Tutorial 1

Thermal block problem

Keywords: Certified reduced basis method, scalar problem

1 Introduction

In this Tutorial, we consider steady heat conduction in a two-dimensional domain Ω, which is shown in Figure 1 and summarize how to use RBniCS for a certified reduced basis approximation of the problem. Let Ω_0 be a disk centered at the origin of radius $r_0 = 0.5$ and define $\Omega_1 = \Omega / \overline{\Omega_0}$. The conductivity k is assumed to be constant on Ω_0 and Ω_1, i.e.

$$k|_{\Omega_0} = k_0 \quad \text{and} \quad k|_{\Omega_1} = 1.$$

For this problem, we consider $P = 2$ parameters. The first one is related to the conductivity in Ω_0, i.e. $\mu_1 \equiv k_0$. The second parameter μ_2 takes into account the constant heat flux over Γ_{base}. The parameter vector $\boldsymbol{\mu}$ is thus given by

$$\boldsymbol{\mu} \equiv (\mu_1, \mu_2)$$

on the parameter domain $\mathbb{P} = [0.1, 10] \times [-1, 1]$.

The (scalar) field variable $u(\boldsymbol{\mu})$ is the temperature, which satisfies Laplace equation in Ω. The following boundary conditions are employed: zero Neumann (zero flux, or insulated) conditions on Γ_{side}; zero Dirichlet (temperature) conditions on Γ_{top}; and inhomogeneous parametrized Neumann conditions (prescribed heat flux μ_2) on Γ_{base}.

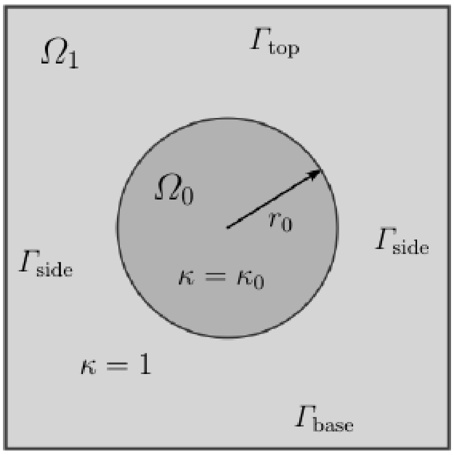

Figure 1: Geometrical set-up.

2 Parametrized formulation

The strong formulation of the parametrized problem is given by: for a given parameter $\boldsymbol{\mu} \in \mathbb{P}$, find $u(\boldsymbol{\mu})$ such that

$$\begin{cases} \operatorname{div}(k(\mu_1)\nabla u(\boldsymbol{\mu})) = 0 & \text{in } \Omega, \qquad (1)\\ u(\boldsymbol{\mu}) = 0 & \text{on } \Gamma_{\text{top}}, \qquad (2)\\ k(\mu_1)\nabla u(\boldsymbol{\mu}) \cdot \mathbf{n} = 0 & \text{on } \Gamma_{\text{side}}, \qquad (3)\\ k(\mu_1)\nabla u(\boldsymbol{\mu}) \cdot \mathbf{n} = \mu_2 & \text{on } \Gamma_{\text{base}}. \qquad (4)\end{cases}$$

The corresponding weak formulation reads: for a given parameter $\boldsymbol{\mu} \in \mathbb{P}$, find $u(\boldsymbol{\mu}) \in \mathbb{V}$ such that

$$a\left(u(\boldsymbol{\mu}), v; \boldsymbol{\mu}\right) = f(v; \boldsymbol{\mu}) \quad \forall v \in \mathbb{V},$$

with

$$a(w, v; \boldsymbol{\mu}) = \int_\Omega k(\mu_1)\nabla w \cdot \nabla v \; d\boldsymbol{x} \quad \text{and} \quad f(v; \boldsymbol{\mu}) = \mu_2 \int_{\Gamma_{\text{base}}} v \; ds, \tag{5}$$

for all $v, w \in \mathbb{V} = \{v \in H^1(\Omega) : v|_{\Gamma_{\text{top}}} = 0\}$.

A compliant output of interest $s(\boldsymbol{\mu})$, i.e.

$$s(\boldsymbol{\mu}) = \mu_2 \int_{\Gamma_{\text{base}}} u(\boldsymbol{\mu}),$$

is computed for each $\boldsymbol{\mu}$.

3 Affine decomposition

For this problem the affine decomposition is straightforward:

$$a(w, v; \boldsymbol{\mu}) = \underbrace{\mu_1}_{\Theta_1^a(\boldsymbol{\mu})} \underbrace{\int_{\Omega_0} \nabla w \cdot \nabla v \; d\boldsymbol{x}}_{a_1(w,v)} + \underbrace{1}_{\Theta_2^a(\boldsymbol{\mu})} \underbrace{\int_{\Omega_1} \nabla w \cdot \nabla v \; d\boldsymbol{x}}_{a_2(w,v)}$$

$$f(v; \boldsymbol{\mu}) = \underbrace{\mu_2}_{\Theta_1^f(\boldsymbol{\mu})} \underbrace{\int_{\Gamma_{\text{base}}} v \; ds,}_{f_1(v)}$$

4 Implementation in RBniCS

The implementation of this Tutorial can be found in solve_tblock.py.

4.1 Main

We are going to use both FEniCS and RBniCS:

```
from dolfin import *
from RBniCS import *
```

Read in the mesh of the domain:

```
mesh = Mesh("data/tblock.xml")
subd = MeshFunction("size_t", mesh, "data/tblock_physical_region.xml")
bound = MeshFunction("size_t", mesh, "data/tblock_facet_region.xml")
```

Create a finite element space (say, P1 Lagrange FE) using FEniCS:

```
V = FunctionSpace(mesh, "Lagrange", 1)
```

Allocate an object of the `Tblock` RBniCS class (see next subsection):

```
tb = Tblock(V, subd, bound)
```

Define $\mathbb{P}$, a random subset $\Xi_{\text{train}} \subset \mathbb{P}$ and the maximum number of basis functions $N_{\max}$:

```
mu_range = [(0.1, 10.0), (-1.0, 1.0)]
tb.setmu_range(mu_range)
tb.setxi_train(100)
tb.setNmax(4)
```

Perform the offline stage and save the reduced order data structures:

```
tb.offline()
```

Perform an online solve and plot the obtained online solution:

```
online_mu = (8.,-1.0)
tb.setmu(online_mu)
tb.online_solve()
```

Finally, perform an error analysis of the accuracy of the ROM w.r.t. the FE solution in FEniCS:

```
tb.setxi_test(500)
tb.error_analysis()
```

4.2 The `Tblock` class

In this Tutorial we are interested in the reduced order modelling for a coercive elliptic problem with compliant output using the *certified reduced basis method*. To this end, the `Tblock` inherits from the RBniCS class `EllipticCoerciveRBBase`.

```
class Tblock(EllipticCoerciveRBBase):
```

In the constructor of the `Tblock` class we initialize the measures `dx` and `ds` for integral computations and the norm of $\mathbb{V}$. Thanks to homogeneous boundary conditions on Γ_{top} we choose the latter to be the $H^1(\Omega)$ seminorm.

```
    def __init__(self, V, subd, bound):
        bc = DirichletBC(V, 0.0, bound, 3)
        # Call the standard initialization
        EllipticCoerciveRBBase.__init__(self, V, [bc])
        # ... and also store FEniCS data structures for assembly
        self.dx = Measure("dx")[subd]
        self.ds = Measure("ds")[bound]
        # Use the H^1 seminorm on V as norm, instead of the H^1 norm
        u = self.u
        v = self.v
        dx = self.dx
        scalar = inner(grad(u),grad(v))*dx
        self.S = assemble(scalar)
        [bc.apply(self.S) for bc in self.bc_list] # make sure to apply BCs
            ↪ to the inner product matrix
```

Define $\Theta^a_1(\boldsymbol{\mu})$ and $\Theta^a_2(\boldsymbol{\mu})$ in the method `compute_theta_a` ...

```
    def compute_theta_a(self):
        mu1 = self.mu[0]
        mu2 = self.mu[1]
        theta_a0 = mu1
        theta_a1 = 1.
        return (theta_a0, theta_a1)
```

.. and $\Theta^f_1(\boldsymbol{\mu})$ in `compute_theta_f`[1]:

```
    def compute_theta_f(self):
        return (self.mu[1],)
```

[1]Notice the comma after the square bracket.

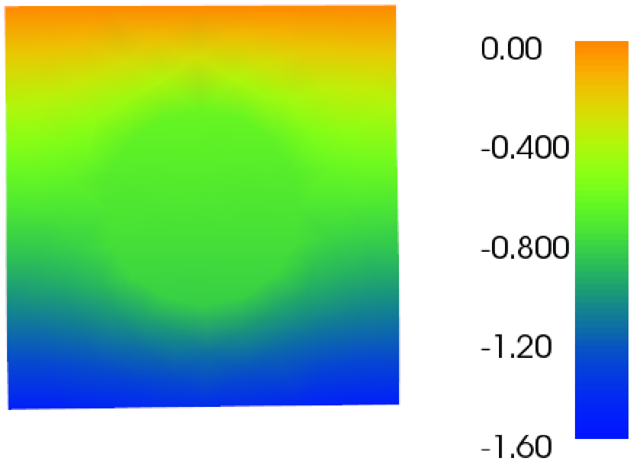

Figure 2: Reduced order solution for $\boldsymbol{\mu} = (8, -1)$

In a similar way, assemble[2] $a_1(w, v)$ and $a_2(w, v)$ in `assemble_truth_a` ...

```
def assemble_truth_a(self):
    u = self.u
    v = self.v
    dx = self.dx
    # Assemble A0
    a0 = inner(grad(u),grad(v))*dx(1) +1e-15*inner(u,v)*dx
    A0 = assemble(a0)
    # Assemble A1
    a1 = inner(grad(u),grad(v))*dx(2) +1e-15*inner(u,v)*dx
    A1 = assemble(a1)
    # Return
    return (A0, A1)
```

... and $f_1(v)$ in `assemble_truth_f`:

```
def assemble_truth_f(self):
    v = self.v
    dx = self.dx
    ds = self.ds
    # Assemble F0
    f0 = v*ds(1) + 1e-15*v*dx
    F0 = assemble(f0)
    # Return
    return (F0,)
```

Finally, implement the `get_alpha_lb` method to compute the lower bound of the coercivity constant. In this Tutorial it can be computed as the minimum between μ_1 and 1:

```
def get_alpha_lb(self):
    return min(self.compute_theta_a())
```

4.3 Run it!

Make sure that both FEniCS and RBniCS are in your `PYTHONPATH`.

```
source $FENICS_INATALL_DIRECTORY/share/fenics/fenics.conf
export PYTHONPATH="$RBNICS_SOURCE_DIRECTORY:$PYTHONPATH"
```

[2]The addition of terms as `1e-15*inner(u,v)*dx` is required to avoid the following PETSc error:
**** Error: Unable to set given (local) rows to identity matrix.*
**** Reason: some diagonal elements not preallocated (try assembler option keep_diagonal).*
**** Where: This error was encountered inside PETScMatrix.cpp.*
As an alternative, the user could also set the option *keep_diagonal* as the error message suggests.

Run the code in solve_tblock.py as follows:

```
python solve_tblock.py
```

Data structures related to the reduced order model will be saved in several subfolders, namely `basis`, `dual`, `pp`, `red_matr` and `snapshots`. The online solution for $\boldsymbol{\mu} = (8, -1)$ is automatically plotted and should look similar to the one in Figure 2.

5 A look under the hood of RBniCS

The core of the RBniCS implementation can be found in the `RBniCS` directory. The base class of all RBniCS components is the class `ParametrizedProblem` defined in `parametrized_problem.py`. This class encapsulate an offline/online decomposition of parametrized problems, such as e.g. the methods required to generate a training set, the parameter domain $\mathbb{P}$, the current value of $\boldsymbol{\mu}$, etc. This class is not meant to be used by the final user.

Its child `EllipticCoerciveBase`, defined in `elliptic_coercive_base.py`, provides additional interfaces for projection based reduced order models of elliptic coervice problems, such as `compute_theta_a` and `assemble_truth_a`, to be properly overridden by the final user. Also this class is not meant to be used directly by the final user.

In this Tutorial we have seen how to employ its child `EllipticCoerciveRBBase`, defined in `elliptic_coercive_rb_base.py`, to apply the reduced basis method to (compliant) elliptic coervice problems. This class *is* meant to be used in practical problems by the user, such as this Tutorial. The core of the reduced basis method is implemented in this class, for what concerns both offline and online stages.

Documentation of methods in RBniCS is provided by comments in the doxygen format, which can be compiled running `doxygen` in the `doc` directory.