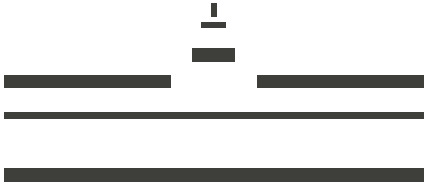


Westfälische
Wilhelms-Universität
Münster

Hierarchical Approximate Proper Orthogonal Decomposition

Christian Himpe, Tobias Leibner and Stephan Rave

living.knowledge
WWU Münster

MPI Magdeburg
March 13, 2018

Outline

- Reduced Basis Methods and POD.

- HAPOD – Hierarchical Approximate POD.

Outline

- Reduced Basis Methods and POD.
- HAPOD – Hierarchical Approximate POD.
- Model Order Reduction with pyMOR.

Reduced Basis Methods and POD

RB for Nonlinear Evolution Equations

Full order model

For given parameter $\mu \in \mathcal{P}$, find $u_\mu(t) \in V_h$ s.t.

$$\partial_t u_\mu(t) + \mathcal{L}_\mu(u_\mu(t)) = 0, \quad u_\mu(0) = u_0,$$

where $\mathcal{L}_\mu : \mathcal{P} \times V_h \to V_h$ is a nonlinear finite volume operator.

RB for Nonlinear Evolution Equations

Full order model

For given parameter $\mu \in \mathcal{P}$, find $u_\mu(t) \in V_h$ s.t.

$$\partial_t u_\mu(t) + \mathcal{L}_\mu(u_\mu(t)) = 0, \quad u_\mu(0) = u_0,$$

where $\mathcal{L}_\mu : \mathcal{P} \times V_h \to V_h$ is a nonlinear finite volume operator.

Reduced order model

For given $V_N \subset V_h$, let $u_{\mu,N}(t) \in V_N$ be given by Galerkin proj. onto V_N, i.e.

$$\partial_t u_{\mu,N}(t) + P_{V_N}(\mathcal{L}_\mu(u_{\mu,N}(t))) = 0, \quad u_{\mu,N}(0) = P_{V_N}(u_0),$$

where $P_{V_N} : V_h \to V_N$ is orthogonal proj. onto V_N.

RB for Nonlinear Evolution Equations

Full order model

For given parameter $\mu \in \mathcal{P}$, find $u_\mu(t) \in V_h$ s.t.

$$\partial_t u_\mu(t) + \mathcal{L}_\mu(u_\mu(t)) = 0, \quad u_\mu(0) = u_0,$$

where $\mathcal{L}_\mu : \mathcal{P} \times V_h \to V_h$ is a nonlinear finite volume operator.

Reduced order model

For given $V_N \subset V_h$, let $u_{\mu,N}(t) \in V_N$ be given by Galerkin proj. onto V_N, i.e.

$$\partial_t u_{\mu,N}(t) + P_{V_N}(\mathcal{L}_\mu(u_{\mu,N}(t))) = 0, \quad u_{\mu,N}(0) = P_{V_N}(u_0),$$

where $P_{V_N} : V_h \to V_N$ is orthogonal proj. onto V_N.

- Still expensive to evaluate projected operator $P_{V_N} \circ \mathcal{L}_\mu : V_N \longrightarrow V_h \longrightarrow V_N$
  $\Longrightarrow$ use hyper-reduction (e.g. empirical interpolation).

Basis Generation

Offline phase

Basis for V_N is computed from **solution snapshots** $u_{\mu_s}(t)$ of full order problem via:

- Proper Orthogonal Decomposition (POD)
- POD-Greedy (= greedy search in μ + POD in t)

Basis Generation

Offline phase

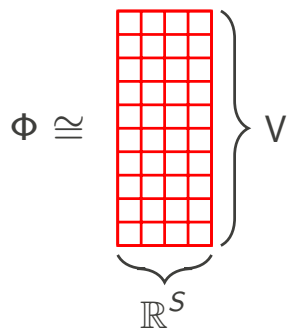
Basis for V_N is computed from **solution snapshots** $u_{\mu_s}(t)$ of full order problem via:

- Proper Orthogonal Decomposition (POD)
- POD-Greedy (= greedy search in μ + POD in t)

POD (a.k.a. PCA, Karhunen–Loève decomposition)

Given Hilbert space V, $\mathcal{S} := \{v_1, \dots, v_S\} \subset V$, the k-th POD mode of $\mathcal{S}$ is the k-th left-singular vector of the mapping

$$\Phi : \mathbb{R}^S \to V, \quad e_s \to \Phi(e_s) := v_s$$



Optimality of POD

Let V_N be the linear span of first N POD modes, then:

$$\sum_{s\in\mathcal{S}} \|s - P_{V_N}(s)\|^2 = \sum_{m=N+1}^{|\mathcal{S}|} \sigma_m^2 = \min_{\substack{X\subset V \\ \dim X \leq N}} \sum_{s\in\mathcal{S}} \|s - P_X(s)\|^2$$

Example: RB Approximation of Li-Ion Battery Models

ulm university universität ulm

Institute of Technical Thermodynamics

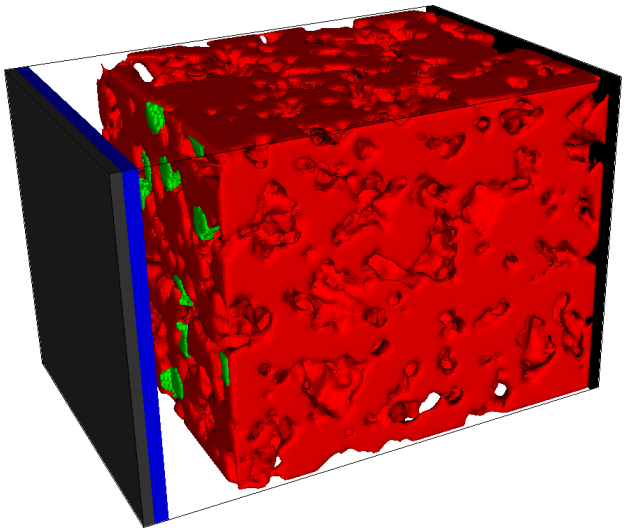
DLR

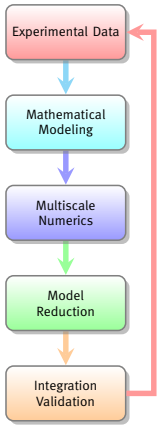
MULTIBAT

SPONSORED BY THE
Federal Ministry of Education and Research

Fraunhofer ITWM

Westfälische Wilhelms-Universität Münster





MULTIBAT: Gain understanding of degradation processes in rechargeable Li-Ion Batteries through mathematical modeling and simulation.

- Focus: Li-Plating.
- Li-plating initiated at interface between active particles and electrolyte.
- Need microscale models which resolve active particle geometry.
- Very large nonlinear discrete models.

Example: Numerical Results

Full order model:

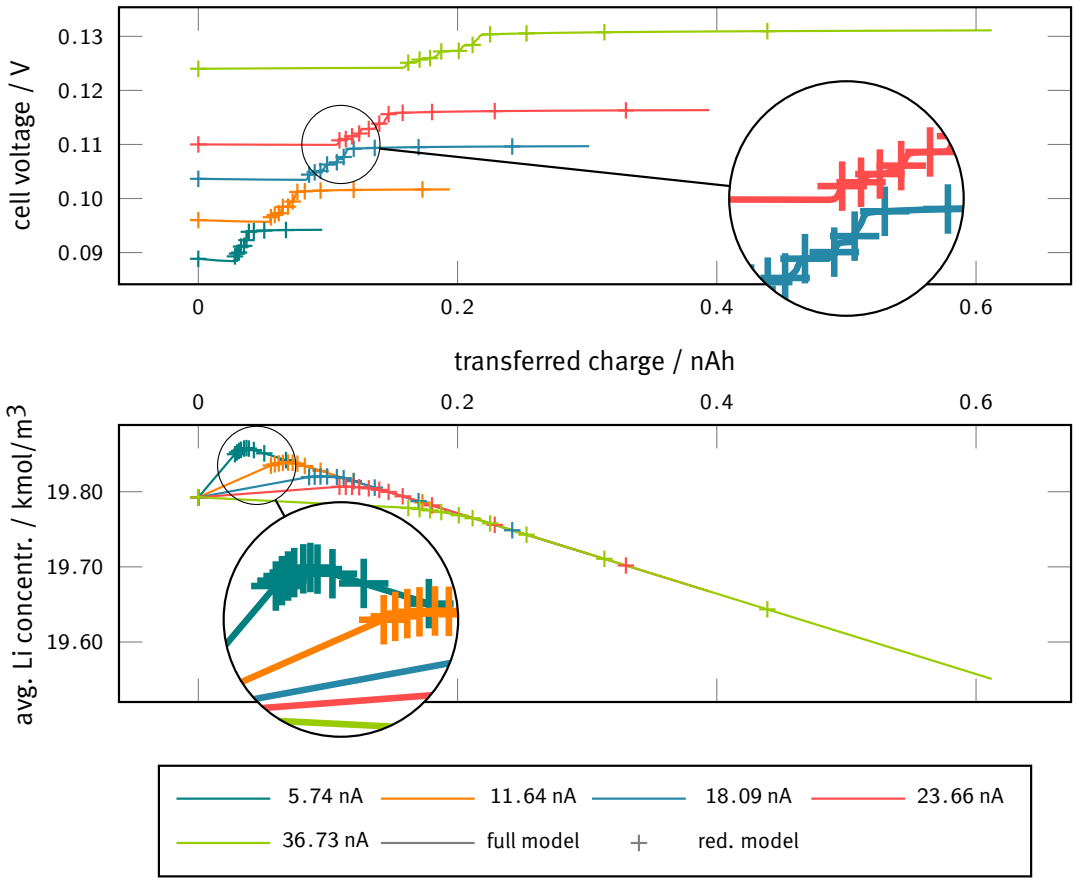
- 2.920.000 DOFs
- Simulation time: $\approx$ 13h

Model order reduction:

- Snapshots: 3
- Reduction time: $\approx$ 9h

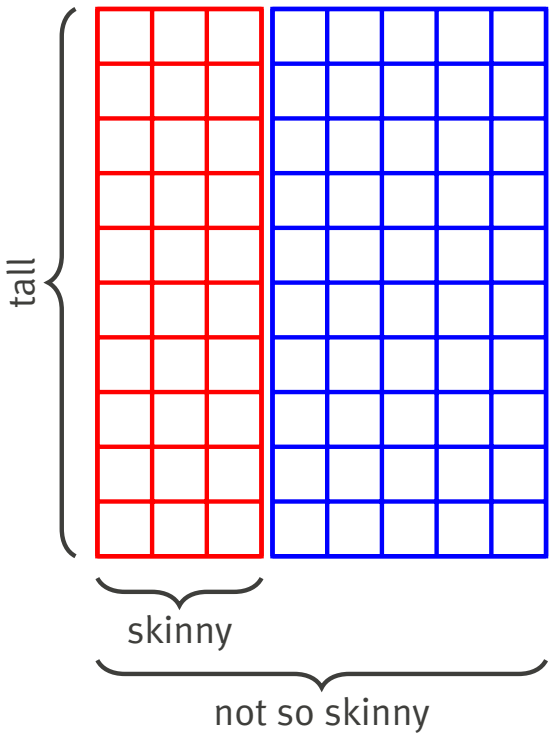
Reduced order model:

- dim $V_N = 145$
- $M = 1484$
- Rel. err.: $< 1.5 \cdot 10^{-3}$
- Simulation time: $\approx$ 5m
- Speedup: **154**



HAPOD – Hierarchical Approximate POD

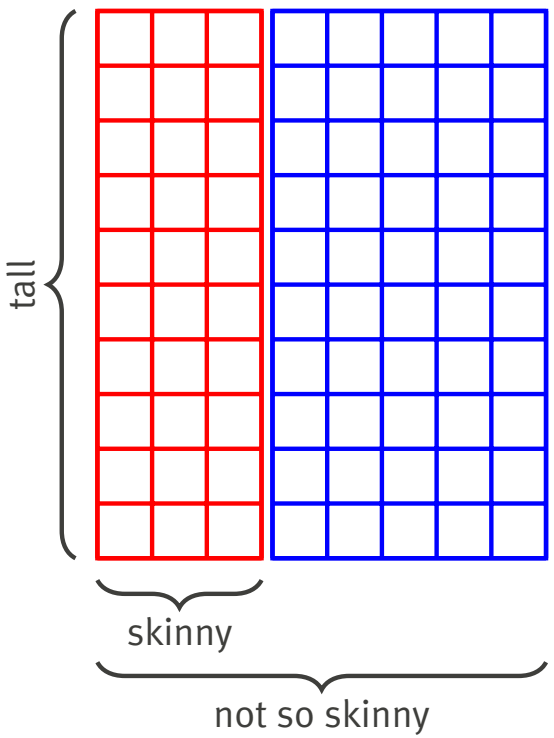
Are your tall and skinny matrices not so skinny anymore?



POD of large snapshot sets:

- large computational effort
- hard to parallelize
- data $>$ RAM $\Longrightarrow$ disaster

Are your tall and skinny matrices not so skinny anymore?



POD of large snapshot sets:

- large computational effort
- hard to parallelize
- data $>$ RAM $\Longrightarrow$ disaster

Solution: PODs of PODs!

Disclaimer

- You might have done this before.

Disclaimer

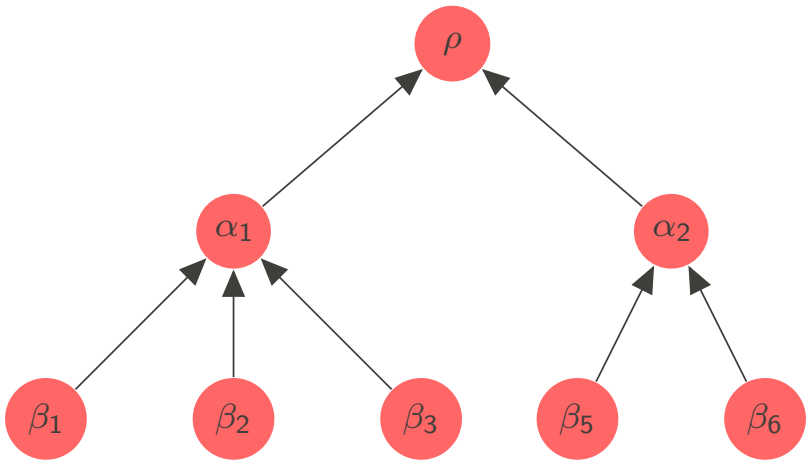
- You might have done this before.

- Others have done it before – often well-hidden in a paper on entirely different topic.
  We are aware of:
  [Qu, Ostrouchov, Samatova, Geist, 2002], [Paul-Dubois-Taine, Amsallem, 2015],
  [Brands, Mergheim, Steinmann, 2016], [Iwen, Ong, 2017].

Disclaimer

- You might have done this before.
- Others have done it before – often well-hidden in a paper on entirely different topic.
  We are aware of:
  [Qu, Ostrouchov, Samatova, Geist, 2002], [Paul-Dubois-Taine, Amsallem, 2015],
  [Brands, Mergheim, Steinmann, 2016], [Iwen, Ong, 2017].
- Our contributions:
  1. Formalization for arbitrary trees of worker nodes.
  2. Extensive theoretical error and performance analysis.
  3. A recipe for selecting local truncation thresholds.
  4. Extensive numerical experiments for different application scenarios.
- Can be trivially extended to low-rank approximation of snapshot matrix by keeping track of right-singular vectors.

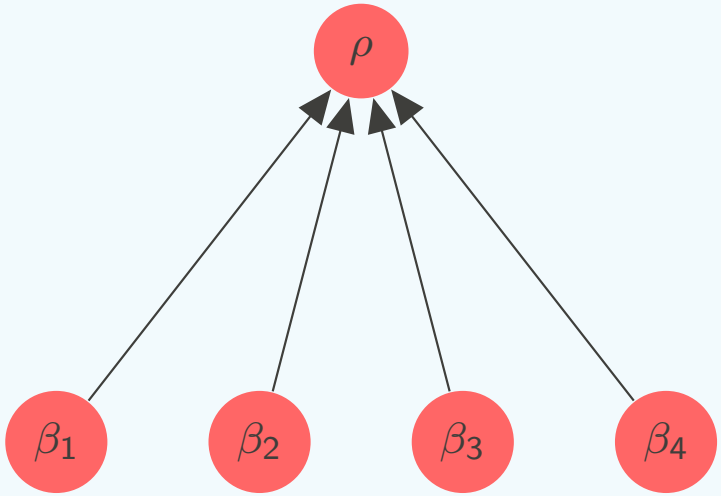
HAPOD – Hierarchical Approximate POD



- Input: Assign snapshot vectors to leaf nodes β_i as input.
- At each node α:
  1. Perform POD of input vectors with given local ℓ^2-error tolerance $\varepsilon(\alpha)$.
  2. Scale POD modes by singular values.
  3. Send scaled modes to parent node as input.
- Output: POD modes at root node ρ.

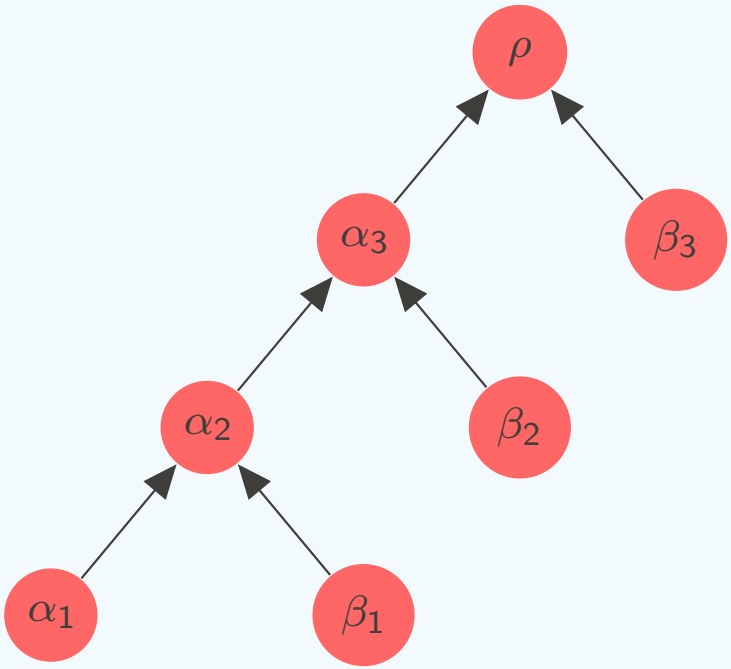
HAPOD – Special Cases

Distributed HAPOD



- Distributed, communication avoiding POD computation.

Incremental HAPOD



- On-the-fly compression of large trajectories.

HAPOD – Some Notation

Trees

$\mathcal{T}$	the tree
$\rho_{\mathcal{T}}$	root node
$\mathcal{N}_{\mathcal{T}}(\alpha)$	nodes of $\mathcal{T}$ below or equal node α
$\mathcal{L}_{\mathcal{T}}$	leafs of $\mathcal{T}$
$L_{\mathcal{T}}$	depth of $\mathcal{T}$

HAPOD

$\mathcal{S}$	snapshot set
$D : \mathcal{S} \to \mathcal{L}_{\mathcal{T}}$	snapshot to leaf assignment
$\varepsilon(\alpha)$	error tolerance at α
$\lvert \mathrm{HAPOD}[\mathcal{S}, \mathcal{T}, D, \varepsilon](\alpha) \rvert$	number of HAPOD modes at α
$\lvert \mathrm{POD}(\mathcal{S}, \varepsilon) \rvert$	number of POD modes for error tolerance ε
P_α	orth. proj. onto HAPOD modes at α
$\widetilde{\mathcal{S}}_\alpha$	snapshots at leafs below α

HAPOD – Theoretical Analysis

Theorem (Error bound[1])

$$\sum_{s \in \widetilde{\mathcal{S}}_\alpha} \|s - P_\alpha(s)\|^2 \leq \sum_{\gamma \in \mathcal{N}_\mathcal{T}(\alpha)} \varepsilon(\gamma)^2.$$

[1] For special cases in appendix of [Paul-Dubois-Taine, Amsallem, 2015].

HAPOD – Theoretical Analysis

Theorem (Error bound[1])

$$\sum_{s \in \widetilde{\mathcal{S}}_\alpha} \|s - P_\alpha(s)\|^2 \leq \sum_{\gamma \in \mathcal{N}_\mathcal{T}(\alpha)} \varepsilon(\gamma)^2.$$

Theorem (Mode bound)

$$\left|\mathrm{HAPOD}[\mathcal{S}, \mathcal{T}, D, \varepsilon](\alpha)\right| \leq \left|\mathrm{POD}\left(\widetilde{\mathcal{S}}_\alpha, \varepsilon(\alpha)\right)\right|.$$

[1] For special cases in appendix of [Paul-Dubois-Taine, Amsallem, 2015].

HAPOD – Theoretical Analysis

Theorem (Error bound[1])

$$\sum_{s \in \widetilde{\mathcal{S}}_\alpha} \|s - P_\alpha(s)\|^2 \leq \sum_{\gamma \in \mathcal{N}_\mathcal{T}(\alpha)} \varepsilon(\gamma)^2.$$

Theorem (Mode bound)

$$\left|\mathrm{HAPOD}[\mathcal{S}, \mathcal{T}, D, \varepsilon](\alpha)\right| \leq \left|\mathrm{POD}\left(\widetilde{\mathcal{S}}_\alpha, \varepsilon(\alpha)\right)\right|.$$

But how to choose ε in practice?

- Prescribe error tolerance ε^* for final HAPOD modes.
- Balance quality of HAPOD space (number of additional modes) and computational efficiency ($\omega \in [0, 1]$).
- Number of input snapshots should be irrelevant for error measure (might be even unknown a priori). Hence, control ℓ^2-*mean* error $\frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} \|s - P_{\rho\mathcal{T}}(s)\|^2$.

[1] For special cases in appendix of [Paul-Dubois-Taine, Amsallem, 2015].

HAPOD – Theoretical Analysis

Theorem (ℓ^2-mean error and mode bounds)

Choose local POD error tolerances $\varepsilon(\alpha)$ for ℓ^2-approximation error as:

$$\varepsilon(\rho_{\mathcal{T}}) := \sqrt{|S|} \cdot \omega \cdot \varepsilon^*, \qquad \varepsilon(\alpha) := \sqrt{\widetilde{\mathcal{S}}_\alpha} \cdot (L_{\mathcal{T}} - 1)^{-1/2} \cdot \sqrt{1 - \omega^2} \cdot \varepsilon^*.$$

Then:

$$\frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} \|s - P_{\rho_{\mathcal{T}}}(s)\|^2 \leq \varepsilon^{*2} \quad \textit{and} \quad |\operatorname{HAPOD}[\mathcal{S}, \mathcal{T}, D, \varepsilon]| \leq |\overline{\operatorname{POD}}(\mathcal{S}, \omega \cdot \varepsilon^*)|,$$

where $\overline{\operatorname{POD}}(\mathcal{S}, \varepsilon) := \operatorname{POD}(\mathcal{S}, |\mathcal{S}| \cdot \varepsilon)$.

Moreover:

$$\begin{aligned} |\operatorname{HAPOD}[\mathcal{S}, \mathcal{T}, D, \varepsilon](\alpha)| &\leq |\overline{\operatorname{POD}}(\widetilde{\mathcal{S}}_\alpha, (L_{\mathcal{T}} - 1)^{-1/2} \cdot \sqrt{1 - \omega^2} \cdot \epsilon^*)| \\ &\leq \min_{N \in \mathbb{N}} (d_N(\mathcal{S}) \leq (L_{\mathcal{T}} - 1)^{-1/2} \cdot \sqrt{1 - \omega^2} \cdot \epsilon^*). \end{aligned}$$

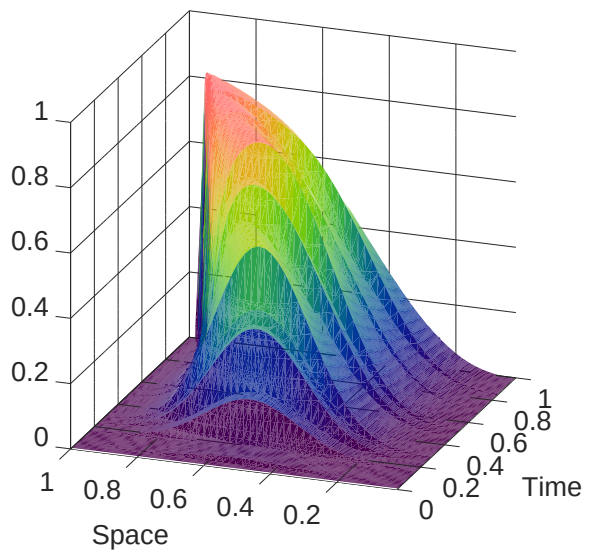
Incremental HAPOD Example

Compress state trajectory of forced inviscid Burgers equation:

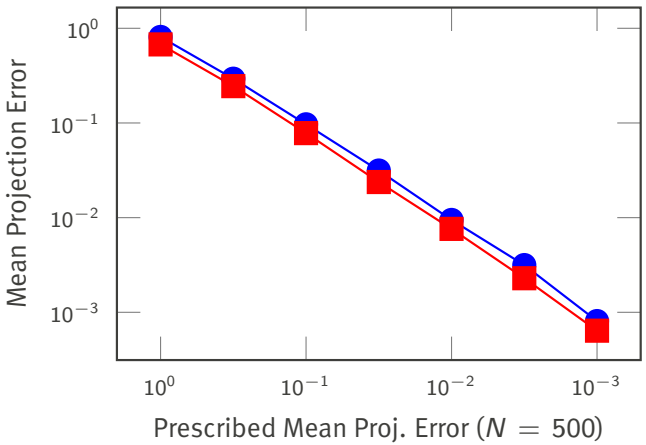
$$\partial_t z(x,t) + z(x,t) \cdot \partial_x z(x,t) = u(t) \exp(-\frac{1}{20}(x - \frac{1}{2})^2), \quad (x,t) \in (0,1) \times (0,1),$$
$$z(x,0) = 0, \qquad x \in [0,1],$$
$$z(0,t) = 0, \qquad t \in [0,1],$$

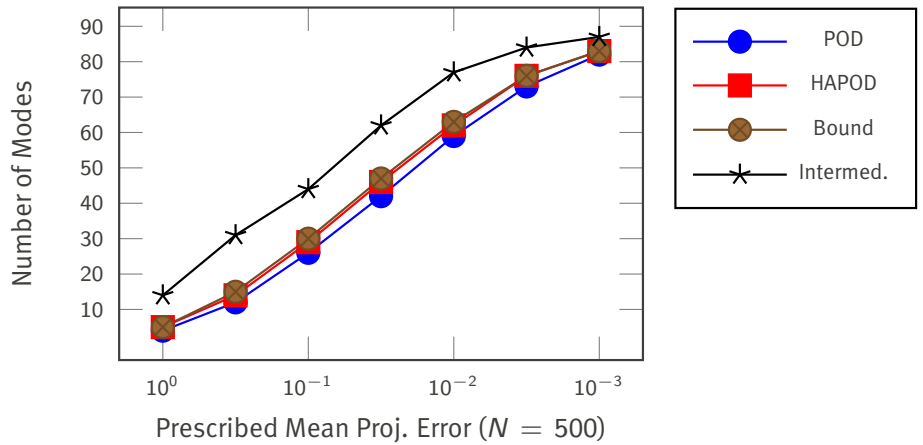
where $u(t) \in [0, 1/5]$ iid. for 0.1% random timesteps, otherwise 0.

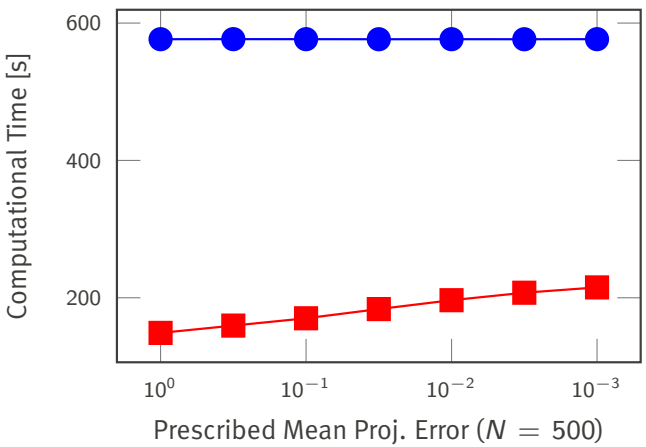
- Upwind finite difference scheme on uniform mesh with $N = 500$ nodes.
- 10^4 explicit Euler steps.
- 100 sub-PODs, $\omega = 0.75$.
- All computations on Raspberry Pi 1B single board computer (512MB RAM).

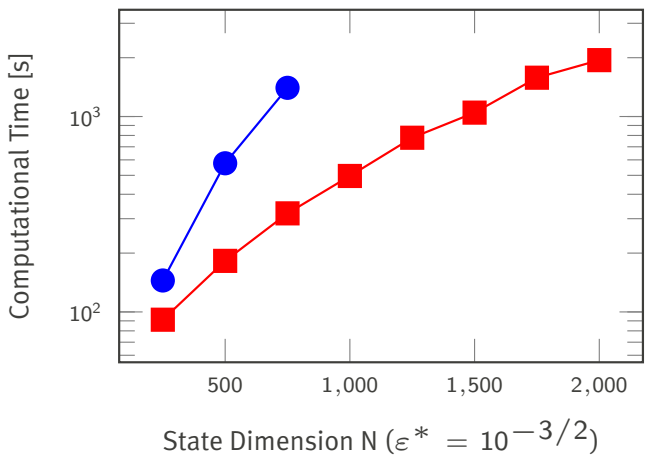


Incremental HAPOD Example







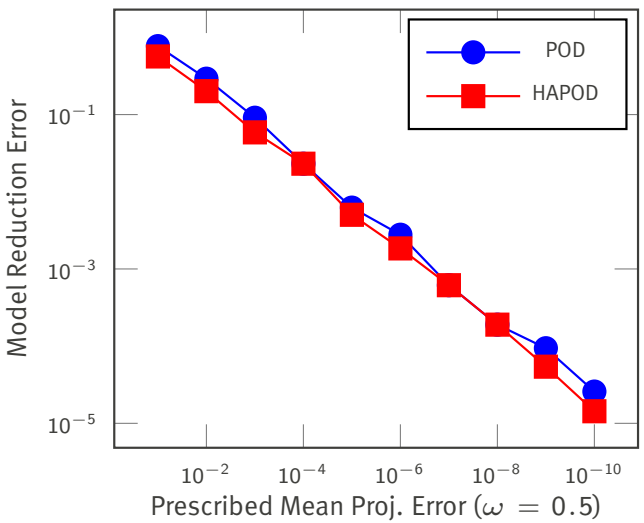


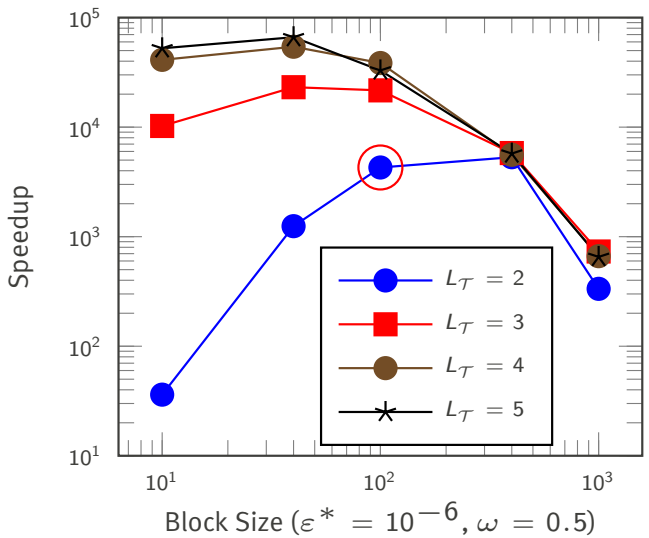
Distributed HAPOD Example

Distributed computation and POD of empirical cross Gramian:

$$\widehat{W}_{X,ij} := \sum_{m=1}^{M} \int_0^\infty \langle x_i^m(t), y_m^j(t) \rangle \, \mathrm{d}t \in \mathbb{R}^{N \times N}$$

- 'Synthetic' benchmark model[2] from MORWiki with parameter $\theta = \frac{1}{10}$.
- Partition $\widehat{W}_X$ into 100 slices of size 10.000×100.



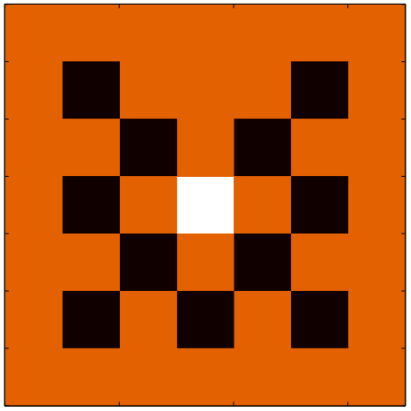


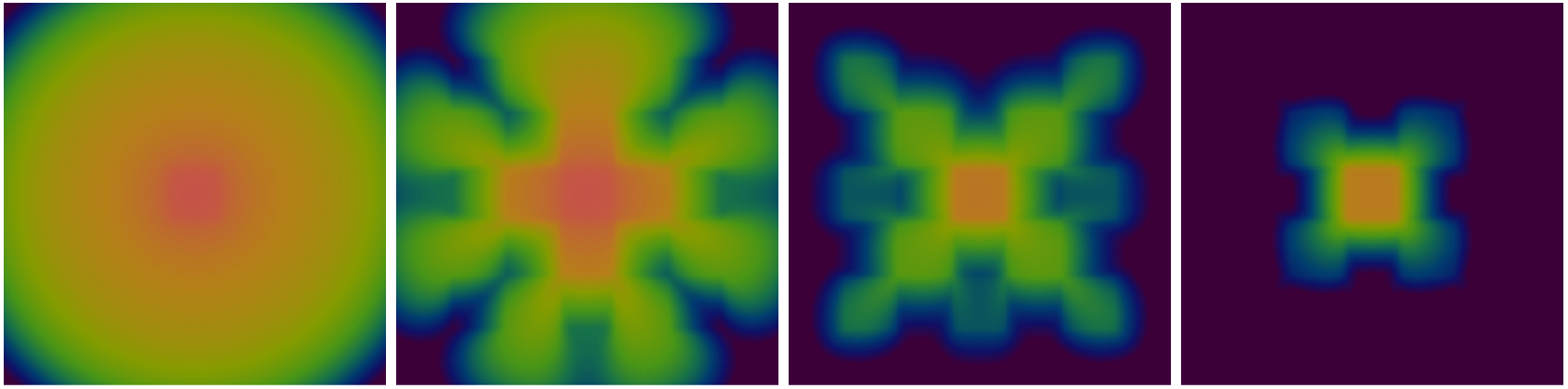
[2] See: `http://modelreduction.org/index.php/Synthetic_parametric_model`

HAPOD – HPC Example

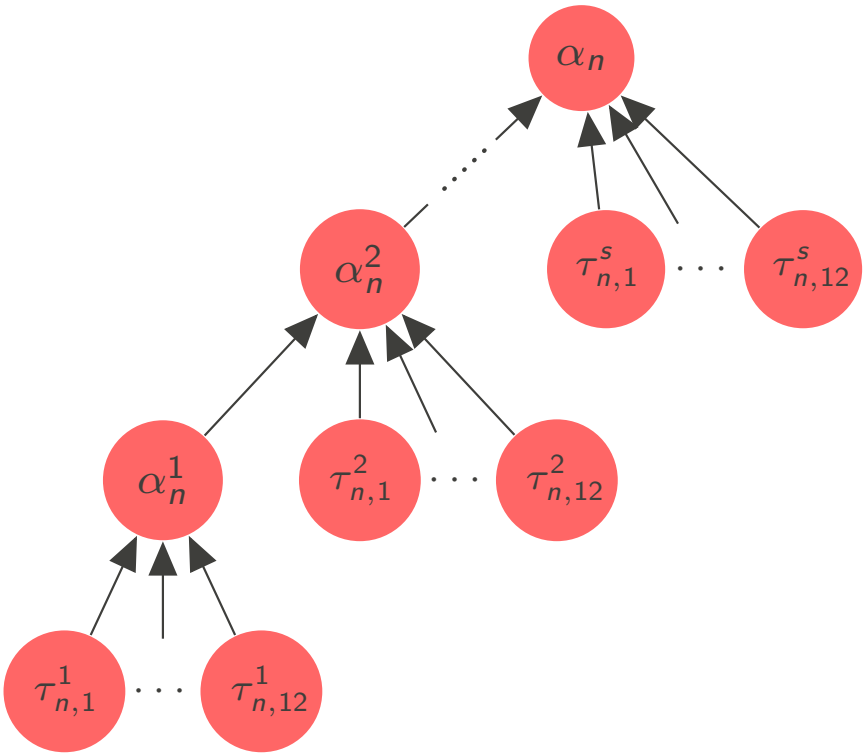
2D neutron transport equation:

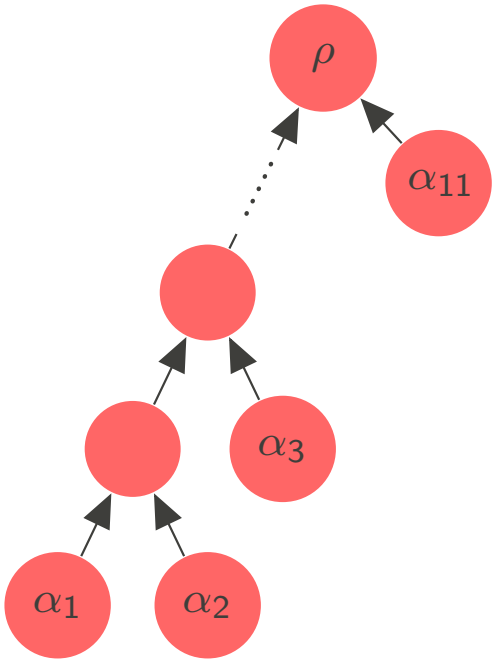
- Moment closure/FV approximation.
- Varying absorbtion and scattering coefficients.
- Distributed snapshot and HAPOD computation on PALMA cluster (125 cores).





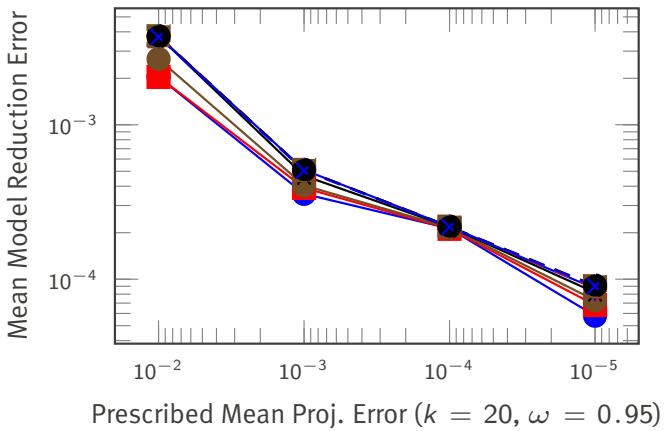
HAPOD – HPC Example

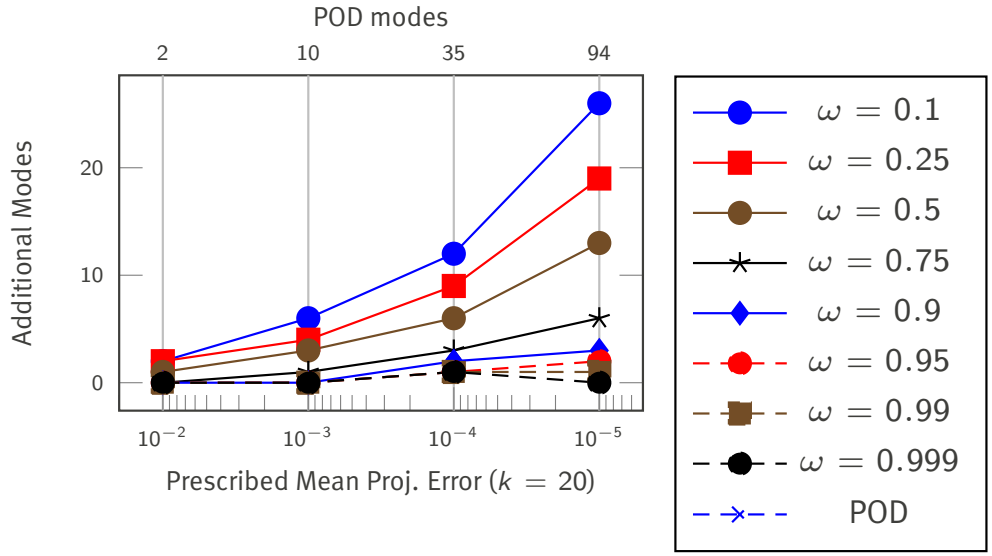


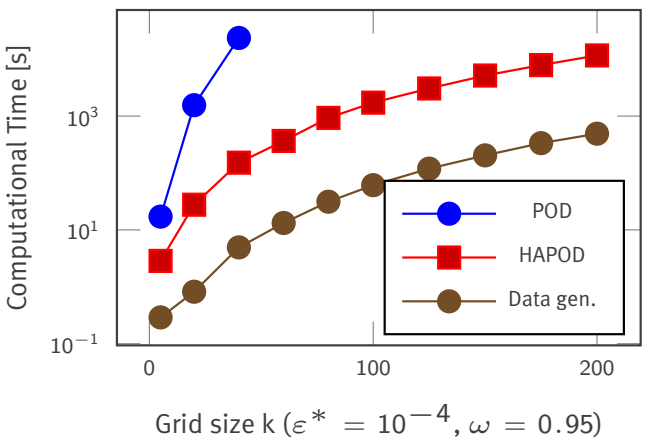


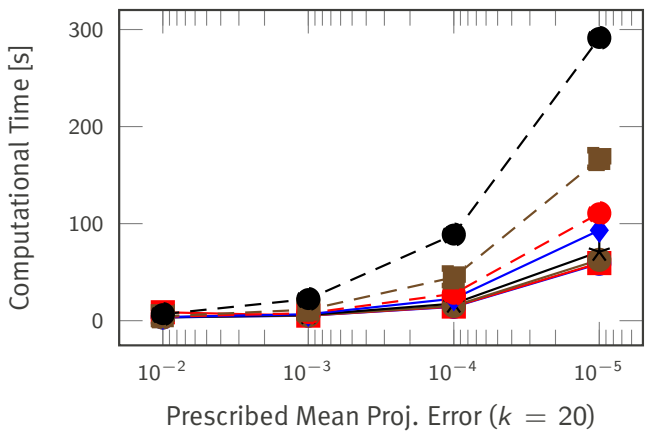
- HAPOD on compute node n. Time steps are split into s slices. Each processor core computes one slice at a time, performs POD and sends resulting modes to main MPI rank on the node.
- Incremental HAPOD is performed on MPI rank 0 with modes collected on each node.

HAPOD – HPC Example

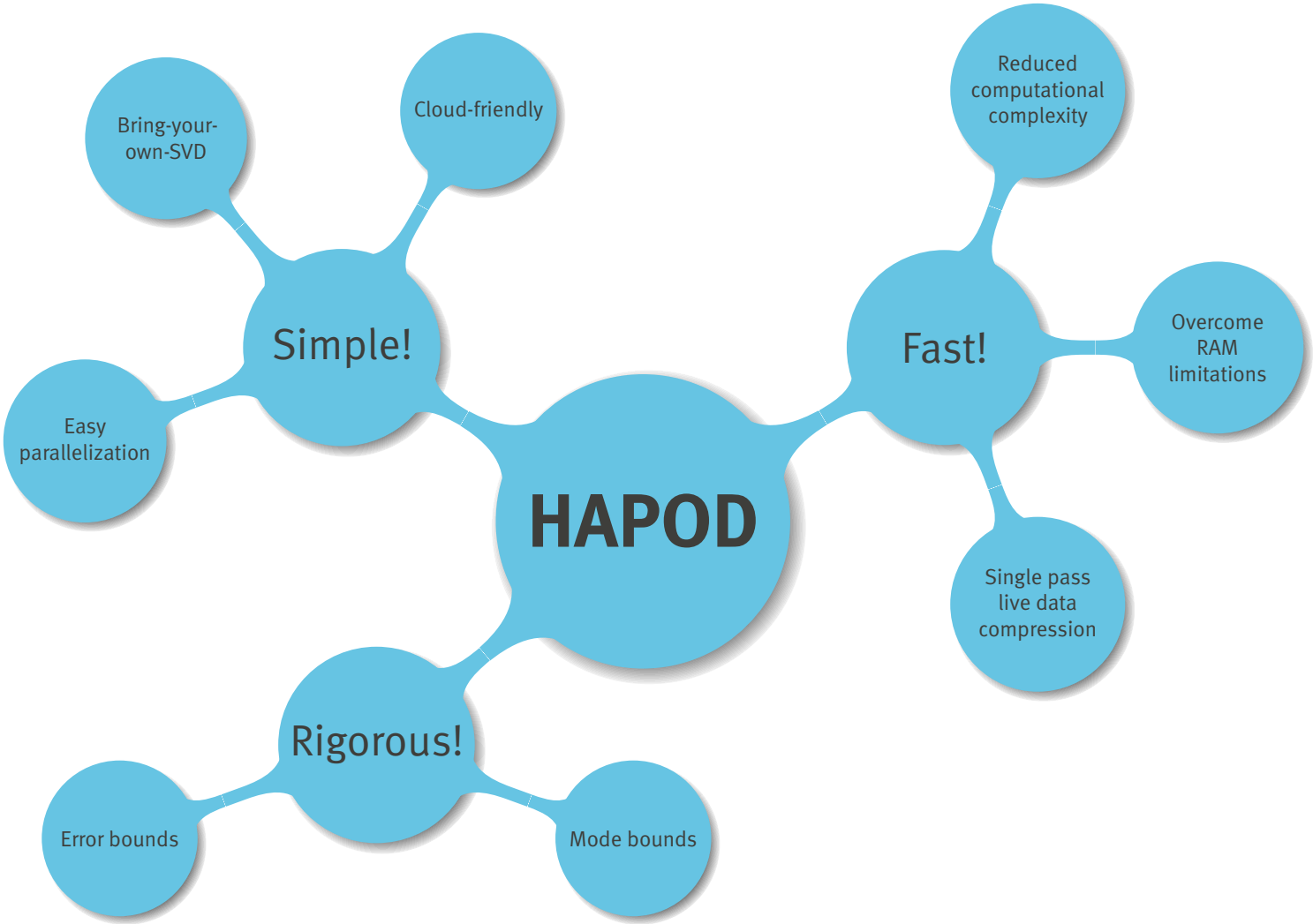








- $\approx 39.000 \cdot k^3$ doubles of snapshot data ($\approx$ 2.5 terabyte for $k = 200$).


HAPOD
Simple!
Bring-your-own-SVD
Cloud-friendly
Easy parallelization
Fast!
Reduced computational complexity
Overcome RAM limitations
Single pass live data compression
Rigorous!
Error bounds
Mode bounds

Model Order Reduction with pyMOR

pyMOR main developers

Rene Milk

Petar Mlinarić

Stephan Rave

Felix Schindler

Classic RB Software Designs

Design 1: write data needed for reduction to disk (e.g. `rbMIT`)

Handle reduction and reduced solving by dedicated MOR code. Read all needed data from disk after run of PDE solver in special MOR output mode.

Design 2: add MOR mode to PDE solver (e.g. `libMesh`)

Add all MOR code needed directly to the PDE solver. Optionally, implement specialized MOR version of solver to run on small devices.

Design 3: communicate only reduced data (e.g. `RBmatlab + dune-rb`)

Write MOR code which communicates with running PDE solver. MOR code can, e.g., instruct solver to enrich basis with snapshot for certain μ and to compute data for reduced model.

RB Software Design Comparison

	via disk	in solver	low-dim
large (e.g. matrix-free) problems			
empirical interpolation			
reusability w. new solver library			
reusability w. new MOR alg.			
MOR alg. easy to implement			
easy to use w. new FOM			
easy to maintain			
MOR and solver dev. decoupled			

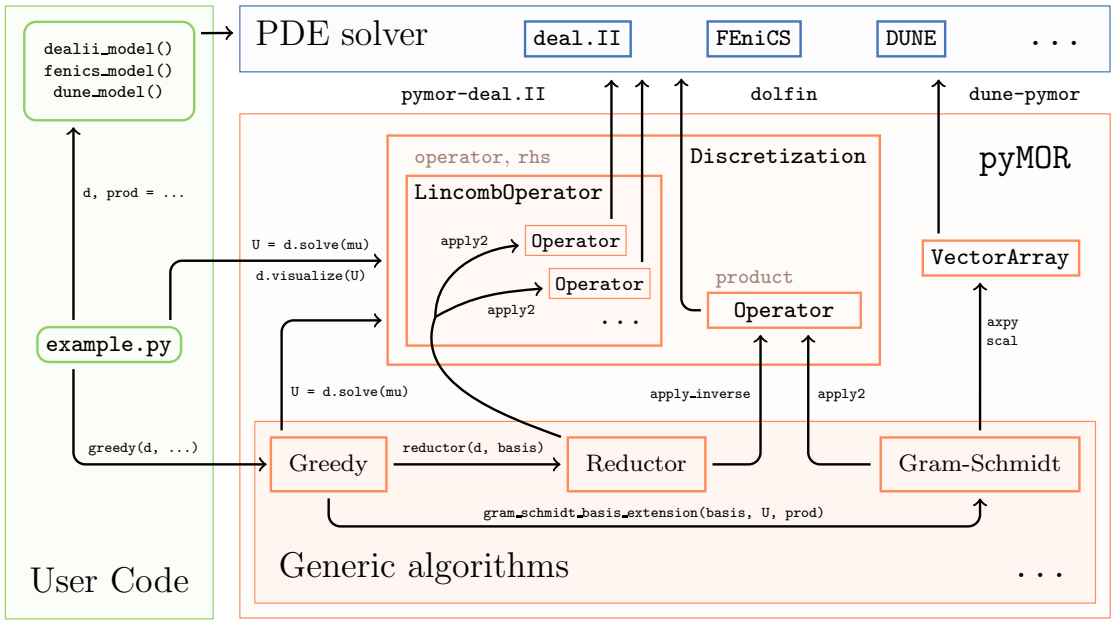
pyMOR – Model Order Reduction with Python

Goal

One tool for algorithm development *and* large-scale applications.

- Started late 2012, 20k lines of Python code, 3.7k single commits.
- BSD-licensed, fork us on Github!
- Quick prototyping with Python 3.
- Comes with small `NumPy`/`SciPy`-based discretization toolkit for getting started quickly.
- Seamless integration with high-performance PDE solvers.

Generic Algorithms and Interfaces for MOR



- `VectorArray`, `Operator`, `Discretization` classes represent objects in solver's memory.
- No communication of high-dimensional data.
- Tight, low-level integration with external solver.
- No MOR-specific code in solver.

Implemented Algorithms

- Gram-Schmidt, POD, HAPOD (in `hapod` branch).
- Adaptive greedy basis generation with different extension algorithms.
- Automatic (Petrov-)Galerkin projection of arbitrarily nested affine combinations of operators.
- Interpolation of arbitrary (nonlinear) operators, EI-Greedy, DEIM.
- A posteriori error estimation.
- Iterative linear solvers, Newton algorithm, time-stepping algorithms.
- System theory Methods: balanced truncation, IRKA, ... (in `sys-mor` branch)

RB Software Design Comparison

	via disk	in solver	low-dim	pyMOR
large (e.g. matrix-free) problems				
empirical interpolation				
reusability w. new solver library				
reusability w. new MOR alg.				
MOR alg. easy to implement				
easy to use w. new FOM				
easy to maintain				
MOR and solver dev. decoupled				

RB Software Design Comparison

	via disk	in solver	low-dim	pyMOR
large (e.g. matrix-free) problems	☹	☺	☺	☺
empirical interpolation	☹	☺	😐	☺
reusability w. new solver library	☺	☹	😐	☺
reusability w. new MOR alg.	☺	☺	😐	☺
MOR alg. easy to implement	☺	😐	☹	😐
easy to use w. new FOM	☺	😐	☹	😐
easy to maintain	☺	😐	☹	☺
MOR and solver dev. decoupled	☺	☹	☹	☺

FEniCS Support Included

- Directly interfaces FEniCS
  LA backend, no copies needed.
- Use same MOR code with both backends!
- Thermal block demo:
  30 LOC FEniCS +
  15 LOC wrapping for pyMOR.
- Easily increase FEM order, etc.

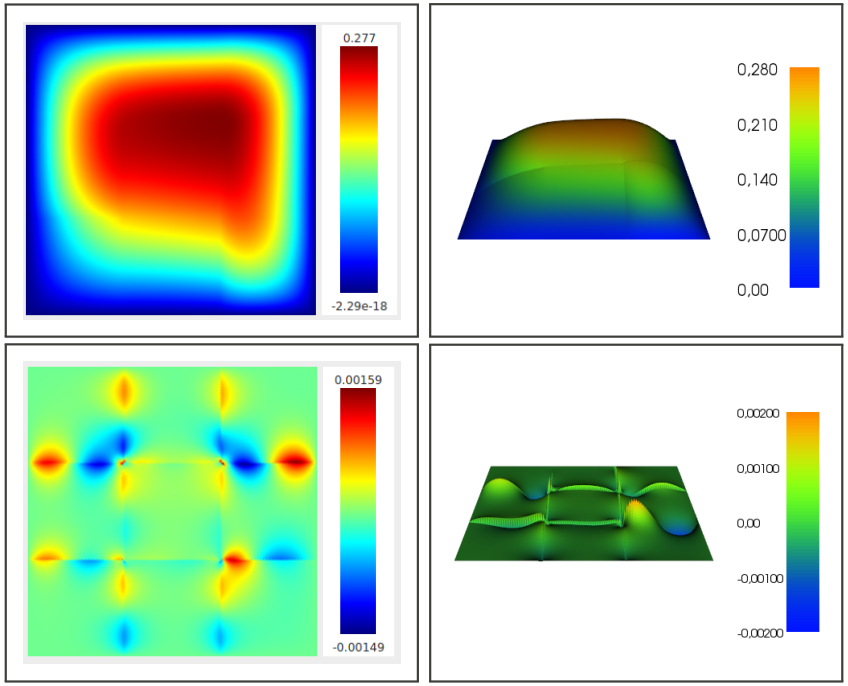
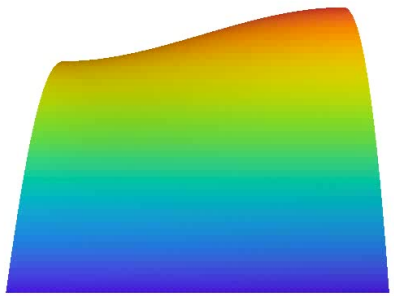


Figure: 3x3 thermal block problem
top: red. solution, bottom: red. error
left: pyMOR solver, right: FEniCS solver

Empirical Operator Interpolation with FEniCS



Two approaches for local operator evaluation:

1. Use `dolfin.assemble_local` (210 ms)
2. Use `dolfin.SubMesh` (35 ms)

See `fenics_nonlinear` branch.

Nonlinear Poisson problem from FEniCS docs (for $\mu = 1$)

$$
\begin{aligned}
-\nabla \cdot \left\{ (1 + \mu u^2(x, y)) \cdot \nabla u(x, y) \right\} &= x \cdot \sin(y) && \text{for } x, y \in (0, 1) \\
u(x, y) &= 1 && \text{for } x = 1 \\
\nabla u(x, y) \cdot n &= 0 && \text{otherwise}
\end{aligned}
$$

- `mesh = UnitSquareMesh(100, 100); V = FunctionSpace(mesh, "CG", 2)`
- Time for solution: 3.4 s
- $\mu \in [1, 1000]$, RB size: 3, EI DOFs: 5, rel. error $< 10^{-6}$

Empirical Operator Interpolation with FEniCS

Determine mesh elements to visit to compute DOFs of interest

```
elements_to_visit = set()
for c in cells(mesh):
    cell_index = c.index()
    local_dofs = V.dofmap.cell_dofs(cell_index)
    if any(ld in dofs_of_interest for ld in local_dofs):
        elements_to_visit.add(cell_index)
```

Local evaluation with assemble_local

```
source_vec[source_dofs] = u
for cell, dof_map in zip(cells, dof_maps):
    local_evaluations = assemble_local(form, cell)
    r[dof_map] += local_evaluations
```

deal.II Support

- `pymor-deall.II` support module

  `https://github.com/pymor/pymor-deal.II`

- Python bindings for
  - `dealii::Vector`,
  - `dealii::SparseMatrix`.
- pyMOR wrapper classes.
- MOR demo for linear elasticity example from tutorial.

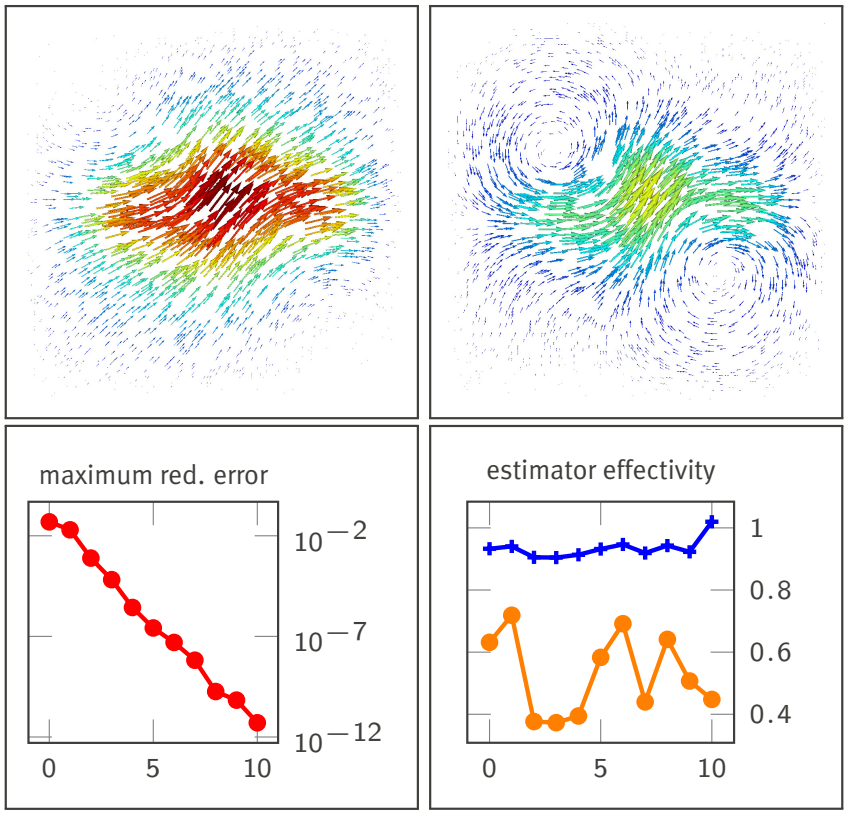


Figure: top: Solutions for $(\mu, \lambda) = (1, 1)$ and $(\mu, \lambda) = (1, 10)$, bottom: red. errs. and max./min. estimator effectivities vs. dim V_N.

NGSolve Support

- Based on NGSolve - Python.
- pyMOR wrappers for vector and matrix classes.
- 3d thermal block demo: 40 LOC NGSolve + 11 LOC wrapping for pyMOR.
- Joint work with Christoph Lehrenfeld.

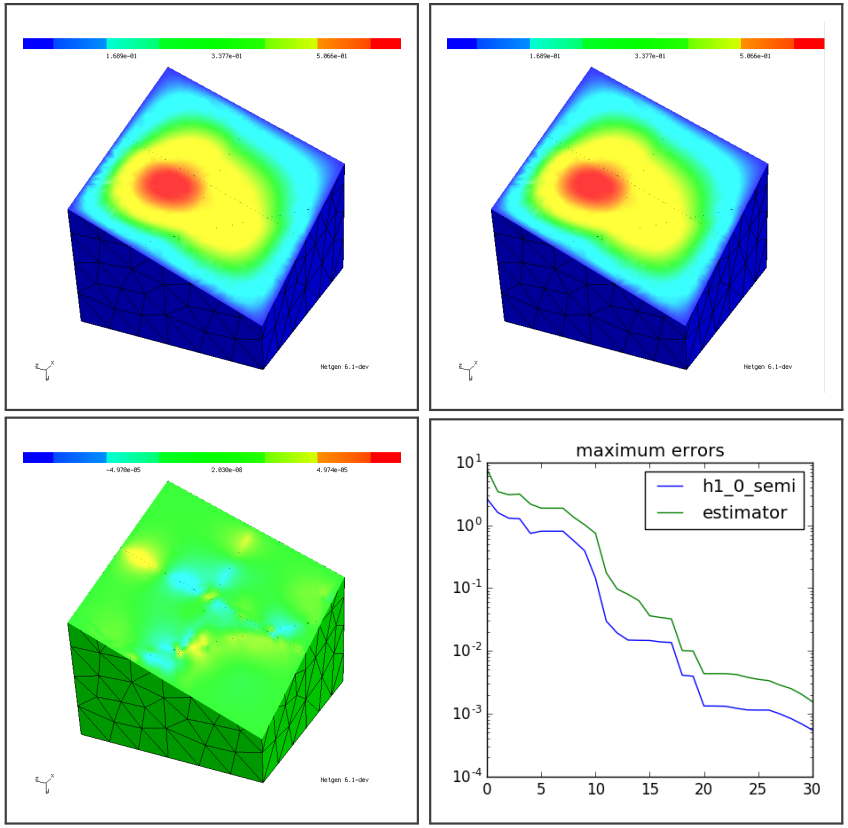


Figure: 3d thermal block problem
top: full/red. sol., bottom: err. for worst approx. μ and max. red. error vs. dim V_N.

Tools for interfacing MPI parallel solvers

- Automatically make sequential bindings MPI aware.
- Reduce HPC-Cluster models without thinking about MPI at all.
- Interactively debug MPI parallel solvers.

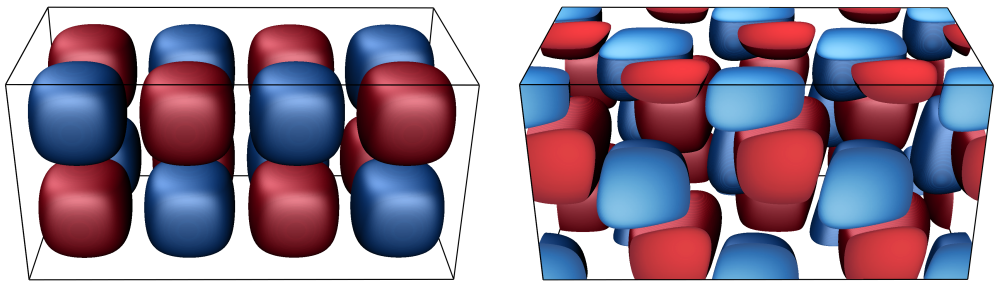


Figure: FV solution of 3D Burgers-type equation ($27.6 \cdot 10^6$ DOFs, 600 time steps) using Dune.

Table: Time (s) needed for solution using `DUNE` / `DUNE` with `pyMOR` timestepping.

`MPI` ranks	1	2	3	6	12	24	48	96	192
`DUNE`	17076	8519	5727	2969	1525	775	395	202	107
`pyMOR`	17742	8904	6014	3139	1606	816	418	213	120
overhead	3.9%	4.5%	5.0%	5.7%	5.3%	5.3%	6.0%	5.4%	11.8%

Thank you for your attention!

C. Himpe, T. Leibner, S. Rave, Hierarchical Approximate Proper Orthogonal Decomposition
arXiv:1607.05210

pyMOR – Generic Algorithms and Interfaces for Model Order Reduction
SIAM J. Sci. Comput., 38(5), pp. S194–S216
`pip install git+https://github.com/pymor/pymor@hapod`

Matlab HAPOD implementation:
`git clone https://github.com/gramian/hapod`

My homepage:
`https://stephanrave.de/`