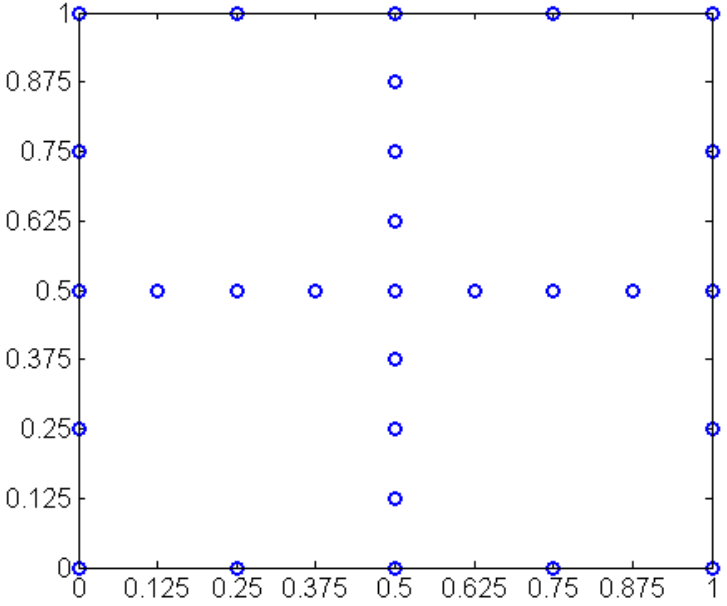
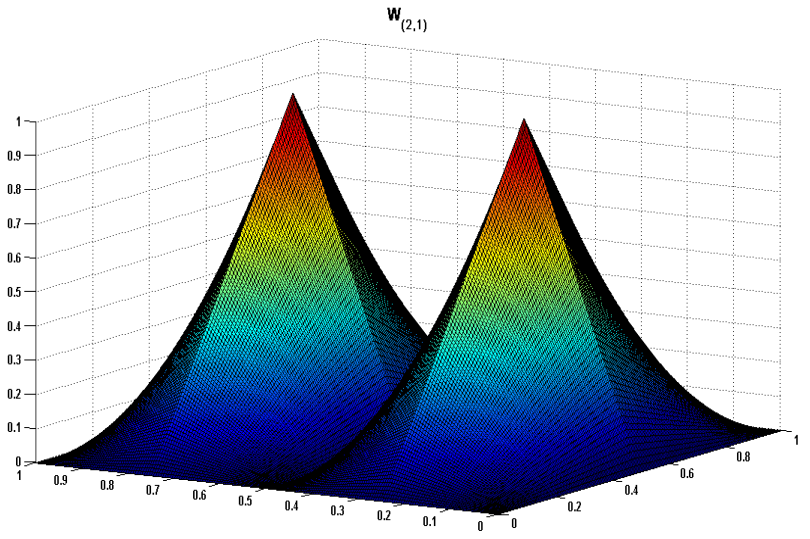
Universität Zürich[UZH]

Using Adaptive Sparse Grids to Solve High-Dimensional Dynamic Models

Johannes Brumm & Simon Scheidegger
ZICE, University of Zürich, Feb. 5th 2014





Outline

I.) From Full (Cartesian) Grids to Sparse Grids

II.) Adaptive Sparse Grids

III.) Time Iteration, Adaptive Sparse Grids & HPC

→ Implementation, Testing, Results

Our motivation

i) Want to solve dynamic stochastic models with high-dimensional state spaces:

$$\text{“}\Theta v = v\text{”} \longleftrightarrow |\Theta v_i - v_{i+1}| < \varepsilon$$

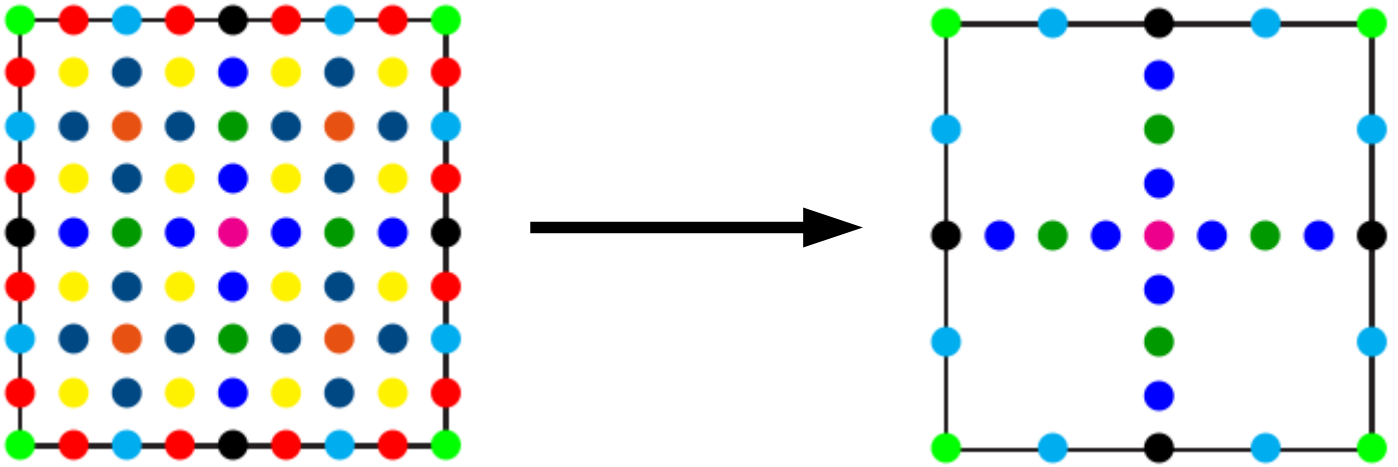
→ Have to interpolate high-dimensional functions

Problem: curse of dimensionality
→ N^d points in ordinary discretization schemes

ii) Want to overcome curse of dimensionality
iii) **Want locality & adaptivity of interpolation scheme**
(ability to handle singularities, kinks,...)

iv) Speed-up → **access HPC systems** (MPI, OpenMP)

I. From Full Grids to Sparse Grids



Cartesian Grid

Sparse Grid

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 $i_t \in \{0, 1, ..., 2^{l_t}\}$

Interpolation on a Full Grid

-Consider a **d-dimensional function** $f : \Omega \to \mathbb{R}$

-In numerical simulations:
f might be expensive to evaluate! (solve PDEs/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:**

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

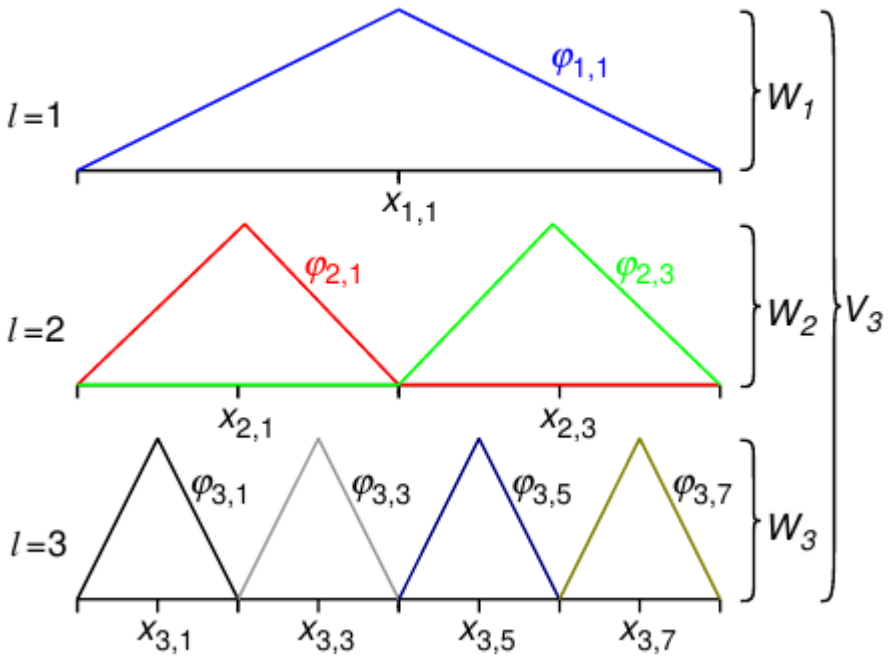


Fig.: **1-d basis functions** $\phi_{l,i}$ and the corresponding **grid points** up level ***l* = 3** in the hierarchical basis.

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

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

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

Multi-d interpolant:

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

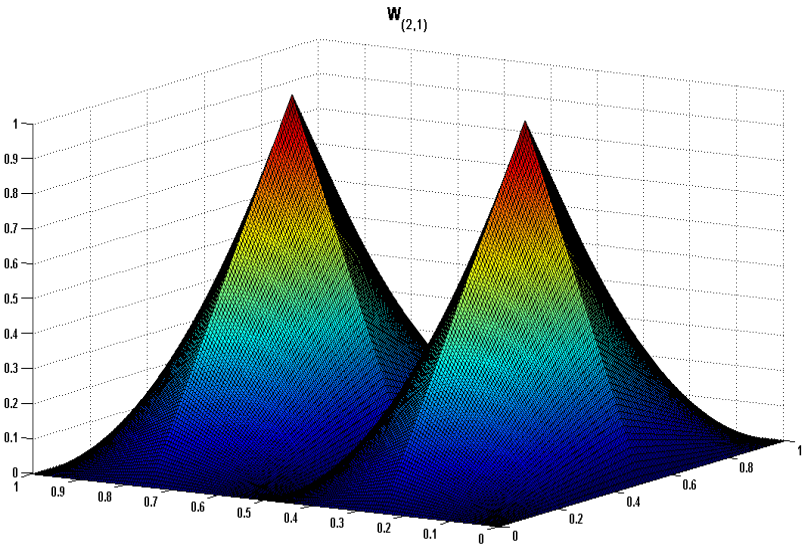


Fig.: Basis functions of the **subspace $W_{2,1}$**

Hierarchical surplus – nested structure

Coefficients of interpolant termed **“hierarchical surpluses”**.

Can easily be determined due to **nested property** of the hierarchical grid.

$$\alpha_{l,i} = f(x_{l,i}) - \frac{f(x_{l,i} - h_l) + f(x_{l,i} + h_l)}{2}$$

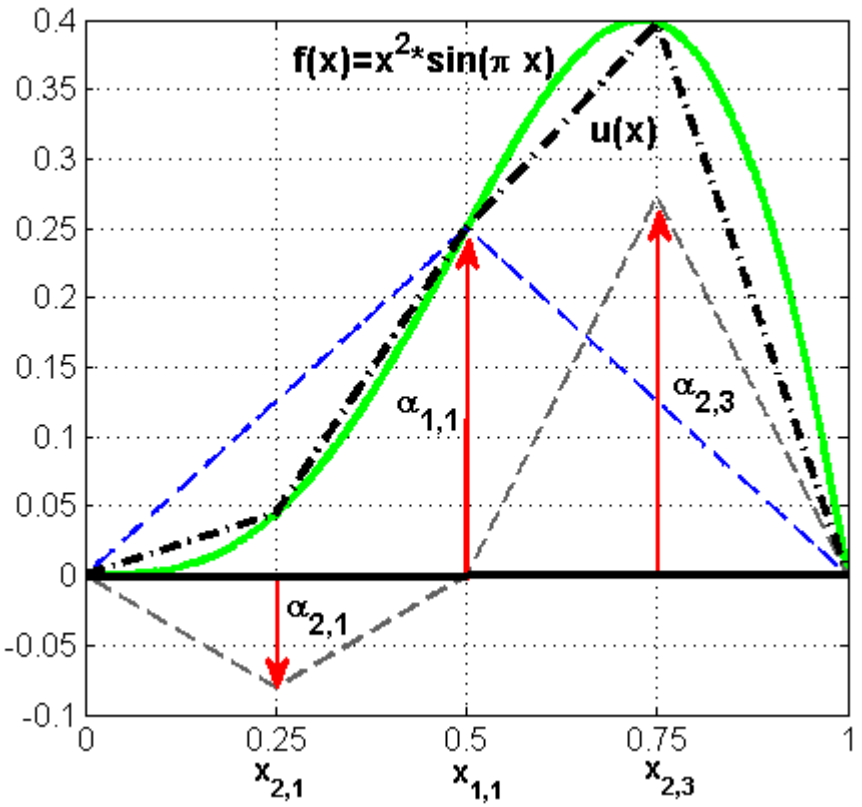


Fig.: Interpolant u of f of level 2.

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: go from one level or refinement to the next

→ **Evaluate function only at points that are unique to the new level**.

Why reality bites...

Interpolant consists of $(2^n - 1)^d$ grid points

For **sufficiently smooth *f*** and its interpolant ***u***, we obtain an asymptotic error decay of

$$\| f(\vec{x}) - u(\vec{x}) \|_{L_2} \in \mathcal{O}\left(h_n^2\right) \quad \text{where} \quad \|f\|_{L_2} := \left(\int_\Omega |f(\vec{x})|^2 d\vec{x} \right)^{1/2}$$

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

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

-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***.

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

Note: This result is optimal for the L_2- Norm and the $L_\infty-$ Norm.

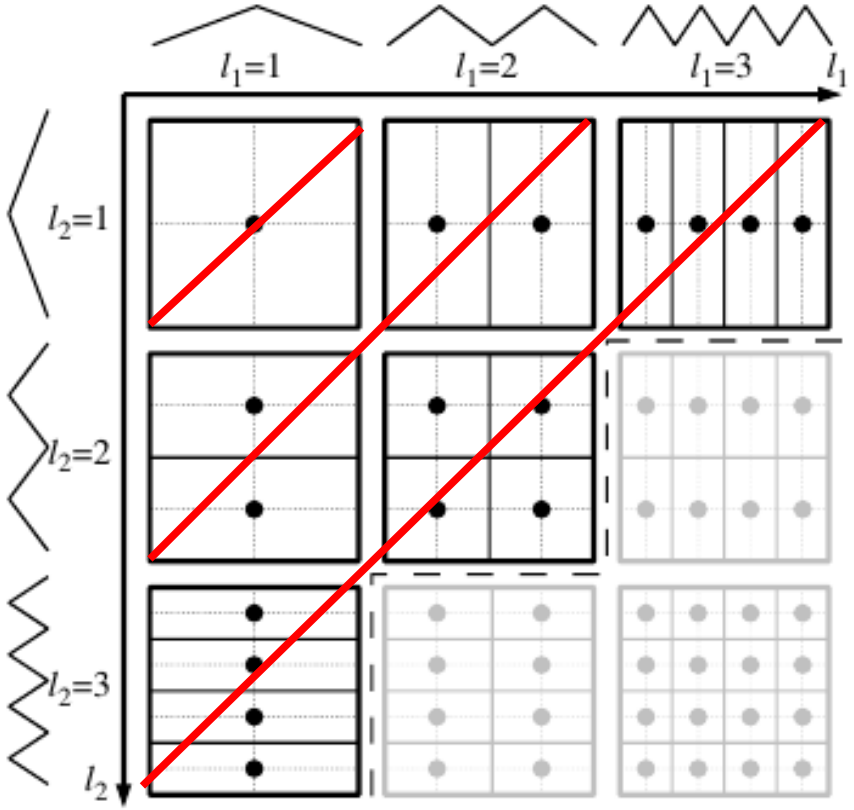
Interpolant: $$f_{0,n}^S(\vec{x}) \approx u(\vec{x}) = \sum_{|l|_1 \leq n+d-1} \sum_{\vec{i} \in I_{\vec{l}}} \alpha_{\vec{l},\vec{i}} \cdot \phi_{\vec{l},\vec{i}}(\vec{x})$$

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

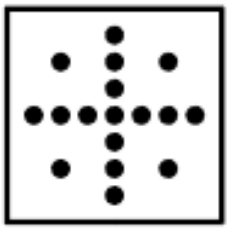
Accuracy of the interpolant: $\mathcal{O}\left(h_n^2 \cdot \log(h_n^{-1})^{d-1}\right)$ vs. $\mathcal{O}\left(h_n^2\right)$

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

Sparse grid construction in 2D



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



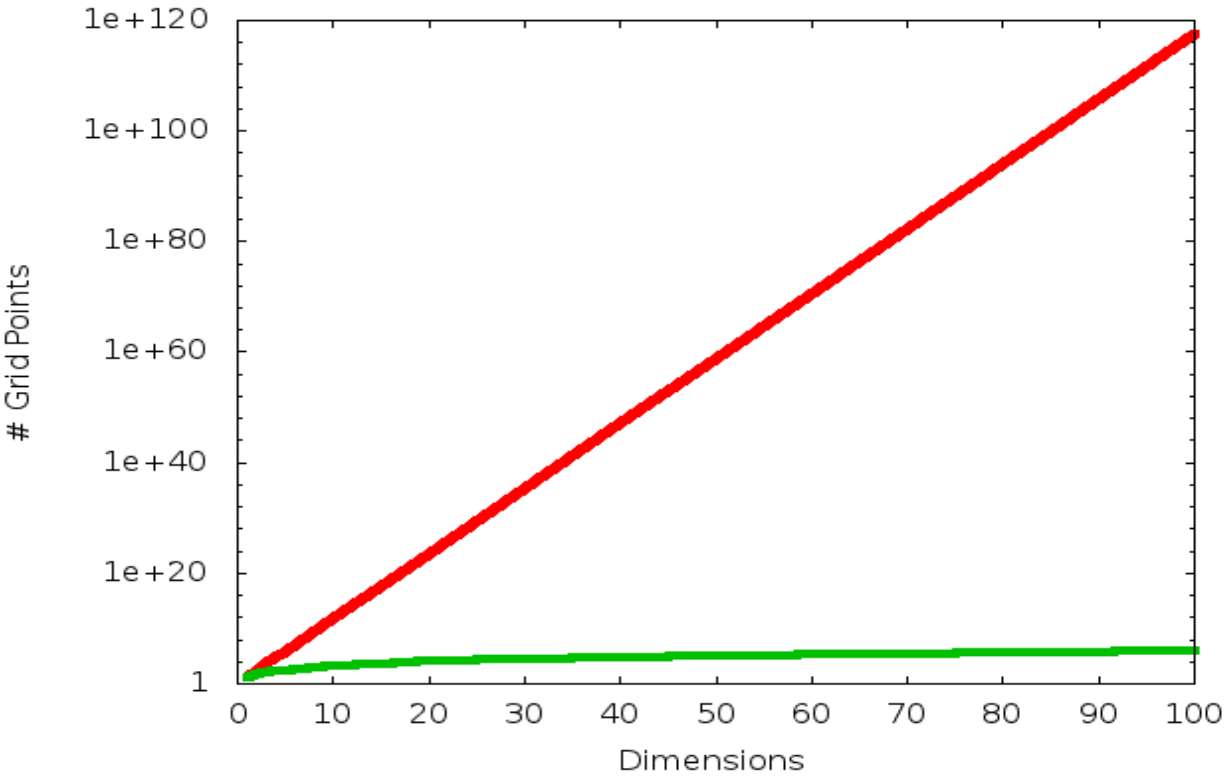
V_3

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.

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).

Where are Sparse Grids used?

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

Sparse grid methods date back to Smolyak(1963)

So far, methods applied to:

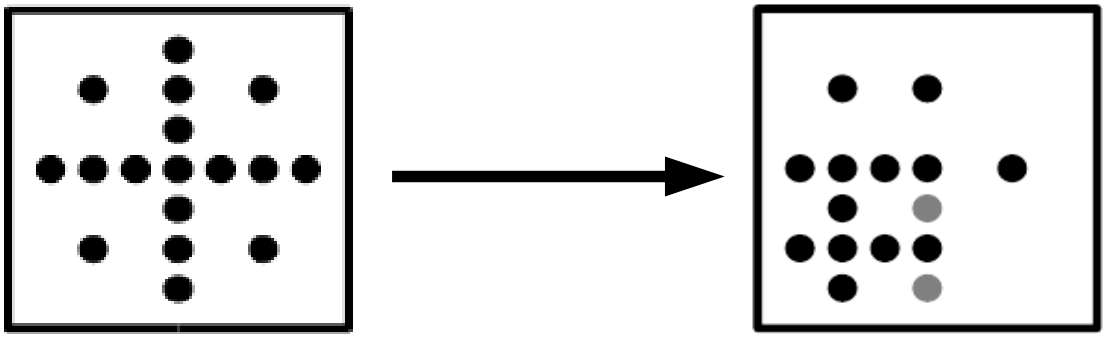
-High-dimensional integration
e.g. Gerstner & Griebel (1998), Bungartz et al. (2003),...

-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, DSGE models in Economics...
e.g. Kubler & Kruger (2004), Winschel & Kraetzig (2010), Judd et al. (2013)

II. Adaptive Sparse Grids



Adaptive grids in general

-Ordinary sparse grids: *a priori* selection of grid points, optimal under certain smoothness conditions.

-Real-world applications: often do not fulfil these prerequisites (functions of interest often show kinks, finite discontinuities, steep gradients...)

→ Sparse grid methods outlined so far may converge slowly (can capture local behaviour only to some limited extend)

-Effective strategy to achieve this: **ADAPTIVITY**

→ **refine sparse grid in regions of high curvature.**

→ **spend less points in the region of smooth variation.**

Sketch of adaptive refinement

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

-Surpluses quickly decay to zero as the level of interpolation increases assuming a smooth fct.

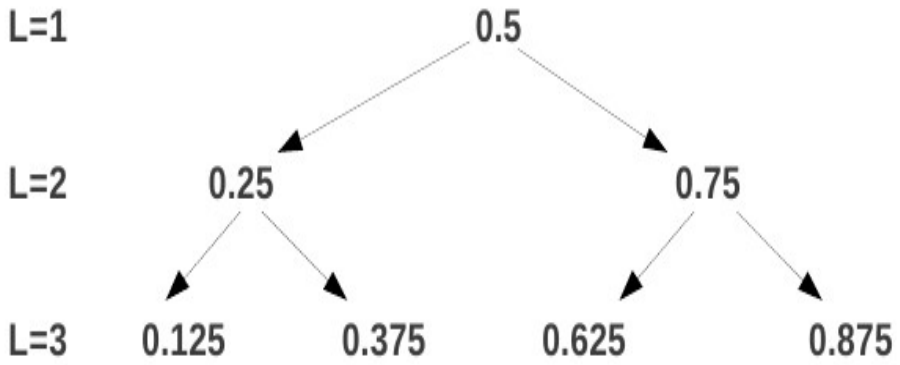
-**Use hierarchical surplus as error indicator**.

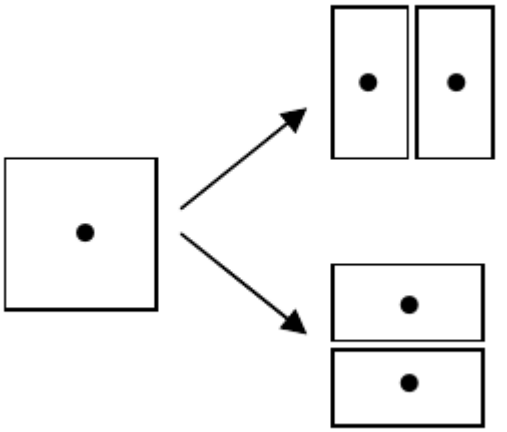
-**Automatically detect "discontinuity regions"** and adaptively refine the points in this region.

-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.

Adaptive Sparse Grid Algorithm

a) Construction of interpolant level by level

b) First calculate hierarchical surplus for each point.

Check whether refinement criterion is satisfied $|\alpha_{\vec{l},\vec{i}}| \geq \epsilon$

If so, generate **2d** neighbouring points.

Evaluate the **surplus for each new point** (in parallel).

c) Stop refinement
→ either nothing to refine, or max. level reached (if discontinuity is strong)

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:

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

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

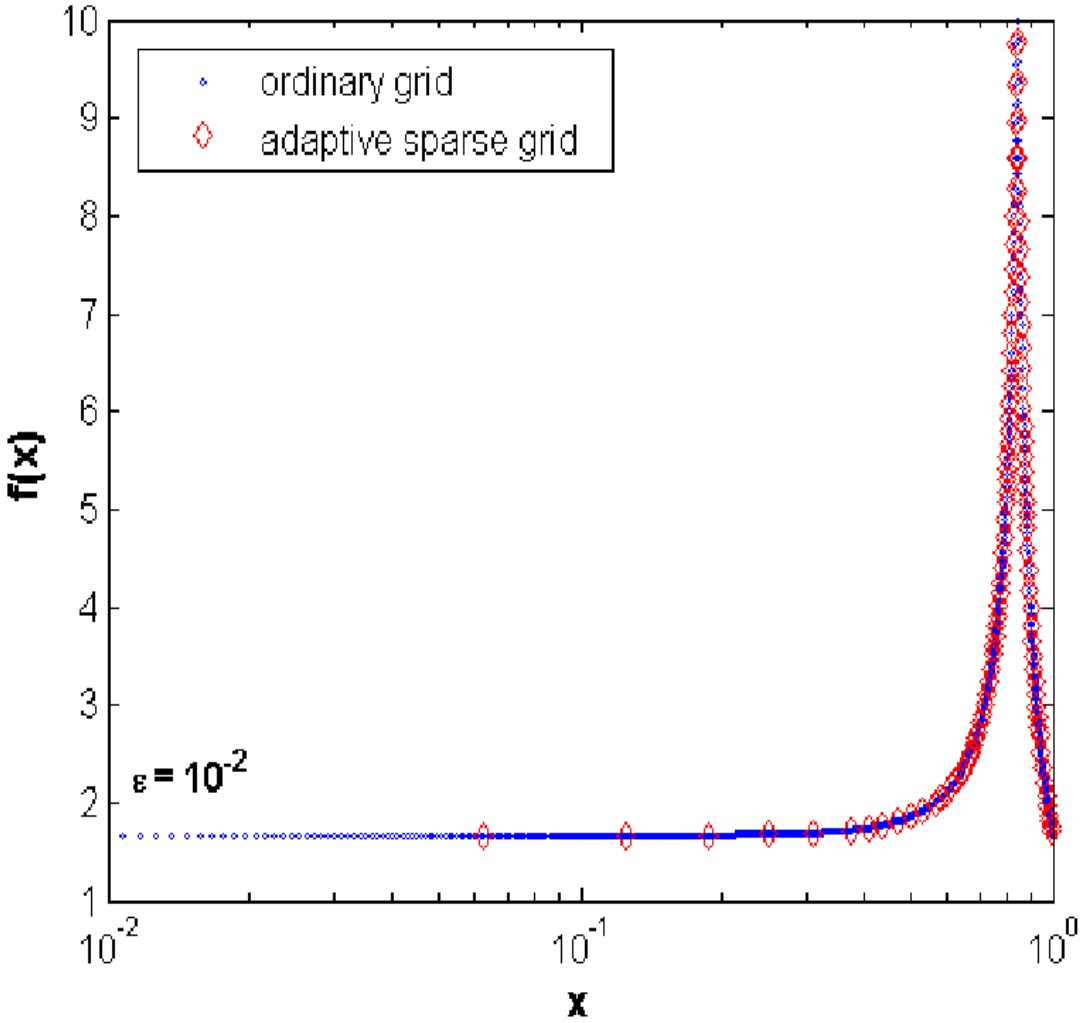


Fig.: Blue: Full grid;red: adaptive sparse grid.

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**

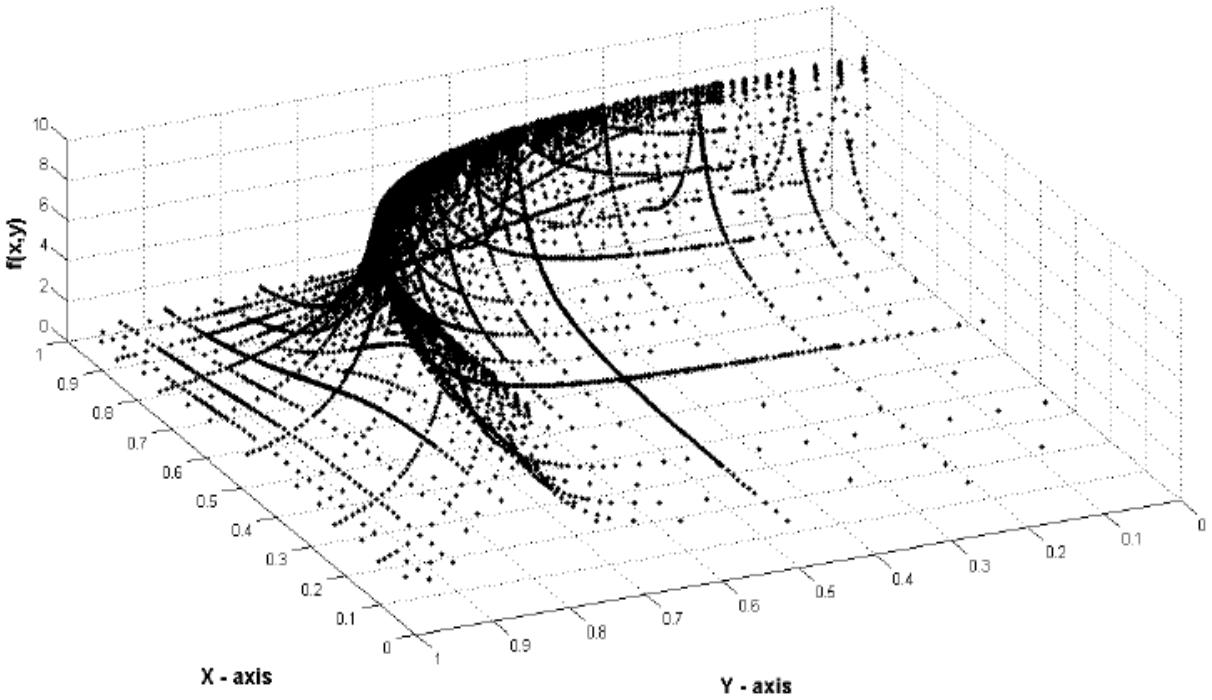


Fig.: 2d test function and its corresponding grid points after 15 refinement steps.

Movie

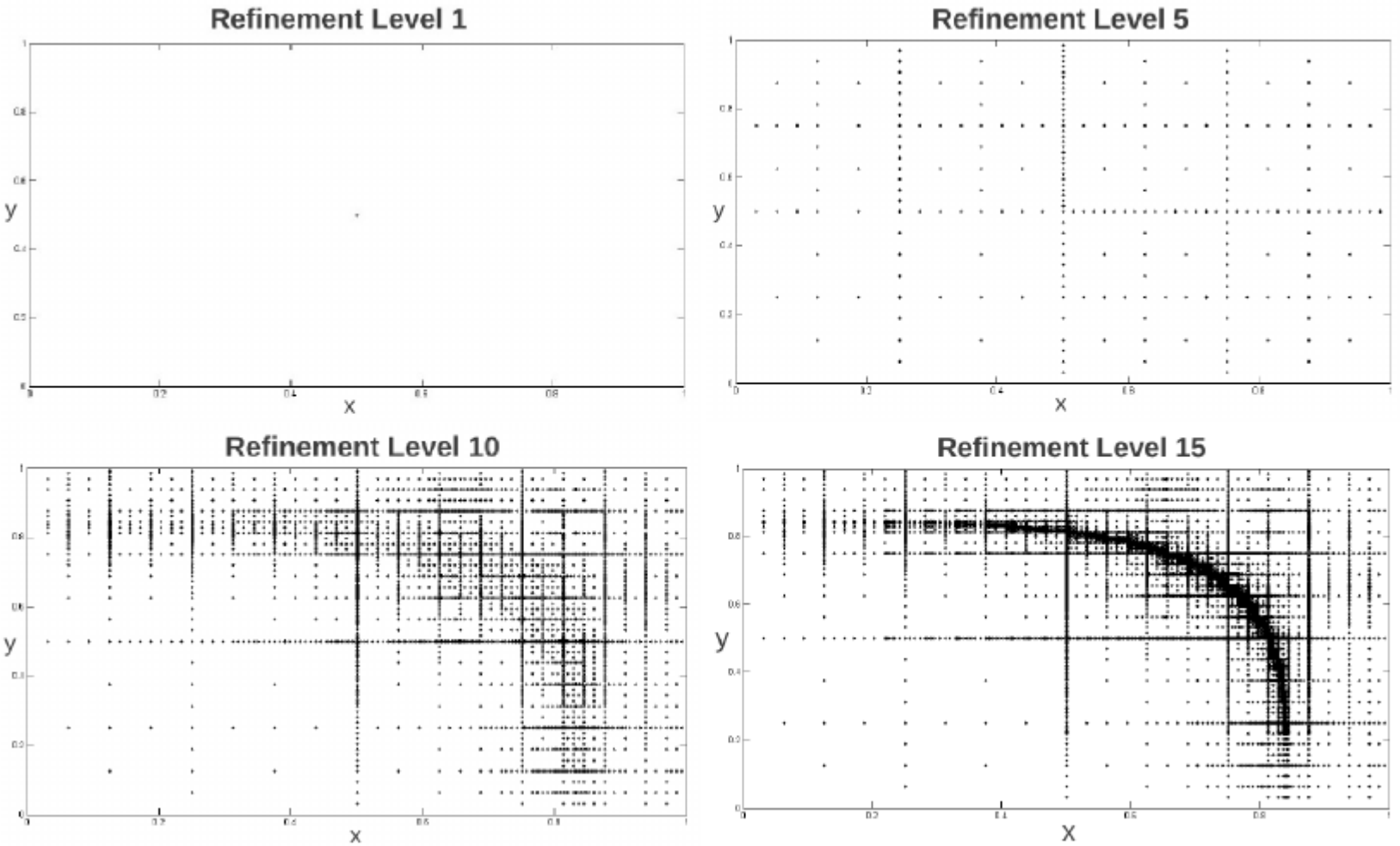
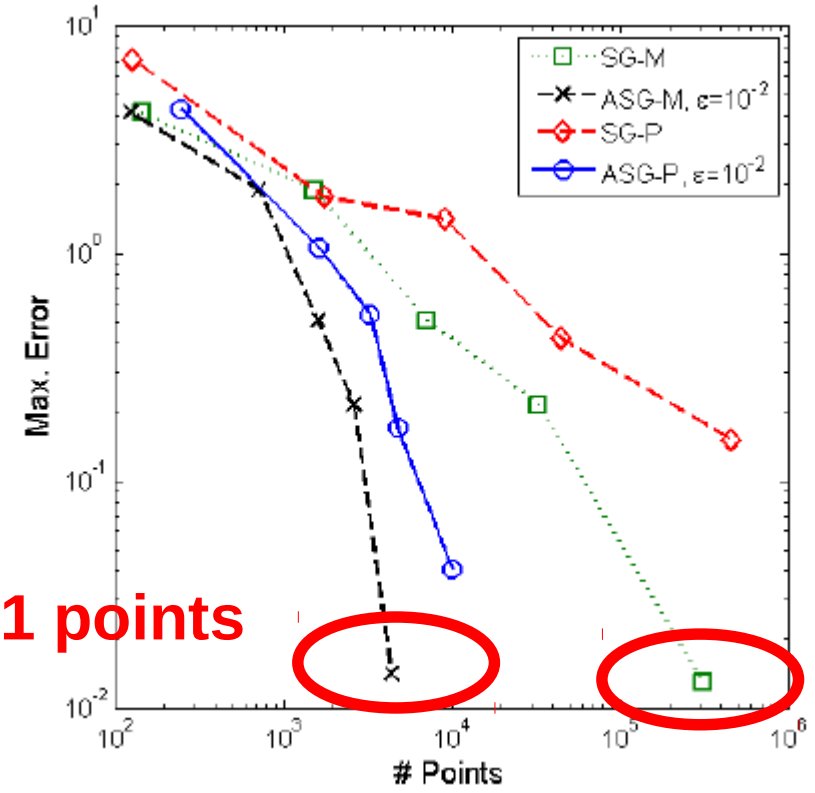


Fig.: Evolution of the adaptive sparse grid with a **threshold for refinement of 10^{-2}** . The refinement levels displayed are L = 1, 5, 10, 15.

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).

III. Putting things together: Time Iteration, Adaptive Sparse Grids & HPC

IRBC: Model ingredients

RECALL: WE WANT TO SOLVE HIGH-DIM MODELS

- Use standard problem for testing comp. methods for high-dim problems.
→ International Real Business Cycle model (IRBC) with adjustment costs

(e.g. Den Haan et al. (2011), Malin et al. (2011))

- *N* countries facing productivity shocks and capital adjustment costs

→ they differ wrt. **productivity** (exogen.) '***a***' & **capital stock** (endogen.) '***k***'

→ dimension of the state space / grid: **dim=2N**

→ one Euler equation per country plus aggregate resource constraint:
N+1 equations characterize equilibrium at each point

- Use time iteration to solve for the optimal policy $\mathbf{p: R_+^{2N} \rightarrow R_+^{N+1}}$

Time iteration algorithm

Time iteration algorithm

1. Make an initial guess for next period's policy function:

$$p_{init} = \left(k_{t+2}^1, \ldots, k_{t+2}^N, \lambda_{t+1}\right).$$

Set $p_{next} = p_{init}$.

2. Make one time iteration step:

(a) Using an adaptive sparse grid procedure, construct a set G of grid points

$$g = \left(a_t^1, \ldots, a_t^N, k_t^1, \ldots, k_t^N\right) \in G \subset S$$

and optimal policies at these points

$$p(g) = \left(k_{t+1}^1(g), \ldots, k_{t+1}^N(g), \lambda_t(g)\right)$$

by solving at each grid point g the system of equilibrium conditions (cf., Eqs. 51 and 52) given next period's policy

$$p_{next} = \left(k_{t+2}^1, \ldots, k_{t+2}^N, \lambda_{t+1}\right).$$

(b) Define the policy function p by interpolating between $\{p(g)\}_{g \in G}$.

(c) Calculate (an approximation for) the error, e.g.

$$\eta = \|p - p_{next}\|_\infty.$$

If $\eta > \epsilon$, set $p_{next} = p$ and go to step 2, else go to step 3.

3. The (approximate) equilibrium policy function is given by p.

Time Iteration Algorithm: Detail

$$g^j_{t+1} := k^j_{t+1}/k^j_t - 1, \; g^j_{t+2} := k^j_{t+2}/k^j_{t+1} - 1,$$

N Euler equations

$$\forall j : \lambda_t \cdot \left[1 + \phi \cdot g^j_{t+1}\right] - \beta \cdot \mathbb{E}_t \left\{ \lambda_{t+1} \cdot \left[a^j_{t+1} \cdot A \cdot \alpha \cdot (k^j_{t+1})^{\alpha-1} + (1-\delta) + \frac{\phi}{2} \cdot g^j_{t+2} \cdot \left(g^j_{t+2} + 2\right) \right] \right\} = 0,$$

$$\sum_{j=1}^{N} \left(a^j_t \cdot A \cdot (k^j_t)^\alpha + k^j_t \cdot \left((1-\delta) - \frac{\phi}{2} \cdot (g^j_{t+1})^2 \right) - k^j_{t+1} - \left(\frac{\lambda_t}{\tau_j} \right)^{-\gamma^j} \right) = 0.$$

1 aggregate resource constraint

We solve the model by iterating on these FOCs, i.e. at each grid point

$$(a^1_t, \ldots, a^N_t, k^1_t, \ldots, k^N_t)$$ → **Current grid points**

solve for the unknown policy variables

$$(k^1_{t+1}, \ldots, k^N_{t+1}, \lambda_t)$$ → **Solution of system of Eqs. At the current iteration**

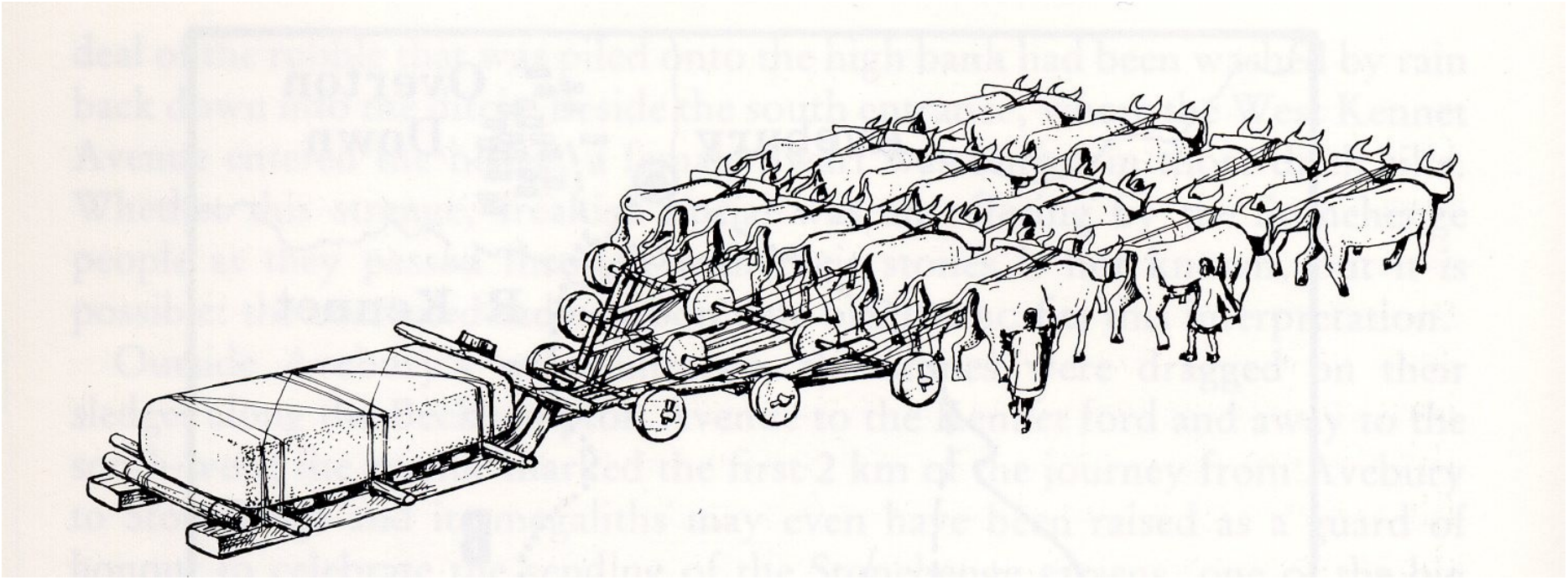
given the known policy functions from last iteration

$$\left(k^1_{t+2}(a_{t+1}, k_{t+1}), \ldots, k^N_{t+2}(a_{t+1}, k_{t+1}), \lambda_{t+1}(a_{t+1}, k_{t+1})\right)$$ → **Interpolant**

evaluated at next periods state

$$(a^1_t, \ldots, a^N_t, k^1_t, \ldots, k^N_t), \text{ where } a^j_{t+1} = (a^j_t)^\rho \cdot e^{\sigma(e_t + e^j_t)}.$$

“To pull a bigger wagon, it is easier to add more oxen than to grow a gigantic ox” (Skjellum et al. 1999)



“To pull a bigger wagon, it is easier to add more oxen than to grow a gigantic ox” (Skjellum et al. 1999)

Left: Piz Daint (CSCS)**~6.7 Petaflops** (115'984 cores)
Right: UZH Schrödinger Cluster , ~5k Cores available
→ **ACCESS ON BOTH MACHINES GRANTED – PI on Schrödinger**

Bluewaters (Ken's Allocation)

Bluewaters Supercomputer, 13.3 Petaflops - University of Illinois

Execution time

“My application runs too slow” / “My problem is too big to be solved on a Laptop”

!!!Once, AND ONLY ONCE serial code is fully optimized!!!

→ Parallelization:

→ Shared memory (OpenMP - https://computing.llnl.gov/tutorials/openMP/)
“expensive & limited hardware / cheap programming effort

→ Distributed memory (MPI - https://computing.llnl.gov/tutorials/mpi/)
”cheap” hardware / “expensive” programming effort.

→ Recently: **Hybrid parallelization schemes** → combine (MPI/OpenMP/GPUs)

Speedup, Efficiency & Amdahl's Law

T(p,N) := time to solve problem of total size N on p processors.

Parallel speedup: $S(p,N) = T(1,N)/T(p,N)$

→ Compute same problem with more processors in **shorter time.**

Parallel Efficiency: $E(p,N) = S(p,N)/p$

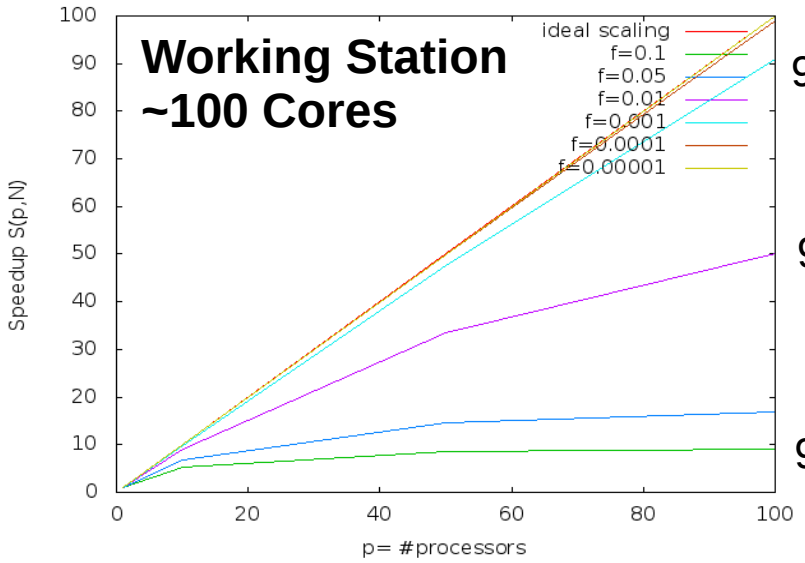
Amdahl's Law: $T(p,N) = f * T(1,N) + \mathbf{(1-f)\ T(1,N)/p}$

f...sequential part of the code that can not be done in parallel

$$S(p,N) = T(1,N)/T(p,N) = 1 / (f + (1-f)/p)$$

For p → infinity, speedup is limited by $S(p,N) < 1/f$

Amdahl's Law: Scaling is tough

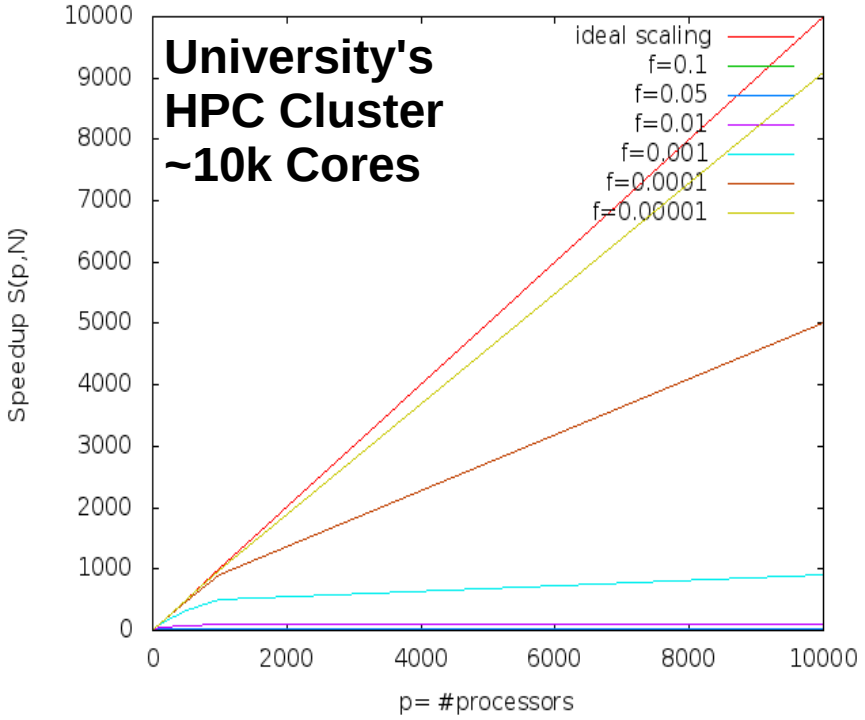


99.9% Parallel

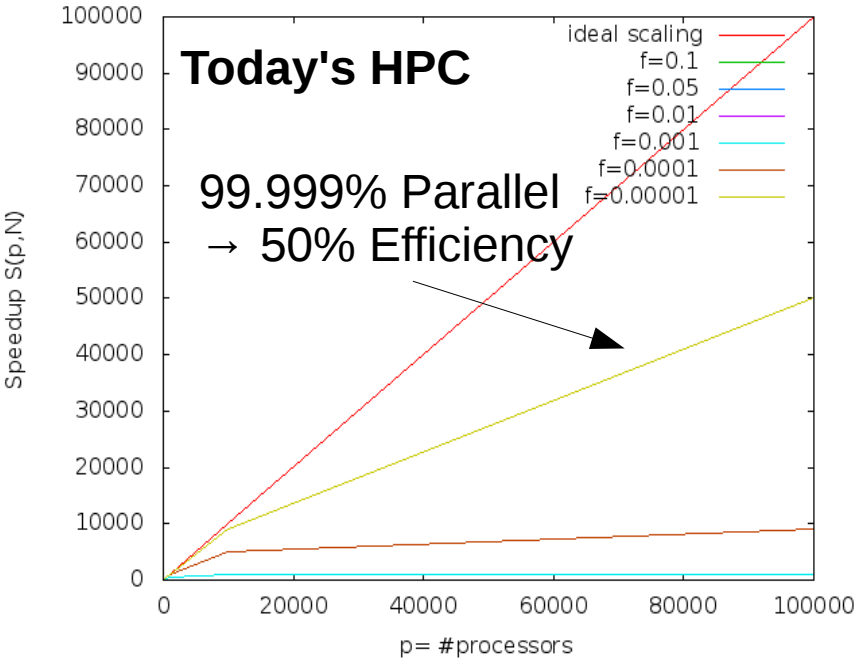
99% Parallel

90% Parallel

For p → infinity:
Speedup is limited by S(p,N) < 1/f



Jniversity of Z



Parallel time iteration algorithm

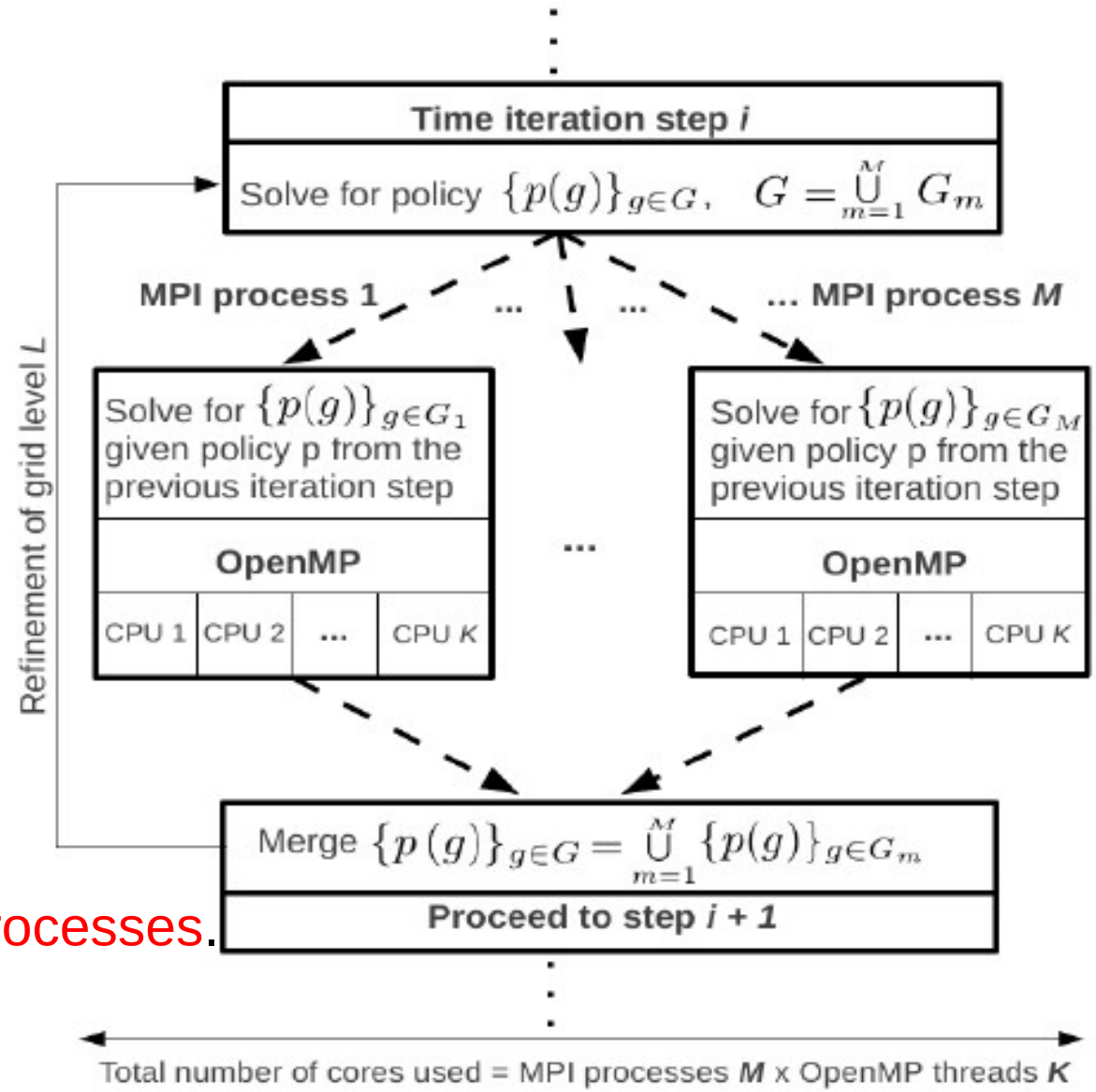
-Our implementation:
Hybrid parallel (MPI & OpenMP).

-newly generated points are distributed via MPI

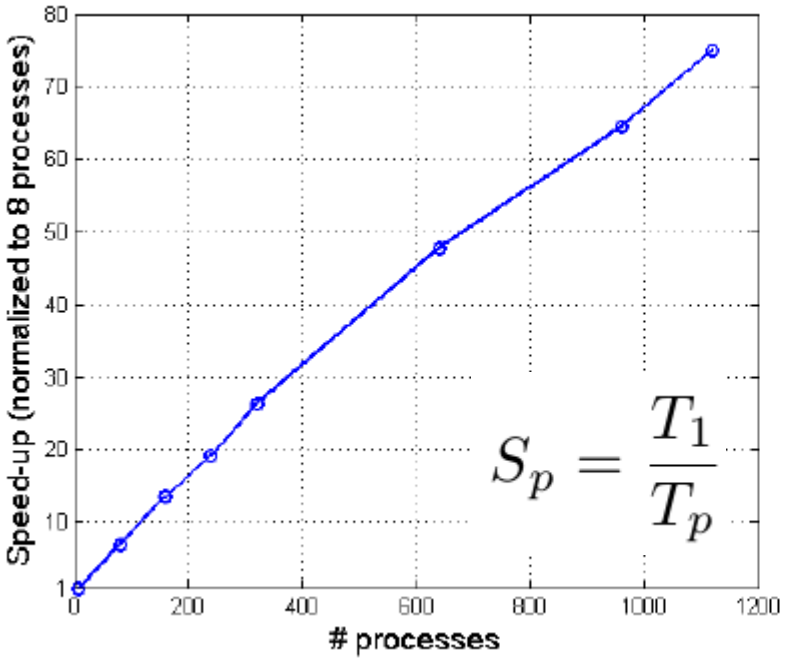
Nonlinear equations locally solved on each node/MPI process by a combination of **IPOPT & PARDISO**
(Waechter & Biegler (2006), Schenk et al. (2008)).

In parallel: `messy' !

→ policy from previous iteration has to be visible on all MPI processes.
→ we have to ensure some sort of `load balancing'.



Scaling & Performance



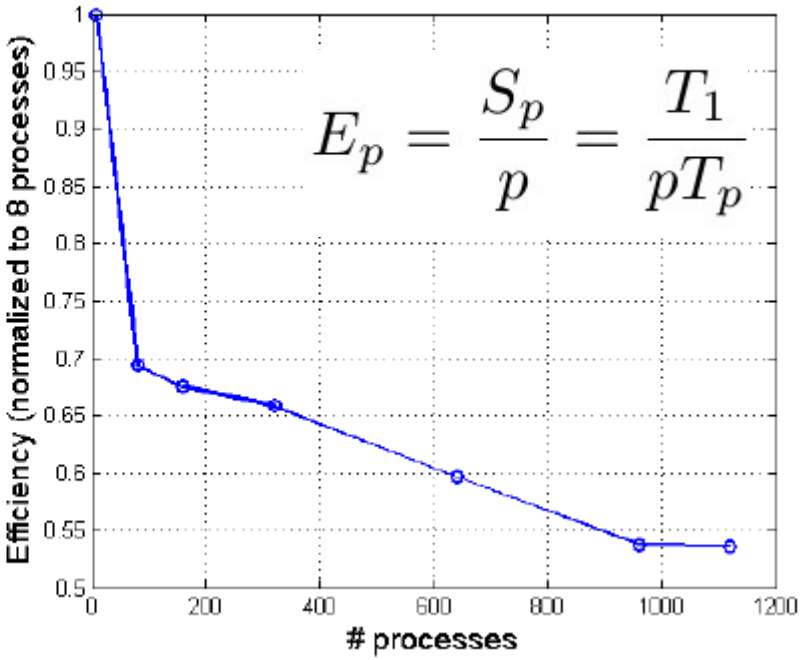


Fig.: Left panel: strong scaling of the code. Right: Efficiency.
Problem: **one timestep of a 10d** IRBC model with fixed sparse grid (level 3).
→ Problem size still fits a laptop → can easily get better scaling for larger problems!
→ The setting here is ~99.95% parallel.

Scaling & Performance II

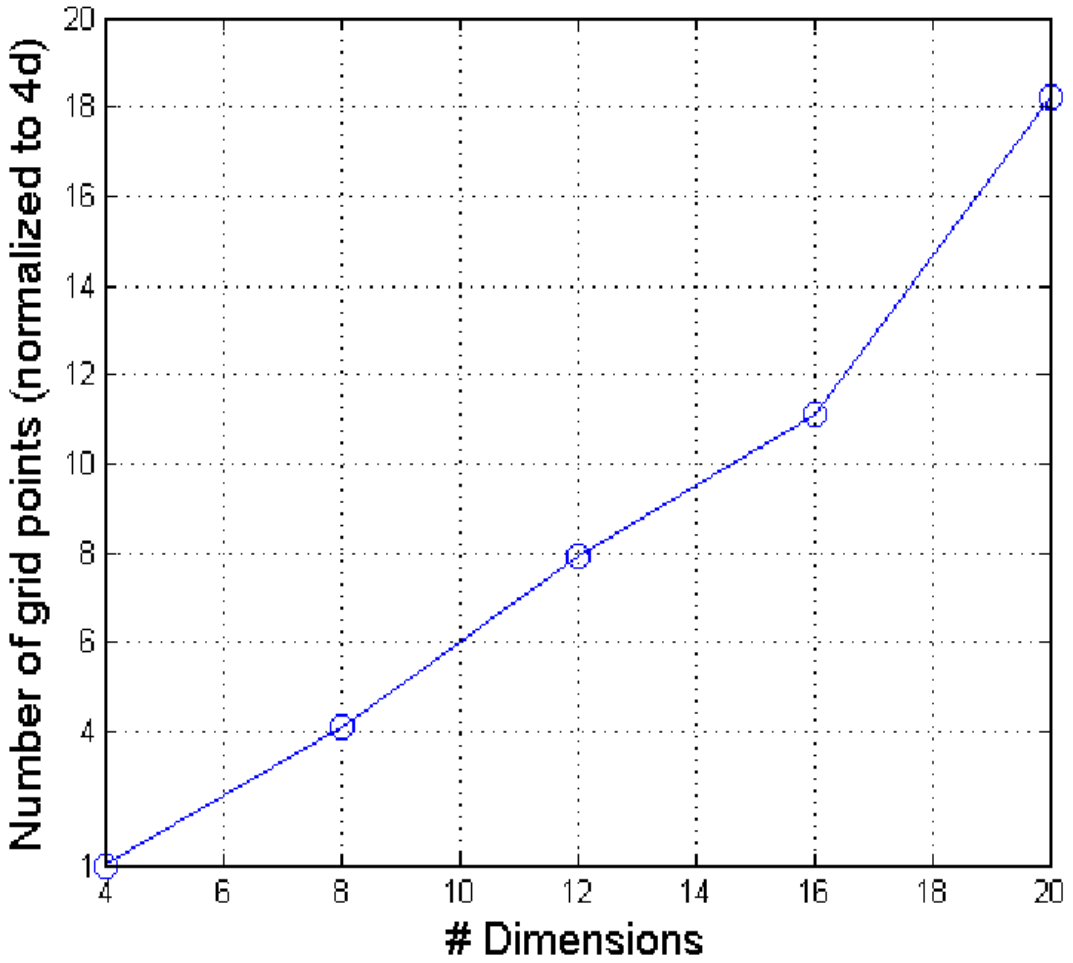


Fig.: Number of grid points grow ~O(d) with increasing dimensionality.
Test: **one timestep**, **adaptive sparse grid** with **refinement criterion *O(10 $^{-3}$)***.

Errors: Results – IRBC

Increase Level
→ Errors get better

Dimension	Level	Points	Max. Error	Avg. Error
4	3	137	-2.82	-3.68
4	4	401	-2.98	-4.19
4	5	1105	-3.19	-4.24
4	6	2929	-3.28	-4.55

Increase Dimension
→ errors remain
of same quality

Dimension	Level	Points	Max. Error	Avg. Error
4	2	41	-2.80	-3.68
8	2	145	-2.96	-3.25
12	2	313	-2.64	-3.27
16	2	545	-2.59	-3.29
20	2	841	-2.58	-3.29
22	2	1013	-2.60	-3.29
24	2	1201	-2.55	-3.29
4	3	137	-2.82	-3.68
8	3	849	-3.04	-3.83
12	3	2649	-2.71	-3.78
16	3	6049	-2.72	-3.80

IRBC with `kinks'

-Want to demonstrate that we are able to handle **non-smooth behaviour**.

-We include **irreversible investment** in the IRBC:
→ **installed capital cannot be consumed or moved to another country**.

Model:

→ for each country, we get an irreversibility constraint: $k^j_{t+1} \geq k^j_t \cdot (1-\delta)$

→ the equilibrium conditions are now given by:

N irrev. constraints: $k^j_{t+1} - k^j_t(1-\delta) \geq 0, \mu^j_t \geq 0, \left(k^j_{t+1} - k^j_t(1-\delta)\right) \cdot \mu^j_t = 0$ (→ `kinks')

N Euler eqs.:
$$\lambda_t \cdot \left[1 + \phi \cdot g^j_{t+1}\right] - \mu^j_t$$
$$- \beta \cdot \mathbb{E}_t \left\{ \lambda_{t+1} \left[a^j_{t+1} \cdot A \cdot \alpha \cdot (k^j_{t+1})^{\alpha-1} + (1-\delta) + \frac{\phi}{2} \cdot g^j_{t+2} \cdot \left(g^j_{t+2} + 2\right) \right] \right\} = 0$$

1 agg. constr. $\sum_{j=1}^{N} \left(a^j_t \cdot A \cdot (k^j_t)^\alpha + k^j_t \cdot \left((1-\delta) - \frac{\phi}{2} \cdot (g^j_{t+1})^2 \right) - k^j_{t+1} - \left(\frac{\lambda_t}{\tau_j} \right)^{-\gamma^j} \right) = 0$

IRBC with irreversible investment

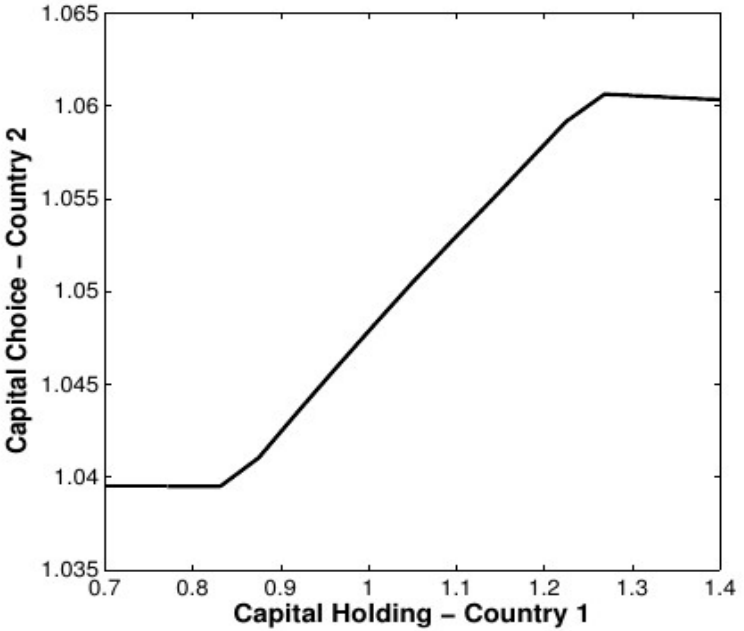


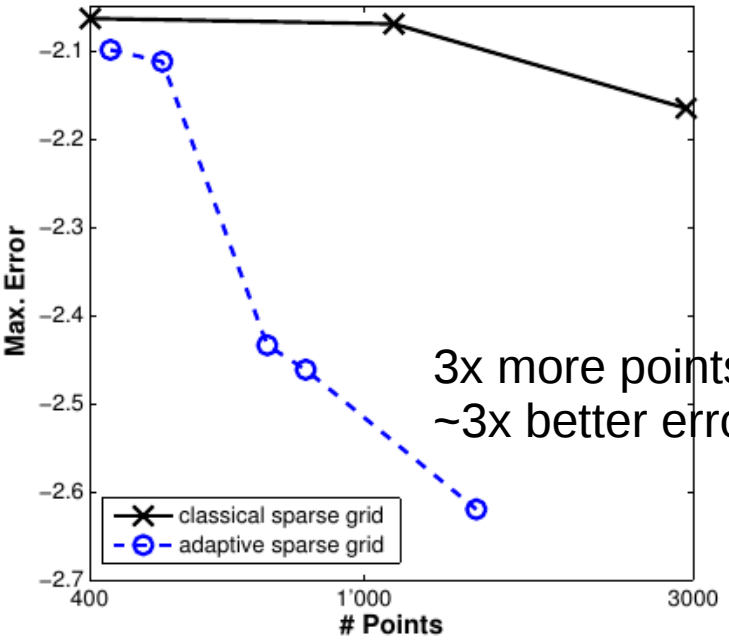
Fig.: Capital choice of country 2 as a function of capital holding of country 1.
All other state variables of this model are kept fixed at steady state (2N = 4d).
The 4-d policy function was interpolated on an adaptive sparse grid (ε = 0.0033).

Note: **kink is (2N -1) - dimensional hypersurface in 2N - dim state space.**

Non-smooth IRBC in 4-d

Dimension	Level	Points	Max. Error	Avg. Error
4	3	137	-1.50	-2.70
4	4	401	-2.06	-2.86
4	5	1'105	-2.07	-2.96
4	6	2'929	-2.17	-3.07

Tab.: Classical sparse grid. **errors hardly improve!**



3x more points
~3x better error

ϵ	Points	Max. Error	Avg. Error	Max. Level Reached
0.0150	429	-2.10	-3.05	5 (1'105)
0.0100	510	-2.11	-3.11	6 (2'929)
0.0067	724	-2.43	-3.20	7 (7'537)
0.0050	823	-2.46	-3.19	8 (18'945)
0.0033	1'454	-2.62	-3.27	10 (113'409)

Tab.: Adaptive sparse grid; **puts resolution where needed!**

Fig.: Sparse grid vs. adaptive sparse grid (ε varies in [0.01 : 0.0033])

Locally mimic a grid of refinement level 10

Non-smooth IRBC in 4-d (II)

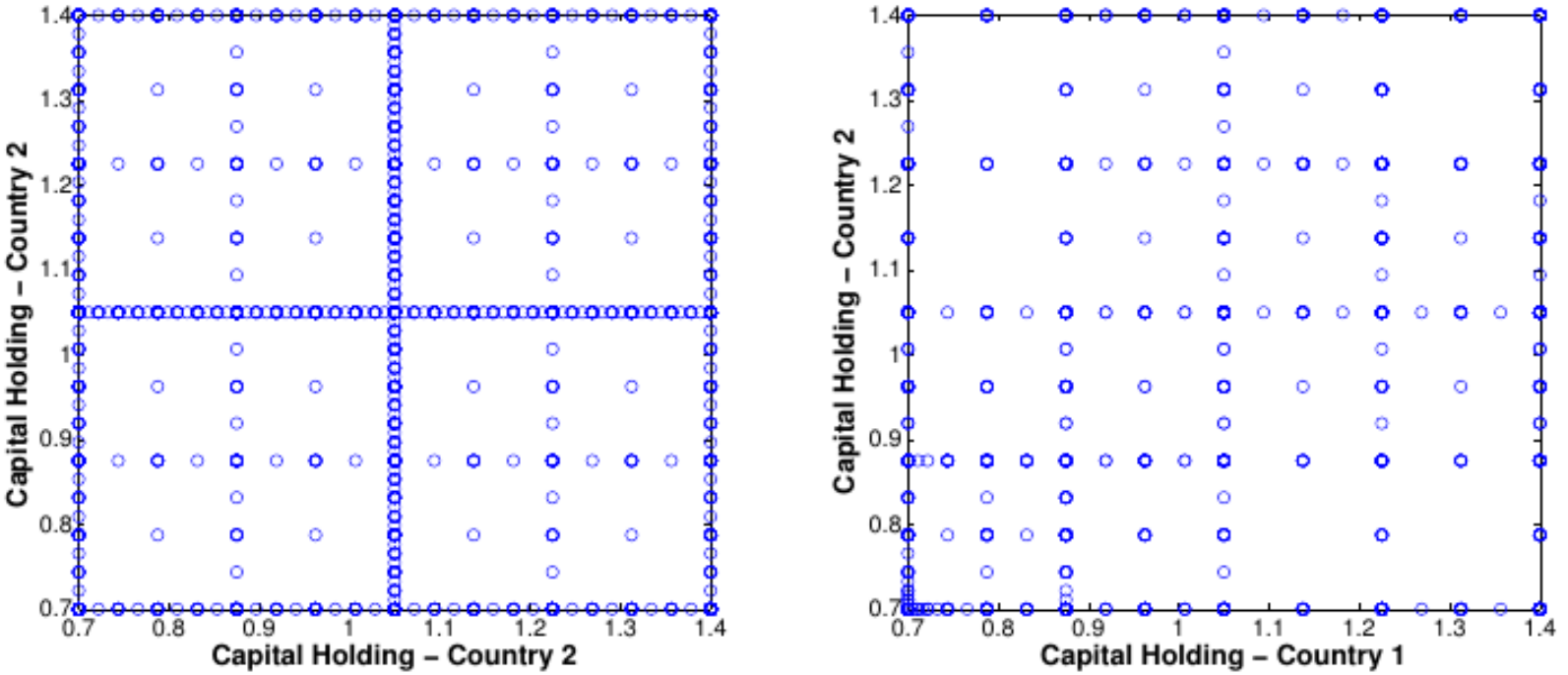


Fig.: 2-d projections of two different grids.
Left: `classical' sparse grid of level 6 (2'929 points),
Right: adaptive grid with refinement threshold ε = 0.0033 (1'454 points).
The x-axis shows capital holding of country 1, the y-axis shows capital holding of country 2, while the productivities of the two countries are kept fixed at their unconditional means.

Non-smooth IRBC in high dimensions

Dimension	Points	Max. Error	Avg. Error	$\lvert V_6^{S,CC} \rvert$
4	510	-2.15	-3.11	2'929
6	1'950	-1.87	-2.71	15'121
8	5'643	-1.63	-2.51	56'737

Tab.: ε = 0.01; maximum refinement level = 6.

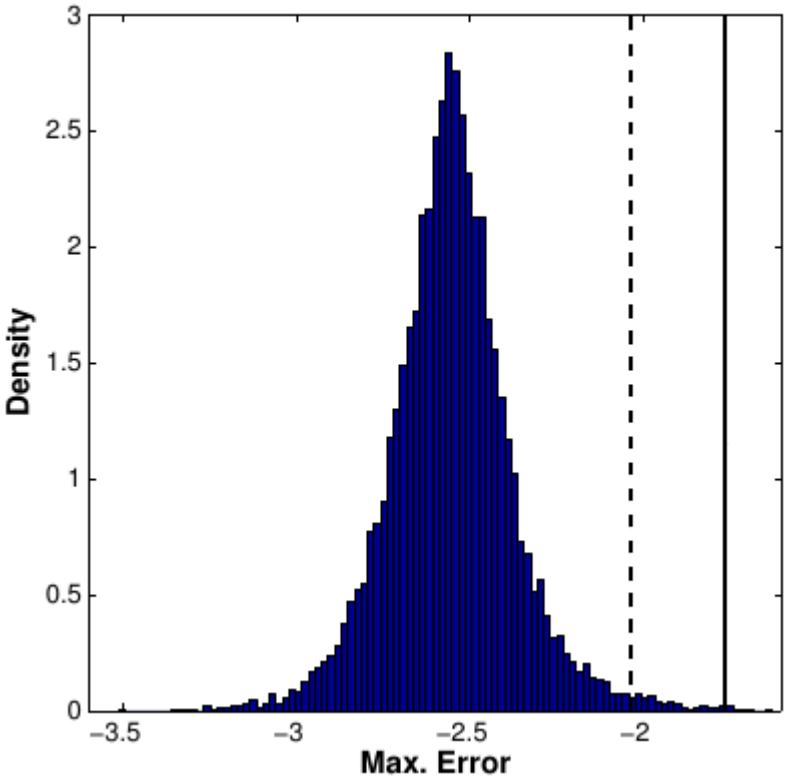


Fig.: Distribution of maximum approximation errors for the non-smooth IRBC model with 8-d state space.

→ **kink is (2N -1) - dim hypersurface in 2N- dim state space.**

→ **gets harder to get good errors with increasing dimensions.**

→ **Mitigate this effect by choosing a larger maximum refinement level.**

Conclusion & Outlook

- **Method perfectly suited to solve high dimensional dynamic models with large amount of heterogeneity! (Method: Scalable & Flexible !!!)**
- First time adaptive sparse grids are applied to high-dimensional economic models
- First ones to solve dynamic models on HPC systems with hybrid parallelism
- Method also well-suited for Dynamic Programming.
- **Pull bigger wagon** → we showed that the kink model does favour adaptivity.

  → solve models/ hard problems that contain features which favour adaptivity.

A I: 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 $\int_\Omega \phi_{l,i}(\vec{x})\, d\vec{x} = J_{\vec{l},\vec{i}}$

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

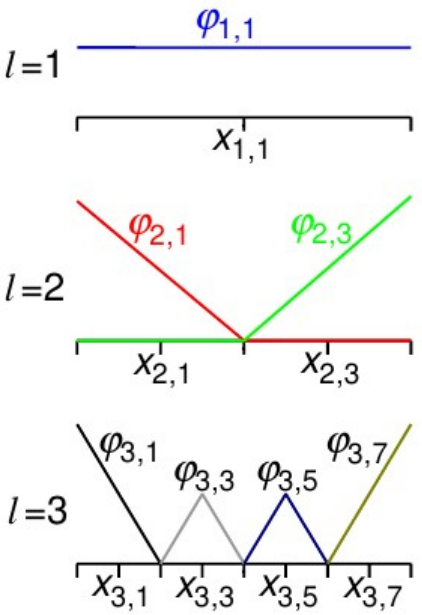
A II: Treatment of non-zero Boundaries

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!



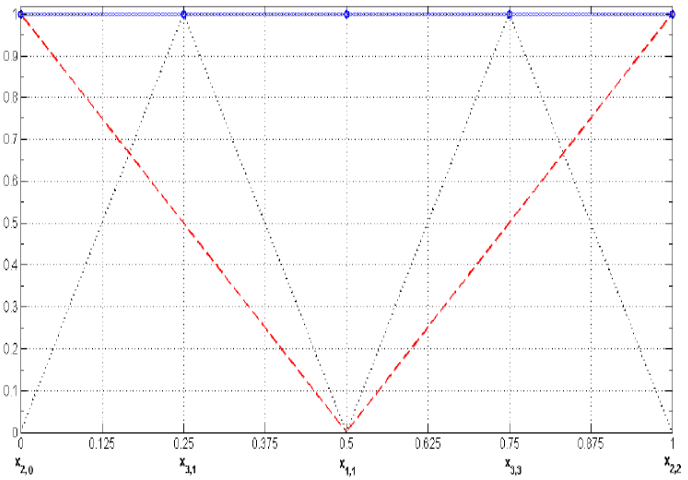


Fig.: Example of modified 1d-basis functions According to Pflüger (2010), which are extrapolating towards the boundary (**left**). They are constant on level 1 and **"folded-up"** if adjacent to the boundary on all other levels. **Right**: "Clenshaw-Curtis" basis.