***Tutorial 2***

# Elastic block problem

**Keywords:** POD–Galerkin method, vector problem

## 1 Introduction

In this Tutorial we consider a linear elasticity example in the two-dimensional domain shown in Figure 1. In this case we will use a Proper Orthogonal Decomposition (POD)–Galerkin method instead of the RB method explored in the previous Tutorial.

## 2 Parametrized formulation

The bilinear form associated to the left-hand-side of the problem is given by:

$$a(\mathbf{w}, \mathbf{v}; \boldsymbol{\mu}) = \sum_{p=1}^{8} \mu_p \int_{\Omega_p} \frac{\partial v_i}{\partial x_j} C_{ijkl} \frac{\partial w_k}{\partial x_l} \, d\boldsymbol{x} + 1 \int_{\Omega_9} \frac{\partial v_i}{\partial x_j} C_{ijkl} \frac{\partial w_k}{\partial x_l} \, d\boldsymbol{x},$$

where $\mu_p$ is the ratio between the Young modulus of the $\Omega_p$ and $\Omega_9$ subdomains, respectively, and

$$\mu_p \in [1, 100] \qquad \text{for } p = 1, \dots, 8.$$

We consider an isotropic material, so the elasticity tensor is given by

$$C_{ijkl} = \lambda_1 \delta_{ij} \delta_{kl} + \lambda_2 (\delta_{ik} \delta_{jl} + \delta_{il} \delta_{jk}),$$

where

$$\lambda_1 = \frac{\nu}{(1+\nu)(1-2\nu)},$$

$$\lambda_2 = \frac{1}{2(1+\nu)},$$

are the Lamè constants for plane strain and $\nu = 0.30$.

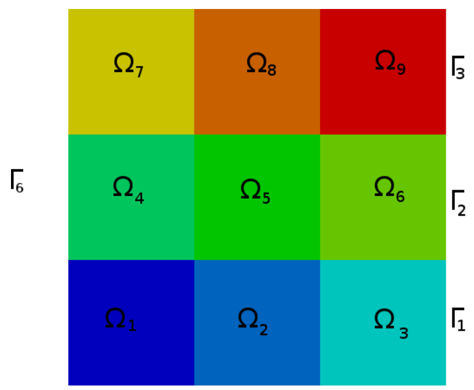


Figure 1: Subdomain division.

Homogeneous Dirichlet boundaries conditions are imposed on $\Gamma_6$:

$$\mathbf{w} = 0 \quad \text{on } \Gamma_6.$$

Inhomogeneous Neumann boundary conditions, corresponding to orthogonal loads, are considered on $\Gamma_1 \cup \Gamma_2 \cup \Gamma_3$ and result in the following right-hand side:

$$f(\mathbf{v}; \boldsymbol{\mu}) = \mu_9 \int_{\Gamma_1} v_1 \; ds + \mu_{10} \int_{\Gamma_2} v_1 \; ds + \mu_{11} \int_{\Gamma_3} v_1 \; ds.$$

where

$$\mu_p \in [-1, 1] \qquad \text{for } p = 9, \dots, 11.$$

Homogeneous Neumann boundary conditions are applied on the remaining part of the boundary. No output of interest $s(\mu)$ is computed in this case.

# 3 Affine decomposition

For this problem the affine decomposition is straightforward:

$$a(\mathbf{w}, \mathbf{v}; \boldsymbol{\mu}) = \sum_{p=1}^{8} \underbrace{\mu_p}_{\Theta_p^a(\boldsymbol{\mu})} \underbrace{\int_{\Omega_p} \frac{\partial v_i}{\partial x_j} C_{ijkl} \frac{\partial w_k}{\partial x_l} \; d\boldsymbol{x}}_{a_p(\mathbf{w},\mathbf{v})} \; + \; \underbrace{1}_{\Theta_9^a(\boldsymbol{\mu})} \underbrace{\int_{\Omega_9} \frac{\partial v_i}{\partial x_j} C_{ijkl} \frac{\partial w_k}{\partial x_l} \; d\boldsymbol{x}}_{a_9(\mathbf{w},\mathbf{v})},$$

$$f(\mathbf{v}; \boldsymbol{\mu}) = \underbrace{\mu_9}_{\Theta_1^f(\boldsymbol{\mu})} \underbrace{\int_{\Gamma_1} v_1 \; ds}_{f_1(\mathbf{v})} + \underbrace{\mu_{10}}_{\Theta_2^f(\boldsymbol{\mu})} \underbrace{\int_{\Gamma_2} v_1 \; ds}_{f_2(\mathbf{v})} + \underbrace{\mu_{11}}_{\Theta_3^f(\boldsymbol{\mu})} \underbrace{\int_{\Gamma_3} v_1 \; ds}_{f_3(\mathbf{v})}.$$

# 4 Implementation in RBniCS

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

## 4.1 The Eblock class

As in the previous Tutorial, in this example we are solving a coercive elliptic problem. In contrast to Tutorial 1, however, in this case we are interested in a POD–Galerkin method. To this end, the `Eblock` class is defined as follows:

```
class Eblock(EllipticCoercivePODBase):
```

In particular, the only modification you need to perform to change a RBniCS script from a reduced basis method to a POD–Galerkin one is to change `EllipticCoerciveRBBase` (reduced basis base class) to `EllipticCoercivePODBase` (POD–Galerkin base class)[1].

The constructor of an instance of the `Eblock` class can be defined similarly to the first Tutorial. In particular, measures for integral computations, traction term and Lamè constants are defined in this constructor.

```
    def __init__(self, V, subd, bound):
        bc = DirichletBC(V, (0.0, 0.0), bound, 6)
        # Call the standard initialization
        EllipticCoercivePODBase.__init__(self, V, [bc])
        # ... and also store FEniCS data structures for assembly
        self.dx = Measure("dx")[subd]
        self.ds = Measure("ds")[bound]
        # ...
        self.f = Constant((1.0, 0.0))
```

[1] And in a similar way the other way around. However, reduced basis classes require a method `get_alpha_lb` that is not used by POD–Galerkin classes.

```
        self.E  = 1.0
        self.nu = 0.3
        self.lambda_1 = self.E*self.nu / ((1.0 + self.nu)*(1.0 - 2.0*self.
            ↪ nu))
        self.lambda_2 = self.E / (2.0*(1.0 + self.nu))
```

The affine expansion of the bilinear form $a(\mathbf{w}, \mathbf{v})$ can be easily assembled thanks to the capability of the FEniCS library:

```
    def compute_theta_a(self):
        mu = self.mu
        mu1 = mu[0]
        ...
        mu8 = mu[7]
        theta_a0 = mu1
        ...
        theta_a7 = mu8
        theta_a8 = 1.
        return (theta_a0, ... ,theta_a8)

    def assemble_truth_a(self):
        u = self.u
        v = self.v
        dx = self.dx
        # Define
        a0 = self.elasticity(u,v)*dx(1) +1e-15*inner(u,v)*dx
        ...
        a8 = self.elasticity(u,v)*dx(9) +1e-15*inner(u,v)*dx
        # Assemble
        A0 = assemble(a0)
        ...
        A8 = assemble(a8)
        # Return
        return (A0, ..., A8)
```

and in a similary way for the right-hand side $f(\mathbf{v})$.

The code solve_elast_pod.py is executed as described in Tutorial 1.

# 5 A look under the hood of RBniCS

The class `EllipticCoercivePODBase`, defined in `elliptic_coercive_pod_base.py`, is employed in this Tutorial, and provides the implementation of POD–Galerkin ROMs of elliptic coervice problems. Its interface is purposely similar to the `EllipticCoerciveRBBase` (for reduced basis method) that has been discussed in Tutorial 1. The core of the POD–Galerkin ROM is implemented in this class, for what concerns both offline and online stages.