WWU
MÜNSTER

PyMOR

Model Order Reduction with Python

Linus Balicki[1], René Fritze[2], Petar Mlinarić[1], Stephan Rave[2], Felix Schindler[2]
[1]Virginia Tech, Blacksburg, USA, [2]Uni Münster, Germany

CMTC / IVV NWZ Mini-Symposium Data Science Technologies

Münster, September 28, 2022

living.knowledge

MM
Mathematics
Münster
Cluster of Excellence

Outline

- Model Order Reduction.
- Model Order Reduction with pyMOR.
- Data-Driven Model Order Reduction with pyMOR.

pyMOR main developers


Linus Balicki


René Fritze

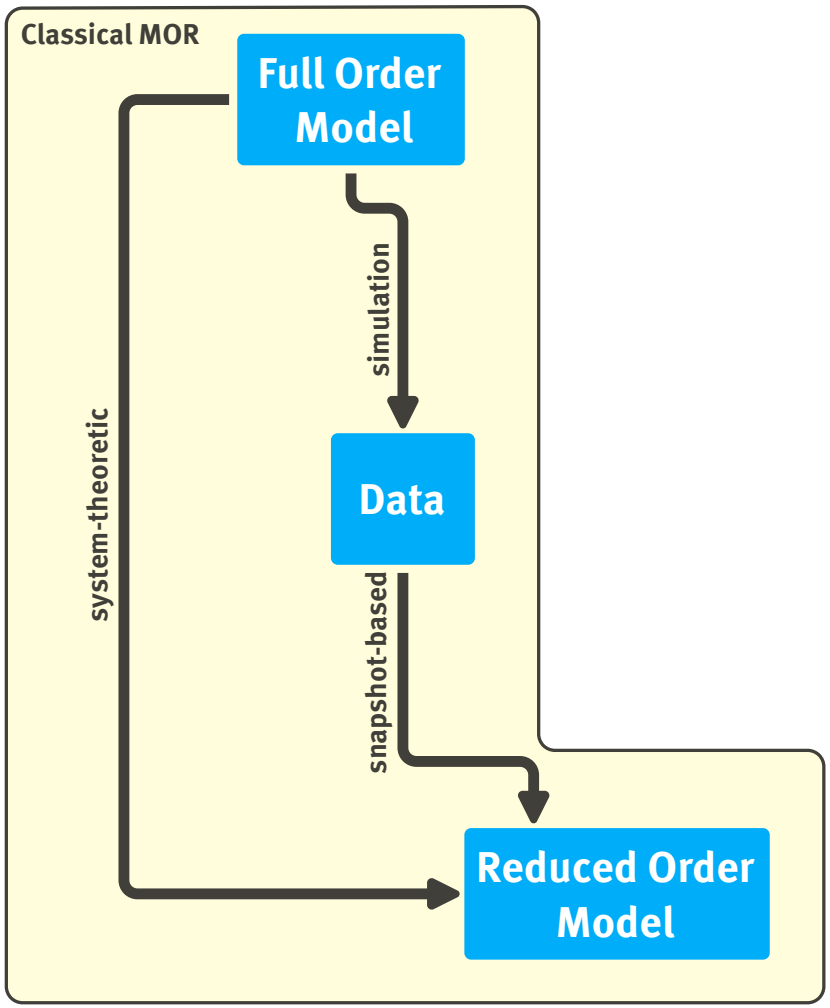

Petar Mlinarić


Stephan Rave


Felix Schindler

Model Order Reduction

Model Order Reduction and Data Science

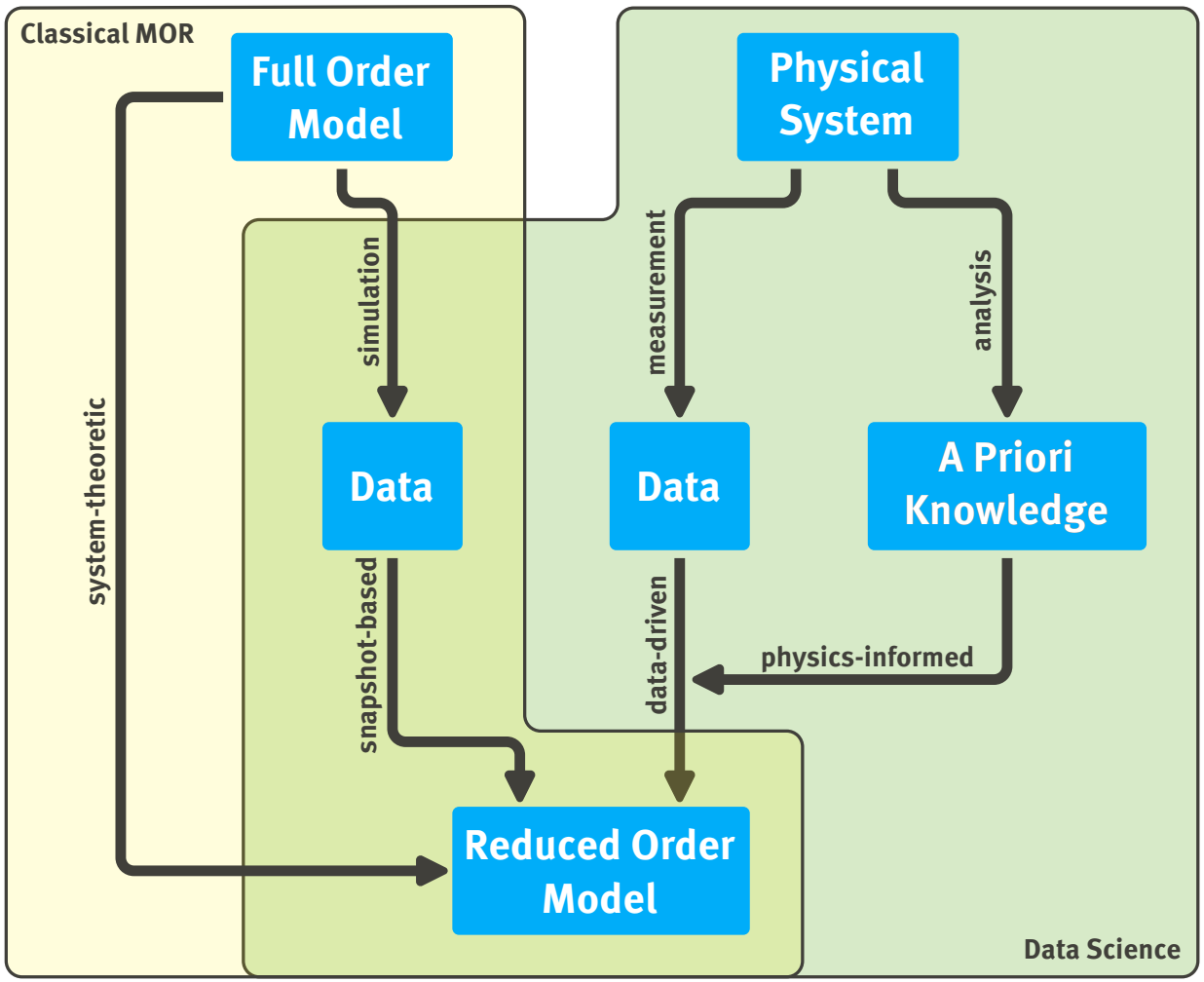


Compute computationally efficient surrogate models for

- optimization,
- control,
- real-time predictions,
- ...

Certification: rigorously control approximation error a priori or a posteriori.

Model Order Reduction and Data Science



Compute computationally efficient surrogate models for

- optimization,
- control,
- real-time predictions,
- ...

Certification: rigorously control approximation error a priori or a posteriori.

System-Theoretic Model Order Reduction

Linear time invariant system (full order model)

Input $u(t) \in \mathbb{R}^m$ to state $x(t) \in \mathbb{R}^n$ to output $y(t) \in \mathbb{R}^p$ mapping is given by

$$\dot{x}(t) = Ax(t) + Bu(t)$$
$$y(t) = Cx(t).$$

System-Theoretic Model Order Reduction

Linear time invariant system (full order model)

Input $u(t) \in \mathbb{R}^m$ to state $x(t) \in \mathbb{R}^n$ to output $y(t) \in \mathbb{R}^p$ mapping is given by

$$\dot{x}(t) = Ax(t) + Bu(t)$$
$$y(t) = Cx(t).$$

Model Reduction Magic

Compute 'good' W, $V \in \mathbb{R}^{n \times r}$, $r \ll n$, $\quad s.t. \quad W^T V = I$

Linear time invariant system (reduced order model)

Input $u(t) \in \mathbb{R}^m$ to state $x(t) \in \mathbb{R}^r$ to output $y(t) \in \mathbb{R}^p$ mapping is given by

$$\dot{x}(t) = (W^T AV)\, x(t) + (W^T B)\, u(t)$$
$$y(t) = (CV)\, x(t).$$

System-Theoretic MOR – Computing V and W

Balanced Truncation

- Compute reachability Gramian $\mathcal{P}$ and observability Gramian $\mathcal{Q}$ subject to

$$A\mathcal{P} + \mathcal{P}A^T + BB^T = 0$$

$$A^T\mathcal{Q} + \mathcal{Q}A + C^TC = 0.$$

- Simultaneously diagonalize $\mathcal{P}$ and $\mathcal{Q}$.
- Select V, W by truncating states which are both hard to reach and hard to observe.

System-Theoretic MOR – Computing V and W

Balanced Truncation

- Compute reachability Gramian $\mathcal{P}$ and observability Gramian $\mathcal{Q}$ subject to

$$A\mathcal{P} + \mathcal{P}A^T + BB^T = 0$$
$$A^T\mathcal{Q} + \mathcal{Q}A + C^TC = 0.$$

- Simultaneously diagonalize $\mathcal{P}$ and $\mathcal{Q}$.
- Select V, W by truncating states which are both hard to reach and hard to observe.

Rational interpolation

- Construct V, W, s.t. the transfer function (Laplace transform of impulse response)

$$H(s) = C(sI - A)^{-1}B$$

  is interpolated (including higher moments) at points $s_1, \dots, s_k$ by a rational function.
- Methods: Moment matching, Padé approximation, IRKA, ...
- Computed using rational Krylov methods.

Reduced Basis Methods

Parametric linear parabolic problem (full order model)

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

$$\begin{aligned} u_\mu(0) &= u_0 \\ \partial_t u_\mu(t) - \Delta u_\mu(t) &= f(v) \\ y_\mu(t) &= g(u_\mu(t)). \end{aligned}$$

Reduced Basis Methods

Parametric linear parabolic problem (full order model)

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

$$u_\mu(0) = u_0$$

$$\int_\Omega v(x)\partial_t u_\mu(t)\,\mathrm{d}x + \int_\Omega \nabla v(x) \cdot \nabla u_\mu(x,t)\,\mathrm{d}x = \int_\Omega f(x)v(x)\,\mathrm{d}x \qquad \forall v \in V_h,$$

$$y_\mu(t) = g(u_\mu(t)).$$

Reduced Basis Methods

Parametric linear parabolic problem (full order model)

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

$$\begin{aligned} u_\mu(0) &= u_0 \\ \langle v, \partial_t u_\mu(t) \rangle + b_\mu(v, u_\mu(t)) &= f(v) \qquad \forall v \in V_h, \\ y_\mu(t) &= g(u_\mu(t)). \end{aligned}$$

Reduced Basis Methods

Parametric linear parabolic problem (full order model)

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

$$\begin{aligned} u_\mu(0) &= u_0 \\ \langle v, \partial_t u_\mu(t) \rangle + b_\mu(v, u_\mu(t)) &= f(v) \qquad \forall v \in V_h, \\ y_\mu(t) &= g(u_\mu(t)). \end{aligned}$$

Parametric linear parabolic problem (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.

$$\begin{aligned} u_{\mu,N}(0) &= P_{V_N}(u_0), \\ \langle v, \partial_t u_{\mu(t),N} \rangle + b_\mu(v, u_{\mu(t),N}) &= f(v) \qquad \forall v \in V_N, \\ y_{\mu,N}(t) &= g(u_{\mu,N}(t)), \end{aligned}$$

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

RB Methods – Computing V_N

Weak greedy basis generation

1: **function** WEAK-GREEDY($\mathcal{S}_{train} \subset \mathcal{P}$, ε)
2: $V_N \leftarrow \{0\}$
3: **while** $\max_{\mu \in \mathcal{S}_{train}}$ ERR-EST(ROM-SOLVE(μ), μ) $> \varepsilon$ **do**
4: $\mu^* \leftarrow$ arg-$\max_{\mu \in \mathcal{S}_{train}}$ ERR-EST(ROM-SOLVE(μ), μ)
5: $V_N \leftarrow$ BASIS-EXT(V_N, FOM-SOLVE(μ^*))
6: **end while**
7: **return** V_N
8: **end function**

BASIS-EXT

1. Compute $u^{\perp}_{\mu^*}(t) = (I - P_{V_N}) u_{\mu^*}(t)$.
2. Add POD($u^{\perp}_{\mu^*}(t)$) to V_N (leading left-singular vectors of snapshot matrix).

ERR-EST

Use residual-based error estimate w.r.t. FOM (finite dimensional $\rightsquigarrow$ can compute dual norms).

RB Methods – Online Efficiency

Parametric linear parabolic problem (reduced order model)

$$
\begin{aligned}
u_{\mu,N}(0) &= P_{V_N}(u_0), \\
\langle v, \partial_t u_{\mu(t),N} \rangle + b_\mu(v, u_{\mu(t),N}) &= f(v) \qquad \forall v \in V_N, \\
y_{\mu,N}(t) &= g(u_{\mu,N}(t)),
\end{aligned}
$$

Parameter separability

Assume that b_μ can be written as

$$
b_\mu(v, u) = \sum_{q=1}^{Q} \theta_q(\mu) b_q(v, u).
$$

RB Methods – Online Efficiency

Parametric linear parabolic problem (reduced order model)

$$
\begin{aligned}
u_{\mu,N}(0) &= P_{V_N}(u_0), \\
\langle v, \partial_t u_{\mu(t),N} \rangle + b_\mu(v, u_{\mu(t),N}) &= f(v) \qquad \forall v \in V_N, \\
y_{\mu,N}(t) &= g(u_{\mu,N}(t)),
\end{aligned}
$$

Parameter separability

Assume that b_μ can be written as

$$
b_\mu(v, u) = \sum_{q=1}^{Q} \theta_q(\mu) b_q(v, u).
$$

Offline/online splitting

By pre-computing

$$
\langle \varphi_i, \varphi_j \rangle, \; b_q(\varphi_i, \varphi_j), \; f(\varphi_i), \; g(\varphi_i)
$$

for a reduced basis $\varphi_1, \dots, \varphi_N$ of V_N, solving ROM becomes independent of $\dim V_h$.

RB Methods – Nonlinear Problems

Parametric nonlinear parabolic problem (full order model)

$$
\begin{aligned}
u_\mu(0) &= u_0, \\
\langle v, \partial_t u_\mu(t) \rangle + \langle v, \mathcal{A}_\mu(u_\mu(t)) \rangle &= f(v) \qquad \forall v \in V_h, \\
y_\mu(t) &= g(u_\mu(t)),
\end{aligned}
$$

where $\mathcal{A}_\mu \colon V_h \to V_h^*$ is a nonlinear operator.

RB Methods – Nonlinear Problems

Parametric nonlinear parabolic problem (full order model)

$$
\begin{aligned}
u_\mu(0) &= u_0, \\
\langle v, \partial_t u_\mu(t) \rangle + \langle v, \mathcal{A}_\mu(u_\mu(t)) \rangle &= f(v) \qquad \forall v \in V_h, \\
y_\mu(t) &= g(u_\mu(t)),
\end{aligned}
$$

where $\mathcal{A}_\mu \colon V_h \to V_h^*$ is a nonlinear operator.

Parametric nonlinear parabolic problem (reduced order model)

$$
\begin{aligned}
u_{\mu,N}(0) &= P_{V_N}(u_0), \\
\langle v, \partial_t u_{\mu(t),N} \rangle + \langle v, \mathcal{A}_\mu(u_{\mu(t),N}) \rangle &= f(v) \qquad \forall v \in V_N, \\
y_{\mu,N}(t) &= g(u_{\mu,N}(t)).
\end{aligned}
$$

RB Methods – Nonlinear Problems

Parametric nonlinear parabolic problem (full order model)

$$\begin{aligned} u_\mu(0) &= u_0, \\ \langle v, \partial_t u_\mu(t)\rangle + \langle v, \mathcal{A}_\mu(u_\mu(t))\rangle &= f(v) \qquad \forall v \in V_h, \\ y_\mu(t) &= g(u_\mu(t)), \end{aligned}$$

where $\mathcal{A}_\mu \colon V_h \to V_h^*$ is a nonlinear operator.

Parametric nonlinear parabolic problem (reduced order model)

$$\begin{aligned} u_{\mu,N}(0) &= P_{V_N}(u_0), \\ \langle v, \partial_t u_{\mu(t),N}\rangle + \langle v, \mathcal{A}_\mu(u_{\mu(t),N})\rangle &= f(v) \qquad \forall v \in V_N, \\ y_{\mu,N}(t) &= g(u_{\mu,N}(t)). \end{aligned}$$

Problem: No offline/online splitting of nonlinear

$$\mathcal{A}_\mu \colon V_N \longrightarrow V_h^* \longrightarrow {V_N}^*.$$

Same problem for non-affinely decomposed b_μ.

RB Methods – Empirical Interpolation

EI – abstract version

Let normed Space V, functionals $\Psi \subseteq V^*$ and training set $\mathcal{M} \subset V$ be given.
Construct via EI-Greedy algorithm:

1. Interpolation basis $b_1, \dots b_M \in \operatorname{span} \mathcal{M}$,
2. Interpolation functionals $\psi_1, \dots, \psi_M \in \Psi$.

The empirical interpolant $\mathcal{I}_M(v)$ of an arbitrary $v \in V$ is then determined by

$$\mathcal{I}_M(v) \in \operatorname{span}\{b_1, \dots, b_M\} \quad \text{and} \quad \psi_m(\mathcal{I}_M(v)) = \psi_m(v) \quad 1 \leq m \leq M.$$

EI Cheat Sheet

	V	Ψ	online
function EI	function space	point evaluations	evaluation at 'magic points'
DEIM	range of (discrete) operator	DOFs	local evaluation at selected DOFs
matrix DEIM	matrices of given shape	matrix entries	assembly of selected entries

RB Methods – Hyper-Reduction

Reduced Order Model (with EI)

Find $u_{\mu,N} \in V_N$ s.t.

$$\langle v, \partial_t u_{\mu,N}(t) \rangle + \langle v, \{I_M \circ \mathcal{A}_{M,\mu} \circ R_{M'}\}(u_{\mu,N}(t)) \rangle = f(v) \quad \forall v \in V_N,$$

where

$R_{M'}: V_h \to \mathbb{R}^{M'}$	restriction to M' DOFs needed for local evaluation
$\mathcal{A}_{M,\mu}: \mathbb{R}^{M'} \to \mathbb{R}^M$	local evaluation of $\mathcal{A}_\mu$ at M interpolation DOFs
$I_M: \mathbb{R}^M \to V_h^*$	linear interpolation operator

Offline/Online splitting

- Pre-compute the linear operators $\langle \cdot, I_M(\cdot) \rangle$ and $R_{M'}$ w.r.t. basis of V_N.
- Effort to evaluate $\langle \cdot, I_M \circ \mathcal{A}_{M,\mu} \circ R_{M'}(\cdot) \rangle$ w.r.t. this basis:

$$\mathcal{O}(MN) + \mathcal{O}(M) + \mathcal{O}(MN).$$

Model Order Reduction with pyMOR

pyMOR's Origin: MULTIBAT

ulm university universität ulm

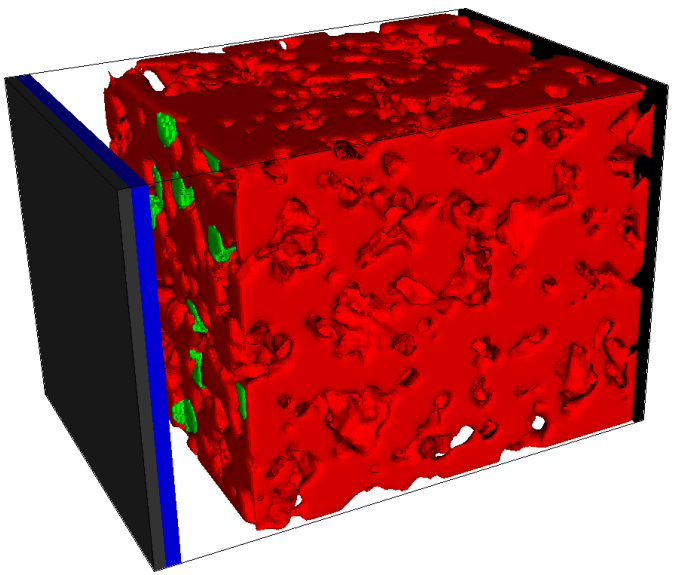
Institute of Technical Thermodynamics

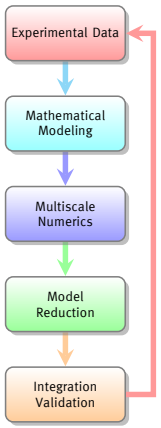
MULTIBAT

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.

How to Implement an RB Method?

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.

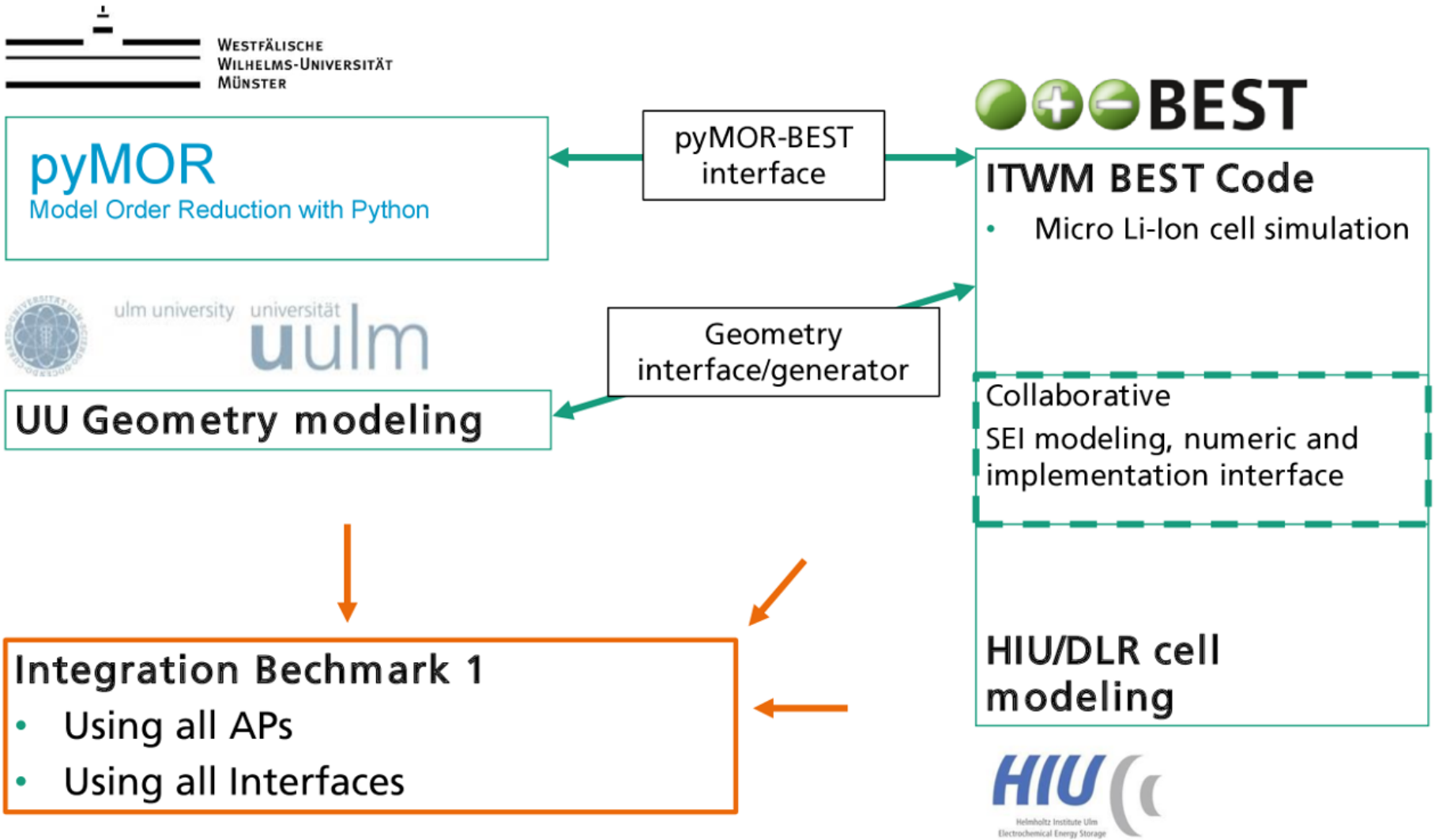
Design 3: communicate only reduced data (e.g. `RBmatlab + dune-rb`)

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.

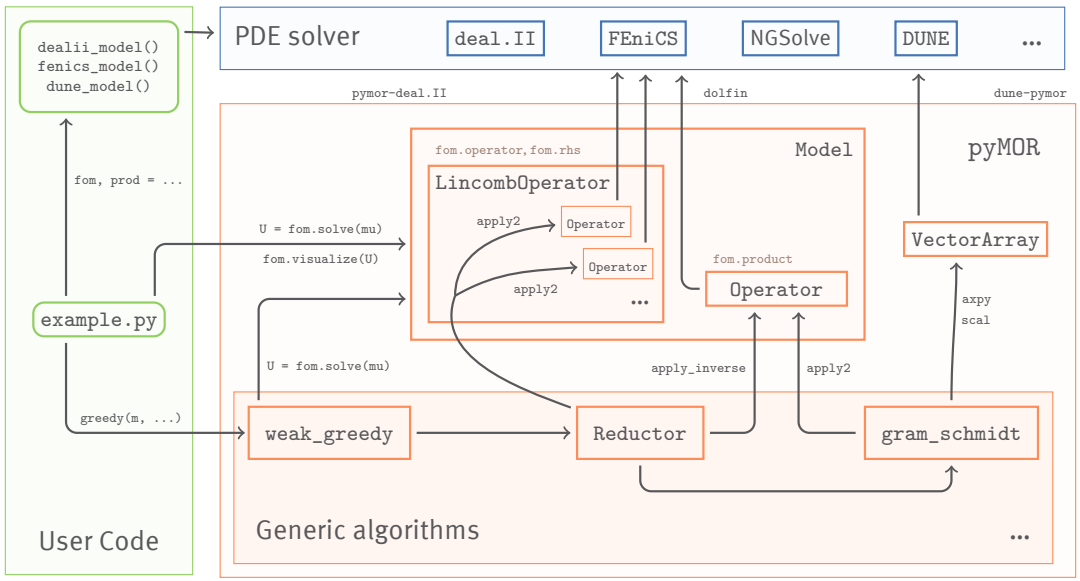
RB Software Design Comparison

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

Software Interfaces in MULTIBAT



Generic Algorithms and Interfaces for MOR



- `VectorArray`, `Operator`, `Model` 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.

RB Software Design Comparison

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

MULTIBAT: Some Results

Model:

- Half-cell with plated Li
- μ = discharge current
- 2.920.000 DOFs

Reduction:

- Snapshots: 3
- $N = 178 + 67$
- $M = 924 + 997$
- Rel. err.: $< 4.5 \cdot 10^{-3}$

Timings:

- Full model: ≈ 15.5h
- Projection: ≈ 14h
- Red. model: ≈ 8m
- Speedup: **120**

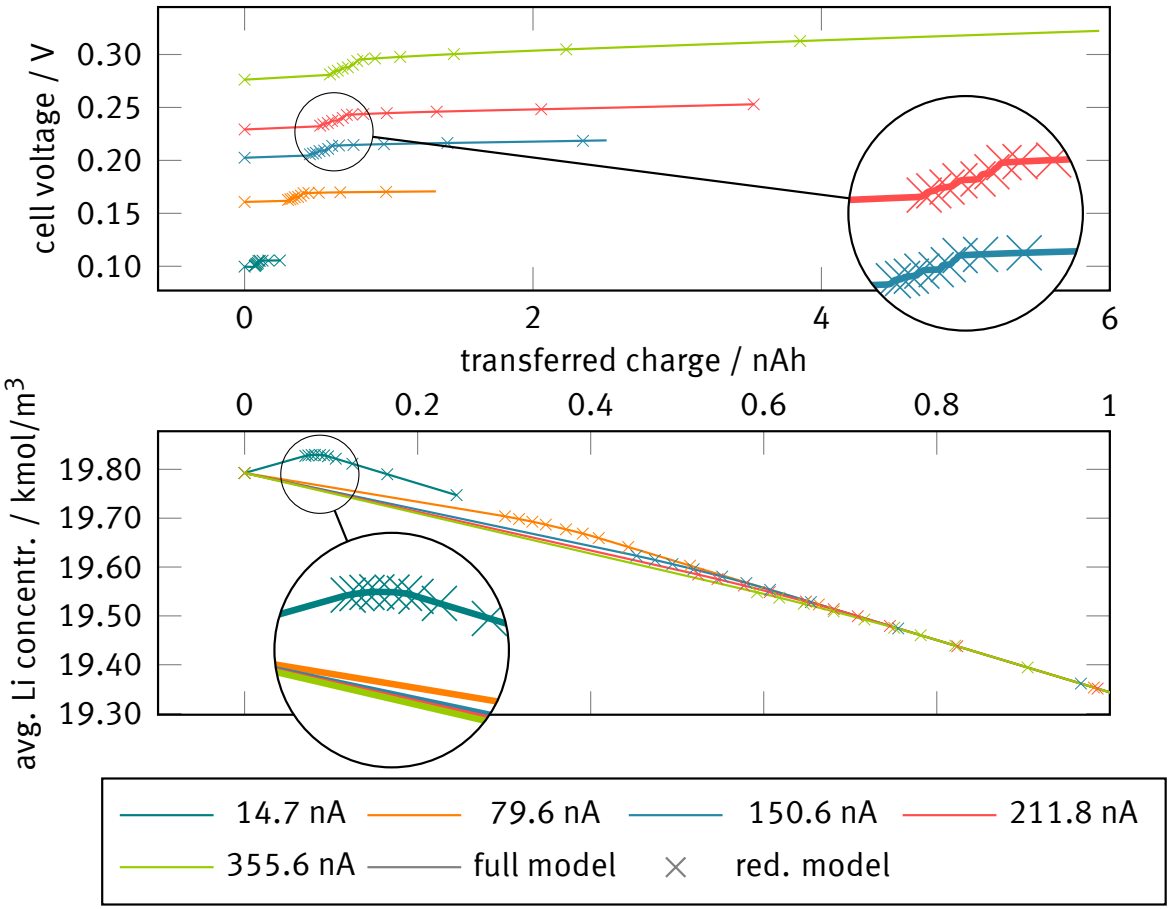


Figure: Validation of reduced order model output for random discharge currents; solid lines: full order model, markers: reduced order model.

pyMOR – Model Order Reduction with Python

Goal 1

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

pyMOR – Model Order Reduction with Python

Goal 1

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

Goal 2

Unified view on MOR.

pyMOR – Model Order Reduction with Python

Goal 1

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

Goal 2

Unified view on MOR.

- Started late 2012, 23k lines of Python code, 8k 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.

pyMOR – Model Order Reduction with Python

Models

StationaryModel
InstationaryModel
LTIModel
PHLTIModel
SecondOrderModel
LinearDelayModel
BilinearModel
TransferFunction
QuadraticHamiltonianModel
LinearStochasticModel

Algorithms

POD
PSD new!
DEIM
DMD new!
IRKA
SAMDP
LGMRES
LSMR
LSQR
certified RB
HAPOD
TF-IRKA
rational Arnoldi
PSD cotangent lift new!
PSD complex SVD new!
modal truncation
time steppers
SLYCOT support
parametric PG projection
adaptive greedy basis generation
non-intrusive MOR with ANNs
low-rank ADI Lyapunov solver
low-rank ADI Riccati solver
bitangential Hermite interpolation
Gram-Schmidt with reiteration
symplectic Gram-Schmidt new!
PSD SVD-like decomposition new!
balanced truncation
empirical interpolation
Arnoldi eigensolver
randomized GSVD new!
randomized eigensolver new!
biorthogonal Gram-Schmidt
tangential rational Krylov
Newton algorithm
second-order BT/IRKA

pyMOR Development

- Development on GitHub.
- Two releases each year.
- CI in WWU cloud via WWU Gitlab.
- Docker images which include all supported PDE solvers.
- Developer documentation (coding style, etc.).
- Partially funded by DFG project (sustainable research software).

pyMOR Community

- Yearly pyMOR school (`school.pymor.org`)
- Ask questions via GitHub Discussions.
- Fix ‘good first issue’.
- Get attribution via `AUTHORS.md`
- Become contributor with push access to feature branches.
- Become main developer with full control over project.
- Internal `gitter.im` chat for contributors and main devs.
- Regular developer meetings (BigBlueButton).
- Code of Conduct in progress.

Getting Started

- Install pyMOR.
  - `pip install pymor`
  - `conda install -c conda-forge pymor`
  - `docker pull pymor/demo:main`
  - `open https://bit.ly/3BapH8S`

Getting Started

- Install pyMOR.
  - `pip install pymor`
  - `conda install -c conda-forge pymor`
  - `docker pull pymor/demo:main`
  - `open https://bit.ly/3BapH8S`

- Learn pyMOR.
  - Interactive tutorials at `docs.pymor.org`.
  - API documentation.
  - Read source code.

Getting Started

- Install pyMOR.
  - `pip install pymor`
  - `conda install -c conda-forge pymor`
  - `docker pull pymor/demo:main`
  - `open https://bit.ly/3BapH8S`
- Learn pyMOR.
  - Interactive tutorials at `docs.pymor.org`.
  - API documentation.
  - Read source code.
- Ask for help! (GitHub Discussions)

Feature Gallery: FEniCS Support

- Directly interfaces FEniCS LA backend, no copies needed.
- Use same MOR code as with builtin discretization toolkit!
- Builtin support for empirical interpolation.
- Thermal block demo:
  30 SLOC FEniCS +
  15 SLOC wrapping for pyMOR.
- Easily increase FEM order, etc.

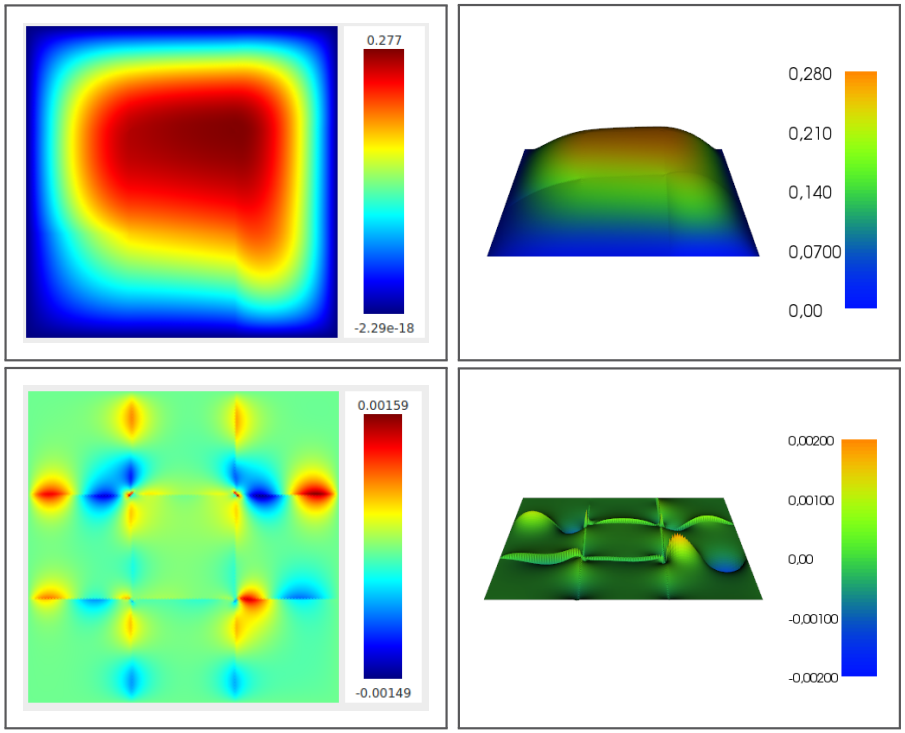


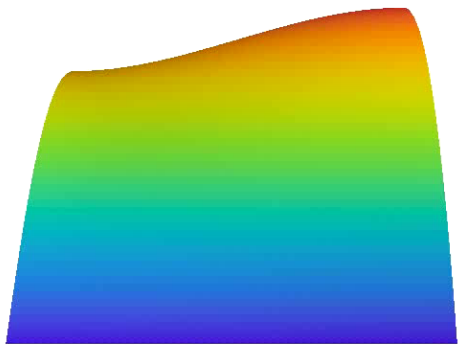
Figure: 3x3 thermal block problem
top: red. solution, bottom: red. error
left: pyMOR solver, right: FEniCS solver

Feature Gallery: Empirical Interpolation with FEniCS

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

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

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



- Local operator evaluation implemented using `dolfin.SubMesh`.
- Speedup: 80.
- See `fenics_nonlinear` demo.

Feature Gallery: 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.

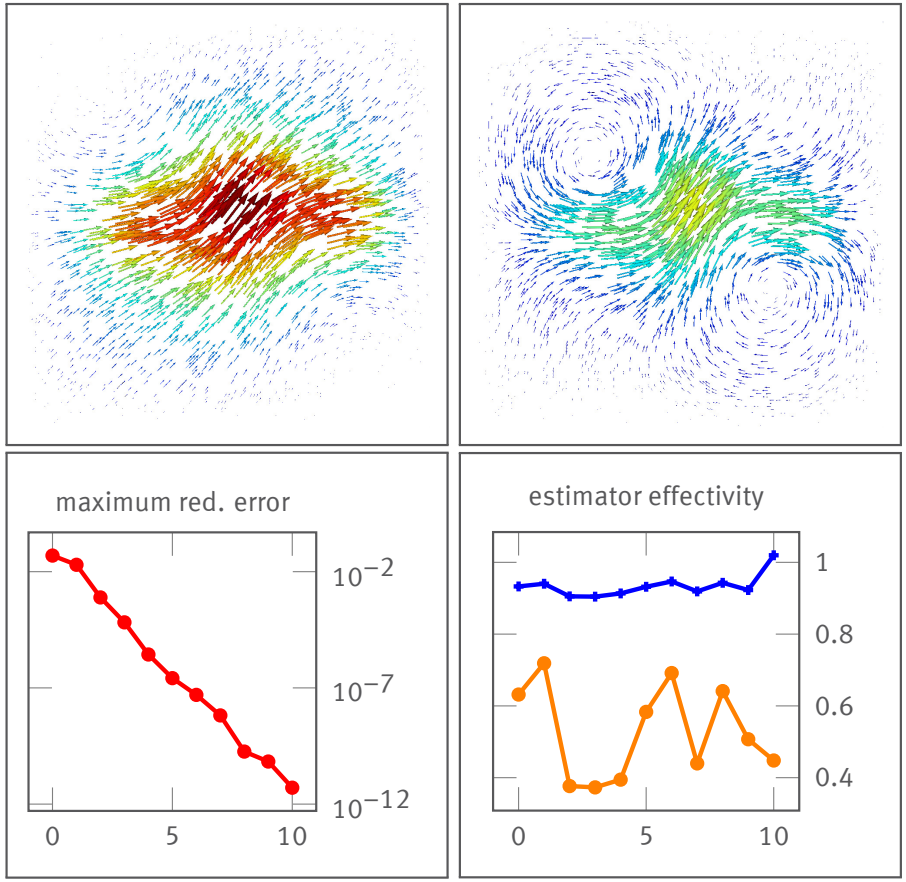


Figure: top: Solutions for $(\mu, \lambda) = (1, 1)$ and $(\mu, \lambda) = (1, 10)$, bottom: red. errs. and max./min. estimator effectivities vs. dim V_N.

Feature Gallery: NGSolve Support

- Based on `NGS-Py` Python bindings for NGSolve.
- pyMOR wrappers for vector and matrix classes.
- 3d thermal block demo included.
- Joint work with Christoph Lehrenfeld.

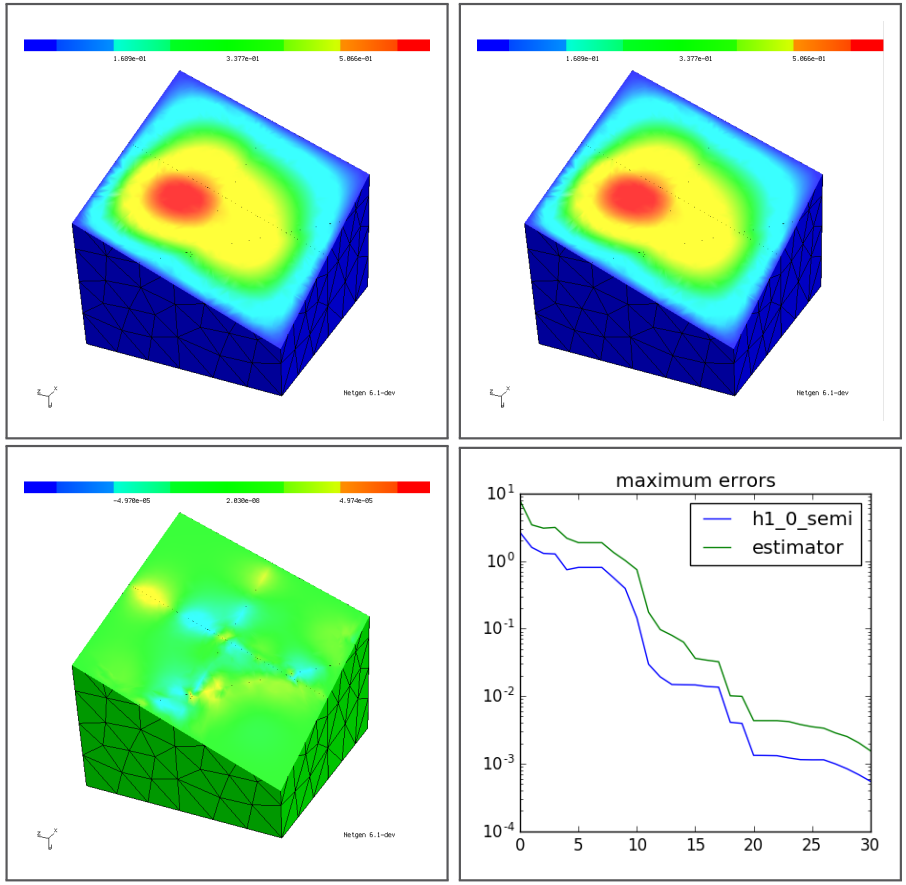


Figure: 3d thermal block problem
top: full/red. sol., bottom: err. for worst approx. μ and max. red. error vs. dim V_N.

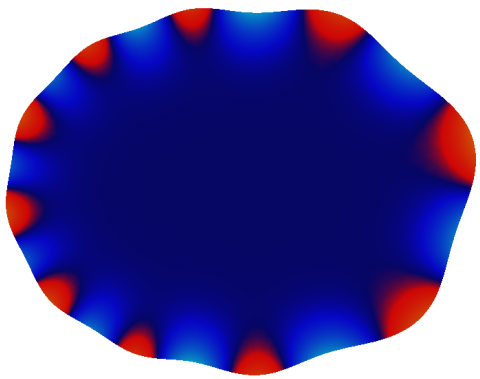
Feature Gallery: MOR for an NGSolve Free Boundary Problem

[Lehrenfeld, R, 19]

Osmotic cell swelling model [Lippoth, Prokert, 2012]

Given $\Omega(0) \subset \mathbb{R}^d$, $u(0) \in H^1(\Omega(0))$ and coefficients $u_{\text{ext}}, \alpha, \beta, \gamma \in \mathbb{R}$, the **concentration** $u(t)$ and **normal velocity** w_Γ of $\Gamma(t)$ is given by:

$$\begin{aligned} \partial_t u - \alpha \Delta u &= 0 && \text{in } \Omega(t), \\ w_\Gamma u + \alpha \partial_{\mathbf{n}} u &= 0 && \text{on } \Gamma(t), \\ -\beta\kappa + \gamma(u - u_{\text{ext}}) &= w_\Gamma && \text{on } \Gamma(t). \end{aligned}$$



- ALE formulation $\rightarrow$ diffusion coeffs nonlinear in deformation field Ψ
- Empirical interpolation w.r.t. Ψ.

Feature Gallery: 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.

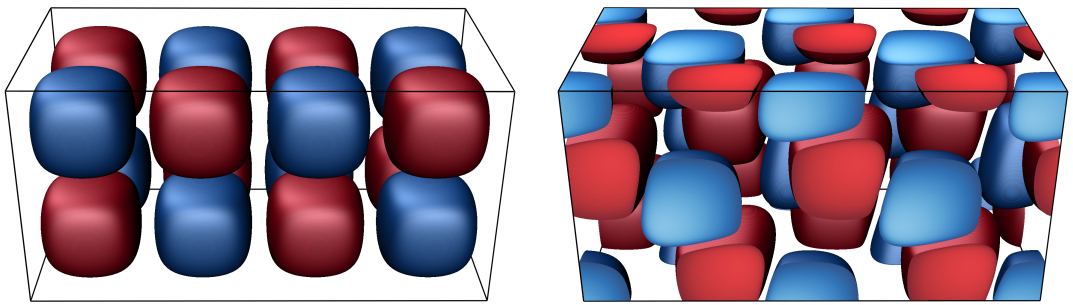


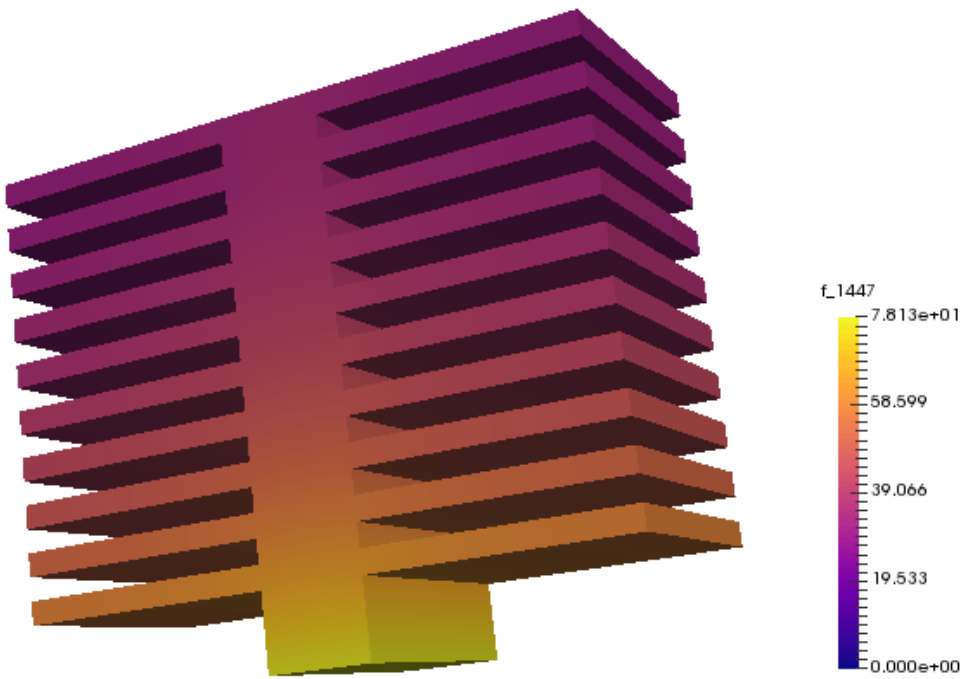
Figure: FV solution of 3D Burgers-type equation ($27.6 \cdot 10^6$ DOFs, 600 time steps) using Dune.

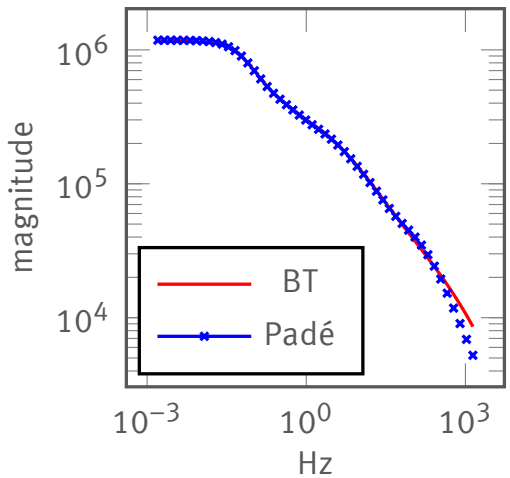
Table: Time (s) needed for solution using DUNE / DUNE with pyMOR timestepping.

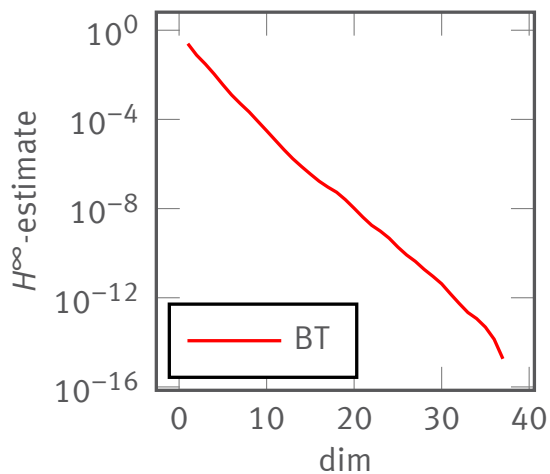
MPI ranks	1	2	3	6	12	24	48	96	192
DUNE	17076	8519	5727	2969	1525	775	395	202	107
pyMOR	17742	8904	6014	3139	1606	816	418	213	120
overhead	3.9%	4.5%	5.0%	5.7%	5.3%	5.3%	6.0%	5.4%	11.8%

Feature Gallery: System-Theoretic MOR with FEniCS

- MPI distributed heatsink model with FEniCS
- Heat conduction with Robin boundary
- Input: heat flow at base
- Output: temperature at base
- MOR: Balanced truncation and Padé approximation







Feature Gallery: System-Theoretic MOR with FEniCS

Model assembly with FEniCS

```
def discretize():
    domain = ...
    mesh = ms.generate_mesh(domain, RESOLUTION)
    subdomain_data = ...

    V = df.FunctionSpace(mesh, 'P', 1)
    u = df.TrialFunction(V)
    v = df.TestFunction(V)
    ds = df.Measure('ds', domain=mesh, subdomain_data=boundary_markers)

    A = df.assemble(- df.Constant(100.) * df.inner(df.grad(u), df.grad(v)) * df.dx
    - df.Constant(0.1) * u * v * ds(1))
    B = df.assemble(df.Constant(1000.) * v * ds(2))
    E = df.assemble(u * v * df.dx)
```

Feature Gallery: System-Theoretic MOR with FEniCS

pyMOR wrapping

```
# def discretize (cont.)

    space = FenicsVectorSpace(V)
    A = FenicsMatrixOperator(A, V, V)
    B = VectorOperator(space.make_array([B]))
    C = B.H
    E = FenicsMatrixOperator(E, V, V)
    fom = LTIModel(A, B, C, None, E)
    return fom
```

Feature Gallery: System-Theoretic MOR with FEniCS

pyMOR wrapping

```
# def discretize (cont.)

    space = FenicsVectorSpace(V)
    A = FenicsMatrixOperator(A, V, V)
    B = VectorOperator(space.make_array([B]))
    C = B.H
    E = FenicsMatrixOperator(E, V, V)
    fom = LTIModel(A, B, C, None, E)
    return fom
```

MPI wrapping

```
from pymor.tools import mpi
if mpi.parallel:
    from pymor.models.mpi import mpi_wrap_model
    fom = mpi_wrap_model(discretize, use_with=True)
else:
    fom = discretize()
```

Feature Gallery: System-Theoretic MOR with FEniCS

Balanced Truncation

```
reductor = BTReductor(fom)
bt_rom = reductor.reduce(10)

bt_rom.mag_plot(np.logspace(-2, 4, 100), Hz=True)
```

Padé approximation

```
k = 10
V = arnoldi(fom.A, fom.E, fom.B, [0] * r)
W = arnoldi(fom.A, fom.E, fom.C, [0] * r, trans=True)
pade_rom = LTIPGReductor(fom, W, V, False).reduce()

pade_rom.mag_plot(np.logspace(-2, 4, 100), Hz=True)
```

Data-Driven MOR with pyMOR

Data-Driven MOR with pyMOR
1. Computing PODs of Large Data Sets

Computing V_N with POD

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)

Computing V_N with POD

Offline phase

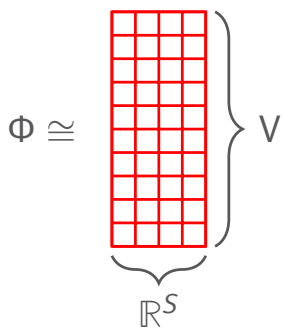
Basis for V_N is computed from **solution snapshots** $u_{\mu_s}(t)$ of full order problem via:

- 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

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

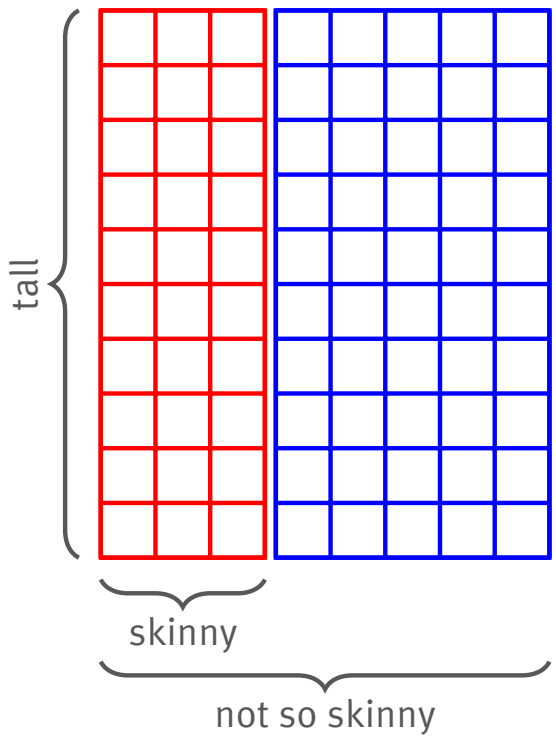


Optimality of POD

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

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

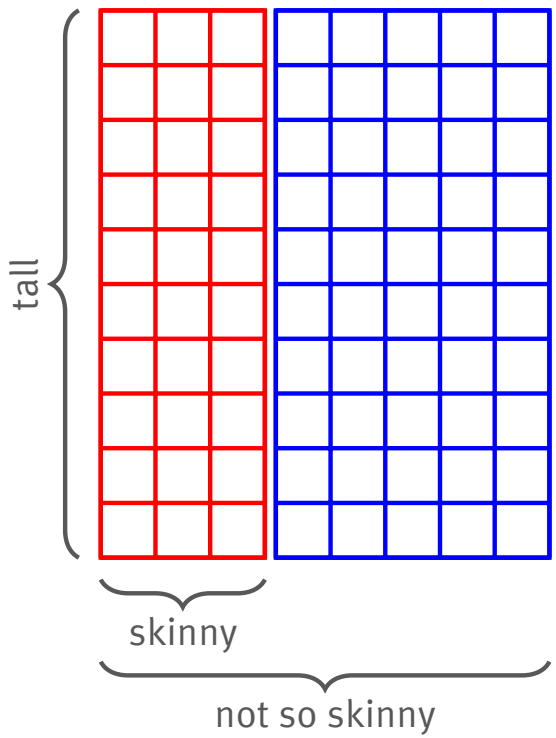
Are your tall and skinny matrices not so skinny anymore?



POD of large snapshot sets:

- large computational effort
- parallelization?
- data $>$ RAM $\Longrightarrow$ disaster

Are your tall and skinny matrices not so skinny anymore?

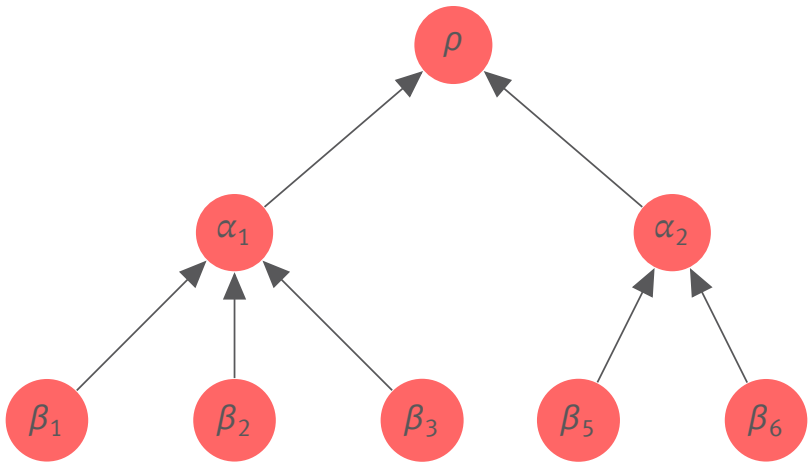


POD of large snapshot sets:

- large computational effort
- parallelization?
- data $\gg$ RAM $\Longrightarrow$ disaster

Solution: PODs of PODs!

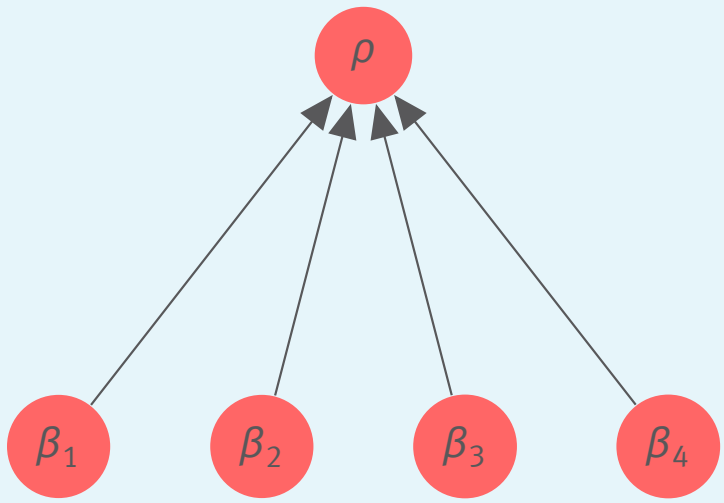
HAPOD – Hierarchical Approximate POD (`pymor.algorithms.hapod`)



- 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.

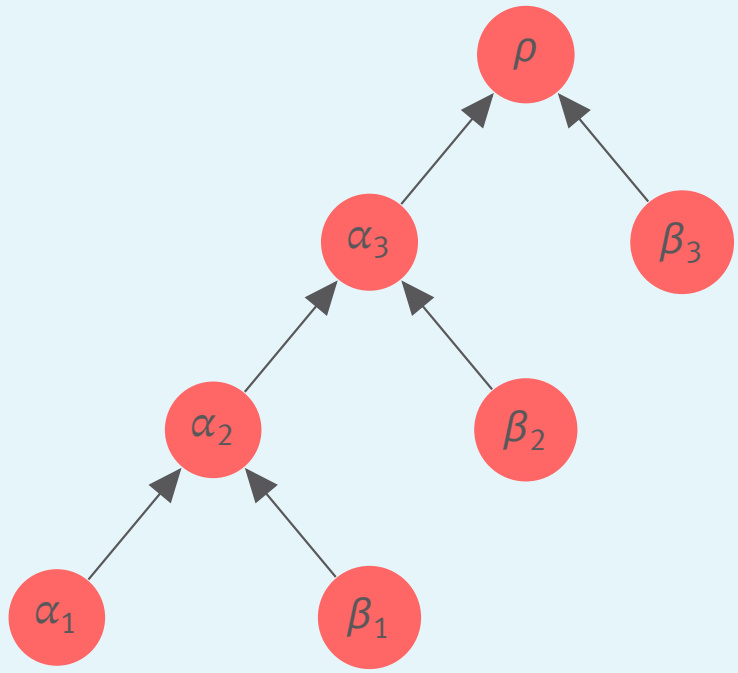
HAPOD – Special Cases

Distributed HAPOD



- Distributed, communication avoiding POD computation.

Incremental HAPOD



- On-the-fly compression of large trajectories.

HAPOD – Theoretical Analysis

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

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

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

Then:

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

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

Moreover:

$$|\operatorname{HAPOD}[\mathcal{S}, \mathcal{T}, D, \varepsilon](\alpha)| \leq |\overline{\operatorname{POD}}(\tilde{S}_\alpha, (L_{\mathcal{T}} - 1)^{-1/2} \cdot \sqrt{1-\omega^2} \cdot \varepsilon^*)|$$

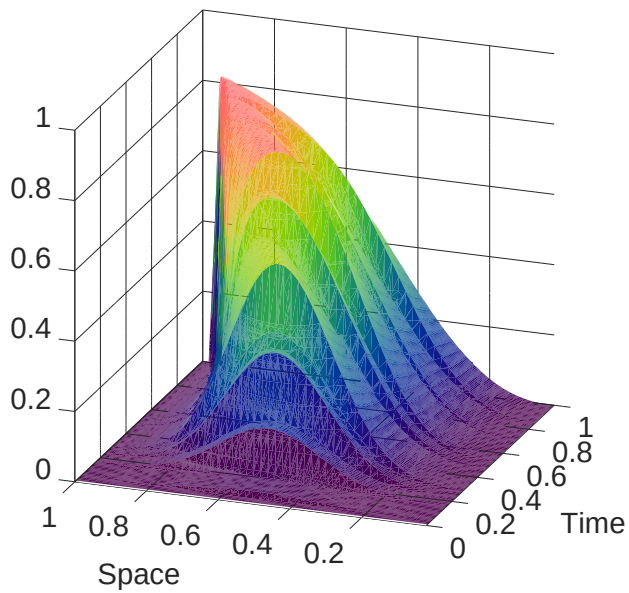
Incremental HAPOD Example

Compress state trajectory of forced inviscid Burgers equation:

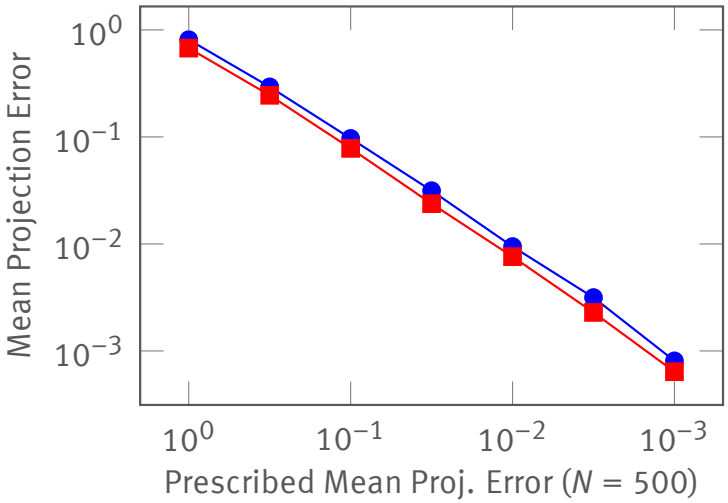
$$\begin{aligned} \partial_t z(x,t) + z(x,t) \cdot \partial_x z(x,t) &= u(t) \exp\left(-\frac{1}{20}\left(x - \frac{1}{2}\right)^2\right), & (x,t) &\in (0,1) \times (0,1), \\ z(x,0) &= 0, & x &\in [0,1], \\ z(0,t) &= 0, & t &\in [0,1], \end{aligned}$$

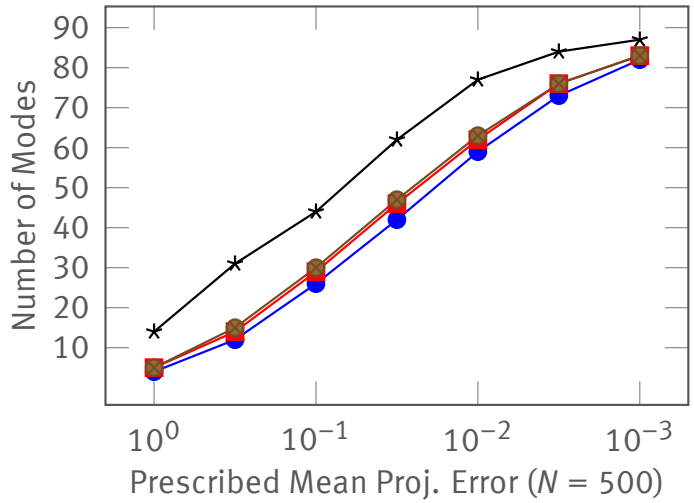
where $u(t) \in [0, 1/5]$ iid. for 0.1% random timesteps, otherwise 0.

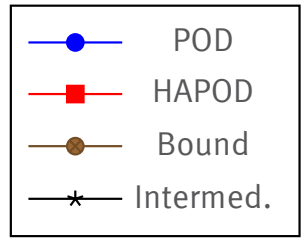
- Upwind finite difference scheme on uniform mesh with $N = 500$ nodes.
- 10^4 explicit Euler steps.
- 100 sub-PODs, $\omega = 0.75$.
- All computations on Raspberry Pi 1B single board computer (512MB RAM).

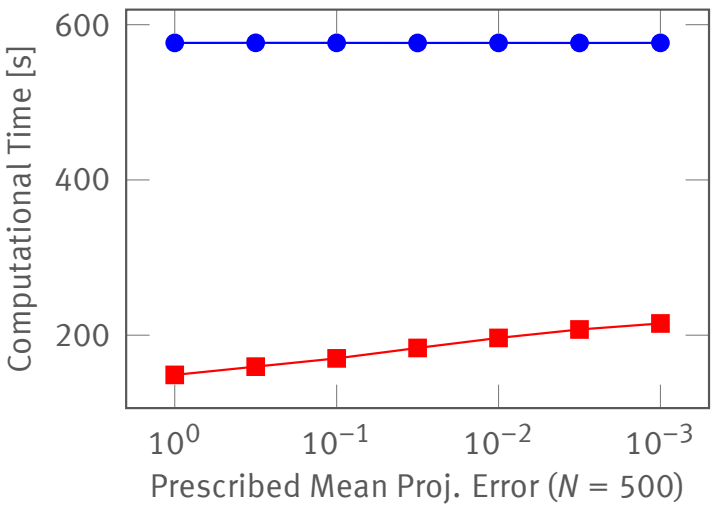


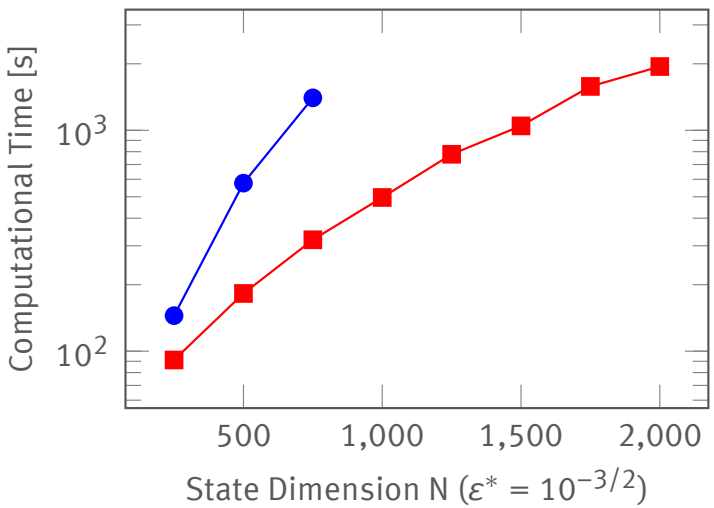
Incremental HAPOD Example









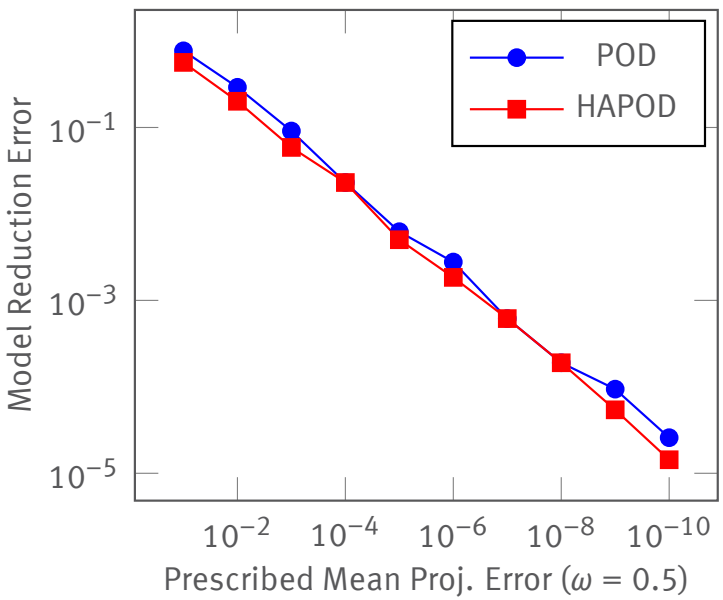


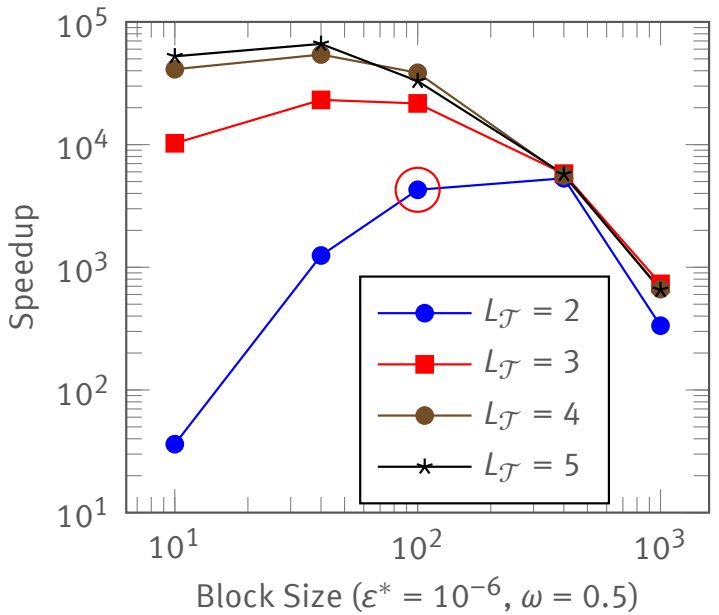
Distributed HAPOD Example

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





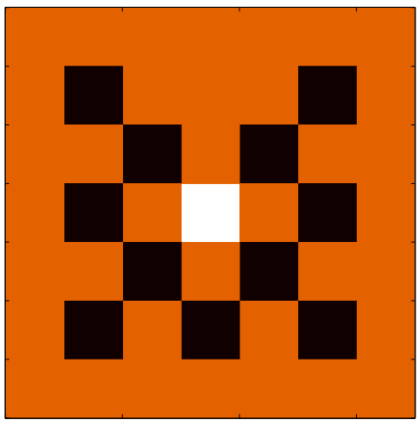
[1] See: `http://modelreduction.org/index.php/Synthetic_parametric_model`

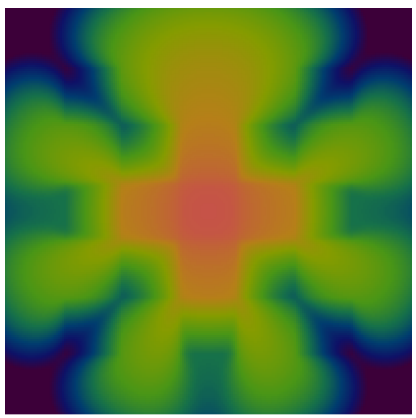
HAPOD – HPC Example

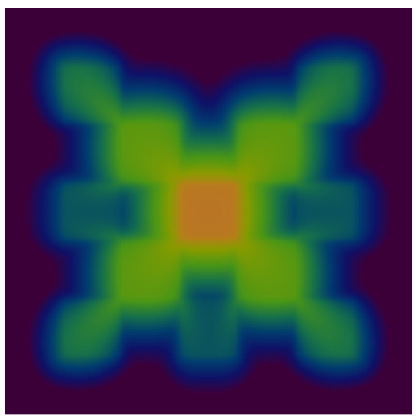
Neutron transport equation

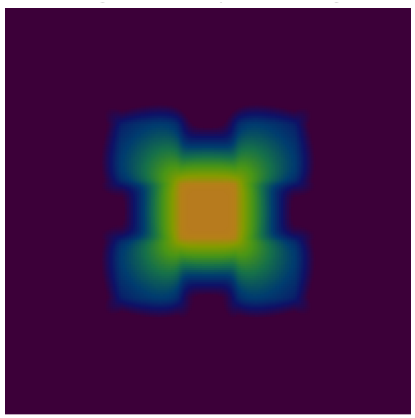
$$\partial_t \psi(t, \mathbf{x}, \mathbf{v}) + \mathbf{v} \cdot \nabla_{\mathbf{x}} \psi(t, \mathbf{x}, \mathbf{v}) + \sigma_t(\mathbf{x}) \psi(t, \mathbf{x}, \mathbf{v}) = \frac{1}{|V|} \sigma_s(\mathbf{x}) \int_V \psi(t, \mathbf{x}, \mathbf{w}) \, \mathrm{d}\mathbf{w} + Q(\mathbf{x})$$

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

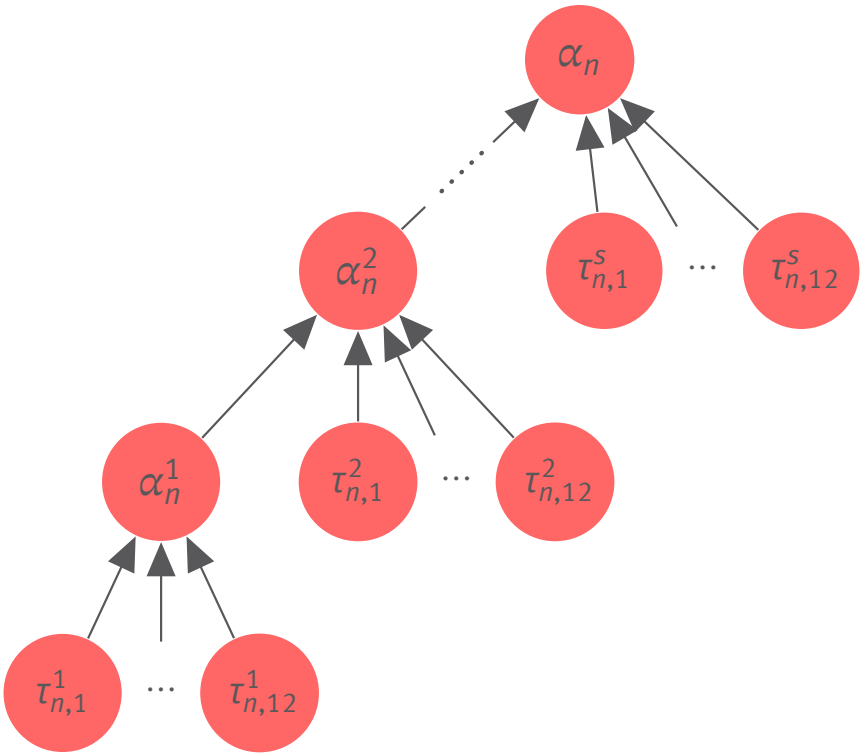




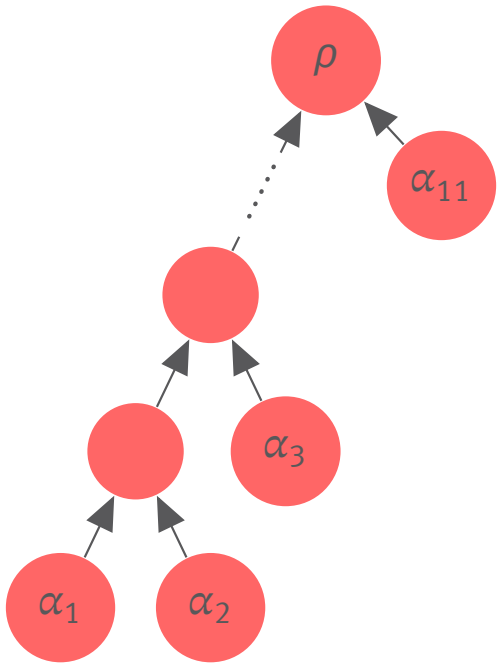




HAPOD – HPC Example

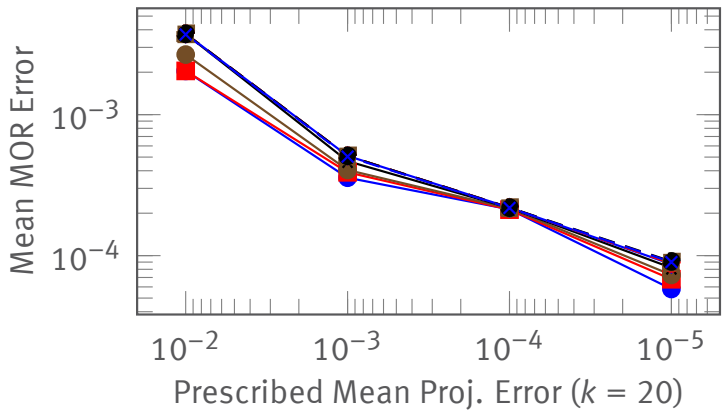


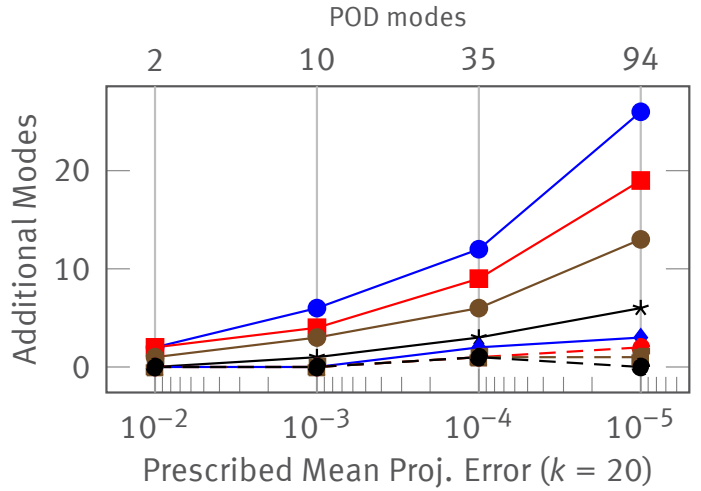
- HAPOD on compute node n. Time steps are split into s slices. Each processor core computes one slice at a time, performs POD and sends resulting modes to main MPI rank on the node.

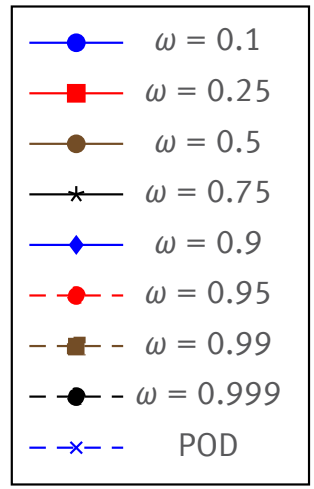


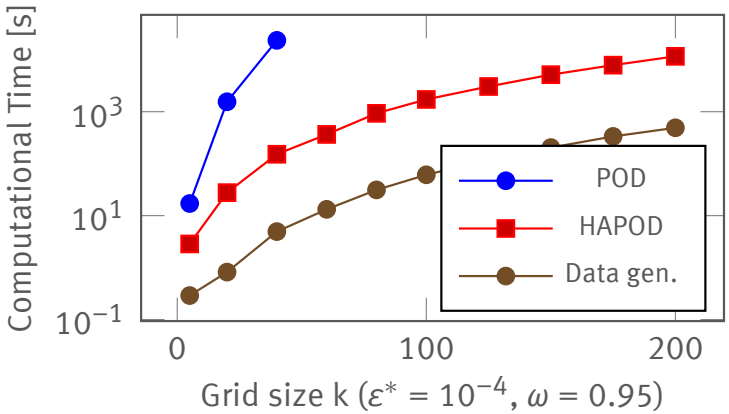
- Incremental HAPOD is performed on MPI rank 0 with modes collected on each node.

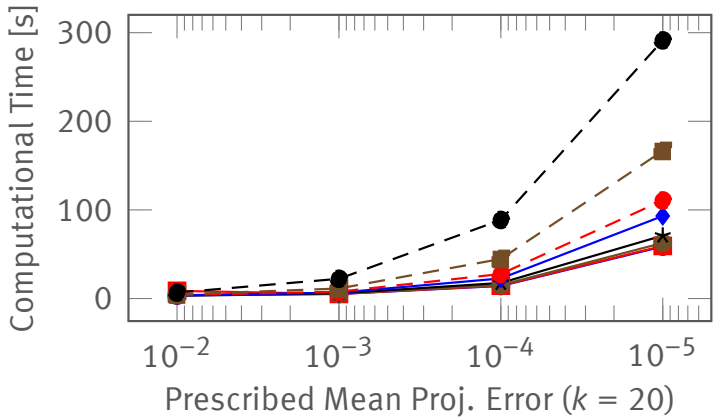
HAPOD – HPC Example











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

Data-Driven MOR with pyMOR
2. Non-intrusive MOR with ANNs

Mathematical Background

- *Two scenarios:*
  1. Given a full-order model $\mu \mapsto u(\mu)$, but no affine decomposition of operators.
  2. Given only a set $\{(\mu_i, u(\mu_i))\}_{i=1}^n$ of snapshots with corresponding parameter values, i.e. purely data-driven setting.

Mathematical Background

- *Two scenarios:*
  1. Given a full-order model $\mu \mapsto u(\mu)$, but no affine decomposition of operators.
  2. Given only a set $\{(\mu_i, u(\mu_i))\}_{i=1}^n$ of snapshots with corresponding parameter values, i.e. purely data-driven setting.

- *Goal:* Approximate the map $\pi_N : \mathcal{P} \to \mathbb{R}^N$, given as

$$\pi_N(\mu) = \underline{u}_N(\mu) \in \mathbb{R}^N,$$

  where $\underline{u}_N(\mu)$ holds the coefficients of the orthogonal projection $u_N(\mu)$ of the full-order solution $u(\mu)$ onto the basis Ψ_N of a reduced space V_N.

Mathematical Background

- *Two scenarios:*
  1. Given a full-order model $\mu \mapsto u(\mu)$, but no affine decomposition of operators.
  2. Given only a set $\{(\mu_i, u(\mu_i))\}_{i=1}^n$ of snapshots with corresponding parameter values, i.e. purely data-driven setting.

- *Goal:* Approximate the map $\pi_N : \mathcal{P} \to \mathbb{R}^N$, given as

$$\pi_N(\mu) = \underline{u}_N(\mu) \in \mathbb{R}^N,$$

  where $\underline{u}_N(\mu)$ holds the coefficients of the orthogonal projection $u_N(\mu)$ of the full-order solution $u(\mu)$ onto the basis Ψ_N of a reduced space V_N.

$\longrightarrow$ No operators required, no Galerkin projection, only solution snapshots!

Mathematical Background

- *Two scenarios:*
  1. Given a full-order model $\mu \mapsto u(\mu)$, but no affine decomposition of operators.
  2. Given only a set $\{(\mu_i, u(\mu_i))\}_{i=1}^n$ of snapshots with corresponding parameter values, i.e. purely data-driven setting.

- *Goal:* Approximate the map $\pi_N : \mathcal{P} \to \mathbb{R}^N$, given as

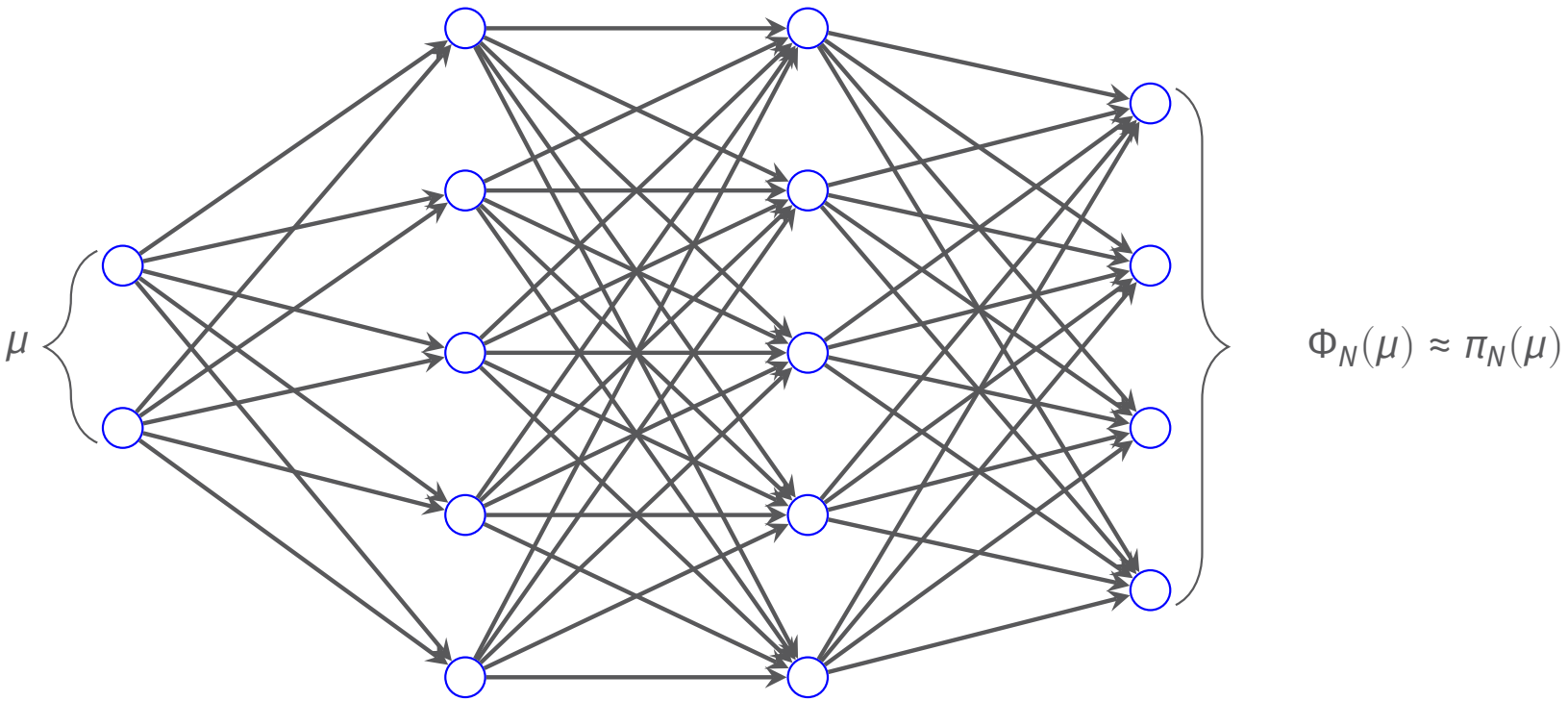
$$\pi_N(\mu) = \underline{u}_N(\mu) \in \mathbb{R}^N,$$

  where $\underline{u}_N(\mu)$ holds the coefficients of the orthogonal projection $u_N(\mu)$ of the full-order solution $u(\mu)$ onto the basis Ψ_N of a reduced space V_N.

- *Idea:* Use a neural network Φ_N to approximate π_N.

Jan S. Hesthaven and Stefano Ubbiali: Non-intrusive reduced order modeling of non-linear problems using neural networks.
J. Comput. Phys., 363:55-78, 2018.

Mathematical Background – Visualization of the Approach



Mathematical Background – Error Analysis

Components:

- High-fidelity space V with $\dim V = n$.
- Reduced subspace $V_N \subset V$ of dimension $\dim V_N = N \ll n$.
- Orthonormal basis Ψ_N of V_N.
- Matrix $\underline{\Psi}_N \in \mathbb{R}^{n \times N}$ with orthonormal columns formed by elements from Ψ_N.
- Approximation Φ_N of the map π_N, where

$$\pi_N(\mu) := \underline{\Psi}_N^\top u(\mu) = \underline{\Psi}_N^\top u_N(\mu) = \underline{u}_N(\mu).$$

Mathematical Background – Error Analysis

Components:

- High-fidelity space V with $\dim V = n$.
- Reduced subspace $V_N \subset V$ of dimension $\dim V_N = N \ll n$.
- Orthonormal basis Ψ_N of V_N.
- Matrix $\underline{\Psi}_N \in \mathbb{R}^{n \times N}$ with orthonormal columns formed by elements from Ψ_N.
- Approximation Φ_N of the map π_N, where

$$\pi_N(\mu) := \underline{\Psi}_N^\top u(\mu) = \underline{\Psi}_N^\top u_N(\mu) = \underline{u}_N(\mu).$$

Error estimate for the neural network-based approach:

$$\begin{aligned}\|u(\mu) - \underline{\Psi}_N \Phi_N(\mu)\| &\leq \|u(\mu) - \underline{\Psi}_N \pi_N(\mu)\| + \|\underline{\Psi}_N \pi_N(\mu) - \underline{\Psi}_N \Phi_N(\mu)\| \\ &= \underbrace{\|u(\mu) - u_N(\mu)\|}_{\substack{\text{best-approximation error in } V_N \\ \text{(orthogonal projection)}}} + \underbrace{\|\pi_N(\mu) - \Phi_N(\mu)\|}_{\substack{\text{approximation error} \\ \text{of the neural network}}}\end{aligned}$$

General Information on the Implementation in pyMOR

- Reduced basis Ψ_N via POD of snapshots $u(\mu_1), \dots, u(\mu_n)$.
- Neural networks implemented and trained using PyTorch (`https://pytorch.org/`).
- Training with snapshots $u(\mu_i)$ projected on reduced space V_N:

$$\{(\mu_i, \underbrace{\underline{u}_N(\mu_i)}_{=\pi_N(\mu_i)}) \in \mathcal{P} \times \mathbb{R}^N : i = 1, \dots, n\}$$

- Validation phase to assess generalization ability.
- Early stopping and multiple restarts of training.
- Customizable training routine (optimizer, epochs, learning rate, ...).

- *Everything hidden in a reductor!*

Some of the new Features in Release 2022.1

- Support for learning rate schedulers.
- Customization of early stopping scheduler.
- Scaling of inputs and outputs.
- Regularization of weights and biases.
- Weighted loss function (output components are weighted by respective singular values).
- Logging of current training and validation loss.
- Purely data-driven usage of neural network reductors.

The `NeuralNetworkReductor` in Action

```
from pymor.basic import *
# Set up the problem and the full-order model
problem = StationaryProblem(...)
fom, _ = discretize_stationary_cg(problem)
parameter_space = fom.parameters.space(...)

# Sample randomly for training and test set
training_set = parameter_space.sample_uniformly(...)
validation_set = parameter_space.sample_randomly(...)

# Set up reductor and compute the reduced-order model
from pymor.reductors.neural_network import NeuralNetworkReductor
reductor = NeuralNetworkReductor(fom, training_set, validation_set,
                                 l2_err=..., ann_mse=...)

rom = reductor.reduce(restarts=...)
```

More details: `https://docs.pymor.org/main/tutorial_mor_with_anns.html`

All available neural network-based reductors

Stationary problems:

- `NeuralNetworkReductor`
  Approximates map from parameter to reduced state.

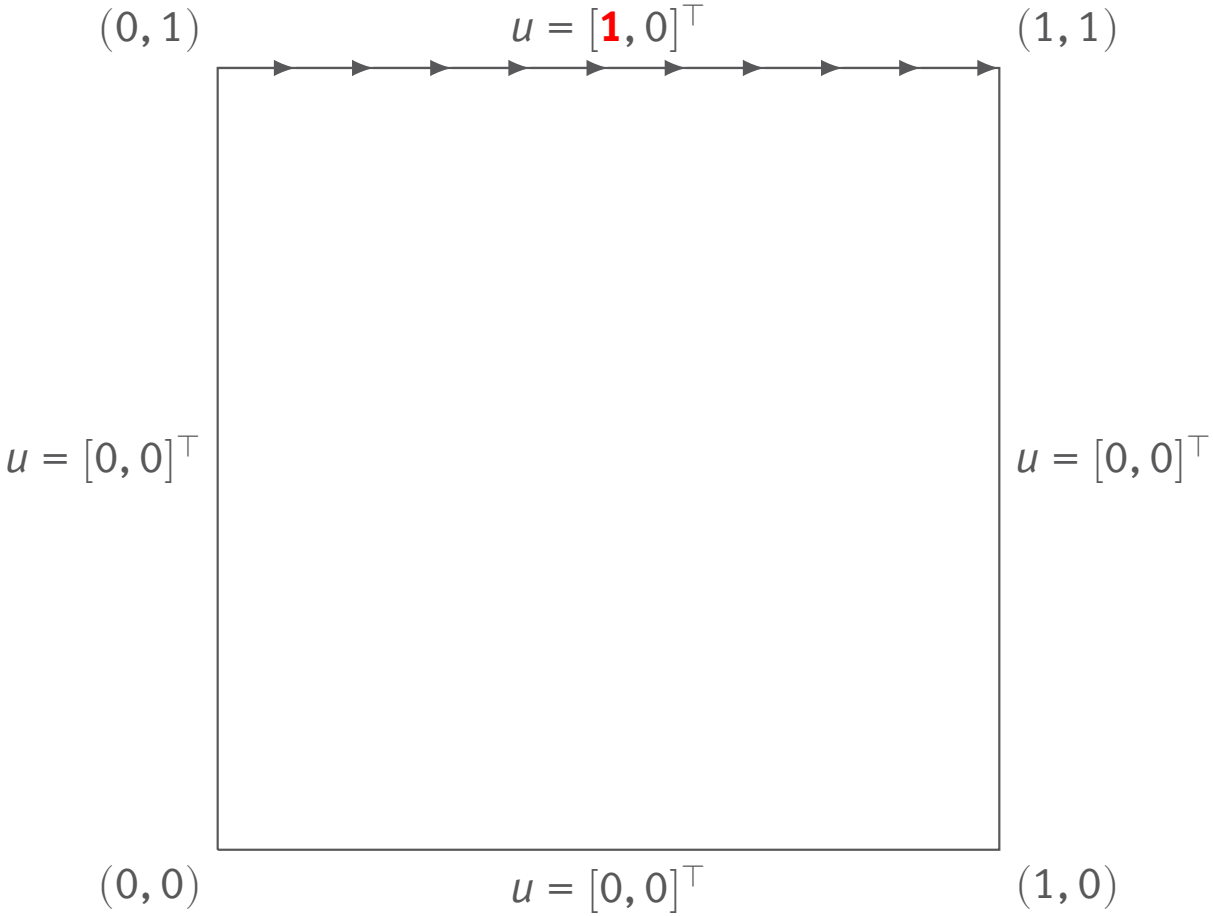
- `NeuralNetworkStatefreeOutputReductor`
  Approximates map from parameter to output.

Instationary problems:

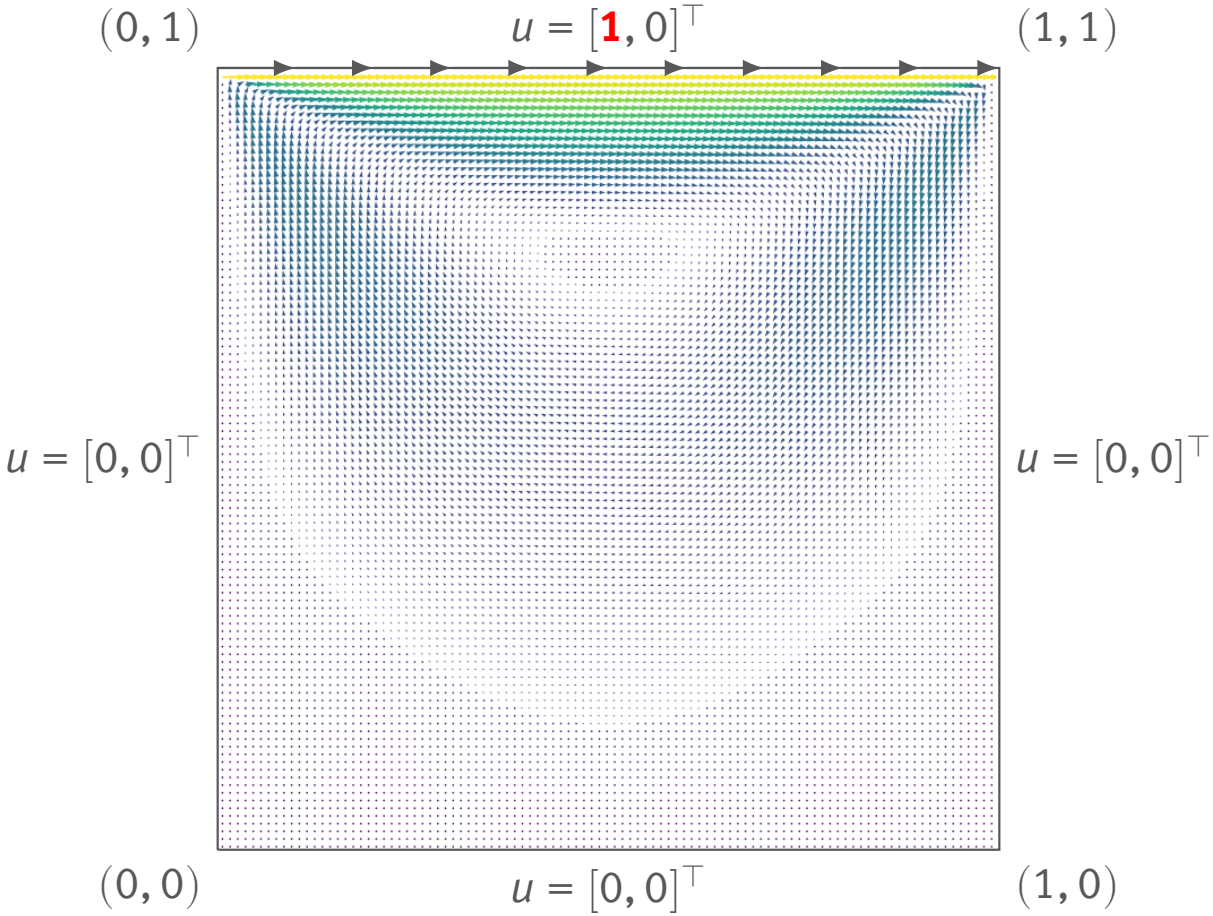
- `NeuralNetworkInstationaryReductor`
  Approximates map from parameter and time to reduced state.

- `NeuralNetworkInstationaryStatefreeOutputReductor`
  Approximates map from parameter and time to output.

FEniCS Example: Lid-Driven Cavity Flow



FEniCS Example: Lid-Driven Cavity Flow



FEniCS Example: Discretization using FEniCS

```python
from pymor.bindings.fenics import FenicsVectorSpace, FenicsOperator, FenicsMatrixOperator
from pymor.algorithms.timestepping import ImplicitEulerTimeStepper

import dolfin as df

# create square mesh
mesh = df.UnitSquareMesh(n, n)

# create Finite Elements for the pressure and the velocity
P = df.FiniteElement('P', mesh.ufl_cell(), 1)
V = df.VectorElement('P', mesh.ufl_cell(), 2, dim=2)
# create mixed element and function space
TH = df.MixedElement([P, V])
W = df.FunctionSpace(mesh, TH)

# extract components of mixed space
W_p = W.sub(0)
W_u = W.sub(1)

# define trial and test functions for mass matrix
u = df.TrialFunction(W_u)
psi_u = df.TestFunction(W_u)

# assemble mass matrix for velocity
mass_mat = df.assemble(df.inner(u, psi_u) * df.dx)
```

FEniCS Example: Discretization using FEniCS

```
# define trial and test functions
psi_p, psi_u = df.TestFunctions(W)
w = df.Function(W)
p, u = df.split(w)

# set Reynolds number, which will serve as parameter
Re = df.Constant(1.)

# define walls
top_wall = "near(x[1], 1.)"
walls = "near(x[0], 0.) | near(x[0], 1.) | near(x[1], 0.)"

# define no slip boundary conditions on all but the top wall
bcu_noslip_const = df.Constant((0., 0.))
bcu_noslip  = df.DirichletBC(W_u, bcu_noslip_const, walls)
# define Dirichlet boundary condition for the velocity on the top wall
bcu_lid_const = df.Constant((1., 0.))
bcu_lid = df.DirichletBC(W_u, bcu_lid_const, top_wall)

# fix pressure at a single point of the domain to obtain unique solutions
pressure_point = "near(x[0],  0.) & (x[1] <= " + str(2./n) + ")"
bcp_const = df.Constant(0.)
bcp = df.DirichletBC(W_p, bcp_const, pressure_point)

# collect boundary conditions
bc = [bcu_noslip, bcu_lid, bcp]
```

FEniCS Example: Discretization using FEniCS

```
mass = -psi_p * df.div(u)
momentum = (df.dot(psi_u, df.dot(df.grad(u), u)) - df.div(psi_u) * p
            + 2.*(1./Re) * df.inner(df.sym(df.grad(psi_u)), df.sym(df.grad(u))))
F = (mass + momentum) * df.dx

df.solve(F == 0, w, bc)

###### pyMOR wrapping
space = FenicsVectorSpace(W)
mass_op = FenicsMatrixOperator(mass_mat, W, W, name='mass')
op = FenicsOperator(F, space, space, w, bc,
                    parameter_setter=lambda mu: Re.assign(mu['Re'].item()),
                    parameters={'Re': 1})

fom = InstationaryModel(
    T=dt * nt,
    initial_data=VectorOperator(op.range.zeros()),
    operator=op,
    rhs=VectorOperator(op.range.zeros()),
    mass=mass_op,
    time_stepper=ImplicitEulerTimeStepper(nt=nt)
)
```

FEniCS Example: Model Order Reduction

Constructing the reduced order model is now similar to before!

```
parameter_space = fom.parameters.space(1., 50.)

training_set = parameter_space.sample_uniformly(training_samples)
validation_set = parameter_space.sample_randomly(validation_samples)

reductor = NeuralNetworkInstationaryReductor(fom, training_set,
     validation_set, basis_size=10, scale_outputs=True, ann_mse=None)
rom = reductor.reduce(hidden_layers='[30, 30, 30]', restarts=0)
```

Data-Driven MOR with pyMOR
3. Further Methods

Further Data-Driven Methods

Dynamic Mode Decomposition (`pymor.algorithms.dmd`)

- Infer discrete-time linear system

$$x_{k+1} = \hat{A} x_k$$

  from time-series data.

Further Data-Driven Methods

Dynamic Mode Decomposition (`pymor.algorithms.dmd`)

- Infer discrete-time linear system

$$x_{k+1} = \hat{A} x_k$$

  from time-series data.

- Given trajectory data

$$X := [x_0, x_1, \dots, x_K]$$

  let

$$X_0 := [x_0, x_1, \dots, x_{K-1}]$$
$$X_1 := [x_1, x_2, \dots, x_K]$$

- Exact DMD:

$$\hat{A} := \operatorname*{arg\,min}_{A \in \mathbb{R}^{K \times K}} \|X_1 - A X_0\|_F$$

Further Data-Driven Methods

Dynamic Mode Decomposition (`pymor.algorithms.dmd`)

- Exact DMD:

$$\hat{A} := \operatorname*{arg\,min}_{A \in \mathbb{R}^{K \times K}} \|X_1 - AX_0\|_F$$

- Eigenvectors/eigenvalues of $\hat{A}$ give information on the spatio-temporal dynamics of the data.
- $\hat{A}$ approximates the Koopman (composition) operator of the dynamical system.
- Reduced-order dynamics via TSVD.

Further Data-Driven Methods

Dynamic Mode Decomposition (`pymor.algorithms.dmd`)

- Exact DMD:

$$\hat{A} := \operatorname*{arg\,min}_{A \in \mathbb{R}^{K \times K}} \|X_1 - AX_0\|_F$$

- Eigenvectors/eigenvalues of $\hat{A}$ give information on the spatio-temporal dynamics of the data.
- $\hat{A}$ approximates the Koopman (composition) operator of the dynamical system.
- Reduced-order dynamics via TSVD.
- Extensions: DMD with control, ioDMD, extended DMD, DMD for continuous-time systems, sparsity promoting DMD, ...

Further Data-Driven Methods

Loewner Framework

- Construct LTI system from frequency-domain data.
- Interpolates transfer function of LTI system at given data points.
- At the same time constructs LTI system with interpolating transfer function.
- pyMOR implementation under development.

Further Data-Driven Methods

Loewner Framework

- Construct LTI system from frequency-domain data.
- Interpolates transfer function of LTI system at given data points.
- At the same time constructs LTI system with interpolating transfer function.
- pyMOR implementation under development.

Operator inference

- Learn dynamical system from data.
- Integrate a priori knowledge on the structure of the system.

Further Data-Driven Methods

Loewner Framework

- Construct LTI system from frequency-domain data.
- Interpolates transfer function of LTI system at given data points.
- At the same time constructs LTI system with interpolating transfer function.
- pyMOR implementation under development.

Operator inference

- Learn dynamical system from data.
- Integrate a priori knowledge on the structure of the system.

Manifold learning, Lift and Learn, autoencoder-based approaches, …

Thank you for your attention!

pyMOR – Generic Algorithms and Interfaces for Model Order Reduction
SIAM J. Sci. Comput., 38(5), 2016.
`http://www.pymor.org/`

System-theoretic model order reduction with pyMOR
PAMM 19, 2019.

Parametric model order reduction using pyMOR
Proceedings of MODRED 2019, Springer, 2020.

MULTIBAT: Unified Workflow for fast electrochemical 3D simulations of lithium-ion cells combining virtual stochastic microstructures, electrochemical degradation models and model order reduction
J. Comp. Sci., 2018.