Algorithms for Scientific Computing

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

Michael Bader
Technical University of Munich

Summer 2022

TUM Uhrenturm

Part I

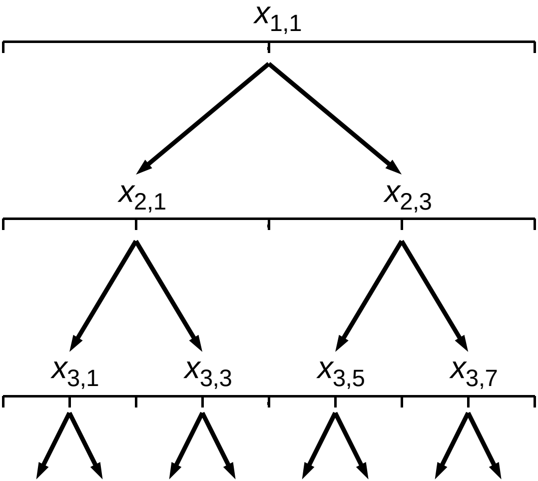
Algorithms vs. Data Structures

- Consider typical sparse grid algorithms, such as: hierarchization/dehierarchization, integration, classification, 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 $v_{\vec{l},\vec{i}}$)?
- Simplest choice: array $\to$ does not allow adaptivity
- 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 ($< l$)
- Thus, in the binary tree structure:
  - Can be found on way from root to node
  - One of the two indices is the 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 may be expensive (esp. for trees)
  - Note: complexity of a tree traversal with “supply of hierarchical neighbors” is at most linear in number of nodes

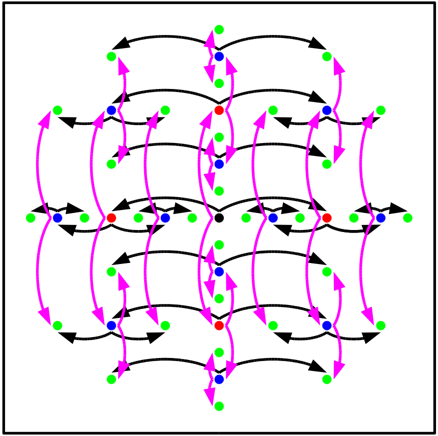
Data Structures and Typical Algorithms ($d > 1$)

- What data structure to use in more than one dimension?
- Algorithmically: use construction of basis functions as product of 1D hat functions. Ideally:
  - Use a loop $1, \ldots, d$ over the dimensions
  - 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 watch closely, you recognize separate binary tree structures for rows (black) and columns (magenta)

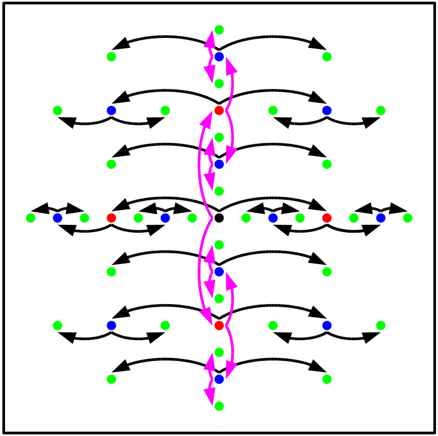
Data Structures and Typical Algorithms ($d > 1$) (3)

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

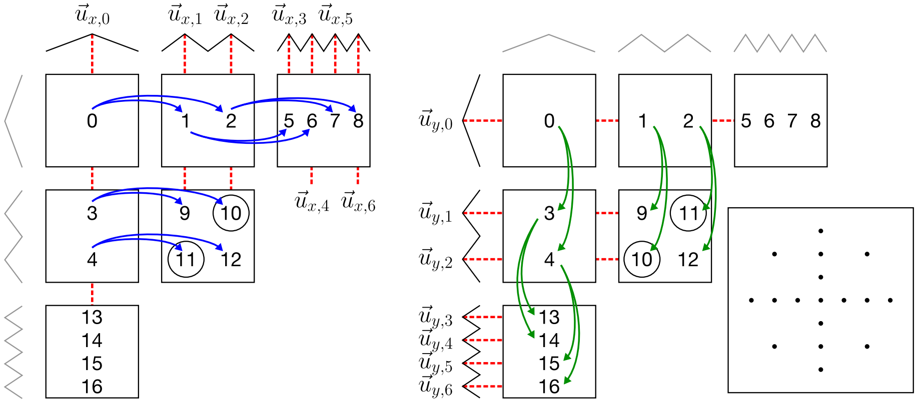
Array-Based Data Structures

- 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 is 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

Classification with Sparse Grids

Recall: Classification in 1D

- Given: training set (normalized)

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

- Find approximation f_N:

$$f_N(x) = \sum_{j=1}^{N} v_j \phi_j(x)$$

- Classical approach: minimize quadratic error

$$\sum_{i=1}^{m} \left(f_N(x_i) - y_i\right)^2 \stackrel{!}{=} \min \quad \Leftrightarrow \quad \sum_{i=1}^{m} \left(\sum_{j=1}^{N} v_j \phi_j(x_i) - y_i\right)^2 \stackrel{!}{=} \min$$

- Solution obtained via “least squares”: $G^T G v = G^T y$, where $G_{ij} = \phi_j(x_i)$

Recall: Least Squares Solution

- minimize quadratic error $\rightsquigarrow$ find values v_j that minimize

$$\sum_{i=1}^{m}\left(\sum_{j=1}^{N} v_j\phi_j(x_i) - y_i\right)^2 \quad \text{or} \quad \sum_{i=1}^{m}\left(\sum_{j=1}^{N} G_{ij}v_j - y_i\right)^2 \quad \text{with } G_{ij} := \phi_j(x_i)$$

- approach: set all partial derivatives $\frac{\partial}{\partial v_k}$ to zero

$$\frac{\partial}{\partial v_k}\left(\sum_{i=1}^{m}\left(\sum_{j=1}^{N} G_{ij}v_j - y_i\right)^2\right) = \sum_{i=1}^{m}\frac{\partial}{\partial v_k}\left(\sum_{j=1}^{N} G_{ij}v_j - y_i\right)^2 = 0$$

$$\Leftrightarrow \sum_{i=1}^{m} 2\left(\sum_{j=1}^{N} G_{ij}v_j - y_i\right)G_{ik} = 2\sum_{i=1}^{m}\left(\sum_{j=1}^{N} G_{ik}G_{ij}v_j - G_{ik}y_i\right) = 0$$

$$\Leftrightarrow \sum_{i=1}^{m}\sum_{j=1}^{N} G_{ik}G_{ij}v_j - \sum_{i=1}^{m} G_{ik}y_i = \underbrace{\sum_{i=1}^{m} G_{ik}\underbrace{\sum_{j=1}^{N} G_{ij}v_j}_{=(Gv)_i}}_{=(G^TGv)_k} - \underbrace{\sum_{i=1}^{m} G_{ik}y_i}_{=(G^Ty)_k} = 0$$

Classification and Regularization

- Possible problem: “overfitting”
  include penalty term to minimize gradient (or similar property) of f_N
  to avoid oscillations due to noise in training data
- Solve **regularized** least squares problem

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

  with $\|g\|_{L_2}^2 := \int g^2 \, \mathrm{d}\vec{x}$
- minimize quadratical error and prevent overfitting
  $\rightarrow$ Parameter λ to control trade-off

Classification and Regularization (2)

How to minimize $\lambda \|\nabla f_N\|^2_{L_2}$:

- Piecewise linear function f_N:

$$\nabla f_N(x) = f_N'(x) = \sum_{i=1}^{N} v_i \phi_i'(x) \quad \Rightarrow \quad \|\nabla f_N\|^2_{L_2} = \int \left(\sum_{i=1}^{N} v_i \phi_i'(x) \right)^2 \mathrm{dx}$$

- Minimize $\rightsquigarrow$ set partial derivatives w.r.t. all v_i to 0:

$$\rightsquigarrow \frac{\partial}{\partial v_k} \left(\|\nabla f_N\|^2_{L_2} \right) = \cdots = 2 \sum_j C_{jk} v_j, \quad \text{with} \;\; C_{jk} := \int \phi_j'(x) \phi_k'(x), \mathrm{dx}$$

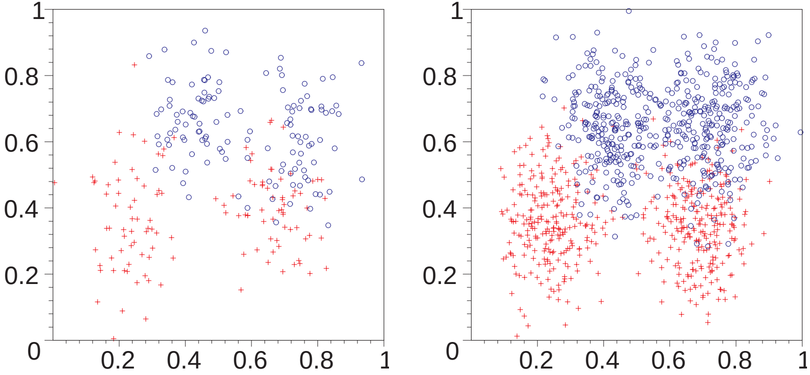
- Thus: to solve regularized least squares problem

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

solve linear system of the following form: $\frac{1}{m} G^T G v + \lambda C v = \frac{1}{m} G^T y$

Example 1 – Ripley Data Set

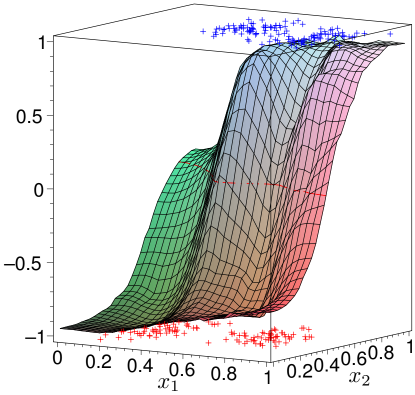
- Artificial, 2*d* data set, frequently used as a benchmark (mixture of Gaussian distributions plus noise)
- 250 points for training, 1000 to test on

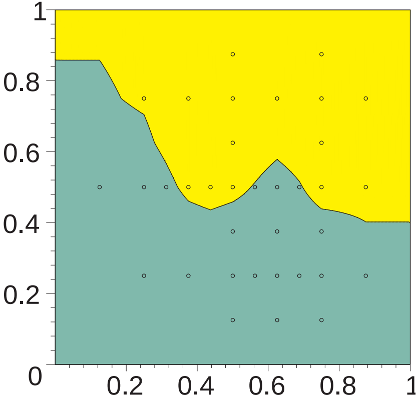


- Constructed to contain 8% of noise

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

- d-dim. problem; find function $f_N(\vec{x}) = \sum_{\vec{l}} \sum_{\vec{i}} v_{\vec{l},\vec{i}} \phi_{\vec{l},\vec{i}}(\vec{x})$ such that

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

- Again leads to N linear equations for N unknowns (m data points)

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

Questions when using Sparse Grids:

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

Recall: Structure of G for Hierarchical Basis

Structure of the matrix G, where $G_{ij} = \phi_j(x_i)$,
where $\{\phi_1, \phi_2, \phi_3\} = \{\phi_{2,1}, \phi_{1,1}, \phi_{2,3}\}$:

- again assume N_j data points per interval $[\xi_{j-1}, \xi_j]$
- Example structure then (again with 3 basis functions, 4 intervals, 10 data points):

$$G^T := \left(\begin{array}{cc|ccc|cccc|c} * & * & * & * & * & 0 & 0 & 0 & 0 & 0 \\ * & * & * & * & * & * & * & * & * & * \\ 0 & 0 & 0 & 0 & 0 & * & * & * & * & * \end{array}\right)$$

- $G^T G$ no longer tridiagonal $\rightarrow$ expect a denser matrix

Towards a Hierachical/Recursive Algorithm

- Consider right-hand side $\frac{1}{m} G^T \vec{y}$:

$$(G^T \vec{y})_i = \sum_j G^T_{ij} y_j = \sum_j \phi_i(\vec{x}_j) y_j$$

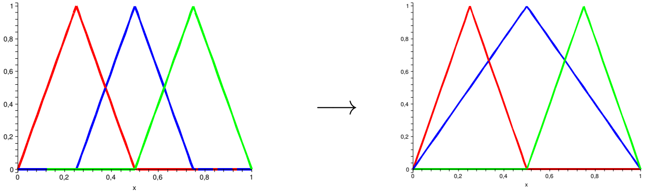
(note that we switch to 1D numbering of basis functions $\rightsquigarrow \psi_i$)

- Consider a nodal basis $\phi_i(\vec{x}_{n,j}) = \delta_{ij}$, then G^T easy to set up and $(G^T \vec{y})_i$ easy to compute
- 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: $\sum_j \psi_i(x_j) y_j = \sum_j (H\phi)_i(x_j) y_j = (HG^T \vec{y})_i$
- Do not set up matrix $HG^T \rightarrow$ perform as two matrix-vector products
- Now: how does H look like and how do we compute $H\vec{y}$?

Recall: Hierarchical Basis Transformation



- represent "wider" hat function $\phi_{1,1}(x)$ via basis functions $\phi_{2,j}(x)$

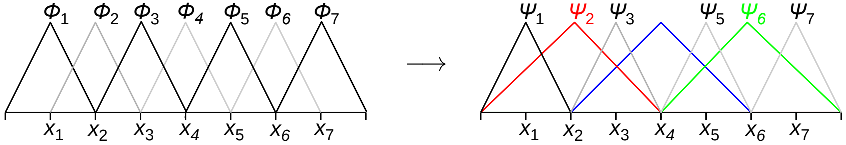
$$\phi_{1,1}(x) = \frac{1}{2}\phi_{2,1}(x) + \phi_{2,2}(x) + \frac{1}{2}\phi_{2,3}(x)$$

- consider vector of hierarchical/nodal basis functions and write transformation as matrix-vector product:

$$\begin{pmatrix} \psi_{2,1}(x) \\ \psi_{2,2}(x) \\ \psi_{2,3}(x) \end{pmatrix} := \begin{pmatrix} \phi_{2,1}(x) \\ \phi_{1,1}(x) \\ \phi_{2,3}(x) \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ \frac{1}{2} & 1 & \frac{1}{2} \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \phi_{2,1}(x) \\ \phi_{2,2}(x) \\ \phi_{2,3}(x) \end{pmatrix}$$

Recall: Hierarchical Basis Transformation (2)

Consider “semi-hierarchical” transform:



Matrices for change of basis are then: ($H_3^{(2)}$ to transform to hierarchical basis)

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

Recall: Hierarchical Basis Transformation (3)

Level-wise hierarchical transform:

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

  For k from 1 to n-1
  $\quad \vec{\phi}_n := H_n^{(k)}\vec{\phi}_n$

- matrix H_n for hierarchical basis transformation is thus:

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

- where each level-wise transform $H_n^{(k)}\vec{\phi}_n$ has a simple loop implementation:

  For j from 2^k *to* 2^n *step* 2^k
  $\quad \phi_{n,j} := \frac{1}{2}\phi_{n,j-2^{k-1}} + \phi_{n,j} + \frac{1}{2}\phi_{n,j+2^{k-1}}$

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!