Westfälische
Wilhelms-Universität
Münster

Reduced Basis methods with a DUNE–Matlab–Communication–Interface

wissen.leben
WWU Münster

M. Drohmann, B. Haasdonk
S. Kaulmann, M. Ohlberger

10/06/2010

› Outline

Reduced basis method

RBmatlab–Dune-RB-Interface

dune-rb Module

RBMatlab

› Reduced Basis Method Overview

RB Scenario:

- Parametrized partial differential equations for (non-stationary) problems
- Applications relying on time-critical or many repeated simulations, e.g. design, control, optimization applications.

Goals:

- Offline-/Online decomposition
- Efficient reduced simulations
- A posteriori error control

References: [Patera&Rozza, 2006], [Haasdonk et al., 2008]

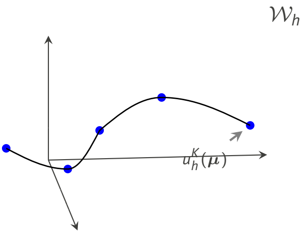
› Reduced Basis Method Overview

RB Scenario:

- Parametrized partial differential equations for (non-stationary) problems
- Applications relying on time-critical or many repeated simulations, e.g. design, control, optimization applications.

Goals:

- Offline-/Online decomposition
- Efficient reduced simulations
- A posteriori error control



References: [Patera&Rozza, 2006], [Haasdonk et al., 2008]

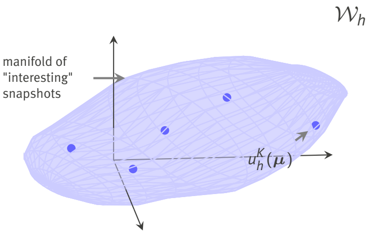
› Reduced Basis Method Overview

RB Scenario:

- Parametrized partial differential equations for (non-stationary) problems
- Applications relying on time-critical or many repeated simulations, e.g. design, control, optimization applications.

Goals:

- Offline-/Online decomposition
- Efficient reduced simulations
- A posteriori error control



References: [Patera&Rozza, 2006], [Haasdonk et al., 2008]

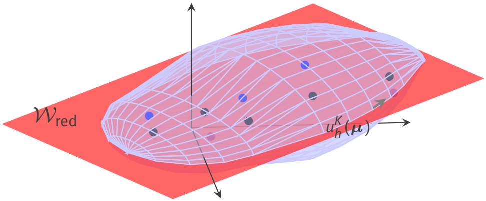
› Reduced Basis Method Overview

RB Scenario:

- Parametrized partial differential equations for (non-stationary) problems
- Applications relying on time-critical or many repeated simulations, e.g. design, control, optimization applications.

Goals:

- Offline-/Online decomposition
- Efficient reduced simulations
- A posteriori error control



References: [Patera&Rozza, 2006], [Haasdonk et al., 2008]

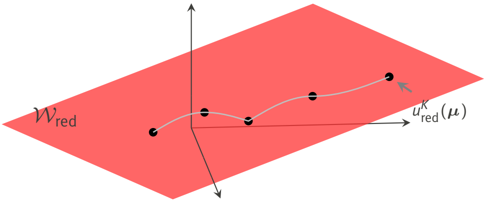
› Reduced Basis Method Overview

RB Scenario:

- Parametrized partial differential equations for (non-stationary) problems
- Applications relying on time-critical or many repeated simulations, e.g. design, control, optimization applications.

Goals:

- Offline-/Online decomposition
- Efficient reduced simulations
- A posteriori error control



References: [Patera&Rozza, 2006], [Haasdonk et al., 2008]

› Derivation of Reduced Numerical Scheme

1. Analytical formulation

For $\boldsymbol{\mu} \in \mathcal{P} \subset \mathbb{R}^p$, find $u : [0, T_{\max}] \to \mathcal{W} \subset L^2(\Omega)$, s.t.

$$\partial_t u(t) - \mathcal{L}(\boldsymbol{\mu})\,[u(t)] = 0, \qquad u(0) = u_0(\boldsymbol{\mu})$$

plus (parameter dependent) boundary conditions.

› Derivation of Reduced Numerical Scheme

1. Analytical formulation

For $\boldsymbol{\mu} \in \mathcal{P} \subset \mathbb{R}^p$, find $u : [0, T_{\max}] \to \mathcal{W} \subset L^2(\Omega)$, s.t.

$$\partial_t u(t) - \mathcal{L}(\boldsymbol{\mu})\,[u(t)] = 0, \qquad u(0) = u_0(\boldsymbol{\mu})$$

plus (parameter dependent) boundary conditions.

2. Discretization (e.g. FV,FE,DG)

For $\boldsymbol{\mu} \in \mathcal{P}$ compute $\left\{u_h^k(\boldsymbol{\mu})\right\}_{k=0}^{K} \subset \mathcal{W}_h \subset L^2(\Omega)$ by

$$u_h^0(\boldsymbol{\mu}) := P_h[u_0(\boldsymbol{\mu})]$$
$$u_h^k(\boldsymbol{\mu}) := u_h^{k-1}(\boldsymbol{\mu}) + \Delta t \mathcal{L}_h(\boldsymbol{\mu})[u_h^{k-1}(\boldsymbol{\mu})].$$

› Derivation of Reduced Numerical Scheme (cont.)

- Generate reduced basis space $\mathcal{W}_{\text{red}} := \text{span}\,\{\varphi_i\}_{i=1}^{N} \subset \mathcal{W}_h$ with POD-Greedy algorithm.
- Assume separable initial data and operator

$$u_0 = \sum_{q=1}^{Q_{u0}} \boxed{\sigma_{u0}^{q}(\boldsymbol{\mu})}\,\boxed{u_0^{q}} \qquad \mathcal{L} = \sum_{q=1}^{Q_L} \boxed{\sigma_L^{q}(\boldsymbol{\mu})}\,\boxed{\mathcal{L}^{q}}$$

  with parameter dependent and independent parts.
- Compute offline–vectors and –matrices from parameter independent contributions for numerical scheme

$$\left(\mathbf{P}^{0,q}\right)_n = \int_\Omega u_0^q \varphi_n, \qquad q = 1,\ldots,Q_{u0},\, n = 1,\ldots,N$$

$$\left(\mathbf{L}^{q}\right)_{n,m} = \int_\Omega \mathcal{L}^q\left[\varphi_m\right]\varphi_n, \qquad q = 1,\ldots,Q_L,\, m,n = 1,\ldots,N$$

› Derivation of Reduced Numerical Scheme (cont.)

- Generate reduced basis space $\mathcal{W}_{\text{red}} := \text{span}\,\{\varphi_i\}_{i=1}^N \subset \mathcal{W}_h$ with POD-Greedy algorithm.
- Assume separable initial data and operator

$$u_0 = \sum_{q=1}^{Q_{u0}} \boxed{\sigma_{u0}^q(\boldsymbol{\mu})}\,\boxed{u_0^q} \qquad \mathcal{L} = \sum_{q=1}^{Q_L} \boxed{\sigma_L^q(\boldsymbol{\mu})}\,\boxed{\mathcal{L}^q}$$

with parameter dependent and independent parts.
- Compute offline–vectors and –matrices from parameter independent contributions for numerical scheme

$$\left(\mathbf{P}^{0,q}\right)_n = \int_\Omega u_0^q \varphi_n, \qquad q = 1, \ldots, Q_{u0}, n = 1, \ldots, N$$

$$\left(\mathbf{L}^q\right)_{n,m} = \int_\Omega \mathcal{L}^q \left[\varphi_m\right] \varphi_n, \qquad q = 1, \ldots, Q_L, m, n = 1, \ldots, N$$

and a posteriori error estimation (not discussed here).

› Derivation of Reduced Numerical Scheme (cont.)

3. Reduced numerical scheme

For $\boldsymbol{\mu} \in \mathcal{P}$ compute $\left\{u_{\text{red}}^{k}(\boldsymbol{\mu})\right\}_{k=0}^{K} \subset \mathcal{W}_{\text{red}} \subset \mathcal{W}_{h}$ by

$$\begin{aligned} \left(u_{\text{red}}^{0}(\boldsymbol{\mu}) - P_h[u_0(\boldsymbol{\mu})], \varphi\right) &= 0 \quad \forall \varphi \in \mathcal{W}_{\text{red}}, \\ \left(u_{\text{red}}^{k}(\boldsymbol{\mu}) - u_{\text{red}}^{k-1}(\boldsymbol{\mu}) + \Delta t \mathcal{L}_h(\boldsymbol{\mu})\left[u_h^{k-1}(\boldsymbol{\mu})\right], \varphi\right) &= 0 \quad \forall \varphi \in \mathcal{W}_{\text{red}}, \end{aligned}$$

› Derivation of Reduced Numerical Scheme (cont.)

3. Reduced numerical scheme

For $\boldsymbol{\mu} \in \mathcal{P}$ compute $\{u^k_{\text{red}}(\boldsymbol{\mu})\}_{k=0}^{K} \subset \mathcal{W}_{\text{red}} \subset \mathcal{W}_h$ by

$$\begin{aligned} \left(u^0_{\text{red}}(\boldsymbol{\mu}) - P_h[u_0(\boldsymbol{\mu})], \varphi\right) &= 0 \quad \forall \varphi \in \mathcal{W}_{\text{red}}, \\ \left(u^k_{\text{red}}(\boldsymbol{\mu}) - u^{k-1}_{\text{red}}(\boldsymbol{\mu}) + \Delta t \mathcal{L}_h(\boldsymbol{\mu}) \left[u^{k-1}_h(\boldsymbol{\mu})\right], \varphi\right) &= 0 \quad \forall \varphi \in \mathcal{W}_{\text{red}}, \end{aligned}$$

or equivalently compute vectors $\mathbf{a}^k(\boldsymbol{\mu}) \subset \mathbb{R}^N$ for $k = 0, \ldots, K$ such that $u^k_{\text{red}}(\boldsymbol{\mu}) = \sum_{n=1}^N a^k_n \varphi_n$ by

$$\mathbf{a}^0(\boldsymbol{\mu}) := \sum_{q=1}^{Q_{u0}} \sigma^q_{u0}(\boldsymbol{\mu}) \mathbf{P}^{0,q},$$

$$\mathbf{a}^k(\boldsymbol{\mu}) := \mathbf{a}^{k-1}(\boldsymbol{\mu}) + \Delta t \sum_{q=1}^{Q_L} \sigma^q_L(\boldsymbol{\mu}) \mathbf{L}^q \mathbf{a}^{k-1}.$$

› Reduced basis generation

Use error estimator $\eta(\boldsymbol{\mu})$, s.t. $\max_{k=0,\dots,K} \left\| u_h^k(\boldsymbol{\mu}) - u_{\text{red}}^k(\boldsymbol{\mu}) \right\|_{\mathcal{W}_h} \leq \eta(\boldsymbol{\mu})$.

Reduced basis generation

Use error estimator $\eta(\boldsymbol{\mu})$, s.t. $\max_{k=0,\ldots,K} \left\| u_h^k(\boldsymbol{\mu}) - u_{\text{red}}^k(\boldsymbol{\mu}) \right\|_{\mathcal{W}_h} \leq \eta(\boldsymbol{\mu})$.

POD-greedy algorithm

INPUT: $M_{\text{train}} \subset \mathcal{P}, \varepsilon_{\text{tol}}, N_{\max}$
OUTPUT: $\mathcal{W}_{\text{red}}$

Initialize reduced basis:

$$\Phi_{N_0} \leftarrow \{\varphi_n\}_{n=1}^{N_0}$$
$$N \leftarrow N_0$$

› Reduced basis generation

Use error estimator $\eta(\boldsymbol{\mu})$, s.t. $\max_{k=0,\dots,K} \left\| u_h^k(\boldsymbol{\mu}) - u_{\text{red}}^k(\boldsymbol{\mu}) \right\|_{\mathcal{W}_h} \leq \eta(\boldsymbol{\mu})$.

POD-greedy algorithm

INPUT: $M_{\text{train}} \subset \mathcal{P}, \varepsilon_{\text{tol}}, N_{\text{max}}$
OUTPUT: $\mathcal{W}_{\text{red}}$

Initialize reduced basis:

$$\Phi_{N_0} \leftarrow \{\varphi_n\}_{n=1}^{N_0}$$
$$N \leftarrow N_0$$

repeat

1. *Find worst approximated trajectory:* $\boldsymbol{\mu}_{\text{max}} \leftarrow \arg\max_{\boldsymbol{\mu} \in M_{\text{train}}} \eta(\boldsymbol{\mu})$
2. *Compute trajectory* $\left\{ u_h^k(\boldsymbol{\mu}_{\text{max}}) \right\}_{k=0}^{K}$.
3. *Compute new basis function:*

$$\varphi_{N+1} \leftarrow \text{POD}\left(\left\{ u_h^k(\boldsymbol{\mu}_{\text{max}}) - P_{\text{red}}\left[u_h^k(\boldsymbol{\mu}_{\text{max}}) \right] \right\}_{k=0}^{K} \right)$$
$$N \leftarrow N + 1$$

until $\eta(\boldsymbol{\mu}_{\text{max}})$ falls beneath given tolerance ε_{tol} **or** $N = N_{\text{max}}$.

$$\mathcal{W}_{\text{red}} \leftarrow \text{span}\,\{\varphi_n\}_{n=1}^{N}$$

wissen.leben WWU Münster

› Reduced basis generation

Use error estimator $\eta(\boldsymbol{\mu})$, s.t. $\max_{k=0,\dots,K} \left\| u_h^k(\boldsymbol{\mu}) - u_{\text{red}}^k(\boldsymbol{\mu}) \right\|_{\mathcal{W}_h} \leq \eta(\boldsymbol{\mu})$.

POD-greedy algorithm

INPUT: $M_{\text{train}} \subset \mathcal{P}, \varepsilon_{\text{tol}}, N_{\max}$
OUTPUT: $\mathcal{W}_{\text{red}}$

Initialize reduced basis:

$$\Phi_{N_0} \leftarrow \{\varphi_n\}_{n=1}^{N_0}$$
$$N \leftarrow N_0$$

repeat

1. *Find worst approximated trajectory:* $\boldsymbol{\mu}_{\max} \leftarrow \arg\max_{\boldsymbol{\mu} \in M_{\text{train}}} \eta(\boldsymbol{\mu})$
2. *Compute trajectory* $\left\{ u_h^k(\boldsymbol{\mu}_{\max}) \right\}_{k=0}^K$.
3. *Compute new basis function:*

$$\varphi_{N+1} \leftarrow \text{POD}\left(\left\{ u_h^k(\boldsymbol{\mu}_{\max}) - P_{\text{red}}\left[u_h^k(\boldsymbol{\mu}_{\max}) \right] \right\}_{k=0}^K \right)$$
$$N \leftarrow N + 1$$

until $\eta(\boldsymbol{\mu}_{\max})$ falls beneath given tolerance ε_{tol} **or** $N = N_{\max}$.

$$\mathcal{W}_{\text{red}} \leftarrow \text{span}\, \{\varphi_n\}_{n=1}^N$$

wissen.leben
WWU Münster

› What else?

- Reduced basis methods also work for nonlinear schemes and non-separable operators. ($\Rightarrow$ empirical interpolation)
- A posteriori error estimators with offline/-online decomposition
- More sophisticated reduced basis generation algorithms

› What else?

- Reduced basis methods also work for nonlinear schemes and non-separable operators. (⇒ empirical interpolation)
- A posteriori error estimators with offline/-online decomposition
- More sophisticated reduced basis generation algorithms

But you got the idea:

1. Usage of existing numerical schemes for basis generation
2. Low-dimensional and efficient online computations

› What else?

- Reduced basis methods also work for nonlinear schemes and non-separable operators. ($\Rightarrow$ empirical interpolation)
- A posteriori error estimators with offline/-online decomposition
- More sophisticated reduced basis generation algorithms

But you got the idea:

1. Usage of existing numerical schemes for basis generation
2. Low-dimensional and efficient online computations

For 1, we want to use DUNE, for 2, we don't.

› Motivation: Comparison of Matlab and Dune

Matlab

+ Easy to learn and use
+ Huge library of mathematical functions (statistical data, postprocessing, plots,...)
- Slow for interpreted code parts
- Memory constraints

Dune

+ Flexible and efficient
+ Provides complex numerical schemes
- Less easy to learn

› Motivation: Comparison of Matlab and Dune

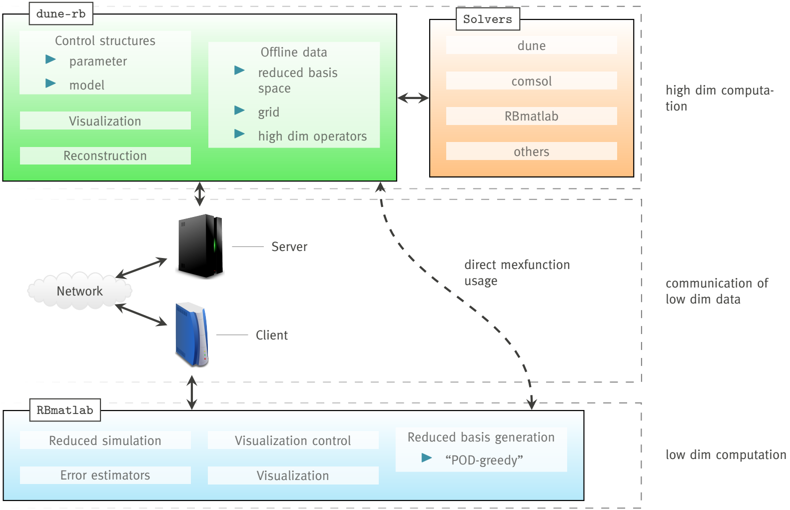
Matlab

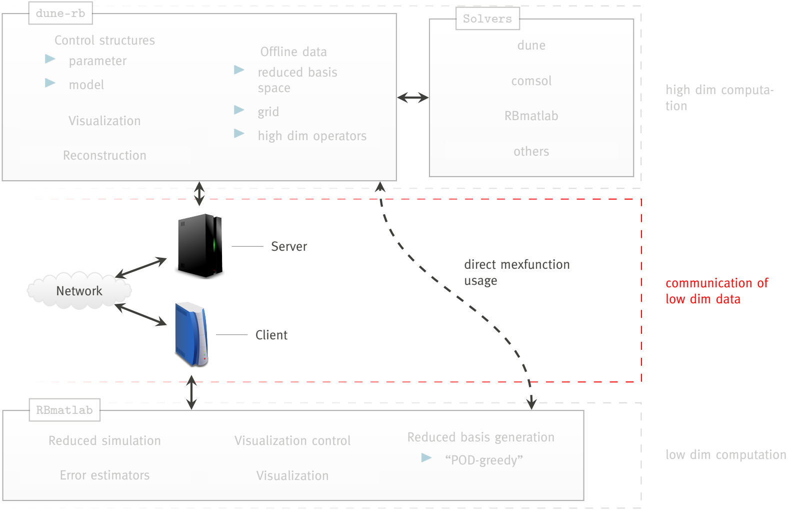
- + Easy to learn and use
- + Huge library of mathematical functions (statistical data, postprocessing, plots,...)
- - Slow for interpreted code parts
- - Memory constraints

Dune

- + Flexible and efficient
- + Provides complex numerical schemes
- - Less easy to learn

$\Rightarrow$ Separate implementation of offline-/online-phases in RBmatlab and dune-rb, respectively.


dune-rb
Control structures
parameter
model
Visualization
Reconstruction
Offline data
reduced basis space
grid
high dim operators
Solvers
dune
comsol
RBmatlab
others
Server
Network
Client
direct mexfunction usage
RBmatlab
Reduced simulation
Error estimators
Visualization control
Visualization
Reduced basis generation
"POD-greedy"
high dim computation
communication of low dim data
low dim computation


dune-rb
Control structures
parameter
model
Visualization
Reconstruction
Offline data
reduced basis space
grid
high dim operators
Solvers
dune
comsol
RBmatlab
others
high dim computation
Server
Network
Client
direct mexfunction usage
communication of low dim data
RBmatlab
Reduced simulation
Error estimators
Visualization control
Visualization
Reduced basis generation
"POD-greedy"
low dim computation

› Data Structures

Matlab provides a C data structure `mxArray*` that can be used as

- Matrix
- String
- Container-Type (Struct, Cell-Array)

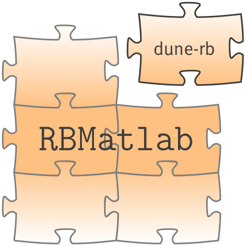
dune-rb provides C++ wrappers

- MXArray (internal data stored as `mxArray*`)
- RBArray (internal data stored as `double*`, `std::string`)

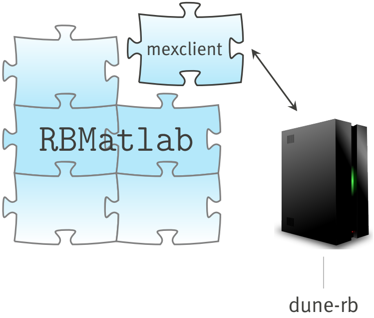
derived from a common interface `SerializedArray`.

› Transmission of Data

Use dune-rb as
Matlab (mex) library



Use dune-rb as
standalone server



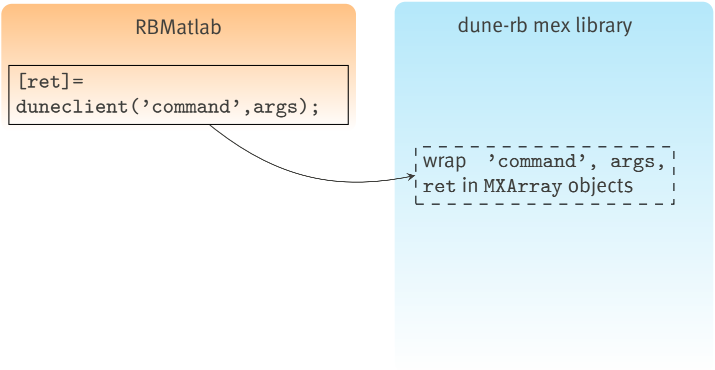
› Communication with mex Library

RBMatlab

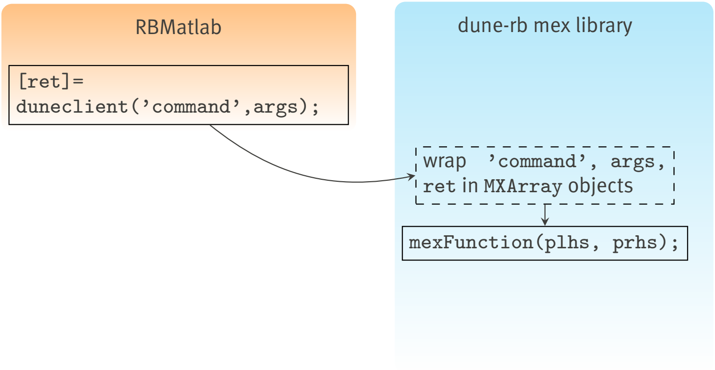
```
[ret]=
duneclient('command',args);
```

dune-rb mex library

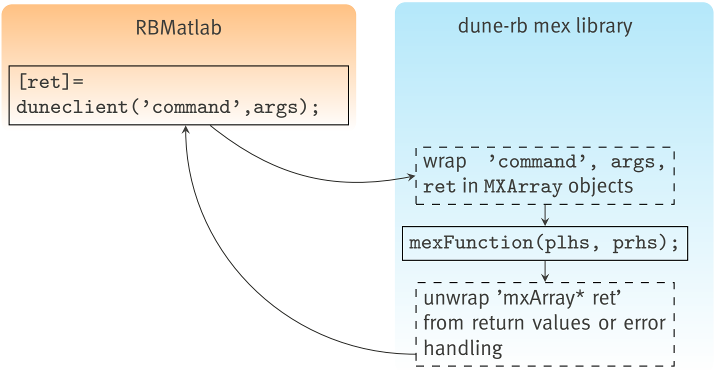
› Communication with mex Library



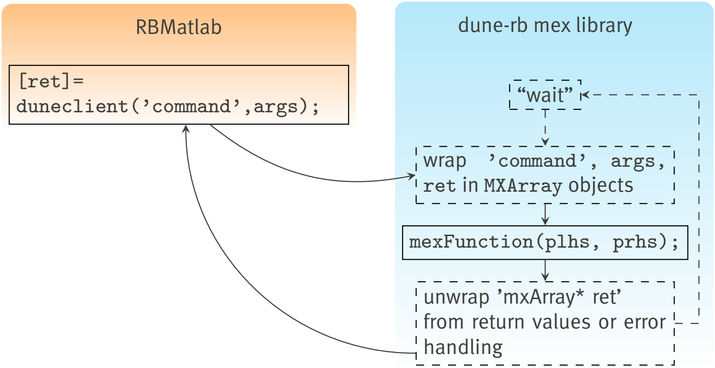
› Communication with mex Library



› Communication with mex Library



› Communication with mex Library



› Communication over TCP/IP

RBMatlab

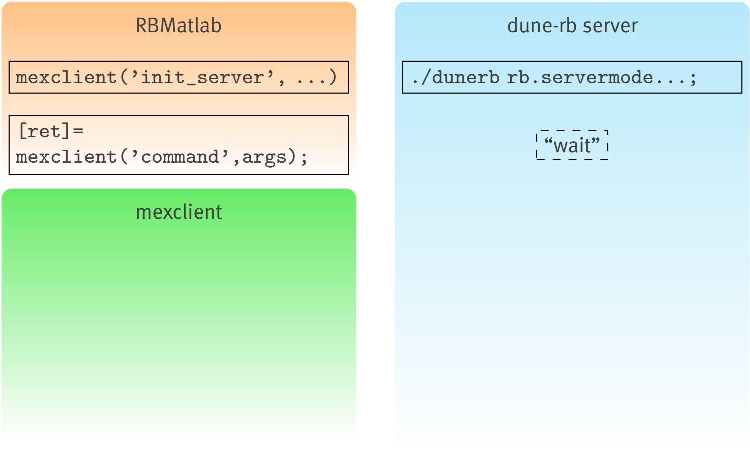
```
mexclient('init_server',
struct('serverhost',
'localhost', 'port', 1234))
```

mexclient

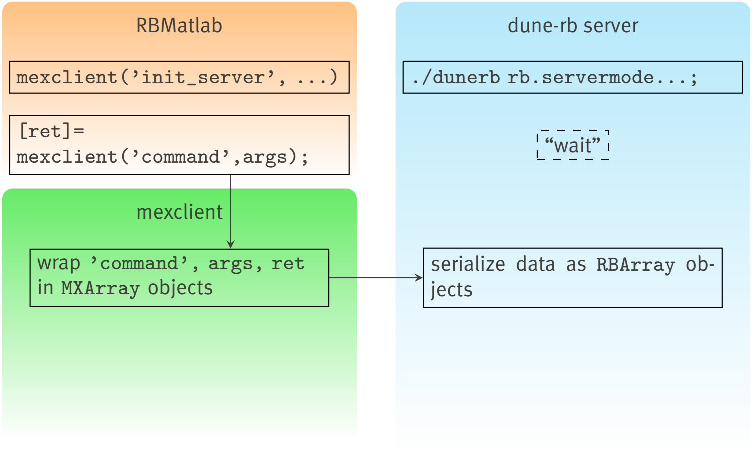
dune-rb server

```
./dunerb rb.servermode:true
rb.port:1234;
```

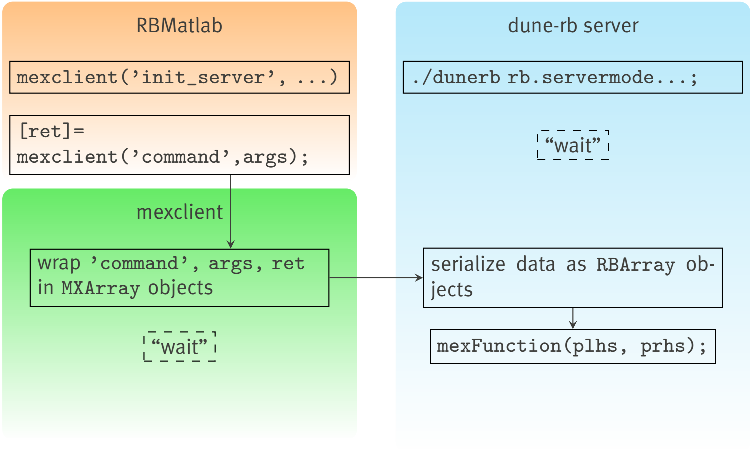
› Communication over TCP/IP



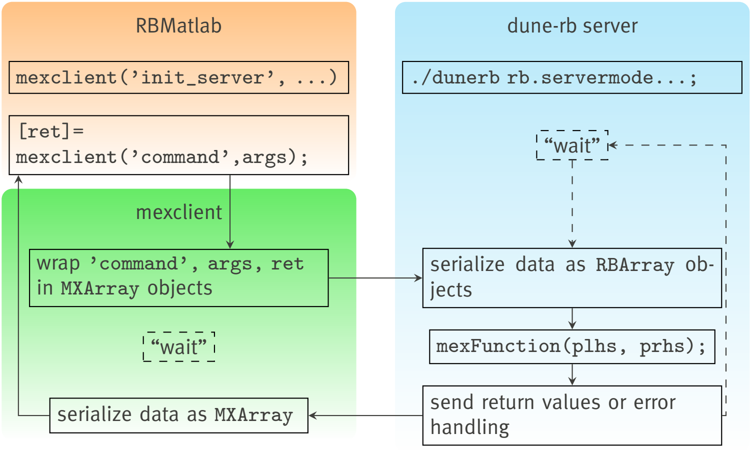
› Communication over TCP/IP



› Communication over TCP/IP



› Communication over TCP/IP



› dune-rb Module

Goals

- Enhance dune-fem operators with parametrization
  - Affine parametrized operators
  - Library of finite volume operators and problems
- Storage and management of reduced bases
  - discrete function lists
  - reduced basis space
- High-dimensional matrix computations
  - PCA
  - Gramian matrix computation
- Reconstruction and visualization of reduced basis solutions.

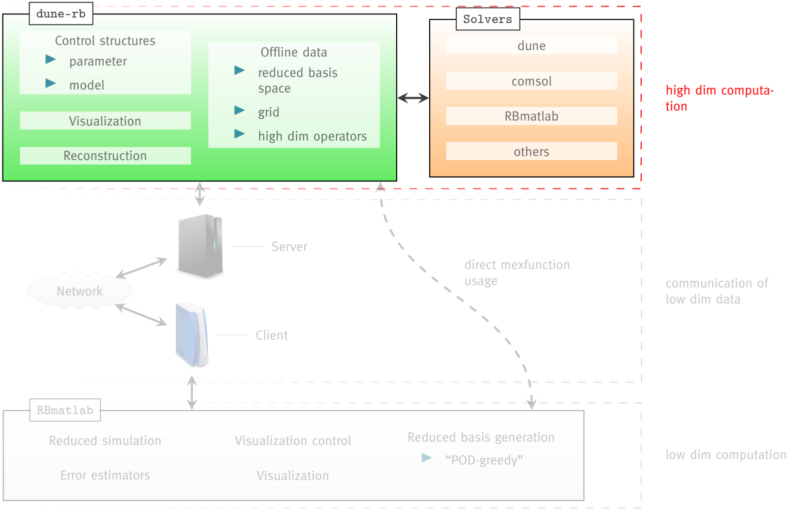
› dune-rb Module

Goals

- Enhance dune-fem operators with parametrization
- Storage and management of reduced bases
- High-dimensional matrix computations
- Reconstruction and visualization of reduced basis solutions.

Dependencies

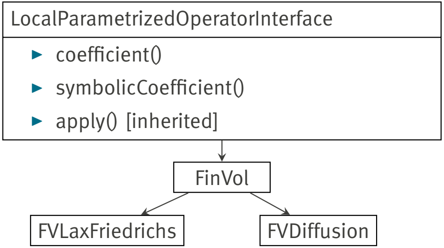
- dune-common, dune-grid
- dune-fem
- LAPACK (for PCA)
- Matlab (for mexclient)


dune-rb
Control structures
parameter
model
Visualization
Reconstruction
Offline data
reduced basis space
grid
high dim operators
Solvers
dune
comsol
RBmatlab
others
high dim computation
Server
Network
Client
direct mexfunction usage
communication of low dim data
RBmatlab
Reduced simulation
Error estimators
Visualization control
Visualization
Reduced basis generation
"POD-greedy"
low dim computation

› Separable Discrete Operators

$$\mathcal{L} \quad := \sum_{q=1}^{Q} \sigma_L^q \mathcal{L}^q$$

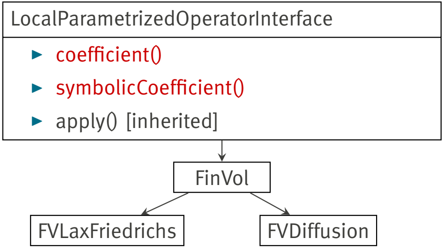
Implement a class for each summand as instance of `LocalParametrizedOperatorInterface`.



› Separable Discrete Operators

$$\mathcal{L} \quad := \sum_{q=1}^{Q} \sigma_L^q \mathcal{L}^q$$

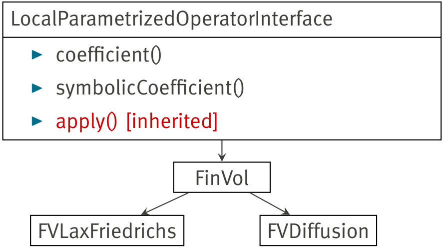
Implement a class for each summand as instance of `LocalParametrizedOperatorInterface`.



› Separable Discrete Operators

$$\mathcal{L}[u] := \sum_{q=1}^{Q} \sigma_L^q \mathcal{L}^q[u]$$

Implement a class for each summand as instance of `LocalParametrizedOperatorInterface`.



› Separable Discrete Operators

$$\mathcal{L} := \sum_{q=1}^{Q} \sigma_L^q \mathcal{L}^q$$

Implement a class for each summand as instance of `LocalParametrizedOperatorInterface`.

```
typedef Tuple<LocalParametrizedOperatorImp,...> LPOTuple;
LPOTuple lpoTuple(lpo1,...);
DiscreteDecomposedOperator<LPOTuple,
                           DiscreteFunction, // type of domain
                           DiscreteFunction> // type of range
  ddo(discreteFunctionSpace, lpoTuple);

ddo.coefficients();
ddo.complete();
ddo.component(i);
```

› Reduced Basis Space (by M. Nolte)

- Derived from `DiscreteFunctionSpaceDefault`

› Reduced Basis Space (by M. Nolte)

- Derived from `DiscreteFunctionSpaceDefault`
- `baseFunctionSet(const Entity& en)` provides restriction of the base functions to the entity `en`

› Reduced Basis Space (by M. Nolte)

- Derived from `DiscreteFunctionSpaceDefault`
- `baseFunctionSet(const Entity& en)` provides restriction of the base functions to the entity `en`
- Allows manipulation of base functions:

› Reduced Basis Space (by M. Nolte)

- Derived from `DiscreteFunctionSpaceDefault`
- `baseFunctionSet(const Entity& en)` provides restriction of the base functions to the entity `en`
- Allows manipulation of base functions:
  - `addBaseFunction()`

› Reduced Basis Space (by M. Nolte)

- Derived from `DiscreteFunctionSpaceDefault`
- `baseFunctionSet(const Entity& en)` provides restriction of the base functions to the entity `en`
- Allows manipulation of base functions:
  - `addBaseFunction()`
  - `setBaseFunction()`

› Reduced Basis Space (by M. Nolte)

- Derived from `DiscreteFunctionSpaceDefault`
- `baseFunctionSet(const Entity& en)` provides restriction of the base functions to the entity `en`
- Allows manipulation of base functions:
  - `addBaseFunction()`
  - `setBaseFunction()`
- Saves base functions in *discrete function list*

› Discrete Function Lists

- Save/ load list of discrete functions

```
typedef DiscreteFunctionList_xdr<DFType> ListType;
ListType list(discreteFunctionSpace, name);
list.push_back(function); // calls AttributeType()
list.push_back(function, attribute);
list.getFunc(i, destination);
list.getFuncByAttribute(attribute, destination);
```

› Discrete Function Lists

- Save/ load list of discrete functions
  - in memory

```
typedef DiscreteFunctionList_xdr<DFType> ListType;
ListType list(discreteFunctionSpace, name);
list.push_back(function); // calls AttributeType()
list.push_back(function, attribute);
list.getFunc(i, destination);
list.getFuncByAttribute(attribute, destination);
```

› Discrete Function Lists

- Save/ load list of discrete functions
  - in memory
  - on hard disk

```
typedef DiscreteFunctionList_xdr<DFType> ListType;
ListType list(discreteFunctionSpace, name);
list.push_back(function); // calls AttributeType()
list.push_back(function, attribute);
list.getFunc(i, destination);
list.getFuncByAttribute(attribute, destination);
```

› Discrete Function Lists

- Save/ load list of discrete functions
  - in memory
  - on hard disk
- Save attribute (of arbitrary serializable type) for each function

```
typedef DiscreteFunctionList_xdr<DFType> ListType;
ListType list(discreteFunctionSpace, name);
list.push_back(function); // calls AttributeType()
list.push_back(function, attribute);
list.getFunc(i, destination);
list.getFuncByAttribute(attribute, destination);
```

› Discrete Function Lists

- Save/ load list of discrete functions
  - in memory
  - on hard disk
- Save attribute (of arbitrary serializable type) for each function
- Access function by index or attribute

```
typedef DiscreteFunctionList_xdr<DFType> ListType;
ListType list(discreteFunctionSpace, name);
list.push_back(function); // calls AttributeType()
list.push_back(function, attribute);
list.getFunc(i, destination);
list.getFuncByAttribute(attribute, destination);
```

› Gramian Pipeline

- Efficient computation of Gramian-matrices with entries like $G_{ij} = \left(\mathcal{L}\left[\varphi_i\right], \varphi_j\right)$
- few grid iterations
- efficient memory management (optimization of hard disk access)

```
GramianPipeline pipe(DFList&)
OpHandle hId = pipe.getIdentityHandle();
OpHandle hL = pipe.registerDiscreteOperator(ℒ);
pipe.addGramMatrixComputation(hL,hId, G);
... // further computations added
pipe.run();
```

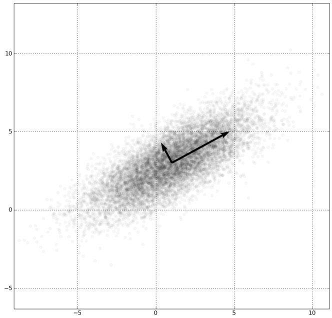
› Gramian Pipeline

- Efficient computation of Gramian-matrices with entries like $G_{ij} = \left(\mathcal{L}\left[\varphi_i\right], \varphi_j\right)$
- few grid iterations
- efficient memory management (optimization of hard disk access)

```
GramianPipeline pipe(DFList&)
OpHandle hId = pipe.getIdentityHandle();
OpHandle hL = pipe.registerDiscreteOperator(ℒ);
pipe.addGramMatrixComputation(hL,hId, G);
... // further computations added
pipe.run();
```

› Principal Component Analysis

- Principal Component Analysis (PCA) of discrete function lists
- Uses `LAPACK::dsyev`
- Usage: `pca(U, pcomps, ratio)`



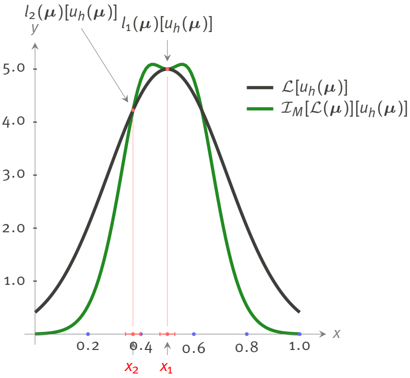
PCA of a Gaussian Scatter plot, Source: Wikipedia

› Prospects: Empirical Interpolation

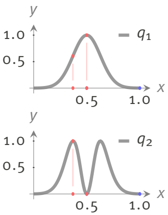
The Empirical Interpolation puts arbitrary discrete operators into “separable” form.

› Prospects: Empirical Interpolation

The Empirical Interpolation puts arbitrary discrete operators into "separable" form.

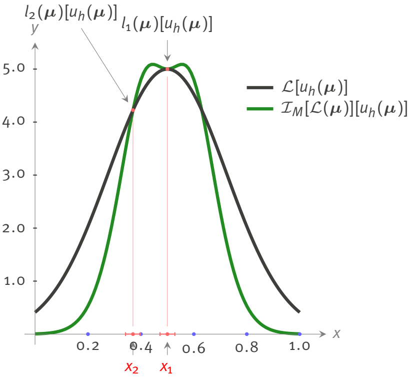


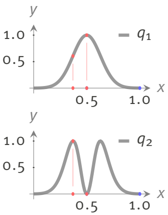
Base functions:



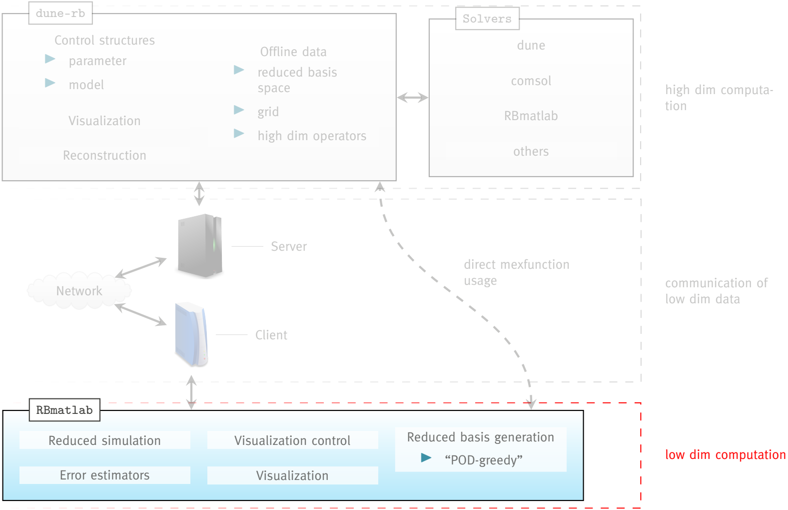
› Prospects: Empirical Interpolation

The Empirical Interpolation puts arbitrary discrete operators into “separable” form.





References: [Barrault et al., 2004], [Haasdonk et al., 2008]


dune-rb
Control structures
parameter
model
Visualization
Reconstruction
Offline data
reduced basis space
grid
high dim operators
Solvers
dune
comsol
RBmatlab
others
high dim computation
Server
Network
Client
direct mexfunction usage
communication of low dim data
RBmatlab
Reduced simulation
Visualization control
Reduced basis generation
“POD-greedy”
Error estimators
Visualization
low dim computation

› RBMatlab

```
model_data    = gen_model_data(model)
```
- generate grid

```
detailed_data = gen_detailed_data(model, model_data)
```
- compute high dim simulations
- generate reduced basis space

```
reduced_data  = gen_reduced_data(model, detailed_data)
```
- generate reduced matrices and vectors for reduced simulation and error estimation

Offline

```
sim_data      = rb_simulation(model, reduced_data)
```
- perfom reduced simulation
- evaluate error estimators

Online

› Further Capabilities of RBMatlab

- Different basis generation algorithms

› Further Capabilities of RBMatlab

- Different basis generation algorithms
  - adaptive/fixed training set search in parameter space

› Further Capabilities of RBMatlab

- Different basis generation algorithms
  - adaptive/fixed training set search in parameter space
  - multiple basis generation

› Further Capabilities of RBMatlab

- Different basis generation algorithms
  - adaptive/fixed training set search in parameter space
  - multiple basis generation
- Visualization

› Further Capabilities of RBMatlab

- Different basis generation algorithms
  - adaptive/fixed training set search in parameter space
  - multiple basis generation
- Visualization
- Post processing

› Further Capabilities of RBMatlab

- Different basis generation algorithms
  - adaptive/fixed training set search in parameter space
  - multiple basis generation
- Visualization
- Post processing
- Empirical interpolation

› Further Capabilities of RBMatlab

- Different basis generation algorithms
  - adaptive/fixed training set search in parameter space
  - multiple basis generation
- Visualization
- Post processing
- Empirical interpolation
- Finite volume discretization of parametrized PDEs

Thank you for your attention!