pyMOR School 2021

Reduced Basis Methods

Exercise Problems

Problem 1 (2D diffusion problem with output)

In this exercise we will solve a two-dimensional parametric diffusion problem, add an output functional and solve a time-dependent version of the problem.

(a) Solve on $\Omega := (0,1)^2$ the steady-state diffusion problem

$$\begin{aligned} \nabla \cdot (-\sigma(x,y;\mu)\nabla u(x,y;\mu)) &= f(x,y) && (x,y) \in \Omega, \\ u(x,y;\mu) &= 0 && (x,y) \in \partial\Omega,\ y = 0, \\ \sigma(x,y;\mu)\nabla u(x,y;\mu) \cdot n(x,y) &= 0 && (x,y) \in \partial\Omega,\ y \neq 0, \end{aligned}$$

where for $\mu \in \mathbb{R}^{>0}$

$$\sigma(x,y;\mu) := \begin{cases} \mu & x \in (0.45, 0.55), y \in (0.5, 1) \\ 1 & \text{otherwise,} \end{cases}$$

and

$$f(x,y) := \begin{cases} 100 & (x-0.25)^2 + (y-0.75)^2 < 0.01 \\ 0 & \text{otherwise.} \end{cases}$$

Use pyMOR's builtin discretization toolkit to discretize the problem. Solve the resulting discrete model for several parameter values and visualize the solution.

Hints:
- *All relevant classes and functions can be found in the* `pymor.basic` *module.*
- *Create a* `StationaryProblem` *to feed into* `discretize_stationary_cg`*.*
- *Use* `RectDomain` *to specify* Ω *and the Dirichlet/Neumann parts of* $\partial\Omega$*.*
- *Use* `ExpressionFunction` *to define* σ *and* f*. Start with a non-parametric version of the diffusivity* σ*.*
- *To refer to the first coordinate use* `x[0]` *in the definition of your* `ExpressionFunction`*. Use* `x[1]` *to refer to the second coordinate.*
- *Use* `{'bar': 1}` *to specify that the* `ExpressionFunction` σ *should depend on a single parameter* `bar`*, which is a vector of dimension 1.*

(b) Add an output functional s to the model, given by the integral

$$s(u) := \int_{\Omega_{out}} u(x,y)\,\mathrm{d}x\mathrm{d}y,$$

where $\Omega_{out} := \{x, y \in \mathbb{R} \,|\, (x-0.75)^2 + (y-0.75)^2 < 0.01\}$. Plot the parameter-to-output map.

Hints:
- *To specify an output functional, use the* **`outputs`** *parameter of* **`StationaryProblem`**.
- *Use the* **`output`** *method of* **`StationaryModel`** *to compute the output.*

(c) Solve the time-dependent version of the problem given by

$$
\begin{aligned}
\partial_t u(x,y,t;\mu)\nabla \cdot (-\sigma(x,y;\mu)\nabla u(x,y,t;\mu)) &= f(x,y) && (x,y) \in \Omega, t \in (0,10),\\
u(x,y,t;\mu) &= 0 && (x,y) \in \partial\Omega, y = 0, t \in (0,10),\\
\sigma(x,y;\mu)\nabla u(x,y,t;\mu) \cdot n(x,y) &= 0 && (x,y) \in \partial\Omega, y \neq 0, t \in (0,10),\\
u(x,y,0;\mu) &= 0 && (x,y) \in \Omega.
\end{aligned}
$$

Visualize the solution for several parameters and plot the time-to-output map.

Hint: *Construct an* **`InstationaryProblem`** *from your given* **`StationaryProblem`** *and feed it into* **`discretize_instationary_cg`**.

Problem 2 (1D diffusion problem)

Apart from 2D models, pyMOR's builtin discretization toolkit also supports 1D problems. Discretize the boundary value problem

$$
\begin{aligned}
(-\sigma(x;\mu) \cdot u'(x;\mu))' &= f(x) && x \in (-1,1),\\
u(-1;\mu) &= 0,\\
u(1;\mu) &= 0,
\end{aligned}
$$

where the source term $f(x)$ and the diffusivity $\sigma(\sigma;\mu)$ are given by

$$
f(x) = \begin{cases} 1 & x < 0 \\ 0 & x > 0 \end{cases} \qquad \text{and} \qquad \sigma(x;\mu) = \begin{cases} 1 & x < 0 \\ e^{\mu} & x > 0. \end{cases}
$$

Solve the resulting model for a few parameter values and visualize the solution.

Hint: *Use* **`LineDomain`** *to specify a one-dimensional domain.*

Problem 3 (Solving advection-diffusion equations)

So far we have only considered pure diffusion equations. In this exercise will add an advection term.

(a) Discretize and solve the following boundary value problem for different values of μ:

$$
\begin{aligned}
-\Delta u(x,y;\mu) + \mu \cdot \nabla \cdot \left(\begin{bmatrix} -y \\ x \end{bmatrix} \cdot u(x,y;\mu) \right) &= f(x,y) && (x,y) \in \Omega := (-1,1) \times (-1,1),\\
u(x,y;\mu) &= 0 && (x,y) \in \partial\Omega.
\end{aligned}
$$

The source term $f(x, y)$ is given as

$$f(x,y) = \begin{cases} 1 & (x-0.5)^2 + y^2 < 0.01 \\ 0 & \text{otherwise.} \end{cases}$$

Hint: Use the `advection` *parameter of* `StationaryProblem` *to specify the flux field* $[-y, x]^T$.

(b) Also solve the time-dependent version of this problem.

Problem 4 (Unstructured meshes and Robin boundary conditions)

pyMOR's discretization toolkit also supports unstructured triangle meshes created with Gmsh. These can be read using `pymor.discretizers.builtin.grids.gmsh.load_gmsh`. In this exercise, we will use pyMOR `domaindescriptions` that are automatically transformed into a Gmsh geometry definition for meshing.

(a) Solve the Poisson equation

$$\begin{aligned} -\Delta u(x) &= f(x) && x \in \Omega, \\ u(x) &= 0 && x \in \partial\Omega, \end{aligned}$$

where the domain Ω is the circular sector defined by

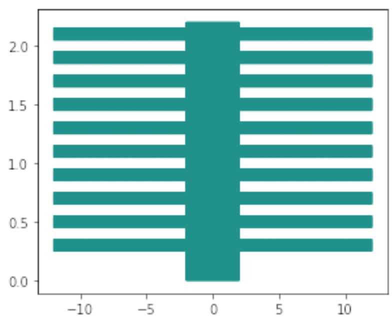
$$\Omega := \left\{ \begin{bmatrix} r \cdot \cos(\phi) \\ r \cdot \sin(\phi) \end{bmatrix} \;\middle|\; 0 \le r < 1,\ 0 \le \phi < 1.9 \cdot \pi \right\}.$$

Hint: Use `CircularSectorDomain` *to define* Ω.

(b) Solve

$$\begin{aligned} -\Delta u(x) &= f(x) && x \in \Omega, \\ \nabla u(x) \cdot n &= 1 && x \in \partial\Omega \cap \mathbb{R} \times \{0\}, \\ u(x) &= 0 && x \in \partial\Omega \setminus \mathbb{R} \times \{0\}, \end{aligned}$$

on the domain Ω given by the following heat-sink geometry:



Hint: *Use* `PolygonalDomain` *to define* Ω.

(c) Let's solve a physically somewhat more realistic model by imposing Robin boundary conditions on the fins, of the heat sink, i.e., solve

$$\begin{aligned} -\sigma \cdot \Delta u(x) &= f(x) && x \in \Omega, \\ \sigma \cdot \nabla u(x) \cdot n &= 80 && x \in \partial\Omega \cap \mathbb{R} \times \{0\}, \\ -\sigma \cdot \nabla u(x) \cdot n &= 1 \cdot (u(x) - 24) && x \in \partial\Omega \setminus \mathbb{R} \times \{0\}, \end{aligned}$$

with $\sigma = 10^3$ for the same heat-sink domain Ω as before.

Hint: *Pass the (constant) Robin data functions* 1 *and* 24 *as a tuple to* `StationaryProblem.__init__` *via the* `robin_data` *parameter.*

Problem 5 (Parameter Separation)

The models defined in problems 1 and 2 from exercise sheet 1 are parameter separable. Reformulate the definitions of the corresponding `StationaryProblems` such that the resulting discrete `Models` reflect that structure.

Hints:

- *Use* `LincombFunction` *to define the relevant data functions as a linear combination of non-parametric* `Functions` *with appropriate constants or* `ParameterFunctionals` *as coefficients.*
- *To specify a* $\vartheta_q(\mu)$ *of the form* $\vartheta_q(\mu) = \mu_i$*, use* `ProjectionParameterFunctional`*. For arbitrary expressions in* μ *use* `ExpressionParameterFunctional`*.*
- `discretze_stationary_cg` *and* `discretize_instationary_cg` *automatically detect* `LincombFunctions` *and assemble corresponding matrices* $\mathbb{A}^{(q)}$*.*
- *If the parameter separation was successful, subsequent calls to* `solve` *should not require the assembly of any finite-element matrix. Check pyMOR's log output to verify that this is the case.*

Problem 6 (Multiple Parameters)

We add an additional parameter to the model in problem 1 of exercise sheet 1 and solve for $\mu \in (\mathbb{R}^{>0})^2$ the PDE

$$\begin{aligned} \nabla \cdot (-\sigma(x,y;\mu)\nabla u(x,y;\mu)) &= f(x,y;\mu) && (x,y) \in \Omega, \\ u(x,y;\mu) &= 0 && (x,y) \in \partial\Omega,\ y = 0, \\ \sigma(x,y;\mu)\nabla u(x,y;\mu) \cdot n(x,y) &= 0 && (x,y) \in \partial\Omega,\ y \neq 0, \end{aligned}$$

where

$$\sigma(x,y;\mu) := \begin{cases} \mu_1 & x \in (0.45, 0.55), y \in (0.5, 1) \\ 1 & \text{otherwise,} \end{cases}$$

and

$$f(x,y;\mu) := \begin{cases} 100 & (x-0.25)^2 + (y-0.75)^2 < 0.01 \\ \mu_2 & \text{otherwise.} \end{cases}$$

Extend your problem definition to include the additional parameter. Ensure parameter separation.

Problem 7 (Orthogonal Projection onto Reduced Space)

In this exercise we will construct a reduced space from some random snapshot data and compute the best-approximation error w.r.t. this space. In the following you can choose any of the parametric discrete models you have created in the first two exercise sheets.

(a) Build a `ParameterSpace` for your problem by either specifying `parameter_ranges` when constructing the `StationaryProblem` or by directly constructing a `ParameterSpace`. Use the `sample_randomly` method to create a number of `Mu` instances holding random parameter values from this space.

(b) Collect the corresponding solution snapshots in a `VectorArray` `U`. We will use these snapshot vectors as a basis for our reduced space V_N.

(c) The best-approximation $u_N^*(\mu) \in V_N$ of $u_h(\mu)$ in V_N satisfying

$$\|u_h(\mu) - u_N^*(\mu)\| = \inf_{v_N \in V_N} \|u_h(\mu) - v_N(\mu)\|$$

is given by the orthogonal projection of $u_h(\mu)$ onto V_N. Hence, $u_N^*(\mu)$ satisfies:

$$(u_N^*(\mu), v_N) = (u_h(\mu), v_N) \qquad \forall v_N \in V_N. \tag{1}$$

Representing $u_N^*(\mu)$ as $u_N^*(\mu) = \sum_{i=1}^{N} \underline{u}_{N,i}^*(\mu) u_i$ where u_i denote the vectors in `U`, find a linear system corresponding to (1) which determines $\underline{u}_N^*(\mu)$ for given μ.

(d) Assemble the linear system using pyMOR and determine the solution $\underline{u}_N^*(\mu)$. In (1) use both the Euclidean and the H^1-inner product to compute a best approximation w.r.t. these norms. Reconstruct $u_N^*(\mu)$ from $\underline{u}_N^*(\mu)$. Visualize $u_N^*(\mu)$ alongside $u_h(\mu)$.

(e) Compute the maximum/average error $\|u_h(\mu) - u_N^*(\mu)\|$ in the Euclidean and H^1-norm for a new validation set of random parameters μ. Verify that the error is zero for the μ used to build `U`.

Hints:

- *To create an empty `VectorArray` of suitable type, use the `empty` method of the `solution_space` of your model. Use the `append` method of the array to append the solution snapshots to it.*
- *To assemble* (1) *use the `inner` and `gramian` methods of `VectorArray`. Use `lincomb` to reconstruct $u_N^*(\mu)$. Norms are computed using the `norm` method. `discrete_stationary_cg` automatically assembles several inner product `Operators`, which are available as attributes of the resulting discrete `Model`.*

Problem 8 (Manual Reduced Basis Projection)

In the last problem we have constructed reduced spaces for parametrized problems using random snapshot data, and we have computed the best-approximation error w.r.t. to these spaces. We will now compute the Galerkin projection into these spaces and compare it with the best-approximation.

(a) Using the basis `VectorArray U`, compute the reduced system matrix $\mathbb{A}^{(N)}(\mu)$ and right-hand side vector $\mathbb{F}^{(N)}$. Solve the resulting linear equation system to determine $\underline{u}_N(\mu)$. Reconstruct $u_N(\mu)$.

(b) Compute the MOR error $\|u_h(\mu) - u_N(\mu)\|_1$ and compare it with the best-approximation error $\|u_h(\mu) - u_N^*(\mu)\|_1$. Compute the maximum/average errors over a validation set of random parameters. Plot these errors in dependence on the basis size. Can you avoid re-assembling the corresponding linear systems for smaller basis sizes?

(c) Measure the times required for assembling $\mathbb{A}^{(N)}(\mu)$, solving for $\underline{u}_N(\mu)$ and reconstructing $u_N(\mu)$. Plot theses timings in dependence on the basis size.

(d) Exploit the parameter separability and pre-assemble $\mathbb{A}^{(N,q)}$. Also measure the time needed to assemble $\mathbb{A}^{(N)}$ using these matrices. Verify that you obtain the same result.

Hints:

- *A `StationaryModel` stores the bilinear form a in the `operator` attribute, ℓ is given by the `rhs` attribute.*
- *To interpret the `Operator` `fom.operator` as a bilinear form and evaluate it, use the `apply2` method.*
- *ℓ is encoded as a linear `Operator` mapping real numbers x to the coefficient vector $x \cdot \mathbb{F}$. To obtain a `VectorArray` containing $\mathbb{F}$ use the `as_vector` method.*
- *Parameter separation in pyMOR `Models` is encoded using `LincombOperators`. These hold the summands $\mathbb{A}^{(q)}$ in the `operators` attribute. The corresponding `ParameterFunctionals` are stored in the `coefficients` attribute.*

Problem 9 (Automatic Operator Projection)

In Problem 2 from exercise sheet 4 we have manually computed and solved the reduced system

$\mathbb{A}^{(N)}(\mu) \cdot \underline{u}_N(\mu) = \mathbb{F}^{(N)}$. In pyMOR, the (Petrov)-Galerkin projection of `Operators` is handled by the `project` method.

(a) Update your code to use `project` and construct a reduced `StationaryModel` from the resulting reduced `Operators`. What happens if you project a `LincombOperator`? What happens if your parametric `Operator` is not decomposed as a `LincombOperator`?

(b) Try to understand how `pymor.algorithms.projection.ProjectRules` works. Use the `insert_rule` method to inject a rule which uses `ProjectedOperator` for every `Operator` with `name 'ignore'`. Use the `with_` method to tag one of the `operators` of a given `LincombOperator` with this name, and check if the new rule behaves as expected.

Problem 10 (Error-vs-Parameter Plot)

Choose a full-order model with one- or two-dimensional parameter domain, build a reduced order model from random parameter samples, and plot the model order reduction error over the parameter domain. Use a logarithmic scale for the error.

Problem 11 (Reductors)

Instead of manually projecting each `Operator` of a `Model` and constructing a reduced `Model` from the projected `Operators`, we can use a `Reductor` to facilitate the process.

(a) Modify your existing code to use the `reduce` method of `StationaryRBReductor` to build the ROM. Use the `reconstruct` method to reconstruct finite-element vectors from the reduced solutions.

(b) Use `CoerciveRBReductor` instead of `StationaryRBReductor` to additionally assemble an error estimator for the ROM. Plot the actual and estimated errors over the parameter domain.

(c) Use `estimator.reduce(N)` to quickly obtain a ROM for V_N when the ROM for $V_{N'}$, $N < N'$, has already been computed. Plot the maximum MOR error in dependence on the basis size.

Problem 12 (Greedy algortihm with pyMOR)

Greedy algorithms for constructing reduced approximation spaces can be found in pyMOR's `algorithms.greedy` and `algorithms.adaptivegreedy` modules.

(a) Use `rb_greedy` to build a reduced basis with the estimated MOR error as a surrogate for the best-approximation error. Plot the maximum MOR error on the training set and on a validation set in dependence on the basis size. Compare the result with reduced spaces obtained from random parameter selection. Also plot the MOR error over the parameter domain.

(b) Set `use_error_estimator` to `False` to study the effect of the error estimator.

(c) Specify a `WorkerPool` to parallelize the greedy search.

(d) Try `rb_adaptive_greedy` as a replacement for `rb_greedy`.

(e) Write a `strong_greedy` method which produces a strong greedy sequence for a given `VectorArray` of snapshot vectors to approximate. Compare the quality of the resulting ROM with the weak greedy ROM.