Westfälische
Wilhelms-Universität
Münster

pyMOR

A New Model Order Reduction Software Framework

living.knowledge
WWU Münster

R. Milk, M. Ohlberger,
S. Rave, F. Schindler

SIAM CSE
Salt Lake City
March 17, 2015

People Involved

with pyMOR


Mario Ohlberger


Rene Milk

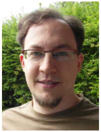

Stephan Rave


Felix Schindler


Andreas Buhr


Michael Laier


Falk Meyer


Michael Schaefer

Outline

- Reduction of Li-Ion Battery Models with the Reduced Basis Method.

- Model Order Reduction with pyMOR.

- What if you don't like pyMOR?

Reduction of Li-Ion Battery Models with the Reduced Basis Method

The MULTIBAT Project

MULTIBAT

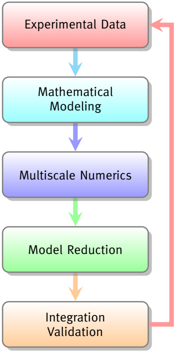
ulm university universität uulm

DLR Institute of Technical Thermodynamics

SPONSORED BY THE Federal Ministry of Education and Research

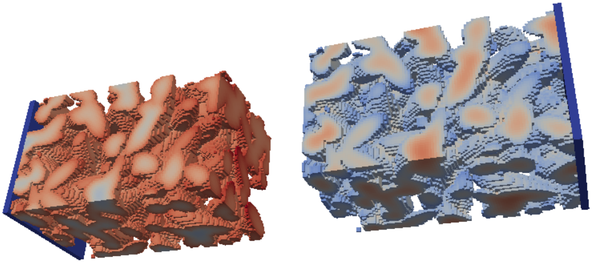
Fraunhofer ITWM

Westfälische Wilhelms-Universität Münster



- Understand degradation processes in rechargeable Li-Ion Batteries through mathematical modeling and simulation.

Microscale Battery Models



- Highly nonlinear finite volume discretization.
- Not quite easy to solve.

Microscale Battery Models

- On each part of domain (electrodes, electrolyte, current collector):

$$\frac{\partial c}{\partial t} - \nabla \cdot (\alpha(c,\phi)\nabla c + \beta(c,\phi)\nabla\phi) = 0 \qquad c : \text{Li}^+ \text{ concentration}$$

$$-\nabla \cdot (\gamma(c,\phi)\nabla c + \delta(c,\phi)\nabla\phi) = 0 \qquad \phi : \text{potential}$$

(α, β, γ, δ constant in first approximation)

- Normal fluxes at particle/electrolyte interface are given by Butler-Volmer kinetics:

$$j_{se} = 2k\sqrt{c_e c_s (c_{max} - c_s)} \sinh\left(\frac{\phi_s - \phi_e - U_0(\frac{c_s}{c_{max}})}{2RT} \cdot F\right)$$

$$N_{se} = \frac{1}{F} \cdot j_{se}$$

Microscale Model

- Finite volume discretization with implicit Euler leads to

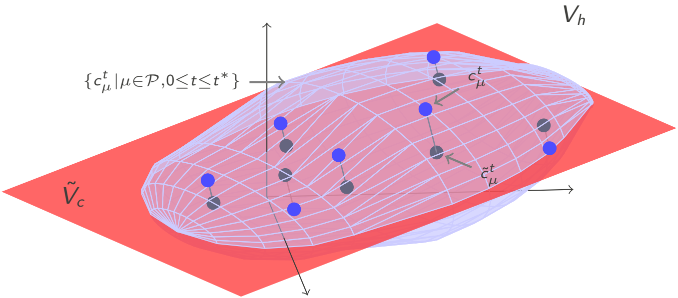
$$\begin{bmatrix} \frac{1}{\Delta t}\left(c_\mu^{(t+1)} - c_\mu^{(t)}\right) \\ 0 \end{bmatrix} + A_\mu\left(\begin{bmatrix} c_\mu^{(t+1)} \\ \phi_\mu^{(t+1)} \end{bmatrix}\right) = 0, \qquad c_\mu^{(t)}, \phi_\mu^{(t)} \in V_h$$

- Model has been implemented at Fraunhofer ITWM in BEST.

- $\mu \in \mathcal{P}$ indicates dependence on model parameters we want to vary (e.g. temperature T, charge rate).

The Reduced Basis Method

- Model order reduction technique for parameterized PDEs.
- Idea: Find solution in *problem adapted* low-dimensional reduced subspace of generic discrete function space via projection of original equation.



The Reduced Basis Method

- **Online phase:** Determine reduced solution by solving Galerkin-projected equation

$$\begin{bmatrix} \frac{1}{\Delta t}\left(\tilde{c}_\mu^{(t+1)} - \tilde{c}_\mu^{(t)}\right) \\ 0 \end{bmatrix} + \{P_{\tilde{V}} \circ A_\mu\}\left(\begin{bmatrix} \tilde{c}_\mu^{(t+1)} \\ \tilde{\phi}_\mu^{(t+1)} \end{bmatrix}\right) = 0, \qquad \begin{bmatrix} \tilde{c}_\mu^{(t)} \\ \tilde{\phi}_\mu^{(t)} \end{bmatrix} \in \tilde{V}_c \oplus \tilde{V}_\phi = \tilde{V}$$

The Reduced Basis Method

- **Online phase:** Determine reduced solution by solving Galerkin-projected equation

$$\begin{bmatrix} \frac{1}{\Delta t}\left(\tilde{c}_\mu^{(t+1)} - \tilde{c}_\mu^{(t)}\right) \\ 0 \end{bmatrix} + \{P_{\tilde{V}} \circ A_\mu\}\left(\begin{bmatrix} \tilde{c}_\mu^{(t+1)} \\ \tilde{\phi}_\mu^{(t+1)} \end{bmatrix}\right) = 0, \qquad \begin{bmatrix} \tilde{c}_\mu^{(t)} \\ \tilde{\phi}_\mu^{(t)} \end{bmatrix} \in \tilde{V}_c \oplus \tilde{V}_\phi = \tilde{V}$$

- **Offline phase:** Build $\tilde{V}_c$, $\tilde{V}_\phi$ using iterative greedy algorithm:

```
function GREEDY(S_train ⊂ P, ε, Ṽ_c^0, Ṽ_φ^0)
    Ṽ_c, Ṽ_φ ← Ṽ_c^0, Ṽ_φ^0
    while max_{μ∈S_train} ERR-EST(RB-SOLVE(μ), μ) > ε do
        μ* ← arg-max_{μ∈S_train} ERR-EST(RB-SOLVE(μ), μ)
        Ṽ_c, Ṽ_φ ← BASIS-EXT(Ṽ_c, Ṽ_φ, SOLVE(μ*))
    end while
    return Ṽ_c, Ṽ_φ
end function
```

Empirical Interpolation

- Evaluation of

$$P_{\tilde{V}} \circ A_\mu : \tilde{V}_c \oplus \tilde{V}_\phi \longrightarrow V_h \oplus V_h \longrightarrow \tilde{V}_c \oplus \tilde{V}_\phi$$

still costly.

Empirical Interpolation

- Evaluation of

$$P_{\tilde{V}} \circ A_\mu : \tilde{V}_c \oplus \tilde{V}_\phi \longrightarrow V_h \oplus V_h \longrightarrow \tilde{V}_c \oplus \tilde{V}_\phi$$

still costly.

- Use locality of finite volume operators: to evaluate M DOFs of $A_\mu(c, \phi)$ need only $M' \leq C \cdot M$ DOFs of (c, ϕ).
- Approximate

$$P_{\tilde{V}} \circ A_\mu \approx P_{\tilde{V}} \circ (I_M \circ \tilde{A}_{M,\mu} \circ R_{M'}) =: P_{\tilde{V}} \circ \mathcal{I}_M[A_\mu]$$

where

$\tilde{A}_{M,\mu}$: A_μ restricted to M interpolation DOFs
I_M: Interpolation operator
$R_{M'}$: Restriction to M' DOFs needed for evaluation

- Use greedy algorithm to determine DOFs and interpolation basis.

Software Design Challenges

- MOR-Code requires ‘access’ to PDE-solver in various ways:
  - solve for arbitrary μ.
  - projected system matrices and functionals.
  - evaluations of non-linear operators (EI-greedy).
  - orthonormalization w.r.t. inner product / Riesz map (error estimation).
  - fast evaluations of restricted non-linear operator.
  - high-dimensional reconstruction of solutions / visualization.
- PDE-solver developed independently from MOR-Code.
- **BEST**-solver should be easily replaceable by prototype Dune -solver.

Software Design Challenges

- MOR-Code requires ‘access’ to PDE-solver in various ways:
  - solve for arbitrary μ.
  - projected system matrices and functionals.
  - evaluations of non-linear operators (EI-greedy).
  - orthonormalization w.r.t. inner product / Riesz map (error estimation).
  - fast evaluations of restricted non-linear operator.
  - high-dimensional reconstruction of solutions / visualization.
- PDE-solver developed independently from MOR-Code.
- **BEST**-solver should be easily replaceable by prototype Dune -solver.
- Offline phase and online phase begin to merge:
  - online enrichment of reduced model.
  - adaptivity of high-dimensional model.

Design 1: Write Data Needed for Reduction to Disk

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.

- Advantages:
  - Easy to implement.
  - Reduction possible without direct solver access.
  - **Same MOR-Code can be used with different PDE-solvers** (e.g. toy problems, HPC-solvers).
  - MOR-Code can be written in language of choice.
- Disadvantages:
  - Have to add MOR-specific code to solver.
  - May have to adapt MOR-code *and* PDE-solver when MOR strategy changes.
  - MOR-code must be able to handle high-dimensional data.
  - Not well-suited for merging offline phase with online phase.

Design 2: Add MOR-Mode to Solver

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

- Advantages:
  - Optimal performance.
  - No additional MOR-software needed.
  - Can reuse non-linear operator for restricted operator evaluation (EI).
  - Maximum flexibility.
- Disadvantages:
  - Possibly hard to implement. (Have to use Fortran/C(++), new data structures needed.)
  - Limited code reuse, possibly not well-suited for experiments.
  - Hard to split PDE-solver and MOR-code development.

Design 3: Communicate only Reduced Data

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.

- Advantages:
  - Good performance.
  - **Same MOR-Code can be used with different PDE-solvers** (e.g. toy problems, HPC-solvers).
  - Can reuse non-linear operator for restricted operator evaluation (EI).
  - MOR-Code can be written in language of choice.
- Disadvantages:
  - Have to add MOR-specific code to solver.
  - Have to adapt MOR-code and PDE-solver when MOR strategy changes.

Model Order Reduction with pyMOR

pyMOR

- Software library for writing MOR applications, in particular with the reduced basis method.
- Joint with Felix Schindler and Rene Milk.
- Completely written in Python.
- Started late 2012, 15k lines of code, 2k single commits.
- BSD-licensed, hosted on Github.
- `http://www.pymor.org/`

Main Design Principles

- Define interfaces between MOR-code and PDE-solver:
  - Do not communicate any high-dimensional data.
  - Gain deep access to solver internals allowing various algorithms to use the same interface. (Make solver a library which can be used in arbitrary ways.)
  - No MOR-specific code should be needed inside solver.
  - Use same interfaces for reduced model.
  - Make no assumptions on how communication takes place (e.g. via network, disk, Python extension module).

Great for performance
and hacking!

Main Design Principles

- Define interfaces between MOR-code and PDE-solver:
  - Do not communicate any high-dimensional data.
  - Gain deep access to solver internals allowing various algorithms to use the same interface. (Make solver a library which can be used in arbitrary ways.)
  - No MOR-specific code should be needed inside solver.
  - Use same interfaces for reduced model.
  - Make no assumptions on how communication takes place (e.g. via network, disk, Python extension module).

Great for performance and hacking!

- Implement broad library of generic MOR-algorithms in terms of these interfaces.
  - Make it easy to tests the mathematics and care of about performance later.
  - Prefer building blocks over complete solutions.

Main Design Principles

- Provide infrastructure for writing MOR-applications:
  - Handling of parameters and parameter spaces.
  - Caching (memory, disk) of high-dimensional solutions.
  - Handling of application-wide defaults.
  - Logging.

- Implement basic high-dimensional discretizations to get started quickly.

Three Interface Classes

VectorArray

- `empty`, `zeros` — create new arrays
- `copy`, `append`, `remove`, `replace` — array management
- `scal`, `axpy`, `dot`, `lincomb` — vectorized linalg. operations

- Ordered collection of vectors of same dimension.
- Choosing **VectorArrays** (instead of vectors) as primitives allows to take advantage of vectorized (MATLAB-style) implementations.
- `NumpyVectorArray` basic type for reduced computations.
- `ListVectorArray` to the rescue if you only have vectors.

Three Interface Classes

Operator

- `apply`, `apply_adjoint` evaluate operator on **VectorArray**
- `apply_inverse` solve linear equation system
- `jacobian` return **Operator** representing Jacobian
- `restricted` return restricted **Operator** (for EI)

- Represents matrices, non-linear operators, scalar products, functionals.
- Create new **Operators** from old ones:
  `LincombOperator`, `EmpiricalInterpoalatedOperator`, `Concatenation`
  `ProjectedOperator`, `FixedParameterOperator`, ...

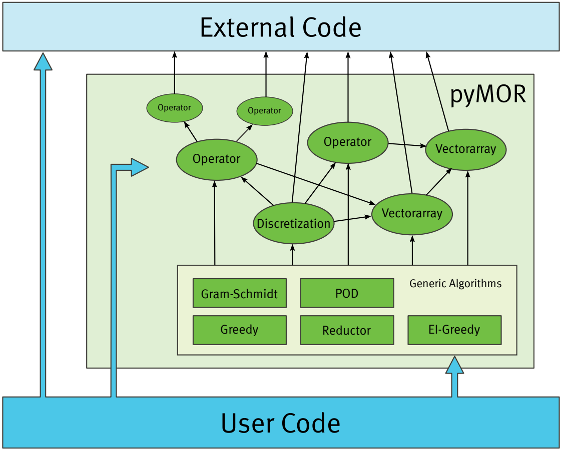
Three Interface Classes

Discretization

• `operators`, `functionals`	dictionaries of **Operators** appearing in problem
• `solve`	return **VectorArray** with solution of problem for given μ
• `estimate`	estimate error for solution **VectorArray**
• `visualize`	visualize a solution **VectorArray**

- Container for **Operators** describing problem to solve.
- Implement `solve` in terms of operations on **Operators** contained in **Discretization** or call optimized solver code.

Interfacing external PDE-solvers



Design 4: pyMOR

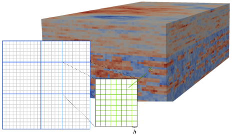
- Advantages:
  - **Same MOR-Code can be used with different PDE-solvers** (e.g. toy problems, HPC-solvers).
  - **Same solver interface can be used for various reduction methods**.
  - No MOR-specific code added to PDE-solver.
  - Can reuse non-linear operator for restricted operator evaluation (EI).
  - MOR-Code can be written in Python.
  - Very high flexibility.
  - Still very good performance.
- Disadvantages:
  - Some re-organization of PDE-solver necessary (break the main loop).

Implemented Algorithms

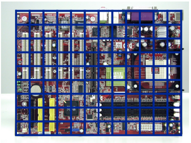
- Gram-Schmidt, POD.
- Greedy basis generation with different extension algorithms.
- Automatic reduction of arbitrarily nested affine combinations of operators.
- Interpolation of arbitrary (nonlinear) operators, EI-Greedy, DEIM.
- A posteriori error estimation.
- Iterative linear solvers, Newton algorithm.
- Time-stepping algorithms.

Main Projects

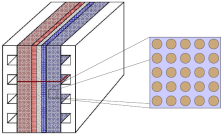
using pyMOR



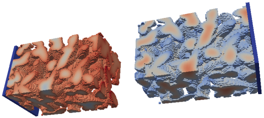
Localized Reduced Basis MultiScale method



Reduction of Maxwell's equations allowing Arbitrary Local Modifications



Reduced basis approximation for multiscale optimization problems



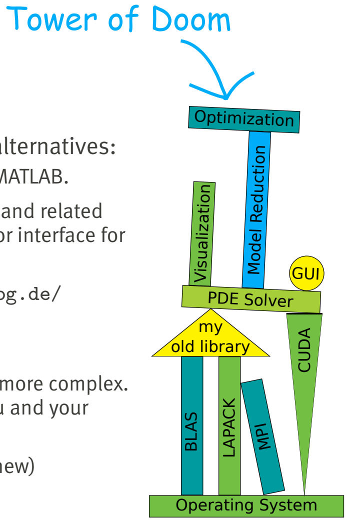
Reduction of microscale Li-ion battery models

What if you don’t like pyMOR?

I don’t like pyMOR because . . .

- I prefer MATLAB.
- I prefer procedual-style programming.
- it’s too complicated.
- does not implement anything I need.
- it’s not written by myself.

Suggestions



- Try not to reinvent the wheel. Look for alternatives:
  - RBMatlab: Reduced Basis toolbox for MATLAB.
  - modred: Python-based library for POD and related methods, parallel algorithms and vector interface for handling large datasets.
  - `http://morwiki.mpi-magdeburg.mpg.de/`
- When writing your own code:
  - Scientific software is getting more and more complex. Defining proper interfaces will help you and your co-workers.
  - Consider implementing (or defining a new) OpenInterfaces standard!

OpenInterfaces

- Common interfaces for scientific computing, e.g.:
  - problem description interface for ODEs / PDEs and control problems
  - high-level ODE / PDE solver interface
  - solver solution interface
  - internal solver algorithm and data structure interface
- Tools for bridging the language barrier. Easy interoperability between

  C++, Python, Matlab, Julia, Fortran, R
- Specification freely available and published under open licenses.
- Community driven development process.

Join us! — `http://www.openinterfaces.org/`

" {Christian Himpe, R}

Summary

Good software interfaces for MOR-software allow you to …

- easily switch between academic and large-scale problems.
- easily reuse your code for new application problems.
- compare different reduction methods for a single problem.
- evaluate a new reduction method for a variety of different problems.
- collaborate more efficiently.

Summary

Good software interfaces for MOR-software allow you to . . .

- easily switch between academic and large-scale problems.
- easily reuse your code for new application problems.
- compare different reduction methods for a single problem.
- evaluate a new reduction method for a variety of different problems.
- collaborate more efficiently.

Especially when they are established within the community.

Thank you for your attention!

AG Ohlberger
http://wwwmath.uni-muenster.de/num/ohlberger/

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

OpenInterfaces
http://www.openinterfaces.org/