT}_j(Y^i), \tag{14}$$

where $j = 0, \ldots, m_i - 1$, $i > 1$, $Y^i = \{f(x^i_{m_i-l}) \mid x^i_{m_i-l} \in X^i,\ l = 0, \ldots, m_i - 1\}$, or, using vector notation, and introducing a "flip" operation that reverses the order of the elements of a vector (e.g., $\mathrm{flip}[(1, 2, 3)] = (3, 2, 1)$),

$$\mathbf{c}^i(f) = 2 \cdot (m_i - 1)^{-1} \cdot \mathrm{DCT}(\mathrm{flip}[f(\mathbf{x}^i)]). \tag{15}$$

The operator $\mathrm{DCT}(\cdot)$ is the discrete cosine transform of type 1 (DCT-I). Several authors have suggested efficient algorithms implementing the DCT-I using suitably modified FFT algorithms (see, e.g., [11, ch. 12]). Note that the inverse of the DCT-I is just the DCT-I multiplied by $2/(m_i - 1)$. Using this inverse operation, we easily obtain

$$\begin{aligned} \mathrm{flip}[f(\mathbf{x}^i)] &= 2 \cdot (m_i - 1)^{-1} \cdot \mathrm{DCT}\left(2^{-1} \cdot (m_i - 1) \cdot \mathbf{c}^i(f)\right), \\ f(\mathbf{x}^i) &= \mathrm{flip}[\mathrm{DCT}((\mathbf{c}^i(f))]. \end{aligned} \tag{16}$$

The DCT-I is not always readily available, or, on many platforms, highly optimized FFT implementations may surpass the performance of a non-optimized DCT-I algorithm. Therefore, we briefly review a procedure referred to as "force-fitting" [11, ch. 12] that allows to implement the DCT-I via a conventional FFT. We start with the DCT-I given by

$$\mathrm{DCT}_j(Y) = \sum_{l=0}^{n}{}'' f_l \cos(\pi jl/n), \tag{17}$$

where $n \geq 2$, $j = 0, \ldots, n$, $Y = \{f_0, \ldots, f_n\}$. The discrete values are then extended to represent an even function about $l = n$ such that $f_{n-l} = f_{n+l}$, $l = 0, \ldots, n - 1$. Applying the discrete Fourier transform to these extended values, we have for $j = 0, \ldots, 2n - 1$ and $Y^* = \{f_0, \ldots, f_{2n-1}\}$

$$F_j(Y^*) = \sum_{l=0}^{2n-1} f_l \; e^{2\pi\sqrt{-1}jl/2n}, \tag{18}$$

which can be computed efficiently by a standard FFT algorithm. From suitably splitting the sum into two parts

$$F_j(Y^*) = \sum_{l=0}^{n-1} f_l e^{\pi\sqrt{-1}jl/n} + \sum_{l=n}^{2n-1} f_l e^{\pi\sqrt{-1}jl/n}, \tag{19}$$

applying $f_{n-l} = f_{n+l}$, and using $\frac{1}{2}(e^{\sqrt{-1}\theta} + e^{-\sqrt{-1}\theta}) = \cos\theta$, it follows that

$$F_j(Y^*) = 2\sum_{l=0}^{n}{}'' f_l \cos(\pi jl/n) = 2\,\mathrm{DCT}_j(Y), \quad j = 0, \ldots, n. \tag{20}$$

In summary, computing the FFT of the extended data will return the same values as the DCT for $j = 0, \ldots, n$ multiplied by a factor of two.

2.4 Combining the advantages

In this section, we describe how the Chebyshev expansion can be used to eliminate the disadvantage of the expensive computation of the hierarchical surpluses in $\mathcal{O}(m_i^2)$.

With the fast DCT, we can compute the coefficients c_j^i from Eq. (11) in $\mathcal{O}(m_i \log m_i)$ complexity. Similarly, we can compute the Chebyshev expansion for each difference formula $\Delta^k(f)$ from Eq. (8), giving

$$\Delta_{\mathrm{T}}^k(f) = \sum_{j=1}^{m_k}{}'' T_{j-1} \cdot c_j^k(w^k(f)), \tag{21}$$

with the coefficients

$$c_j^k(w^k(f)) = \frac{2}{m_k - 1}\sum_{l=1}^{m_k}{}'' T_{j-1}(x_l^k) \cdot {}^*w_l^k(f), \tag{22}$$

where the hierarchical surpluses ${}^*w_l^k$ are given by

$${}^*w_l^k = \begin{cases} w_{(l+1)/2}^k, & k = 1, 2,\ l = 1, \ldots, m_k, \\ 0 & \text{j even },\ k = 2,\ l = 2, \\ 0 & \text{j odd },\ k > 2,\ l = 1, \ldots, m_k, \\ w_{l/2}^k, & k > 2,\ l = 2, \ldots, m_k - 1. \end{cases} \tag{23}$$

Note that in Eqs. (21) and (22), the index j runs over $1, \ldots, m_k$, not over $1, \ldots, m_k^\Delta$ as in Eq. (8). The missing values ${}^*w_j^k$ compared to the w_j^k are set equal to zero (since there is no hierarchical contribution at the support nodes of the difference formulas $\Delta^1, \ldots, \Delta^{k-1}$ in the difference formula Δ^k). Although the terms equal to zero could be omitted in Eq. (22), we use this notation to maintain the form of Eq. (11), allowing immediate application of the DCT to compute the coefficients.

With Eqs. (21), (22), and (23), we can thus transform the set of hierarchical surpluses of each difference formula Δ^k into a corresponding set of Chebyshev coefficients. But what we are really interested in is *evaluating* an interpolation formula $U^{k-1}(f)$ at the support nodes ${}^\Delta x_j^k \in X_\Delta^k$, since this part is responsible for the quadratic complexity in the computation of the hierarchical surpluses $w_j^k(f)$ in Eq. (8). Due to Eq. (7), this essentially boils down to evaluating a difference formula $\Delta^k(f)$ for any set of support nodes X_Δ^k, $i > k$. To achieve this, a technique can be used that is referred to as "upsampling" in signal processing [2]. The procedure is explained in the following.

Let $i > k$. Furthermore, let ${}^*\mathbf{c}^k(w^k) = \left({}^*c_1^k(w^k(f)), \ldots, {}^*c_{m_i}^k(w^k(f))\right)$ be an extended vector with

$$ {}^*c_j^k(w^k(f)) = \begin{cases} c_j^k(w^k(f)) & j = 1, \ldots, m_k, \\ 0 & j = m_k + 1, \ldots, m_i. \end{cases} \tag{24} $$

Then, by applying the inverse DCT from Eq. (16) to the vector, we obtain

$$ \Delta_{\mathrm{T}}^k(\mathbf{x}^i) = \mathrm{flip}[\mathrm{DCT}({}^*\mathbf{c}^k(w^k))]. \tag{25} $$

The procedure is called "upsampling", since we have used the Chebyshev coefficients obtained via the DCT applied to "sampled" values (at a coarse grid X^k) to obtain interpolated values via extension of the coefficient data and the according inverse transform (at a fine grid X^i).

We have now assembled all the tools to formulate the first algorithm of this paper. Algorithm 1 computes the hierarchical surpluses $w_j^i(f)$ of an interpolant $U_{\mathrm{H}}^i(f)$, or, in other words, constructs a hierarchical polynomial interpolant of f via DCT in one dimension. The upsampling process is performed by the subroutine `dctupsample`, Algorithm 2.

Algorithm 1 `adaptvalsdct`

In: $f : [-1, 1] \to \mathbb{R}$: Objective function.
δ_{rel}, δ_{abs}: Relative and absolute error tolerance.
$n_{\max} \in \mathbb{N}$: Max. # of nodes.
Out: Z_i: Ordered set of hierarchical surpluses.

Let $e_{\mathrm{est}} = \infty$, $i = 1$, $n = 1$, ${}^{\Delta}\mathbf{x}^1 = (x_1^1)$.
Let $\mathbf{w}^1 = (f(x_1^1))$, $y_{\max} = f(x_1^1)$, $y_{\min} = f(x_1^1)$.
While $e_{\mathrm{est}} \geq \max\{\delta_{\mathrm{rel}} \cdot (y_{\max} - y_{\min}),\ \delta_{\mathrm{abs}}\}$ **And** $n < n_{\max}$ **Do**
  Let $i = i + 1$.
  Let ${}^{\Delta}\mathbf{x}^i = (x_1^i, \ldots, x_{m_i^{\Delta}}^i)$ with $x_j^i \in X_{\Delta}^i$.
  Let $\mathbf{y} = f({}^{\Delta}\mathbf{x}^i)$.
  Let $y_{\max} = \max\{y_{\max}, \mathbf{y}\}$, $y_{\min} = \min\{y_{\min}, \mathbf{y}\}$.
  For $k = 1$ **To** $i - 1$ **Do**
    Let $\Delta^k({}^{\Delta}\mathbf{x}^i) =$ `dctupsample`$(\mathbf{w}^k, i, k)$.
    Let $\mathbf{y} = \mathbf{y} - \Delta^k({}^{\Delta}\mathbf{x}^i)$.
  End For
  Let $\mathbf{w}^i = \mathbf{y}$, $n = n + m_i$, $e_{\mathrm{est}} = \max\limits_{j=1,\ldots,m_i^{\Delta}} |w_j^i|$.
End While
Let $Z_i = \{\mathbf{w}^1, \ldots, \mathbf{w}^i\}$.

The complexity of Algorithm 1 is $\mathcal{O}(m_i \log^2 m_i)$, and thus, by a factor or order $\mathcal{O}(\log m_i)$ slower than computing the Chebyshev coefficients directly for $U_{\mathrm{T}}^i(f)$. This is the price we pay for the hierarchical decomposition, since we have to perform the upsampling procedure i times that results in the additional factor $\mathcal{O}(\log m_i)$.

Remark 1. The evaluation of the interpolant can be performed efficiently in $\mathcal{O}(m_i)$ using the barycentric-hierarchical form from Eq. (9), as discussed in Section 2.2. Note that an implementation must prevent the evaluation of the interpolant at the support nodes, which can be easily avoided by a conditional statement prior to the computation of Eq. (9), see [3, 8].

Algorithm 2 `dctupsample`$(\mathbf{w}^k, i, k)$

In: $\mathbf{w}^k$: Vector of hierarchical surpluses of level k.
k, i: Base and target level index.
Out: $\Delta^k({}^{\Delta}\mathbf{x}^i)$: Interpolated values of difference function Δ^k at the grid points ${}^{\Delta}\mathbf{x}^i$.

1: Let ${}^*\mathbf{w}^k = ({}^*w_1^k, \ldots, {}^*w_{m_k}^k)$, where ${}^*w_j^k$ according to Eq. (23).
2: Let $\mathbf{c}^k = 2 \cdot (m_k - 1)^{-1} \cdot \mathrm{DCT}(\mathrm{flip}[{}^*\mathbf{w}^k])$.
3: Let ${}^*\mathbf{c}^k = (c_1^k, \ldots, c_{m_k}^k, 0, \ldots, 0)$. $\{\textit{append } m_i - m_k \textit{ zeros}\}$
4: Let $\Delta_{\mathrm{T}}^k(\mathbf{x}^i) = \mathrm{flip}[\mathrm{DCT}({}^*\mathbf{c}^i)]$.
5: Let $\Delta_{\mathrm{T}}^k({}^{\Delta}\mathbf{x}^i) = \begin{cases} \Delta_{\mathrm{T}}^k\big((x_1^i, x_3^i)\big), & i = 2, \\ \Delta_{\mathrm{T}}^k\big((x_2^i, x_4^i, \ldots, x_{m_i-1}^i)\big), & i > 2. \end{cases}$

Remark 2. Note that in the univariate case, it is possible to formulate an algorithm that computes the hierarchical surpluses in $\mathcal{O}(m_i \log m_i)$ by replacing the steps performed in the inner loop (lines 8-11) by directly upsampling $U^{i-1}(f)({}^{\Delta}\mathbf{x}^{i-1})$ to $U^{i-1}(f)({}^{\Delta}\mathbf{x}^i)$ and subtracting the result from $\mathbf{y}$. But this simplification does not extend in general to the multivariate case, and is therefore not further studied in the context of this article.

3 The multivariate case

3.1 Full tensor products

One-dimensional, nodal interpolation formulas according to Eq. (1) can extended easily to the $d-$variate case by using tensor products such that

$$(U^i \otimes \cdots \otimes U^i)(f) = \sum_{x^{i_1} \in X^{i_1}} \cdots \sum_{x^{i_d} \in X^{i_d}} (a_{x^{i_1}} \otimes \cdots \otimes a_{x^{i_d}}) \cdot f(x^{i_1}, \ldots, x^{i_d}), \tag{26}$$

where $f : [-1, 1]^d \to \mathbb{R}$, and for the scalar functions a_k, the tensor product is just defined as $(a_1 \otimes \cdots \otimes a_d)(x_1, \ldots, x_d) = \prod_{k=1}^d a_k(x_k)$. From an algorithmic point of view, employing tensor products offers the advantage that the implementation is straightforward due to their one-dimensional building blocks. In particular, the one-dimensional techniques generalize very well to the multivariate case, such as the barycentric form (see [8, Algorithm 3] for the multivariate case), the Chebyshev expansion form with its fast DFT/FFT algorithms (see, e.g., [11, ch. 12]). Similarly, we could adapt Algorithm 1, our new algorithm for the efficient computation of hierarchical surpluses in 1D, to full tensor product grids.

However, using full tensor product grids has a significant disadvantage. With increasing problem dimension d, the number of support nodes N grows exponentially, i.e., $N = n^d$ if the number of support nodes in each dimension is n. This renders the application of full tensor product grids impractical for most real-world applications, where the objective models are often both high-dimensional ($d \gg 10$) and also expensive to evaluate.

3.2 Sparse grids

Preliminaries One can overcome this limitation to a large extend with sparse grids. Sparse grid interpolation use a sequence of tensor products of univariate interpolation formulas with nested support nodes that are suitably chosen independently in each dimension. As already mentioned in the introduction, the sparse grid scheme can be used in conjunction with many different types of basis functions. In this paper, we use the Chebyshev-Gauss-Lobatto setting, i.e., the basis functions and sets of support nodes as defined in Eqs. (2), (3), (4). To keep the focus of the paper on the new aspects regarding the application of the fast transforms, we only treat regular sparse grids here, but the algorithm proposed in the following can be applied immediately to the dimension-adaptive approach as well, since it does not affect the sparse grid construction logic. With $U^0 = 0$, $\Delta^i = U^i - U^{i-1}$, $|\mathbf{i}| = i_1 + \cdots + i_d$ for $\mathbf{i} \in \mathbb{N}^d$, and $q \geq d, q \in \mathbb{N}$, the sparse grid algorithm is given by

$$A_{q,d}(f) = \sum_{|\mathbf{i}| \leq q} (\Delta^{i_1} \otimes \cdots \otimes \Delta^{i_d})(f), \tag{27}$$

or, alternatively, by [1, Eq. (3)]

$$A_{q,d}(f) = \sum_{q-d+1 \leq |\mathbf{i}| \leq q} (-1)^{q-|\mathbf{i}|} \cdot \binom{d-1}{q-|\mathbf{i}|} \cdot (U^{i_1} \otimes \cdots \otimes U^{i_d})(f). \tag{28}$$

To compute $A_{q,d}(f)$, only the function values at the sparse grid

$$H_{q,d} = \bigcup_{|\mathbf{i}| \leq q} (X_\Delta^{i_1} \times \cdots \times X_\Delta^{i_d}) \tag{29}$$

are needed. While Eq. (27) uses tensor products of difference formulas to construct the interpolant, Eq. (28) (often referred to as *combination technique*) uses a superposition of tensor products of nodal interpolation formulas. For the reasons explained in [8, p. 50] and the fact that Eq. (28) is not suitable for enhancement with dimensional adaptivity, we use Eq. (27). Analogously to the univariate case (Eq. 7), it can be rewritten explicitly in terms of hierarchical surpluses and basis functions as

$$A_{q,d}(f) = \sum_{|\mathbf{i}| \leq q} \sum_{\mathbf{j}} \underbrace{(a_{j_1}^{i_1} \otimes \cdots \otimes a_{j_d}^{i_d})}_{:= a_{\mathbf{j}}^{\mathbf{i}}} \cdot \underbrace{\left(f({}^\Delta x_{j_1}^{i_1}, \ldots, {}^\Delta x_{j_d}^{i_d}) - A_{q-1,d}(f)({}^\Delta x_{j_1}^{i_1}, \ldots, {}^\Delta x_{j_d}^{i_d}) \right)}_{:= w_{\mathbf{j}}^{\mathbf{i}}}, \tag{30}$$

with $\mathbf{j}$ denoting the multi-index $(j_1, \ldots, j_d)$, $j_l = 1, \ldots, m_{i_l}^\Delta$, and $l = 1, \ldots, d$.

Recalling the main result from [1], we have the following error estimate. For $f \in F_d^k$,

$$F_d^k = \{ f : [-1,1]^d \to \mathbb{R} \mid D^\beta f \text{ continuous if } \beta_i \leq k \ \forall \ i \}, \tag{31}$$

with $\beta \in \mathbb{N}_0^d$, the order of the interpolation error in the maximum norm is given by

$$\| f - A_{q,d}(f) \|_\infty = \mathcal{O}(N^{-k} \cdot |\log N|^{(k+2)(d+1)+1}), \tag{32}$$

where $N = \# H_{q,d}$.

The proposed algorithm To compute the hierarchical surpluses $w_{\mathbf{j}}^{\mathbf{i}}$, we have proposed the algorithms `spvals` and `spadaptvals` [8, Algorithms 4 and 9] that both use the subroutine `spalstep` [8, Algorithm 5]. These can remain unchanged apart from replacing the line containing the call to the interpolation routine `spinterpstep` in `spvalstep` by

$$(\Delta Y)_{l_{\text{new}}} += \texttt{spdctupstep}(d, (Z^{\text{old}})_{l_{\text{old}}}, (\Delta H^{\text{new}})_{l_{\text{new}}}, \mathbf{i}^{\text{old}}),$$

where $(Z^{\text{old}})_{l_{\text{old}}}$ holds the hierarchical surpluses $\mathbf{w}$ of multi-index $\mathbf{k} = \mathbf{i}^{\text{old}}$. The `spdctupstep` function must return the values of the difference function $(\Delta^{k_1} \otimes \cdots \otimes \Delta^{k_1})(f)$ at the grid points $X_\Delta^{i_1} \times \cdots \times X_\Delta^{i_1}$ of multi-index $\mathbf{i} = (\Delta H^{\text{new}})_{l_{\text{new}}}$. The new routine `spdctupstep` that achieves this task is outlined in Algorithm 3. To the most part, the notation chosen here to denote the sparse grid data adheres to [8, ch. 3]. However, we assume that the hierarchical surpluses and the grid points of each corresponding multi-index are stored as d-dimensional arrays which facilitates the array indexing required by Algorithm 3.

Algorithm 3 `spdctupstep`$(d, \mathbf{w}, \mathbf{i}, \mathbf{k})$

In: $d \in \mathbb{N}$. $\mathbf{w}$: d-dimensional array of hierarchical surpluses of subgrid index $\mathbf{k}$.
$\mathbf{k}$, $\mathbf{i}$: Base and target multi-index.

Out: $\Delta^{\mathbf{k}}({}^{\Delta}\mathbf{x}^{\mathbf{i}})$: Interpolated values of difference function $(\Delta^{k_1} \otimes \cdots \otimes \Delta^{k_d})$ at points ${}^{\Delta}\mathbf{x}^{\mathbf{i}}$.

1: Let $D = \{1, 2, \ldots, d\} \setminus \{l \mid k_l = i_l,\ l = 1, \ldots, d\}$.
2: **For** $d_{\text{act}} = (D)_1$ **To** $(D)_{\#D}$ **Do**
3: Let $\mathbf{r} = (i_1, \ldots, i_{d_{\text{act}}}, k_{d_{\text{act}}+1}, k_d)$.
4: Let $\hat{\mathbf{w}}$ be an array of size $(s_1, \ldots, s_d)$ with $s_1 \times \cdots \times s_d = m_{r_1}^{\Delta} \times \cdots \times m_{r_d}^{\Delta}$.
5: Let $\hat{\mathbf{y}}$ be a vector of size $m_{k_{d_{\text{act}}}}^{\Delta}$.
6: Let $\mathbf{y}$ be a vector of size $m_{i_{d_{\text{act}}}}^{\Delta}$.
7: **For All** $\mathbf{j} \in \{\mathbf{j} = (j_1, \ldots, j_d) \mid j_{d_{\text{act}}} = 1, j_l \in s_l\ \forall\ l \in \{1, \ldots, d\} \setminus d_{\text{act}}\}$ **Do**
8: **If** $k_{d_{\text{act}}} = 1$ **Then** {*Trivial case: difference function constant in dimension* d_{act}.}
9: Let $\hat{\mathbf{w}}_{(j_1,\ldots,j_d)} = \mathbf{w}_{(j_1,\ldots,j_{d_{\text{act}}-1},1,j_{d_{\text{act}}+1},\ldots,j_d)}$, for each $j_{d_{\text{act}}} = 1, \ldots, m_{i_{d_{\text{act}}}}^{\Delta}$.
10: **Else** {*Perform upsampling.*}
11: Let $\mathbf{y}_{j_{d_{\text{act}}}} = \mathbf{w}_{(j_1,\ldots,j_d)}$, for each $j_{d_{\text{act}}} = 1, \ldots, m_{k_{d_{\text{act}}}}^{\Delta}$.
12: $\hat{\mathbf{y}} = \texttt{dctupsample}(\mathbf{y}, i_{d_{\text{act}}}, k_{d_{\text{act}}})$.
13: Let $\hat{\mathbf{w}}_{(j_1,\ldots,j_d)} = \hat{\mathbf{y}}_{j_{d_{\text{act}}}}$, for each $j_{d_{\text{act}}} = 1, \ldots, m_{i_{d_{\text{act}}}}^{\Delta}$.
14: **End If**
15: **End For**
16: Let $\mathbf{w} = \hat{\mathbf{w}}$.
17: **End For**
18: Let $\Delta^{\mathbf{k}}({}^{\Delta}\mathbf{x}^{\mathbf{i}}) = \mathbf{w}$.

We proceed with a brief description of the key steps of the algorithm. Line 1 determines the dimensions that require upsampling: the dimensions where the target depth is equal to the base depth are omitted. Then, in the main loop (lines 2–17), the original surplus array $\mathbf{w}$ at multi-index $\mathbf{k}$ that corresponds to the values of the d-variate difference function $\Delta^{\mathbf{k}}(f) = (\Delta^{k_1} \otimes \cdots \otimes \Delta^{k_1})(f)$ at the points ${}^{\Delta}\mathbf{x}^{\mathbf{k}} = X_\Delta^{k_1} \times \cdots \times X_\Delta^{k_d}$ is then successively transformed with respect to each active dimension d_{act} to eventually hold the values $\Delta^{\mathbf{k}}(f)({}^{\Delta}\mathbf{x}^{\mathbf{i}})$. Line 4 allocates a re-sized array to hold the new upsampled data (the size increases only with respect to the single dimension d_{act} at a time). The inner loop (lines 7–15) applies the univariate upsampling routine `dctupsample` to the vectors $\mathbf{y}$ extracted from the current array $\mathbf{w}$, where all indices but the active dimension d_{act} remain fixed at a time.

Complexity The complexity of `spdctupstep` algorithm follows from the following observations. Considering the outer loop (line 2, conservatively neglecting the case $k_l = i_l$ from line 1), the multi-index $\mathbf{j}$ (line 7, the number of permutations is $\prod_{j=1}^{d} s_j$), and the cost of `dctupsample` (line 12), we get

$$\begin{aligned}\text{cost}(\texttt{spdctupstep}, \mathbf{i}, \mathbf{k}) &\leq \mathcal{O}\Big(\sum_{l=1}^{d}\Big[\prod_{j=1}^{l-1} m_{i_j}^{\Delta} \cdot \prod_{j=l+1}^{d} m_{k_j}^{\Delta} \cdot m_{i_l} \cdot \log_2 m_{i_l}\Big]\Big), \\ &\leq \mathcal{O}\Big(\sum_{l=1}^{d}\Big[\prod_{j=1}^{l} m_{i_j}^{\Delta} \cdot \log_2 m_{i_l}\Big]\Big) = \mathcal{O}\big(n(\mathbf{i}) \cdot |\mathbf{i}|\big), \end{aligned} \tag{33}$$

where $n(\mathbf{i})$ denotes the number of grid points of the sub-grid $X_{\Delta}^{i_1} \times \cdots \times X_{\Delta}^{i_d} = {}^{\Delta}x^{\mathbf{i}}$.

We can determine the new overall cost of computing the hierarchical surpluses of a sparse grid interpolant $A_{q,d}(f)$, $q = n + d$, with the `spvals` algorithm by using the results from [8, Eq. (3.65), Eq. (3.66)] and by replacing the cost of `spinterpstep` by the cost of our new algorithm `spdctupstep` to obtain

$$\text{cost}(\texttt{spvals}, n, d, \text{CGL}) = \mathcal{O}\Big(\sum_{k=0}^{n}\sum_{l=0}^{k-1}\sum_{\mathbf{i}\in\Delta S_{k,d}}\sum_{\substack{\mathbf{k}\in\Delta S_{l,d} \\ k_l \leq I_l}} \text{cost}(\texttt{spdctupstep}, \mathbf{i}, \mathbf{k})\Big),$$

where $\Delta S_{k,d} = \{\mathbf{i} \in \mathbb{N}^d \mid |\mathbf{i}| = k + d,\ |\mathbf{i}| = i_1, \ldots, i_d\}$. Using [8, Eq. (3.69)] and inserting Eq. (33), this can be further simplified conservatively to

$$\leq \mathcal{O}\Big(\sum_{k=0}^{n}(k+d)\sum_{\mathbf{i}\in\Delta S_{k,d}} n(\mathbf{i}) \cdot \big(\prod_{l=1}^{d} i_l - 1\big)\Big),$$

and with [8, Eq. (3.57), Eq. (3.73)], we arrive at

$$\text{cost}(\texttt{spvals}, n, d, \text{CGL}) \leq \mathcal{O}\Big(N \cdot (n+d) \cdot \Big(\frac{n+d}{d}\Big)^{d}\Big). \tag{34}$$

Remark 3. In the trivial case $d = 1$, the resulting complexity of computing the hierarchical surpluses $A_{n+1,n}(f) = U_{\text{H}}^{i}(f)$ is, as expected, $\mathcal{O}(m_{n+1}^{\Delta} \cdot (n+1)^2) = \mathcal{O}(m_i \log^2 m_i)$, where $i = n + 1$.

Remark 4. The resulting Eq. (34) is also interesting in so far that up to the factor $(n+d)/d$, we obtain the same upper bound for the complexity as in the piecewise linear sparse grid case, see [8, Eq. (3.72)]. Note that we have used either the same or additional upper bounds in deriving the complexity estimation here compared to the linear case.

Remark 5. If the sparse data structure and the algorithms discussed in [8] are used, this complexity is reduced significanty for large d, since the dimension d appearing in the above equations can be replaced by $\min\{d, \max\{n, 1\}\}$. This is due to the constant basis functions used at the lowest level, which are completely omitted in our sparse data approach, and therefore do not contribute to the computational cost. Thus, for $1 \leq n \leq d$, Eq. (34) becomes

$$\text{cost}(\texttt{spvals}, n, d, \text{CGL}) \leq \mathcal{O}\big(N \cdot n \cdot 2^n\big), \tag{35}$$

which explains the good performance of the algorithms in higher dimensions d: For fixed $n > 0$, we have $\#H_{n+d+1,d+1} > \#H_{n+d,d}$, or, reversing this statement, a decreasing value n for a fixed number of grid points N with increasing dimension d. Thus, Eq. (35) gives a decreasing complexity for fixed N and increasing d up to the trivial case where $N = \#H_{d+1,d}$.

Remark 6. For the evaluation of the interpolant $A_{q,d}(f)$, the algorithm `spinterp` developed in [8] based on the barycentric form can be used, achieving a complexity of $\mathcal{O}(N)$.

4 Numerical results

In this section, we do not present any results on the performance of the new algorithm in terms of stability and convergence (which is excellent due to the well-known, well behaved properties of the Chebyshev transform in the univariate case). The hierarchical surpluses computed with the new method are virtually unchanged from the ones computed with the original algorithm of [8, ch. 3] using the barycentric form (which also exhibits excellent stability properties, see [7]). Please refer to [1, 8] for numerical results on the approximation error for various test functions.

Instead, we present computation time results of our reference implementation in MATLAB, which has been integrated into the Sparse Grid Interpolation Toolbox available from our institute. To optimize the performance, we have vectorized the inner loop of `spdctupstep`. Furthermore, we use the sparse data structure as discussed in [8]. The DCT is implemented via MATLAB[2]'s FFT function. The performance tests were carried out using MATLAB V7.0.4 running on a Linux i686 1.8GHz PC. MATLAB V7.0.4 uses the FFTW [5] library to implement the FFT. Fig. 3 shows the performance of the `spvals` algorithm (computing time vs. number of grid points of $H_{k+d,d}$ using our new `spdctupstep` algorithm (marked "with DCT") compared to the previous version using `spinterpstep` (marked "no DCT"). To asses the performance also in terms of increasing dimension, we have indicated the linear-time complexity curve $\mathcal{O}(N)$ at the same position in each sub-plot.

The performance of our hierarchical surplus computation algorithm `spvals` using the new DCT-based subroutine `spdctupstep` outperforms the formerly used algorithm in lower dimensions $d \leq 4$ once the initial overhead of applying the more involved scheme of upsampling is compensated by the better asymptotic convergence rate. As d increases further, the polynomial degree in each variable decreases for a fixed number of points. Thus, the advantage of the fast transform is no longer present; only using significantly more points and thus increasing the polynomial degree would again change the picture in favor of the new DCT-based algorithm. However, the performance does not degenerate for larger dimensions d (see Remark 5).

5 Concluding remarks

In this article, we described how to apply the fast DCT in a sparse grid setting. Starting with an explanation of the main ideas in the univariate case, we generalized the approach to

[2] MATLAB is a registered trademark of The MathWorks, Inc.

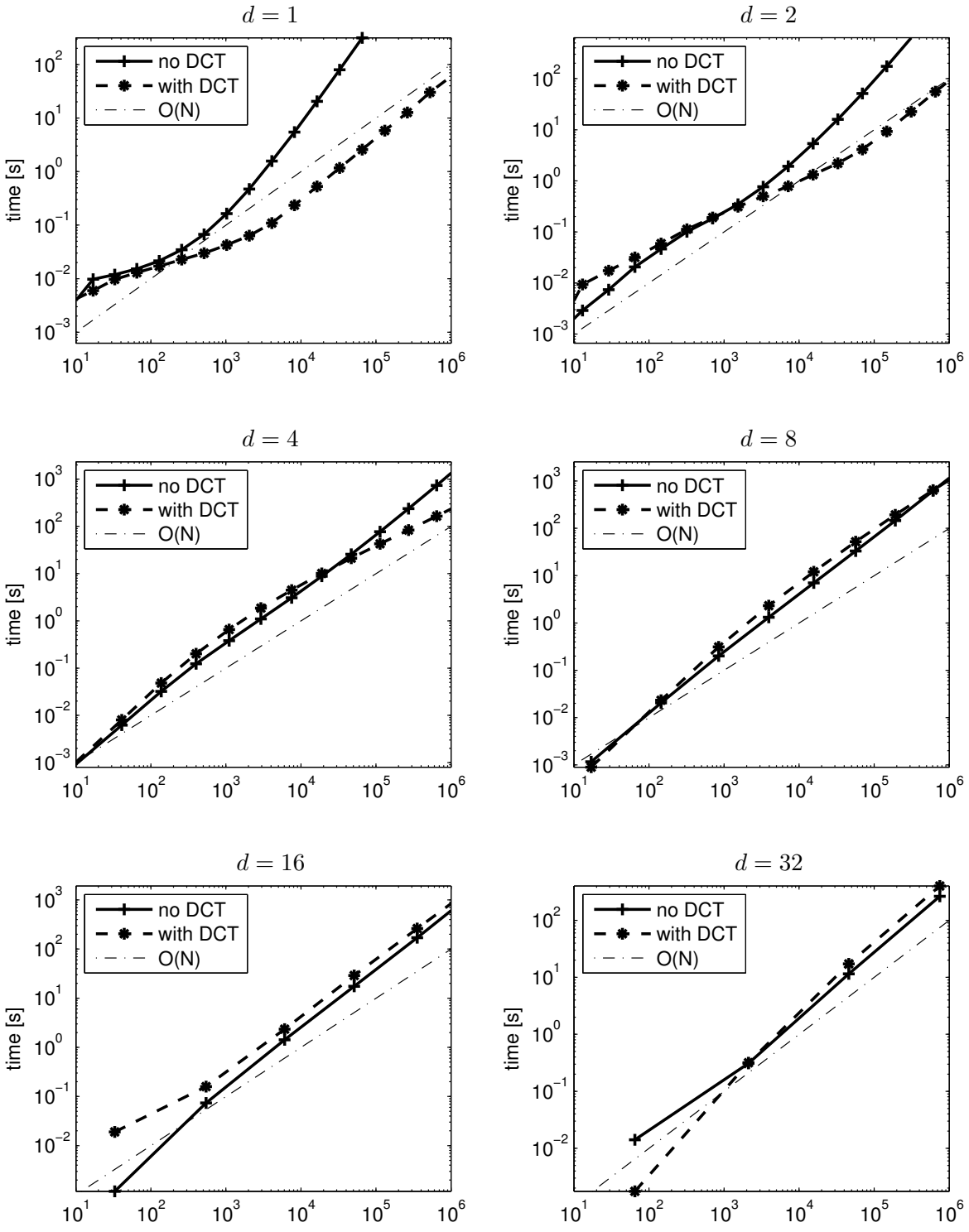


Figure 3: Computing time vs. number of grid points depending on d, (old ("no DCT") vs. new ("with DCT") algorithm.

multivariate sparse grid interpolants, proposed a suitable algorithm, and presented complexity estimations. We demonstrated the excellent performance of the new algorithm by providing numerical results of our reference implementation in MATLAB. Our modular approach makes is easy to integrate the new algorithms into a dimension-adaptive setting, which we have done within our Sparse Grid Interpolation Toolbox. As main application area, we see sparse grid interpolation as a powerful tool in obtaining highly accurate surrogate functions that are both cheap to obtain and evaluate.

Note that the surplus information computed with the spvals algorithm can also be easily used to perform numerical quadrature over d-dimensional boxes. Here, the surpluses must only be mutiplied by appropriate weights (the weights can be precomputed indepentently of the objective function) and then added up.

References

[1] V. Barthelmann, E. Novak, and K. Ritter. High dimensional polynomial interpolation on sparse grids. *Adv. Comput. Math.*, 12(4):273–288, 2000.

[2] Z. Battles and L. N. Trefethen. An extension of MATLAB to continuous functions and operators. *SIAM J. Sci. Comput.*, 25(5):1743–1770 (electronic), 2004.

[3] J.-P. Berrut and L. N. Trefethen. Barycentric Lagrange interpolation. *SIAM Review*, 46(3):501–517, 2004.

[4] H.-J. Bungartz and M. Griebel. Sparse grids. *Acta Numerica*, 13:147–269, 2004.

[5] M. Frigo and S. G. Johnson. The design and implementation of FFTW3. In *Proceedings of the IEEE*, number 2 in 93, pages 437–451, 2005.

[6] T. Gerstner and M. Griebel. Dimension-adaptive tensor-product quadrature. *Computing*, 71(1):65–87, 2003.

[7] N. J. Higham. The numerical stability of barycentric Lagrange interpolation. *IMA Journal of Numerical Analysis*, 24:547–556, 2004.

[8] A. Klimke. *Uncertainty modeling using fuzzy arithmetic and sparse grids.* PhD thesis, Universität Stuttgart, Shaker Verlag, Aachen, 2006.

[9] A. Klimke and B. Wohlmuth. Piecewise multilinear hierarchical sparse grid interpolation in MATLAB. *ACM Transactions on Mathematical Software*, 31(4), 2005.

[10] A. Klimke and B. Wohlmuth. Constructing dimension-adaptive sparse grid interpolants using parallel function evaluations, IANS preprint 2006/002. Technical report, University of Stuttgart, 2006.

[11] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical recipes in C.* Cambridge University Press, Cambridge, 1992.

[12] A. Schreiber. *Smolyak's method for multivariate interpolation.* PhD thesis, Georg-August-Universität Göttingen, Germany, 2000.

[13] F. Sprengel. Interpolation and wavelets on sparse Gauss-Chebyshev grids. In W. Haussmann et al., editor, *Multivariate Approximation*, volume 101 of *Mathematical Research*, pages 269–286. Akademie Verlag, Berlin, 1997.

[14] F. Sprengel. Periodic interpolation and wavelets on sparse grids. *Numer. Algorithms*, 17(1-2):147–169, 1998.

Erschienene Preprints ab Nummer 2006/001

Komplette Liste: `http://preprints.ians.uni-stuttgart.de`

2006/001 *Klimke, A.:* Sparse Grid Interpolation Toolbox - User's Guide

2006/002 *Klimke, A., Wohlmuth, B.:* Constructing Dimension-Adaptive Sparse Grid Interpolants using Parallel Function Evaluations

2006/003 *Hartmann, S., Brunssen, S., Ramm, E., Wohlmuth, B.:* Application of a primal-dual active set strategy for unilateral non-linear dynamic contact problems of thin-walled structures

2006/004 *Sändig, A.-M.:* Partielle Differentialgleichungen Vorlesung im Wintersemester 2005/2006

2006/005 *Nicaise, S., Witowski, K., Wohlmuth, B.:* An a posteriori error estimator for the Lame equation based on H(div)-conforming stress approximations

2006/006 *Geis, W. (ed.), Sändig, A.-M. (ed.):* Second International Workshop - Direct and Inverse Problems in Piezoelectricity,Hirschegg (Kleinwalsertal), Austria, July 16-19, 2006

2006/007 *Klimke, A.:* Efficient Construction of Hierarchical Polynomial Sparse Grid Interpolants using the Fast Discrete Cosine Transform