dune-pyMor
Model Order Reduction with Python and Dune
pyMor

- Software package for Model Order Reduction, in particular Reduced Basis (RB) Method.

- Completely written in Python.
pyMor

▶ Software package for Model Order Reduction, in particular Reduced Basis (RB) Method.

▶ Completely written in Python.

▶ Joint with Felix Albrecht and Rene Milk.

▶ Ca. 10k lines of code.

▶ BSD-License.

▶ https://github.com/pyMor

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

- Software package for Model Order Reduction, in particular Reduced Basis (RB) Method.

- Completely written in Python.

- Joint with Felix Albrecht and Rene Milk.

- Ca. 10k lines of code.

- BSD-License.

- https://github.com/pyMor

- 0.2 release coming soon!

Stephan Rave (stephan.rave@wwu.de)
Why Python?

- Agile software development.
- Model reduction algorithms not performance-critical (in contrast to high dimensional solvers).
- NumPy delivers MATLAB-like performance for matrix operations.
- Plays nicely with other programming languages.
- 34.653 python packages on PyPI (Python Package Index).
- Open source.
- It's a great language!

Stephan Rave (stephan.rave@wwu.de)
Why Python?

- Agile software development.
Why Python?

▶ Agile software development.

▶ Model reduction algorithms not performance-critical (in contrast to high dimensional solvers).

▶ NumPy delivers MATLAB™-like performance for matrix operations.
Why Python?

▶ Agile software development.

▶ Model reduction algorithms not performance-critical (in contrast to high dimensional solvers).

▶ NumPy delivers MATLAB™-like performance for matrix operations.

▶ Plays nicely with other programming languages.

▶ 34,653 python packages on PyPI (Python Package Index).

▶ Open source.

Stephan Rave (stephan.rave@wwu.de)
Why Python?

- Agile software development.

- Model reduction algorithms not performance-critical (in contrast to high dimensional solvers).

- NumPy delivers MATLAB™-like performance for matrix operations.

- Plays nicely with other programming languages.

- 34,653 python packages on PyPI (Python Package Index).

- Open source.

- It’s a great language!

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

1. The Reduced Basis Method.
2. Design of pyMor and dune-pyMor.
3. dune-pyMor in action.
Reduced Basis Method
in a Nutshell

Discrete Problem

For given parameter $\mu \in \mathcal{P}$, find $u_{\mu,h} \in V_h$ satisfying

$$
\sum_{k=1}^{K} \theta_k(\mu) B_k(u_{\mu,h}, v_h) = F(v_h) \quad \forall v_h \in V_h.
$$

(∗)
Reduced Basis Method
in a Nutshell

Discrete Problem
For given parameter \( \mu \in \mathcal{P} \), find \( u_{\mu,h} \in V_h \) satisfying
\[
\sum_{k=1}^{K} \theta_k(\mu) B_k(u_{\mu,h}, v_h) = F(v_h) \quad \forall v_h \in V_h.
\] (\( \ast \))

- Assume that solving (\( \ast \)) is very expensive.

Stephan Rave (stephan.rave@wwu.de)
Reduced Basis Method
in a Nutshell

Discrete Problem

For given parameter $\mu \in \mathcal{P}$, find $u_{\mu,h} \in V_h$ satisfying

$$
\sum_{k=1}^{K} \theta_k(\mu) B_k(u_{\mu,h}, v_h) = F(v_h) \quad \forall v_h \in V_h.
$$

(*)&

- Assume that solving (*) is very expensive.
- Need to solve for many $\mu \in \mathcal{P}$. 

Stephan Rave (stephan.rave@wwu.de)
Reduced Basis Method
in a Nutshell

Discrete Problem

For given parameter $\mu \in \mathcal{P}$, find $u_{\mu,h} \in V_h$ satisfying

$$
\sum_{k=1}^{K} \theta_k(\mu) B_k(u_{\mu,h}, v_h) = F(v_h) \quad \forall v_h \in V_h. 
$$

\[(*)\]

- Assume that solving \((*)\) is very expensive.
- Need to solve for many $\mu \in \mathcal{P}$.
- Offline-Phase: Use some fancy algorithm to
  - compute snapshots $S := \{u_{\mu_s,h} \mid s = 1, \ldots, S\}$
  - determine $V_N \subseteq \text{span}(S)$ with $N = \dim V_N \ll \dim V_h$. 

Stephan Rave (stephan.rave@wwu.de)
Reduced Basis Method
in a Nutshell

**Discrete Problem**

For given parameter $\mu \in \mathcal{P}$, find $u_{\mu,h} \in V_h$ satisfying

$$\sum_{k=1}^{K} \theta_k(\mu) B_k(u_{\mu,h}, v_h) = F(v_h) \quad \forall v_h \in V_h.$$  

\[\text{(∗)}\]

- Assume that solving \text{(∗)} is very expensive.
- Need to solve for many $\mu \in \mathcal{P}$.
- Offline-Phase: Use some fancy algorithm to
  - compute snapshots $S := \{u_{\mu_s,h} | s = 1, \ldots, S\}$
  - determine $V_N \subseteq \text{span}(S)$ with $N = \dim V_N \ll \dim V_h$.
- Online-Phase: For new $\mu$, solve \text{(∗)} restricted to $V_N$. 

Stephan Rave (stephan.rave@wwu.de)
Reduced Basis Method
in a Nutshell

**Reduced Problem**

For given parameter $\mu \in \mathcal{P}$, find $u_{\mu, N} \in V_N$ satisfying

$$
\sum_{k=1}^{K} \theta_k(\mu) B_k(u_{\mu, N}, v_N) = F(v_N) \quad \forall v_N \in V_N.
$$

(***)

- Assume that solving (*) is very expensive.
- Need to solve for many $\mu \in \mathcal{P}$.
- Offline-Phase: Use some fancy algorithm to
  - compute snapshots $S := \{u_{\mu_s, h} \mid s = 1, \ldots, S\}$
  - determine $V_N \subseteq \text{span}(S)$ with $N = \dim V_N \ll \dim V_h$.
- Online-Phase: For new $\mu$, solve (*) restricted to $V_N$. 

Stephan Rave (stephan.rave@wwu.de)
Reduced Basis Method
in a Nutshell

Reduced Problem

\[ \sum_{k=1}^{K} \theta_k(\mu) B_k(u_{\mu,N}, v_N) = F(v_N) \quad \forall v_N \in V_N. \]  

Let \( b_1, \ldots, b_N \) be a basis of \( V_N \) and define

\[ B_k = [B_k(b_j, b_i)]_{i,j=1}^{N}, \quad F = [F(b_i)]_{i=1}^{N}. \]
Reduced Basis Method
in a Nutshell

Reduced Problem

\[ \sum_{k=1}^{K} \theta_k(\mu) B_k(u_{\mu,N}, v_N) = F(v_N) \quad \forall v_N \in V_N. \] (**) 

▶ Let \( b_1, \ldots, b_N \) be a basis of \( V_N \) and define 

\[ B_k = [B_k(b_j, b_i)]_{i,j=1}^{N} \quad F = [F(b_i)]_{i=1}^{N}. \] 

▶ In coordinates, (**) becomes 

\[ \sum_{k=1}^{K} \theta(\mu) B_k \cdot u_{\mu,N} = F \] 

with reconstruction equation 

\[ u_{\mu,N} = \sum_{i=1}^{N} b_i \cdot u_{\mu,N,i}. \] 

Stephan Rave (stephan.rave@wwu.de)
Reduced Basis Method

Example

\[ \Omega = \bigcup_{k=1}^{4} \Omega_k, \ P = [\alpha, 1]^4, \ \alpha > 0 \]

\[ a_\mu(x) = \sum_{k=1}^{4} \mu_k \cdot \chi_{\Omega_k}(x), \quad x \in \Omega, \ \mu \in P \]

Thermal-Block Problem

For \( f \in L^2(\Omega) \) and \( \mu \in P \), find \( u_\mu \in H^1_0(\Omega) \) s.t.

\[ -\nabla \cdot (a_\mu \nabla u_\mu) = f \]
Reduced Basis Method

Example

\[ \Omega = \bigcup_{k=1}^{4} \Omega_k, \ P = [\alpha,1]^4, \ \alpha > 0 \]
\[ a_\mu(x) = \sum_{k=1}^{4} \mu_k \cdot \chi_{\Omega_k}(x), \quad x \in \Omega, \ \mu \in P \]

Thermal-Block Problem

For \( f \in L^2(\Omega) \) and \( \mu \in P \), find \( u_\mu \in H^1_0(\Omega) \) s.t.
\[ \sum_{k=1}^{4} \mu_k \int_{\Omega_k} \nabla u_\mu \cdot \nabla v = \int_{\Omega} f \cdot v \quad \forall v \in H^1_0(\Omega) \]
Architecture of pyMor

Workflow

AnalyticalProblem \rightarrow \text{discretize} \rightarrow \text{Discretization} \rightarrow \text{reduce} \rightarrow \text{Discretization} \rightarrow \text{Reconstructor}
Architecture of pyMor

Workflow

AnalyticalProblem -> discretize -> Discretization

Discretization -> reduce -> Discretization

Discretization -> Operator

Reconstructor

GenericRBReconstructor

ThermalblockProblem -> discretize_elliptic_cg -> discretize

StationaryDiscretization

 rhs

operator

LincombOperator

DiffusionOperatorP₁

NumpyMatrixOperator

Operator

L₂ProductFunctionalP₁

NumpyMatrixOperator

Operator

NumpyMatrixOperator
Architecture of pyMor

Interfaces
Architecture of pyMor

Interfaces

External Code

pyMor

Operator
Operator
Operator
Operator
Discretization
Vectorarray
Vectorarray
Gram-Schmidt
POD
Greedy
Reducer
EI-Greedy

Generic Algorithms

User Code

Stephan Rave (stephan.rave@wwu.de)
Accessing External Code from pyMor
as a Python Extension Module

Stephan Rave (stephan.rave@wwu.de)
Accessing External Code from pyMor
as a Python Extension Module

▶ Implement Operators, VectorArrays, Discretizations.
Accessing External Code from pyMor
as a Python Extension Module

▶ Implement Operators, VectorArrays, Discretizations.

▶ Create Python module (e.g. using PyBindGen).
Accessing External Code from pyMor
as a Python Extension Module

- Implement Operators, VectorArrays, Discretizations.
- Create Python module (e.g. using PyBindGen).
- Write Python wrapper classes implementing the full pyMor interfaces.
DUNE-Bindings with dune-pyMor

Stephan Rave (stephan.rave@wwu.de)
DUNE-Bindings with dune-pyMor

- Implement Operators, Vectors, Discretizations by deriving from the corresponding dune-pyMor interface classes.
DUNE-Bindings with dune-pyMor

- Implement Operators, Vectors, Discretizations by deriving from the corresponding dune-pyMor interface classes.
  - Dune::DynamicMatrix and Eigen based operators already provided.
  - Bindings for LA backend of PDELab in development.
DUNE-Bindings with dune-pyMor

- Implement Operators, Vectors, Discretizations by deriving from the corresponding dune-pyMor interface classes.
  - Dune::DynamicMatrix and Eigen based operators already provided.
  - Bindings for LA backend of PDELab in development.

- Utilize provided helper methods (inject_operator, ...) and build system to semi-automatically create Python module.
DUNE-Bindings with dune-pyMor

- Implement Operators, Vectors, Discretizations by deriving from the corresponding dune-pyMor interface classes.
  - Dune::DynamicMatrix and Eigen based operators already provided.
  - Bindings for LA backend of PDELab in development.

- Utilize provided helper methods (inject_operator, ...) and build system to semi-automatically create Python module.

- In pyMor, simply call dune.pymor.core.wrap_module to obtain wrappers for all implemented DUNE classes.

Stephan Rave (stephan.rave@wwu.de)
Live Demo
Thank you for your attention!

pyMor – Model Order Reduction with Python
https://github.com/pyMor

dune-pyMor demo application
https://github.com/pyMor/dune-hdd-demos
Interfaces
VectorArrays

```python
class VectorArrayInterface(BasicInterface):
    # array creation
    @classmethod
    def empty(cls, dim, reserve=0): pass
    @classmethod
    def zeros(cls, dim, count=1): pass

    # comparing arrays
    def __len__(self): pass
    @property
    def dim(self): pass
    def almost_equal(self, other, ind, o_ind, rtol, atol): pass

    # array manipulation
    def copy(self, ind): pass
    def append(self, other, o_ind, remove_from_other=False): pass
    def remove(self, ind): pass
    def replace(self, other, ind, o_ind, remove_from_other=False): pass
    # ...
```

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

```python
# class VectorArrayInterface (continued)
# linear algebra

def scal(self, alpha, ind): pass

def axpy(self, alpha, x, ind, x_ind): pass

def dot(self, other, pairwise, ind, o_ind): pass

def lincomb(self, coefficients, ind): pass

def l2_norm(self, ind): pass

# empirical interpolation

def components(self, component_indices, ind): pass

def amax(self, ind): pass
```

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

class OperatorInterface(ImmutableInterface, Parametric, Named):
    dim_source = None
    dim_range = None
    type_source = None
    type_range = None
    linear = False
    invert_options = None

    def apply(self, U, ind, mu): pass
    def apply2(self, V, U, U_ind, V_ind, mu, product, pairwise): pass
    def apply_inverse(self, U, ind, mu, options): pass

@staticmethod
    def lincomb(operators, coefficients, ...): pass
Interfaces
RB-Projection in pyMor (more or less)

\[
\sum_{k=1}^{K} \theta_k(\mu)B_k(u_{\mu,N}, v_N) = F(v_N) \quad \forall v_N \in V_N.
\]

```python
def reduce_generic_rb(discretization, RB):
    projected_ops = {k: rb_project_operator(op, RB)
                     for k, op in discretization.operators.items()}
    rd = discretization.with_(operators=projected_operators)
    rc = GenericRBReconstructor(RB)
    return rd, rc

def rb_project_operator(op, RB):
    source_basis = RB
    range_basis = RB if op.dim_range = op.dim_source else None
    return op.projected(source_basis, range_basis)

class LincombOperatorBase(OperatorBase, LincombOperatorInterface):
    def projected(self, source_basis, range_basis):
        proj_ops = [op.projected(source_basis, range_basis)
                     for op in self.operators]
        return proj_ops[0].lincomb(proj_operators, self.coefficients)
```
Interfaces
RB-Projection in pyMor (more or less)

\[ \sum_{k=1}^{K} \theta_k(\mu) B_k(u_{\mu,N}, v_N) = F(v_N) \quad \forall v_N \in V_N. \]

```python
1 class OperatorBase(OperatorInterface):
2     def projected(self, source_basis, range_basis):
3         assert self.linear and not self.parametric
4         mat = self.apply2(source_basis, range_basis, pairwise=False)
5         return NumpyMatrixOperator(mat)
6
class GenericRBReconstructor(ImmutableInterface):
7     def __init__(self, RB):
8         self.RB = RB
9
10     def reconstruct(self, U):
11         assert isinstance(U, NumpyVectorArray)
12         return self.RB.lincomb(U.data)
```