Westfälische
Wilhelms-Universität
Münster

pyMOR

Generic Model Order Reduction Algorithms for MPI Distributed PDE Solvers

living.knowledge
WWU Münster

Rene Milk, Stephan Rave
and Felix Schindler

Warwick
July 4, 2016

Outline

- The Reduced Basis Method in a Nutshell
- Interfacing External Solvers with pyMOR
- Support for MPI Distributed Solvers

pyMOR Contributors

Mario Ohlberger

Rene Milk

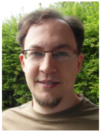
Stephan Rave

Felix Schindler

Andreas Buhr

Michael Laier

Petar Mlinarić



Michael Schaefer

The Reduced Basis Method in a Nutshell

Parametric Model Order Reduction

Consider parametric problems

$$\Phi : \mathcal{P} \to V, \qquad s : V \to \mathbb{R}^S$$

where

- $\mathcal{P} \subset \mathbb{R}^P$ *compact* set (parameter domain).
- V Hilbert space (solution state space, $\dim V \gg 0$, possibly $\dim V = \infty$).
- Φ maps parameters to solutions (*hard* to compute).
- s maps state vectors to quantities of interest.

Objective

Compute

$$s \circ \Phi : \mathbb{R}^P \to V \to \mathbb{R}^S$$

for *many* $\mu \in \mathcal{P}$ or *quickly* for unknown single $\mu \in \mathcal{P}$.

The Reduced Basis Method in a Nutshell

Objective

Compute

$$s \circ \Phi : \mathbb{R}^P \to V \to \mathbb{R}^S.$$

- When Φ, s sufficiently smooth, quickly computable low-dimensional approximation of $s \circ \Phi$ should exist.
- **Idea 1:** State space projection:
  - Define approximation $\Phi_N : \mathcal{P} \to V_N$ via Galerkin projection, $\dim V_N =: N \ll \dim V$.
  - Approximate $s \circ \Phi \approx s \circ \Phi_N$.
- **Idea 2:** $V_N \subseteq \operatorname{span}\{\Phi(\mu_1), \ldots, \Phi(\mu_k)\}$.
- **Idea 3:** Construct V_N iteratively via greedy search of $\mathcal{P}$ using quickly computable surrogate $\eta_N(\Phi_N(\mu), \mu) \geq \|\Phi(\mu) - \Phi_N(\mu)\|$.

The Easiest Case

Full order problem

$\Phi(\mu) = u_\mu \in V$ is the solution of variational problem

$$a_\mu(u_\mu, v) = f(v) \qquad \forall v \in V,$$

where $a_\mu : V \times V \to \mathbb{R}$ is continuous, coercive bilinear form, $f \in V'$.

Reduced order problem

For given $V_N \subset V$, let $\Phi_N(\mu) := u_{\mu,N} \in V_N$ be the Galerkin projection of u_μ onto V_N, i.e.

$$a_\mu(u_{\mu,N}, v) = f(v) \qquad \forall v \in V_N.$$

- Since a_μ is coercive, $u_{\mu,N}$ is well-defined.

Error Estimates

Theorem (Céa)

Let c_μ denote the coercivity constant of a_μ. Then

$$\|u_\mu - u_{\mu,N}\| \leq \frac{\|a_\mu\|}{c_\mu} \inf_{v \in V_N} \|u_\mu - v\|.$$

Proposition

The quantity $\Delta_\mu(u_{\mu,N}) := c_\mu^{-1} \cdot \|f(\cdot) - a_\mu(u_{\mu,N}, \cdot)\|_{-1}$ is a reliable and effective a posteriori estimate for the model reduction error:

$$\|u_\mu - u_{\mu,N}\| \leq \Delta_\mu(u_{\mu,N}) \leq \frac{\|a_\mu\|}{c_\mu^{-1}} \cdot \|u_\mu - u_{\mu,N}\|.$$

Offline-/Online-Decomposition

Affinely decomposed bilinear form

Have decomposition:

$$a_\mu = \sum_{q=1}^{Q} \theta_q(\mu) \cdot a_q \qquad \forall \mu \in \mathcal{P},$$

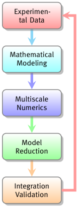
Proposition

Let $\varphi_1, \cdots, \varphi_N$ be a basis of V_N. If $\left[a_q(\varphi_l, \varphi_k)\right]_{k,l}$ are precomputed, the reduced problem can be solved with effort $\mathcal{O}(QN^2 + N^3)$.

Empirical Interpolation (EIM, DEIM)

- Replace a_μ by $\tilde{a}_\mu$ which interpolates a_μ at few selected DOFs.
- $\tilde{a}_\mu$ quickly evaluable for most standard discretization schemes.
- Use greedy algorithm to determine DOFs and interpolation basis.

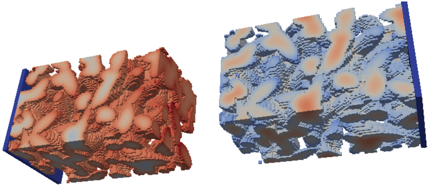
Model Order Reduction for Battery Models



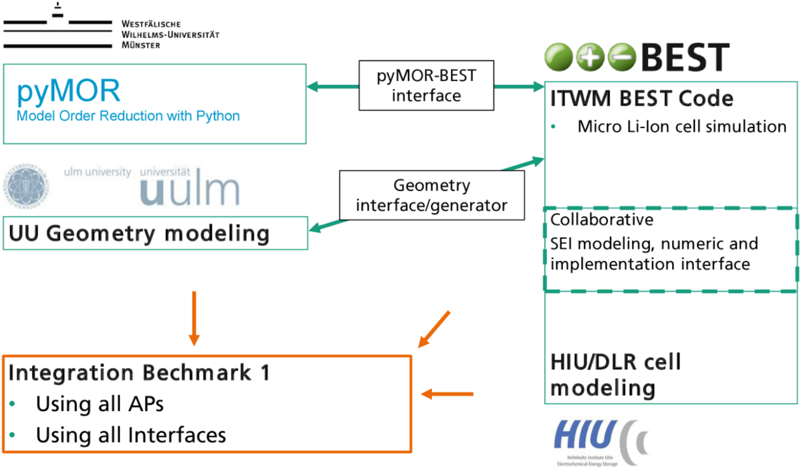
MULTIBAT: Mathematical modeling and simulation of microscale Li-ion battery models.

What to expect from MOR:

- Full model:
  1.749.600 DOFs
  time: 6.5h
- Reduced solution:
  error: $3.05 \cdot 10^{-3}$
  time: 79s
- Speedup: 290



Software Interfaces in MULTIBAT



Interfacing External Solvers with pyMOR

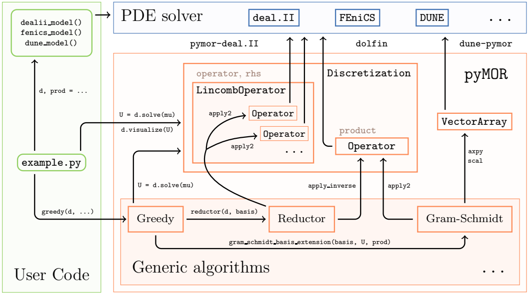
pyMOR – Model Order Reduction with Python

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.

Interfacing External Solvers



- `VectorArray`, `Operator`, `Discretization` classes represent objects in solver's memory.
- No communication of high-dimensional data.
- Possible Implementations:
  - Build solver as Python extension module.
  - Communicate with solver via network protocol.

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.

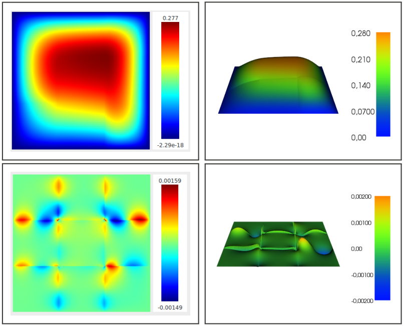


Figure: 3x3 thermal block problem
top: red. solution, bottom: red. error
left: pyMOR solver, right: FEniCS solver

deal.II Support

- `pymor-deall.II` support module (prototype)
- `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.

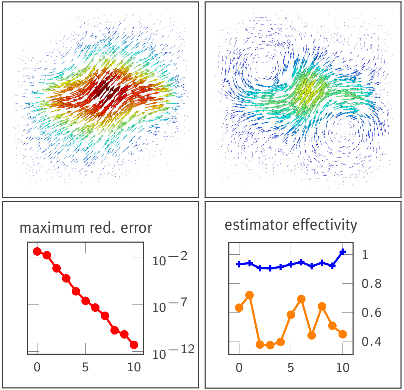


Figure: top: Solutions for $(\mu, \lambda) = (1, 1)$ and $(\mu, \lambda) = (1, 10)$, bottom: red. errs. and max./min. estimator effectivities vs. dim V_N.

Upcoming: NGSolve Support Included

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

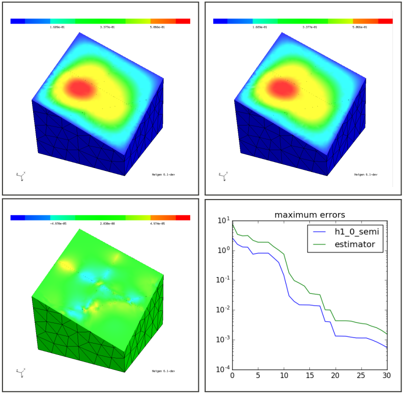


Figure: 3d thermal block problem
top: full/red. sol., bottom: err. for worst approx. μ and max. red. error vs. dim V_N.

Support for MPI Distributed Solvers

Supporting MPI distributed solvers

Goal

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

Problem:

- Solver code has to run MPI distributed.
- MOR code has to operate on MPI distributed solver objects.
- User should not have to care.

Solution:

- Launch event loop on MPI ranks $\neq 0$.
- Use proxy objects on rank 0 representing MPI distributed data.
- Implement interface methods by using event loop to dispatch into MPI distributed operations.

Implementation – Event Loop

```python
def event_loop():
    while True:
        try:
            method, args, kwargs = loads(comm.bcast(None))
            if method == 'QUIT':
                break
            else:
                method(*args, **kwargs)
        except:
            ...

def call(method, *args, **kwargs):
    assert rank0
    comm.bcast(dumps((method, args, kwargs)), root=0)
    return method(*args, **kwargs)


def function_call(f, *args, **kwargs):
    return f(*((get_object(arg) if type(arg) is ObjectId else arg) for arg in args),
             **{k: (get_object(v) if type(v) is ObjectId else v) for k, v in kwargs.iteritems()})


def function_call_manage(f, *args, **kwargs):
    return manage_object(function_call(f, *args, **kwargs))
```

Implementation – Proxy Classes

```python
class MPIVectorArrayAutoComm(MPIVectorArray):
    ...
    def l2_norm(self, ind=None):
        return mpi.call(_MPIVectorArrayAutoComm_l2_norm, self.obj_id, ind=ind)
    ...


def _MPIVectorArrayAutoComm_l2_norm(self, ind=None):
    self = mpi.get_object(self)
    local_results = self.l2_norm(ind=ind)
    results = np.empty((mpi.size,) + local_results.shape, dtype=np.float64) if mpi.rank0 else None
    mpi.comm.Gather(local_results, results, root=0)
    if mpi.rank0:
        return np.sqrt(np.sum(results ** 2, axis=0))
```

User Code

```python
# import pyMOR and launch event loop
import pymor

# imports
...

# instantiate discretization on each rank and wrap
# everything with MPI helper clasess
d = mpi_wrap_discretization(
    lambda: discretize_dune_burgers('params.ini', (1, 2)),
    array_type=MPIVectorArrayAutoComm,
    use_with=True,
)

# generate reduced model
U = d.solve(1.5)
basis, svals = pod(U, rtol=1e-3)
rd, rc, _ = reduce_generic_rb(d, basis)

# compute reduction error for same mu
u = rd.solve(1.5)
print((U - rc.reconstruct(u)).l2_norm() / U.l2_norm())
```

pyMOR's MPI support

- Automatically make sequential bindings MPI aware.
- Use same (sequential) MOR code without any change.
- Interactively debug MPI parallel solvers.

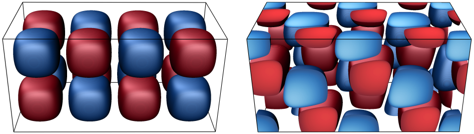
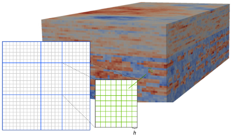


Figure: FV solution of 3D Burgers-type equation ($27.6 \cdot 10^6$ DOFs, 600 timesteps) using Dune.

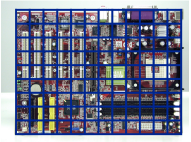
Table: Time (s) needed for solution using DUNE / DUNE with pyMOR timestepping.

MPI ranks	1	2	3	6	12	24	48	96	192
DUNE	16858	8532	5726	2959	1526	773	396	203	107
pyMOR	17683	8940	6050	3124	1604	815	417	213	110
overhead	4.9%	4.8%	5.7%	5.6%	5.1%	5.4%	5.3%	4.9%	2.8%

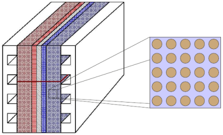
Some Projects using pyMOR



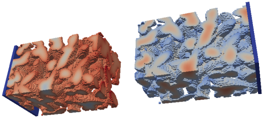
Localized Reduced Basis MultiScale method



Reduction of Maxwell's equations allowing Arbitrary Local Modifications



Reduced basis approximation for multiscale optimization problems



Reduction of microscale Li-ion battery models

Thank you for your attention!

pyMOR – Model Order Reduction with Python
http://www.pymor.org/
arXiv:1506.07094

My homepage
http://stephanrave.de/