Space Filling Curves and Hierarchical Basis

Klaus Speer

Abstract

Real world phenomena can be best described using differential equations. After linearisation we have to deal with huge linear systems of equations. Traditional direct solvers cannot be used to find the correct solution. Even common iterative solvers would need an unacceptable amount of time to solve the system. In order to speed up the convergence of solvers we need to invent new techniques, a possible one is the usage of hierarchical basis. For the overall computer time not only the computational costs in terms of numerical operations are of interest. Moreover data transfer to and from the processor to the corresponding storage cannot be neglected. The following text will shortly introduce the ideas of space filling curves to exploit the advantages of computer caches more effectively and of hierarchical basis to speed up the numerical simulation itself. At the end of the section about hierarchical basis we will have a short outlook to generating systems and their advantages with higher dimensional problems.

Contents

1 Space Filling Curves

We can construct a mapping from a one dimensional interval to a finite higher dimensional interval. If this mapping passes through every point of the target space the mapping is called a "space filling curve".

$$I = [0,1] \rightarrow \Omega = [0,1]^d$$
$$d \in (2,3,...)$$

1.1 Peano's space filling curve

The Italian mathematician Giuseppe Peano (1858 - 1932) was the first one who constructed (only algebraically) such a space filling curve in 1890. His idea was to split the unit interval and the target interval into 3^n smaller intervals, where n defines the dimension of Ω , and to define a mapping between these. He defined any $t \in I = [0,1]$ using a ternary system. In a ternary system t_i can take a value out of (0,1,2).

$$t = (0 \cdot t_1 t_2 t_3)$$
$$t = \sum_{i=1}^{n} t_i \cdot (3)^{-i}$$

example:

$$t = (0_3 \cdot 1201)$$
$$t = \frac{1}{3^1} + \frac{2}{3^2} + \frac{0}{3^3} + \frac{1}{3^4}$$

When using an infinite ternary one can represent any number in the unit interval I. Peano now created a mapping from the unit interval (represented in a ternary system) to a higher dimensional space (also represented in a ternary system) based on the operator k.

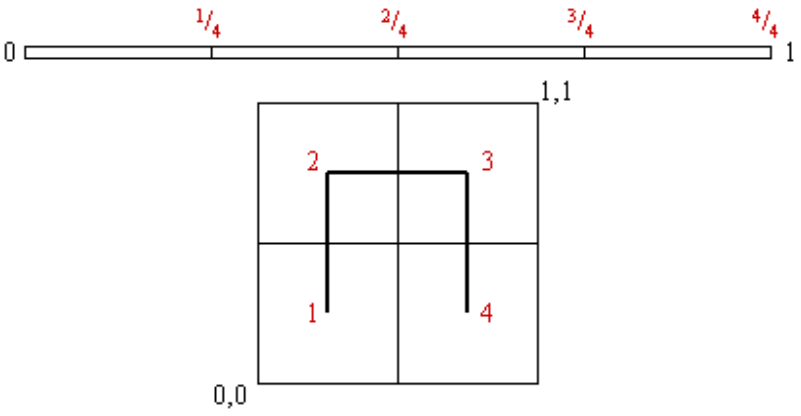
$$k(t_i) = 2 - t_i$$

k^i denotes the i^{th}-iterate of the operator k. For the two dimensional target space the mapping according to Peano looks like the following:

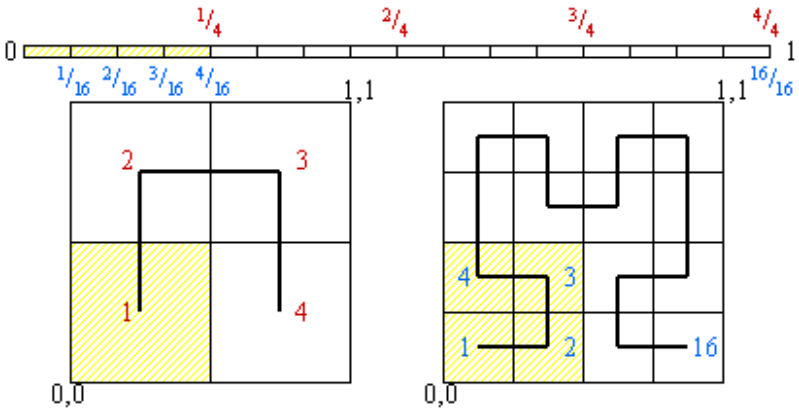
$$p : I \rightarrow \Omega : [0,1]^2$$
$$p(t) = \begin{pmatrix} 0_3 \cdot t_1 k^{t_2}(t_3) k^{t_2+t_4}(t_5)... \\ 0_3 \cdot k^{t_1}(t_2) k^{t_1+t_3}(t_4)... \end{pmatrix}$$

1.2 Hilbert's space filling curve

The German mathematician David Hilbert (1862 - 1943) was the first one to give the so far only algebraically described space filling curves a geometric interpretation. Like Peano he splat I and Ω into the same number of subintervals and defined a mapping between these intervals. (Unlike Peano he didn't use the number 3 as basis to refine/split the starting interval, but he used 2 instead.) According to him we first split the intervals I and Ω into 2^d (d = dimension of Ω) sub intervals and define a mapping between these intervals.



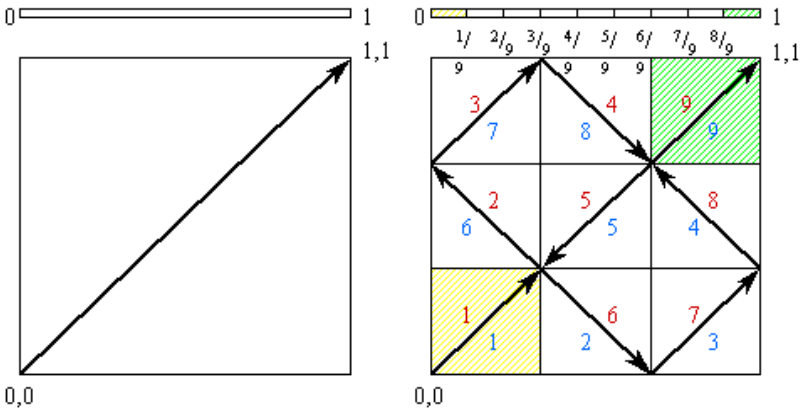
Above picture shows the splitting of both I and Ω into four intervals (red marked) and a possible way to orientate the sub areas in Ω. When moving through I we follow the curve in Ω. Now we can recursively split the sub areas further. To do so we pick out the first sub interval of $I : I_1 = [0, 1/4]$. It has been mapped to the first area of $\Omega : \Omega_1 = [0, 1/2] \times [0, 1/2]$. We subdivide both I_1 and Ω_1 again into 4 subintervals (blue marked).



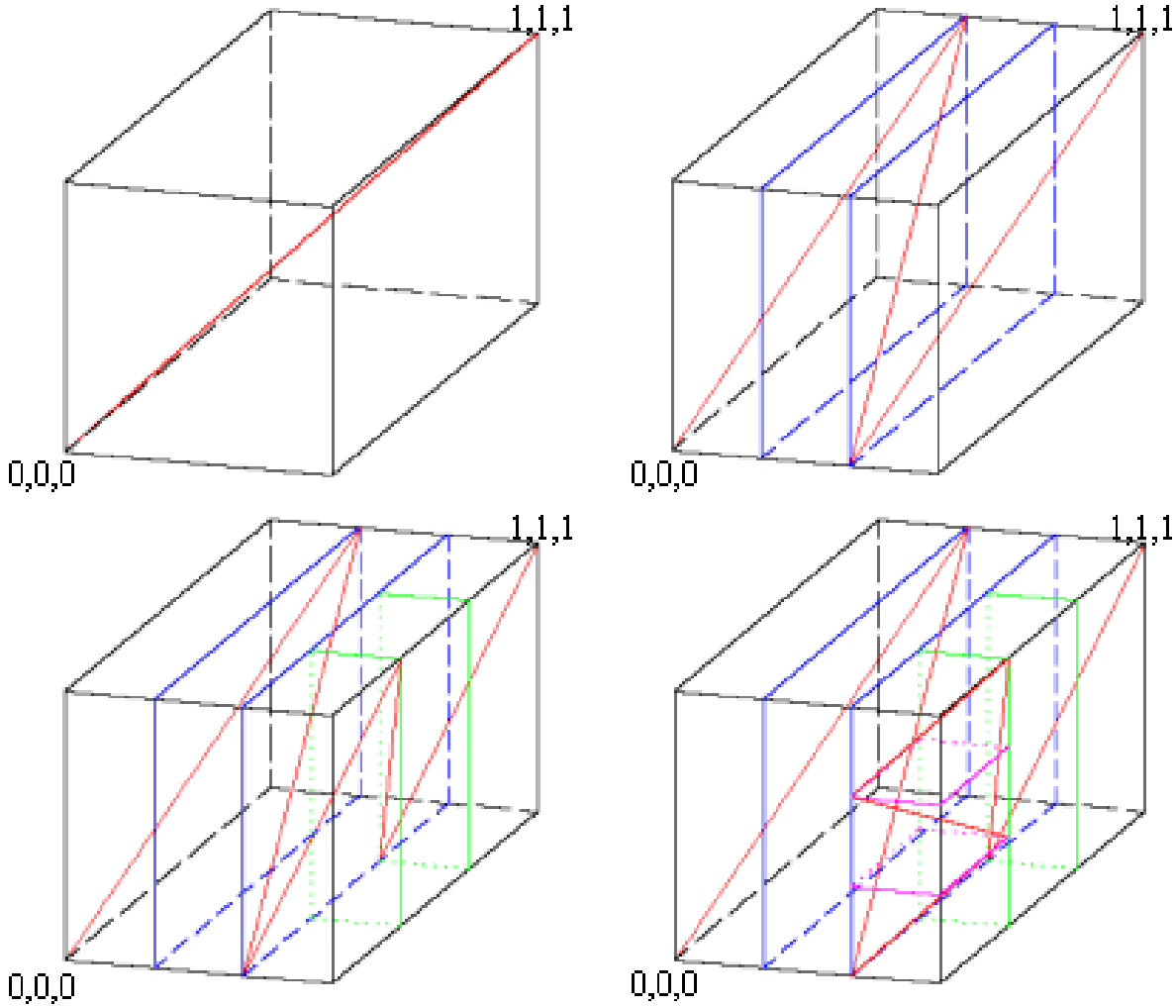
The emerging new sub intervals lie within the old intervals, e.g. the interval $I_1 = [0, 1/16]$ is mapped to the interval $\Omega_1 = [0, 1/4] \times [0, 1/4]$, $I_2 = [1/16, 2/16]$ is mapped to $\Omega_2 = [1/4, 2/4] \times [0, 1/4]$ and so on. This important fact is called full inclusion. So we can refine the curve anywhere we want and still get a continous mapping. (Useful for only locally refined grids.)

1.3 Geometric interpretation of Peano's space filling curve

Peano's space filling curve got a geometric interpretation a while after Hilbert described his space filling curve. Starting points are the unit inverals I and Ω. The Peano curve we want to examine always passes from (0,0,...) to (1,1,...). Symmetric splitting in all dimensions into 3 sub intervals results for Ω^2 in below illustration. The way we run through Ω can be freely chosen. Two possibilities are marked by red and blue numbers in the right sketch. Only the start and end points are fixed. As for Hilbert's curve we can also state full inclusion for the Peano curve.



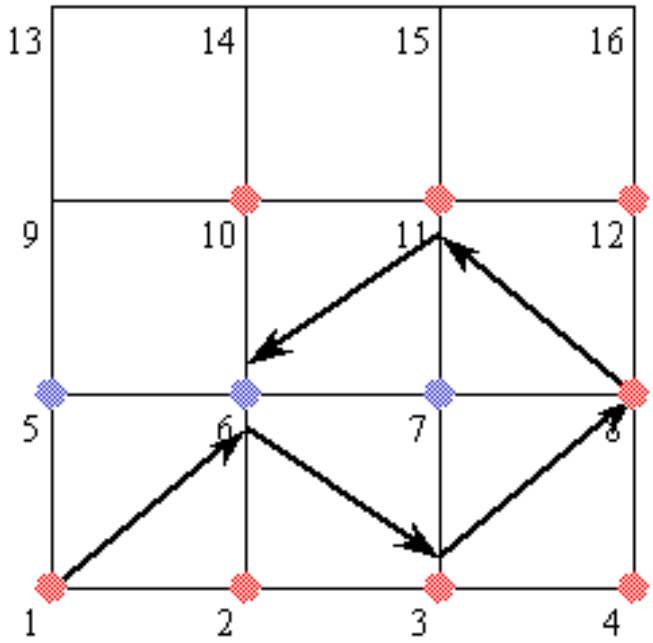
For the Peano curve we will also examine the three dimensional case. We construct it step by step. First we partition the unit cube into 3 equally sized cuboids by splitting along the x-axis (blue), then refine one of these cuboids along the y-axis (green) and finally refine along the z-axis (magenta). If we would have performed this splitting symmetrically for all emerging sub areas we would have ended with 27 ($= 3^3$) cubes, but it's not necessary to do so.



1.4 Usage of space filling curves in numerical simulations

Numerical simulations deal with computations on grids with finite resolution. The order in which we perform the computations on the different cells is up to us. We only have to work on all cells. To fulfil the demands of efficient data transfer the order should reduce the necessary load operations to a minimum and even this transfer operations should be structured in a way standard computer optimisation techniques can exploit (pre-fetching). Organising data in matrix/array structures leads to expensive cache misses. Space filling curves on the other hand satisfy all the above mentioned demands and are therefore an interesting method in numerical simulations.

To show the value of space filling curves we will start with a simple two dimensional grid. Below illustration shows the run through the unit square using a Peano curve. When working on the first cell we need the data points (1,2,5,6), in the second cell we need (2,3,6,7) and so on. Let's look on the line given by the blue points: these values are first processed in increasing order (5,6,7) and then in decreasing order (7,6,5). For space filling curves like Hilbert's and Peano's curves we will find inner point lines which are always processed in reversing order! If we have a suitable data mechanism with which we can first build up a data queue and then retrieve the data in the opposing order we could automatise the data transfer and reduce cache misses.

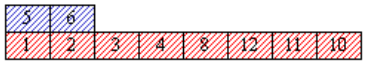


Writing to and from such a staple leads to stacks. Stack is a data structure on which only two operations are allowed:

- Push:
  write data to the stack, this information will lie on top of the staple
- Pop:
  read data back from the stack, one can only access the very upper element of the stack

As we have seen in above example it's sufficient to only access the most upper element for our demands. We can read the elements one after the other from the stack. For the simple example 2 stacks (blue and red) are enough to store all the necessary data. After processing the second square the stacks would look like:

When following the Peano curve to the fifth element the red stack grows step by step, we always push new information to the stack. But after the fifth square the blue stack has been reduced to the elements (5,6) as the element 7 has been read off (and won't be used any longer). The red stack is that big because of the boundary points (1,2,3,4,8,12) which aren't processed backward (boundary points don't lie on inner lines!).

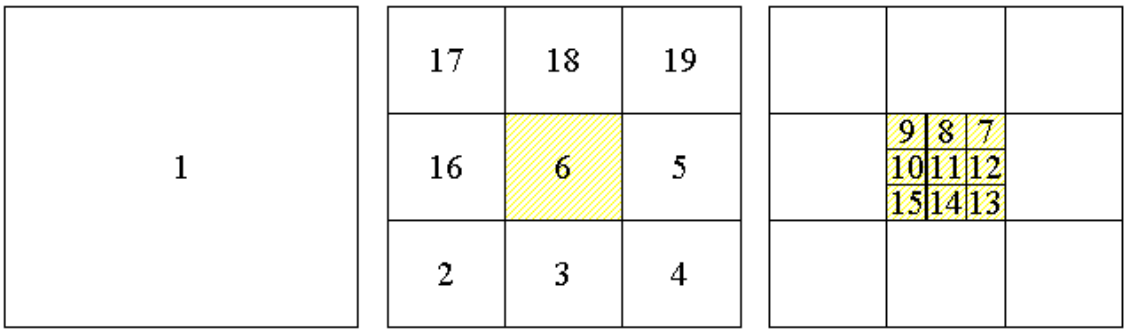


At this point we can already state some necessary properties of space filling curves to allow their usage within numerical simulations.

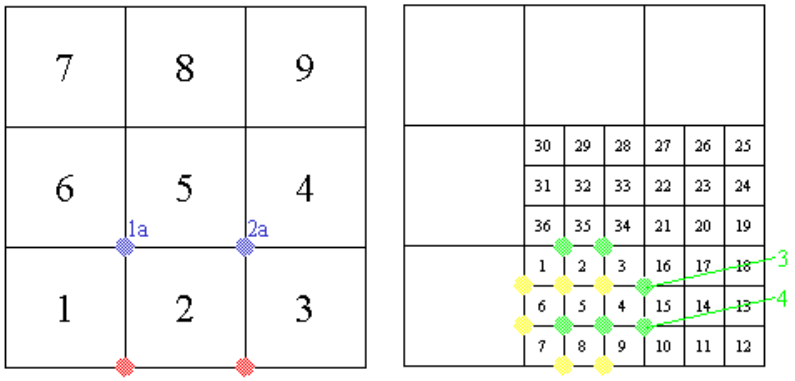
- The processing order has to be inverted when we switch from one side of the domain to the other side. For the two dimensional case this can be fulfilled easily but from dimension three on this has to hold for inner planes, too.
- Later on we want to locally refine our grid at interesting points. This refinement mustn't change the order of existing points to guarantee consistence. New points should only be inserted between existing (coarser) points.

So far only Peano's curve seems to meet above criteria independent of the examined dimension. Hilbert's space filling curve works well for two dimensional problems (with less costs than Peano's curve due to only splitting in 4 sub squares instead of 9), but doesn't show the needed behaviour in higher dimensions.

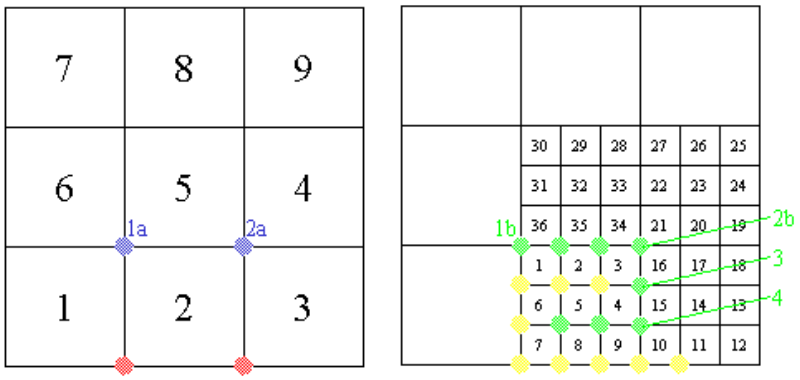
Now we want to extend our method to adaptively refined grids and hierarchical data. We will explain this idea by using below illustration. We first process data on the coarsest level (cell 1), then proceed with the finer levels (2,...) and wherever we split further, we first work on the coarse cell (6), then finish completely the finer level(s) for this cell (cells 7 to 15) and finally come back to the coarser level and finish it (cells 16 to 19). This approach is called top-down depth-first approach.



We will first examine the case when using hierarchical basis. Points 1a and 2a lie on the coarse level, points 3 and 4 exist on the finer level. If we would use two stacks like for nodal basis (above described situation with two stacks: blue and red), the blue and green points would be stored in one stack and also the red and yellow points. If we now perform our top-down depth-first algorithm after computing cell 9 (fine level) point 4 would be the upper most element in the corresponding stack. Next coarse cell 3 is to be computed, therefore we need point 2a, which is blocked by points 3 and 4. This example already shows that two stacks aren't enough any longer. Instead we introduce 2 new stacks. So we store the values of the coarser level in so-called point stacks (blue, red: 0D-stacks) and the values of the finer level in so-called line stacks (green, yellow: 1D-stacks).



Now we examine the usage of a generating system. Here points can hold more values (on different levels: 1a, 2a on coarse level, 1b and 2b on fine level, i.e. for the computation of cells 1 and 3 (both on fine level) we would use points 1b and 2b (on the same level of resolution like the green/yellow points).



To work with a generating system the 4 stacks for hierarcical basis are insufficient. To deal with various levels of resolution and so with various point levels (a, b, c, ...) we need 4 0D-stacks and 4 1D-stacks for the two dimensional case. One can take as a rule for the number of stacks the following:

- 2-dimensional case:
  A plane in 2D has four corners (= 4 0D-stacks) and four lines (= 4 1D-stacks).
- 3-dimensional case:
  A cube in 3D has eight corners (= 8 0D-stacks), 12 lines (=12 1D-stacks) and 6 planes (= 6 2D-stacks).

1.5 Conclusion

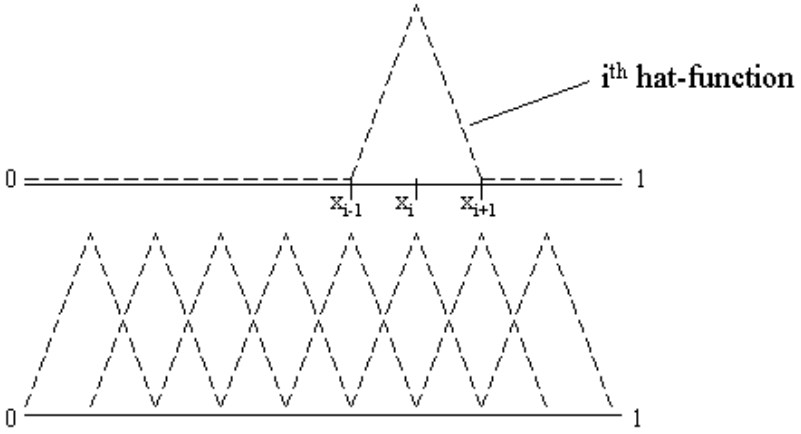
As a summary about space filling curves we only emphasize the worth of Peano's space filling curve. It allows us to organize the needed data for computation in a very efficient way: storing the information in a stack according to Peano's curve leads to high spatiality of the cache. Moreover the clear structure can be detected automatically (we don't need to invest additional work) by modern computer optimisation techniques like pre-fetching.

2 Hierarchical Basis

As already mentioned simulation time is composed of both performing the computational steps and data transfer. The latter can be optimised by space filling curves as shown. Now we will examine ideas how to reduce the computational costs, too. First recall the usage of nodal basis in FEM-analysis. Starting from its disadvantages we move on to hierarchical basis and motivated by higher dimensional problems end with an outlook to generating systems.

2.1 Nodal basis

Nodal basis are the standard approach for FEM-analysis. A function can be approximated by a linear superposition of various piecewise linear functions (hat-functions). When talking about nodal basis these hat-functions form a basis. Below example shows a set of hat-functions that don't treat the boundaries.



The i^{th} hat-function has the value 1 at position i, 0 at the two neighbouring positions and is linear between. We can approximate any function $u(x)$ by weighting these different ansatz functions.

$$u(x) = \sum_{i=1}^{n} u_i \times \varphi_i(x)$$

$$n : \text{number of hat-functions}$$
$$\varphi_i(x) : i^{\text{th}} \text{ hat-function}$$
$$u_i : \text{function value } u(x = x_i)$$

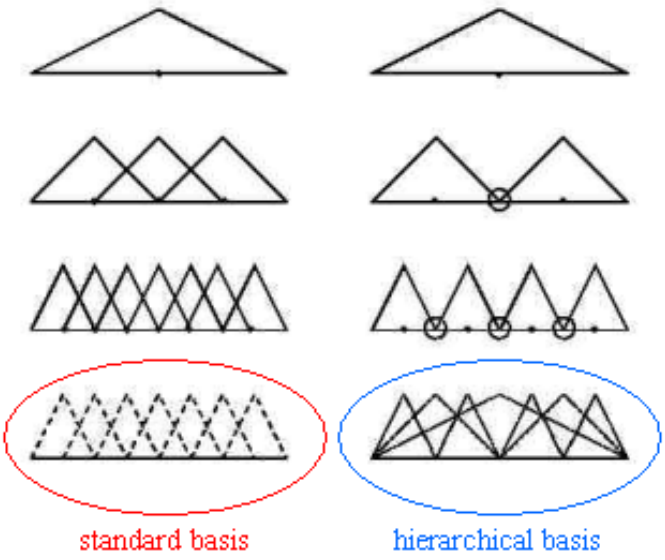
Using this approach will lead in case of the one dimensional Poisson equation $\ddot{u} = f(x)$ after discretization to below shown matrix A for the equation system $A \cdot u(x) = b(x)$. The condition depends on the ratio $\lambda_{max}/\lambda_{min}$, λ_{max} is given by the row sum (absolute values) whereas λ_{min} will head with increasing problem size (grid size) towards zero. This leads to a high condition number causing slow convergence of iterative solvers.

$$A = \begin{bmatrix} 1 & -\frac{1}{2} & & & & & \\ -\frac{1}{2} & 1 & -\frac{1}{2} & & & & \\ & -\frac{1}{2} & 1 & \ddots & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & \ddots & 1 & -\frac{1}{2} & \\ & & & & -\frac{1}{2} & 1 & -\frac{1}{2} \\ & & & & & -\frac{1}{2} & 1 \end{bmatrix}$$

Refining the grid (e.g. after a first rough simulation) at certain interesting positions would lead to a loss of so far made calculations. We cannot reuse already gained information. So a method showing faster convergence (hopefully independent of problem size) and supporting reuse when refining is desirable.

2.2 Hierarchical basis

Contrary to nodal basis which defines all hat-functions within the examined grid on the same level hierarchical basis starts with a hat-function on the coarsest level and then recursively refines on points which aren't used for hierarchically higher hat-functions. (Throughout the text we will use red colour to mark nodal basis and blue colour to mark hierarchical basis).

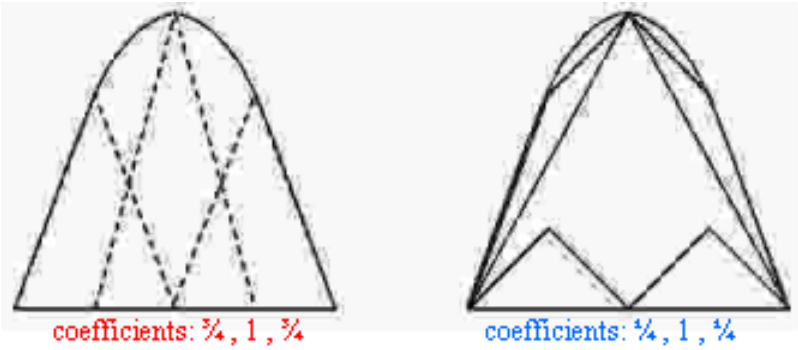


As with standard basis we can also approximate the function $u(x)$ by a hierarchical basis.

$$u(x) = \sum_{i=1}^{n} v_i \times \phi_i(x)$$

n : number of hat-functions
$\phi_i(x)$: i^{th} hierarchical hat-function
v_i : weighting coefficient

Unlike the nodal basis hierarchical basis cannot be summed up by weighting the single hat-functions with the function value at the corresponding positions. This is because of part of the function value $u(x = x_i)$ is already given by hierarchically higher hat-functions. In other words: We need the difference between absolute function value and already defined value by higher ansatz functions; this difference is given by the coefficient v_i. (Of course the first coefficient v_1 equals the function value as there are no coarser levels.) We can illustrate this by approximating the parabola defined by the points (0,0), (0.5,1) and (1,0).



The coefficients for the nodal basis equal the function values, the ones for the hierarchical basis on the finer level (1/4) are given by the difference between the exact function value (3/4) and the already defined value by the coarser hat-function (1/2). We can already state that the approximation by hierarchical basis will work more efficiently as the coefficients for lower hat-functions fall fast whereas the coefficients for the nodal basis are given by the function values at the corresponding points. This is the geometric interpretation of the faster convergence we will see later on. With hierarchical basis we are moreover able to refine locally without loosing the so far invested work. In fact we need the information of coarser levels for the

approximation and so refinement is nothing but introducing a new level in the hierarchical system.

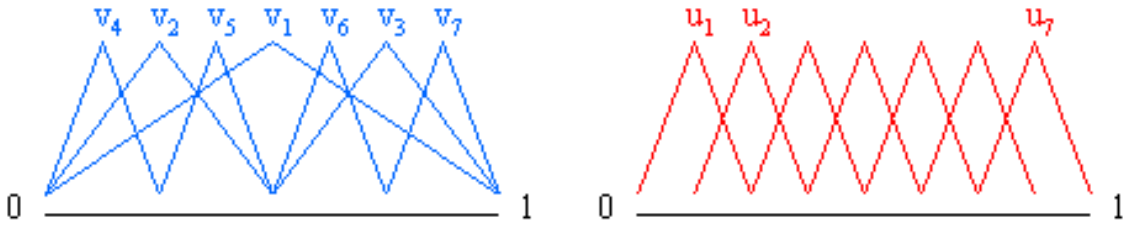
As shown we can describe the same function using either nodal or hierarchical basis. Therefore a transformation between these two basis is possible. Building the coefficients v_i out of u_i works according to the following formula:

$$v_{2k+1} = u_{2k+1} - \frac{u_{2k+2}+u_{2k}}{2}$$
$$k \leq \frac{n}{2} - 1$$

When applying this formula recursively on u_i we will get v_i. So we can find the values (-1/2, 1, -1/2) in the transformation matrix H.

$$v = H \cdot u$$
$$u = T \cdot v$$

For our aim - solving $A \cdot u(x) = b(x)$ more efficiently - the opposing transformation is of more importance. So let's create the transformation matrix T for the parabola example using 7 inner points for the hat-functions. The hat-functions are enumerated as shown below.



The corresponding transformation would look like the following (without treatment of boundaries):

$$\begin{bmatrix} u_1 \\ \vdots \\ u_7 \end{bmatrix} = T \cdot \begin{bmatrix} v_1 \\ \vdots \\ v_7 \end{bmatrix}$$

$$\begin{bmatrix} 7/16 \\ 3/4 \\ 15/16 \\ 1 \\ 15/16 \\ 3/4 \\ 7/16 \end{bmatrix} = \begin{bmatrix} 1/4 & 1/2 & & 1 & & & \\ 1/2 & 1 & & & & & \\ 3/4 & 1/2 & & & 1 & & \\ 1 & & & & & & \\ 3/4 & & 1/2 & & & 1 & \\ 1/2 & & 1 & & & & \\ 1/4 & & 1/2 & & & & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1/4 \\ 1/4 \\ 1/16 \\ 1/16 \\ 1/16 \\ 1/16 \end{bmatrix}$$

With this mapping we can rewrite our equation system $A \cdot u = b$. We first replace u by $T \cdot v$ and then multiply the whole equation from the left with T'.

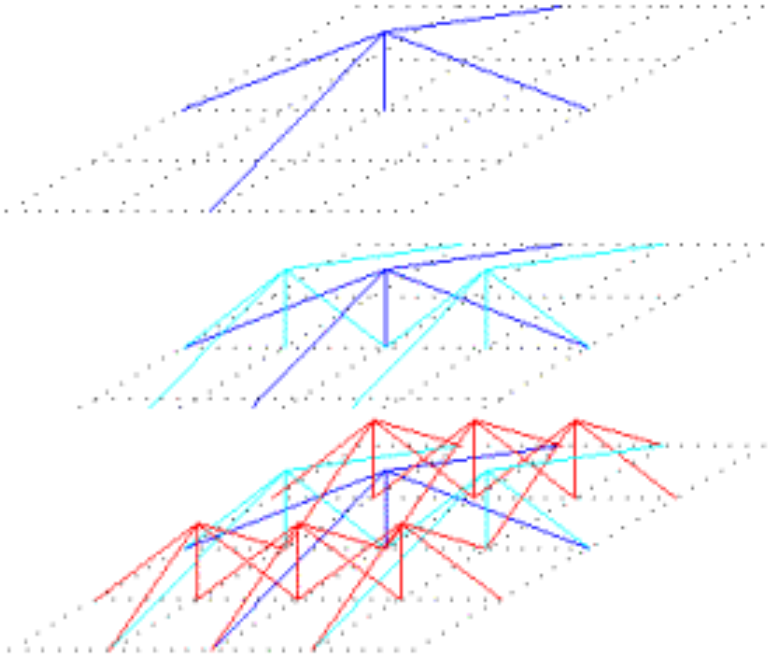
$$A \cdot u = b \Leftrightarrow A \cdot (T \cdot v) = b \Leftrightarrow T' \cdot A \cdot T \cdot v = T' \cdot b$$
$$A_2 = T' \cdot A \cdot T$$
$$b_2 = T' \cdot b$$

Our gain lies in the matrix A_2: it's simply a diagonal matrix for the one dimensional case. We can read off directly the solution with respect to v. Transforming v back to u is very simple: just multiply with T.

$$T'AT = \begin{bmatrix} 1/4 & & & & & & & \\ & 1/2 & & & & & & \\ & & 1/2 & & & & & \\ & & & 1 & & & & \\ & & & & 1 & & & \\ & & & & & 1 & & \\ & & & & & & 1 & \end{bmatrix}$$

We can also interpret this numerically: the smallest eigenvalue λ_{min} of the new system is no longer dependent on the problem size but a constant resulting in a good condition number and so the new system is much more easy to solve than the system using standard nodal representation.

Now we have a look on hierarchical hat-functions in 2D. These ansatz functions can be understood as product of the corresponding one dimensional hat-functions. Below sequence shows a possible way of generating those functions. We first start with the hat-function on the coarsest possible level (blue), then refine in one direction (cyan) and finally refine in the other direction (red).



When refining we can select to either first completely split one direction before refining the other or refine in changing order, i.e. to always work on both directions before going to a finer level. Of course the refinement needn't to be equal in both directions, we can use m levels for x and n levels for y.

As with the one dimensional problem we can exchange the nodal basis by the hierarchical one and so calculate $T'AT$ for the two dimensional case. It's not a diagonal matrix any longer but shows much better condition than the matrix A we would get when using nodal basis.

For higher dimensional problems (from dimension two on) there exists another interesting approach which will even beat the usage of hierarchical basis: generating system.

2.3 Generating system (outlook)

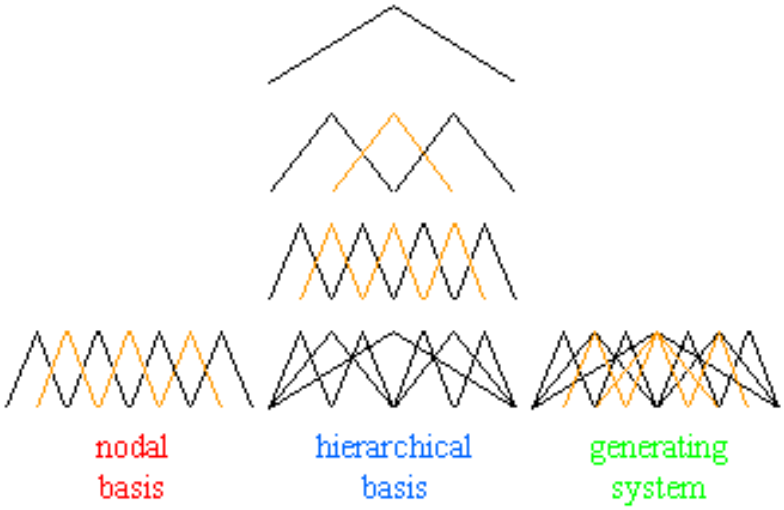
So far we restricted ourselves to ansatz functions forming a basis to describe the function $u(x)$. Using a basis has the advantage of describing a function uniquely. But we only need

to describe a function; the way of building this function is up to us. We can describe the same function using the necessary linearly independent ansatz functions and additionally as much linearly dependent functions as we like to use. As example we will build up the same 3 dimensional vector a using first a set of vectors forming a basis and then a set of linearly dependent vectors spanning the same vector space as the basis.

$$a = \begin{pmatrix} 3 \\ 2 \\ 4 \end{pmatrix} = 3 \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} + 2 \cdot \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} + 4 \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

$$a = \begin{pmatrix} 3 \\ 2 \\ 4 \end{pmatrix} = 1 \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} + 2 \cdot \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} + 2 \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} + 2 \cdot \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$$

Contrary to the hierarchical basis which only refines on points that aren't already defined by coarser hat-function a generating system refines on these points, too. One can also think of a generating system as the sum of all nodal basis of all levels of considerartion.



Using a generating system to approximate $u(x)$ leads to a linear system of equations in which as many eigenvalues equal zero as there are linearly dependent hat-functions. For the condition only the smallest non-negative eigenvalue is of interest. This is now independent of the examined dimension and problem size a constant and leads to fast convergence of the afterwards applied solver.

The solution we will find with respect to the generating system is not unique (one can build the same unique value in different ways using a generating system, see above vector example). Moreover the received solution depends on the starting value. But transforming any of these solutions to the nodal basis will always result in the same unique representation, i.e. the difference in the result only lies in the representation, not in the real result.