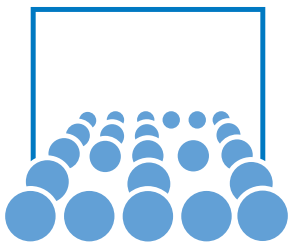
Algorithms of Scientific Computing

Hierarchical Methods and Sparse Grids
– Algorithms and Data Structures for Sparse Grids –

Michael Bader

Technische Universität München

Summer Term 2015



Part I

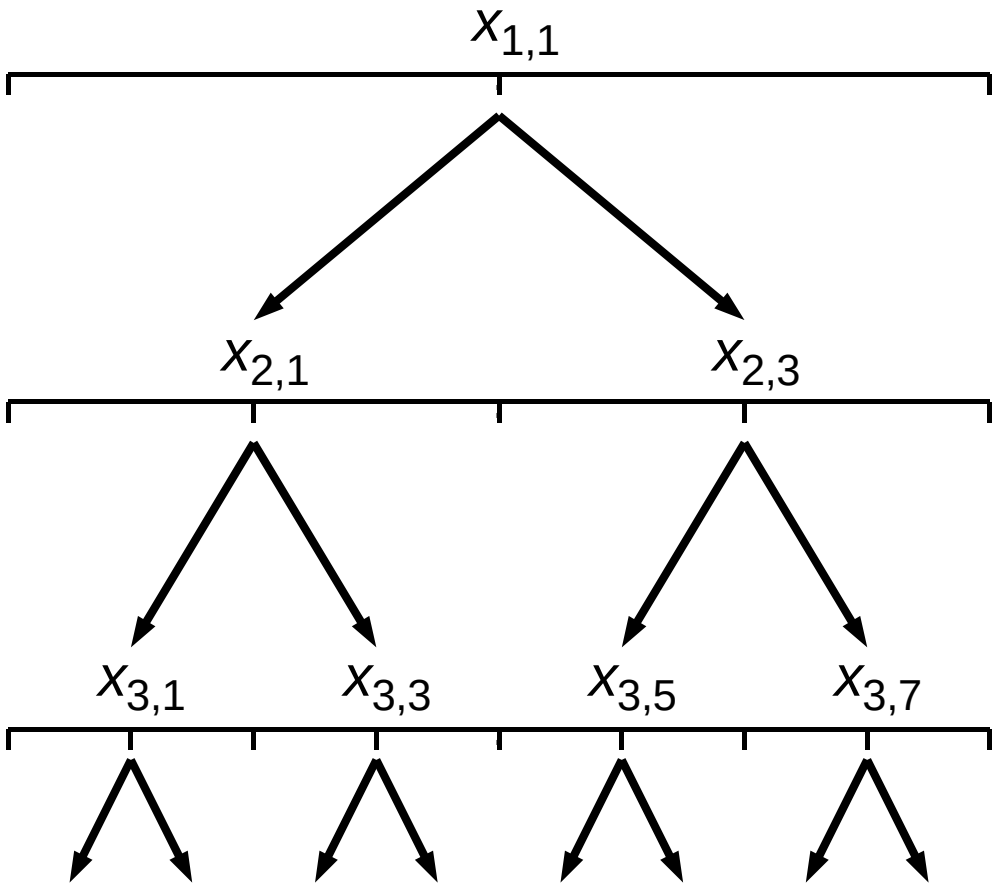
Algorithms and Data Structures

- We will now look at typical sparse grid algorithms
- Can, e.g., be used for hierarchization/dehierarchization, integration, data mining, solution of PDE, ...
- important: *adaptive* representation
- Algorithms depend on data structure:
  - Efficient traversal of sparse grid necessary
  - Thus, we deal with data structures for sparse grids, too

Data Structures ($d = 1$)

- How to store function $u : [0, 1] \to \mathbb{R}$ in hierarchical representation (i.e. surplusses $v_{\vec{l},\vec{i}}$)?
- Order and store grid points and associated values in binary tree
  - Root is node $x_{1,1} = 1/2$
  - Children of node $x_{l,i}$ are – if existent – the grid points $x_{l+1,2i-1}$ and $x_{l+1,2i+1}$ of level $l + 1$
  - Alternative point of view if child does not exist: Complete subtree of binary tree starting from child with all surplusses set to 0

Data Structures ($d = 1$) (2)



Typical Algorithms ($d = 1$)

Hierarchization and Dehierarchization

- Prototype for typical algorithm (c.f. tutorials)
- Our data structure has to allow
  1. Iteration over all grid points, considering the hierarchical relations
     - E.g. for hierarchization: first handle all grid points in the support of $\phi_{l,i}$, then compute $v_{l,i}$
  2. Access to *hierarchical neighbors*: grid points at interval boundaries of support of $\phi_{l,i}$ (if possible – exception for points 0 and 1 as not in the tree), e.g. to compute

$$v_{l,i} = u_{l,i} - \frac{1}{2}(u_l + u_r).$$

Typical Algorithms ($d = 1$) (2)

- Hierarchical neighbors are easy to find geometrically

$$x_{l,i-1}, \qquad x_{l,i+1}$$

- But have even indices $\Rightarrow$ really are on another level ($< l$)
- In the binary tree structure:
  - Can be found on way from root to node
  - One is parent node
- For hierarchization/dehierarchization: pass hierarchical neighbors as additional parameters
- Developing algorithms:
  - Try to store all information to process one node at the node and its hierarchical neighbors
  - Access to other nodes typically expensive
  - Tree traversal with “supply of hierarchical neighbors” only linear in number of nodes

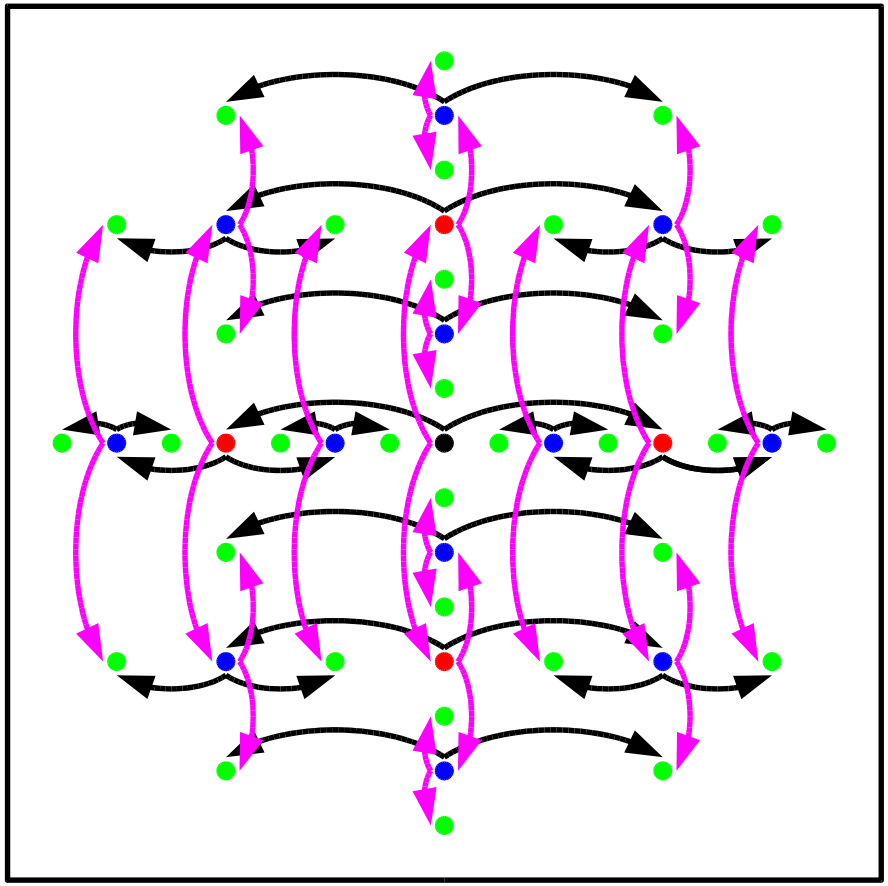
Data Structures and Typical Algorithms ($d > 1$)

- What data structure to use in more than one dimension?
- Algorithmically: use construction of basis functions as product of one-dimensional hats. Ideally:
  - Use a loop $1, \ldots, d$ over the dimension
  - Apply $1d$ algorithm on one-dimensional structures in each dimension (see also worksheet 7)

$\Rightarrow$ Need access to hierarchical neighbors in each spacial direction; implies to create binary tree structure in each dimension

- Disadvantages:
  - Storage requirements ($2d$ pointers)
  - High effort to keep structure consistent when inserting or deleting points

Data Structures and Typical Algorithms ($d > 1$) (2)



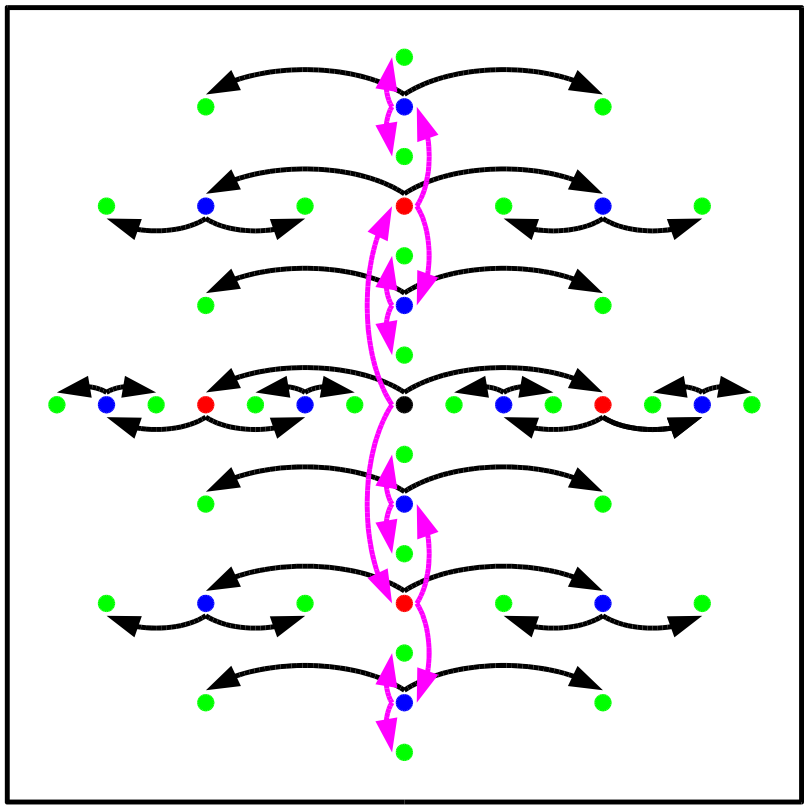
If you could recognize anything, it would be binary tree structures for rows (black) and columns (magenta)

Data Structures and Typical Algorithms ($d > 1$) (3)

Often better:

- Store in a node only two pointers for one direction (e.g. x_1)
- A binary tree of nodes is a row (a $1d$ structure parallel to the x_1 axis)
- For next spacial direction x_2, only a binary tree in x_2 direction required
- Stores one plane parallel to x_1–x_2 coordinate plane; nodes are the binary trees with $1d$ structures
- For each additional spatial direction x_d build binary tree with $(d-1)$-dimensional structures as nodes
- Disadvantage: Access to hierarchical neighbors not that easy any more (except for x_1-direction)
- But can be achieved without much more computational effort by suitable reordering of loops and tree traversals

Data Structures and Typical Algorithms ($d > 1$) (4)



Already more clear: One plane (two-dimensional structure) consists of one binary tree (magenta) of which the nodes are binary trees (black) for each row

Data Structures and Typical Algorithms ($d > 1$) (5)

Hash table

- Much more comfortable (and not too inefficient) alternative
- Store coefficients as target values, with, e.g., $(\vec{l}, \vec{i})$ as keys
- No need to care about tree structures
- Only requires computation of indices of accessed nodes (hierarchical neighbor, . . .)

$\Rightarrow$ Best solution for your own sparse grid experiments

Further assumptions on data structures

- Algorithms will assume that all hierarchical neighbors exist for each grid point

$\Rightarrow$ If creating grid points adaptively, create them if necessary

- No further assumptions

Data Structures for Regular Sparse Grids

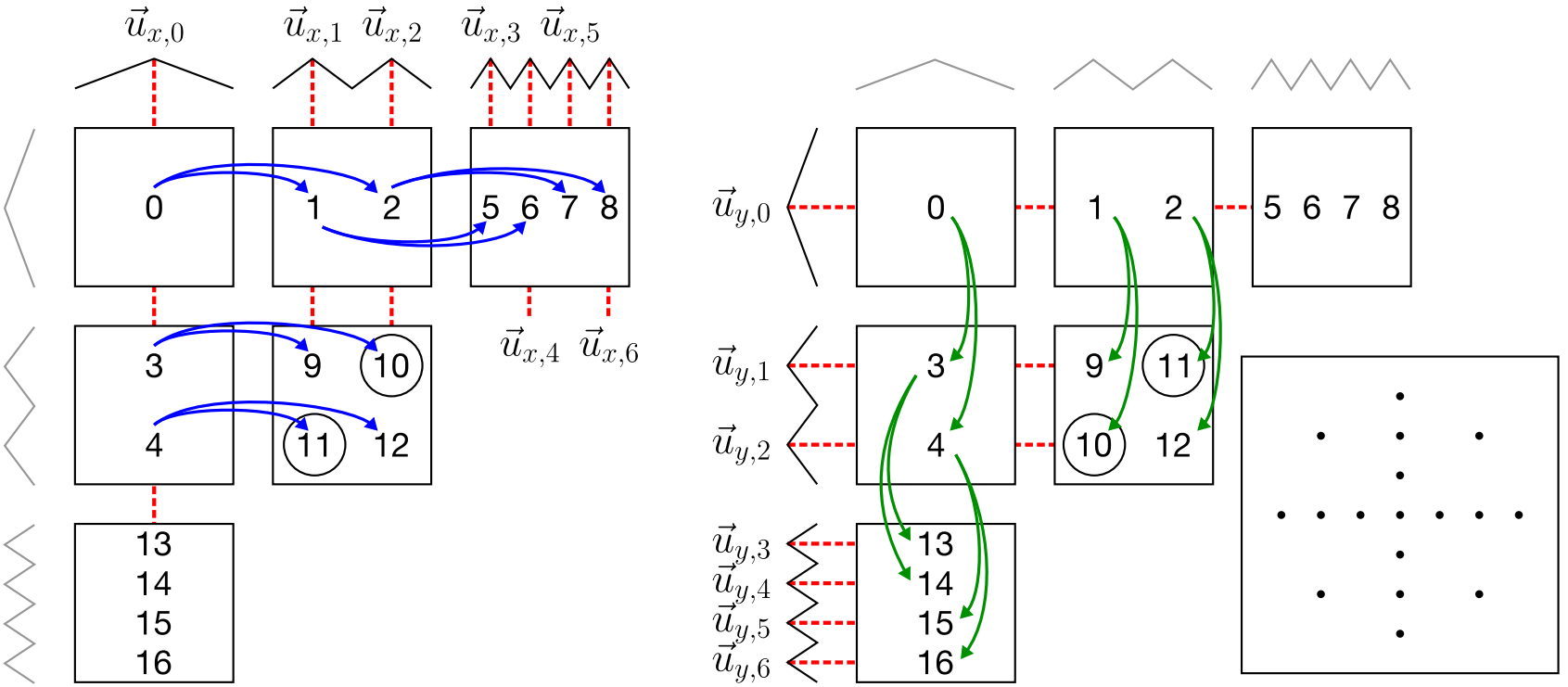
Array-Based Data Structures

- Cartesian meshes with $2^{l_1} \times 2^{l_2} \times \ldots \times 2^{l_d}$ grid points
- suggests classical array indexing similar to $i \cdot n + j$
  $\rightarrow$ question: what is the "fastest-running" index?
- number of grid points per subspace ist constant along "diagonals", i.e., for constant $|l|_1$
  - $\rightarrow$ sequentialized storage scheme for subspaces
  - $\rightarrow$ start of each subgrid can be easily computed
  - $\rightarrow$ index offset to hierarchical neighbours may be computed
- additional considerations: best layout for vectorization, parallelization, etc.

Towards Dimensional Adaptivity

- add or remove entire subspaces/subgrids in an adaptive fashion
- may introduce higher accuracy only in selected dimensions

Data Structures for Regular Sparse Grids (2)



Example – Array-Based Data Structures (Buse et al., ISPDC 2012)

- note uniform vs. non-uniform index offset for access to hierarchical parents/neighbours in x- vs. y direction

Summary

Data Structures

- array-based for regular sparse grids and combination technique (see tutorials)
- hierarchical adaptivity reflected by tree-based data structures (but: more complicated in higher dimensions)
- hash-based data structures
- dimensional adaptivity allows to stick to array-based data structures

Algorithms

- hierarchisation and dehierarchisation: tree-based recursion plus “hierarchical neighbours”
- archimedes quadrature $\rightarrow$ recursion on dimensions
- much more complicated algorithms, if we want to use sparse grids for solution of partial differential equations

Part II

Remember: Classification with Sparse Grids

Recall: Sparse Grid Classification

- Given a training set (normalized)

$$S := \left\{ (\vec{x}_i, y_i) \in [0,1]^d \times \{+1, -1\} \right\}_{i=1}^{m}$$

- Reconstruct piecewise d-linear sparse grid approximation u of f:

$$f_N(\vec{x}) = \sum_{i=1}^{N} v_i \phi_i(\vec{x})$$

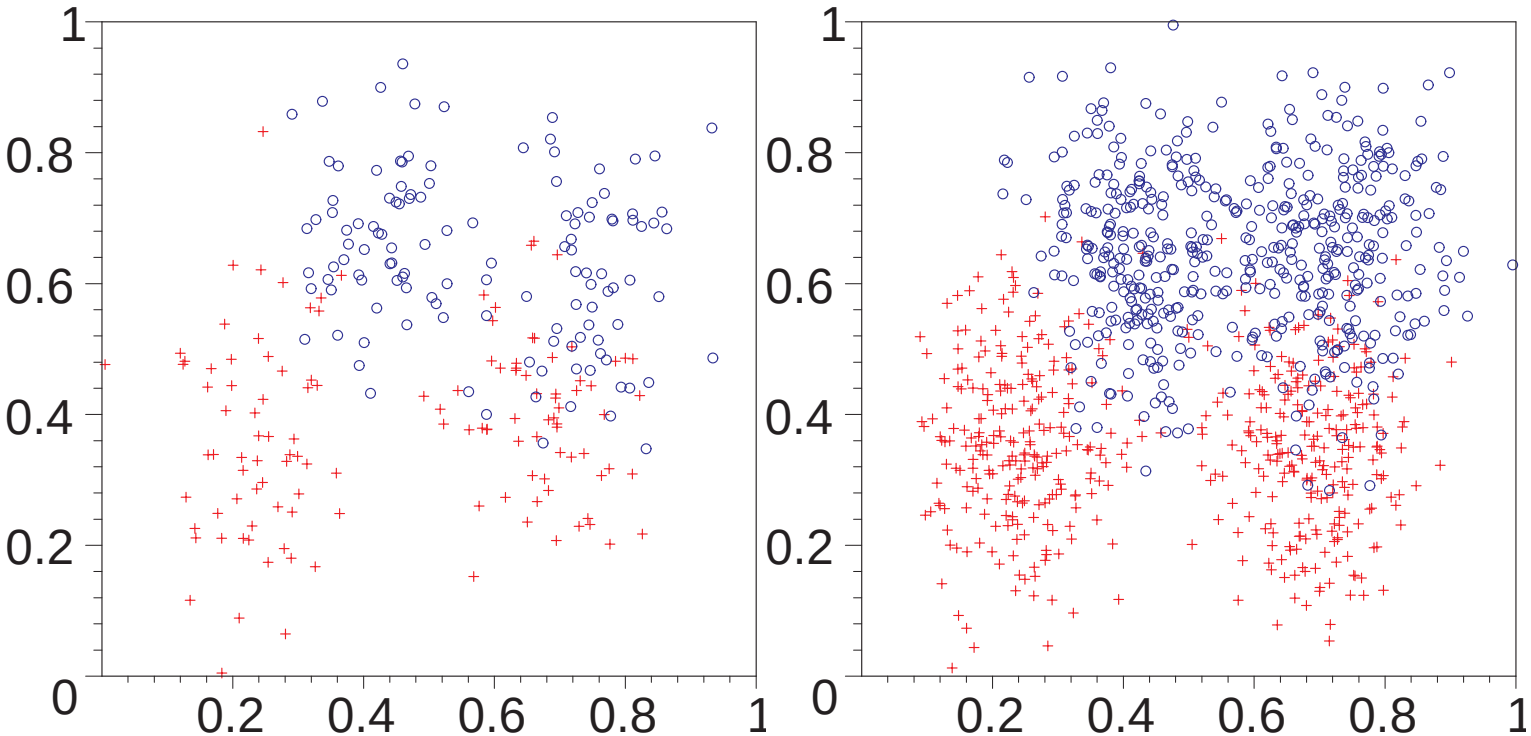
- Solve regularized least squares problem

$$f_N \stackrel{!}{=} \arg\min_{f_N \in V_N} \left(\frac{1}{m} \sum_{i=1}^{m} \left(y_i - f_N(\vec{x}_i) \right)^2 + \lambda \| \nabla f_N \|_{L_2}^2 \right)$$

with $\|g\|_{L_2}^2 := \int g^2 \, \mathrm{d}\vec{x}$

Example 1 – Ripley Data Set

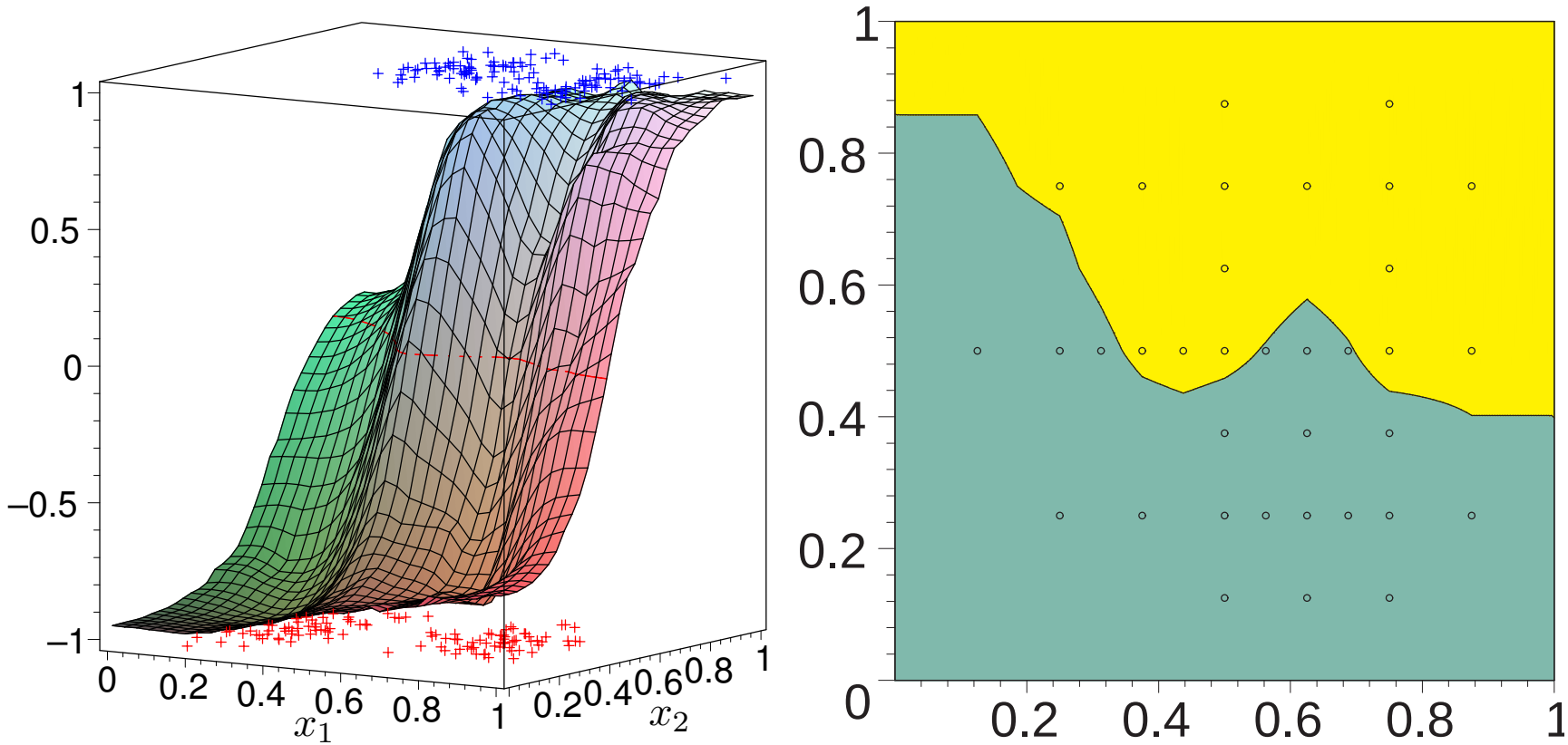
- Artificial, *2d* data set, frequently used as a benchmark (mixture of Gaussian distributions plus noise)
- 250 points for training, 1000 to test on



- Constructed to contain 8% of noise

Ripley Data Set Using Sparse Grids

- Compute adaptive sparse grid classifier, e.g.:



- Best accuracy: 91.5% on test data (max. 92%)
- Suitable treatment of boundary needed

From Minimization to System of Linear Equations

- Leads to N linear equations for N unknowns

$$\left(\frac{1}{m}BB^T + \lambda C\right)\vec{v} = \frac{1}{m}B\vec{y}\,,$$

- With matrices C and B

$$(C)_{ij} = \langle \nabla\phi_i(\vec{x}), \nabla\phi_j(\vec{x})\rangle_{L_2}, \qquad (B)_{ij} = \phi_i(\vec{x}_j)$$

Now: use Sparse Grids!

- How do the matrices B and C look like?
- Should we explicitly set up B and C or is there a better solution?
- In 1D: C is a **diagonal matrix**! (However: B is complicated)
- In general: is there a level-wise hierarchical/recursive algorithm?

Towards a Hierachical/Recursive Algorithm

- Consider right-hand side $\frac{1}{m} B\vec{y}$:

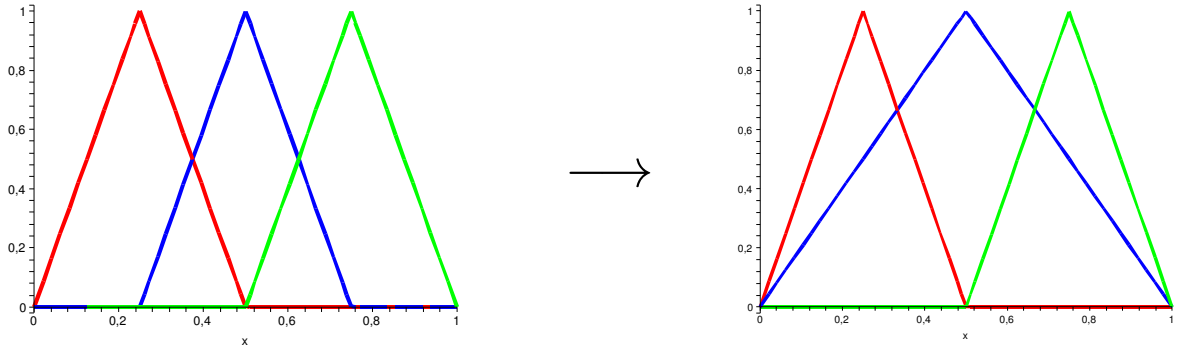
$$(B\vec{y})_i = \sum_j B_{ij} y_j = \sum_j \phi_i(x_j) y_j$$

- Consider a nodal basis $\phi_i(x_j) = \delta_{ij}$, then $B = I$ and $(B\vec{y})_i = y_i$
- Hierarchical transform when using hierarchical basis $\psi_i(x)$?

Approach:

- Consider vectors of basis functions $\vec{\psi} = (\psi_i)_i$ and $\vec{\phi} = (\phi)_i$
- Show that then $\psi = H\phi$ (matrix-vector product)
- Then: $\sum_j \psi_i(x_j) y_j = \sum_j (H\phi)_i(x_j) y_j = (HB\vec{y})_i$
- As $(B\vec{y})_i = y_i$ for nodal basis: $\sum_j \psi_i(x_j) y_j = (H\vec{y})_i$
- Now: how does H look like and how do we compute $H\vec{y}$?

Hierarchical Transform on Basis Functions



- represent hat functions $\phi_{n-1,i}(x)$ via fine-level functions $\phi_{n,j}(x)$

$$\phi_{n-1,i}(x) = \frac{1}{2}\phi_{n,2i-1}(x) + \phi_{n,2i}(x) + \frac{1}{2}\phi_{n,2i+1}(x)$$

- hierarchical-basis transformation as matrix-vector product:

$$\begin{pmatrix} \psi_{n,i-1}(x) \\ \psi_{n,i}(x) \\ \psi_{n,i+1}(x) \end{pmatrix} := \begin{pmatrix} \phi_{n,2i-1}(x) \\ \phi_{n-1,i}(x) \\ \phi_{n,2i+1}(x) \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ \frac{1}{2} & 1 & \frac{1}{2} \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \phi_{n,2i-1}(x) \\ \phi_{n,2i}(x) \\ \phi_{n,2i+1}(x) \end{pmatrix}$$

Hierarchical Basis Transformation (2)

- hierarchical basis transformation: $\psi_{n,i}(x) = \sum_j H_{i,j}\phi_{n,j}(x)$
- written as matrix-vector product: $\vec{\psi}_n = H_n\vec{\phi}_n$
- H can be written as a sequence of level-wise transforms:

$$H_n = H_n^{(1)} H_n^{(2)} \ldots H_n^{(n-2)} H_n^{(n-1)}$$

- where each transform has a shape similar to

$$H_3^{(2)} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & 1 & \frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 1 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & 1 & \frac{1}{2} \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

- important: can be implemented in linear complexity!

Classification with Sparse Grids

Notes on Implementation

- in higher dimensions: nodal basis leads to simple matrix structures, but the systems of equations are difficult to solve
  $\rightarrow$ **most important:** curse of dimensionality kicks in
- hierachical basis leads to system of equations that can be solved efficiently;
  $\rightarrow$ complicated matrix structures,
  $\rightarrow$ algorithms based on hierarchization/dehierarchization
- **sparse grids:** implementation is not just a hierarchization of node basis
  $\rightarrow$ complicated hierarchical, recursive algorithms
  $\rightarrow$ mitigates curse of dimensionality!