Adaptive sparse grid multilevel methods for elliptic PDEs based on finite differences

M. Griebel, Bonn

Abstract

We present a multilevel approach for the solution of partial differential equations. It is based on a multiscale basis which is constructed from a one-dimensional multiscale basis by the tensor product approach. Together with the use of hash tables as data structure, this allows in a simple way for adaptive refinement and is, due to the tensor product approach, well suited for higher dimensional problems. Also, the adaptive treatment of partial differential equations, the discretization (involving finite differences) and the solution (here by preconditioned BiCG) can be programmed easily. We describe the basic features of the method, discuss the discretization, the solution and the refinement procedures and report on the results of different numerical experiments.

AMS subject classifications: 65N06, 65N50, 68Y99, 68P05.
Key words: Sparse grids, finite difference, multiscale method, hash tables.

1 Introduction

In this paper, we present an adaptive multilevel approach for the solution of partial differential equations. It is based on a multiscale basis which is constructed from a one-dimensional hierarchical basis by the tensor product approach. Then, a continuous function u can be represented with respect to this basis as an infinite series. Also any approximation to the function with a prescribed error tolerance ε can be represented by a truncation of the infinite series to a finite one. Here, the size of the coefficients give a direct guideline and provide a reasonable error indicator.
Based on this finite dimensional representation of a discrete function, standard operations on functions can be implemented straightforwardly. Addition or subtraction of two functions can be implemented by just the addition or subtraction of their coefficient values. In the same way, scalar multiplication can be realized. The multiplication of two functions is achieved in a point-wise fashion. To this end, the coefficients of the two functions are transformed to their nodal values in the 'active' grid points, then, for all points, their nodal values are multiplied and finally, the result is transformed back to the hierarchical basis representation, e.g. after some further compression with respect to the threshold ε. Division of two functions can be obtained analogously. In this way we set up an algebra of operations with truncated functions.
Furthermore, employing the transformation to the nodal values in a special way, differential operators acting on ε-truncated functions can be implemented straightforwardly using finite difference stencils. Here, first order derivatives can be discretized either by local central differences leading to a second order consistent discrete derivative or, by using local upwind stencils, leading to first order but stable discrete operators. Analogously, second derivatives can be programmed easily resulting in a consistency order of two. Additionally, in the adaptive case, basically

the complexity, i.e. work count versus accuracy, of the regular sparse grid case can be obtained but this involves some further modifications of the stencils in the discretization.
For the treatment of partial differential equations, first a discretization must be set up (in general adaptively), and second, the discretized problem must be solved e.g. by an iterative method. In basically any iterative method, the action of the discrete differential operator onto a vector must be computed. This can be achieved in a simple way by putting this action of the discrete operator on a solution iterate together from the derivatives, the multiplications (with the (truncated) coefficient functions of the differential operator), the subtractions and the summations of our function algebra. Furthermore, multigrid methods and multilevel preconditioners based on prewavelets can be implemented straightforwardly.
Together with a simple refinement and coarsening of the truncated function representation on the basis of the coefficient values as error indicators, we have all ingredients for an adaptive multilevel method at hand: error indication, local refinement and coarsening, discretization and solution of the resulting linear system.
As an underlying data structure for the adaptively resolved data and solution approximations we decided to use a hash table approach. Hash tables are well known in computer science to store and retrieve data with minimal storage overhead and nearly direct access properties (in a statistical sense). However up to now, they were not yet used for the adaptive multilevel treatment of PDEs. There, tree-like data structures are the state of the art. But especially in the 3D and higher dimensional case such an approach is complicated, programming is very difficult and a large storage overhead is involved. These disadvantages are avoided by using hash tables. Furthermore, due to its inherent tensor product approach, the method is perfectly suited for higher-dimensional PDEs.
Our approach is closely related to the sparse grid method [9, 10, 16, 21, 42] and can be seen as an efficient implementation of it using finite difference stencils. Let $N = 2^n$, where n denotes the level of discretization. In case of piecewise d-linear hierarchical basis functions it can be shown [9, 20, 42] that the number of degrees of freedom and, using a multigrid method, thus the amount of operations to solve an elliptic PDE is proportional to $N \cdot (\log N)^{d-1}$ whereas the achieved accuracy is $O(N^{-2} \cdot (\log N)^{d-1})$ with respect to the L_2- and L_∞-norm and $O(N^{-1})$ with respect to the energy norm. This holds under the assumption that the solution fulfills a specific smoothness requirement, i.e. that its $2d$-th mixed derivative is bounded. In case that this prerequisite is not fulfilled, i.e. in case of singularities or strong variations in the solution, adaptive refinement helps and allows to maintain the complexity advantage of sparse grids also in these cases.
Note finally that the sparse grid approach (without adaptive refinement) is closely related to the technique of hyperbolic crosses [2], boolean methods [13] and discrete blending [6, 15, 28]. It can even be tracked back to Smolyak [38], see also [39].
The outline of this paper is as follows: In section 2 we give the subspace splitting representation of a function based on the tensor product approach and, beside some notation, we introduce finite dimensional approximations to the function by truncating the infinite series associated to the multiscale representation. This leads to a finite set of active level and index number pairs with associated hierarchical coefficients. This information can be stored in a hash table data structure on top of which multiscale algorithms can work easily.
Section 3 discusses how a whole algebra of operations and operators on such truncated function representations can be realized. This is in a similar spirit as round off error analysis for floating point numbers. We consider here the addition, subtraction and multiplication of two function representations and introduce also methods to implement discrete differential operators for first and second derivatives which are based on finite differences.

In section 4 we use these operations and operators to discretize (elliptic) partial differential equations. There, the right hand side and the coefficient functions of the differential operator must be resolved up to a prescribed accuracy. Then, for running an iterative method, the residual and thus the action of the differential operator on the actual iterate must be computed. This is done by means of the operations and operators of the previous section. As basic iterative method we use the BiCG approach. Besides, multilevel preconditioners and multilevel solvers can also be constructed. The existing algorithms can be combined easily to an overall adaptive refinement and solution procedure. Section 6 presents the results of numerical experiments with our sparse grid finite difference method. Finally we give some concluding remarks.

2 Multilevel representation of functions

2.1 Multilevel subspace splitting and tensor product basis

Let $\bar{\Omega} := [0,1]^d$ be the d-dimensional unit cube and let us consider the family of grids

$$\{\Omega_{\mathbf{l}}\}_{\mathbf{l}\in\mathbb{N}^d}$$

on Ω with mesh size $h_{\mathbf{l}} := (h_1,..,h_d) := (2^{-l_1},..,2^{-l_d})$, i.e. with in general different mesh sizes in the different coordinate directions, but equidistant mesh size with respect to one coordinate direction. The grid points contained in a grid $\Omega_{\mathbf{l}}$ are the points

$$x_{\mathbf{l},\mathbf{i}} := (x_{l_1,i_1},..,x_{l_d,i_d})$$

with $x_{l_j,i_j} := i_j \cdot h_{l_j} = i_j \cdot 2^{-l_j}, i_j = 0,..,2^{l_j}$. For reasons of simplicity, we restrict ourselves in the following to functions on Ω that vanish on the boundary. We consider on each of these grids the space of piecewise d-linear functions

$$V_{\mathbf{l}} := \mathrm{span}\{\phi_{\mathbf{l},\mathbf{i}}, i_j = 1,..,2^{l_j}-1, j = 1,..,d\}$$

which is spanned by the usual d-dimensional hat functions

$$\phi_{\mathbf{l},\mathbf{i}}(\mathbf{x}) := \prod_{j=1}^{d} \phi_{l_j,i_j}(x_j)$$

where $\mathbf{x} := (x_1,..,x_d)$. Here, the 1D-functions $\phi_{l_j,i_j}(x_j)$ can be created from a unique one-dimensional mother function

$$\phi(x_j) = \begin{cases} 1-|x_j| & \text{if } x \in (-1,1), \\ 0 & \text{otherwise,} \end{cases}$$

by dilation and translation, i.e.

$$\phi_{l_j,i_j}(x_j) = \phi\left(\frac{x_j - i_j \cdot h_j}{2^{-l_j}}\right).$$

Here and in the following, $\mathbf{l} = (l_1,..,l_d) \in \mathbb{N}^d, l_j > 0$, is a multi-index which indicates the number of a level of a grid or space, and $\mathbf{i} = (i_1,..,i_d) \in \mathbb{N}^d, i_j = 1,..,2^{l_j}-1$, is a multi-index which indicates the location of an interior grid point $x_{\mathbf{l},\mathbf{i}}$ and the corresponding center of the basis function $\phi_{\mathbf{l},\mathbf{i}}(\mathbf{x})$.
We now can define the difference spaces

$$W_{\mathbf{l}} := V_{\mathbf{l}} \ominus \sum_{j=1}^{d} V_{\mathbf{l}-\mathbf{e}_j}$$

where $\mathbf{e}_j$ denotes the j-th unit vector. To complete this definition we formally set

$$V_{\mathbf{l}-\mathbf{e}_j} = 0 \quad \text{if } l_j = 0.$$

We then have the following multilevel splitting of the Hilbert space

$$V = \sum_{l_1=1}^{\infty} \cdots \sum_{l_d=1}^{\infty} W_{(l_1,..l_d)} = \bigoplus_{\mathbf{l}\geq \mathit{1}} W_{\mathbf{l}} \tag{1}$$

which is up to completion the underlying Sobolev space, i.e $\bar{V} = H_0^1$. Here and in the following, let $\mathit{1} := (1,..,1)$, let $\geq$ denote elementwise comparison and let $|\mathbf{l}|_\infty := \max_{j=1}^d l_j$ denote the discrete L_∞-norm. Note that the splitting is into a direct sum by definition.
Note also that with the discrete spaces

$$V_n^{(\infty)} := \bigoplus_{\mathbf{l}\geq \mathit{1}, |\mathbf{l}|_\infty \leq n} W_{\mathbf{l}}$$

the limit

$$\lim_{n\to\infty} V_n^{(\infty)} = \lim_{n\to\infty} \bigoplus_{\mathbf{l}\geq \mathit{1}, |\mathbf{l}|_1 \leq n} W_{\mathbf{l}}$$

exists because $V_n^{(\infty)} \subset V_{n+1}^{(\infty)}$ and $\bigcup_{n=1}^\infty V_n^{(\infty)}$ is dense in $H_0^1(\bar{\Omega})$.
Since

$$W_{\mathbf{l}} = \text{span}\{\phi_{\mathbf{l},\mathbf{i}}(\mathbf{x}), i_j = 1,..,2^{l_j}-1, i_j \text{ odd}, j = 1,..,d\}, \tag{2}$$

the family of functions

$$\{\phi_{\mathbf{l},\mathbf{i}}(\mathbf{x}), i_j = 1,..,2^{l_j}-1, i_j \text{ odd }, j = 1,..,d\}_{\mathbf{l}\geq \mathit{1}} \tag{3}$$

is just a hierarchical basis [14, 40, 41] of $H_0^1([0,1]^d)$ which generalizes the one-dimensional hierarchical basis of [14] to the d-dimensional case by means of a tensor product approach. Note here that the supports of all basis functions $\phi_{\mathbf{l},\mathbf{i}}(\mathbf{x})$ are mutually disjoint which span $W_{\mathbf{l}}$. Furthermore, note similarities with the construction in [26].
Now, any function of $u \in V$ can be split accordingly by

$$u = \sum_{\mathbf{l}\geq \mathit{1}} u_{\mathbf{l}}(\mathbf{x}) = \sum_{\mathbf{l}\geq \mathit{1}} \sum_{\mathbf{i}\in \mathbf{I}_{\mathbf{l}}} u_{\mathbf{l},\mathbf{i}} \cdot \phi_{\mathbf{l},\mathbf{i}}(\mathbf{x}) \quad \text{where} \quad u_{\mathbf{l}}(\mathbf{x}) \in W_{\mathbf{l}} \tag{4}$$

and

$$u_{\mathbf{l}}(\mathbf{x}) = \sum_{\mathbf{i}\in \mathbf{I}_{\mathbf{l}}} u_{\mathbf{l},\mathbf{i}} \cdot \phi_{\mathbf{l},\mathbf{i}}(\mathbf{x}) \tag{5}$$

where $u_{\mathbf{l},\mathbf{i}} \in \mathbb{R}$ are the coefficient values of the hierarchical basis representation and $\mathbf{I}_{\mathbf{l}}$ denotes the set of indices

$$\mathbf{I}_{\mathbf{l}} := \{(i_1,..,i_d) \in \mathbb{N}^d, i_j = 1,..,2^{l_j}-1, i_j \text{ odd}, j = 1,..,d\}.$$

Note that since (3) forms a basis of V, the coefficient values are determined uniquely.
Now we consider the coefficient values $u_{\mathbf{l},\mathbf{i}}$ in more detail. They can be computed from the function values $u(\mathbf{x}_{\mathbf{l},\mathbf{i}})$ in the following way:

$$u_{\mathbf{l},\mathbf{i}} = \left(\prod_{j=1}^d I_{x_{l_j,i_j},l_j} \right) u =: I_{\mathbf{x}_{\mathbf{l},\mathbf{i}},\mathbf{l}} u. \tag{6}$$

where

$$I_{x_{l_j,i_j,l_j}} := \left[\ -\tfrac{1}{2} \quad 1 \quad -\tfrac{1}{2}\ \right]_{x_{l_j,i_j,l_j}} \quad 0 < i_j < 2^{l_j}. \tag{7}$$

This is due to the definition of the spaces $W_{\mathbf{l}}$ and their basis functions (2). Here, as usual in multigrid terminology, $I_{\mathbf{x}_{\mathbf{l},\mathbf{i}},\mathbf{l}}$ denotes a d-dimensional stencil which gives the coefficients for a linear combination of nodal values of u, see also [24], p. 48 (4.2.12).

As described in more detail in [11, 42], two partial integration steps for each coordinate direction lead us from (6) to the following representation in terms of an integral transformation.

Lemma 1: *Let $\psi_{l_j,i_j} = -2^{-(l_j+1)} \cdot \phi_{l_j,i_j}(x_j)$, and let $\psi_{\mathbf{l},\mathbf{i}}(\mathbf{x}) := \prod_{j=1}^{d} \psi_{l_j,i_j}(x_j)$. Furthermore, let u be such that its derivative $\partial^{2d}u / \prod_{j=1}^{d} \partial x_j^2$ exists and belongs to $\mathcal{C}^0(\bar{\Omega})$. For any coefficient value $u_{\mathbf{l},\mathbf{i}}$ in the representation (4) there holds:*

$$u_{\mathbf{l},\mathbf{i}} = \int_{\Omega} \psi_{\mathbf{l},\mathbf{i}}(\mathbf{x}) \cdot \frac{\partial^{2d} u(\mathbf{x})}{\prod_{j=1}^{d} \partial x_j^2}\ \mathrm{d}\Omega. \tag{8}$$

For functions not vanishing on the boundary, similar formulas exist for the coefficients which belong to points situated on the boundary of the domain. Depending on the dimension of the boundary manifold these formulas involve less derivatives and some Dirac functions in the product definition of $\psi_{\mathbf{l},\mathbf{i}}$. Note that if the considered function u is not smooth enough, i.e. not sufficiently differentiable, then a more general formula exists. It involves the $2d$-th variation of a function instead of the $2d$-th derivative and the whole definition boils down to that of the variation in the sense of Hardy and Krause, see [33], pp. 19-20. Note furthermore that the computation of the hierarchical coefficients can be performed by d successive applications of the one-dimensional transformation due to the tensor product construction of the hierarchical basis.

The size of the coefficients $u_{\mathbf{l},\mathbf{i}}$ reflects the smoothness of the function u. For sufficiently smooth functions, $u_{\mathbf{l},\mathbf{i}}$ is proportional to $2^{-<\mathbf{r},\mathbf{l}>}$, with $<\mathbf{r},\mathbf{l}> = \sum_{j=1}^{d} r_j \cdot l_j$, $\mathbf{r} = (r_1, ..., r_d)$, and $r_j > 0$ depending on the degree of smoothness with respect to the $j-th$ coordinate direction. But also for non-smooth functions, singularities are indicated by the size and behavior of the coefficients similar to wavelets [29, 30]. This is not a surprise since our hierarchical basis can be seen as some sort of wavelet in a weak distributional sense. Successive partial integration of (8) results in

$$u_{\mathbf{l},\mathbf{i}} = \int_{\Omega} \frac{\partial^{2d} \psi_{\mathbf{l},\mathbf{i}}(\mathbf{x})}{\prod_{j=1}^{d} \partial x_j^2} \cdot u\ \mathrm{d}\Omega =: \int_{\Omega} \omega_{\mathbf{l},\mathbf{i}}(\mathbf{x}) \cdot u\ \mathrm{d}\Omega$$

with $\omega_{\mathbf{l},\mathbf{i}}(\mathbf{x})$ equal to a linear combination of Dirac pulses with similar oscillating structure as certain wavelets.

2.2 Finite dimensional subspaces, truncation

We now turn to finite dimensional subspaces and the corresponding interpolants of a function. The usual case is that of an uniform grid, i.e.

$$u^n = \sum_{|\mathbf{l}|_\infty \le n} \sum_{\mathbf{i}} u_{\mathbf{l},\mathbf{i}} \cdot \phi_{\mathbf{l},\mathbf{i}}$$

where $n \in \mathbb{N}$ is a given number which denotes the level of discretization and $|\mathbf{l}|_\infty = \max_j l_j$. The associated interpolation error estimates are well known and thus not repeated here.

Besides, our tensor product approach also allows for the following approach, which is known under the name sparse grid, see also [9, 10, 11, 16, 17, 20, 21]. Let $|\mathbf{l}|_1 := \sum_{j=1}^d l_j$ and consider

$$u^{n,S} = \sum_{|\mathbf{l}|_1 \le n+d-1} \sum_{\mathbf{i}} u_{\mathbf{l},\mathbf{i}} \cdot \phi_{\mathbf{l},\mathbf{i}}. \tag{9}$$

The dimension of the underlying sparse grid space is only $O(n^{d-1} \cdot 2^n)$ in comparison to $O(2^{d \cdot n})$ of that of the regular full grid space. However the accuracy of the sparse grid interpolant $u^{n,S}$ is nearly that of the full grid interpolant u^n, i.e. it is of the order $O(n^{d-1} \cdot 2^{-2n})$ with respect to the L_2- and L_∞-norm and it is even $O(2^{-n})$ with respect to the energy norm provided that the function u is sufficiently smooth. For the above estimates, basically the $\partial^{2d} / \prod_{j=1}^d$-th derivative of u must be bounded. If this is not fulfilled, i.e. especially in case of singularities or steep boundary layers etc., we have to use sufficiently refined sparse grids instead. To this end, let $\varepsilon \in \mathbb{R}$ be a given threshold. Now we switch from the infinite representation (4) to the (hopefully) finite dimensional approximation

$$u^{\varepsilon, ||.||}(\mathbf{x}) = \sum_{\substack{\mathbf{l},\mathbf{i} \\ ||u_{\mathbf{l},\mathbf{i}} \cdot \phi_{\mathbf{l},\mathbf{i}}(\mathbf{x})|| \ge \varepsilon}} u_{\mathbf{l},\mathbf{i}} \cdot \phi_{\mathbf{l},\mathbf{i}}(\mathbf{x}) \tag{10}$$

i.e. we simply omit all basis functions and coefficients whose values with respect to a given $||.||$ are smaller than the threshold ε. Here, depending on the chosen norm or semi-norm, different approaches can be obtained. We have the error indicators

$$||u_{\mathbf{l},\mathbf{i}} \cdot \phi_{\mathbf{l},\mathbf{i}}(\mathbf{x})|| = |u_{\mathbf{l},\mathbf{i}}| \cdot ||\phi_{\mathbf{l},\mathbf{i}}(\mathbf{x})|| =: |u_{\mathbf{l},\mathbf{i}}| \cdot \gamma_{\mathbf{l}} \tag{11}$$

where

$$\gamma_{\mathbf{l}} = \begin{cases} 1 & \text{for } ||.|| = ||.||_{L_\infty}, \\ 2^{-d} \cdot 2^{-|\mathbf{l}|_1} & \text{for } ||.|| = ||.||_{L_1}, \\ (2/3)^{d/2} \cdot 2^{-|\mathbf{l}|_1/2} & \text{for } ||.|| = ||.||_{L_2}, \\ (2 \cdot (2/3)^{d-1} \cdot 2^{-|\mathbf{l}|_1} \cdot \sum_{j=1}^d 2^{2l_j})^{1/2} & \text{for } ||.|| = ||.||_{H_1}. \end{cases}$$

At the boundary of $\bar{\Omega}$, the necessary modifications are obvious. Note that (in the interior of $\bar{\Omega}$) the value $\gamma_{\mathbf{l}}$ is independent of the index $\mathbf{i}$ but depends only on the level number $\mathbf{l}$. This is due to the construction of the basis functions by dilatation and translation.

The most local choice is surely the maximum-norm. Then the thresholding boils down to simply taking the absolute value of the respective hierarchical coefficient. However for practical purposes, this norm can be too sharp and may result in non-terminating algorithms. This can be seen easily form the following simple one-dimensional example: Let

$$u(x) = \begin{cases} 0 & 0 \le x < 1/2, \\ 1 & 1/2 \le x \le 1. \end{cases}$$

A short calculation shows that the hierarchical coefficients $u_{l,i}$ with $i = 2^{l-1} - 1, l = 2, 3, ..., \infty$ possess the value $1/2$ whereas all other interior coefficients are zero (except $u_{1,1} = 1/2$). For the boundary coefficients we have $u_{0,0} = 0$ and $u_{0,1} = 1$. Thus, a local but infinite tail of coefficients with value $1/2$ appears next to the jump. However, for the other norms, additional damping values come in. Consequently also for the values with index $i = 2^{l-1} - 1, l = 2, 3, ..., \infty$ the threshold criterion $|u_{l,i}| \cdot \gamma_l < \varepsilon$ gets fulfilled for a sufficient large level number l. Then, we obtain a finite set of active indices also for non-differentiable functions. In practical applications often a combination of the maximum-norm and an other norm gives good results. Furthermore, from a practical point of view, there is an other difficulty. We can not first compute the infinite table of coefficients $u_{\mathbf{l},\mathbf{i}}$ and then omit the respective entries by means of the truncation criterion in a bottom up approach. Instead, we should proceed in a top down approach recursively level by level starting from the coarsest one. This is demonstrated by the following simple recursive bisection procedure for the one-dimensional case:

```
adapt(l, i, ε)
  hv := u(x_{l,i}) − (u(x_{l,i−1}) + u(x_{l,i+1}))/2;
  if |hv| · ||φ_{l,i}(x)|| < ε then skip;
  else u_{l,i} := hv;
       adapt(l + 1, 2i − 1, ε);
       adapt(l + 1, 2i + 1, ε);
  endif
```

For a given one-dimensional continuous function u, this procedure builds up the interior coefficient values $u_{l,i}$ of the truncated hierarchical representation. The representation is complete with $u_{0,0} = u(x_{0,0})$, $u_{0,1} = u(x_{0,1})$. The modification for the d-dimensional case is obvious, the parts of the function u living on the boundaries must also be resolved adaptively.

Then, it may happen that such a procedure terminates too early and does not resolve and compute large coefficients on very fine levels. An extreme example is the simple one-dimensional function $u(x) = \sin(x)$ on $[0, 2\pi]$. Here we have $u_{1,1} = (\sin(0) + \sin(2\pi))/2 - \sin(\pi) = 0$ and the above procedure terminates immediately. Of course, this problem can be circumvented by a more clever error indicator which, for example, also takes the values of the hierarchical neighbors into account or involves more sophisticated area weighted norms. But in principle such modifications of the truncation criterion are useless: Give me your error indicator and I give you a function for which the recursive resolution procedure will fail. The same problem appears if the function u is just a small spike on a very fine level, for example $u(x) = \phi_{20,35}$. There is no hope to detect it by a recursive bisection approach and to resolve such a function properly.

Furthermore, for practical purposes, the descriptive notation (10) should not allow for 'holes' in the table of the $u_{\mathbf{l},\mathbf{i}}$-coefficient values. We therefore restrict our notation (10) to

$$u^{\varepsilon,||.||}(\mathbf{x}) = \sum_{(\mathbf{l},\mathbf{i})\in\mathcal{A}(u,\varepsilon,||.||)} u_{\mathbf{l},\mathbf{i}} \cdot \phi_{\mathbf{l},\mathbf{i}}(\mathbf{x}) \tag{12}$$

where $\mathcal{A}(u, \varepsilon, ||.||)$ denotes the set of 'active' indices, i.e.

$$\mathcal{A}(u,\varepsilon,||.||) := \left\{ \begin{array}{ll} (\mathbf{l},\mathbf{i}) : \mathbf{i} \in I_{\mathbf{l}}, & ||u_{\mathbf{l},\mathbf{i}} \cdot \phi_{\mathbf{l},\mathbf{i}}(\mathbf{x})|| \geq \varepsilon \ \vee \\ & \exists\ (\mathbf{k},\mathbf{j}) : \mathbf{k} \geq \mathbf{l}, ||u_{\mathbf{k},\mathbf{j}} \cdot \phi_{\mathbf{k},\mathbf{j}}(\mathbf{x})|| \geq \varepsilon, \\ & \qquad \mathrm{supp}(\phi_{\mathbf{k},\mathbf{j}}) \cap \mathrm{supp}(\phi_{\mathbf{l},\mathbf{i}}) \neq \emptyset \end{array} \right\}. \tag{13}$$

Here, $\mathrm{supp}(\phi) = \{\mathbf{x} : \ \phi(\mathbf{x}) > 0\}$ is the open support of ϕ.

For the following assume that we have constructed a finite table of active hierarchical coefficients $u_{\mathbf{l},\mathbf{i}}$ which are associated to the active index set $\mathcal{A}(u, \varepsilon, ||.||)$. Of course, by definition, these coefficients are not equivalent (except of the corner points of our d-dimensional cube) to the values the function u possesses in the associated points $\mathbf{x}_{\mathbf{l},\mathbf{i}}$. Now the question arises how these nodal values $u(\mathbf{x}_{\mathbf{l},\mathbf{i}})$ can be reconstructed from the hierarchical coefficients $u_{\mathbf{l},\mathbf{i}}$ of our finite table. To this end, consider again the simple one-dimensional case:

$E_1(l, i, ua, ub)$
if $\exists u_{l,i}$ then
$n_{l,i} := (ua + ub)/2 + u_{l,i}$
$E_1(l + 1, 2i - 1, ua, n_{l,i})$
$E_1(l + 1, 2i + 1, n_{l,i}, ub)$
else *skip*;
endif
	$H_1(l, i, ua, ub)$
	if $\exists n_{l,i}$ then
	$u_{l,i} := n_{l,i} - (ua + ub)/2$
	$H_1(l + 1, 2i - 1, ua, n_{l,i})$
	$H_1(l + 1, 2i + 1, n_{l,i}, ub)$
	else *skip*;
	endif

With the procedure E_1, the nodal values are reconstructed and stored in $n_{l,i}$. We start with $E_1(1, 1, u(0), u(1))$. Analogously, the computation of the hierarchical coefficients $u_{l,i}$ from a finite table of given nodal values $n_{l,i}$ is performed by the procedure H_1. We start with $H_1(1, 1, u(0), u(1))$. Note that these transformations are closely related to the prolongation procedure in classical multigrid and the pyramid scheme and the refinement equation for the wavelet transformation. We see that, in both cases, we proceed level by level due to the recursion. From this it gets clear why we don't want to allow for 'holes' in the table of active coefficients. The procedures E_1 and H_1 implement the matrix vector multiplications

$$\vec{n} = E_1 \vec{u} \qquad \vec{u} = H_1 \vec{n}$$

where $\vec{u}$ and $\vec{n}$ contains the active hierarchical and nodal values respectively. For reasons of simplicity we denote the associated matrices with the same letters as the algorithms.

The generalization of the transformations H_1 and E_1 to the d-dimensional case is straightforward: All we have to do is to call the procedures of the one-dimensional case successively for all dimensions d under consideration on all $d-1$-dimensional manifolds, see also [10] for more details on that dimension-recursive process. To this end we define the d-dimensional operators $\mathcal{H}_j$ and $\mathcal{E}_j$ where the transformations take place in the j-th coordinate direction by

$$\mathcal{H}_j := I_1 \otimes ... \otimes I_{j-1} \otimes H_j \otimes I_{j+1} \otimes ... \otimes I_d = \left(\bigotimes_{i=1, i \neq j}^{d} I_i \right) \otimes H_j,$$

$$\mathcal{E}_j := I_1 \otimes ... \otimes I_{j-1} \otimes E_j \otimes I_{j+1} \otimes ... \otimes I_d = \left(\bigotimes_{i=1, i \neq j}^{d} I_i \right) \otimes E_j.$$

Here, H_j and E_j correspond to the one-dimensional transformations H_1 and E_1 (now used for the j-th coordinate) and I_j denotes the one-dimensional identity for the j-th coordinate. Then, we obtain with

$$\mathcal{H} = \mathcal{H}_1 \circ ... \circ \mathcal{H}_d =: \bigcirc_{j=1}^{d} \mathcal{H}_j$$

$$\mathcal{E} = \mathcal{E}_1 \circ ... \circ \mathcal{E}_d =: \bigcirc_{j=1}^{d} \mathcal{E}_j$$

the transformations involving all coordinate directions.

Note that the number of operations involved in $\mathcal{H}$ and $\mathcal{E}$ is proportional to the number of active indices.

Analogously to *adapt*, a procedure *compress* can be written easily which deletes in a bottom up way the active coefficients $u_{\mathbf{l},\mathbf{i}}$ with criterion $||u_{\mathbf{l},\mathbf{i}} \cdot \phi_{\mathbf{l},\mathbf{i}}|| < \varepsilon$. Here again, 'holes' in the resulting table of active indices must be avoided. The generalization to the d-dimensional case is straightforward.

2.3 Hash table storage as data structure for multiscale methods

Now we are in the following situation: We (hopefully) resolved a continuous function by a top down approach up to a prescribed accuracy using the threshold ε. In other words we computed a finite set of active indices and the corresponding coefficient values. We now have to find a data structure which allows to store, to retrieve and to access these data efficiently.

A first approach might be a binary tree structure. Tree data structures are quite common in many adaptive codes for the multilevel solution of PDEs [4, 5, 32, 36]. There, different trees represent the hierarchies of nodes, edges and elements, while entities on one level of a tree represent one grid. Refining the finest grid means adding new leaves to the tree. However, in order to administrate the nodes (unknowns), edges (stiffness matrix) and elements (grid), the leaves of the trees have to be linked. This results in a number of pointers, both for the tree and for the links between the trees. Many software packages need a value of 400 and more bytes of memory per unknown for a scalar problem. In three dimensions, numbers can be even higher. Thus, there is more memory required for the administration of the data than for the numerical data itself. But note the very economical data structure BASIS3 [27] which only uses about 80 bytes additional memory per unknown in two and three-dimensions.

The tree approach for our sparse grid method is inspired by the recursive bisection algorithm to build the table of active coefficients. For the higher dimensional case, we could use the tensor product structure, i.e. we could work recursively in the number of dimensions and would obtain a binary tree (d-th dimension) with nodes that have pointers to binary trees ($d-1$-th dimension), and so on. To understand this better, consider the two-dimensional case. Every row in the adaptive sparse grid is a one-dimensional adaptive grid which can be represented by a binary tree containing the corresponding grid points. The set of all existing rows in the grid can be represented as a binary tree with pointers in each node to the row grid lines. The modification for the boundaries is obvious. Besides, also graph like data structures can be used. For more details, see [3].

Now, performing numerical operations on one grid often requires a complete tree traversal. In addition to the computational operations, a number of indexing and administration operations have to be performed which degrades overall performance. Of course it is possible to eliminate some of the tree traversals by establishing additional data structures like linked lists or sparse matrices at the expense of additional memory, but this results in more storage requirements and complicates programming.

Therefore, we decided to use a hash table concept [31] instead. It is quite well known in computer science for many years but, as far as we know, it was never applied for adaptive multiscale methods. Thus, in the following, we discard the tree approach and consider directly our plain data. There, the structure information is simply the finite set of active indices $(\mathbf{l}, \mathbf{i}) \in \mathcal{A}(u, \varepsilon, ||.||)$.

The idea of hash table storage is to map each entity (in our case the index pair $(\mathbf{l}, \mathbf{i}) \in \mathbb{N}^{2d}$) to a *hash-key* which is used as an address in the hash table. The entity and its associated data are stored and can be retrieved at that address in the hash table which is implemented as a linear array of cells (buckets). The mapping is done by a (deterministic) hash function. Since there are many more possible different entities than different hash keys, the hash function is not injective. Algorithms to resolve collisions are needed. It may also happen that some entries in the hash table are left empty, because no present entity is mapped to that key.

To deal with the collision case, basically two approaches are commonly used. The first is the double hashing technique. Here, in case of a collision we compute a new adress by adding (modulo size of the hash table) to the present adress the result of the evaluation of a second hash function. This step is iterated until, for example in the insertion case, a free cell in the hash table is found. (Of course if the hash table is full, it must be sufficiently enlarged.) The other technique, which we will use in the following, resolves the collision case by the so-called chaining approach. Here, instead of only one data entry per hash table adress, a whole list of entries is dynamically stored. In case of a collision, this list is searched and eventually enlarged. For an example of insertion of new data in case of a collision see Figure 1. The analogous approach is used in retrieve and delete operations. For further details on hash tables, collision treatment and optimal strategies see [31].

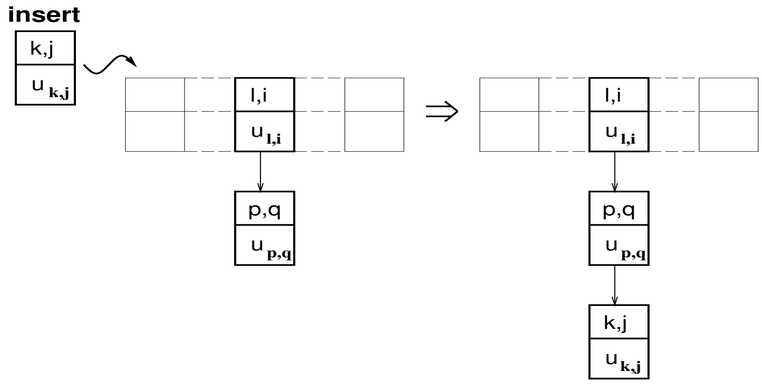

Figure 1: Collision treatment by chaining for the case of insertion of data.

Hash tables allow to deal with locally adapted or compressed data in a simple way. They give more or less direct access to the stored data (if the hash function scatters the entries broad enough and there are enough different cells in the hash table), i.e. they are proven to have a $O(1)$ complexity with a low constant if a statistical setting is assumed. Furthermore, they need no additional storage overhead for logical connectivities, like tree-type data structures which are usually used in adaptive finite element codes, see [5] and the references cited therein. Finally, they are easy to program and to handle and they allow a straightforward implementation of multilevel algorithms on top of them. Meanwhile we applied the hash storage technique also successfully for a conventional adaptive multigrid method, see [22, 23].

Besides a hand-written code, we presently use the hash table implementation of an extended version of the C++standard template library (STL) [35], which uses chaining, i.e. linked lists, for the resolution of collisions and which provides automatic resizing. So the number of cells will be kept proportional to the number of entries and we will only have to bother with a well suited hash function.

We developed different hash functions and tested them for adaptively refined sparse grids in the case of smooth and singular functions (point- and line singularities) for two-, three- and higher-dimensional cases. We used the hash function

$$\mathrm{h}(\mathbf{l}, \mathbf{i}, d) = \left(\sum_{j=1}^{d} \left(2^{l_j} \cdot i_j\right) \cdot P(j) \cdot P(p-j) \right) \bmod m \tag{14}$$

with $p = 43 \cdot (d-2) \cdot 10$, where $P(k)$ is the k-th prime number and m denotes the size of the hash table. The form of this hash function is gained by a straightforward generalization of the well known principles for hash functions to the d-dimensional case. The specific choice of the above prime numbers was determined by exhaustive numerical experiments. Here, we considered various types of lower-dimensional

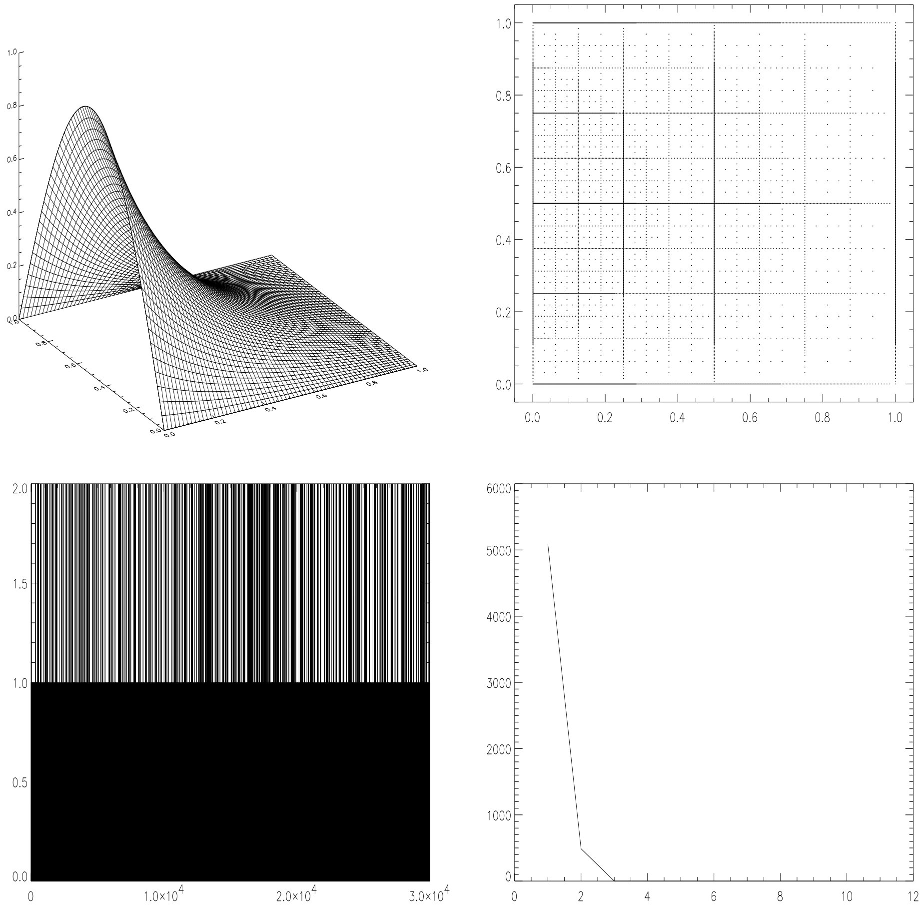

Figure 2: Smooth function, $u(x, y) = \sinh(\pi(1-x))\sin(\pi y)/\sinh(\pi), \varepsilon = 1.9 \cdot 10^{-6}, \Omega = [0,1]^2$, the size of the hash table is $3 \cdot 10^4$.

singular functions (point-, line-, surface-type singularities, etc.) as well as smooth functions with up to nine dimensions. It turned out that in all considered cases (smooth and singular, low and high-dimensional) the hash function scatters the data quite well and distributes them more or less equally over the hash-table. Examples are given in Figure 2, 3 and 4.

There, we show the function (upper left) and the grid obtained by adaptive refinement of the function with a given tolerance ε (upper right). Furthermore, we give the distribution of the points in the hash table (lower left). The x-axis denotes the address in the hash table, the y-axis gives the number of data to be stored under this location by chaining, i.e. it gives the length of the respective chain. Finally, we show a diagram illustrating the number of chains with the same length (lower right). The x-axis denotes the length of the chain, the y-axis counts the number of chains with length x. For both, the smooth function and the functions with point singularity and line singularity where strong adaptive refinement takes place, we clearly see that the hash function works well: The data get equally distributed over the hash table, the hash table is equally filled and the involved chains are not degenerated.

The costs of a hash table access consist of the evaluation of the hash function and the collision treatment. The evaluation of (14) involves $4d$ integer operations if we

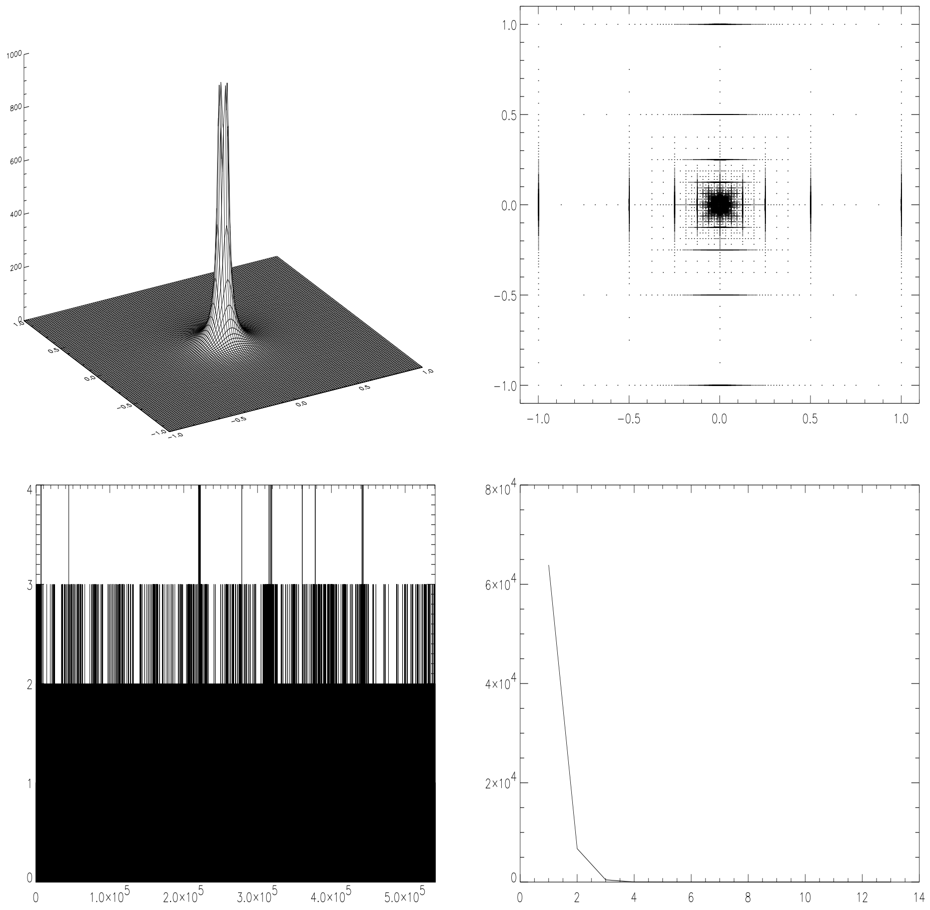

Figure 3: Regularized point singularity, $u(x,y) = 1/(|10^{-3} - x^2 - y^2| + \delta), \delta = 10^{-3}, \varepsilon = 10^{-2}, \Omega = [-1,1]^2$, the size of the hash table is 540.672.

precompute and store the values $P(j) \cdot P(p-j)$. The collision treatment involves q comparisons of level and index number pairs ($2d$ integer comparisons) where q denotes the respective chain length. Altogether, the costs of a hash table access are bounded by

$$(4 + \tilde{q} \cdot 2) \cdot d \tag{15}$$

integer operations with $\tilde{q}$ denoting the maximal chain length. If we use a hash table implementation with automatic resizing like that of the C++standard template library, we can control the maximal chain length explicitly.
Using such a hash table approach, programming of algorithms that work with finite dimensional approximations of functions is easy. We address the data and work with them simply by means of the multi-indices $\mathbf{l}$ and $\mathbf{i}$. All code can be written just using these indices as (abstract) data structure and the nasty details where and how the data are stored are completely hidden in the hash table module.

3 Operations and operators

Up to now we are able to represent and store finite dimensional approximations to functions properly. Now, we turn to standard operations like addition, subtraction,

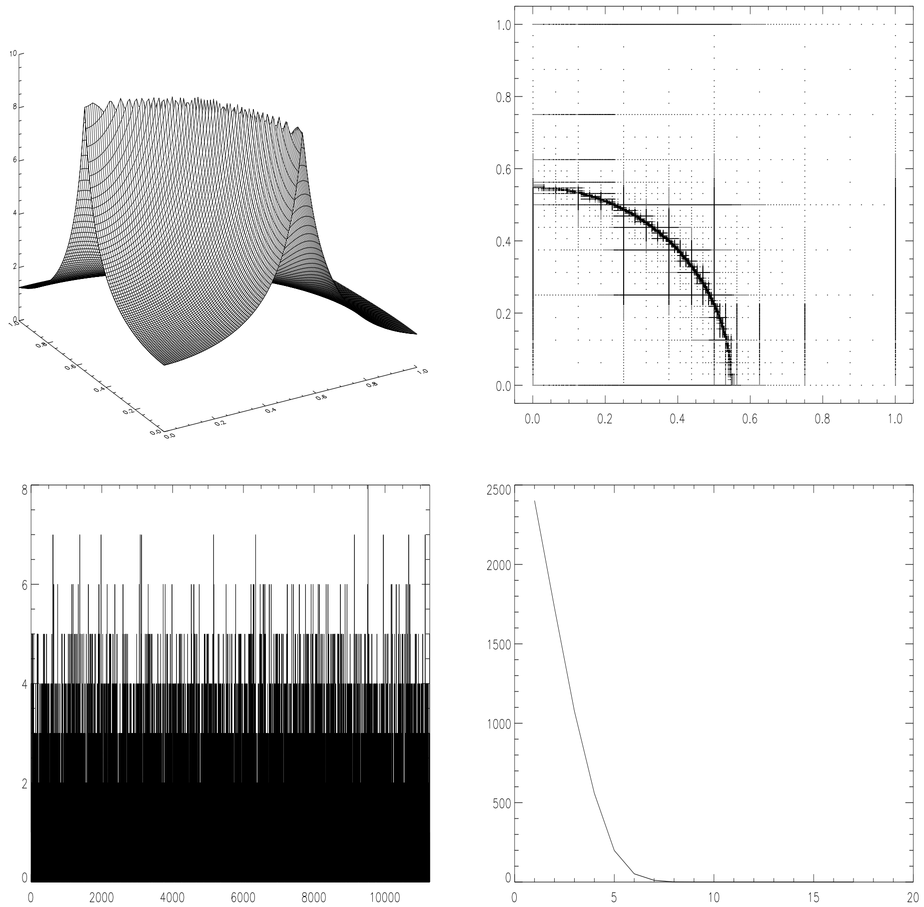

Figure 4: Regularized line singularity, $u(x,y) = 1/(|0.3 - x^2 - y^2| + \delta), \delta = 10^{-1}, \varepsilon = 5 \cdot 10^{-5}$, the size of the hash table is 11.264.

multiplication, etc., working with such ε-truncated functions. Furthermore, we consider differential operators on such functions.

3.1 An algebra of function operations

Using the representation (4) of a continuous function, it is directly clear how the addition or subtraction of two functions can be performed: All we have to do is to add or subtract their coefficients to obtain the result with respect to the representation (4), i.e.

$$u \pm v = \sum_{\mathbf{l},\mathbf{i}} u_{\mathbf{l},\mathbf{i}} \cdot \phi_{\mathbf{l},\mathbf{i}}(\mathbf{x}) \pm \sum_{\mathbf{l},\mathbf{i}} v_{\mathbf{l},\mathbf{i}} \cdot \phi_{\mathbf{l},\mathbf{i}}(\mathbf{x}) = \sum_{\mathbf{l},\mathbf{i}} (u_{\mathbf{l},\mathbf{i}} \pm v_{\mathbf{l},\mathbf{i}}) \cdot \phi_{\mathbf{l},\mathbf{i}}(\mathbf{x}).$$

In the same way, we can proceed in the finite dimensional case. Due to the finite number of active coefficients, the summation process of the coefficients is finite. Of course, the summation must be extended to the union of the two index sets induced by u and v and e.g. different truncation values ε_1 and ε_2, i.e.

$$u^{\varepsilon_1,||\cdot||} \pm v^{\varepsilon_2,||\cdot||} = \sum_{(\mathbf{l},\mathbf{i}) \in \mathcal{A}(u,\varepsilon_1,||.||) \cup \mathcal{A}(v,\varepsilon_2,||.||)} (u_{\mathbf{l},\mathbf{i}} \pm v_{\mathbf{l},\mathbf{i}}) \cdot \phi_{\mathbf{l},\mathbf{i}}(\mathbf{x}),$$

with missing coefficients taken to be zero.

The accuracy of the result can be controlled using the following theorem.

Theorem 1: *Let $u_1^{\varepsilon_1,||\cdot||}(\mathbf{x})$ and $u_2^{\varepsilon_2,||\cdot||}(\mathbf{x})$ be truncated functions of $u_1(\mathbf{x})$ and $u_2(\mathbf{x})$ with active index sets $\mathcal{A}_1$ and $\mathcal{A}_2$ in the sense of (13) with the thresholds ε_1 and ε_2 and error bounds*

$$|u_1(\mathbf{x}) - u_1^{\varepsilon_1,||\cdot||}(\mathbf{x})| < \tilde{\varepsilon}_1, \qquad |u_2(\mathbf{x}) - u_2^{\varepsilon_2,||\cdot||}(\mathbf{x})| < \tilde{\varepsilon}_2.$$

Then, the error bound for $u_1^{\varepsilon_1}(\mathbf{x}) + u_2^{\varepsilon_2}(\mathbf{x})$ is $\tilde{\varepsilon}_1 + \tilde{\varepsilon}_2$ on the index set $\mathcal{A}_3$ which is obtained by the union of the two index sets $\mathcal{A}_1$ and $\mathcal{A}_2$.

Proof:

$$\begin{aligned} & |\,(u_1(\mathbf{x}) + u_2(\mathbf{x})) - \left(u_1^{\varepsilon_1,||\cdot||}(\mathbf{x}) + u_2^{\varepsilon_2,||\cdot||}(\mathbf{x})\right)\,| = \\ = \;& |\left(u_1(\mathbf{x}) - u_1^{\varepsilon_1,||\cdot||}(\mathbf{x})\right) + \left(u_2(\mathbf{x}) - u_2^{\varepsilon_2,||\cdot||}(\mathbf{x})\right)| \\ \leq \;& |u_1(\mathbf{x}) - u_1^{\varepsilon_1,||\cdot||}(\mathbf{x})| + |u_2(\mathbf{x}) - u_2^{\varepsilon_2,||\cdot||}(\mathbf{x})| \\ \leq \;& \tilde{\varepsilon}_1 + \tilde{\varepsilon}_2 \end{aligned} \tag{16}$$

□

Note that the truncation parameters $\varepsilon_1, \varepsilon_2$ and the accuracies $\tilde{\varepsilon}_1, \tilde{\varepsilon}_2$ are in general different but closely related. Provided that the solution is sufficiently smooth, they are proportional to each other with a proportionality constant depending on the norm used in the adaptive refinement process. Otherwise things are more complicated, but here the hierarchical basis theory [14] and the wavelet theory [29, 30] give further insight, see also [12].
However, when we consider the multiplication of two functions u and v in hierarchical representation, things are not so simple any more. Now the pointwise multiplication of pairs of basis functions is involved, i.e.

$$u \cdot v = \sum_{\mathbf{l},\mathbf{i}} u_{l,i} \cdot \phi_{\mathbf{l},\mathbf{i}}(\mathbf{x}) \cdot \sum_{\mathbf{l},\mathbf{i}} v_{\mathbf{l},\mathbf{i}} \cdot \phi_{\mathbf{l},\mathbf{i}}(\mathbf{x}) = \sum_{\mathbf{l},\mathbf{i}} \sum_{\mathbf{k},\mathbf{j}} (u_{\mathbf{l},\mathbf{i}} \cdot v_{\mathbf{k},\mathbf{j}}) \cdot \phi_{\mathbf{l},\mathbf{i}}(\mathbf{x}) \cdot \phi_{\mathbf{k},\mathbf{j}}(\mathbf{x})$$

but this results in general not more directly in a representation of the type (4). We could try to express each product of two basis functions (which can be locally a quadratic function) by means of the hierarchical representation in an infinite series, i.e.

$$\phi_{\mathbf{l},\mathbf{i}}(\mathbf{x}) \cdot \phi_{\mathbf{k},\mathbf{j}}(\mathbf{x}) =: \sum_{\mathbf{m},\mathbf{r}} w_{\mathbf{m},\mathbf{r}}^{\mathbf{l},\mathbf{i},\mathbf{k},\mathbf{j}} \phi_{\mathbf{m},\mathbf{r}}(\mathbf{x})$$

but the computation of the coefficients $w_{\mathbf{m},\mathbf{r}}^{\mathbf{l},\mathbf{i},\mathbf{k},\mathbf{j}}$ and their reordering and summation is complicated and expensive. Furthermore, in the finite dimensional case, this still would require an infinite series representation for the result.
Therefore, we proceed as follows: We evaluate the multiplication pointwise in the set of points $\{\mathbf{x}_{\mathbf{l},\mathbf{i}} : (\mathbf{l},\mathbf{i}) \in \mathcal{A}(u, \varepsilon_1, ||.||) \cup \mathcal{A}(v, \varepsilon_2, ||.||)\}$. To this end, for $u^{\varepsilon_1,||\cdot||}$ and $v^{\varepsilon_2,||\cdot||}$, we compute recursively by means of $\mathcal{E}$ the nodal values in the points with indices that belong to the *union* of the two active index sets. Then, for each point, we multiply the associated values. This gives us (pointwise) the result of the multiplication of the two approximate functions. Now, the problem remains to span a continuous function again. To this end, we use the values in the points to compute the hierarchical representation of the result by means of $\mathcal{H}$. Finally, to get rid of resulting very small coefficients, we can make use of the *compress* procedure. Note that, in the intermediate step, we have no longer a basis representation. Furthermore, between grid points we introduced a slight error, which however can be

shown to be of the size of the approximation order only. Thus, our approach uses the multiplied nodal values to reconstruct by linear interpolation the result of the function multiplication. This is in the same spirit as [1, 25].

The *relative* accuracy of the result can be controlled using the following theorem.

Theorem 2: *Let $u_1^{\varepsilon_1,||\cdot||}(\mathbf{x})$ and $u_2^{\varepsilon_2,||\cdot||}(\mathbf{x})$ be truncated functions of $u_1(\mathbf{x})$ and $u_2(\mathbf{x})$ with active index sets $\mathcal{A}_1$ and $\mathcal{A}_2$ in the sense of (13) with the thresholds ε_1 and ε_2 and the error bounds*

$$|u_1(\mathbf{x}) - u_1^{\varepsilon_1,||\cdot||}(\mathbf{x})| < \varepsilon_1, \qquad |u_2(\mathbf{x}) - u_2^{\varepsilon_2,||\cdot||}(\mathbf{x})| < \varepsilon_2.$$

Then, error bounds for $u_1^{\varepsilon_1,||\cdot||}(\mathbf{x}) \cdot u_2^{\varepsilon_2,||\cdot||}(\mathbf{x})$ are

$$|u_1(\mathbf{x})| \cdot \tilde{\varepsilon}_2 + \tilde{\varepsilon}_1 \cdot |u_2^{\varepsilon_2,||\cdot||}(\mathbf{x})| \quad \textit{and} \qquad \tilde{\varepsilon}_1 \cdot |u_2(\mathbf{x})| + |u_1^{\varepsilon_1,||\cdot||}(\mathbf{x})| \cdot \tilde{\varepsilon}_2,$$

respectively, and a symmetric, but weaker error bound for $u_1^{\varepsilon_1,||\cdot||}(\mathbf{x}) \cdot u_2^{\varepsilon_2,||\cdot||}(\mathbf{x})$ is

$$\max\left(|u_2(\mathbf{x})|, |u_2^{\varepsilon_2,||\cdot||}(\mathbf{x})|\right) \cdot \tilde{\varepsilon}_1 + \max\left(|u_1(\mathbf{x})|, |u_1^{\varepsilon_1,||\cdot||}(\mathbf{x})|\right) \cdot \tilde{\varepsilon}_2$$

on the set $\mathcal{A}_3 = \mathcal{A}_1 \cup \mathcal{A}_2$.

Proof:

$$|u_1(\mathbf{x}) \cdot u_2(\mathbf{x}) - u_1^{\varepsilon_1,||\cdot||}(\mathbf{x}) \cdot u_2^{\varepsilon_2,||\cdot||}(\mathbf{x})| =$$

$$\begin{aligned}
&= |u_1(\mathbf{x}) \cdot u_2(\mathbf{x}) - u_1^{\varepsilon_1,||\cdot||}(\mathbf{x}) \cdot u_2^{\varepsilon_2,||\cdot||}(\mathbf{x}) + \overbrace{u_1(\mathbf{x}) \cdot u_2^{\varepsilon_2,||\cdot||}(\mathbf{x}) - u_1(\mathbf{x}) \cdot u_2^{\varepsilon_2,||\cdot||}(\mathbf{x})}^{=0}| \\
&= |u_1(\mathbf{x}) \cdot \left(u_2(\mathbf{x}) - u_2^{\varepsilon_2,||\cdot||}(\mathbf{x})\right) + \left(u_1(\mathbf{x}) - u_1^{\varepsilon_1,||\cdot||}(\mathbf{x})\right) \cdot u_2^{\varepsilon_2,||\cdot||}(\mathbf{x})| \\
&\leq |u_1(\mathbf{x})| \cdot \tilde{\varepsilon}_2 + \tilde{\varepsilon}_1 \cdot |u_2^{\varepsilon_2,||\cdot||}(\mathbf{x})|
\end{aligned}$$

Analogously, we obtain

$$|u_1(\mathbf{x}) \cdot u_2(\mathbf{x}) - u_1^{\varepsilon_1,||\cdot||}(\mathbf{x}) \cdot u_2^{\varepsilon_2,||\cdot||}(\mathbf{x})| \quad \leq \quad \tilde{\varepsilon}_1 \cdot |u_2(\mathbf{x})| + |u_1^{\varepsilon_1,||\cdot||}(\mathbf{x})| \cdot \tilde{\varepsilon}_2.$$

Summation and division by 2 gives finally

$$\begin{aligned}
&|u_1(\mathbf{x}) \cdot u_2(\mathbf{x}) - u_1^{\varepsilon_1,||\cdot||}(\mathbf{x}) \cdot u_2^{\varepsilon_2,||\cdot||}(\mathbf{x}))| \leq \\
&\leq \frac{1}{2}\left(\left(|u_2(\mathbf{x})| + |u_2^{\varepsilon_2,||\cdot||}(\mathbf{x})|\right) \cdot \tilde{\varepsilon}_1 + \left(|u_1(\mathbf{x})| + |u_1^{\varepsilon_1,||\cdot||}(\mathbf{x})|\right) \cdot \tilde{\varepsilon}_2\right) \\
&\leq \max\left(|u_2(\mathbf{x})|, |u_2^{\varepsilon_2,||\cdot||}(\mathbf{x})|\right) \cdot \tilde{\varepsilon}_1 + \max\left(|u_1(\mathbf{x})|, |u_1^{\varepsilon_1,||\cdot||}(\mathbf{x})|\right) \cdot \tilde{\varepsilon}_2
\end{aligned}$$

□

An analogous result (with obvious modifications and restrictions, division by zero) can be shown for the division of two functions.

3.2 Differential operators and finite differences

In this section we want to develop finite difference operators for second order elliptic partial differential operators L of the type

$$L = \sum_{i=1}^{d}\sum_{j=1}^{d} a_{ij}(\mathbf{x}) \frac{\partial}{\partial x_i}\frac{\partial}{\partial x_j} + \sum_{i=1}^{d} b_i(\mathbf{x}) \frac{\partial}{\partial x_i} + c(\mathbf{x}) \qquad (17)$$

in d dimensions. The application of the operator L to u involves summations and multiplications of functions and first and second order derivatives. In the previous

subsection, we described how to implement the summation and multiplication of discrete functions. What is left is the realization of the derivatives.
We consider the differentiation of a functions u in hierarchical representation. Now the differentiation of the basis functions is required, i.e.

$$\frac{\partial u}{\partial x_j} = \frac{\partial}{\partial x_j} \sum_{\mathbf{l},\mathbf{i}} u_{l,i} \cdot \phi_{\mathbf{l},\mathbf{i}}(\mathbf{x}) = \sum_{\mathbf{l},\mathbf{i}} u_{l,i} \cdot \frac{\partial \phi_{\mathbf{l},\mathbf{i}}(\mathbf{x})}{\partial x_j},$$

but this results, except for the constant function case, not in a representation of the type (4) with basis functions $\phi_{\mathbf{l},\mathbf{i}}$. For the respective coordinate direction, we could switch to the Haar system instead or we could try to express each differentiated basis function (which is, for the respective coordinate direction, locally a constant function involving jumps) by means of the hierarchical representation in an infinite series, i.e.

$$\frac{\partial \phi_{\mathbf{l},\mathbf{i}}(\mathbf{x})}{\partial x_j} =: \sum_{\mathbf{m},\mathbf{r}} w_{\mathbf{m},\mathbf{r}}^{\mathbf{l},\mathbf{i}} \cdot \phi_{\mathbf{m},\mathbf{r}}(\mathbf{x}),$$

but the computation of the coefficients $w_{\mathbf{m},\mathbf{r}}^{\mathbf{l},\mathbf{i}}$ and their reordering and summation is costly and complicated. Furthermore, in the finite dimensional case, this again would require an infinite series representation for the result due to the jumps.
Therefore, we proceed as follows: For a derivative in the $j-th$ coordinate direction, we apply $\mathcal{E}_j$ to the hierarchical coefficients, i.e. we perform a transformation to nodal representation but *only* with respect to the $j-th$ coordinate direction. Then, for every interior grid point, we apply a standard 1D difference stencil. It is chosen as the *narrowest* stencil (in the j-th coordinate direction) available on the sparse grid. Finally we use $\mathcal{H}_j$ to obtain the representation of the result in hierarchical basis. For regular sparse grids, compare (9), the well-known second order finite difference stencil for the first derivative (*centered difference*)

$$\frac{1}{2 \cdot 2^{-lmax_j}} \left[\begin{array}{ccc} -1 & 0 & 1 \end{array} \right]_{x_{l_j,i_j},lmax_j} \tag{18}$$

or the both first order stencils (*backward or forward difference*)

$$\frac{1}{2^{-lmax_j}} \left[\begin{array}{ccc} -1 & 1 & 0 \end{array} \right]_{x_{l_j,i_j},lmax_j}, \qquad \frac{1}{2^{-lmax_j}} \left[\begin{array}{ccc} 0 & -1 & 1 \end{array} \right]_{x_{l_j,i_j},lmax_j} \tag{19}$$

can be applied where $lmax_j := n + d - 1 - \sum_{j'=1,j'\neq j}^{d} l_{j'}$. For the approximation of a second derivative we consider the usual stencil

$$\frac{1}{2^{-2 \cdot lmax_j}} \left[\begin{array}{ccc} 1 & -2 & 1 \end{array} \right]_{x_{l_j,i_j},lmax_j}. \tag{20}$$

These stencils belong for each interior grid point to an equidistant grid (in the j-th coordinate direction) with local mesh size 2^{-lmax_j}.
In the adaptive refinement case, we have in general no longer an equidistant grid. Then, the stencil for each node is still chosen as the narrowest finite difference stencil (in the j-th coordinate direction) available on the adaptive sparse grid. Its entries are now the coefficients known from finite differences on non-uniform grids.
We obtain the operator

$$\frac{\partial^2}{\partial x_j^2} \approx \mathcal{D}_{jj}^S := \mathcal{H}_j \circ \mathcal{D}_{jj} \circ \mathcal{E}_j$$

for the second derivative in direction j and the operators

$$\frac{\partial^{+,-,0}}{\partial x_j} \approx \mathcal{D}_j^{S,+,-,0} := \mathcal{H}_j \circ \mathcal{D}_j^{+,-,0} \circ \mathcal{E}_j$$

for the first derivative in direction j, respectively. Here, in the regular sparse grid case, $\mathcal{D}_{jj}$ represents the application of the one-dimensional stencil (20) for coordinate direction j in each interior grid point and $\mathcal{D}_j^{0,+,-}$ represents the application of the one-dimensional stencils (18) and (19), respectively, for coordinate direction j in each interior grid point. In the adaptive case, their non-uniform analogues are taken.
For example, the costs of an application of the d-dimensional Laplacian $\sum_{j=1}^d \mathcal{D}_{jj}^S$ consist of the costs of the operators $\mathcal{H}_j$, $\mathcal{D}_{jj}$, $\mathcal{H}_j$ and the costs of the summation over d. To be precise, we have

$$d \cdot (cost(\mathcal{H}_j) + cost(\mathcal{D}_{jj}) + cost(\mathcal{E}_j)) + (d-1) \cdot cost(+).$$

Let M denotes the total amount of sparse grid points. Since $cost(\mathcal{H}_j) = cost(\mathcal{E}_j) = 3M$, $cost(\mathcal{D}_{jj}) = 4M$ (or $6M$ in the adaptive case, respectively) and $cost(+) = M$ (neglecting boundary effects), we obtain altogether a work count of

$$work(\sum_{j=1}^{d} \mathcal{D}_{jj}^S) = (11 \cdot d - 1) \cdot M$$

or $(13 \cdot d - 1) \cdot M$ floating point operations, respectively. Additionally, besides $(2M + 2M) \cdot d$ simple integer in-/decrements (get neighboring points in $\mathcal{H}_j$ and $\mathcal{E}_j$, $j = 1, ..., d$) and a few integer operations plus some tricky bit mask operations (get nearest neighboring points in $\mathcal{D}_{jj}$, $j = 1, .., d$), the integer operation overhead according to (15) for a total of approximately $10d \cdot M$ hash table accesses must be taken into account. In the general case of adaptively refined sparse grids, finding the index pairs of the nearest neighboring points in $\mathcal{D}_{jj}$ is more involved. Here, an alternative implementation possibility is to store and update these indices explicitly and thus to trade work against storage. Note that the overall work count still compares favorable with many other existing adaptive implementations for two- and three-dimensional problems.
We also can switch from the hierarchical representation to the representation in terms of pure nodal values in each grid point via an equivalency transformation $\mathcal{E} \circ \mathcal{D}_{j(j)}^{S,(+,-,0)} \circ \mathcal{H}$. We obtain

$$\hat{\mathcal{D}}_{jj}^S := \mathcal{E} \circ \mathcal{H}_j \circ \mathcal{D}_{jj} \circ \mathcal{E}_j \circ \mathcal{H} = \mathcal{E}_{\neq j} \circ \mathcal{D}_{jj} \circ \mathcal{H}_{\neq j} \tag{21}$$

for the second derivative in direction j and

$$\hat{\mathcal{D}}_j^{S,+,-,0} := \mathcal{E} \circ \mathcal{H}_j \circ \mathcal{D}_j^{+,-,0} \circ \mathcal{E}_j \circ \mathcal{H} = \mathcal{E}_{\neq j} \circ \mathcal{D}_j^{+,-,0} \circ \mathcal{H}_{\neq j}$$

for the first derivatives in direction j, respectively. Here, $\mathcal{H}_{\neq j} := \mathcal{H}_j \circ \mathcal{E}$ denotes the transformation from nodal representation to hierarchical representation (in the product sense) for all directions except the j-th coordinate direction, and $\mathcal{E}_{\neq j} := \mathcal{E}_j \circ \mathcal{H}$ denotes the analogue for the inverse transformation. Note that the spectral properties of both representations are the same due to $\mathcal{E} = \mathcal{H}^{-1}$.
At least for the case of the regular sparse grid, using Taylor series expansion for the difference stencil, the transformation and its inverse, plugging these into each other and performing a lengthy and technical computation, the consistency order of our hierarchical difference operators can be proven, see [37] for details.

Theorem 3: *The operator $\hat{\mathcal{D}}_{jj}^S$ is a second order consistent discretization for the second derivative on the regular sparse grid of level n, $1 \le j \le d$, i.e.*

$$||\hat{\mathcal{D}}_{jj}^S R^S u - \tilde{R}^S \Delta u||_\infty \le c \cdot h^2 ||u||_{C^{3+d-1,1}(\bar{\Omega})} \tag{22}$$

for all $u \in C^{3+d-1,1}(\bar{\Omega})$. *Here,* $h = 2^{-n}$ *and* R^S *and* $\tilde{R}^S$ *denote pointwise restriction mappings to the sparse grid points of level* n.

The operator $\hat{\mathcal{D}}_j^{S,0}$ *is a second order consistent discretization for the first derivative on the regular sparse grid of level* n, *and the operators* $\hat{\mathcal{D}}_j^{S,+,-}$ *are first order consistent discretizations for the first derivative on the regular sparse grid of level* n, $1 \leq j \leq d$.

Proof: See [37].

□

Now, as an example, we consider the discretization $\mathcal{D}_{11}^S + \mathcal{D}_{22}^S$ of the Laplacian in two dimensions on a regular sparse grid. Note first that the eigenvalues of the corresponding matrix are real numbers despite the non-symmetry of the matrix. The resulting largest and smallest eigenvalues are given in Table 1.

Table 1: Eigenvalues $\lambda_{\min}$ and $\lambda_{\max}$ and condition number κ of the discretized Laplacian $\mathcal{D}_{11}^S + \mathcal{D}_{22}^S$, regular sparse grid case.

level n	λ_{min}	λ_{max}	κ	factor
1	18.387503	69.61	3.79	—
2	19.273832	263.30	13.66	3.61
3	19.587081	1031.81	52.67	3.86
4	19.691734	4103.95	208.43	3.96
5	19.724926	16391.99	831.24	3.99
6	19.735029	65543.99	3320.36	3.99

We see that λ_{min} converges clearly to $2 \cdot \pi^2 = 19.739209$, i.e. to the smallest eigenvalue of the continuous operator. Furthermore we see from Table 1 that the condition number is proportional to $2^{2 \cdot n}$ like that of the 5 point stencil matrix on a regular uniform grid. From this and Theorem 3 (consistency), we can infer that our discretization is stable for regular sparse grids.
Mixed derivatives which are put together from second order discretizations $\hat{\mathcal{D}}_j^{S,0}$ for the first derivatives in different coordinate directions possess also consistencies of second order.
In the case of adaptively refined sparse grids, in general, we encounter non-equidistant grid spacing when we want to apply our one-dimensional stencils within the $\hat{\mathcal{D}}^S$-operators. Then, due to lack of symmetry, an order of consistency is lost (lower order terms in the Taylor series). To cope with this problem the associated finite difference 3-point stencils for non-equidistant mesh sizes must be locally modified in the same way as it is common for standard finite difference discretizations on non-equidistant grids. This is in the same spirit as the Shortley-Weller trick, see [24], pp. 78f. Also switching locally to a one-dimensional 5-point stencil helps to maintain the second order, if wanted.

4 Adaptive solution of elliptic partial differential equations

Now we are able to assemble the discrete version of our differential operator (17). It involves the summation and multiplication (by sufficiently resolved coefficient functions) and the necessary difference operators. Written in matrix notation, this would result in a linear system of equations to be solved. However, in our approach, we never assemble the system matrix. For most iterative solution methods this is

not necessary anyway. Instead, only the action of the system matrix onto the vector of an actual iterate is needed.
The corresponding matrix $\mathcal{L}^S$ is in general non-symmmetric, even if we restrict ourselves to a symmetric, selfadjoint continuous operator (17). Therefore, we use as basic iterative scheme the BiCGstab iteration. This iteration is convergent in any case. However its convergence is quite slow. To speed it up, we can apply techniques known from multigrid. Here, the BiCGstab iteration over an (adaptively refined) sparse grid serves as a smoother. To this end, similar to (21), we switch from the hierarchical representation of our operator by the equivalency transformation $\mathcal{E} \circ \mathcal{L}^S \circ \mathcal{H}$ to the nodal representation $\hat{\mathcal{L}}^S$. The right hand side is transformed accordingly. It is this transformed system to which we apply the BiCGstab method if we use it as a smoother within a multigrid method.
The coarser grids can be obtained by applying successively one step of the *compress* operator globally, i.e. without any threshold. Now switching from coarse grids to finer grids involve basically the application of the hierarchical basis interpolation, which is trivially obtained by setting the hierarchical values on finer levels to zero. This implements the prolongation operator. The implementation of the restriction operator is obvious. Altogether we have now all ingredients to set up a (multiplicative) multigrid iteration [8, 25]. On this method and its convergence properties will be reported elsewhere, especially in the case of adaptively refined sparse grids.
Besides, also additive multilevel preconditioners have been developed. They are either in the spirit of Bramble, Pasciak and Xu [7, 34] with necessary modifications [18, 19] to cope with problems posed by the tensor product construction of our hierarchical basis. Alternatively, we developed a multilevel preconditioner based on the prewavelet approach [19]. Note that, in all cases of multilevel iterative solvers, good convergence rates independent on the number of unknowns can be obtained. However, the convergence rates are not independent of the coefficient functions of the operator. Singular perturbed problems exhibit a slowing down of the multilevel solver. Thus, as for conventional multigrid methods, the problem of robustness is not yet solved.
These solvers or a few iterates of them can now be used within a cycle to adapt the grid to the solution without a priori knowledge where to refine. Solving the problem on one grid and employing an error indicator gives information, where a finer grid is needed to resolve the solution. We start with a very coarse grid and iterate the procedure, always adding new nodes. If a final error tolerance is matched, we have a solution on a fine adapted grid. The technique is the same as for other adaptive refinement methods [4, 5, 32, 36] and can be applied straightforwardly also in our case. The difference is, however, in the refinement process. We work node-oriented not element-oriented. If a node index $(\mathbf{l}, \mathbf{i})$ has been flagged for refinement then the nodes of then next 'level', which lie in its influence cone, i.e. the set of nodes with indices $\{(\mathbf{l} + \mathbf{e}_j, \mathbf{i} + (i_j \pm 1) \cdot \mathbf{e}_j), j = 1, ..d\}$ are set active. As an indicator of the local error we use the criterions due to (11) to flag a node for future refinement.

5 Numerical experiments

In the following, we present the results of numerical experiments obtained with our new sparse grid method.
First we consider the case of regular, i.e. non-adapted sparse grids.

Problem 1: We first consider the Poisson equation $-\Delta u = f$ in $\Omega = (0,1)^2$ with Dirichlet boundary conditions and the exact solution $u(x, y) = \sinh(\pi(1 - x))\sin(\pi y)/\sinh(\pi)$. The results for different levels of discretization are given in Table 2. Here and in the following, we show the number of points involved in the sparse grid finite difference discretization, the resulting approximation error with respect to the L_2- and L_∞-norms and the pointwise error in the point $(1/\pi, 1/\pi)$ and give furthermore their quotients on two successive levels. To some extent, this reveals the order of approximation. We clearly see that the accuracy behaves like $O(2^{-2\cdot n})$. This is in contrast to the sparse grid Galerkin method, see [9, 10]. There, only an accuracy of order $O(n \cdot 2^{-2\cdot n})$ is achieved in two dimensions. Note that this additional n-term is due to inherent interpolation by means of the hierarchical basis, i.e. due to the Galerkin integration of the (interpolated) right hand side.

Table 2: Error for Problem 1, regular sparse grid.

level n	points	L_∞-error	quotient	L_2-error	quotient	pointwise	quotient
2	49	3.15_{-3}	-	1.75_{-3}	-	2.74_{-3}	-
3	113	7.95_{-4}	3.96	4.34_{-4}	4.03	7.11_{-4}	3.85
4	257	2.00_{-4}	3.97	1.08_{-4}	4.02	1.80_{-4}	3.95
5	577	5.03_{-5}	3.98	2.69_{-5}	4.01	4.48_{-5}	4.02
6	1281	1.26_{-5}	4.00	6.71_{-6}	4.01	1.12_{-5}	4.00
7	2817	3.15_{-6}	4.00	1.67_{-6}	4.01	2.80_{-6}	4.00

Problem 2: Now we consider the Helmholz equation $-\Delta u + cu = f$ in $\Omega = (0,1)^2$, $c(x,y) = y/(x+0.1)$, with Dirichlet boundary conditions and the exact solution $u(x,y) = e^{x+y}$. Table 3 summarizes the results. We see that the approximation order is even better than for Problem 1. This is due to the term $c \cdot u$. In contrast to the sparse grid Galerkin method, for the discretization of the Helmholz term no mass matrix is necessary. Thus no interpolation occurs and the approximation gets better with rising value of c.

Table 3: Error for Problem 2, regular sparse grid.

level n	points	L_∞-error	quotient	L_2-error	quotient	pointwise	quotient
2	49	4.56_{-4}	-	2.55_{-4}	-	2.81_{-4}	-
3	113	1.10_{-4}	4.15	6.16_{-5}	4.14	6.93_{-5}	4.05
4	257	2.59_{-5}	4.25	1.45_{-5}	4.25	1.63_{-5}	4.25
5	577	6.05_{-6}	4.28	3.36_{-6}	4.32	3.78_{-6}	4.31
6	1281	1.39_{-6}	4.39	7.70_{-7}	4.36	8.65_{-7}	4.37
7	2817	3.16_{-7}	4.40	1.75_{-7}	4.40	1.96_{-7}	4.41

Problem 3: Next we have a look at the convection-diffusion equation $-\Delta u + \vec{\beta} \cdot \nabla u + cu = f$ in $\Omega = (0,1)^2$, $\vec{\beta} = (1,1)$, $c = 1$, with Dirichlet boundary conditions and the exact solution $u(x,y) = 4\sin(\pi x)\sin(\pi y)$. Note that, for the discretization of the ∇-term, we used the central difference approach here. The results are given in Table 4. We only see an approximation order of $O(n \cdot 2^{-2n})$ or less. The convection term reduces the accuracy somewhat, even if a central difference stencil is used. Of course for larger convection strength, we would run into trouble with this discretization (due to stability reasons) just like for a standard full grid discretization.

Problem 4: Therefore we consider a second convection diffusion equation $-\Delta u + \vec{\beta} \cdot \nabla u + cu = f$ in $\Omega = (0,1)^2$, $\beta_1 = 1 + x^2 + xy$, $\beta_2 = e^{x+y} - e^y + \sin(xy)$, $c(x,y) = \sin(x)\sin(y)$, with Dirichlet boundary conditions and the exact solution $u(x,y) = 4\sin(\pi x)\sin(\pi y)$. Now, we used the upwind approach for the discretization of the

Table 4: Error for Problem 3, regular sparse grid.

level n	points	L_∞-error	quotient	L_2-error	quotient	pointwise	quotient
2	49	2.59_{-1}	-	1.42_{-1}	-	1.79_{-1}	-
3	113	8.73_{-2}	2.97	4.65_{-2}	3.05	6.64_{-2}	2.70
4	257	2.87_{-2}	3.04	1.49_{-2}	3.12	2.24_{-2}	2.96
5	577	9.09_{-3}	3.16	4.64_{-3}	3.21	7.06_{-3}	3.17
6	1281	2.74_{-3}	3.31	1.39_{-3}	3.34	2.13_{-3}	3.31
7	2817	8.05_{-4}	3.40	4.08_{-4}	3.41	6.26_{-4}	3.40

Table 5: Error for Problem 4, regular sparse grid.

Level	points	L_∞-error	quotient	L_2-error	quotient	pointwise	quotient
2	49	3.15_{-1}	-	1.59_{-1}	-	1.36_{-1}	-
3	113	1.92_{-1}	1.64	9.78_{-2}	1.63	9.03_{-2}	1.51
4	257	1.12_{-1}	1.71	5.52_{-2}	1.77	5.26_{-2}	1.72
5	577	5.99_{-2}	1.87	2.96_{-2}	1.86	2.86_{-2}	1.84
6	1281	3.12_{-2}	1.92	1.54_{-2}	1.92	1.50_{-2}	1.91
7	2817	1.59_{-2}	1.96	7.85_{-3}	1.96	7.71_{-3}	1.95

convection terms. Then, as can be seen from Table 5, the accuracy deteriorates to first order, i.e. to about $O(n \cdot 2^{-n})$. We conducted further experiments for other model problems with large convection terms using upwind discretizations. There, the same order of approximation was observed. The results remained stable independent of the convection strength.

Now we turn to the case of adaptively refined grids.

Problem 1, adaptive case: First we reconsider Problem 1 but solve it on adaptively refined grids. Here, the solution is adapted successively using the criterion $|u_{\mathbf{l},\mathbf{i}}| \cdot 1 \geq \varepsilon$, i.e using the maximum-norm approach. Of course, the solution of Problem 1 is quite smooth. Therefore, no substantial difference to the case of the regular sparse grid can be expected. From Table 6 we see that, due to the adaptivity, even slightly more grid points are needed to reach the same absolute error size as for the regular sparse grid case of Table 2. With respect to the ratio relative error reduction versus number of grid points they are comparable.

Table 6: Error for Problem 1, adaptively refined sparse grid.

ε	points	L_∞-error	quotient	L_2-error	quotient	pointwise	quotient
0.1	12	4.23_{-1}	-	2.24_{-1}	-	2.42_{-1}	-
0.0125	38	1.21_{-2}	34.95	4.49_{-3}	49.88	8.95_{-3}	27.40
0.25_{-2}	105	2.33_{-3}	5.19	8.06_{-4}	5.57	2.14_{-4}	41.82
0.3_{-3}	354	3.83_{-4}	6.08	1.78_{-4}	4.53	1.76_{-5}	12.16
0.35_{-4}	1207	9.09_{-5}	4.21	4.80_{-5}	3.71	3.84_{-5}	0.458
0.35_{-5}	4448	1.53_{-5}	5.94	8.13_{-6}	5.90	5.17_{-6}	7.43

Problem 5: Finally we consider a convection-diffusion problem with non-constant convection coefficients which involves boundary layers. These must be resolved adaptively. We deal with the equation $-\Delta u + \vec{\beta} \cdot \nabla u + cu = f$ in $\Omega = (0,1)^2$, $\vec{\beta} = (e^{3xy}, \sin(\pi(x+y)))$, $c = 1$, with Dirichlet boundary conditions and the exact solution $u(x,y) = x^3 \cdot e^{3.5\pi y}$. Figure 5 shows the solution of this problem and also an example of a sparse grid produced by adaptive refinement. Here, again, we used the simple error indicator which is based on the maximum norm. It seems to work reasonable also in case of convection-diffusion. The results are summarized in

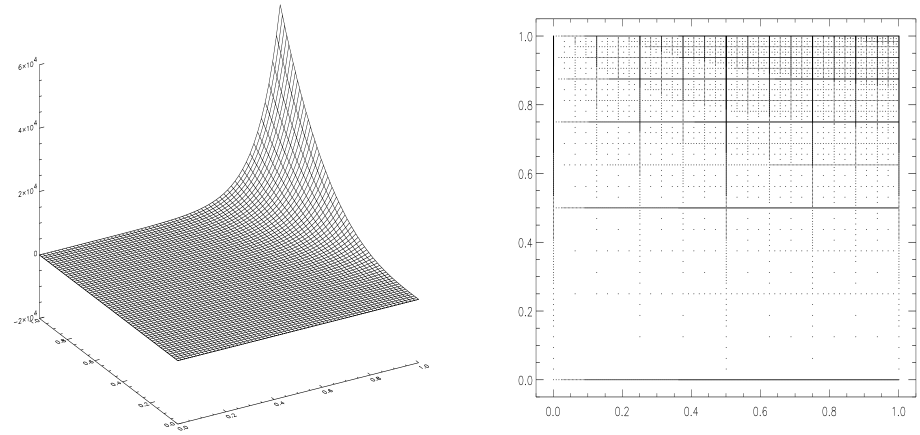

Figure 5: Solution (left), and adaptively refined grid (right), $\varepsilon = 0.03052$.

Table 7. For the discretization, we used the upwind approach to be on the safe side.

Table 7: Error for Problem 5, adaptively refined sparse grid, upwind difference approach.

ε	points	L_∞-error	quotient	L_2-error	quotient	pointwise	quotient
500	47	1.87_{+3}	-	4.67_{+2}	-	2.65_{+2}	-
125	106	1.84_{+2}	10.16	7.78_{+1}	6.00	1.06_{+2}	2.50
31.25	233	8.86_{+1}	2.08	2.39_{+1}	3.26	1.51_{+1}	7.02
7.813	516	3.06_{+1}	2.90	1.73_{+1}	1.38	2.82_{+1}	0.54
1.953	1135	1.38_{+1}	2.22	6.69_{+0}	2.59	1.11_{+1}	2.54
0.488	2475	6.98_{+0}	1.98	3.33_{+0}	2.01	5.60_{+0}	1.98

In comparison to the results of Table 5 (also upwind discretization and moderate convection but on a regular sparse grid without any adaptivity), now, a much better relative accuracy of the solution is obtained with less grid points, i.e. adaptivity pays off and helps to approximately regain the convergence order (i.e. error versus number of grid points) of a smooth solution case. Note that in further experiments a local convex linear combination of the upwind and central difference operator similar to the flux difference splitting approach gave also good results with a better approximation order.

6 Concluding remarks

In this paper, we discussed the basic features of a simple adaptive method to solve elliptic PDEs on sparse grids using a finite difference approach. We developed an algebra of operations and we implemented differential operators for finite dimensional function representations. These representations are stored and manipulated using a hash table technique. Due to the tensor product approach, the method can be easily generalized to the higher-dimensional case. Although we gave only results for two-dimensional problems, the method works also fine for three- and even higher-dimensional PDEs. We recently treated 9-dimensional problems.
Surely, the rectangular domain is a restriction of the method. Note however that this disadvantage can be partly avoided by adaptively resolving complicated domains according to their characteristic functions or by the transformation technique as suggested in [10].
Note finally, that the interplay between the threshold ε used in the error indicators (11) and the achieved error accuracy is not fully clear for singular solutions. Here, more work is necessary for the construction of global error estimators from the local indicators we use in our sparse grid discretizations up to now.

Acknowledgement: I would like to thank Thomas Schiekofer very much for programming the new approach and providing the data of the numerical experiments.

References

[1] F. Arandiga, R. Donat, A. Harten, Multiresolution based on weighted averages of the hat function: Linear reconstruction, CAM Report 96-25, Dept. of Mathematics, UCLA, 1996.

[2] K.I. Babenko, Approximation by trigonometric polynomials in a certain class of periodic functions of several variables, Dokl. Akad. Nauk SSSR, 132, 982-985, (Russian); 672-765 (English translation), 1960.

[3] R. Balder, Adaptive Verfahren für elliptische und parabolische Differentialgleichungen auf dünnen Gittern, Dissertation TU München, 1994.

[4] R. E. Bank and T. F. Dupont. An optimal order process for solving elliptic finite element equations. Math. Comp., 36, 967-975, 1981.

[5] P. Bastian. Parallele adaptive Mehrgitterverfahren. B. G. Teubner, Stuttgart, 1996.

[6] G. Baszenski, F.J. Delvos, S. Jester, Blending approximation with sine functions, in Numerical Methods of Approximation 9, D. Braess and L.L. Schumaker eds., Int. Ser. Num. Math., 105, 1-19, 1992.

[7] J. Bramble, J.P. Pasciak, J. Xu, Parallel multilevel preconditioners, Math. Comp. 31, 333-390, 1990.

[8] A. Brandt, Guide to multigrid development, in Multigrid Methods, W. Hackbusch and U. Trottenberg eds., Lecture Notes in Mathematics 960, Springer, Berlin, Heidelberg, New York, 1982.

[9] H.J. Bungartz, Dünne Gitter und deren Anwendung bei der adaptiven Lösung der dreidimensionalen Poisson-Gleichung, Dissertation, Institut für Informatik, TU München, 1992.

[10] H.J. Bungartz, T. Dornseiffer, Sparse grids: Recent developments for elliptic partial differential equations, Rep. TUM-I9702, Institut für Informatik, TU München, 1997, submitted to Proc 5th Europ. Conf. on Multigrid Methods, Stuttgart, 1996.

[11] H. Bungartz, M. Griebel, A note on the complexity of the Poisson equation and related elliptic equations for spaces of bounded mixed derivative, Report SFB256 No 524, Uni Bonn, 1997, submitted to J. Complexity.

[12] S. Dahlke, Besov regularity for Dirichlet problems for divergence form operators in Lipschitz domains, Report 138, Institut für Geometrie und Praktische Mathematik, RWTH-Aachen, 1997.

[13] F.J. Delvos, d-Variate Boolean Interpolation, J. Approx. Theory, 34, 99-114, 1982.

[14] G. Faber, Über stetige Funktionen, Mathematische Annalen 66, 81-94, 1909.

[15] W.J. Gordon, Distributive lattices and the approximation of multivariate functions, in Approximation with Special Emphasis on Spline Functions, I.J. Schoenberg ed., Academic Press, New York, 223-277, 1969.

[16] M. Griebel, A parallelizable and vectorizable multi-level algorithm on sparse grids, in Parallel Algorithms for Partial Differential Equations, Proceedings of the Sixth GAMM-Seminar, Kiel, 1990, Notes on Numerical Fluid Mechanics 31, 94-199, W. Hackbusch ed., Vieweg Verlag, Braunschweig, 1991.

[17] M. Griebel, Multilevelmethoden als Iterationsverfahren über Erzeugendensystemen, Teubner Skripten zur Numerik, Teubner Verlag, Stuttgart, 1994

[18] M. Griebel, P. Oswald, On additive Schwarz preconditioners for sparse grid discretization, Numer. Math., 66(4), 449-464, 1994.

[19] M. Griebel, P. Oswald, Tensor-product-type subspace splittings and multilevel iterative methods for anisotropic problems, Advances in Computational Mathematics, 4, 171-206, 1995.

[20] M. Griebel, M. Schneider, C. Zenger, A combination technique for the solution of sparse grid problems, in P. de Groen und R. Beauwens, Editoren, Iterative Methods in Linear Algebra, S. 263-281, IMACS, Elsevier, North Holland, 1992.

[21] M. Griebel, S. Zimmer, C. Zenger, Multilevel Gauss-Seidel-algorithms for full and sparse grid problems, Computing 49, 127-148, 1993.

[22] M. Griebel, G. Zumbusch, Hash-storage techniques for adaptive multilevel solvers and their domain decomposition parallelization, Proceedings of the Tenth International Conference on Domain Decomposition Methods Boulder, Colorado , USA, August 10-14, 1997.

[23] M. Griebel, G. Zumbusch, Parallel multigrid in an adaptive PDE solver based on hashing, Proceedings ParCo'97, Bonn, September 1997.

[24] W. Hackbusch, Theorie und Numerik elliptischer Differentialgleichungen, Teubner Studienbücher Mathematik, Teubner-Verlag, Stuttgart, 1986.

[25] W. Hackbusch, Multigrid Methods and Applications, Springer-Verlag, Berlin, Heidelberg, New York, 1985.

[25] A. Harten, Multiresolution representation and numerical algorithms: A brief review, NASA ICASE report No 94-59, 1994.

[26] P. Hemker, C. Pflaum, Approximation on partially ordered sets of regular grids, Applied Num. Anal. 25, 55-87, 1997.

[27] P. Hemker, P. de Zeeuw, BASIS3: A data structure for 3-dimensional sparse grids. Technical Report NM-R9321, CWI Amsterdam, The Netherlands, 1993.

[28] J.P. Hennnart, E.M. Mund, On the h- and p-versions of the extrapolated Gordon's projector with applications to elliptic equations, SIAM J. Sci. Stat. Comput., 9, 773-791, 1988.

[29] M. Holschneider, Localization properties of wavelet transforms, J. Math. Phys 34, 3227-3244, 1993.

[30] M. Holschneider, P. Tchamitchian, Pointwise regularity of Riemann's 'nowhere differentiable' function, Inventiones Mathematicae, vol 15, 157-175, 1991.

[31] D. E. Knuth. The Art of Computer Programming, volume 3, chapter 6.4. Addison-Wesley, 1975.

[32] W. F. Mitchell. A comparison of adaptive refinement techniques for elliptic problems. ACM Trans. Math. Software, 15, 326–347, 1989.

[33] H. Niederreiter, Random number generation and quasi-Monte Carlo methods, CBMS-NFS Regional conference series in applied mathematics, 63, SIAM, 1992.

[34] P. Oswald, Multilevel Finite Element Approximation: Theory and Applications. Teubner Skripten zur Numerik, Teubner, Stuttgart, 1994.

[35] P. J. Plauger, A. Stepanov, M. Lee, and D. Musser. The Standard Template Library. Prentice-Hall, 1996.

[36] M. C. Rivara. Algorithms for refining triangular grids suitable for adaptive and multigrid techniques. Internat. J. Numer. Methods Engrg., 20, 745–756, 1984.

[37] T. Schiekofer, Die Methode der Finiten Differenzen auf Dünnen Gittern zur adaptiven Multilevel-Lösung partieller Differentialgleichungen, Dissertation Universität Bonn, Institut für Angewandte Mathematik, to appear 1998.

[38] S.A. Smolyak, Quadrature and interpolation formulas for tensor products of certain classes of functions. Dokl. Akad. Nauk SSSR, 148, 1042-1045 (Russian), 240-243 (English translation), 1963.

[39] G.W. Wasilkowsi, H. Wozniakowski, Explicit cost bounds of algorithms for multivariate tensor product problems, J. Complexity, 11, 1–56, 1995.

[40] H. Yserentant, On the multi-level splitting of finite element spaces, Numer. Math., 49, 379-412, 1986.

[41] H. Yserentant, Hierarchical bases, in Proc. ICIAM91, Washington, R.E. O'Malley et al. eds., SIAM, Philadelphia, 1992.

[42] C. Zenger, Sparse grids, in Parallel Algorithms for Partial Differential Equations, Proceedings of the Sixth GAMM-Seminar, Kiel, 1990, Notes on Numerical Fluid Mechanics 31, W. Hackbusch ed., Vieweg Verlag, Braunschweig 1991.

Michael Griebel,
Institut für Angewandte Mathematik,
Universität Bonn,
D-53115 Bonn, Wegelerstr. 6,
e-mail: griebel@iam.uni-bonn.de