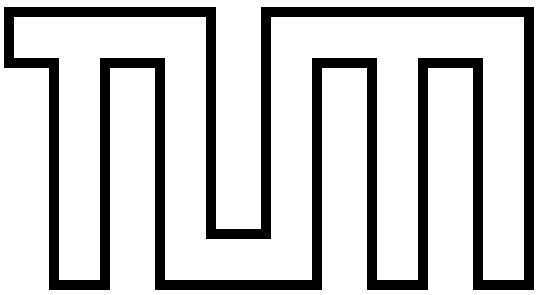


FAKULTÄT FÜR INFORMATIK

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

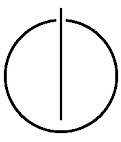
Interdisziplinäres Projekt

Integration of Prewavelet Ansatz Functions into SG++

Richard Philipp Röttger

Supervisor: Univ.-Prof. Dr. Hans-Joachim Bungartz

Advisor: Dipl.-Inf. Dirk Pflüger



Contents

1 Summary **3**

2 Scientific Background **3**
2.1 Hierarchical Hat Basis . . . 3
2.2 Sparse Grids . . . 4

3 Introduction to Prewavelets **5**
3.1 Motivation . . . 5
3.2 Definition . . . 6
3.3 Advantages and Drawbacks . . . 7
3.4 Usage of Prewavelets . . . 8

4 Algorithms **8**
4.1 Get Affected Basis Functions . . . 8
4.2 Hierarchization and Dehierarchization . . . 10
4.2.1 Transformation into the Hat Basis . . . 10
4.2.2 Transformation into the Prewavelet Basis . . . 11
4.3 Laplace . . . 13
4.3.1 Gradient Part . . . 14
4.3.2 Non-Gradient Part . . . 16

5 Adaptive Sparse Grids **17**

6 Implementation **18**

7 Application **19**
7.1 Stationary Heat Equation . . . 19
7.2 Results . . . 20
7.2.1 Runtime Behavior . . . 20
7.2.2 Convergence Speed . . . 20

8 Conclusion **25**

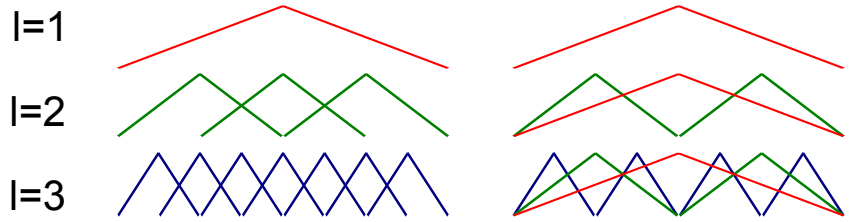


Figure 1: The normal hat basis (left) and the hierarchical counterpart (right) for several levels.

1 Summary

In the framework of this interdisciplinary project (IDP) we integrated prewavelets as ansatz functions into the SG++ library. SG++ is a library allowing the user to use sparse grids within a variety of different applications. The package does not only provide methods for the creation of sparse grids, but also algorithms for the most common functions like hierarchization or the application of the Laplace matrix. Furthermore, SG++ itself already supports several different ansatz function.

As the application of the Laplace matrix is algorithmically expensive, several attempts have been made to ease the application of this matrix. In this IDP we used prewavelets, which are semi-orthogonal ansatz functions and likely to reduce the calculation efforts for the Laplace matrix. On the downside, the usage of prewavelets leads to more complex algorithms for several other tasks, like the hierarchization. In this work, we describe the different algorithms which were implemented and emphasize the difference to the algorithms to the normal hat basis.

The performance of the prewavelet ansatz functions was tested with the stationary heat equation. Here, the convergence behavior was of special interest. Unfortunately it turned out that the convergence behavior with prewavelets is inferior to the normal version both in the one-dimensional case as well as for higher-dimensional problems. That is surprising, especially as the condition number for the Laplace matrix in higher dimensions is significant smaller in comparison to the standard version. Furthermore, the more complex algorithms caused by the usage of prewavelets led to a slowdown of the calculations.

2 Scientific Background

2.1 Hierarchical Hat Basis

In order to approximate a function with a piecewise linear function, the most straightforward approach is to discretize the function space with a regular full grid and use hat ansatz functions at all grid points. Thus, the approximation is obtained as

$$\hat{f}(x) = \sum \alpha_i \phi_i(x)$$

with α_i being the values of the real function at the grid point i and ϕ_i denoting the ith hat function. Figure 1 depicts the hat functions for several equidistant discretizations of the one-dimensional function space $\Omega = [0, 1]$. In this work we consider $f(x) = 0$ for all x on the border $\partial\Omega$ of the function space Ω, and for simplification we define the space as $\Omega := [0, 1]^d$.

This straightforward approach has the disadvantage, that, for example for the calculation of the integral of $\hat{f}$, all results have to be omitted whenever a higher resolution is chosen. This can be solved through the use of hierarchical basis functions. Figure 1 also depicts the hierarchical basis functions. In the figure it can also be seen, that the hierarchical basis utilizes hat functions of different levels. In order to form a basis the grid points with an odd index have to be omitted in each level, as these points are already covered by basis functions of the levels above.

In contrast to the straightforward approach, in the hierarchical approach the different α are no longer directly corresponding to the function value at the grid point. The approximated function itself is then calculated by

$$\hat{f}(x) = \sum \alpha_{\mathbf{l},\mathbf{i}} \cdot \phi_{\mathbf{l},\mathbf{i}}(x)$$

where $\alpha_{\mathbf{l},\mathbf{i}}$ denotes the hierarchical surpluses of the according ansatz function ϕ_i. The pair of multi-indices $\mathbf{l}, \mathbf{i}$ with $\mathbf{l} = \{l_1, \ldots, l_d\}$ and $\mathbf{i} = \{i_1, \ldots, i_d\}$ corresponds to the i_kth grid point on the l_kth level in dimension k. The process of calculating these $\alpha_{\mathbf{l},\mathbf{i}}$ in a way that $\hat{f}(x) = f(x)$ holds true for every grid point x is called hierarchization, the reverse process is the so-called dehierarchization. `SG++` provides several methods to use these approximated functions, for example evaluating them at a given point x or applying the Laplace operator.

2.2 Sparse Grids

As already mentioned, `SG++` is a library which provides sparse grids and algorithms optimized for them. Here, we only give the most important facts about sparse grids. The reader can find an exhaustive discussion in [2]. The main advantage of sparse grids is that they break the curse of dimensionality. If we encounter a high dimensional problem, the discretization employing a normal grid would require $O\left(h_n^{-d}\right)$ grid points. Here d denotes the dimensionality of the problem, $h_n = 2^{-n}$ the mesh size in each dimension. With the utilization of sparse grids, the number of grid points can be reduced to as low as $O\left(h_n^{-1} n^{d-1}\right)$ whereas the quality of the approximation remains nearly the same. This allows tackling high-dimensional problems, for example for classification or finance mathematics.

Basically, sparse grids emerge by omitting subspaces with a bad cost-benefit ratio, thus by leaving out several "unimportant" points. It is always favorable to build algorithms for d-dimensional problems out of their one-dimensional counterparts and combine the results of each dimension to the final result. On the other hand, this method requires that

for every point in the grid, the neighbors in all dimensions are available. On sparse grids, this is not true, thus the simple combination of the results of each dimension does not lead to correct results. To handle this problem, the unidirectional principle was introduced, which ensures the correct data transport between the different grid points in sparse grids. Exhaustive explanations of this method can be found in [2, 9]. Here, we only describe the general principle. As already mentioned, sparse grids lack a dense neighborship on the one hand, but on the other hand every hierarchical father in every dimension in fact is in the grid. Thus, we have to ensure, that the information transport is realized by means of the hierarchical ancestors and not neighbors. In order to realize this idea, the algorithms are split in an up and down-part: The up-part considers the influence of all grid points in the current dimension which are hierarchically below the current level. The down-part considers the influence of all hierarchical ancestors and of all points with the same level. Now, in order to combine the one-dimensional algorithms to work on a d-dimensional problem as well, the unidirectional algorithm ensures that before the application of the down-part all necessary up-parts have already been calculated. Thus all information is "stored" in the hierarchical father and can now be transported back by means of the down-algorithm.

3 Introduction to Prewavelets

3.1 Motivation

The standard hierarchical hat basis ansatz functions have the advantage of having a disjoint support on every level and further more, they are derived from a quite simple pattern. Moreover, many algorithms benefit from the disjoint support of these basis functions. On the other hand, they do not form an orthogonal basis. A basis is called orthogonal, if for any basis functions ϕ_i and ϕ_k with $k \neq i$,

$$\langle \phi_i, \phi_k \rangle = 0$$

holds true for a given scalar product. In this work, we limit our efforts to the L_2 scalar product which is defined by

$$\langle \phi_i, \phi_k \rangle = \left(\int_0^1 \phi_i(x) \phi_k(x) dx \right)^{\frac{1}{2}}.$$

An orthogonal basis has many advantages compared to a non-orthogonal basis. As an obvious point, the calculation of the scalar product itself is eased, as nearly all components are zero. This fact has practical relevance, as the calculation of the scalar product is part of many algorithms, for example in order to solve PDEs [3] or for classification by regression [9]. Another advantage is that in an environment with an orthogonal basis the Pythagorean theorem holds true, thus many calculations and transformations can be performed, for example

$$\left\| \sum \phi_i \right\|^2 = \sum \|\phi_i\|^2 .$$

Put in other words, an orthogonal basis has several preferable properties. On the other hand, the basis functions themselves should be as simple as possible, with a compact support and, furthermore, still cover the same function space as the hierarchical hat basis. Unfortunately, such a basis does not exists; thus a compromise between orthogonality and simplicity has to be found: semi-orthogonality. A basis is called semi-orthogonal if

$$\forall \phi_{i,l}, \phi_{j,k}, l \neq k : \langle \phi_{i,l}, \phi_{j,k} \rangle = 0$$

holds true. That means, that the semi-orthogonal basis functions are only orthogonal to basis functions of different hierarchical levels. The prewavelet basis functions fulfill these requirements. With that compromise, the basis stays manageable and several algorithms benefit from the semi-orthogonality. In the next section, we describe, how such a basis can be derived.

3.2 Definition

In the following, the basis functions of the linear hat basis are denoted as $\phi_{l,i}$ whereas the basis functions of the prewavelet basis are denoted as $\psi_{l,i}$. As already stated in the foregoing chapter, the prewavelet basis has to fulfill the semi-orthogonality and should be as "simple" as possible. Now we define the term "simple" with conditions which have to be met:

1. The basis functions have to cover the same function space as the linear hat functions. Thus, we can write the prewavelets as a combination of hat functions:

   $$\psi_{l,i} = \sum_j \beta_j \phi_{l,i+j}.$$

   The prewavelets are only built of hat functions of one level. Note that we also consider hat basis functions of the generating system, which are not included in the actual hierarchical hat basis.

2. We also want to have a recurring motive, thus prewavelets of different levels have the same factors β_j and differ only in size of the support but not in shape.

3. We want only to differentiate two types of prewavelets: (a) border prewavelets and (b) standard prewavelets. This condition implies that there are only three β_j available to construct border prewavelets, as level two has only three possible hat functions. The same argument applies for the normal prewavelets and level three, which leads to five β_j.

4. The normal prewavelets should be symmetric, that means $\beta_{-2} = \beta_2$ and $\beta_{-1} = \beta_1$.

5. And, of course, the semi-orthogonality has to be fulfilled.

Each of these conditions lead to at least one linear equation, which can be used to determine the different β_j. The result of this process is the prewavelet basis:

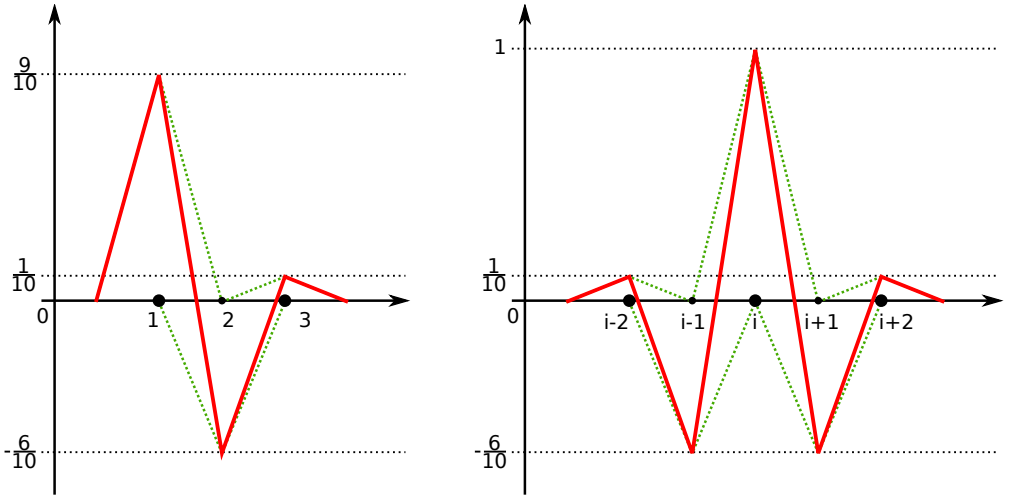


Figure 2: Prewavelet mother functions for the left boundary and a normal prewavelet (solid red line) in company with the used normal hat basis (dashed green line). The big dots indicate the position of the neighboring prewavelet, the thin dots indicate the skipped points of the sparse grid.

$$
\begin{aligned}
\psi_{1,1} &= \phi_{1,1} \\
\psi_{l,1} &= \frac{9}{10}\phi_{l,1} - \frac{6}{10}\phi_{l,2} + \frac{1}{10}\phi_{l,3} \\
\psi_{l,2^l-1} &= \frac{9}{10}\phi_{l,2^l-1} - \frac{6}{10}\phi_{l,2^l-2} + \frac{1}{10}\phi_{l,2^l-3} \\
\psi_{l,i} &= \phi_{l,i} - \frac{6}{10}\phi_{l,i\pm1} + \frac{1}{10}\phi_{i\pm2} \quad \text{for } i \neq 1 \text{ and } i \neq 2^l - 1
\end{aligned}
$$

Figure 2 depicts prewavelets and the generating linear hat basis functions. Please note, that the prewavelets have a compact support, but no longer a disjoint support on each level. In chapter 4, we show the consequences of this fact for the algorithms.

3.3 Advantages and Drawbacks

The prewavelets as described above have several advantages compared to the normal hat basis but also some drawbacks have to be considered.

- Because of the semi-orthogonality of the prewavelet basis functions, some algorithms (please see chapter 4 for details) turn out to be less complex than with the normal hat basis.
- The normal hierarchical hat basis allows a relative cheap evaluation of differential operators and discretization schemes and upper error estimators can be derived.

But there is no lower error estimator, which means the absolute value of the hierarchical coefficient is just a local error indicator, but no true error estimator. This can be avoided with the use of prewavelets, as they form a stable L_2 multiscale basis [2].

The consequences are twofold: First, we assume to speed up the calculation of the Laplace matrix as the non-gradient part of the calculation is eased. And second, as the hierarchical coefficients are a true error estimator we assume an improvement in the efficiency of the adaptive refinement of the sparse grid. These advantages are bought with a wider support of the prewavelet basis functions. The consequences imply:

- Evaluation also requires the consideration of the neighboring basis functions.
- Hierarchization cannot directly be performed in an efficient way. Thus, a hierarchization in the normal hat basis is needed. In a second step, the hierarchical hat basis can efficiently be transformed into the prewavelet basis.

3.4 Usage of Prewavelets

Prewavelets have been applied to solve problems in a variety of fields. Especially, if the semi-orthogonal property promises a simplification and thus a speed-up of the underlying algorithms, these basis functions were chosen. Among them are:

- Solving triangulation problems [4, 5, 6, 7].
- Solving differential equations [1, 8]. In the framework of this IDP, we used prewavelets to calculate the result of the stationary heat equation too, but not for the sake of solving the equation but to grasp the convergence behavior of the Laplace operator compared to the normal hat basis.
- Solving elliptic partial differential equations [3].

4 Algorithms

The following sections, we present the algorithms implemented into `SG++`. Some of these algorithms were taken from [3].

4.1 Get Affected Basis Functions

In order to evaluate the function value at an arbitrary point $x \in [0,1]^d$, it would be possible to directly calculate it by

$$\hat{f}(x) = \sum \alpha_i \cdot \phi_i(x).$$

But due to the fact that the ansatz functions have a limited support, only a small fraction of all ansatz functions has to be evaluated. With the normal hat basis, only one ansatz

Algorithm 1 Algorithm to find all affected basis functions for the evaluation of point x.

```
//x: arbitrary location, where the grid should be evaluated
//current_dim: Needed to avoid adding points twice
//iterator points to the current grid point
//results contains the affected grid points and the result of the evaluation.
func GetAffected(x, current_dim, iterator, result)
     result = result ∪ [iterator, iterator.evaluate_at(x)]
     for d from current_dim to max_dim do
          if point is in support of iterator.left_child(d) then
               rec(point, d, iterator.left_child(d), result);
          fi
          if point is in support of iterator.right_child(d) then
               rec(point, d, iterator.right_child(d), result);
          fi
     od
end

call GetAffected(point, 0, top_of_grid, {})
```

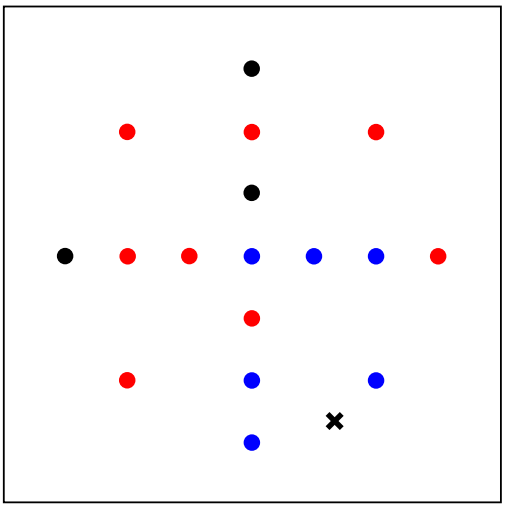


Figure 3: This picture shows the effect of the wider support on the involved basis functions in order to evaluate a arbitrary point of the function. The point is displayed as the black cross. The blue points depict the involved basis functions of a sparse grid using the normal hat basis, where as the red point indicate the additional basis function required for the sparse grid.

function per subspace is involved. All other basis functions have no influence on the evaluation of the function. In contrast to this, prewavelets possess a wider support, thus the neighboring basis functions have to be evaluated as well. Figure 3 depicts the additional involved basis functions compared to a sparse grid using the hat basis. Therefore, the algorithm starts at the top point of the grid and recursively steps down to all hierarchical children whose support is touched by x and calls the function for its neighbors as well. Listing 1 shows the implementation of such an algorithm.

4.2 Hierarchization and Dehierarchization

The algorithm for the hierarchization and dehierarchization is based on the one-dimensional version which is consecutively called for all one-dimensional subspaces of the entire grid. Thus, we only discuss the algorithms for the one-dimensional case with the grid G_n^1 (1D-Grid with level n).

There is no efficient way to directly hierarchize into the prewavelet basis but an efficient way to transform the hierarchical hat basis into the hierarchical prewavelet basis. Thus, the general approach for the hierarchization is:

1. Perform a hierarchization into the normal hat basis with the existing algorithms. These coefficients are denoted as $h_{l,i}$.
2. Transform these coefficients into prewavelet coefficients $u_{l,i}$.

The dehierarchization is performed in the reverse order. As the algorithms for the hierarchization and dehierarchization are already integrated into `SG++`, we focus only on the transformation between the hat basis and the prewavelet basis. We start with the transformation from the prewavelets to the hat basis, as this process is straight forward, and the transformation from the hat basis to the prewavelet basis is the reversion of the first.

4.2.1 Transformation into the Hat Basis

As already mentioned, we denote the hierarchical coefficients of the hat basis as $h_{l,i}$ and of the prewavelet basis as $u_{l,i}$. The transformation parallels the application of a matrix P which can be divided into two matrices $S \cdot Q$:

$$\mathbf{h} = P \cdot Q \cdot \mathbf{u}$$

The matrix Q can be seen as the application of the $\begin{bmatrix} \frac{1}{10} & -\frac{6}{10} & 1 & -\frac{6}{10} & \frac{1}{10} \end{bmatrix}$ star on the prewavelets (respectively $\begin{bmatrix} \frac{9}{10} & -\frac{6}{10} & \frac{1}{10} \end{bmatrix}$ at the border), thus as the decomposition of the prewavelets into their hat components. As already mentioned, the prewavelets

consits of hat basis functions of the generating system, thus the application of the matrix P eliminates the introduced redundancy. For example the matrices for G_2^1 are:

$$\begin{pmatrix} \phi_{1,1} \\ \phi_{2,1} \\ \phi_{2,3} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & -\frac{1}{2} & 0 \\ 0 & 0 & -\frac{1}{2} & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & \frac{9}{10} & \frac{1}{10} \\ 0 & -\frac{6}{10} & -\frac{6}{10} \\ 0 & \frac{1}{10} & \frac{9}{10} \end{pmatrix} \cdot \begin{pmatrix} \psi_{1,1} \\ \psi_{2,1} \\ \psi_{2,3} \end{pmatrix}$$

The generalization of the approach leads to

$$\begin{aligned} h_{l,i} &= u_{l,i} + \frac{1}{10}u_{l,i\pm2} + t_{l+1,2i} - \frac{1}{2}t_{l,i\pm1} \qquad \text{if } (l,i)\text{is an inner point} \\ h_{l,i} &= \frac{9}{10}u_{l,i} + \frac{1}{10}u_{l,i\pm2} + t_{l+1,2i} - \frac{1}{2}t_{l,i\pm1} \qquad \text{if } (l,i)\text{is at the border} \end{aligned}$$

with the temporary values $t_{l,i}$ for all $(l,i) \neq G_n^1$:

$$t_{l,i} = -\frac{6}{10}u_{l,i\pm1} + t_{l+1,2i}.$$

All values for basis functions which are not part of the grid are set to 0. The algorithm starts to calculate all coefficients of the finest level and then proceeds with the next coarser level, after calculating the new temporary values. Figure 4 depicts all involved variables for the calculation of a temporary value and a coefficient.

4.2.2 Transformation into the Prewavelet Basis

The transformation into the prewavelet basis parallels the inversion of the foregoing algorithm. This leads to the following equations:

$$\begin{aligned} \frac{16}{10}u_{l,i} + \frac{4}{10}u_{l,i\pm2} &= h_{l,i} - t_{l+1,2i} + \frac{1}{2}t_{i+1,2(i\pm1)} \\ \frac{12}{10}u_{l,1} + \frac{4}{10}u_{l,3} &= h_{l,1} - t_{l+1,2} + \frac{1}{2}t_{i+1,4} \\ \frac{12}{10}u_{l,2^l-1} + \frac{4}{10}u_{l,2^l-3} &= h_{l,2^l-1} - t_{l+1,2(2^l-1)} + \frac{1}{2}t_{i+1,2(2^l-2)} \end{aligned}$$

Please note, that the coefficients of the prewavelets form a tridiagonal system on each level. For solving these tridiagonal systems, efficient algorithms are known. Figure 5 depicts all involved variables for the calculations.

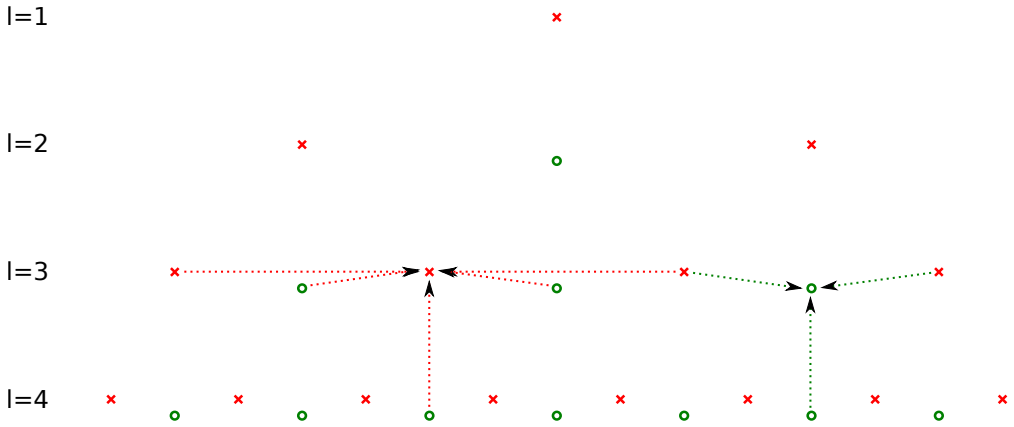


Figure 4: This picture shows all involved grid points (red crosses) and temporary values (green circles) to calculate the new hierarchical coefficients (red arrows) and new temporary values (green arrows). The algorithm works level-wise, starting at the bottom. The calculation of the temporary values (right part of the picture) remains the same as for the hierarchization. In the left part of the picture, the calculation of the new hierarchical surplus $h_{3,3}$ is shown. Here, the red arrows indicate, that for the calculation of $h_{3,3}$ the hierarchical surpluses of the prewavelet basis $u_{3,1}$ and $u_{3,5}$ are needed, which are already known.

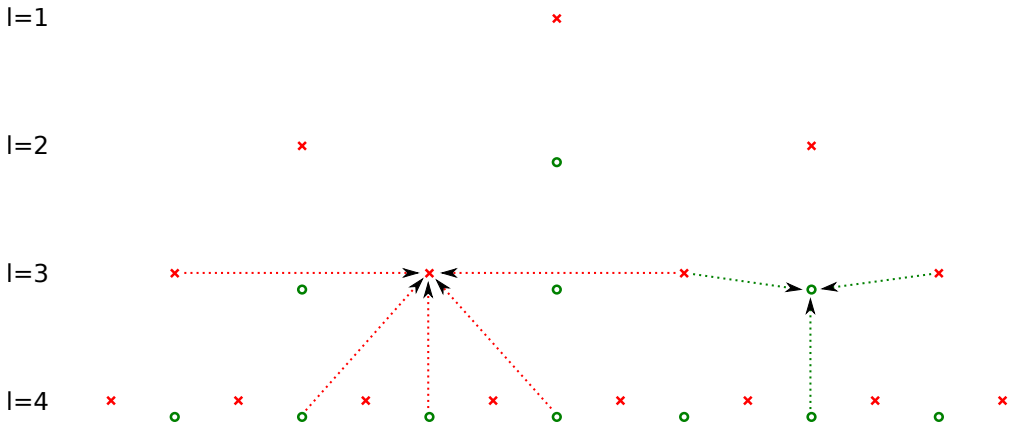


Figure 5: This picture shows all involved grid points (red crosses) and temporary values (green circles) to calculate the new hierarchical coefficients (red arrows) and new temporary values (green arrows) for calculating the hierarchical surpluses. The algorithm works level-wise, starting at the bottom. In the left part of the picture, the new surplus $u_{3,3}$ is calculated. The red arrows from the points $(3,1)$ and $(3,5)$ indicate, that also the not yet known surpluses $u_{3,1}$ and $u_{3,5}$ are needed, which leads to the triangular system of equations. The involved temporary values are already known, as they where calculated in a previous iteration step. On the right side of the picture, the calculation of the temporary value $(3,6)$ is shown.

4.3 Laplace

The framework SG++ provides the user with the possibility to apply the Laplace matrix to a given function. As already mentioned in the introduction, the application of the Laplace operator is a very common task in order to solve PDEs. Furthermore, it is one of the most calculation intensive algorithms in SG++. Especially for this algorithm we expect the benefits of the semi-orthogonal property of the prewavelets.

In order to understand all parts of the algorithm, we briefly explain the application of the Laplace operator. For this algorithm we use the finite elements discretization with the Ritz-Galerkin method, thus the function spaces of the ansatz functions and of the test functions are the same, with the basis functions ϕ_i. The weak formulation of the original problem $\Delta u(x) = f(x)$ is

$$\int_\Omega \nabla u(x) \nabla v(x) dx = \int_\Omega f(x) v(x) dx$$

Now let z be an index, which provides a proper level-wise ordering of the basis functions of our sparse grid G_n with $N = |G_n|$. Furthermore, let $u(x)$ be approximated by $u(x) = \sum_{z=1}^{N} \alpha_z \phi_z$. This leads to

$$\int_\Omega \nabla u(x) \nabla v(x) dx =$$

$$= \sum_{m=1}^{d} \sum_{z=1}^{N} \alpha_z \underbrace{\int_{x_m} \frac{\partial \phi_{z_m}(x_m)}{\partial x_m} \cdot \frac{\partial \phi_{k_m}(x_m)}{\partial x_m} dx_m}_{(1)} \cdot \underbrace{\prod_{s=1; s \neq m}^{d} \int_{x_s} \phi_{z_s}(x_s) \phi_{k_s}(x_s) dx_s}_{(2)} \quad \forall k$$

which can be condensed into the matrix $A := (a_{k,z}) \in \mathbb{R}^{N \times N}$ with $a_{k,z} = \int_\Omega \nabla \phi_k(x) \nabla \phi_z(x) dx$. With that equation it can be seen why this algorithm is split into the gradient-part (1) and the non-gradient part (2). Furthermore, the benefit of the prewavelets can be seen: the calculation of (2) can be eased in comparison to the hat basis, as there the majority of the multiplications are evaluated to 0. On the other hand, part (1) is more complicated than with the normal hat basis. But while part (1) is only required once per matrix entry, part (2) is required $d - 1$ times. Thus, we expect a benefit from this "switch" of the calculation intensive parts.

Due to the unidirectional principle, which was briefly introduced in chapter 2.2, these two parts again are split into an up-part (which considers only the parts above the diagonal, given a proper level-wise order of the ansatz functions) and a down part (which considers the diagonal itself and the elements below the diagonal). Again, for a detailed explanation of the unidirectional principle, please refer to [2]. In SG++, a template for the unidirectional principle already exists. Thus, for the implementation of a new version of the algorithms, four parts have to be implemented:

- gradient-part up and down: The result of these parts parallel the calculation of part (1) of the equation.

- non-gradient-part up and down: This parallels the calculation of part (2) of the equation.

As the non-trivial calling order of these function is already implemented, we concentrate our efforts on the development of the four mentioned parts in the next chapters.

4.3.1 Gradient Part

The gradient part of the Laplace operator unfortunately turns out to be more complex than the normal hat basis counterpart. Due to the wider support, we have to treat up to 4 special cases, namely the first and last basis function as well the second and second last basis function. That leads to a little bit more complicated code, but the algorithm itself is quite straight forward. In the following part, we use the source vector $\mathbf{u} = (u_{l,i})$ and the result vector $\mathbf{r} = (r_{l,i})$. The general principle of these algorithms is to calculate the results of one level by taking the cumulative influence of the level above (down-part) or the level below (up-part) into account. The cumulative influence is stored in temporary variables which are updated after the result of one level is calculated.

Down-Part The down part consists of basically two parts: First, the influence of the four neighboring basis functions and second, the influence of the basis functions above. The influence of the basis functions above is stored in temporary values. Please refer to figure 6, which depicts all involved functions in order to calculate the temporary variables and the actual results. This approach leads to

$$t_{k,,j} = \begin{cases} \frac{1}{h_k}\left(\frac{32}{10}u_{k,j} + \frac{8}{10}u_{k,j\pm 2}\right) & (k,j) \in G_n^1 \\ \frac{1}{h_k}\left(\frac{24}{10}u_{k,j} + \frac{8}{10}u_{k,j\pm 2}\right) & (k,j) \in G_n^1 \text{ and next to border} \\ t_{k-1,\frac{j}{2}} - \frac{1}{h_k}\left(\frac{23}{10}u_{k,j\pm 1} + \frac{1}{10}u_{k,j\pm 3}\right) & (k,j) \notin G_n^1 \\ t_{k-1,\frac{j}{2}} - \frac{1}{h_k}\left(\frac{22}{10}u_{k,j\pm 1} + \frac{1}{10}u_{k,j\pm 3}\right) & (k,j) \notin G_n^1 \text{ and } (k,j\pm 1)\text{next to border} \end{cases}$$

as temporary variables. These are used to calculate the actual result:

$$\begin{aligned} r_{k,j} &= -\frac{6}{10}t_{k-1,\frac{j\pm 1}{2}} \\ &\quad + \frac{1}{h_k}u_{k,j} \cdot \begin{cases} \frac{612}{100} & (k,j) \text{ inner point} \\ \frac{356}{100} & (k,j) \text{ next to border} \end{cases} \\ &\quad + \frac{1}{h_k}u_{k,j\pm 2} \cdot \begin{cases} \frac{256}{100} & (k,j) \text{ and } (k,j\pm 2)\text{inner point} \\ \frac{242}{100} & (k,j) \text{ or } (k,j\pm 2)\text{next to border} \\ \frac{228}{100} & (k,j) \text{ next to border} \end{cases} \\ &\quad + \frac{1}{h_k}u_{k,j\pm 4} \cdot \frac{14}{100} \\ r_{1,1} &= 4u_{1,1} \end{aligned}$$

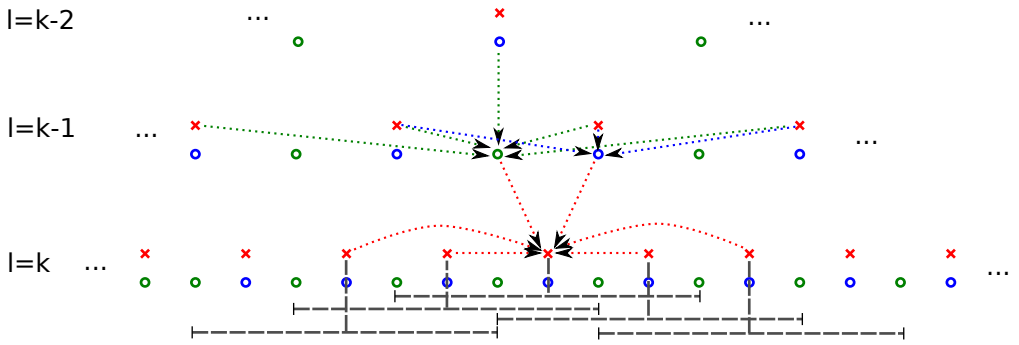


Figure 6: This figure depicts the down part of the application of the Laplace matrix. The algorithm works level-wise from the top to the bottom. In the top part of the picture, all involved values for the calculation of the temp values $t_{k-1,i-1}$ and $t_{k-1,i}$ can be seen. Please note the differences in the calculation of temporary values at grid points (blue circles) and between grid points (green circles). These temporary values are required for the calculation of the result $r_{k,2i-1}$ in the lower part of the picture. In order to calculate $r_{k,2i-1}$, the surpluses of the two left and two right neighbors on the same level are also required. This is due to the wide supports, which are indicated for the prewavelet $(k, 2i-1)$ and its two left and right neighbors with the dashed gray lines.

All non grid points at the borders disappear.

Up-Part The Up-Part of the algorithm turns out to be less complicated. This is because only the influence of the level directly below must be taken into account. All other levels can be accumulated into the temporary variables, as for the calculation of the influence on the gradient of a linear function just the start and endpoints have to be evaluated. All peaks within this area vanish.

$$t_{k,j} = -\frac{6}{10} u_{k,j\pm1} + t_{k+1,2j} \qquad (k,j) \notin G_n^1$$

Calculation of the inner points within the grid:

$$\begin{aligned} r_{k,j} \quad = \quad & \frac{1}{h_k}\left(2t_{k+1,2j} - t_{k+1,2(j\pm1)}\right) \\ & -\frac{6}{10}\frac{1}{h_k}\left(-t_{k+1,2(j\pm2)} + 2t_{k+1,2(j\pm1)} - 2t_{k+1,2j}\right) \\ & +\frac{1}{10}\frac{1}{h_k}\left(-t_{k+1,2(j\pm1)} + 2t_{k+1,2(j\pm2)} - t_{k+1,2(j\pm2)}\right) \end{aligned}$$

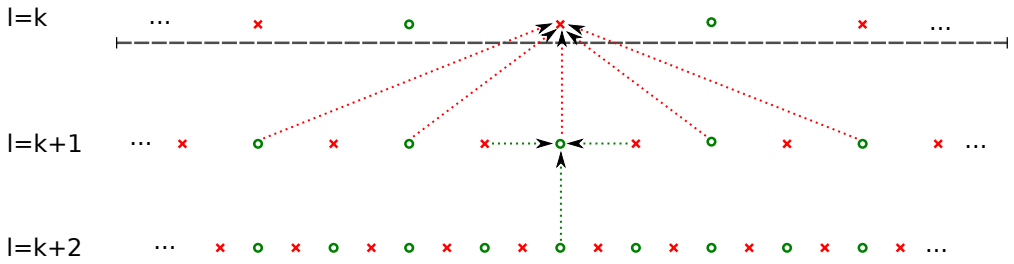


Figure 7: This figure depicts the up part of the application of the Laplace matrix. The algorithm works level-wise from the bottom to the top. In the lower part, the calculation of one temporary point (green circles) is displayed, in the upper part of one result point. Furthermore, all required temporary points for the calculation of the result are indicated. It can be seen that these temporary point are within the support of the result point, which is displayed as a gray dashed line.

Calculation of the border points of the grid:

$$\begin{aligned} r_{k,j} \quad = \quad & \frac{9}{10}\frac{1}{h_k}\left(2t_{k+1,2j} - t_{k+1,2(j\pm1)}\right) \\ & -\frac{6}{10}\frac{1}{h_k}\left(-t_{k+1,2(j\pm2)} + 2t_{k+1,2(j\pm1)} - t_{k+1,2j}\right) \\ & +\frac{1}{10}\frac{1}{h_k}\left(-t_{k+1,2(j\pm1)} + 2t_{k+1,2(j\pm2)} - t_{k+1,2(j\pm2)}\right) \end{aligned}$$

All non grid points at the borders disappear. Please refer to the figure 7, which depicts all involved functions in order to calculate the temporary variables and the actual results.

4.3.2 Non-Gradient Part

The vast majority of calculations during the application of the Laplace matrix is done here. In this part of the algorithm, the prewavelets have the advantage of being semi-

orthogonal, thus most entries of the matrix disappear. The following entries have to be calculated:

$$\begin{aligned}
m_{(1,1)(1,1)} &= \frac{1}{3} \\
m_{(2,1)(2,3)} &= m_{(2,3)(2,1)} = \frac{1}{25} \\
m_{(l,1)(l,1)} &= m_{(l,2^l-1)(l,2^l-1)} = \frac{44}{75} h_l \\
m_{(l,1)(l,3)} &= m_{(l,2^l-1)(l,2^l-3)} = \frac{11}{75} h_l \\
m_{(l,i)(l,i)} &= \frac{18}{25} h_l \\
m_{(l,i)(l,i\pm2)} &= \frac{2}{15} h_l \\
m_{(l,i)(l,i\pm4)} &= -\frac{11}{75} h_l
\end{aligned}$$

Please note that we again just look at the one-dimensional case. The entry $m_{(l_1,i_1)(l_2,i_2)}$ refers to the row of point (l_1, i_1) and column of the point (l_2, i_2).

5 Adaptive Sparse Grids

With adaptive sparse grids, we encounter some problems, mainly due to the wide support of the prewavelets. The normal adaptive sparse grid ensures that all hierarchical parents of a refined grid point are also in the grid, thus the data transport between all necessary grid points is possible. For the algorithms presented here, this is enough for adaptive sparse grids with prewavelets with the exception of the algorithms using the up-down mechanisms. For algorithms utilizing the unidirectional principle, several required points are missing in the grid. Figure 8 depicts the problems which arise by the usage of a wider support.

In order to solve this issue, the missing grid points are added to the grid and set to zero. Afterwards, the algorithm performs its calculation and produces a result vector. In this result vector, the results of the additionally added points are set back to zero, as the point should not influence the overall result (they are "not officially" in the grid). The influence of all other missing grid points is simply set to zero.

In a last remark, we have to discuss which points have to be added to the grid additionally. A newly added grid point (l, i) also needs his two left and two right neighbors, as the support of the grid point touches them. Thus, the points $(l, i \pm 2)$ and $(l, i \pm 4)$ in every dimension have to be added as well. Furthermore, also the diagonal neighbors have to be added. As a necessary presumption, all hierarchical parents of these grid points have to be added as well. Fortunately, they are already in the grid. The reason is as follows:

1. The father of (l, i) in every dimension is in the the grid.

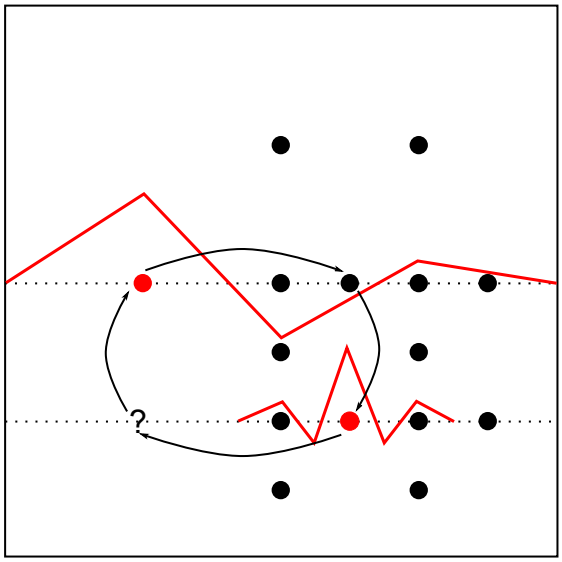


Figure 8: This figure depicts the problems with the data transport using the up-down algorithm. Here, we look at two grid points (red) and the possible ways of information transport. Furthermore, the width of these two prewavelet basis function is indicated with the red lines. It can be seen, that these two basis functions do have an overlapping support, but in the lower-left corner, one necessary grid point for data transport (indicated with '?') is missing. Please note: in a regular sparse grid, the point illustrated with a '?' would be in the grid.

2. The father of the two left grid points $(l, i-2)$ and $(l, i-4)$ is the left father of (l, i). But the neighbors of this father were added when the father itself was added to the grid. This argument holds true for the right grid points as well.

But not only the left and right neighbors in each dimension have to be added, the prewavelet basis functions require also the direct neighbors on the diagonals. This leads to even bigger grids compared to the normal adaptive sparse grid.

Caution Even with these additional points, we encountered that in rare cases the hierarchization does not work properly. Unfortunately, this issue is not yet resolved.

6 Implementation

The implementation of the algorithms for the hierarchization and dehierarchization in `SG++` followed the descriptions of the algorithms as `SG++` already provides an algorithm template (`sweep1D`) to apply an operation on all one-dimensional subspaces of the underlying grid. Thus, only the one-dimensional algorithms had to be implemented as described in the chapters above.

Several modifications had to be made in order to support the Laplace operator with prewavelets. As already mentioned, in order to support adaptive sparse grids, several transport grid points had to be added. To implement this requirement, several possibilities exist. But as we were extending an existing framework, we needed an approach which does not have any influence on already existing parts of the framework. Thus, we decided to give the `PrewaveletGrid` class a second storage, the shadow storage, in order to hold all additional grid points which are only needed for the calculation of the Laplace operator. After each refinement, all neighbors of the newly added points needs to be added into the shadow storage. We have to be careful if a point is added to the grid which is already in the shadow storage. If that happens, the shadow storage has to be modified in order to only hold grid points which are not in the actual grid. This ensures, that every point is only once either in the normal storage or in the shadow storage.

When applying the Laplace operator, all grid points of the shadow storage are added to the actual storage, the alpha and result vectors are expanded accordingly. After the calculation of the Laplace operation, all grid points from the shadow register are deleted from the actual grid. This is possible, as new grid points are always added to the end of the storage, so there are no "free" points in the vector. The result vector gets shrunk to the right size as well. This enables the usage of a prewavelet grid without altering the already common usage of the `SG++` framework.

7 Application

7.1 Stationary Heat Equation

We are particularly interested in the convergence speed and the runtime of the conjugated gradients method (CG) for the Laplace matrix, as the calculation of this matrix is the most complicated and time consuming one of the operations considered. And we expect exactly by the application of this operation a benefit of the usage of prewavelets. Thus, we decided to solve a problem which only consists of the Laplace matrix, in order to observe the direct influence of the matrix: the stationary heat equation. The heat equation itself is defined by

$$\begin{aligned} \kappa \left(\sum_{i=1}^{d} \frac{\partial^2 T}{\partial x_i^2} \right) &= \frac{\partial T}{\partial t}. \\ \kappa \triangle T &= T_t \end{aligned}$$

In this equation, κ denotes the material's conductivity and T denotes the temperature. As we are interested in the convergence speed of the application of the Laplace matrix and not in the actual temperature distribution within a body of a specific material, we set to $\kappa = 1$. The stationary result is reached, when the temperature in the space is constant, thus the first derivative with respect to the time t is 0. This leads to

$$\triangle T = 0.$$

With the classical finite element discretization, the approximation of T with $\sum \phi_i \alpha_i$ with a coefficient vector α leads to a system of linear equations,

$$C\alpha = 0$$

where C denotes the Laplace matrix.

It is obvious, that the target function is 0 and thus the equation is fulfilled with $\alpha_i = 0$. In order to investigate the convergence behavior of the Laplace matrix, we initialize the coefficients by approximating various functions, like the sinus or some linear functions for example. After each step of the conjugate gradient solver, we can observe the convergence of the initial function to 0. As a measure for the current error, we take the L_2-norm of the error. As the target function happens to be 0, the L_2-norm of the error is the L_2-norm of the current function itself and can be calculated as follows:

$$\|u\| = \|\sum \alpha_i \phi_i\| = \sum \alpha_i \|\phi_i\| = \sum \alpha_i \sqrt{(2/3)^d 2^{-|\mathbf{l}|_1}}$$

7.2 Results

7.2.1 Runtime Behavior

Before we investigate the convergence speed of the conjugated gradients, we analyze the runtime of the application of the Laplace matrix. Therefore we measure the average runtime of the application for a given grid. We especially expect the runtime of the prewavelet version to outperform the hat basis version in higher dimensions, as there the advantage of the eased calculation of the Laplace normal part of the algorithm is the most called function.

Unfortunately, it turns out that the prewavelets lack in performance behind the hat basis version, even for increasing dimensionality. The results of the tests are shown in table 1 and in figure 9. Here striking is that the relative speed penalty for the usage of the prewavelets (called "Ratio" in the figure and table) seems not to get better with increasing dimensionality. In a secend step, we fix the dimensionality and vary the level of the grid considered. Here, the behavior is basically the same, even though, the ratio seems to get better with increasing size of the grid. These results are astonishing, especially that the ratio does not seem to get influenced by the grid size, or at least not in the expected amount.

Additionally, not only the runtime of one application of the Laplace matrix is worse in comparison to the normal hat basis, but also the convergence speed of the conjugated gradients method is lower. This point is covered in the next section.

7.2.2 Convergence Speed

During the runtime test we observed differences in the convergence speed of the conjugated gradients method. Normally, the condition number (ratio of the absolute biggest

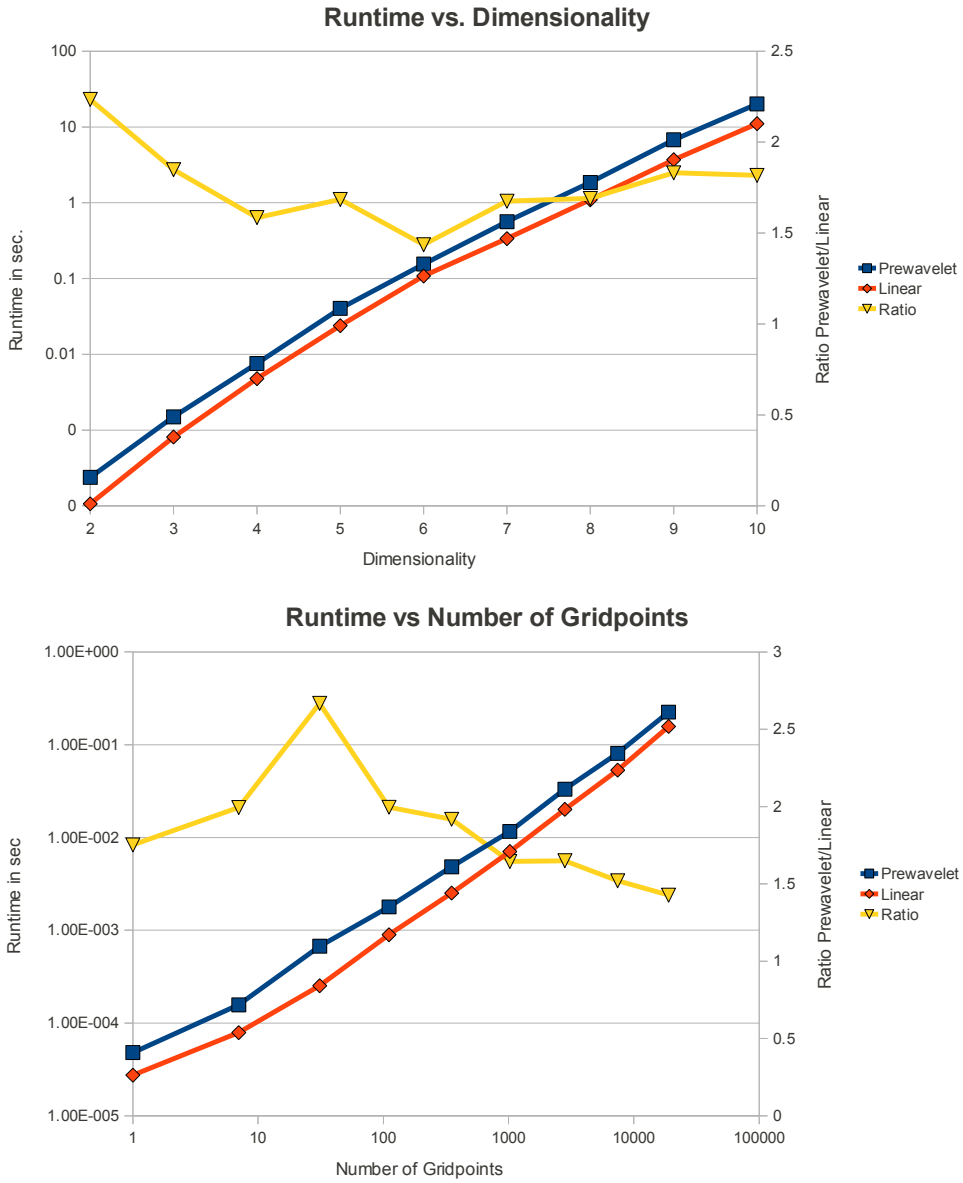


Figure 9: These two charts display the runtime behavior for the calculation of the Laplace matrix. The runtime in both charts is the average runtime of one application of Laplace matrix and displayed on the left y-axis. Please note the logarithmic scale of this axis. On the right y-axis, the ratio of the runtime with prewavelets and with the normal hat basis is displayed.
The upper image uses a grid of level 4, and plots the runtime in dependency of different dimensionalities. The lower chart uses the fixed dimensionality 3, and plots the runtime against altering levels ranging from 1 to 9. Please note the logarithmic scale of the x-axis in the lower chart. For the exact results, please refer to table 1.

Runtime vs. Dimensionality

Dim	Level	Size	Prewavelet	Linear	Ratio
2	4	49	$2.38{\cdot}10^{-4}$	$1.06 \cdot 10^{-4}$	2.23
3	4	111	$1.50{\cdot}10^{-3}$	$8.10 \cdot 10^{-4}$	1.85
4	4	209	$7.58{\cdot}10^{-3}$	$4.78 \cdot 10^{-3}$	1.58
5	4	351	$4.03{\cdot}10^{-2}$	$2.39 \cdot 10^{-2}$	1.69
6	4	545	0.155	0.108	1.44
7	4	799	0.564	0.336	1.68
8	4	1121	1.85	1.10	1.69
9	4	1519	6.78	3.70	1.83
10	4	2001	20.1	11.1	1.82

Runtime vs. Number of Gridpoints

Dim	Level	Size	Prewavelet	Linear	Ratio
3	1	1	$4.81{\cdot}10^{-5}$	$2.74{\cdot}10^{-5}$	1.75
3	2	7	$1.57{\cdot}10^{-4}$	$7.88{\cdot}10^{-5}$	1.99
3	3	31	$6.73{\cdot}10^{-4}$	$2.52{\cdot}10^{-4}$	2.67
3	4	111	$1.78{\cdot}10^{-3}$	$8.94{\cdot}10^{-4}$	2.00
3	5	351	$4.83{\cdot}10^{-3}$	$2.52{\cdot}10^{-3}$	1.99
3	6	1023	$1.16{\cdot}10^{-2}$	$7.06{\cdot}10^{-3}$	1.65
3	7	2815	$3.32{\cdot}10^{-2}$	$2.01{\cdot}10^{-2}$	1.65
3	8	7423	$8.08{\cdot}10^{-2}$	$5.32{\cdot}10^{-2}$	1.52
3	9	18943	0.225	0.158	1.43

Table 1: Comparison of the different runtimes. The column "Size" is the number of grid points of the given grid. The measured times are in seconds and given in the columns "Prewavelet" for the grid with prewavelet ansatz functions and in "Linear" for the grid with the normal hat basis. The column ratio displays the ratio of the runtime with prewavelets and with the normal hat basis.
In the upper table, the level is fixed with a varying dimensionality, whereas in the lower table, the dimensionality is fixed to three. For a visualization of these results, please refer to figure 9.

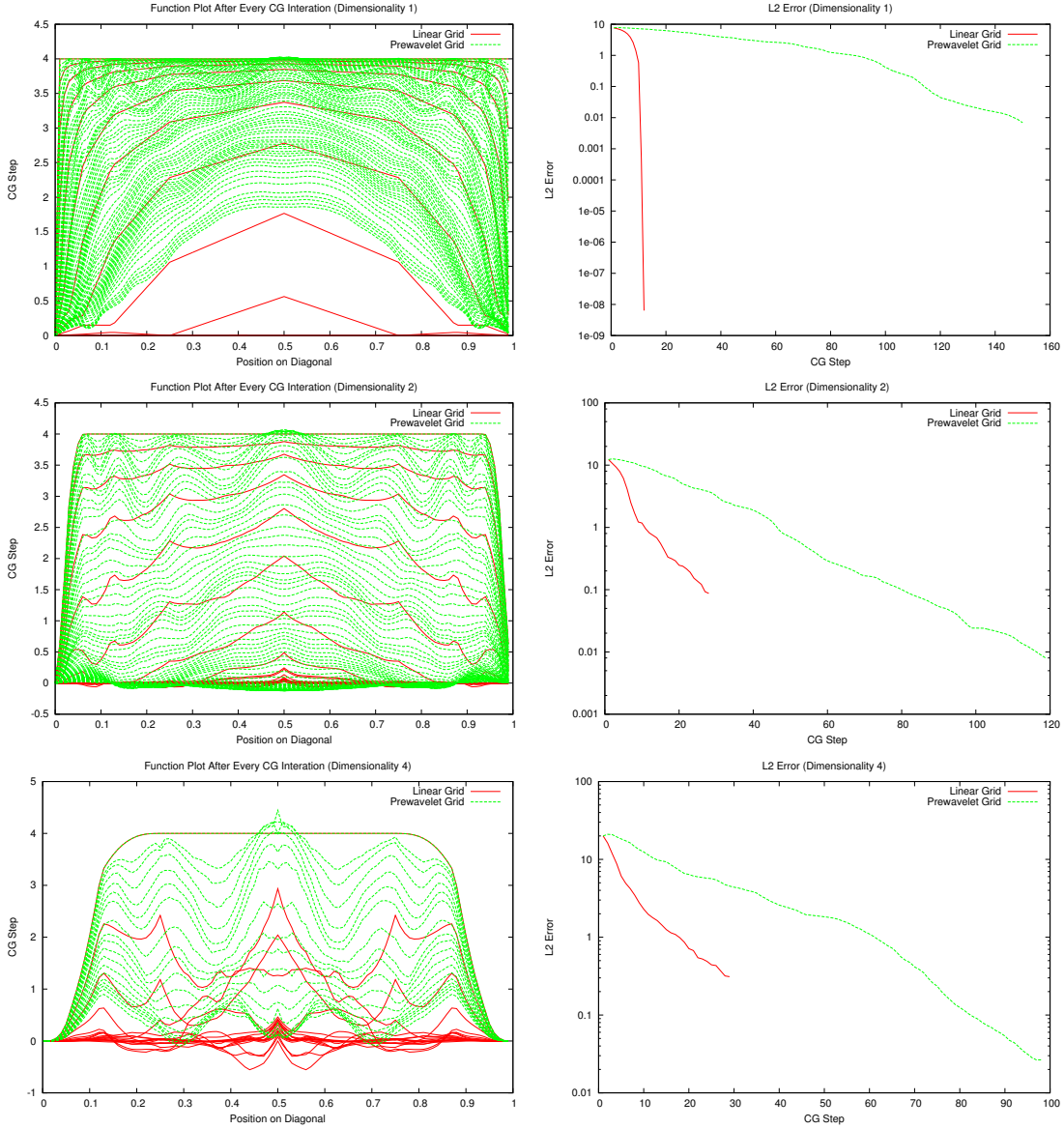


Figure 10: These figures depict the convergence speed for solving the stationary heat equation for different dimensionality. The initial function is simply $f(x) = 4$ and should converge to $f(x) = 0$. On the left part of this figure, the current solutions are plotted after every CG iteration. The green functions indicate the current solution of the grid using the prewavelets, the red functions using the linear hat ansatz function. It can be seen, that the prewavelet version converges far slower to the solution $f(x) = 0$ than the linear version. On the right, the L_2 error is plotted (on a logarithmic scale) against the number of performed CG iterations. It is obvious, that the error with the prewavelet function (green) reduces much slower than the linear version (red).

			Prewavelet		**Linear**	
Dim	Level	Size	Plain	Preconditioned	Plain	Preconditioned
1	10	1023	5740.31	151.28	512.00	1.00
2	8	1793	910.33	102.94	1632.00	385.60
3	6	1023	182.45	77.68	2055.58	2011.36
4	6	2561	499.38	78.58	24049	17952

Table 2: Comparison of the different condition numbers. The column "plain" means that the matrix is used as-it-is and in the column "preconditioned" a matrix with a diagonal preconditioner is employed. The column size is the number of n grid points of the given grid i.e., the matrices have the size $n \times n$. In the one-dimensional case, the grid with linear hat ansatz functions (in fact, the matrix is only a diagonal matrix) clearly outperforms the grid with prewavelets, but for increasing dimensionality, the prewavelet grid is superior to the linear grid in terms of condition number.

eigenvalue and the absolute smallest eigenvalue of the matrix) of the matrix gives a hint of the convergence speed of those solvers like the conjugated gradients: The lower the condition number the better the convergence.

The condition numbers of the matrices for different dimensions can be found in Table 2. For one dimension, the condition number of the matrix with normal hat ansatz functions outperforms the version with prewavelets. That is quite obvious as the matrix with the normal hat ansatz functions is only a diagonal matrix in the one-dimensional case. But with increasing dimensionality, the prewavelet version is superior to the linear version. Furthermore, the simple diagonal preconditioner is far more effective on the prewavelet version than on the linear variant.

Again, this is a surprising result, as the condition numbers would suggest that the prewavelet version shows a better convergence behavior. Thus, one would expect the linear version to be better (in terms of convergence) than the prewavelet version in the one-dimensional case and then getting outperformed with increasing dimensionality. But the linear version constantly beats the prewavelet version as depicted in figure 10. The initial assignment of the α-vector is a constant function with the value 4. In order to produce meaningful figures, the current function was evaluated along the diagonal through the function space after each iteration step of the CG algorithm. We also ran several test with different initial assignments of α (i.e. combinations of sine and cosine combination) which all produced similar results.

Together, with the usage of the prewavelet basis function, a single calculation of the Laplace matrix is more time consuming than with the linear result, and the worse convergence speed requires more of those applications.

8 Conclusion

After running several tests, it turned out that the expected gain in speed and convergence behavior cannot be observed, eventhough the condition numbers would suggest a better convergence performance.

We identified three reasons for weak runtime performance with the usage of prewavelets:

1. The additional effort due to the wider support overweights the advantages of the semi-orthogonality. Due to the wider support, calculations like the evaluation require more steps than the standard version with the hat basis.
2. The implementation of the normal hat functions has already undergone several iterations in development in which the performance was constantly improved. This is not true for the prewavelets and one can expect performance gains after further reviewing the code.
3. The framework was initially not designed to support basis functions with wider supports, which required solutions like the shadow storage, which all together lead to a poor performance in comparison to the normal hat basis.

In addition, the adaptive sparse grids with prewavelet basis functions contain significantly more grid points than an adaptive sparse grid with the normal hat basis. Until level three, the adaptive sparse grid with prewavelets is as dense as the regular sparse grid.

At the end of the day, the utilization of prewavelets led to no gain in speed or precession but to several drawbacks, especially in terms of speed, which makes this method not applicable in our point of view.

References

[1] C. Bourgeois and S. Nicaise. Prewavelet analysis of the heat equation. *Numerische Mathematik*, 87(3):407–434, 2001.

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

[3] C. Feuersänger. Dünngitterverfahren für hochdimensionale elliptische partielle Differentialgleichungen. *Master's thesis, Institut für Numerische Simulation, Universität Bonn*, 2005.

[4] M.S. Floater and E.G. Quak. A semi-prewavelet approach to piecewise linear prewavelets on triangulations. *Approximation Theory IX*, 2:60–73, 1998.

[5] M.S. Floater and E.G. Quak. Piecewise linear prewavelets on arbitrary triangulations. *Numerische Mathematik*, 82(2):221–252, 1999.

[6] M.S. Floater and E.G. Quak. Linear independence and stability of piecewise linear prewavelets on arbitrary triangulations. *SIAM Journal on Numerical Analysis*, 38(1):58–79, 2001.

[7] U. Kotyczka and P. Oswald. Piecewise linear prewavelets of small support. *Series in approximations and decompositions*, 6:235–242, 1995.

[8] A Niedermeier. Lösung von Differentialgleichungen durch Prewavelets. *Diploma Thesis, Zentrum Mathematik, TU München*, 1998.

[9] D. Pflüger. Data mining mit dünnen gittern. *Diploma Thesis, Universität Stuttgart*, 2005.