Exercise 1: Hierarchization in Higher Dimensions

In this exercise we will implement the multi-recursive algorithm for hierarchization of a multi-dimensional regular sparse grid. The structure of the code resembles strongly the one-dimensional case, and so we again have a class (PagodaFunction) representing our grid points.

(i) Implement the refinement criterion MinLevelCriterion that adds all points up to a specified level to a given grid.
   
   **Hint:** In your grid traversal, try to avoid multiple visits to the same grid points.

(ii) Implement the function hierarchize efficiently using a recursive approach.
   
   **Hint:** The underlying traversal algorithm can be implemented similar to the one in (i).

(iii) Implement a function to compute the volume of the sparse grid interpolant.

*For the solution code look at the pdf exported from IPython Notebook.*

Exercise 2: The Combination Technique – A Different View on Sparse Grids

Dealing with hierarchical bases often turns out to be sophisticated. On this worksheet we will therefore see how the so-called combination technique finds a sparse grid interpolant, that approximates a function on a number of full grids, each consisting only of a “relatively small” number of grid points.

Let $u_l (l \in \mathbb{N}^2)$ for a $u : [0, 1]^2 \rightarrow \mathbb{R}$ the interpolant in $V_l$ (interpolating piecewise bilinearly at the inner grid points, at the boundary $u$ is assumed to be zero again).

(i) $V_l$ can be decomposed into a set of subspaces $W_l$. Accordingly, the interpolant $u_l \in V_l$ can be written as a sum of $w_l \in W_l$.

   Spot the grid associated with $u_{(3,2)}$ in the right part of Figure 1. Identify those subspaces in the left part that are needed to reconstruct $u_{(3,2)}$. 
We need to “collect” those $w_l$ with indices bound component-wise by $l$. In subspace scheme (see left hand side of Figure 1) these are the subspaces in the rectangular left upper part relative to $l$. Written as a sum this gives us:

$$u_l = \sum_{l' \leq l} w_{l'}$$

Note that weights are not needed yet.

(ii) Use the result from (i) to rewrite

$$\sum_{|l|_1 = n+1} u_l, \quad n \in \mathbb{N}$$

for the two-dimensional case as a weighted sum of $w_l$.

**Hint:** Look at the subspace scheme in Figure 1 and count the occurrences of each subspace in the sum. What do you notice when comparing $w_l$ with common level $n = |l|_1 + \text{dim} - 1$?

Using the previous result we start with

$$\sum_{|l|_1 = n+1} u_l = \sum_{|l|_1 = n+1} \sum_{l' \leq l} w_{l'}.$$  

For the reorganization part we first count which $w_{l'}$ appears how many times in the sums. (remember, this is 2d!)

- $|l'|_1 = n + 1$: each $w_l$ has one occurrence
- $|l'|_1 = n$: each $w_l$ has two occurrences
- ...
\[ |l'|_1 = k: \text{ each has } n + 2 - k \text{ occurrences} \]

Technical explanation: Let \( l \) be of level \( n + 1 \), i.e. \( |l|_1 = n + 1 \). From \( l \) construct all possible \( l' \) of level \( n \) with \( |l'|_1 \leq l \). Those are exactly two, as there are two components (2d!) that can be decreased by 1. Applying this scheme down to level 1 then leads to the result above.

Written as an explicit sum formula this is:

\[
\sum_{|l|_1 = n + 1} w_l = \sum_{|l|_1 \leq n + 1} (n + 2 - |l'|_1) w_{l'}.
\]

(iii) In the final step use the previous results to give a representation of the sparse grid interpolant

\[
u_n^D := \sum_{|l|_1 \leq n + 1} w_l
\]

as a weighted sum of \( u_l \). Again, count the occurrences of the \( w_l \).

By looking at the subspace scheme rather than by looking at the formulas it becomes clear that the following holds:

\[
\sum_{|l|_1 \leq n + 1} w_l = \sum_{|l|_1 = n + 1} u_l - \sum_{|l|_1 = n} u_l
\]

(iv) Assume you are talking to a person who knows how to approximate the volume \( F_2(u) \) through the trapezoidal rule (in 2d) with respect to \( u_l \). Give instructions on how to write a program that implements a sparse grid approximation of \( F_2(u) \). Remember Archimedes quadrature.

- First idea: Replace volume \( F_2(u) \) by the sparse grid volume approximation \( F_2(u_n^D) \).

- Second idea: Think of the interpolant as a sum of \( u_l \). We know those \( u_l \) (interpolating \( u \) on regular grids) as well as their volumes (trapezoidal rule in 2d).

- Together with the weights from the previous part we get

\[
F_2(u) \approx F_2(u_n^D) = \sum_{|l|_1 = n + 1} F_2(u_l) - \sum_{|l|_1 = n} F_2(u_l)
\]

(v) Compare this method with Archimedes quadrature — what are the (dis-)advantages?

Advantages:

- Simpler program code (Haven’t you tried coding them? Do it! You’ll agree...)

- It might be possible to reuse an existing program for the trapezoidal rule on common regular grids (advantage is even bigger for more complex applications, e.g. when computing a sparse grid solution for a fluid simulation)

- For comprehensive computations the program is more likely and easy to be parallelized as the single grids are processed independently from each other

Disadvantages:

- No straight forward approach to include adaptivity, i.e. it’s not possible to automatically find the right evaluation points

- Recursion is much more beautiful!