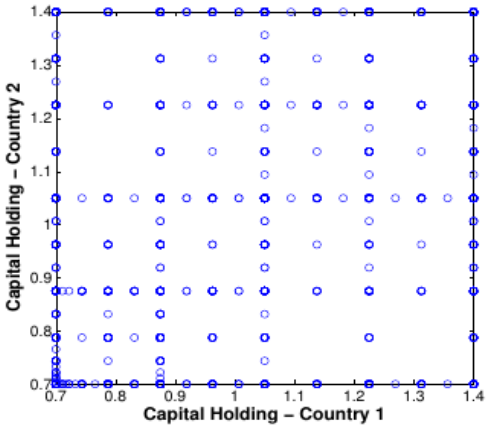
UNIL | Université de Lausanne

An introduction to global solution methods

Simon Scheidegger
simon.scheidegger@unil.ch

Rochester, November 18th - November 20th, 2019



Road-map – fast forward:

Lecture 1: Monday, November 18th

- Introduction to Sparse Grids and Adaptive Sparse Grids.

- Dynamic Programming with (Adaptive) Sparse Grids.

Lecture 2: Tuesday, November 19th

- Introduction Machine Learning (supervised and unsupervised machine learning).

- Basics on Gaussian Process Regression (supervised machine learning).

Road-map – fast forward (2):

Lecture 3: Wednesday, November 20th

- Bayesian Gaussian Mixture Models (unsupervised machine learning).

- Dimension-reduction with the active subspace method.

- Solving dynamic models on high-dimensional, (irregularly-shaped) state spaces.

Today – (Adaptive) Sparse Grids

I. Motivation – “the curse of dimensionality”

II. From Full (Cartesian) Grids to Sparse Grids

III. Adaptive Sparse Grids

IV. Gain hands-on experience with some libraries

V. A growth model solved by DP and Sparse Grids

I. Why Global Solution Methods?

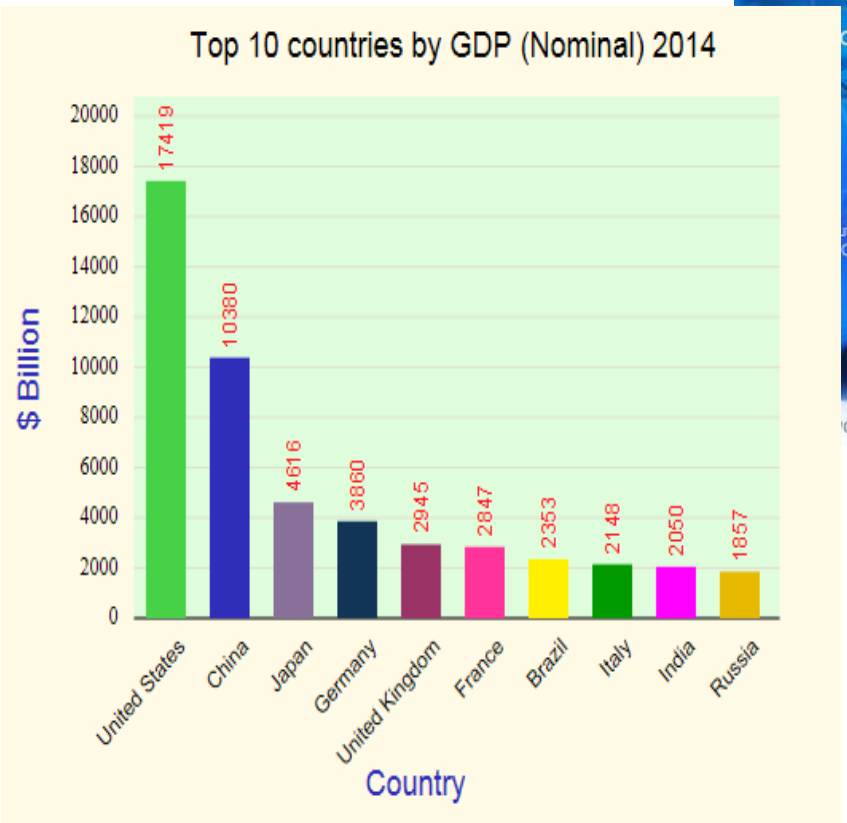
- Modern dynamic economic models are extremely **rich** to capture **all the effects of interest**:
  - large stochastic shocks that lead to highly non-linear policies (e.g., rare disaster shocks).
  - many agents that lead to a high-dimensional state space.
  - ...
- Standard solution techniques such as log-linearisation **often fail to deliver reliable results** across the entire domain of interest.
- The latter method may be useful to describe an economy that operates in normal times, but **fails** in the presence of non-linearities such as occasionally binding constraints, among other types of salient features of the economic reality that the modellers would like to capture appropriately in their models.

Example – Heterogeneity in IRBC models

- Model trade imbalance

- FX rates

-...



- How many regions does a minimal model have?
- Are policy functions smooth? (borrowing constraints)

→ **Model heterogeneous & high-dimensional**

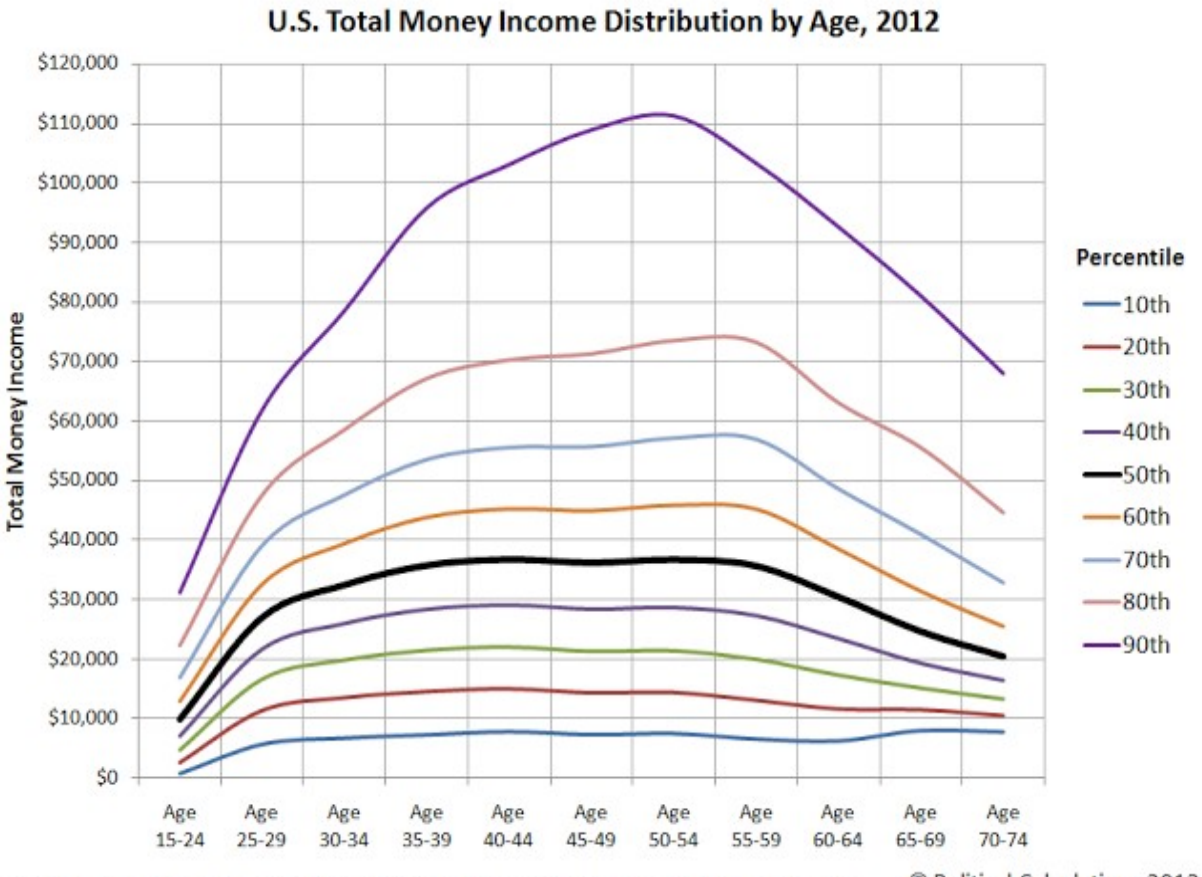
Example – Heterogeneity in OLG* models

*Overlapping generation models

To model e.g. social security:

- How many age groups?
- borrowing constraints?
- aggregate shocks?
- …

→ Model: heterogeneous & high-dimensional



Financial markets: non-Gaussian returns

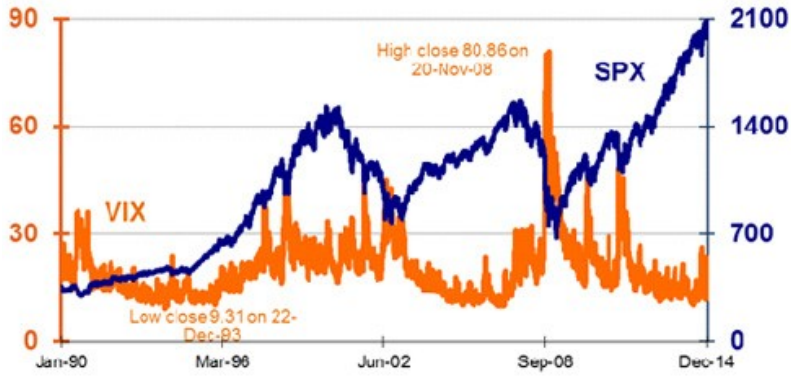
- Derivative contracts giving a right to buy or sell an underlying security.
  - *European* if exercise at expiration only.
  - ***American*** if exercise any time until expiration.

- American options are extremely challenging:
  → Dynamic optimization problem.

- Basic models do not describe dynamics accurately (e.g., Hull (2011)).

- Financial returns are often not Gaussian.

- Realistic models are hard to deal with, as they need many factors.
  → Curse of dimensionality.



Dynamic Programming/Value Function Iteration

e.g. Stokey, Lucas & Prescott (1989), Judd (1998), ...

Dynamic programming seeks a time-invariant policy function ***p*** mapping a state $\mathbf{x}_t$ into the control $\mathbf{u}_t$ such that for all $t \in \mathbb{N}$ $u_t = p(x_t)$

The solution is approached in the limit as $j \longrightarrow \infty$ by iterations on:

$$V_{j+1}(x) = \max_u \{r(x,u) + \beta V_j(\tilde{x})\}$$

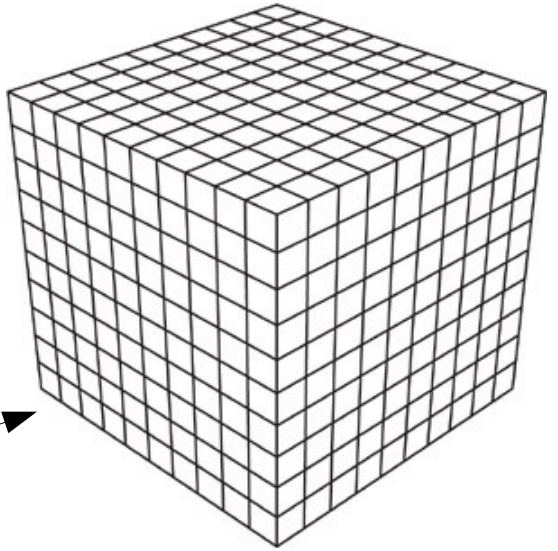
s.t.

$$\tilde{x} = g(x,u)$$

x: grid point, describes your system.
State-space potentially **high-dimensional**.

`old solution':
high-dimensional function on which **we interpolate**.

→ **N^d** points in ordinary discretization schemes.



→ Use-case for (adaptive) sparse grids.
→ Use-case for Gaussian Process regression.

How many is dimensions is high dimensions?

?

How many is dimensions is high dimensions?

Number of parameters (the dimension)	**Number of model runs** (at 10 points per dimension)	**Time for parameter study** (at 1 second per run)
1	10	10 sec
2	100	~ 1.6 min
3	1,000	~ 16 min
4	10,000	~ 2.7 hours
5	100,000	~ 1.1 days
6	1,000,000	~ 1.6 weeks
…	…	…
20	1e20	3 trillion years (240x age of the universe)

How many is dimensions is high dimensions?

Number of parameters (the dimension)	**Number of model runs** (at 10 points per dimension)	**Time for parameter study** (at 1 second per run)
1	10	10 sec
2	100	~ 1.6 min
3	1,000	~ 16 min
4	10,000	~ 2.7 hours
5	100,000	~ 1.1 days
6	1,000,000	~ 1.6 weeks
…	…	…
20	1e20	3 trillion years (240x age of the universe)

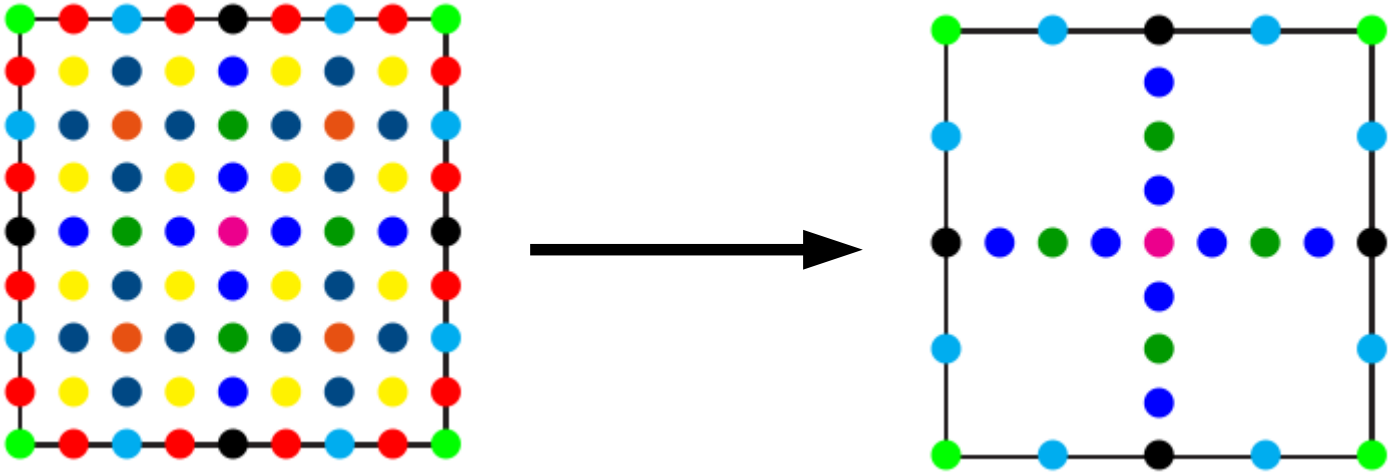
Dimension reduction
Exploit symmetries, e.g., via the active subspace method

Deal with #Points
Adaptive Sparse Grids

High-performance computing
Reduces time to solution, but not the problem size

II. From Full Grids to Sparse Grids

(see, e.g. Zenger (1991), Bungartz & Griebel (2004), Garcke (2012), Pflüger (2010),...)



Cartesian Grid

Sparse Grid

Interpolation on a Full Grid

-Consider a **1-dimensional function** $f : \Omega \to \mathbb{R}$ **on [0,1]**

-In numerical simulations:
f might be expensive to evaluate! (solve **PDE**s/system of **non-linear Eqs**.)
But: need to be able to evaluate *f* at arbitrary points using a numerical code

-Construct an interpolant ***u*** of ***f*** $f(\vec{x}) \approx u(\vec{x}) := \sum_i \alpha_i \varphi_i(\vec{x})$

-With suitable basis functions: $\varphi_i(\vec{x})$
and coefficients: α_i

- For simplicity: focus on case where $f|_{\partial\Omega} = 0$

Basis Functions

-Hierarchical basis based on **hat functions**

$$\phi(x) = \begin{cases} 1 - |x| & \text{if } x \in [-1, 1] \\ 0 & \text{else} \end{cases}$$

-Used to generate a **family of basis functions** $\phi_{l,i}$ having support $[x_{l,i} - h_l, x_{l,i} + h_l]$ by **dilation** and **translation**

$$\phi_{l,i}(x) := \phi\left(\frac{x - i \cdot h_l}{h_l}\right)$$

Hierarchical Increment Spaces

Hierarchical increment spaces:

$$W_l := \operatorname{span}\{\phi_{l,i} : i \in I_l\}$$

with the **index set**

$$I_l = \{i \in \mathbb{N}, 1 \leq i \leq 2^l - 1, i \text{ odd}\}$$

The corresponding **function space**:

$$V_l = \bigoplus_{k \leq l} W_k$$

The **1d-interpolant:**

$$f(x) \approx u(x) = \sum_{k=1}^{l} \sum_{i \in I_k} \alpha_{k,i} \phi_{k,i}(x)$$

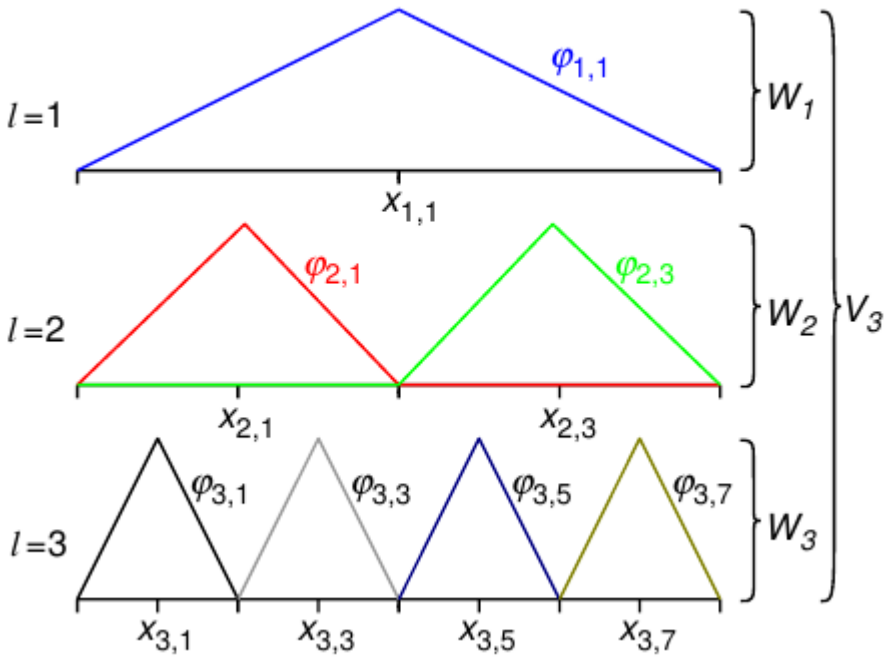


Fig.: **1-d basis functions** $\phi_{l,i}$ and the corresponding **grid points** up level ***l* = 3** in the hierarchical basis.

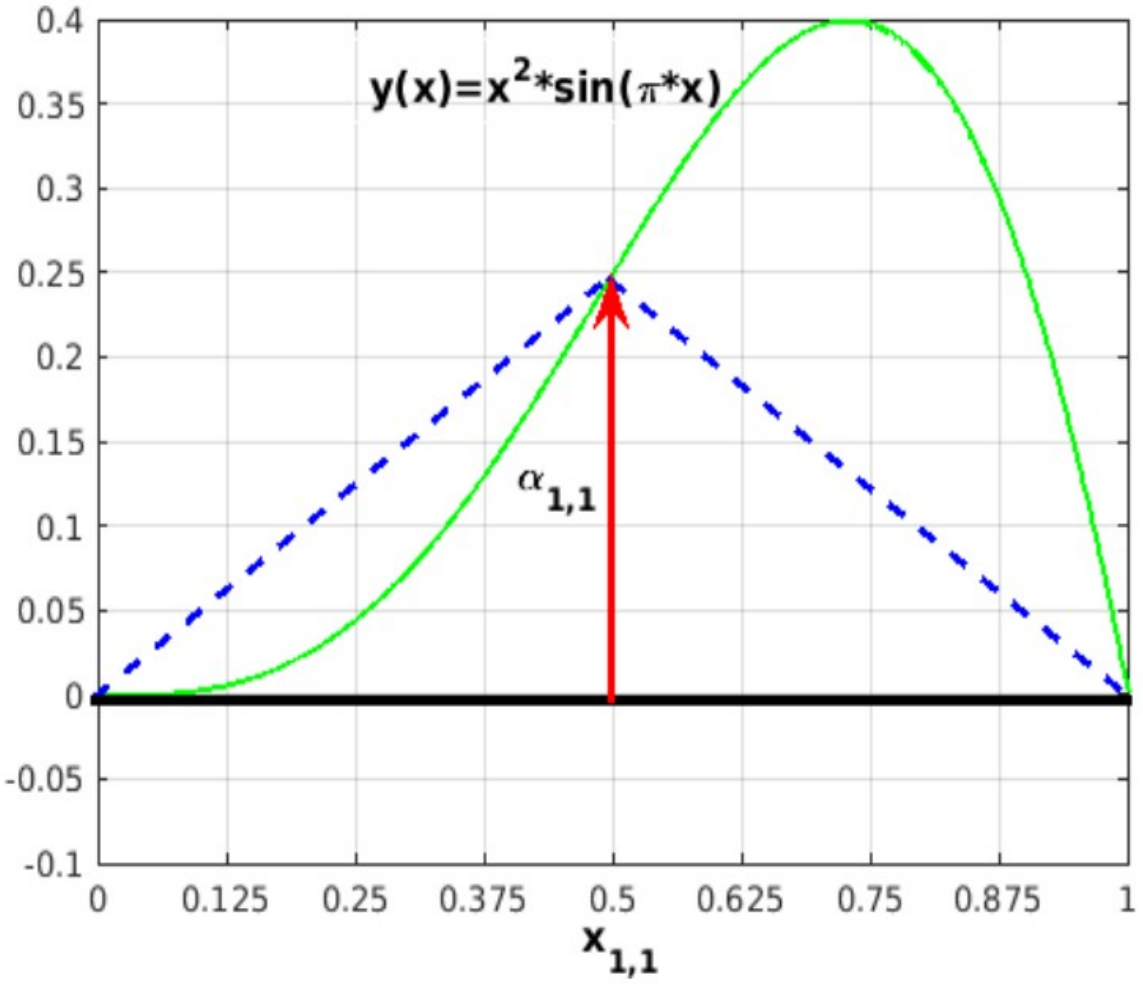
Note: supports of all basis functions of W_k mutually disjoint!

Piecewise Linear Interpolation: Level I

Coefficients:
hierarchical surpluses

They correct the interpolant of level *l*-1 at $\vec{x}_{l,i}$ to the actual value of $f(\vec{x}_{l,i})$

Nested structure:
Evaluate function only at points that are unique to the new level.

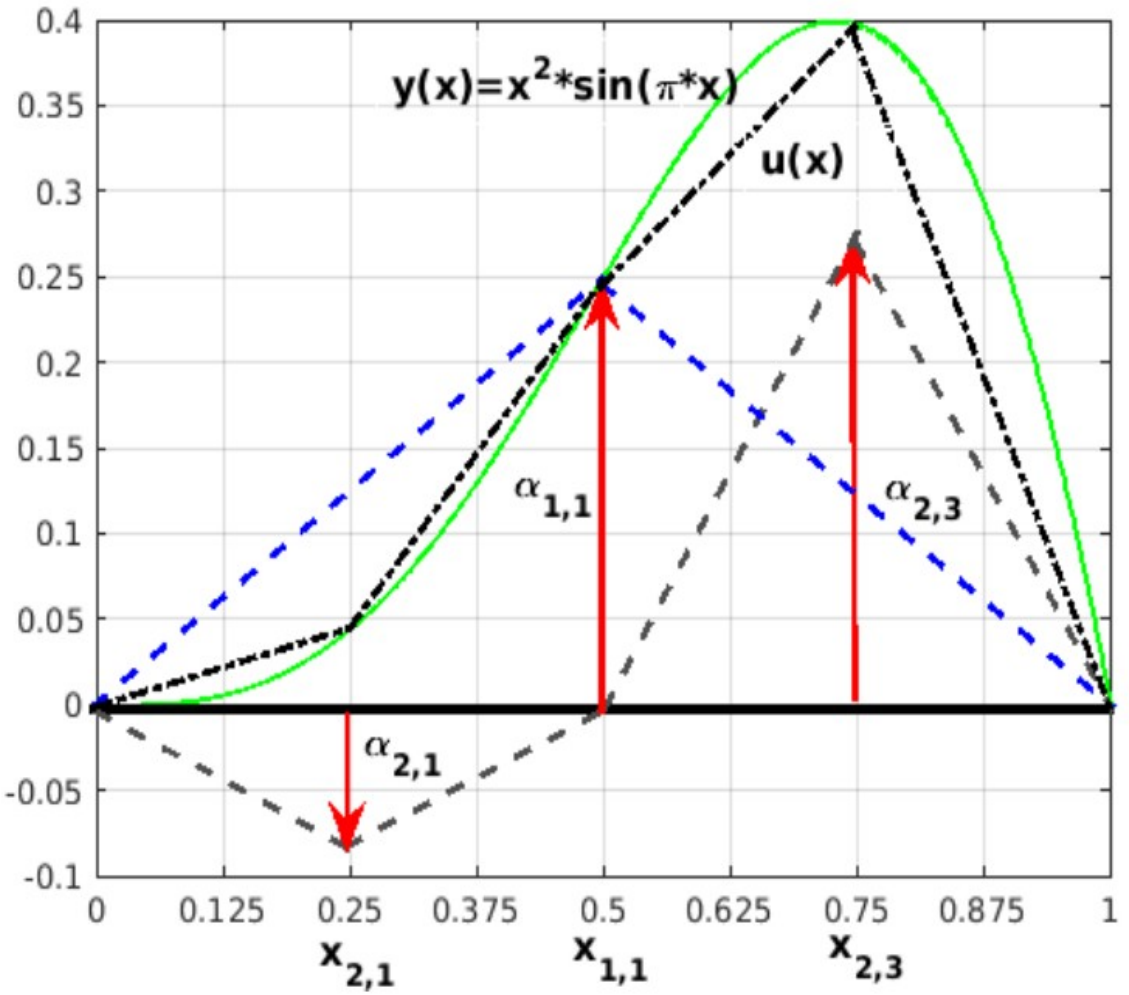


Piecewise Linear Interpolation: Level II

Coefficients:
hierarchical surpluses

They correct the interpolant of level *l*-1 at $\vec{x}_{l,i}$ to the actual value of $f(\vec{x}_{l,i})$

Nested structure:
Evaluate function only at points that are unique to the new level.

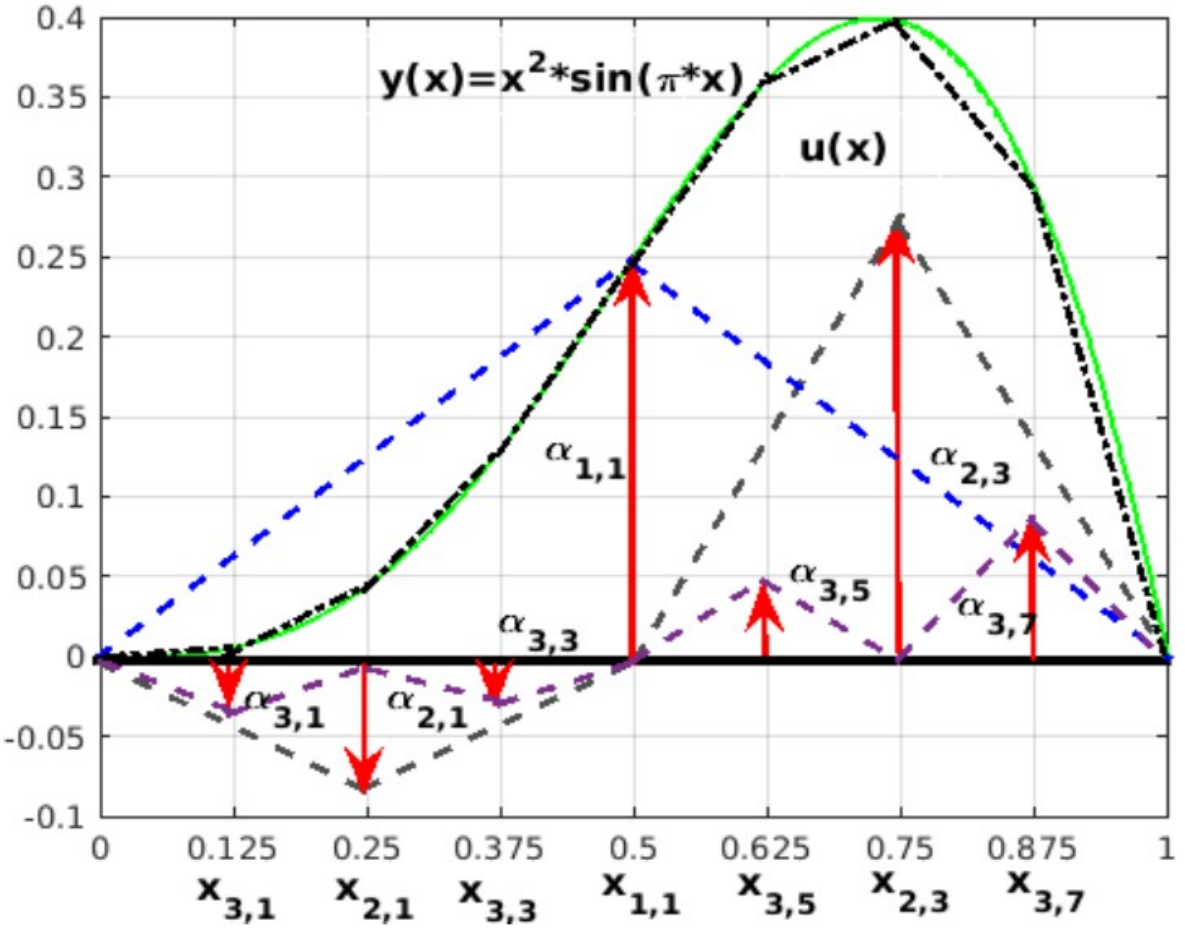


Piecewise Linear Interpolation: Level III

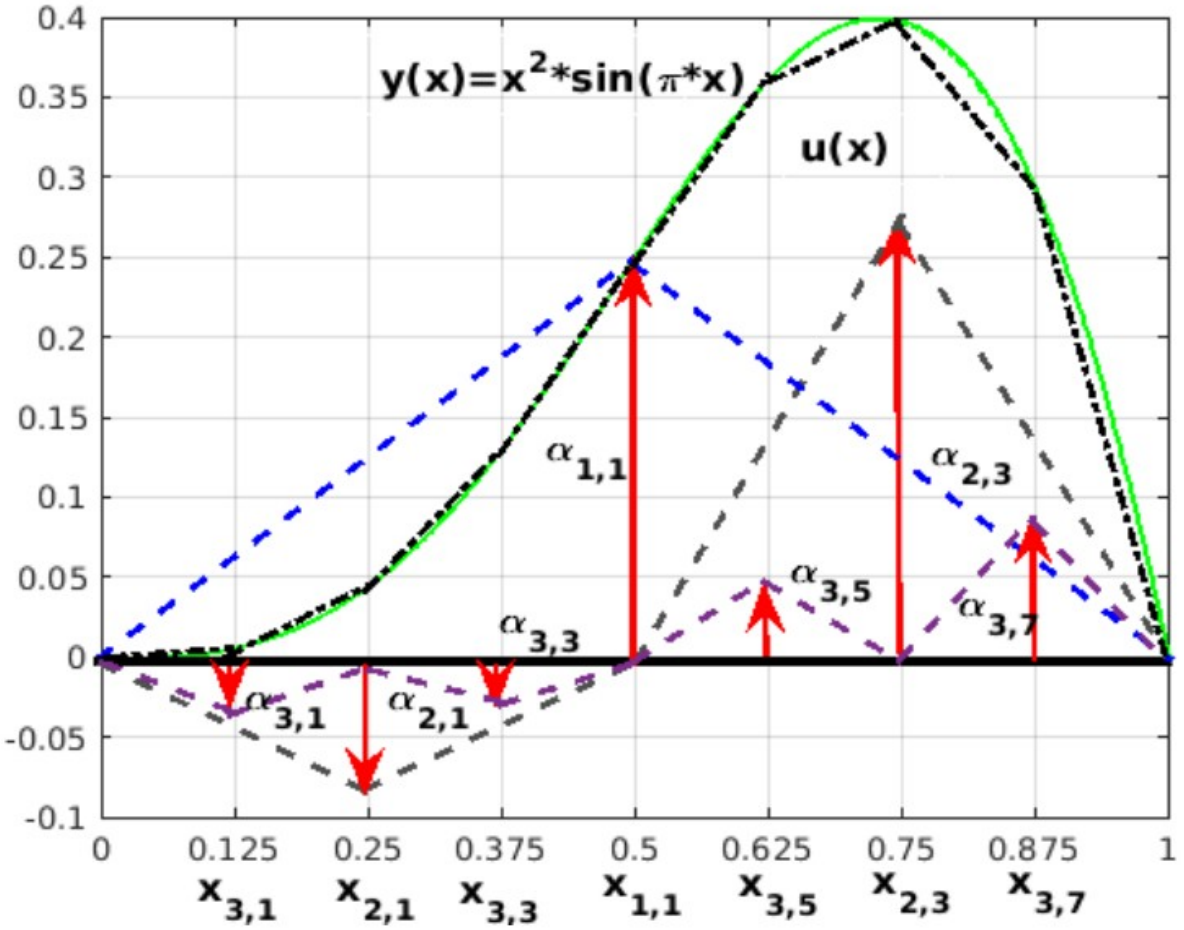
Coefficients:
hierarchical surpluses

They correct the interpolant of level *l*-1 at $\vec{x}_{l,i}$ to the actual value of $f(\vec{x}_{l,i})$

Nested structure:
Evaluate function only at points that are unique to the new level.



MOVIE


y(x)=x²*sin(π*x)
u(x)
α1,1
α2,3
α3,5
α3,7
α3,3
α3,1
α2,1
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
-0.05
-0.1
0
0.125
0.25
0.375
0.5
0.625
0.75
0.875
1
x3,1
x2,1
x3,3
x1,1
x3,5
x2,3
x3,7

Non-zero Boundary Conditions

Want to be able to handle non-zero boundaries:

$$f|_{\partial\Omega} \neq 0$$

If we add naively points at boundaries, $\mathbf{3^d}$ support nodes will be added.

Numerically cheapest way:
Modify basis functions and interpolate towards boundary.
Various choices possible!

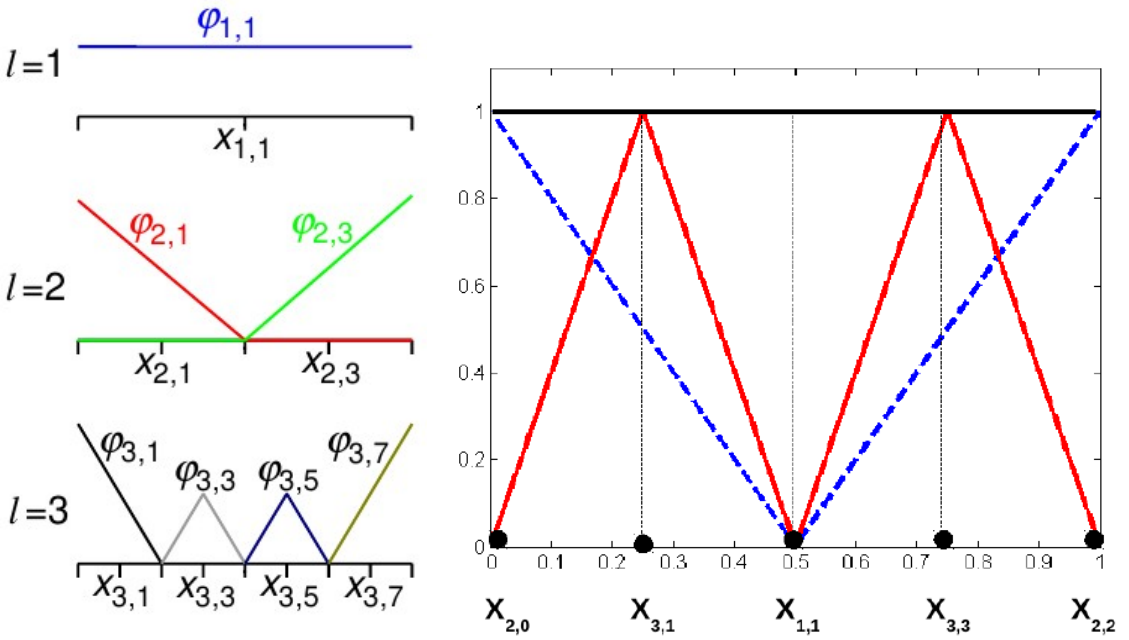
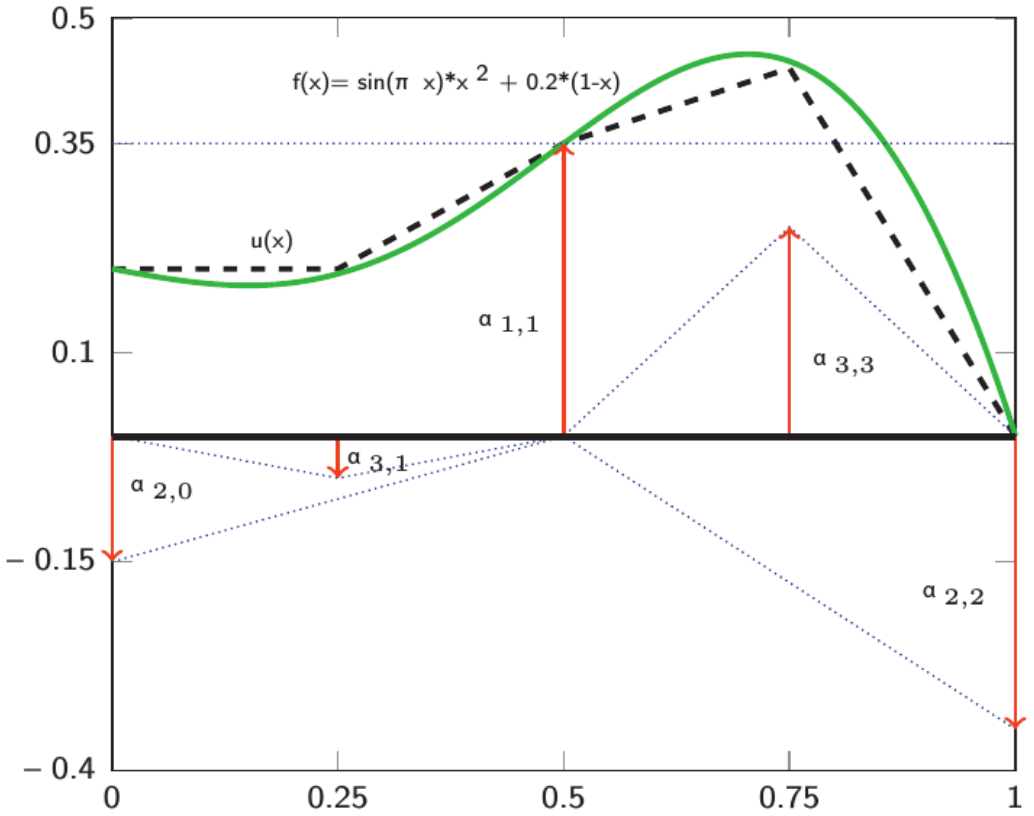


Fig.: Example of modified 1d-basis functions According to Pflüger (2010), which are extrapolating towards the boundary (**left**). They are constant on level 1 and **"folded-up"** if adjacent to the boundary on all other levels. **Right**: "Modified" hat basis.

Piecewise Linear Interpolation: 3 Levels



- Construction of u (x) interpolating f (x) with hierarchical linear basis functions of levels 1, 2, and 3.
- The hierarchical surpluses that belong to the respective basis functions are indicated by arrows.
- They are simply the difference between the function values at the current and the previous interpolation levels.

Examples for more basis functions

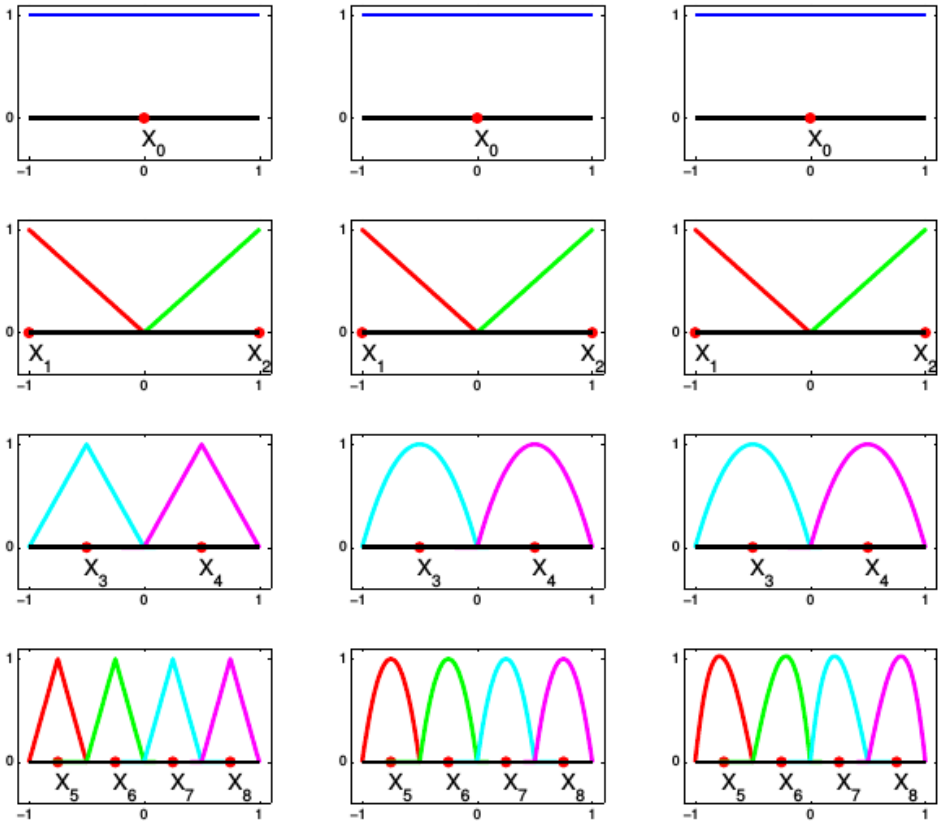


Figure 1: Local polynomial points (*rule_localp*) and functions, left to right: linear, quadratic, and cubic functions.

Fig. taken from Tasmanian User Manual

Some definitions & notation

(see, e.g. Zenger (1991), Bungartz & Griebel (2004), Garcke (2012), Pflüger (2010),...)

- We will focus on the domain $\Omega = [0,1]^d$

  d: dimensionality; **other domains: rescale**

- introduce **multi-indices**:

  grid refinement level: $\vec{l} = (l_1, ..., l_d) \in \mathbb{N}^d$

  spatial position: $\vec{i} = (i_1, ..., i_d) \in \mathbb{N}^d$

- Discrete, (Cartesian) full grid $\Omega_{\vec{l}}$ on Ω

- Grid $\Omega_{\vec{l}}$ consists of points: $\vec{x}_{\vec{l},\vec{i}} := (x_{l_1,i_1}, ..., x_{l_d,i_d})$

Where $x_{l_t,i_t} := i_t \cdot h_{l_t} = i_t \cdot 2^{-l_t}$ and $i_t \in \{0, 1, ..., 2^{l_t}\}$

Multi-Dimensional Interpolant

Extension to multi-d by a **tensor-product construction**:

Multi-d basis: $$\phi_{\vec{l},\vec{i}}(\vec{x}) := \prod_{t=1}^{d} \phi_{l_t,i_t}(x_t)$$

Index set: $$I_{\vec{l}} := \{\vec{i} : 1 \leq i_t \leq 2^{l_t} - 1, i_t \text{ odd}, 1 \leq t \leq d\}$$

Hierarchical increments: $$W_{\vec{l}} := \text{span}\{\phi_{\vec{l},\vec{i}} : \vec{i} \in I_{\vec{l}}\}$$

Multi-d interpolant:

$$\longrightarrow \quad f(\vec{x}) \approx u(\vec{x}) = \sum_{|l|_\infty \leq n} \sum_{\vec{i} \in I_{\vec{l}}} \alpha_{\vec{l},\vec{i}} \cdot \phi_{\vec{l},\vec{i}}(\vec{x})$$

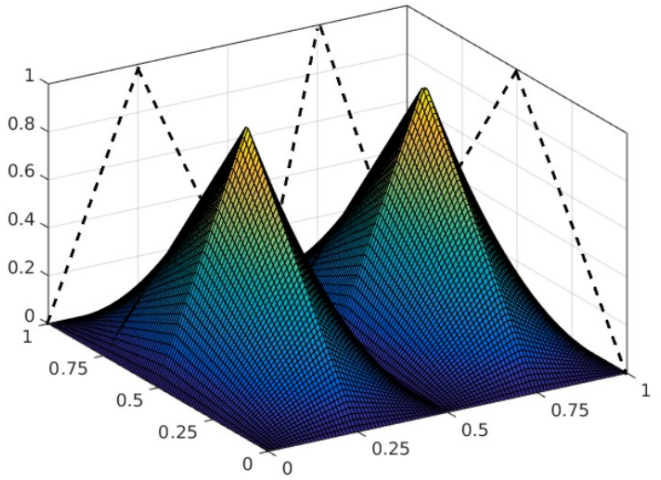


Fig.: Basis functions of the **subspace $W_{2,1}$**

Why reality bites...

Interpolant consists of ***O(2^{nd})*** grid points

For **sufficiently smooth *f*** and its interpolant ***u***, we obtain an asymptotic error decay of $\| f(\vec{x}) - u(\vec{x}) \|_{L_2} \in \mathcal{O}(h_n^2)$

But at the cost of $\mathcal{O}(h_n^{-d}) = \mathcal{O}(2^{nd})$

function evaluations → **“curse of dimensionality”**

Hard to handle more than 4 dimensions numerically

⟶ e.g. d=10, n = 4, 15 points/d, **5.8 x 10^{11}** grid points

`Breaking' the curse of dimensionality (I)

Question: **"can we construct discrete approximation spaces that are better in the sense that the same number of invested grid points leads to a higher order of accuracy?"** **YES √**

(see, e.g. Bungartz & Griebel (2004))

→ If **second mixed derivatives are bounded**, then the hierarchical **surpluses decay rapidly** with increasing approximation level.

$$|\alpha_{\vec{l},\vec{i}}| = \mathcal{O}\left(2^{-2|\vec{l}|_1}\right)$$

`Breaking' the curse of dimensionality (II)

(see, e.g. Bungartz & Griebel (2004))

Strategy of constructing sparse grid: leave out those subspaces from full grid that only contribute little to the overall interpolant.

Optimization w.r.t. number of degrees of freedom (grid points) and the approximation accuracy leads to the sparse grid space of level ***n***.

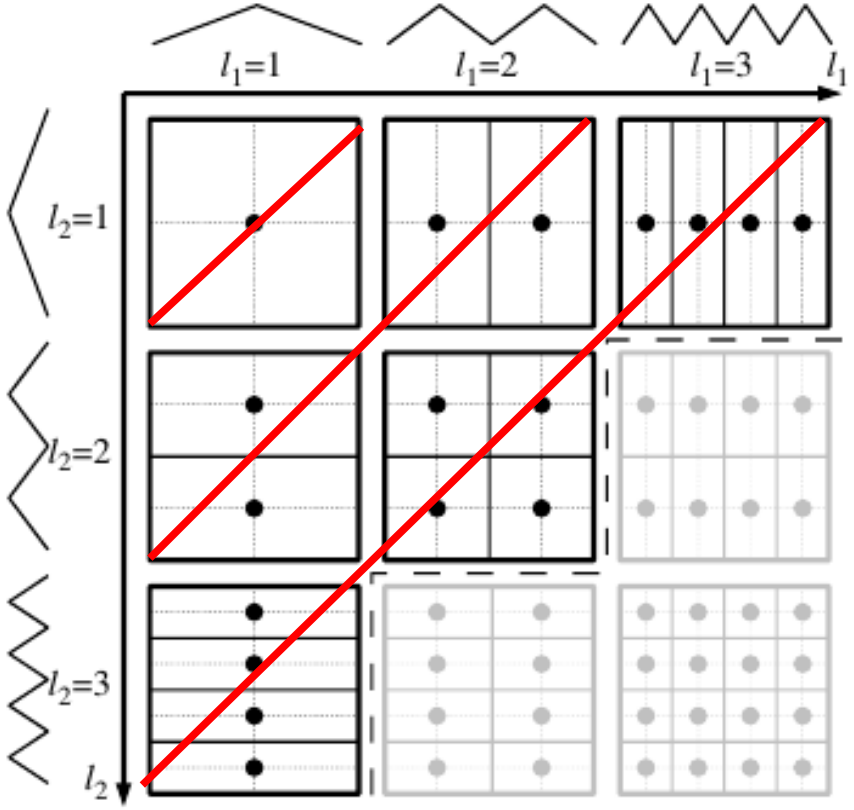
$$V_{0,n}^S := \bigoplus_{|\vec{l}|_1 \leq n+d-1} W_{\vec{l}}$$

Interpolant: $$f_{0,n}^S(\vec{x}) \approx u(\vec{x}) = \sum_{|l|_1 \leq n+d-1} \sum_{\vec{i} \in I_{\vec{l}}} \alpha_{\vec{l},\vec{i}} \cdot \phi_{\vec{l},\vec{i}}(\vec{x})$$

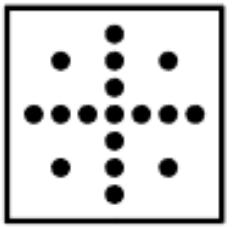
grid points: $\mathcal{O}\left(h_n^{-1} \cdot \left(\log(h_n^{-1})\right)^{d-1}\right) = \mathcal{O}\left(2^n \cdot n^{d-1}\right)$ << $\mathcal{O}\left(h_n^{-d}\right) = \mathcal{O}\left(2^{nd}\right)$

Accuracy of the interpolant: $\mathcal{O}\left(h_n^2 \cdot \log(h_n^{-1})^{d-1}\right)$ vs. $\mathcal{O}\left(h_n^2\right)$

Sparse grid construction in 2D



$$V_{0,n}^S := \bigoplus_{|\vec{l}|_1 \leq n+d-1} W_{\vec{l}}$$



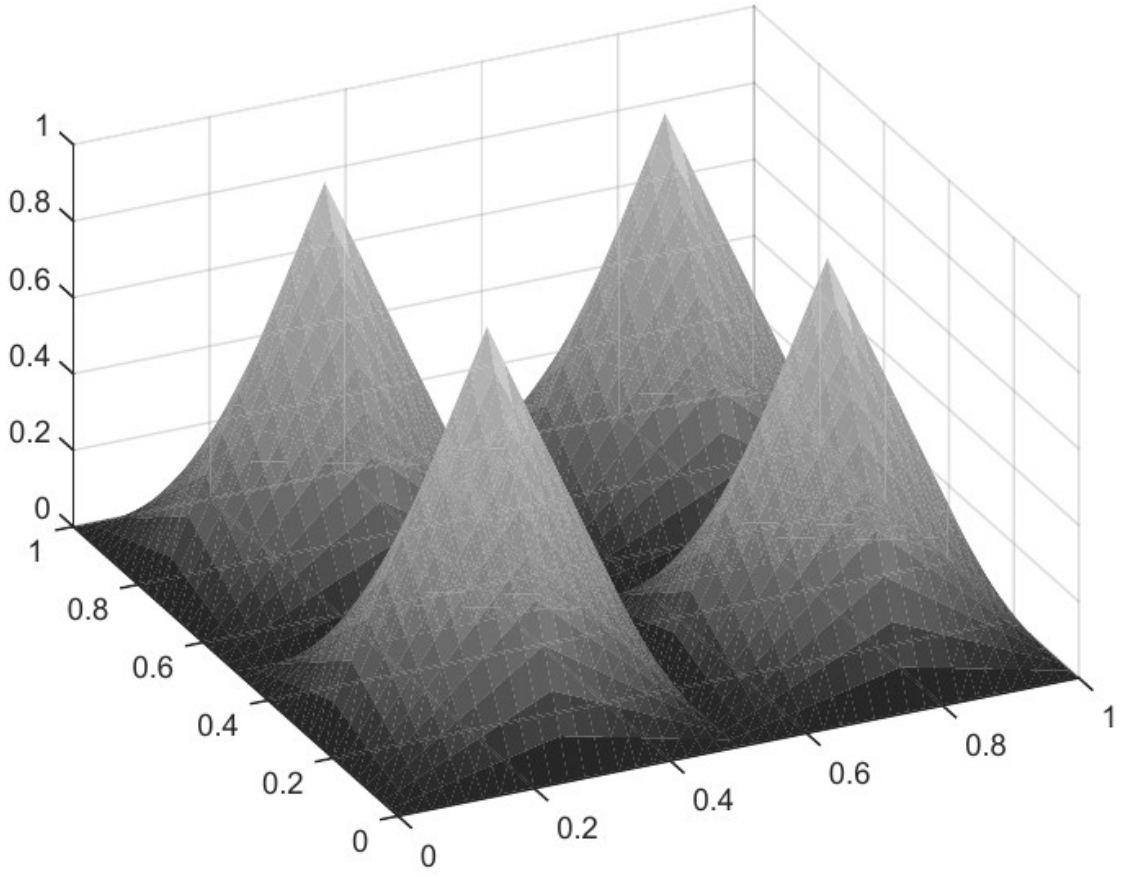
V_3

Sparse grid: 17 pt.
Full grid : 49 pt.

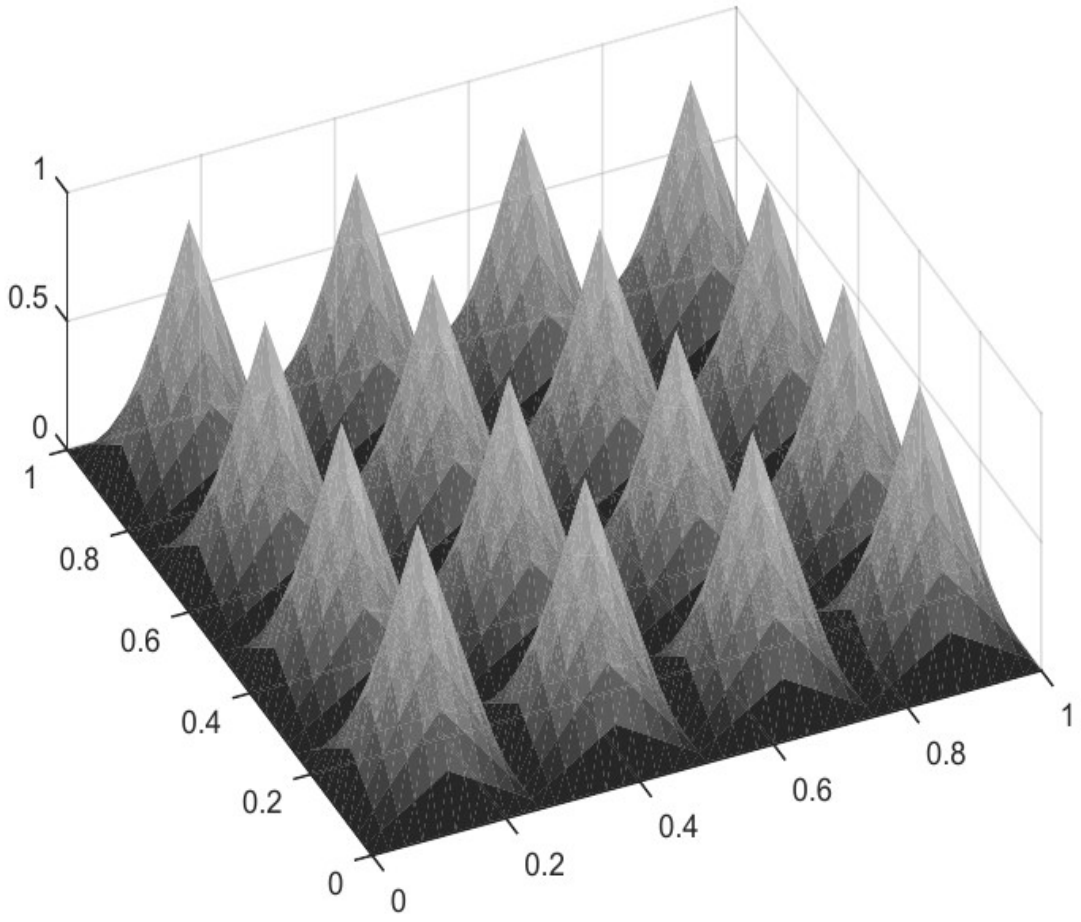
Fig.: Two-dimensional subspaces $\mathbf{W_l}$ up to $\boldsymbol{l}$=3 (h_3 = 1/8) in each dimension.

The optimal a priori selection of subspaces in shown in black **(left)** and the Corresponding sparse grid of level n = 3 **(right)**.
For the full grid, the gray subspaces have to be used as well.

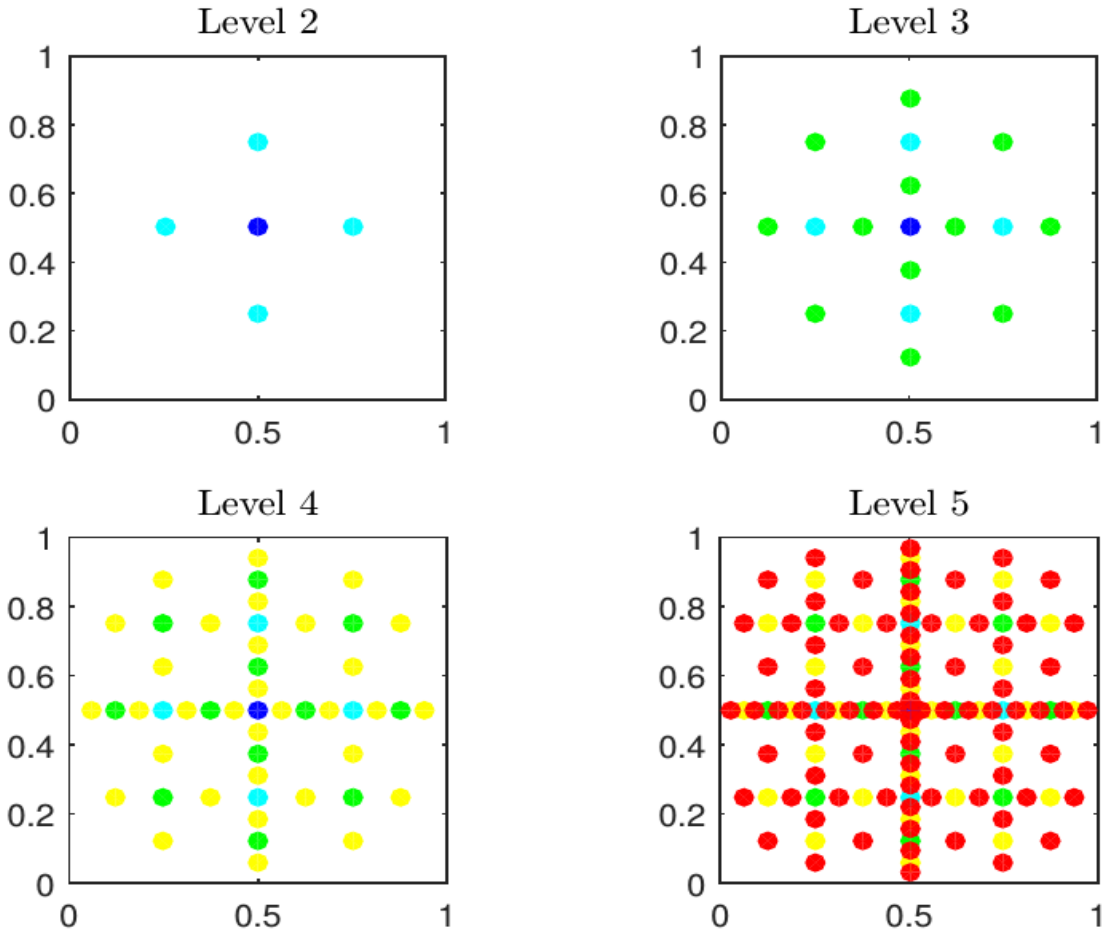
Basis Functions of $W_{2,2}$ — Included in V_3



Basis Functions of $W_{3,3}$ — not Included in V_3

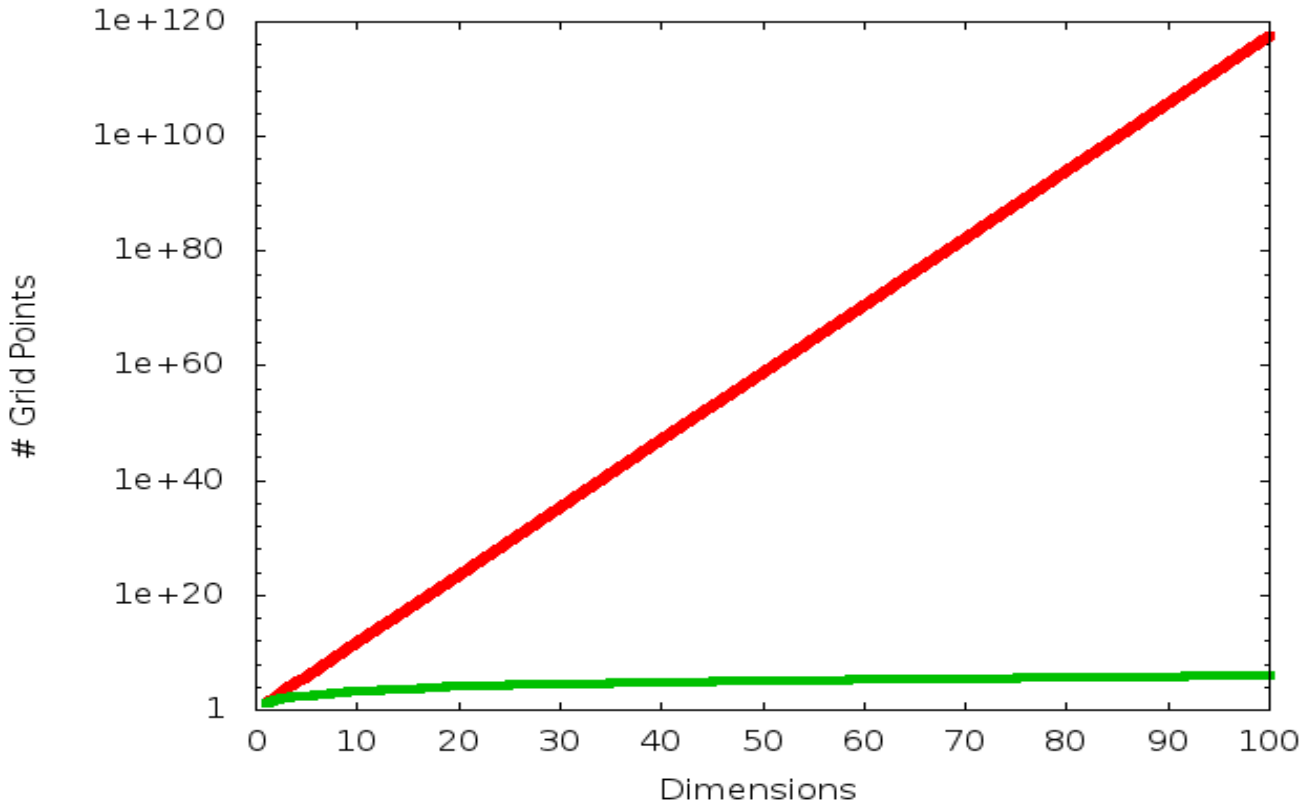


Sparse Grid of Increasing level



Grid Points

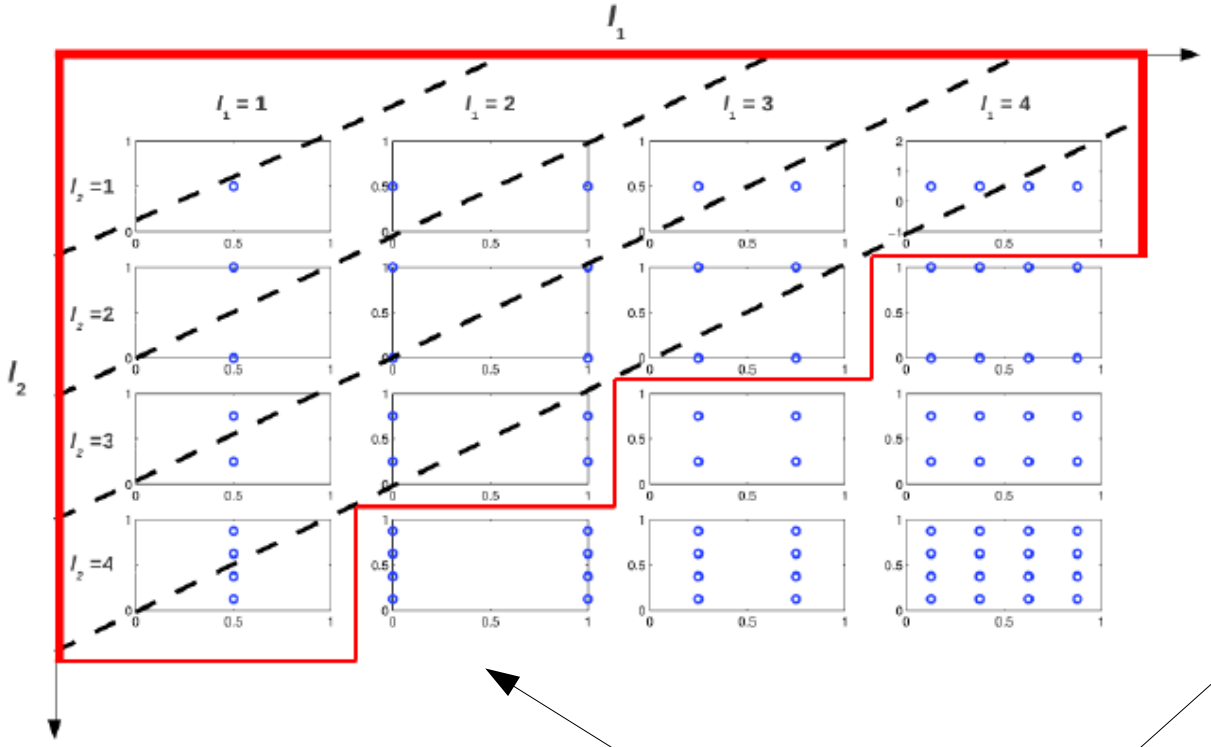
d	$\lvert V_n \rvert$	$\lvert V_{0,n}^S \rvert$
1	15	15
2	225	49
3	3375	111
4	50'625	209
5	759'375	351
10	$5.77 \cdot 10^{11}$	2'001
15	$4.37 \cdot 10^{17}$	5'951
20	$3.33 \cdot 10^{23}$	13'201
30	$1.92 \cdot 10^{35}$	41'601
40	$1.11 \cdot 10^{47}$	95'201
50	$6.38 \cdot 10^{58}$	182'001
100	>Googol	1'394'001

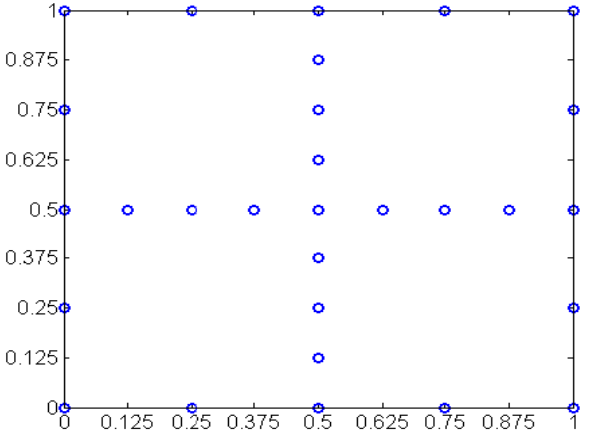


Tab.: Number of grid points for several types of sparse grids of level $n = 4$.
Middle: Full grid; **right:** **classical sparse grid** with **no points at the boundaries.**
Fig.: Number of grid points growing with dimension (full grid vs. sparse grid).

Sparse Grid with non-zero boundaries

(see, e.g. Bungartz & Griebel (2004))





Sparse grid: 29 pt.
Full grid : 81 pt.

$$V_n^{S,CC} := \bigoplus_{|\vec{l}|_1 \le n+d-1} W_{\vec{l}}.$$

Number of Grid Points

TABLE VIII

NUMBER OF GRID POINTS FOR SEVERAL DIFFERENT GRID TYPES OF LEVEL 4[a]

d	$\lvert V_4 \rvert$	$\lvert V_4^S \rvert$	$\lvert V_4^{S,\mathrm{NZ}} \rvert$	$\lvert V_5^{S,\mathrm{NZ}} \rvert$	$\lvert V_6^{S,\mathrm{NZ}} \rvert$
1	15	15	9	17	33
2	225	49	29	65	145
3	3,375	111	69	177	441
4	50,625	209	137	401	1,105
5	759,375	351	241	801	2,433
10	$5.77 \cdot 10^{11}$	2,001	1,581	8,801	41,265
20	$3.33 \cdot 10^{23}$	13,201	11,561	120,401	1,018,129
50	$6.38 \cdot 10^{58}$	182,001	171,901	4,352,001	88,362,321
100	>Googol	1,394,001	1,353,801	68,074,001	$2.74 \cdot 10^{9}$

[a]The first column is dimension; the second column is the full grid; the third column is the SG with no points at the boundaries; the fourth to sixth columns are the SG with nonzero boundaries (levels four to six).

Hierarchical Integration

High-dimensional integration easy with sparse grids, e.g. compute expectations
Let's assume uniform probability density:

$$\mathbb{E}\left[u(\vec{x})\right] = \sum_{|l|_1 \leq n+d-1} \sum_{\vec{i} \in I_{\vec{l}}} \alpha_{\vec{l},\vec{i}} \int_{\Omega} \phi_{\vec{l},\vec{i}}(\vec{x}) d\vec{x}$$

The one-dimensional integral can now be computed analytically (Ma & Zabras (2008))

$$\int_0^1 \phi_{l,i}(x)\, dx = \begin{cases} 1, & \text{if } l = 1 \\ \frac{1}{4} & \text{if } l = 2 \\ \\ 2^{1-l} & \text{else} \end{cases}$$

Note that this result is independent of the location of the interpolant to dilation
And translation properties of the hierarchical basis functions.
→ **Multi-d integrals are therefore again products of 1-d integrals.**

We denote $\int_{\Omega} \phi_{l,i}(\vec{x})\, d\vec{x} = J_{\vec{l},\vec{i}}$

$$\longrightarrow \mathbb{E}\left[u(\vec{x})\right] = \sum_{|l|_1 \leq n+d-1} \sum_{\vec{i} \in I_{\vec{l}}} \alpha_{\vec{l},\vec{i}} \cdot J_{\vec{l},\vec{i}}$$

where were Sparse Grids used?

For a review, see, e.g. Bungartz & Griebel (2004)

Sparse grid methods date back to Smolyak(1963)

BUT: Smolyak used global polynomials!

So far, methods applied to:

-High-dimensional integration

e.g. Gerstner & Griebel (1998), Bungartz et al. (2003),...

-Interpolation

e.g. Barthelmann et al. (2000), Klimke & Wohlmut (2005),...

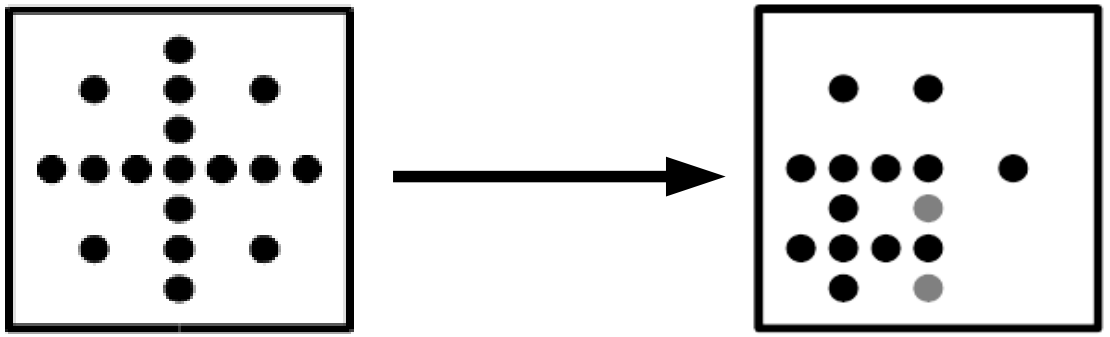
-Solution of PDEs

e.g. Zenger (1991), Griebel (1998),...

More fields of application: regressions, data mining, likelihood estimations, option pricing, data compression, dynamic economic models...

e.g. Kubler & Kruger (2004), Winschel & Kraetzig (2010), Judd et al. (2013) → **Smolyak; global basis functions**.

III. Adaptive Sparse Grids



Sketch of adaptive refinement

See, e.g. Ma & Zabaras (2008), Pflüger (2010), Bungartz (2003),..

-Surpluses should quickly decay to zero

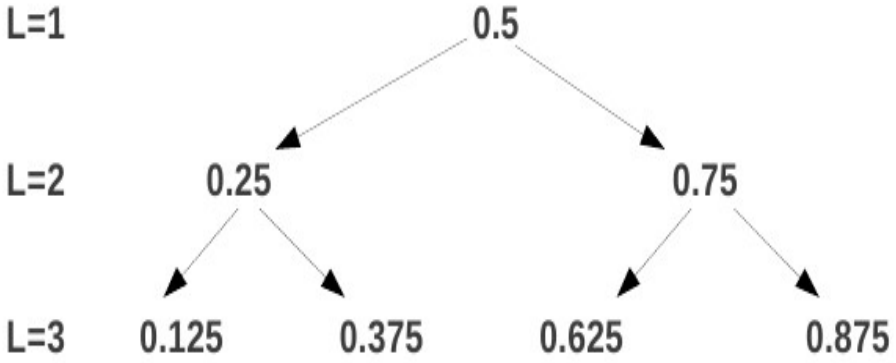
-Use hierarchical surplus as error indicator.

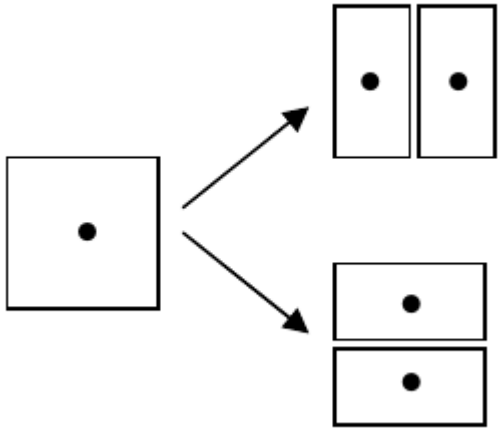
-**Automatically detect "discontinuity regions"** and adaptively refine the points in this region.

-Each grid point has ***2d*** neighbours

-**Add neighbour points, i.e. locally refine interpolation level from *l* to *l+1***

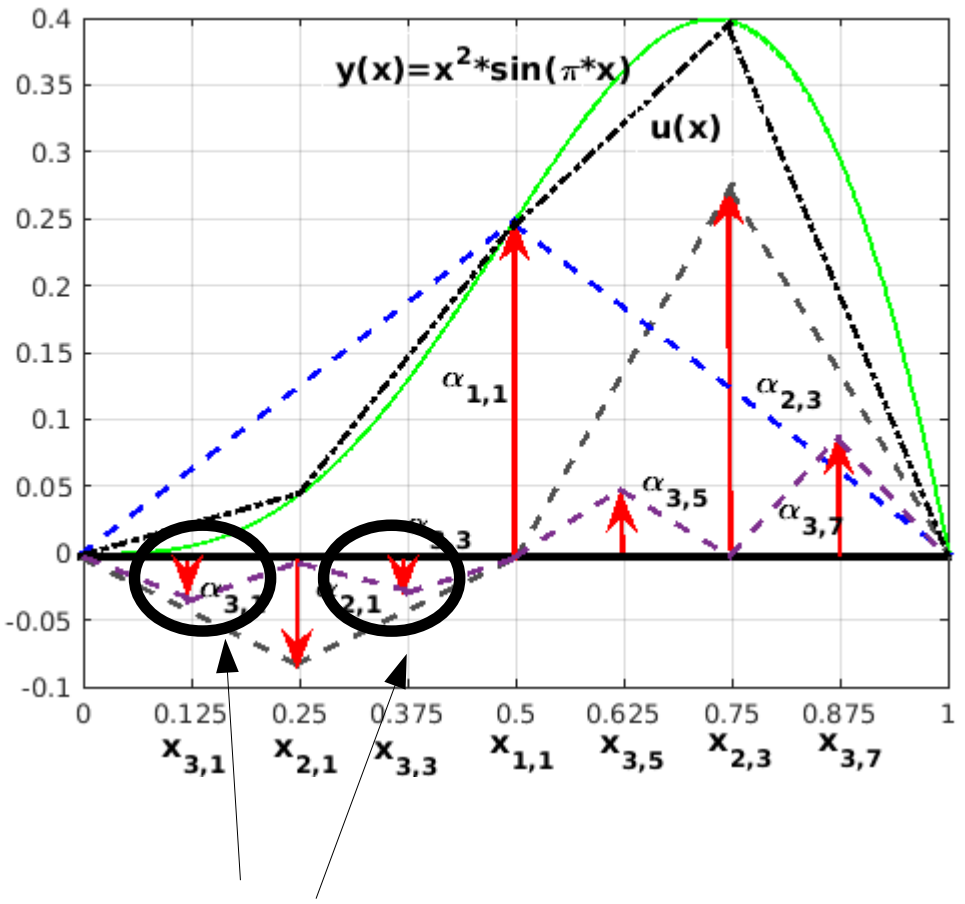
-Criterion: **e.g.** $|\alpha_{\vec{l},\vec{i}}| \geq \epsilon$





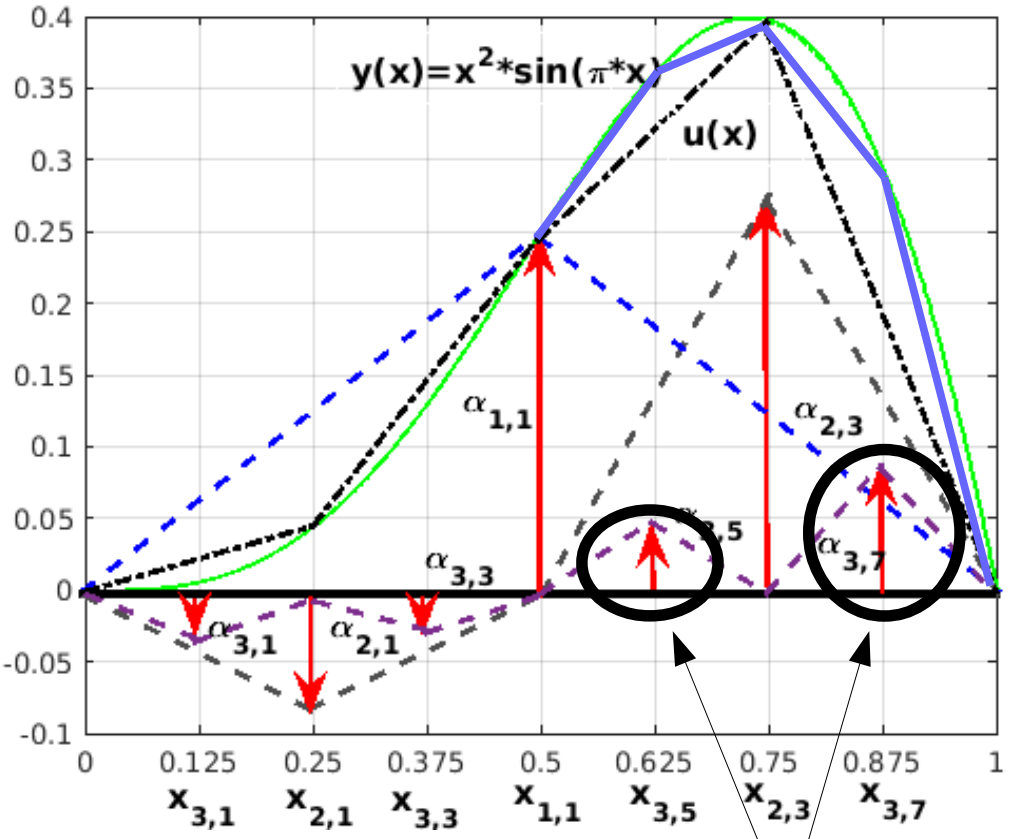
top panel: tree-like structure of sparse grid.
lower panel: locally refined sparse grid in 2D.

Example I



Small – below threshold

Example II



Add points – above threshold

Test in 1d

(See Genz (1984) for test functions)

Test function:

$$f(x) = \frac{1}{|0.5 - x^4| + 0.01}$$

Error both for full grid and adapt. sparse grid of $O(10^{-2})$.

Error measure:

→ 1000 random points from [0,1]

$$e = \max_{i=1,\ldots,1000} |f(\vec{x}_i) - u(\vec{x}_i)|$$

Full grid: **1023** points
Adaptive sparse grid: **109** points.

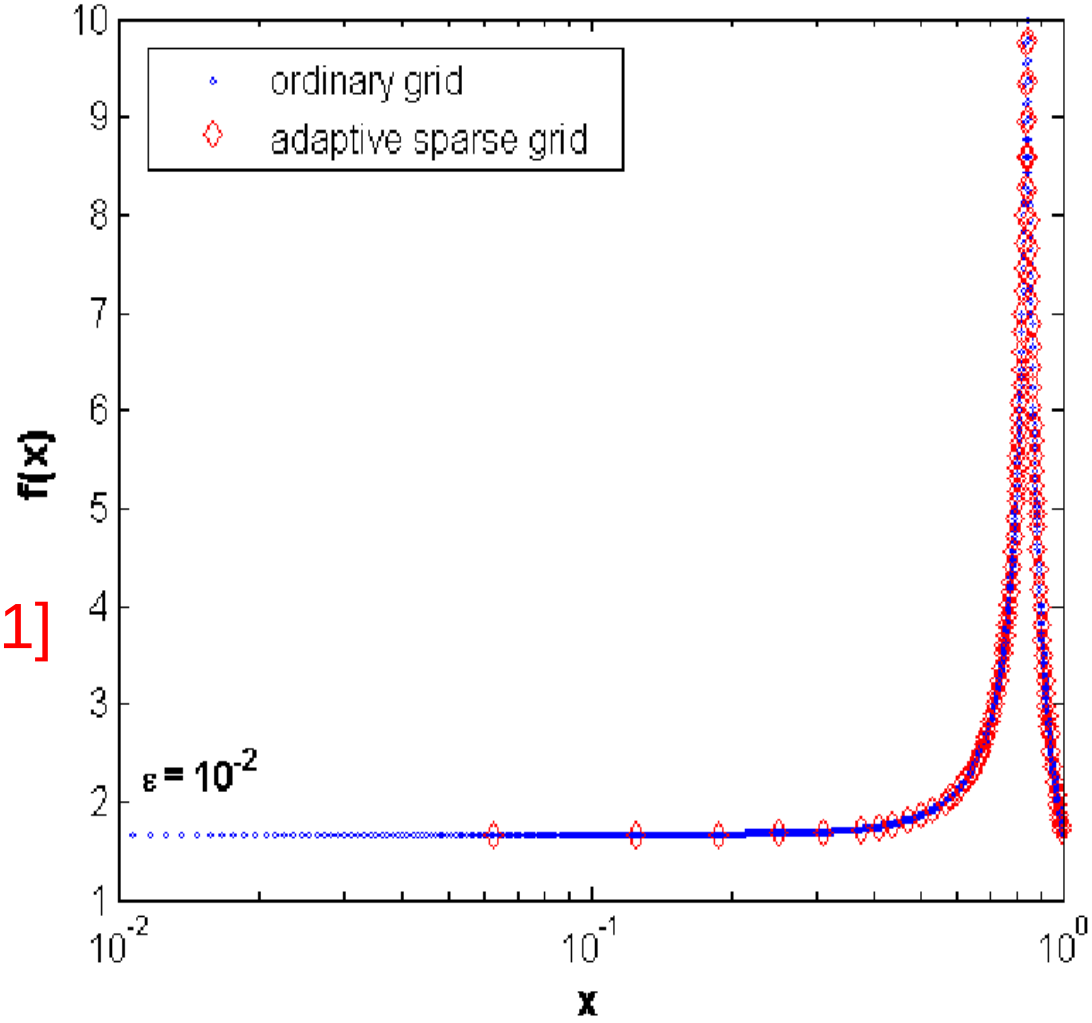


Fig.: Blue: Full grid;red: adaptive sparse grid.

Test in 2d

Test function: $$\frac{1}{|0.5 - x^4 - y^4| + 0.1}$$

Error: $O(10^{-2})$

Full grid:
→ $O(10^9)$ points

Sparse grid:
→ **311,297 points**

Adaptive sparse grid:
→ **4,411 points**

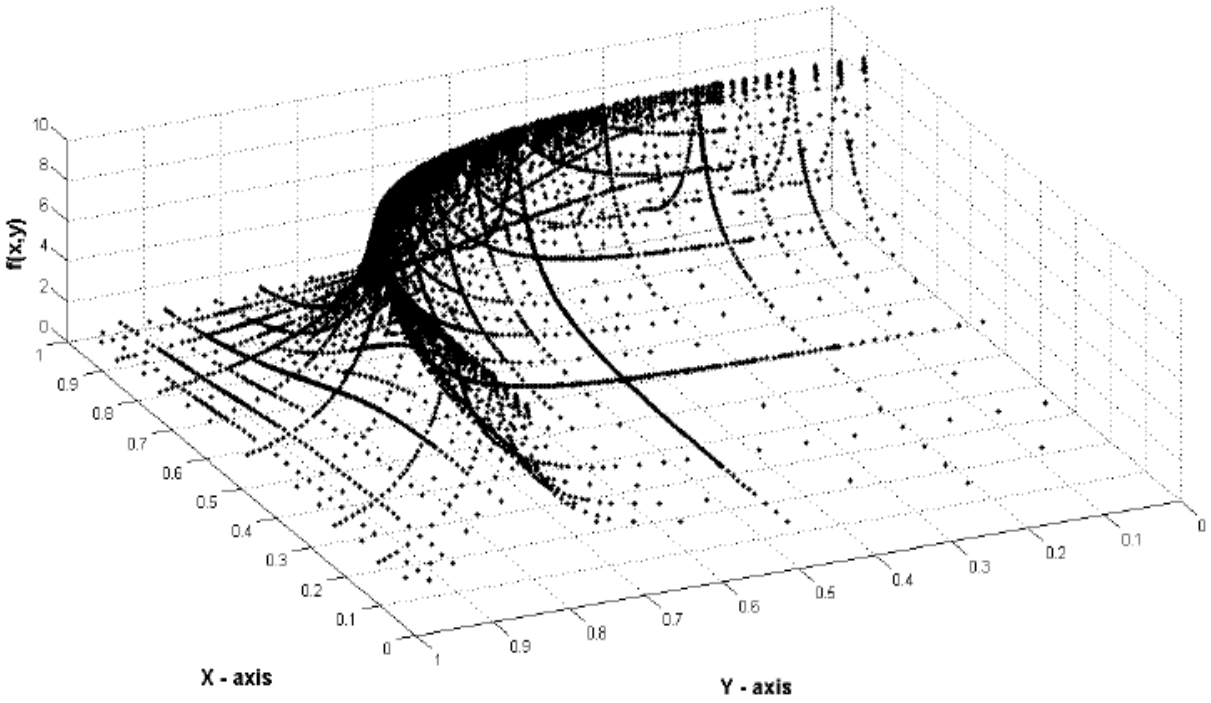


Fig.: 2d test function and its corresponding grid points after 15 refinement steps.

Movie

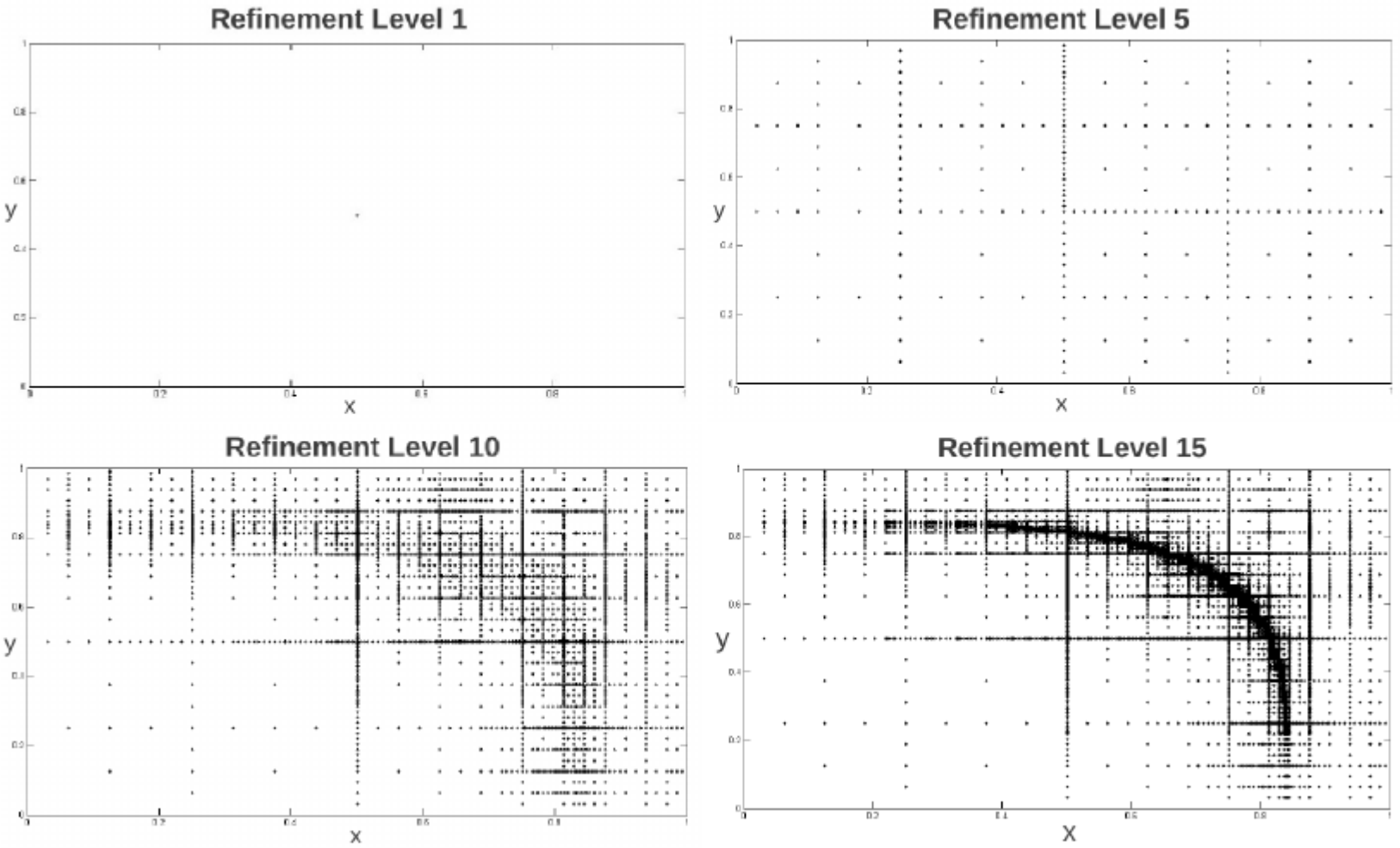
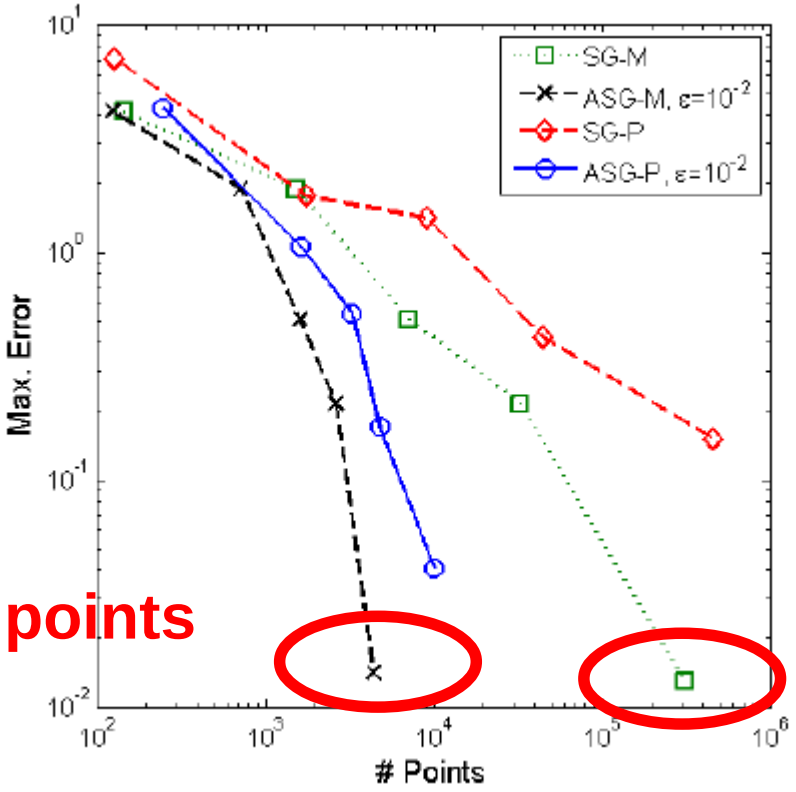


Fig.: Evolution of the adaptive sparse grid with a **threshold for refinement of 10^{-2}** . The refinement levels displayed are L = 1, 5, 10, 15.

Convergence

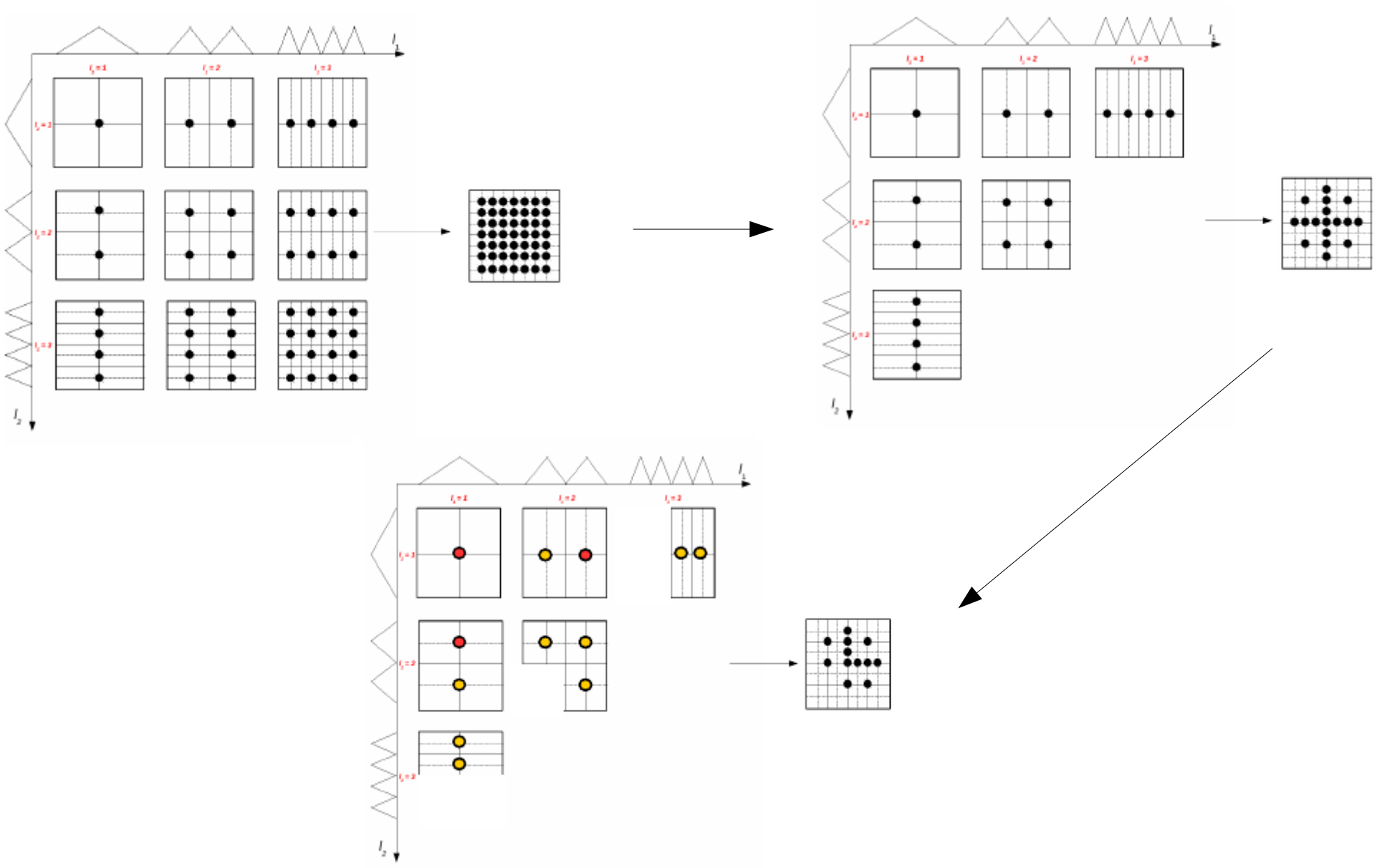


4,411 points

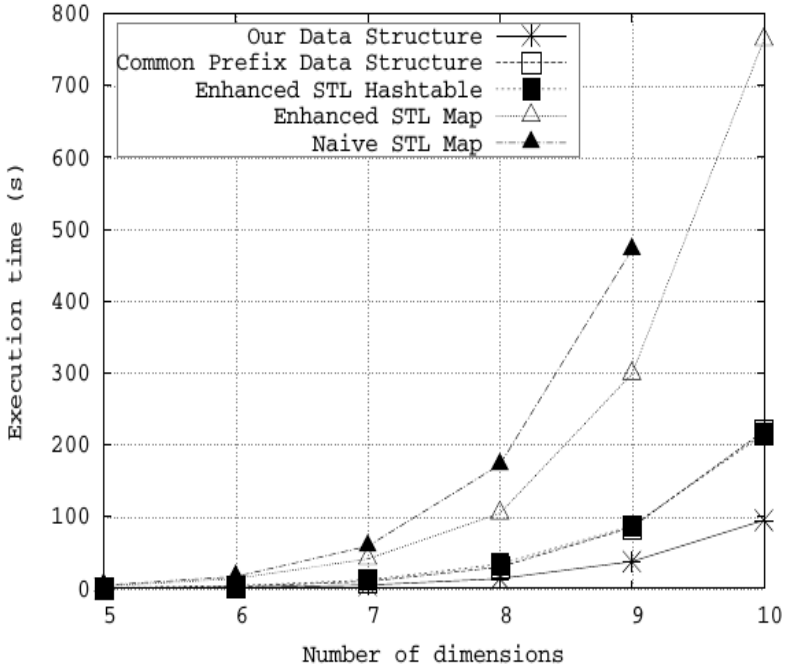
311,297 points

Fig.: Comparison of the interpolation error for **conventional and adaptive sparse grid interpolation** (two different adaptive sparse grid choices).

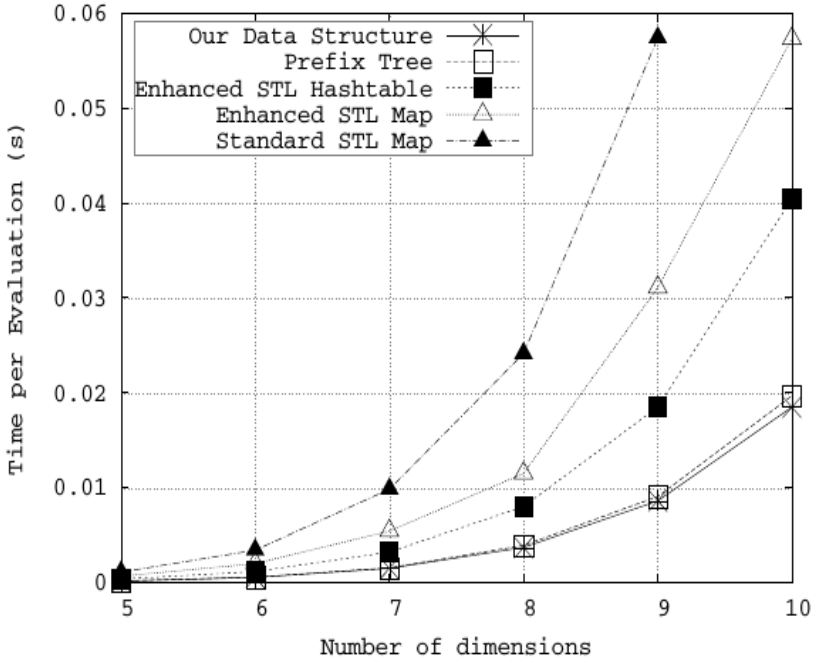
From Cartesian to adaptive sparse grids



Limitation of sparse grids: Execution times in higher dimension



(a) Runtime for sequential hierarchization.



(b) Runtime for sequential evaluation.

going to higher dimensions gets polynomially harder → we need parallel programming

Figs. From Murarasu et al. (2011)

Limitations of sparse grids (II)

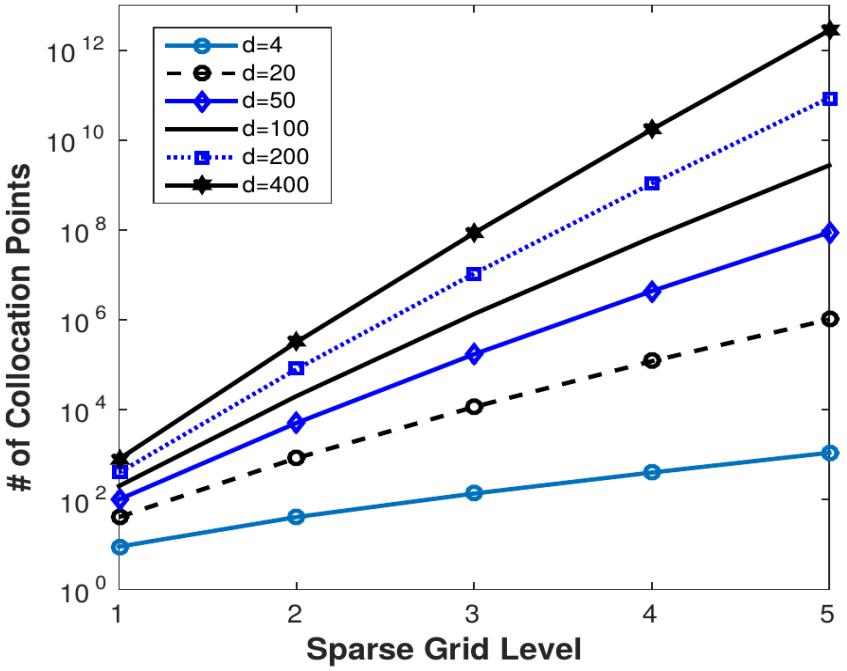


Fig.: classical sparse grids of varying dimension and increasing refinement level

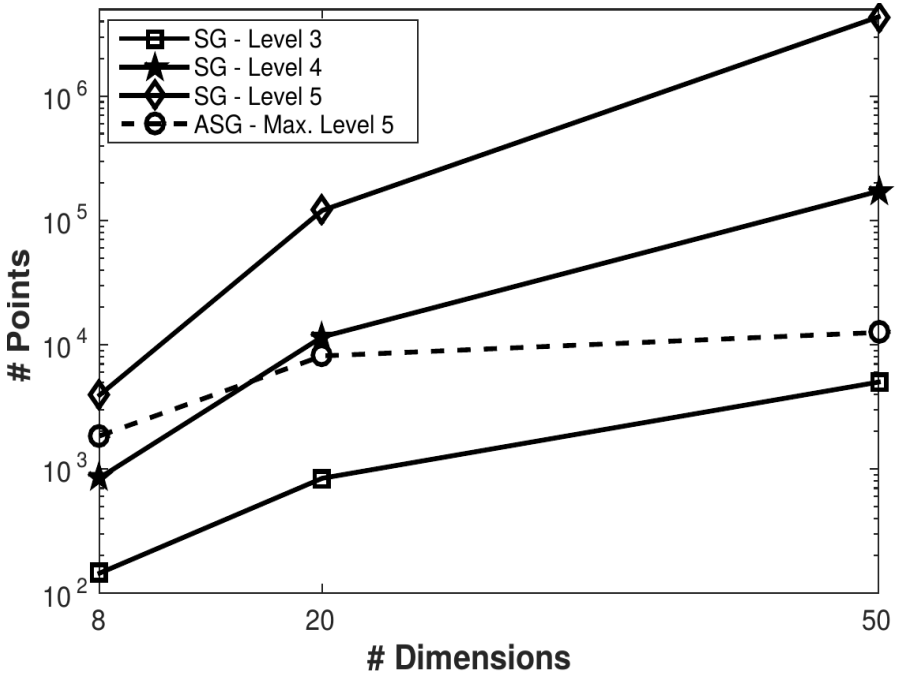


Fig.: IRBC model, solved both with classical sparse grids of varying dimension and increasing refinement level.

Major issue: a complex problem may require a high resolution in order to obtain a "reasonable" solution, i.e., a high sparse grid refinement level. For high-dimensional problems, the amount of points added to the sparse grid grow fast with the increasing level (still slower than exponential) but still make problems quickly intractable **(left panel)**. ASGs can alleviate this issue to some extend **(right panel)**.

Note – Install all libraries

- Note: I prepared you the packages with all the dependencies.

- in order to install, follow these steps:

1. log onto a Unix-based system, and go to your repository.
> cd Lecture_1/SparseGridCode

2. Install Tasmanian (SG library), SPINTERP (SG library), IPOPT, and PYIPOPT (optimizer)

> install_SG.sh

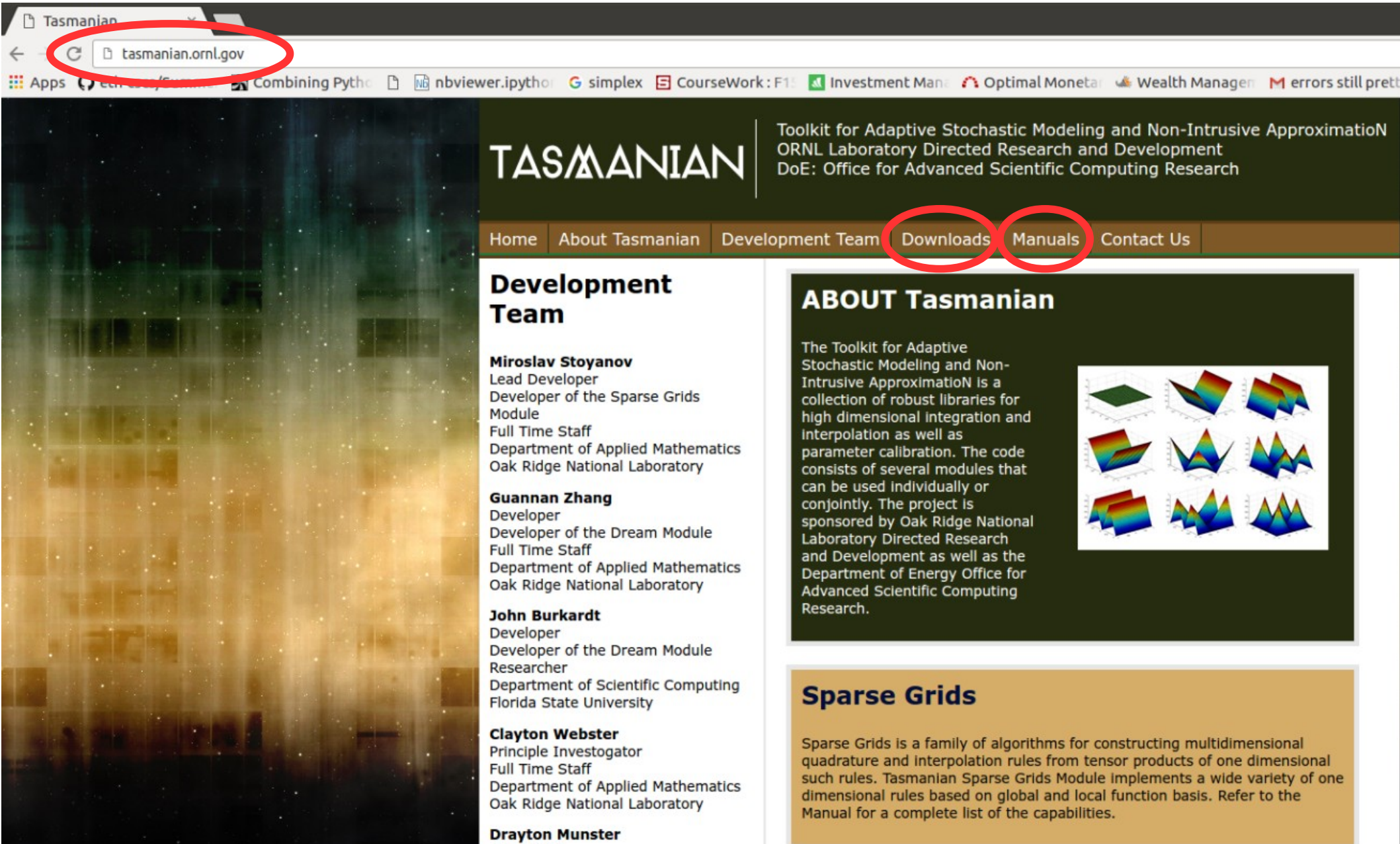
Add some lines to the .bashrc

You need to add the following lines the the .bashrc

$ vi .bashrc

#IPOPT
export LD_LIBRARY_PATH=PATH_TO_REPOSITORY/Lecture_1/SparseGridCode/pyipopt/Ipopt-3.12.5/build/lib:$LD_LIBRARY_PATH

TASMANIAN – open source code



Software tutorial

The Toolkit for Adaptive Stochastic Modeling and Non-Intrusive ApproximatioN

http://tasmanian.ornl.gov/

TASMANIAN Sparse Grids.

Very recent open source library written in CPP:

→ Contains “ordinary and adaptive” sparse grids.

→ Many more basis functions (global polynomials, wavelets,…).

→ Interfaces to **Python** and **Matlab**.
→ **You better use it out of C++ or Python**

→ Moderately parallelized.

TASMANIAN in Python

!!! READ THE * MANUAL (RTFM) !!!**

1. go to simple example:
> cd Lecture_1/SparseGridCode/analytical_examples/TASMANIAN_Python

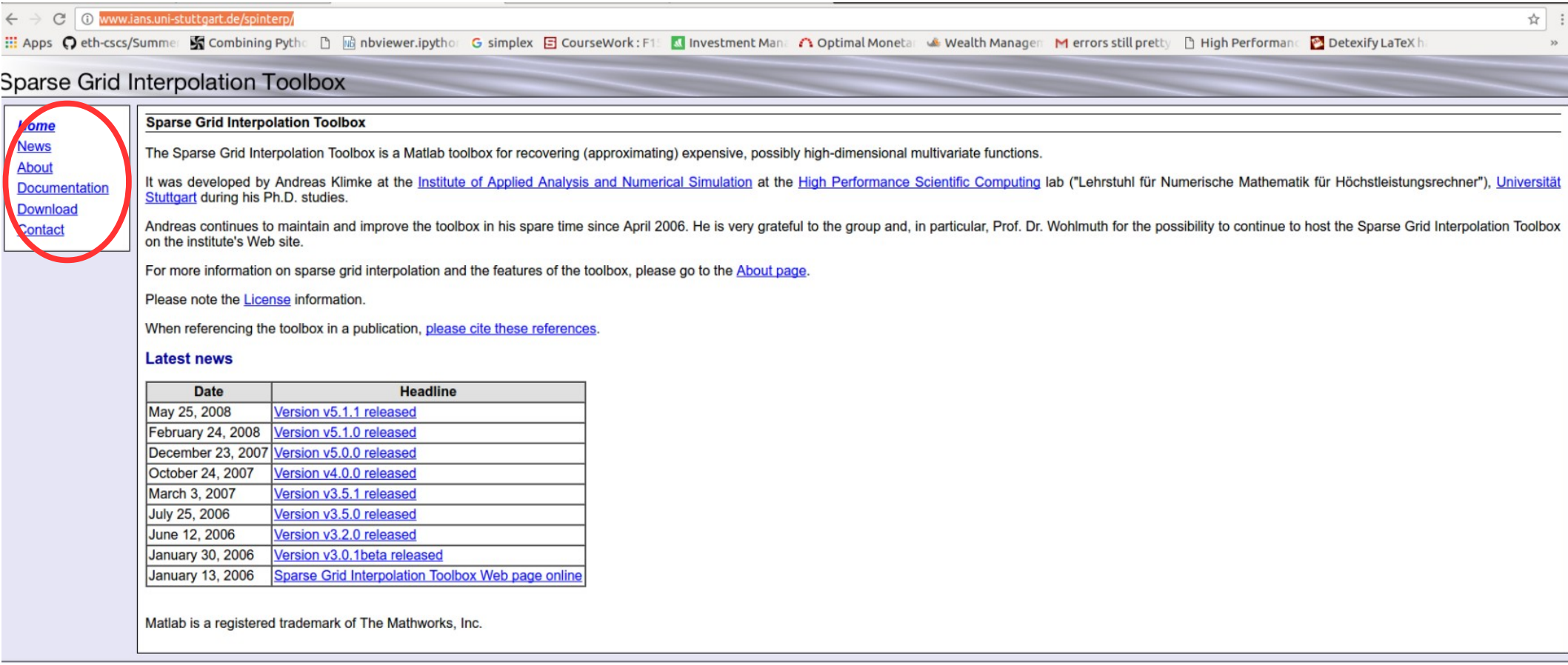
2. let's have a look at the example:
> tsg_example.py

3. run example:
> python tsg_example.py

4. NOTE: Tasmanian [-1,1]^d instead of [0,1]^d

***more examples provided from TASMANIAN in InterfacePython/example.py → beyond scope of lecture.**

Alternative Toolboxes (I)



Paper for this code: readings/p561-klimke.pdf

Alternative Toolboxes (II)

http://www.ians.uni-stuttgart.de/spinterp/

→ spinterp.

→ Matlab-based implementation of sparse grids

→ Not updated since 2008 (page sometimes even down)

→ Piecewise linear basis function and few others (global)

→ Dimensional adaptivity as options

→ **no general adaptivity**

→ **not parallel**

Run Example Code

→ **Start MATLAB without graphical interface**

```
> matlab -nojvm
```

→ **Go to example and run it.**

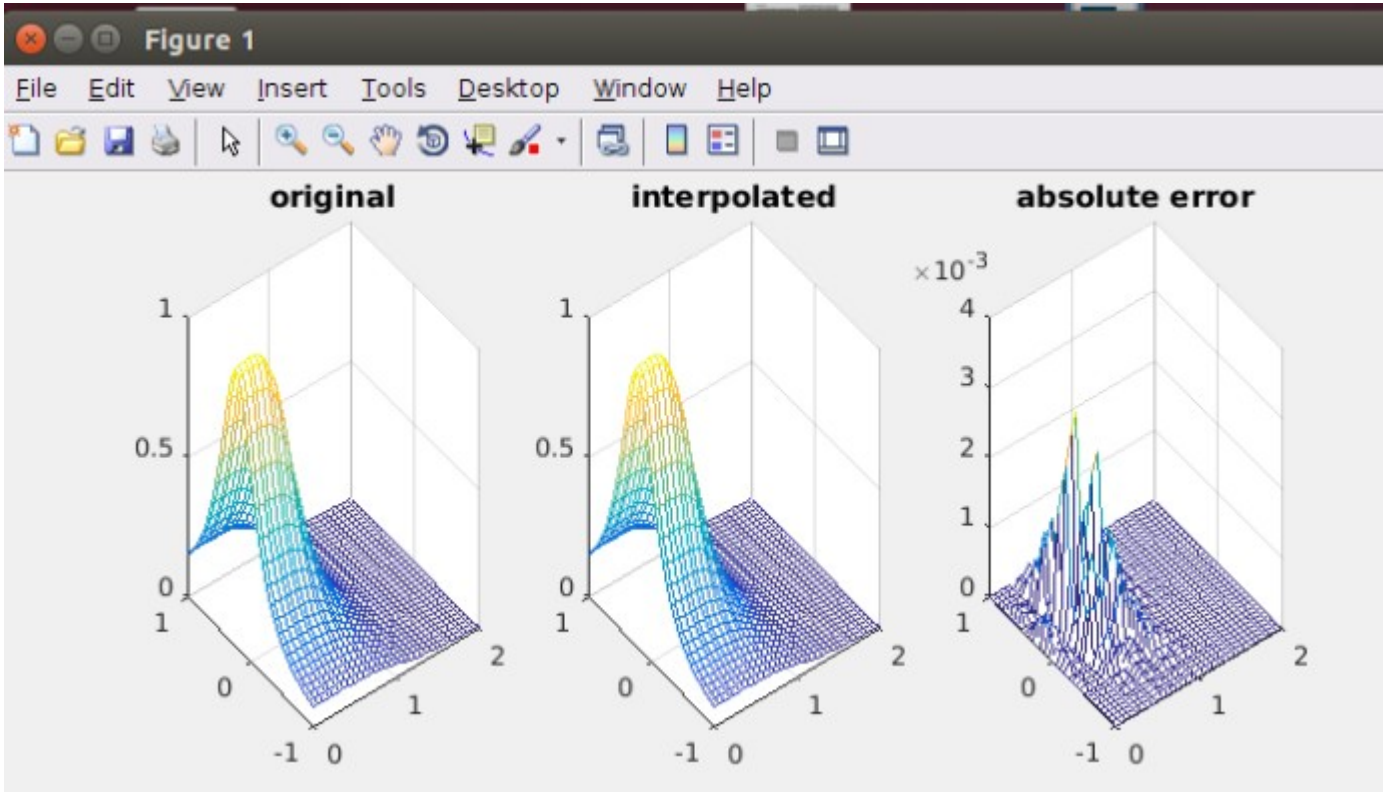
```
> cd Lecture_1/SparseGridCode/spinterp_v5.1.1
> addpath('spinterp_v5.1.1')
> spinit
> cd examples
> spdemo
```

```
%
% A 2D-example for multi-linear sparse grid interpolation using the
% Clenshaw-Curtis grid and vectorized processing of the function.
%
%    See also SPINTERP, SPVALS.

% Author : Andreas Klimke, Universität Stuttgart
% Version: 1.1
% Date   : September 29, 2003

% ------------------------------------------------------------
% Sparse Grid Interpolation Toolbox
% Copyright (c) 2006 W. Andreas Klimke, Universitaet Stuttgart
% Copyright (c) 2007-2008 W. A. Klimke. All Rights Reserved.
% See LICENSE.txt for license.
% email: klimkeas@ians.uni-stuttgart.de
% web  : http://www.ians.uni-stuttgart.de/spinterp
% ------------------------------------------------------------

% Some function f
f = inline('1./((x*2-0.3).^4 +(y*3-0.7).^2+1)');

% Define problem dimension
d = 2;

% Create full grid for plotting
gs = 33;
[X,Y] = meshgrid(linspace(0,2,gs),linspace(-1,1,gs));

% Set options: Switch vectorized processing on.
options = spset('Vectorized', 'on', 'SparseIndices', 'off');

% Compute sparse grid weights over domain [0,2]x[-1,1]
z = spvals(f, d, [0 2; -1 1], options);

% Compute inpterpolated values at full grid
ip = spinterp(z, X, Y);

% Plot original function, interpolation, and error
subplot(1,3,1);
mesh(X,Y,f(X,Y));
title('original');

subplot(1,3,2);
mesh(X,Y,ip);
title('interpolated');

subplot(1,3,3);
mesh(X,Y,abs(f(X,Y)-ip));
title('absolute error');

disp(' ');
disp('Sparse grid representation of the function:');
```

Test function

Interpolate

What you should see...



Other Toolboxes (III)

http://sgpp.sparsegrids.org/

- C++ with some plug-ins to other languages
- Multiple local basis functions

sgpp.sparsegrids.org

SG++

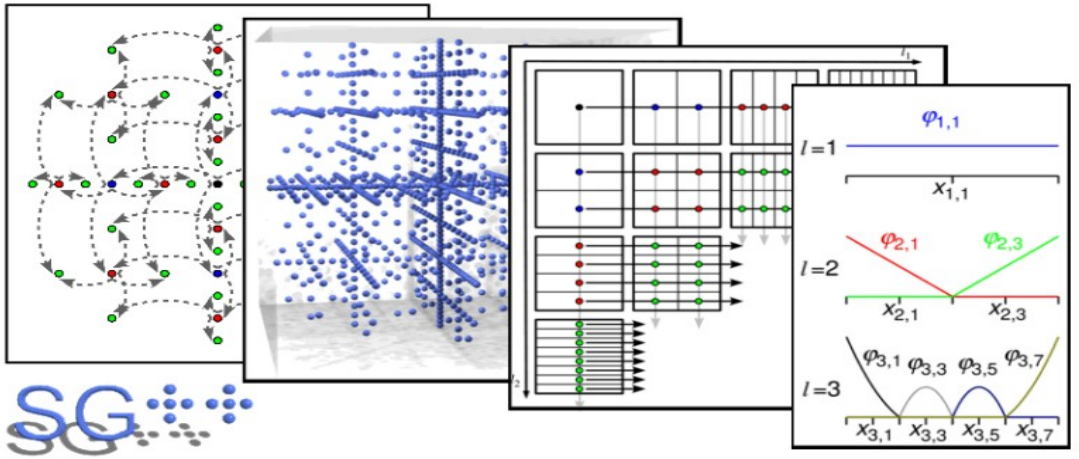
Main Page | Related Pages | Namespaces | Classes | Files

- SG++
  - SG++ Documentation
  - Todo List
  - Namespaces
  - Classes
  - Files

SG++ Documentation

Welcome to the SG++ documentation.
The current version of SG++ can be found at Downloads and Version History.

If you use any part of the software or any resource of this webpage and/or documentation, you implicitly accept the copyright (see the Copyright). This includes that you have to cite one of the papers dealing with sparse grids when publishing work with the help of SG++ (see below).



Images taken from [1]

[1] D. Pflüger, Spatially Adaptive Sparse Grids for Higher-Dimensional Problems. Verlag Dr. Hut, München, 2010. ISBN 9-783-868-53555-6.

Overview

Exercises – for later

Create sparse grids based on different analytical test functions, e.g. Genz (1984).

→ different test functions can be obtained by varying $c = (c_1, \ldots, c_d)$ (c> 0) and $w = (w_1, \ldots, w_d)$

→ difficulty of functions is monotonically increasing with c.

→ randomly generate 1,000 test points and compute error(s): $e = \max_{i=1,\ldots,1000} |f(\vec{x_i}) - u(\vec{x_i})|.$

→ **play with adaptive/non-adaptive sparse grids/refinement level and criterion.**

→ generate convergence plots (number of points versus error – as done above).

Genz (1984) test functions

1. OSCILLATORY: $f_1(x) = \cos\left(2\pi w_1 + \sum_{i=1}^{d} c_i x_i\right),$
2. PRODUCT PEAK: $f_2(x) = \prod_{i=1}^{d} \left(c_i^{-2} + (x_i - w_i)^2\right)^{-1},$
3. CORNER PEAK: $f_3(x) = \left(1 + \sum_{i=1}^{d} c_i x_i\right)^{-(d+1)},$
4. GAUSSIAN: $f_4(x) = \exp\left(-\sum_{i=1}^{d} c_i^2 t (x_i - w_i)^2\right),$
5. CONTINUOUS: $f_5(x) = \exp\left(-\sum_{i=1}^{d} c_i |x_i - w_i|\right),$
6. DISCONTINUOUS: $f_6(x) = \begin{cases} 0, & \text{if } x_1 > w_1 \text{ or } x_2 > w_2, \\ \exp\left(\sum_{i=1}^{d} c_i x_i\right), & \text{otherwise.} \end{cases}$

Value Function Iteration

The solution is approached in the limit as $j \to \infty$ by iterations on at every coordinate of the discretized grid.

$$V_{j+1}(x) = \max_u \{r(x,u) + \beta V_j(\tilde{x})\}$$

s.t.

$$\tilde{x} = g(x,u)$$

x: grid point, describes your system. State-space potentially **high-dimensional.**

`old solution': high-dimensional function, approximated by sparse grid Interpolation method on which we Interpolate.

Use-case for (adaptive) sparse grids

Growth Model & Dynamic Programming & ASG

To demonstrate the capabilities of sparse grids, we consider an **infinite-horizon discrete-time multi-dimensional optimal growth model**
(see, e.g., Scheidegger & Bilionis (2019), and references therein).

The model has few parameters and is relatively easy to explain, whereas the dimensionality of the problem can be scaled up in a straightforward but meaningful way.

→ state-space depends linearly on the number of ***D* sectors** considered.

→ there are D sectors with **capital** $\mathbf{k}_t = (k_{t,1}, ..., k_{t,D})$

and elastic **labour supply** $\mathbf{l}_t = (l_{t,1}, ..., l_{t,D})$

Growth model

The production function of sector ***i*** at time ***t*** is $f(k_{t,i}, l_{t,i}),\ \text{for}\ i = 1, ..., D.$

Consumption: $\mathbf{c}_t = (c_{t,1}, ..., c_{t,D})$

Investment of the sectors at time *t:* $\mathbf{I}_t = (I_{t,1}, ..., I_{t,D})$

→ The goal now is to find **optimal consumption** and **labour supply decisions** such that expected total utility over an infinite time horizon is maximized.

Model

$$V_0(\mathbf{k}_0) = \max_{\mathbf{k}_t, \mathbf{I}_t, \mathbf{c}_t, \mathbf{l}_t, \mathbf{\Gamma}_t} \left\{ \sum_{t=0}^{\infty} \beta^t \cdot u(\mathbf{c}_t, \mathbf{l}_t) \right\},$$

$$s.t.$$

$$k_{t+1,j} = (1-\delta) \cdot k_{t,j} + I_{t,j} \qquad j = 1, ..., D$$

$$\Gamma_{t,j} = \frac{\zeta}{2} k_{t,j} \left(\frac{I_{t,j}}{k_{t,j}} - \delta \right)^2, \quad j = 1, ..., D$$

$$\sum_{j=1}^{D} (c_{t,j} + I_{t,j} - \delta \cdot k_{t,j}) = \sum_{j=1}^{D} (f(k_{t,j}, l_{t,j}) - \Gamma_{t,j})$$

Model (II)

Convex adjustment cost of sector *j*: $\mathbf{\Gamma}_t = (\Gamma_{t,1},,...,\Gamma_{t,D})$

Capital depreciation: δ

Discount factor: β

Recursive formulation

$$V(\mathbf{k}) = \max_{\mathbf{I},\mathbf{c},\mathbf{l}} \left(u(c,l) + \beta \quad \left\{ V_{next}(k^+) \right\} \right),$$

$$s.t.$$

$$k_j^+ = (1-\delta)\cdot k_j + I_j \qquad j = 1, ..., D$$

$$\Gamma_j = \frac{\zeta}{2} k_j \left(\frac{I_j}{k_j} - \delta \right)^2, \quad j = 1, ..., D$$

$$\sum_{j=1}^{D} (c_j + I_j - \delta \cdot k_j) = \sum_{j=1}^{D} (f(k_j, l_j) - \Gamma_j)$$

where we indicate the next period's variables with a superscript "+". $\mathbf{k} = (k_1, ..., k_D)$ represents the state vector, $\mathbf{l} = (l_1, ..., l_D)$, $\mathbf{c} = (c_1, ..., c_D)$, and $\mathbf{I} = (I_1, ..., I_D)$ are $3D$ control variables. $\mathbf{k}^+ = \left(k_1^+, ..., k_D^+\right)$ is the vector of next period's variables. Today's and tomorrow's states are restricted to the finite range $[\underline{\mathbf{k}}, \overline{\mathbf{k}}]^D$, where the lower edge of the computational domain is given by $\underline{\mathbf{k}} = (\underline{k_1}, ..., \underline{k_D})$, and the upper bound is given by $\overline{\mathbf{k}} = (\overline{k_1}, ..., \overline{k_D})$. Moreover, $\mathbf{c} > 0$ and $\mathbf{l} > 0$ holds component-wise.

Utility function etc.

Productivity: $$f(k_j, l_j) \quad = \quad A \cdot k_i^{\psi} \cdot l_i^{1-\psi}$$

Utility: $$u\left(\mathbf{c}, \mathbf{l}\right) = \sum_{i=1}^{d} \left[\frac{\left(c_i / A\right)^{1-\gamma} - 1}{1-\gamma} - \left(1-\psi\right) \frac{l_i^{1+\eta} - 1}{1+\eta} \right]$$

Terminal Value function: $$V^{\infty}\left(\mathbf{k}\right) = u\left(f(k, \mathbf{e}), \mathbf{e}\right) / (\mathbf{1} - \beta)$$

where $\mathbf{e}$ is the unit vector

Parametrization

Parameter	Value
β	0.8
δ	0.025
ζ	0.5
$[\underline{\mathbf{k}}, \overline{\mathbf{k}}]^D$	$[0.2, 3.0]^D$
ψ	0.36
A	$(1-\beta)/(\psi \cdot \beta)$
γ	2
η	1

Map to sparse grid ←

Value function iteration

$$V(\mathbf{k}) = \max_{\mathbf{I},\mathbf{c},\mathbf{l}} \left(u(c,l) + \beta \quad \left\{ V_{next}(k^+) \right\} \right),$$

$$s.t.$$

$$k_j^+ = (1-\delta)\cdot k_j + I_j \qquad , \quad j = 1, ..., D$$

$$\Gamma_j = \frac{\zeta}{2} k_j \left(\frac{I_j}{k_j} - \delta \right)^2 , \quad j = 1, ..., D$$

$$\sum_{j=1}^{D} (c_j + I_j - \delta \cdot k_j) = \sum_{j=1}^{D} (f(k_j, l_j) - \Gamma_j)$$

State k: sparse grid coordinates

V_{next} : sparse grid interpolator from the previous iteration step

Solve this optimization problem at every point in the sparse grid!

Attention: Take care of the econ domain / sparse grid domain

Convergence measures (due to contraction mapping)

Average error:

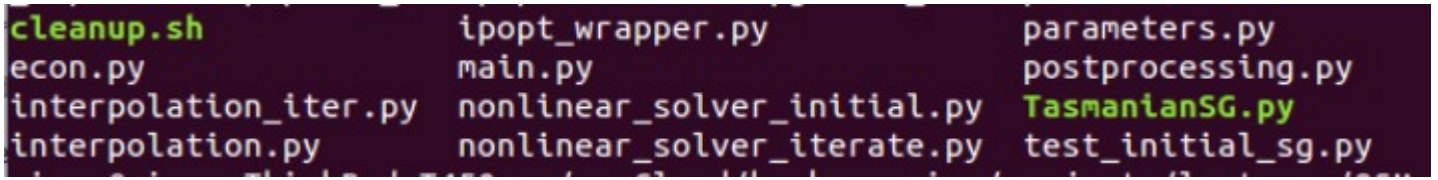
$$e^s = \frac{1}{N}\sum_{i=1}^{N} |V^s(\mathbf{x^i}) - V^{s-1}(\mathbf{x^i})|$$

Max. error:

$$a^s = \max_{i=1,N} |V^s(\mathbf{x^i}) - V^{s-1}(\mathbf{x^i})|$$

Setup of Code

Go here: Lecture_1/SparseGridCode/growth_model/serial



main.py: driver routine

econ.py: contains production function, utility,...

nonlinear_solver_initial/iterate.py: interface SG ↔ IPOPT (optimizer).

ipopt_wrapper.py: specifies the optimization problem (objective function,...).

interpolation.py: interface value function iteration ↔ sparse grid.

postprocessing.py: auxiliary routines, e.g., to compute the error.

Code snippet – main.py

```
#=====================================================================
# Start with Value Function Iteration

# terminal value function
valnew=TasmanianSG.TasmanianSparseGrid()
if (numstart==0):
    valnew=interpol.sparse_grid(n_agents, iDepth)
    valnew.write("valnew_1." + str(numstart) + ".txt") #write file to disk for restart

# value function during iteration
else:
    valnew.read("valnew_1." + str(numstart) + ".txt")  #write file to disk for restart

valold=TasmanianSG.TasmanianSparseGrid()
valold=valnew

for i in range(numstart, numits):
    valnew=TasmanianSG.TasmanianSparseGrid()
    valnew=interpol_iter.sparse_grid_iter(n_agents, iDepth, valold)
    valold=TasmanianSG.TasmanianSparseGrid()
    valold=valnew
    valnew.write("valnew_1." + str(i+1) + ".txt")

    #=================================================================
print "============================================================"
print " "
print " Computation of a growth model of dimension ", n_agents ," finished after ", numits, " steps"
print " "
print "============================================================"
#=====================================================================

# compute errors
avg_err=post.ls_error(n_agents, numstart, numits, No_samples)

#=====================================================================
print "============================================================"
print " "
print " Errors are computed -- see error.txt"
print " "
print "============================================================"
#=====================================================================
```

Code snippet – parameters.py

```
# Depth of "Classical" Sparse grid
iDepth=2
iOut=1          # how many outputs
which_basis = 1 #linear basis function (2: quadratic local basis)

# control of iterations
numstart = 0   # which is iteration to start (numstart = 0: start from scratch, number=/0: restart)
numits = 10    # which is the iteration to end

# How many random points for computing the errors
No_samples = 1000

#=========================================================================

# Model Paramters
n_agents=2  # number of continuous dimensions of the model

beta=0.8
rho=0.95
zeta=0.5
psi=0.36
gamma=2.0
delta=0.025
eta=1
big_A=(1.0-beta)/(psi*beta)

# Ranges For States
range_cube=1 # range of [0..1]^d in 1D
k_bar=0.2
k_up=3.0

# Ranges for Controls
c_bar=1e-2
c_up=1.0

l_bar=1e-2
l_up=1.0

inv_bar=1e-2
inv_up=1.0

#=========================================================================
```

Code snippet – econ.py

```
#=====================================================================
#utility function u(c,l)

def utility(cons=[], lab=[]):
    sum_util=0.0
    n=len(cons)
    for i in range(n):
        nom1=(cons[i]/big_A)**(1.0-gamma) -1.0
        den1=1.0-gamma

        nom2=(1.0-psi)*((lab[i]**(1.0+eta)) -1.0)
        den2=1.0+eta

        sum_util+=(nom1/den1 - nom2/den2)

    util=sum_util

    return util


#=====================================================================
# output_f

def output_f(kap=[], lab=[]):
    fun_val = big_A*(kap**psi)*(lab**(1.0 - psi))
    return fun_val
```

Code snippet – ipopt_wrapper.py

```
#=========================================================================
#   Objective Function to start VFI (in our case, the value function)

def EV_F(X, k_init, n_agents):

    # Extract Variables
    cons=X[0:n_agents]
    lab=X[n_agents:2*n_agents]
    inv=X[2*n_agents:3*n_agents]

    knext= (1-delta)*k_init + inv
    # Compute Value Function

    VT_sum=utility(cons, lab) + beta*V_INFINITY(knext)

    return VT_sum

# V infinity
def V_INFINITY(k=[]):
    e=np.ones(len(k))
    c=output_f(k,e)
    v_infinity=utility(c,e)/(1-beta)
    return v_infinity

#=========================================================================
#   Objective Function during VFI (note - we need to interpolate on an "old" sprase grid)

def EV_F_ITER(X, k_init, n_agents, grid):

    # Extract Variables
    cons=X[0:n_agents]
    lab=X[n_agents:2*n_agents]
    inv=X[2*n_agents:3*n_agents]

    knext= (1-delta)*k_init + inv

    # Compute Value Function

    VT_sum=utility(cons, lab) + beta*grid.evaluate(knext)

    return VT_sum

#=========================================================================
```

Run the Growth model code

→ Model implemented in Python (TASMANIAN)

→ Optimizer used: IPOPT & PYIPOPT (python interface)

→ Lecture_1/SparseGridCode/growth_model/serial_growth

→ run with

>python main.py

A stochastic growth model

→ Model with stochastic production

$$f(k_i, l_i, \theta_i) = \theta_i A k_i^{\psi} l_i^{1-\psi}$$

→ Here we assume 5 possible values of
Θ_i = {0.9, 0.95, 1.00, 1.05, 1.10}

→ for simplicity, we assume **Π(*,*) = 1/5** (no Markov chain)

→ solve

$$V_t(k, \theta) = \max_{c,l,I} u(c, l) + \beta \mathbb{E}\left\{ V_{t+1}(k^+, \theta^+) \mid \theta \right\}$$

Code snippet – main.py

```
import nonlinear_solver_initial as solver    #solves opt. problems for terminal VF
import nonlinear_solver_iterate as solviter  #solves opt. problems during VFI
from parameters import *                     #parameters of model
import interpolation as interpol             #interface to sparse grid library/terminal VF
import interpolation_iter as interpol_iter   #interface to sparse grid library/iteration
import test_initial_sg as initial
import postprocessing as post                #computes the L2 and Linfinity error of the model


import TasmanianSG                           #sparse grid library
import numpy as np

#=====================================================================
# Start with Value Function Iteration

valnew=[]
if (numstart==0):
    valnew=interpol.sparse_grid(n_agents, iDepth)

    for itheta in range(ntheta):
        valnew[itheta].write("valnew_"+str(theta_range[itheta])+"_" + str(numstart) + ".txt")

else:
    for itheta in range(ntheta):
        valnew.append(TasmanianSG.TasmanianSparseGrid())
        valnew[itheta].read("valnew_"+str(theta_range[itheta])+"_" + str(numstart) + ".txt")

valold=[]
valold=valnew

for i in range(numstart, numits):
    valnew=[]
    valnew=interpol_iter.sparse_grid_iter(n_agents, iDepth, valold)
    valold=[]
    valold=valnew

    for itheta in range(ntheta):
        valnew[itheta].write("valnew_"+str(theta_range[itheta])+ "_" + str(i+1) + ".txt")

#=====================================================================
print "============================================================="
print " "
print " Computation of a growth model of dimension ", n_agents ," finished after ", numits, " steps"
print " "
print "============================================================="
#=====================================================================
```

Code snippet – IPOPT_wrapper.py

```
#=============================================================================
#   Objective Function during VFI (note - we need to interpolate on an "old" sprase grid)
def EV_F_ITER(X, k_init, theta_init, n_agents, grid_list):

    # Extract Variables
    cons=X[0:n_agents]
    lab=X[n_agents:2*n_agents]
    inv=X[2*n_agents:3*n_agents]

    knext= (1-delta)*k_init + inv

    # Compute E[V(next, theta)]
    exp_v=0.0

    for itheta in range(ntheta):
        theta_next=theta_range[itheta]
        exp_v+=prob(theta_init, theta_next)*grid_list[itheta].evaluate(knext)


    # Compute Value Function
    VT_sum=utility(cons, lab) + beta*exp_v

    return VT_sum
```

Run the Growth model code

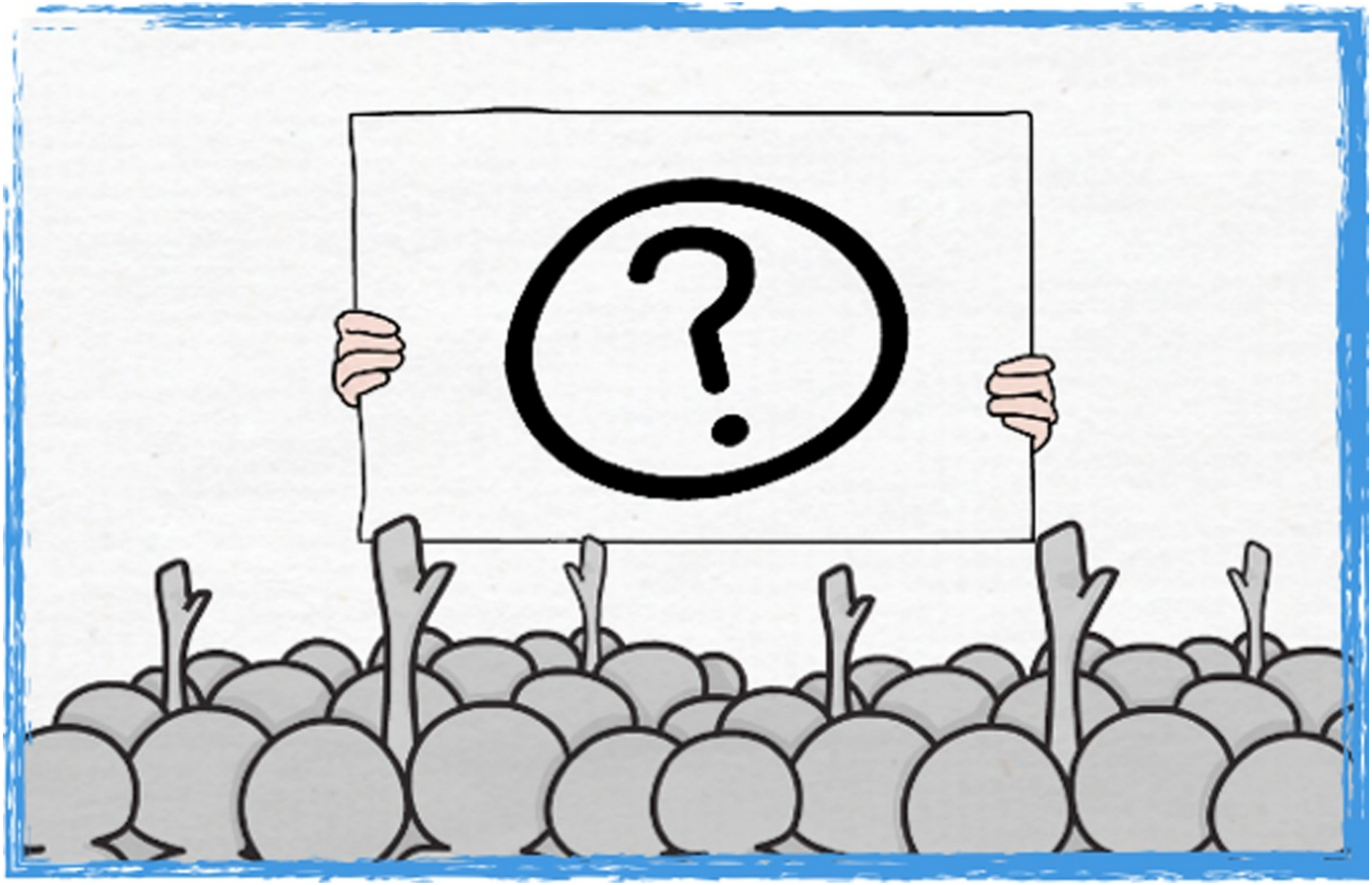
→ Model implemented in Python (TASMANIAN)

→ Optimizer used: IPOPT & PYIPOPT (python interface)

→ Lecture_1/SparseGridCode/growth_model/serial_stochastic

→ run with

>python main.py



A1: Formal Details of Sparse Grids

Proposition 1: *The number of grid points in the sparse grid space* V_n^S *is given by*

$$|V_n^S| = \sum_{i=0}^{n-1} 2^i \binom{d-1+i}{d-1} = 2^n \cdot \left(\frac{n^{d-1}}{(d-1)!} + \mathcal{O}(n^{d-2}) \right).$$

For the proof of this proposition see Lemma 3.6 of [1].
Note that Proposition 1 directly implies that

$$|V_n^S| = \mathcal{O}(2^n \cdot n^{d-1}).$$

[1] H.-J. Bungartz and M. Griebel. Sparse grids. Acta Numerica, 13:1–123, 2004.

A1: Formal Details of Sparse Grids (II)

- As mentioned before, SGs arise from a “cost-benefit” consideration:
  → find the approximation space

$$V^{opt} \subset V := \bigcup_{n=1}^{\infty} V_n$$

that provides the highest accuracy for a given number of grid points.

- Clearly, the answer to this question depends on the class of functions we would like to interpolate.

- The theory of SGs considers the Sobolev space of functions with bounded second-order mixed derivatives:

$$H_2(\Omega) := \{f : \Omega \to \mathbb{R} : D^{\vec{l}} f \in L_2(\Omega), |\vec{l}|_\infty \leq 2, f|_{\partial\Omega} = 0\},$$

where $D^{\vec{l}} f := \dfrac{\partial^{|\vec{l}|_1}}{\partial x_1^{l_1} \cdots \partial x_d^{l_d}} f.$ $\quad |\boldsymbol{l}|_\infty = \max_{1 \leq t \leq d} l_t \quad |\boldsymbol{l}|_1 = \sum_{t=1}^{d} l_t$

A1: Formal Details of Sparse Grids (III)

- V^{opt} depends on the norm $||\cdot||$ in which the interpolation error is measured.
- Proposition 2, below, holds for the L_2 and L_∞ norms as well as for

$$|f|_{\alpha,2} := \left(\int_\Omega |D^\zeta f|^2 d\vec{x} \right)^{\frac{1}{2}}, \quad |f|_{\alpha,\infty} := ||D^\alpha f||_\infty$$

with $\alpha = 2$.

A1: Formal Details of Sparse Grids (IV)

- In order to leverage on the hierarchical setting introduced, we only allow discrete spaces of the type

$$U_{\vec{I}} := \bigoplus_{\vec{l} \subset \vec{I}} W_{\vec{l}}$$

- for an arbitrary index set $\vec{I}$ as candidates for the optimization process.

- We use $f_{U_{\vec{I}}} \in U_{\vec{I}}$ to denote the interpolant of f from the approximation space $U_{\vec{I}}$.

A1: Formal Details of Sparse Grids (V)

We are now in the position to state precisely in which sense “classical” SGs are optimal.

Proposition 2: *The sparse grid approximation space*

$$V_n^S = \bigoplus_{|\vec{l}|_1 \leq n+d-1} W_{\vec{l}}$$

is the solution to the constrained optimization problem

$$\min_{U_{\vec{I}} \subset V : |U_{\vec{I}}| \leq |V_n^S|} \quad \max_{f \in H_2(\Omega) : |f|=1} \|f - u_{U_{\vec{I}}}\|.$$

A1: Interpretation of Proposition 2

- In words, the sparse grid V_n^S minimizes among all approximation spaces $U_{\vec{l}}$ which do not have more grid points $(|U_{\vec{l}}| \leq |V_n^S|)$ the maximal approximation error reached when interpolating functions with bounded second-order mixed derivatives ($f \in H_2(\Omega)$) and a given variability.

- Finally, note that “classical” SGs are also the solution to the reverse optimization problem of achieving some desired accuracy with the smallest possible number of grid points.