Nonlinear Model Order Reduction with pyMOR
Outline

▶ Reduced Basis Methods and Empirical Interpolation.
▶ Model Order Reduction with pyMOR.
▶ Empirical Interpolation with pyMOR.

pyMOR main developers

Rene Milk  Petar Mlinarić  Stephan Rave  Felix Schindler
Reduced Basis Methods and Empirical Interpolation
RB for Nonlinear Evolution Equations

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

$$\partial_t u_\mu(t) + L_\mu(u_\mu(t)) = 0, \quad u_\mu(0) = u_0,$$

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

Reduced order problem
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.

$$\partial_t u_{\mu,N}(t) + P_N(L_\mu(u_{\mu,N}(t))) = 0, \quad u_{\mu,N}(0) = P_N(u_0),$$

where $P_N : V_h \to V_N$ is orthogonal proj. onto $V_N$.  

Stephan Rave (stephan.rave@wwu.de)
# RB for Nonlinear Evolution Equations

## Full order problem

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

$$\partial_t u_\mu(t) + L_\mu(u_\mu(t)) = 0, \quad u_\mu(0) = u_0,$$

where $L_\mu : P \times V_h \to V_h$ is a nonlinear finite volume operator.

## Reduced order problem

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.

$$\partial_t u_{\mu,N}(t) + P_N(L_\mu(u_{\mu,N}(t))) = 0, \quad u_{\mu,N}(0) = P_N(u_0),$$

where $P_N : V_h \to V_N$ is orthogonal proj. onto $V_N$.

**Problem:** Still expensive to evaluate

$$P_N \circ L_\mu : V_N \to V_h \to V_N.$$
Empirical Interpolation

**El – abstract version**

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

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

The empirical interpolant $I_M(v)$ of an arbitrary $v \in V$ is then determined by

$$I_M(v) \in \text{span}\{b_1, \ldots, b_M\} \quad \text{and} \quad \psi_m(I_M(v)) = \psi_m(v) \quad 1 \leq m \leq M.$$

**El Cheat Sheet**

<table>
<thead>
<tr>
<th></th>
<th>$V$</th>
<th>$\Psi$</th>
<th>online</th>
</tr>
</thead>
<tbody>
<tr>
<td>function El</td>
<td>function space</td>
<td>point evaluations</td>
<td>evaluation at ‘magic points’</td>
</tr>
<tr>
<td>operator El</td>
<td>range of (discrete) operator</td>
<td>DOFs</td>
<td>local evaluation at selected DOFs</td>
</tr>
<tr>
<td>matrix DEIM</td>
<td>matrices of given shape</td>
<td>matrix entries</td>
<td>assembly of selected entries</td>
</tr>
</tbody>
</table>

Stephan Rave (stephan.rave@wwu.de)
Hyper-Reduction with Empirical Operator Interpolation
(a.k.a. DEIM)

Reduced order problem (with EI)

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

$$\partial_t u_{\mu,N}(t) + \left\{ (P_N \circ I_M) \circ L_{M,\mu} \circ R_{M'} \right\}(u_{\mu,N}(t)) = 0, \quad u_{\mu,N}(0) = P_N(u_0).$$

where

- $R_{M'}: V_h \rightarrow \mathbb{R}^{M'}$ restriction to $M'$ DOFs needed for local evaluation
- $L_{M,\mu}: \mathbb{R}^{M'} \rightarrow \mathbb{R}^M$ local evaluation of $L_{\mu}$ at $M$ interpolation DOFs
- $I_M: \mathbb{R}^M \rightarrow V_h$ linear interpolation operator

Offline-Online decomposition

- Precompute the linear operators $P_N \circ I_M$ and $R_{M'}$ w.r.t. basis of $V_N$.
- Effort to evaluate $(P_N \circ I_M) \circ L_{M,\mu} \circ R_{M'}$ w.r.t. this basis:

$$\mathcal{O}(MN) + \mathcal{O}(M) + \mathcal{O}(MN).$$
Reduced Basis Approximation of Battery Models

**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.
- Huge nonlinear discrete models.
Basic Microscale Model

Variables:

- \( c \) : Li\(^+\) concentration
- \( \phi \) : electrical potential

Electrolyte:

\[
\begin{align*}
\frac{\partial c}{\partial t} - \nabla \cdot (D_e \nabla c) &= 0 \\
- \nabla \cdot \left( \kappa \frac{1 - t_+}{F} RT \frac{1}{c} \nabla c - \kappa \nabla \phi \right) &= 0
\end{align*}
\]

Electrodes:

\[
\begin{align*}
\frac{\partial c}{\partial t} - \nabla \cdot (D_s \nabla c) &= 0 \\
- \nabla \cdot (\sigma \nabla \phi) &= 0
\end{align*}
\]

Coupling: Normal fluxes at interfaces given by Butler-Volmer kinetics

\[
\begin{align*}
N_{se} &= 1 F \cdot j_{se} \\
j_{se} &= 2k \sqrt{c_e c_s (c_{max} - c_s)} \sinh \left( \frac{\eta}{2RT} \cdot F \right) \\
\eta &= \phi_s - \phi_e - U_0 \left( \frac{c_s}{c_{max}} \right)
\end{align*}
\]

- Finite volume discretization implemented by Fraunhofer ITWM in BEST.
- Numerical fluxes on interfaces := Butler-Volmer fluxes.
- Newton + algebraic multigrid solver.

Stephan Rave (stephan.rave@wwu.de)
Experiments

- 2,920,000 DOFs.
- Snapshots: 3
- \( \dim V_N = 98 + 47 \)
- \( M = 710 + 774 \)
- Rel. err.: \(< 1.5 \cdot 10^{-3} \)
- Full model: \( \approx 13 \)h
- Reduction: \( \approx 9 \)h
- Red. model: \( \approx 5 \)m
- Speedup: 154
Model Order Reduction with pyMOR
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 \( \mu \) and to compute data for reduced model.
## RB Software Design Comparison

<table>
<thead>
<tr>
<th></th>
<th>via disk</th>
<th>in solver</th>
<th>low-dim</th>
</tr>
</thead>
<tbody>
<tr>
<td>large (e.g. matrix-free) problems</td>
<td>😞</td>
<td>☺</td>
<td>☻</td>
</tr>
<tr>
<td>empirical interpolation</td>
<td>😞</td>
<td>☺</td>
<td>☻</td>
</tr>
<tr>
<td>reusability w. new solver</td>
<td>☻</td>
<td>😞</td>
<td>☻</td>
</tr>
<tr>
<td>reusability w. new MOR alg.</td>
<td>☻</td>
<td>☻</td>
<td>☻</td>
</tr>
<tr>
<td>MOR alg. easy to implement</td>
<td>☻</td>
<td>☻</td>
<td>☻</td>
</tr>
<tr>
<td>easy to use w. new solver</td>
<td>☻</td>
<td>☻</td>
<td>☻</td>
</tr>
<tr>
<td>easy to maintain</td>
<td>☻</td>
<td>☻</td>
<td>☻</td>
</tr>
<tr>
<td>MOR and solver dev. decoupled</td>
<td>☻</td>
<td>☻</td>
<td>☻</td>
</tr>
</tbody>
</table>
pyMOR – Model Order Reduction with Python

**Goal**

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

- Started late 2012, 20k lines of Python code, 3k single commits.
- BSD-licensed, fork us on Github!
- Quick prototyping with Python 2/3.
- Comes with small NumPy/SciPy-based discretization toolkit for getting started quickly.
- Seamless integration with high-performance PDE solvers.

Stephan Rave (stephan.rave@wwu.de)
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.

- Greedy basis generation with different extension algorithms.

- Automatic (Petrov-)Galerkin projection of arbitrarily nested affine combinations of operators.

- Interpolation of arbitrary (nonlinear) operators, EL-Greedy, DEIM.

- A posteriori error estimation.

- Iterative linear solvers, Newton algorithm, time-stepping algorithms.

- New! System theory Methods: balanced truncation, IRKA, … (sys-mor branch)
## RB Software Design Comparison

<table>
<thead>
<tr>
<th>Feature</th>
<th>via disk</th>
<th>in solver</th>
<th>low-dim</th>
<th>pyMOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>large (e.g. matrix-free) problems</td>
<td>🙁</td>
<td>😊</td>
<td>😊</td>
<td>😊</td>
</tr>
<tr>
<td>empirical interpolation</td>
<td>🙁</td>
<td>😊</td>
<td>😊</td>
<td>😊</td>
</tr>
<tr>
<td>reusability w. new solver</td>
<td>😊</td>
<td>🙁</td>
<td>😊</td>
<td>😊</td>
</tr>
<tr>
<td>reusability w. new MOR alg.</td>
<td>😊</td>
<td>🙁</td>
<td>😊</td>
<td>😊</td>
</tr>
<tr>
<td>MOR alg. easy to implement</td>
<td>😊</td>
<td>😊</td>
<td>🙁</td>
<td>😊</td>
</tr>
<tr>
<td>easy to use w. new solver</td>
<td>😊</td>
<td>😊</td>
<td>🙁</td>
<td>😊</td>
</tr>
<tr>
<td>easy to maintain</td>
<td>😊</td>
<td>😊</td>
<td>🙁</td>
<td>😊</td>
</tr>
<tr>
<td>MOR and solver dev. decoupled</td>
<td>😊</td>
<td>😊</td>
<td>🙁</td>
<td>😊</td>
</tr>
</tbody>
</table>
### RB Software Design Comparison

<table>
<thead>
<tr>
<th>Feature</th>
<th>via disk</th>
<th>in solver</th>
<th>low-dim</th>
<th>pyMOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>large (e.g. matrix-free) problems</td>
<td>😞</td>
<td>😊</td>
<td>😊</td>
<td>😊</td>
</tr>
<tr>
<td>empirical interpolation</td>
<td>😞</td>
<td>😊</td>
<td>😊</td>
<td>😊</td>
</tr>
<tr>
<td>reusability w. new solver</td>
<td>😊</td>
<td>😞</td>
<td>😊</td>
<td>😊</td>
</tr>
<tr>
<td>reusability w. new MOR alg.</td>
<td>😊</td>
<td>😊</td>
<td>😊</td>
<td>😊</td>
</tr>
<tr>
<td>MOR alg. easy to implement</td>
<td>😊</td>
<td>😊</td>
<td>😞</td>
<td>😊</td>
</tr>
<tr>
<td>easy to use w. new solver</td>
<td>😊</td>
<td>😊</td>
<td>😞</td>
<td>😊</td>
</tr>
<tr>
<td>easy to maintain</td>
<td>😊</td>
<td>😊</td>
<td>😞</td>
<td>😊</td>
</tr>
<tr>
<td>MOR and solver dev. decoupled</td>
<td>😊</td>
<td>😞</td>
<td>😞</td>
<td>😊</td>
</tr>
</tbody>
</table>
FEniCS Support Included

- Directly interfaces FEniCS LA backend, no copies needed.
- Use same MOR code with both backends!
- Only 150 SLOC for bindings.
- 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
deal.II Support

- **pymor-deal.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 $V_N$.  

Stephan Rave (stephan.rave@wwu.de)
NGSolve Support

- Based on NGS-Py Python bindings for NGSolve.
- See ngsolve branch of pyMOR repo.
- 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. $\mu$
and max. red. error vs. dim $V_N$. 

Stephan Rave (stephan.rave@wwu.de)
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.

<table>
<thead>
<tr>
<th>MPI ranks</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>6</th>
<th>12</th>
<th>24</th>
<th>48</th>
<th>96</th>
<th>192</th>
</tr>
</thead>
<tbody>
<tr>
<td>DUNE</td>
<td>17076</td>
<td>8519</td>
<td>5727</td>
<td>2969</td>
<td>1525</td>
<td>775</td>
<td>395</td>
<td>202</td>
<td>107</td>
</tr>
<tr>
<td>pyMOR</td>
<td>17742</td>
<td>8904</td>
<td>6014</td>
<td>3139</td>
<td>1606</td>
<td>816</td>
<td>418</td>
<td>213</td>
<td>120</td>
</tr>
<tr>
<td>overhead</td>
<td>3.9%</td>
<td>4.5%</td>
<td>5.0%</td>
<td>5.7%</td>
<td>5.3%</td>
<td>5.3%</td>
<td>6.0%</td>
<td>5.4%</td>
<td>11.8%</td>
</tr>
</tbody>
</table>
Empirical Interpolation with pyMOR
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 $\mu = 1$)

\[
-\nabla \cdot \left\{ (1 + \mu u^2(x, y)) \cdot \nabla u(x, y) \right\} = x \cdot \sin(y) \quad \text{for } x, y \in (0, 1)
\]

\[
u(x, y) = 1 \quad \text{for } x = 1
\]

\[
\nabla u(x, y) \cdot n = 0 \quad \text{otherwise}
\]

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

Determine mesh elements to visit to compute DOFs of interest

```python
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

```python
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
```
Matrix DEIM with NGSolve

Osmotic cell swelling problem

\[
\begin{align*}
\partial_t u - \alpha \Delta u &= 0 & \text{in } \Omega(t) \\
\nu_n u + \alpha \partial_n u &= 0 & \text{on } \partial \Omega(t) \\
-\beta \kappa + \gamma (u - u_0) &= \nu_n & \text{on } \partial \Omega(t)
\end{align*}
\]

Transformed concentration equation on reference domain

\[
\int_{\hat{\Omega}} J^{n+1} \hat{u}^{n+1} \hat{v} \, dx + \Delta t \int_{\hat{\Omega}} J^{n+1} \nu \cdot (F^{n+1} - T \cdot \nabla_{\hat{x}} \hat{v}) \hat{u}^{n+1} \, dx \\
+ \Delta t \int_{\hat{\Omega}} \alpha J^{n+1} (F^{n+1} - T \cdot \nabla_{\hat{x}} u) \cdot (F^{n+1} - T \cdot \nabla_{\hat{x}} \hat{v}) \, dx = \int_{\hat{\Omega}} J^n \hat{u}^n \hat{v} \, dx
\]

- Nonlinear in transformation $\Psi$, $F := \partial_x \Psi$, $J := |\det(F)|$, $\nu := \partial_t \Psi$.
- Use Matrix-DEIM w.r.t. $\Psi$. 

Stephan Rave (stephan.rave@wwu.de)
Matrix DEIM with NGSolve

Determine mesh elements to visit to compute desired matrix entries

```python
elements_to_visit = []
for el in self.range.V.Elements():
    local_global_map = []
    for i, (row, col) in enumerate(matrix_entries):
        try: local_global_map.append((i, (el.dofs.index(row), el.dofs.index(col))))
        except ValueError: pass
    if local_global_map:
        elements_to_visit.append((el.nr, local_global_map))
```

Local matrix assembly

```python
for el_nr, local_global_map in elements_to_visit
    el_id = ElementId(VOL, el_nr)
    el = fespace.GetFE(el_id)
    trafo = mesh.GetTrafo(el_id)
    el_mat = sum(i.CalcElementMatrix(el, trafo).NumPy() for i in bilinear_form.integrators)
    for global_dof, row_col in local_global_map:
        v[global_dof] += el_mat[row_col]
```
Thank you for your attention!

My homepage
http://stephanrave.de/

pyMOR – Generic Algorithms and Interfaces for Model Order Reduction
http://www.pymor.org/

MULTIBAT: Unified Workflow for fast electrochemical 3D simulations of lithium-ion cells combining virtual stochastic microstructures, electrochemical degradation models and model order reduction (submitted)
http://j.mp/multibat