POLITECNICO DI TORINO

I Facoltá di Ingegneria

Corso di Laurea Magistrale in Ingegneria Civile

Tesi di Laurea Magistrale

Proper generalized decomposition applied to solid mechanics problems

Author
Giuseppe D'Ettorre

Supervisors:
Prof. dr. ir. Rosario Ceravolo
Politecnico di Torino,
Prof. dr. ir. Geert Degrande
KU Leuven
Assistant:
Ir. Pieter Reumers
KU Leuven

2018

KU LEUVEN

FACULTEIT INGENIEURSWETENSCHAPPEN

Proper generalized decomposition method applied to solid mechanics problems

Giuseppe D'Ettorre

Thesis submitted for the degree of Master of Science in Civil Engineering, option Structural Engineering

Thesis supervisor:
Prof. dr. ir. Geert Degrande

Assessors:
Prof. dr. ir. Karl Meerbergen
Ir. Pieter Reumers

Mentor:
Ir. Pieter Reumers

Academic year 2017 – 2018

Preface

First of all, I would like to express my sincere gratefulness to my promotor prof. Geert Degrande for his kind support and excellent supervision. I would like to thank Ir. Pieter Reumers for his guidance and support and prof. Rosario Ceravolo for his disponibility. They inspired my interest on this topic and provided lots of instruction. I would also like to thank prof. Karl Meerbergen and the Structural Mechanics Section of KU Leuven. I spent almost a year in Leuven and I am lucky to have spent this year in this historical university.
Finally, I owe my thanks to my beloved family who supported me during this unforgettable year.

Giuseppe D'Ettorre

Contents

List of Figures **5**

Listings **7**

1 Introduction **11**
1.1 Proper generalized decomposition method 11
1.1.1 PGD algorithm 13
1.2 PGD applications 14
1.2.1 Palpation of the liver 14
1.2.2 Water agitation: Mataró harbor (Spain) 16
1.2.3 Parametric model of a thermal process 18
1.2.4 Structural mechanics applications 19
1.3 Goals of this MSc thesis 20
1.4 Organization of the text 20

2 The finite element method **23**
2.1 Introduction 23
2.2 General solution of solid mechanics problems 24
2.2.1 Problem geometry 24
2.2.2 Weighted residual method: Galerkin approach 25
2.3 Mathematical formulation of solid mechanics problems 26
2.3.1 Weighted residual method applied to solid mechanics problems 28

3 Boundary conditions as extra problem dimension **31**
3.1 Introduction 31
3.2 The load position as extra parameter of the problem 31
3.2.1 Computation of $\mathbf{R}(\mathbf{x})$ assuming $S(s)$ is known 33
3.2.2 Computation of $S(s)$ assuming $\mathbf{R}(\mathbf{x})$ is known 36
3.3 Numerical integration: Gaussian quadrature rules 37
3.4 Geometry 38
3.5 Results 39
3.5.1 Off-line phase results 39
3.5.2 On-line phase results 41

4 Material parameter as extra problem dimension **45**

4.1 Introduction 45
4.2 The Poisson's ratio as extra parameter of the problem 45
4.2.1 Computation of $\mathbf{R}(\mathbf{x})$ assuming $P(\nu)$ is known 46
4.2.2 Computation of $P(\nu)$ assuming $\mathbf{R}(\mathbf{x})$ is known 49
4.3 Implentation in Matlab: Gaussian quadrature rules 51
4.3.1 First approach of implementation 51
4.3.2 Second approach of implementation 53
4.4 Geometry 55
4.5 Results 55
4.5.1 Off-line phase results 56
4.5.2 On-line phase results 59

5 Conclusions and future works **61**
5.1 Conclusive remarks 61
5.2 Recommendations for further research 62

Bibliography **63**

A PGD code for load position **65**

B PGD code for Poisson's ratio **75**
B.1 First approach of implementation 75
B.2 Second approach of implementation 86

List of Figures

1.1 Finite element model for the human liver [12]. 15
1.2 (a) Interactive palpation of a liver, (b) Use of haptic device in the on-line phase [12]. 15
1.3 Mataró harbor: wave amplification for the particular case $\omega = 0.61 rad/s$ and $\theta = 194.5$. It shows the spatial FEM (left column), PGD with 2000 nonlinear iterations (middle column) and PGD with 400 nonlinear iterations (right column) [11]. 16
1.4 Barcelona harbor: problem statement. Spatial domain with values of the absorbing coefficient α on the boundary and and contour of the bathymetry [11]. 17
1.5 Barcelona harbor: wave-height in different harbor areas. In particular: short waves with $(0.61, 1.08\pi)$ (left), mid waves with $(0.55, 1.24\pi)$ (middle) and long waves with $(0.42, 1.08\pi)$ (right). The spatial FEM solution (top) and the PG PGD interpolated solution (bottom) with 1500 PGD-projected terms (8000 solves) are shown [11]. 18
1.6 Thermal process consisting of two heating devices located on the die walls [8]. 18
1.7 Online phase: the user is able to choose the temperatures θ_1 and θ_2 and get in real time the temperature in any point of the system [8]. 19
1.8 (a) Web implementation of the algorithm. It represents a linear elastic beam, (b) Implementation on a *html* file with javascript. It represents the response of a cantilever beam [5]. 20

2.1 (a)Solid, (b) FE mesh, (c) Extraction of one particular element [13]. . . 23
2.2 Displacement and force (stress, traction) boundary conditions for the plane stress problem [13]. 25

3.1 (a) Linear triangular element, (b) Shape function N_1^e of a triangular element. 38
3.2 FE mesh of the beam. 39
3.3 Spatial modes: (a) Spatial mode $\mathbf{F}_1$, (b) Spatial mode $\mathbf{F}_2$, (c) Spatial mode $\mathbf{F}_3$, (d) Spatial mode $\mathbf{F}_{10}$. 40

3.4 Load modes: (a) Load mode $\mathbf{G}_1$, (b) Load mode $\mathbf{G}_2$, (c) Load mode $\mathbf{G}_3$, (d) Load mode $\mathbf{G}_{10}$. 40
3.5 Relative errors for varying iterations. 41
3.6 Example of two loaded points and evalutaion of the displacement in the node 2. 41
3.7 Deformed shapes due to the unit load. Note that the scaling is equal for both deformations. 42
3.8 Beam loaded in node 3. Displacement evaluated in node 3 and 71. . . . 42

4.1 Function $\underline{\mathrm{R}}^{\mathrm{T}}\mathbf{K}(\nu)\underline{\mathrm{R}}$ 52
4.2 Function $\underline{\mathrm{R}}^{\mathrm{T}}\mathbf{K}(\nu)\underline{\mathrm{F}}_1$ 52
4.3 Poisson's modes: (a) Poisson's mode $\mathbf{G}_1$, (b) Poisson's mode $\mathbf{G}_2$, (c) Poisson's mode $\mathbf{G}_3$, (d) Poisson's mode $\mathbf{G}_8$. 56
4.4 Relative errors using the first and the second approach of implementation. 57
4.5 CPU time per iterations using the first and the second approach. 58

Listings

A.1 Off-line phase code for the load position problem. 65
A.2 On-line phase code for the load position problem. 68
A.3 Matlab code for the enrichment function. 70
A.4 Matlab code for the function M2. 72
B.1 Off-line phase code for the Poisson's ratio problem (first approach). . 75
B.2 On-line phase code for the Poisson's ratio problem (first approach). . 78
B.3 Element stiffness matrix used to compute the LHS of equation (4.16). 79
B.4 Element stiffness matrix used to compute the RHS of equation (4.16). 81
B.5 LHS of equation (4.23). 83
B.6 RHS of equation (4.23). 85
B.7 Off-line phase code for the Poisson's ratio problem (second approach). 86
B.8 On-line phase code for the Poisson's ratio problem (second approach). 89
B.9 Matlab code for the enrichment function. 90
B.10 Matlab code for the function M1. 92
B.11 Matlab code for the function matrixR1. 93
B.12 Matlab code for the function sourceR1. 94
B.13 Matlab code for the function matrixP1. 95
B.14 Matlab code for the function sourceP1. 96

Abstract

Despite the progressive improvements in terms of simulation capabilities and techniques as well, some engineering problems remain very expensive from a computational point of view. Many techniques were developed in the last decades in order to circumvent the computational costs issue, such as the proper orthogonal decomposition (POD). In this thesis another technique called proper generalized decomposition method (PGD method) is analysed as an alternative and incredible reduced order modelling (ROM) technique. The PGD method is based on the assumption of separability of the solution and it has demonstrated its capability of solving multidimensional problems. The PGD method leads to two main steps: the off-line phase and the on-line phase. The PGD method involves in an iterative procedure where the enrichment functions are determined by a fixed-point alogorithm. The novelty of the PGD method is that there is no a *priori* knowledge of the solution required as in the POD method. It permits to compute, once and for all, a meta model that includes all the possible solutions where model parameters can be set as extra problem dimensions. For that, the off-line phase leads to a vademecum and the on-line phase uses the results from the first step to obtain in real time the response of the FE model under a particular combination of model parameters. In this thesis the PGD method is applied to static solid mechanics problems considering the boundary conditions and material parameters as extra problem dimensions, in particular the load position and the Poisson's ratio. The PGD method has been implemented in the StaBIL framework which is a Matlab toolbox created by the Structural Mechanics Section of KU Leuven. In these applications, the PGD method has demonstrated its power since it leads a very low computation time and so achievable even on average performance platforms.

Chapter

1 Introduction

Models in engineering and science describe complex systems using a large number of variables and parameters. The goal of *model order reduction* is to use different techniques to reduce the number of variables and thus to reduce the computational cost and to achieve fast-response for real-time simulations. The model reduction is based on the dimension reduction of the discretized solution space of the governing partial differential equations and the parametric space to be explored. This thesis will discuss a recently introduced model order reduction technique: the proper generalized decomposition method (PGD method) applied to solid mechanics problems. The PGD method is a dimensionality reduction algorithm. Solving decoupled problems is computationally much less expensive than solving multidimensional problems. This new simulation technique has been proposed recently by researches of the University of Zaragoza (Spain) [2].

1.1 Proper generalized decomposition method

The proper generalized decomposition method is based on the assumption of a separated form of the unknown field variables and it has demostrated its capabilities in dealing with high dimensional problems overcoming the strong limitations of classical approaches. The method allows the introduction of several material or geometrical parameters as extra problem dimensions. In the PGD method the problem is solved only once in order to obtain a general solution that includes all the possible solutions for every parameter affecting the solution. For that the PGD method leads a sort a *computational vademecum*. Under this consideration, the real time simulation is immediately available even for complex scenarios.
The PGD method is based on two steps:

- off-line phase
- on-line phase

Combining these steps, the solution can be obtained. In the first step the general PGD solution is obtained while the second step uses the results from the first step in order to show in real time the response, for instance the displacement of a cantilever

beam under a varying load position or for varying Poisson's ratio. Furthermore, when the solution is computed in the off-line phase, it can be uploaded in devices like smartphones, tablets or any digital platform.
The reduction made by the PGD method makes it possible to solve multidimensional models efficiently by means of treating parameters as extra coordinates. The main novelty of the PGD method is the construction of a sum of separated functions a *priori*, without any prior knowledge of the solution, using an iterative scheme.
Considere a PDE of a given problem, in general the solution depends on different parameters m. Assuming that at iteration n of the procedure the convergence is reached, the general form of the solution u^n up to iteration n is:

$$u^n\left(x,t,p^1,p^2,....,p^m\right) \simeq \sum_{i=1}^{n} F_i\left(x\right)\cdot T_i\left(t\right)\cdot P_i^1\left(p^1\right)\cdot ....\cdot P_i^m\left(p^m\right) \tag{1.1}$$

where the functions F_i , T_i , P_i^m are in principle unknown and x , t , p_i^m represent the parameters affecting the solution which are defined in a general domain Ω_i of moderate dimension $\Omega_i \subset \mathbb{R}^{\mathrm{d}}$. Note that the choice of an appropriate truncation level n depends on the level of accuracy.
Equation (1.1) introduces many extra coordinates, e.g. the spatial coordinate, the time coordinate and other coordinates p_i^m, thus the dimensionality of the resulting model increases. However, the PGD method allows to treat these problems advantageously. It is an iterative method that allows to compute the unknown functions by implementing a loop until convergence. The PGD method will be applied to solid mechanics by using a fixed point algorithm that usually provides very good results problems [2, 4].
The PGD method, unlike many other model order reduction techiques, does not need preliminar experiments or statistical treatment like the Proper Orthogonal Decomposition (POD) method. The POD method uses so-called *snapshots* that are empirical realizations of the problem under different parameter values of the considered problem. The PGD method results in a generalization of the POD method, which is the classical approach in reduced-order modelling. Once the off-line phase is completed, it allows the user to change the parameters during the simulation and see the results in real-time.
Many problems in science and engineering are defined in high-dimensional spaces and this introduces difficulties in a context of mesh-based techniques such as finite differences, finite volumes or finite elements. Indeed, the *curse of dimensionality* increases the computation time: the solution has to be recomputed for every combination of parameters. However, from an implementation point of view, the PGD method has a clear barrier to entry and it could appear complex to implement.
Simulation of real engineering problems as simple physical equations becomes a task of hours, indeed a simple but efficient example is the Schrodinger equation. A nice statement about it has been made by R.B. Laughlin: "no computer existing, or that will ever exist, can break this barrier because it is a catastrophe of dimension" [10]. The implementation of augmented learning strategies for simple or complex engineering or physical problems is very interesting but at the same time incredibly

challenging. So the possibility of solving them by using handheld platforms seems out of reach.
In conclusion, traditional simulation-based engineering sciences make use of *static data*. The word *static data* means that inputs, like the parameters defining the problems, cannot be changed during the simulation like in a classic finite element software. Nowadays there is an interesting link between the simulation tools and external dynamic data for real-time simulation and that is why these applications are becoming more frequent. The *Dynamic Data Driven Application System* (DDDAS) constitutes one of the most applications of simulation-based engineering sciences [16].

1.1.1 PGD algorithm

A typical solver for a general PGD method is a fixed point scheme. The basic idea is to compute each of the terms in equation (1.1) one at a time assuming the others are known until reaching convergence of the procedure considering a given tolerance. In general, a PGD solver performs in several steps that depend on the number of functions to be determined. Typically, the main loop is the search for modes which is controlled by the choice of specific maximum number of modes n. The word *modes* is used to describe the outputs of the PGD implementation which can be the *mechanical modes* for the *spatial coordinate* $\mathbf{x}$ or the *extra modes* for the *extra coordinates* p_i^m. The stopping criterion has been defined introducing the norm of the displacements between interations n and $n-1$.
The PGD method is an iterative procedure and it is expected that the contribution of the following iterations to the solution becomes smaller and smaller when n increases. Within the main loop, there are several iterations called n, and the loop is controlled by a stopping criterion which compares the given tolerance TOL, typically $TOL = 10^{-3}$, with the solution of two consecutive iterations:

$$\frac{|u^n - u^{n-1}|}{|u^{n-1}|} < TOL \tag{1.2}$$

To ensure the end of the iterative procedure, a maximum iteration number $nmax_{iter}$ is also specified. In this thesis the fixed point algorithm has been used since it gives very good results. In numerical analysis, fixed point iteration is a method of computing iterated functions. In more detail, given a function f defined on the real numbers with real values and given a point x_0 in the domain of f, the fixed point algorithm is:

$$x^{n+1} = f(x^n) \qquad \text{with} \qquad n = 0, 1, 2, ... \tag{1.3}$$

which leads to the sequence of successive approximations x_0, x_1, x_2,... by fixed point method. Finally, the convergence is checked by equation (1.2).
A typical PGD algorithm involves the following code. In particular: n is the iteration number, TOL is the tolerance, $nmax_{iter}$ is the maximum iterations number, $error_{iter}$ is the relative error up to iteration n and $Exitflag$ is an integer that is a code for the reason the solver halted its iterations. In general positive $Exitflag$ corresponds to successful outcomes.

Algorithm 1 Typical PGD algorithm

Result: Functions to approximate the solution.

Create spatial mesh and parametric mesh for the model
 Initialise $\mathbf{F}$ *and* P^m
 Specify the input n, TOL, $nmax_{iter}$
 while $error_{iter} > TOL$ & $n < nmax_{iter}$ **do**
  Initialise $\mathbf{F_i}$ *and* P_i^m
  while $ExitFlag > TOL$ **do**
   Solve the spatial function $\mathbf{F_i}$
   for $j = 1$ *to* m **do**
    Solve the extra function P_i^m
   end
  end
  Evaluate and update the amplitude $error_{iter}$
   Check the convergence:

$$\frac{|u^n - u^{n-1}|}{|u^{n-1}|} < TOL$$

  Save the amplitude $error_{iter}$, functions $\mathbf{F_i}$ and P_i^m into vademecum
end

1.2 PGD applications

Recently, an experiment by MIT showed that Ethiopian children were able to learn to read by themselves with the aid of tablets specially equipped [17]. Thus the *augmented learning* has emerged as a new way to adapt the environment to the learner. The use of devices permit benefits like portability, interactivity and sociability [15].
As described this technique features the separated representation of the solution, so that the relationship between the solution complexity scale and the dimension of solution space is reduced from exponentially to linearly [2].
The power of the PGD method is demonstrated in many different fields, such as structural analysis, structural optimization, computational rheology, computational fluid dynamics, heat transfer and surgery applications as well [4]. One of the first works proposed by F.Chinesta and his coworkers was the polymer modeling problem [1].

1.2.1 Palpation of the liver

An interesting PGD application lies in the field of virtual surgery training, for instance, surgery planning. This constitutes a *third generation* of surgery simulators [12]. The liver is the biggest gland in the human body, after the skin. Its FE model has been obtained from a mesh composed of 8559 nodes and 10519 tetrahedra. Figure

1.1 shows the finite element model for the human liver from two different points of view.

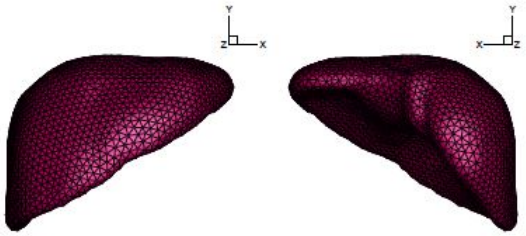
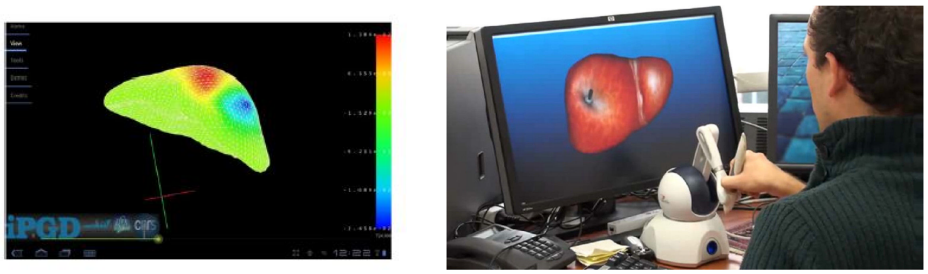


FIGURE 1.1: Finite element model for the human liver [12].

The liver is connected to the diaphragm by the coronary ligament so it seems reasonable to assume it to be constrained at the posterior face by the rest of the organs, while the anterior face is accessible to the surgeon. Even if the assumed boundary conditions are not strictly correct from a physiological point of view, the main goal of the authors was to show that the model can be solved under real-time constraints with reasonable accuracy. The details on the the mechanical parameters of the liver were not very accurate. The authors used a Young's modulus of 160 kPa and Poisson's ratio of 0.48, corresponding to a nearly incompressible material. The boundary $\overline{\Gamma}$ where the load can be located is a surface and it includes 2009 of the 8559 nodes of the model. The model's solution was decomposed by a total of $k = 167$ functional pairs $\mathbf{X}_j^k(\mathbf{x}) \cdot \mathbf{Y}_j^k(p)$, where the function $\mathbf{X}$ is the spatial function and so it includes the three components of the displacements for each node, while the function $\mathbf{Y}$ is the additional function that depends only on the extra coordinate: the load position p. The solution provided by the method agrees well with the reference finite element solutions obtained by employing full-Newton-Raphson iterative schemes. The computed solution can be stored in a compact form so that is possible to use



(a) (b)

FIGURE 1.2: (a) Interactive palpation of a liver, (b) Use of haptic device in the on-line phase [12].

it on devices such as smartphones and tablets. Figure 1.2 shows the online phase where the finite element model for the human liver is uploaded and the user is able to touch the liver with an haptic device. This application leads to so-called *simulator of third generation.*

1.2.2 Water agitation: Mataró harbor (Spain)

Solving the Helmohltz equation for a large number of input variables in a heterogeneous and unbounded domain represents a challange due to the particular structure of the Helmholtz operator and the sensitivity of the solution to small variations of the data. In this application, the authors have considered a reduced order model to determine the wave propagation in every part of the domain for any incoming wave direction and frequency [11]. The original problem involves an exponential growth of degrees of freedom (the so-called *curse of dimensionality*) when using standard mesh-based discretization techniques. For that a reduced order model can circumvent this critical difficulty and solve the problem in an intrusive way with the same accuracy of a FEM, but with less computational effort.
In this section the water agitation in Mataró harbor, located in the norther part of Barcelona is studied. In this problem, the number of reflected waves increase and so the problem from the computational point of view is hard. Using the PGD method, the solution is fully parameterized with space, frequency and incoming direction, so that the solution depends on several parameters $\mathbf{u}(x, y, \omega, \theta)$. Its approximated

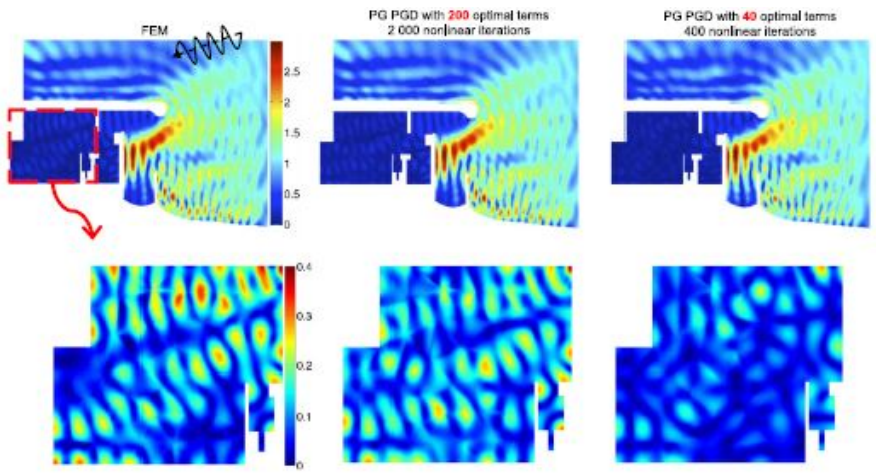


Figure 1.3: Mataró harbor: wave amplification for the particular case $\omega = 0.61 rad/s$ and $\theta = 194.5$. It shows the spatial FEM (left column), PGD with 2000 nonlinear iterations (middle column) and PGD with 400 nonlinear iterations (right column) [11].

solution, called $\mathbf{u}^{\mathrm{n}}$, can be written in separated representation as:

$$\mathbf{u}^n(x, y, \omega, \theta) = \sum_{i=1}^{n} \mathbf{F_1^i}(x, y)\mathbf{F_2^i}(\omega, \theta) \tag{1.4}$$

therefore the solution is splitted in two dimensions.

Incidents waves are in accordance with observations in the region: $\omega \in [0.39, 0.63]$ (from 10s to 16s of wave period) and $\theta \in [1.05\pi, 3\pi/2]$. The model has been built using 15 757 nodes for (x, y) and 50×50 nodes for (ω, θ). In this case the computational cost of the offline phase is determined by the number of spatial problems and so the number of iterations needed for the convergence. Actually the authors remark a drastic increase of the computational cost to reach an engineering accuracy in the areas where a lot of reflections are involved. Figure 1.3 shows the wave amplification for an unfavorable propagation case and the spatial computation with FE method is used as a reference.
Finally, a drastic increase on the computational cost is observed to reach an engineering accuracy in the area where a lot of reflections are involved. The exterior harbor region requires a total of 400 nonlinear iterations, at least 5 times more are required to capture the wave amplification in the interior region, since there are much more reflective areas [11]. A similar problem has been solved for the Barcelona harbor. In this case the geometry is more complex since the size of the harbor is bigger. The model has been built using 2×10^5 nodes for (x, y) and 100×50 nodes for (ω, θ). Figure 1.4 shows the problem statement and figure 1.5 shows a particular solution for a particular combination of ω and θ.

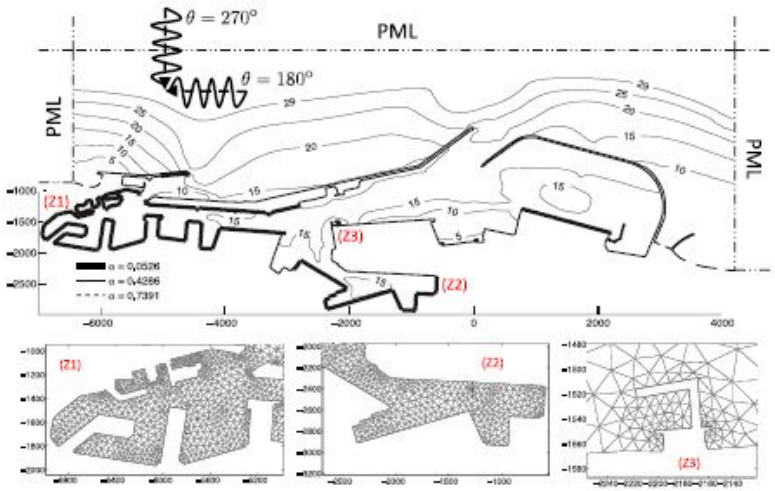


FIGURE 1.4: Barcelona harbor: problem statement. Spatial domain with values of the absorbing coefficient α on the boundary and and contour of the bathymetry [11].

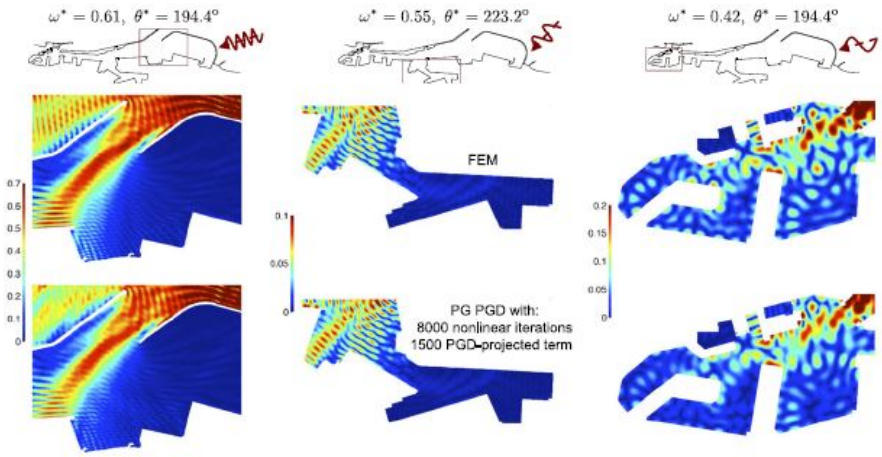


Figure 1.5: Barcelona harbor: wave-height in different harbor areas. In particular: short waves with $(0.61, 1.08\pi)$ (left), mid waves with $(0.55, 1.24\pi)$ (middle) and long waves with $(0.42, 1.08\pi)$ (right). The spatial FEM solution (top) and the PG PGD interpolated solution (bottom) with 1500 PGD-projected terms (8000 solves) are shown [11].

1.2.3 Parametric model of a thermal process

The PGD method has been applied to generate a parametric model of a material flowing in a heated die [8]. The 2D thermal process is sketched in figure 1.6. The thermal flow is assumed with a velocity v inside a die Ω of length L and width H. The die is composed of two heating devices of length L_1 and L_2, whose temperature θ_1 and θ_2 can change within an interval $[\theta_{min}, \theta_{max}]$.

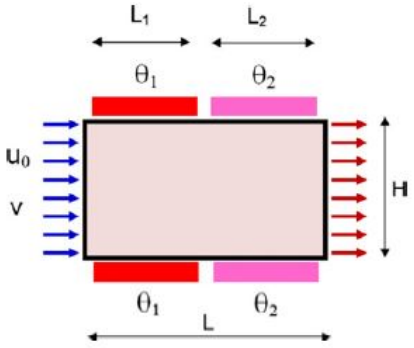


Figure 1.6: Thermal process consisting of two heating devices located on the die walls [8].

The steady temperature field $u(\mathbf{x}, \theta_1, \theta_2)$ in any point of the die can be obtained from the solution of the 2D heat transfer equation that involves advection and diffusion mechanics. The velocity has been assumed unidirectional. The heat transfer equation looks like:

$$\rho c(v\frac{\partial u}{\partial x}) = k\Delta u \tag{1.5}$$

where k is the thermal conductivity, ρ is the density and c is the specific heat. In this example the extra coordinates considered by the authors are the temperatures θ_1 and θ_2, but other parameters such as ρ, k and c can be set as extra parameters as well. The problem is multimensional and the solution depends on several parameters $u(x, y, \theta_1, \theta_2)$. Considering a separeted representation of temperature field, the solution can be written as:

$$u(x, y, \theta_1, \theta_2) = \sum_{i=1}^{n} F_i(x, y)\Theta_i^1(\theta_1)\Theta_i^2(\theta_2) \tag{1.6}$$

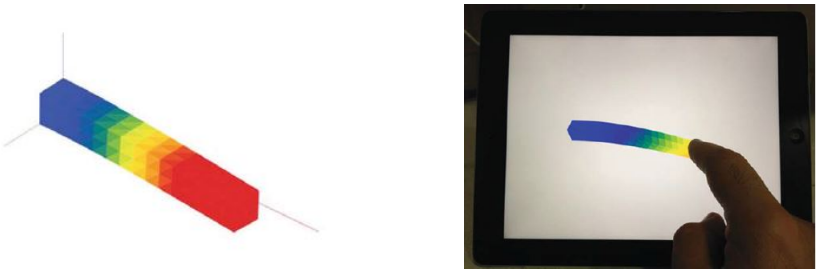
Applying the PGD method to this problem results in three steps where the functions F_i, Θ_i^1 and Θ_i^2 are determined iteratively. Finally, the user can choose in the online phase any value of the temperature and obtain in real-time the response, in this case the temperature in any point of the wall. Figure 1.7 shows that the user can upload the meta-model on devices as smartphones or tablets.

FIGURE 1.7: Online phase: the user is able to choose the temperatures θ_1 and θ_2 and get in real time the temperature in any point of the system [8].

1.2.4 Structural mechanics applications

Recently, the PGD method has been applied to structural mechanics problems. One of these application, for instance, is the study of a cracked plate where the extra parameters are the thickness B of the plate and the Poisson's ratio ν [9]. The user is able to choose these parameters and obtain in real time the solution $u(x, y, z, B, \nu)$. Another interesting application is the field of structural dynamics. In this example, the model order reduction of initial and boundary value problems is a particularly challenging task. Indeed, the initial conditions such as displacement and velocity can have a large number of values. For that it should be parameterized in a proper

way [5]. The execution of the program produces a window with the tip displacement and the user can play with the mouse and see the static displacements of the beam under a unit load applied at the upper surface. Figure 1.8 shows the finite element model and the PGD program uploaded on a tablet. The user is able to move the point of application of the load and see the beam displacements in real time. More applications are avaiable on the University of Zaragoza website [14].



(a) (b)

Figure 1.8: (a) Web implementation of the algorithm. It represents a linear elastic beam, (b) Implementation on a *html* file with javascript. It represents the response of a cantilever beam [5].

1.3 Goals of this MSc thesis

In this MSc thesis, the PGD method will be applied to static solid mechanics problems. The application of this approach is very interesting since it is possible to obtain the solution, for instance the displacements, in real time. Introducing *extra problem dimensions* in the model, a parametric problem is obtained. The extra parameters considered in this thesis are the *load position* and the *Poisson's ratio*. The numerical integrations of all the necessary mathematical formulations that describe the problem, within the PGD framework, have been performed in StaBIL, a Matlab toolbox realized by the Structural Mechanics Section of KU Leuven.
For the first problem, the resulting model is a beam displayed on the screen and the user is able to choose the load position by clicking on the beam surface and obtain in real time the beam configuration for varying load positions. The second problem shows the capability of the program to display several beam configurations for varying Poisson's values. In this case, the user is able to choose the Poisson's ratio digiting on the keyboard the desidered value.

1.4 Organization of the text

This work applies the PGD method to static solid mechanics problems. In particular, the PGD method considering the load position acting on the beam and the Poisson's ratio as extra parameters. In order to find similarities with the FE method, the FE solution has been provided using StaBIL.

- *Chapter 1* gives a general introduction to the PGD method with some of its applications. In particular the PGD method as third generation of the so called Reduced Order Models (ROM) and the description of the two main steps is given: off-line phase and on-line phase. Further the goals of the thesis and the structure of the text are explained.

- *Chapter 2* gives the FE formulations considering a static solid mechanics problems. Starting from the PDEs of a 3D solid continuum, the strong and the weak form of the problem introducing the discretization using the FE approach are derived. Finally, the definition of the stiffness matrix $\mathbf{K}$ and the external load vector $\mathbf{f}$.

- *Chapter 3* analyzes the first application of the PGD method applied to solid mechanics problem considering the load position as extra parameter. The load is supposed acting on the surface of a clamped beam. In the spirit of the PGD method the solution of the problem is expressed with a separeted representation considering two functions: the first one depends on the spatial coordinate $\mathbf{x}$ and the second one on the load position s. The load position's domain can be expressed by a linear domain using 1D elements. After implementation in Matlab, the PGD solution has been compared with the FE solution found with StaBIL.

- *Chapter 4* analyzes another challenging problem using the PGD method where the Poisson's ratio is considered as extra parameter of the problem. In this case the solution in separated representation is expressed by two functions: the first one depends on the spatial coordinate $\mathbf{x}$ and the second one depends on the Poisson's ratio ν. The Poisson's domain can be expressed by linear domain using 1D elements. All the numerical integrations has been performed in Matlab and in order to find similarities, the PGD solution has been compared with the FE solution found with StaBIL.

- *Chapter 5* makes final conclusions on the thesis and shows some detected issues. Moreover, possible future works and future development are suggested.

Chapter

2 The finite element method

2.1 Introduction

The finite element method (FEM) is used for the numerical solution of differential equations [6]. From a mathematical point of view, a problem is often decribed by differential equations. For solid mechanics problems, the entire structure is composed of several small elements and these elements are connected at points called *nodes*. The assembly of the elements is called a *model*. The particular arrangement of elements is called *the finite element mesh* (figure 2.1). The system of equations is discretized and

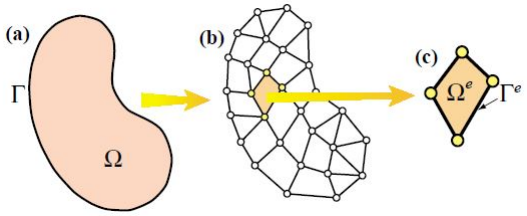


FIGURE 2.1: (a)Solid, (b) FE mesh, (c) Extraction of one particular element [13].

solved for the unkown nodal field variables. The field quantity over the entire domain is thus approximated element by element. The FEM solution is not exact, but it can be improved using a finer mesh. The modeling step leads to a mathematical problem in order to represent a physical problem. A model can be devised after the physical nature of the problem has been understood. A geometric model becomes a *mathematical model* when its behavior is described by selected differential equations and boundary conditions. It is very important to remark that finite element analysis is a simulation and not reality and is applied to the mathematical problem. The matemathical model is an idealization where the geometry, material proprieties, loads and boundary conditions are simplified.
The field is discretized because the equations are discretized using the shape functions. The entire continuous field is represented by a piecewise continuous field defined by a finite number of nodal quantities and simple interpolation within each element. Introducing this approximative model two types of errors have now been introduced:

modelling errors and *discretization errors*. The first can be reduced by improving the model and the second error can be reduced using more elements. Also the numerical integration introduces *numerical error* since it is not exact.
The essence of the finite element method is approximation by piecewise interpolation of a field quantity. A polynomial interpolation is often used. In the general case, it is possible to write the solution of the problem by interpolating the nodal values of the variable field with interpolation functions. In order to define the approximation of a field variable , it is important to define the degrees of freedom of the problem. These are entities that govern the spatial variation of the field.
Finally performing a finite element analysis involves the following steps:

- *Preprocessing:* Input data describes geometry, material propreties, loads, and boundary conditions. Software can automatically prepare much of the FE mesh but the element type and mesh should be provided.
- *Numerical analysis:* The software generates system of matrices, such as the stiffness matrix for the static analysis of a problem. The equilibrium equation $\mathbf{KU} = \mathbf{P}$ is solved in order to determine the values of the field quantities $\mathbf{U}$ at the nodes of the FE mesh.
- *Postprocessing:* The FEM solution and quantities derived from it are graphically displayed. For instance, the evalutation of strains/stresses from the nodal displacements is possible.

2.2 General solution of solid mechanics problems

In this chapter the weighted residual method is applied to derive the solutions of general solid mechanics. There are two main steps:

1. Description of the PDE of the problem and its equivalent integral expression
2. Discretizing the governing equations using the *shape functions*

2.2.1 Problem geometry

The problem is defined on a physical domain Ω in the space $\mathbb{R}^3$ and its boundary is a closed curve Γ. The boundary Γ (figure 2.2) is divided in two parts that do not overlap:

- the Dirichlet boundary Γ_u
$$u - \bar{u} = 0 \tag{2.1}$$
- the Neumann boundary Γ_t
$$t - \bar{t} = 0 \tag{2.2}$$

The Dirichlet boundary conditions impose displacement restrictions $\bar{u}$, while the Neumann boundary conditions impose traction restrictions $\bar{t}$.

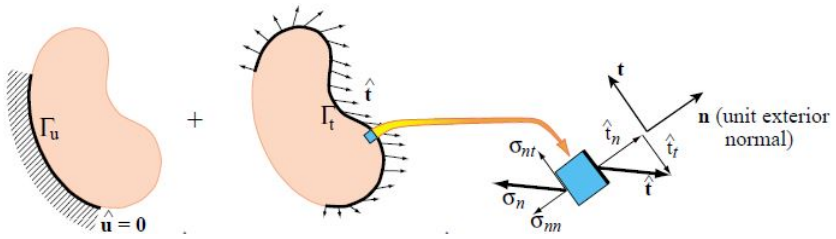


FIGURE 2.2: Displacement and force (stress, traction) boundary conditions for the plane stress problem [13].

2.2.2 Weighted residual method: Galerkin approach

The weighted residual method is a method developed to obtain an approximate solution to the differential equations describing the problem [6]. The differential equation can be written in the general form as:

$$A(\Phi) = \mathbf{L}(\Phi) + f(\mathbf{x}) = 0 \tag{2.3}$$

where $\Phi(\mathbf{x})$ is the unknown field variable and $f(\mathbf{x})$ is a known function. $\mathbf{L}$ denotes the differential operator involving spatial derivative of Φ, which specifies the actual form of the differential equation. The weighted residual form method involves two major steps. In the first step, an approximate solution based on the general behaviour of the dependent variable is assumed. The assumed solution is selected so that it satisfies the boundary conditions for Φ:

$$B(\Phi) = \mathbf{M}(\Phi) + t(\mathbf{x}) = 0 \tag{2.4}$$

The boundary conditions include prescriptions of displacements or trections on the surface of a body. In general a distributed load can act in the tangential or normal direction. On any boundary (including one not perpendicular to a coordinate axis) normal and tangential loads can be expressed as surface tractions, which are forces per unit of surface area.

Let $\Psi(x) \approx \Phi(x)$ be an approximate solution to the differential equation 2.3. When $\Psi(x)$ is substituted in the differential equation, it is unlikely that the equation is satisfied. So, we obtain:

$$A(\Psi) = \mathbf{L}(\Psi) + f \neq 0 \qquad \text{in} \qquad \Omega \tag{2.5}$$

$$B(\Psi) = \mathbf{M}(\Psi) + t \neq 0 \qquad \text{on} \qquad \Gamma \tag{2.6}$$

Multiplying equations (2.5) and (2.6) by an arbitrary *weight function* $w(\mathbf{x})$ and integrating over the domain Ω and boundary Γ we obtain:

$$\int_\Omega w\left[\mathbf{L}(\Psi) + f\right] \mathrm{d}\Omega + \int_\Gamma w\left[\mathbf{M}(\Psi) + t\right] \mathrm{d}\Gamma \neq 0 \tag{2.7}$$

Since the assumed solution is only approximate, it does not in general satisfy the differential equation and hence result in an error called *residual*. The residual is then minimized over the entire solution domain to obtain a good approximation of the solution. The second step is to solve the system of equations resulting from the first step subject to the prescribed boundary conditions to yield the approximate solution. In the Galerkin version of the weighted residual form, the shape functions for the displacements and the virtual displacement fields are the same. And so the weight function is:

$$w_i = N_i \qquad\qquad i = 1, 2, ..., n \tag{2.8}$$

The approximation is then:

$$\Psi \approx \hat{\Psi} = N_1 u_1 + N_2 u_2 + ... = \sum_{i=1}^{n} N_i u_i = \mathbf{N}\underline{\mathbf{u}} \tag{2.9}$$

In this expression $N_1, N_2, ..., N_n$ are n indipendent functions of x,y and z. The unknowns coefficients u_1, u_2,...,u_n have to be determined in order to obtain a best solution.

2.3 Mathematical formulation of solid mechanics problems

This section analyzes a 3D solid mechanics problem using the FEM.
Let σ_{ij} be the stress and ϵ_{ij} the strain, where i is the direction of the stress/strain and j the normal direction. The equilibrium equations along x-direction, y direction and z-direction are:

$$\frac{\partial \sigma_{xx}}{\partial x} + \frac{\partial \sigma_{yx}}{\partial y} + \frac{\partial \sigma_{zx}}{\partial z} + \rho b_x = 0 \tag{2.10}$$

$$\frac{\partial \sigma_{xy}}{\partial x} + \frac{\partial \sigma_{yy}}{\partial y} + \frac{\partial \sigma_{zy}}{\partial z} + \rho b_y = 0 \tag{2.11}$$

$$\frac{\partial \sigma_{xz}}{\partial x} + \frac{\partial \sigma_{yz}}{\partial y} + \frac{\partial \sigma_{zz}}{\partial z} + \rho b_z = 0 \tag{2.12}$$

In matrix-vector notation:

$$\begin{bmatrix} \frac{\partial}{\partial x} & 0 & 0 & \frac{\partial}{\partial y} & 0 & \frac{\partial}{\partial z} \\ 0 & \frac{\partial}{\partial y} & 0 & \frac{\partial}{\partial x} & \frac{\partial}{\partial z} & 0 \\ 0 & 0 & \frac{\partial}{\partial z} & 0 & \frac{\partial}{\partial y} & \frac{\partial}{\partial x} \end{bmatrix} \begin{Bmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{zz} \\ \sigma_{yx} \\ \sigma_{zy} \\ \sigma_{xz} \end{Bmatrix} + \rho \begin{Bmatrix} b_x \\ b_y \\ b_z \end{Bmatrix} = 0 \tag{2.13}$$

Or in a short form:

$$\mathbf{L}^{\mathrm{T}}\boldsymbol{\sigma} + \rho\mathbf{b} = 0 \tag{2.14}$$

where $\mathbf{b}$ are the body forces defined per unit volume, positive when acting in positive coordinate directions, and ρ is the material density.

The two boundary conditions considered are:

- Neumann BC: imposed traction on $\mathbf{\Gamma}_t$

$$\begin{aligned} t_x &= \sigma_{xx}n_x + \sigma_{yx}n_y + \sigma_{zx}n_z = \bar{t}_x \\ t_y &= \sigma_{xy}n_x + \sigma_{yy}n_y + \sigma_{zy}n_z = \bar{t}_y \\ t_z &= \sigma_{xz}n_x + \sigma_{yz}n_y + \sigma_{zz}n_z = \bar{t}_z \end{aligned} \tag{2.15}$$

- Dirichlet BC: imposed displacement on $\mathbf{\Gamma}_u$

$$\begin{aligned} u_x - \bar{u}_x &= 0 \\ u_y - \bar{u}_y &= 0 \\ u_z - \bar{u}_z &= 0 \end{aligned} \tag{2.16}$$

The strain-displacement relations describe how a body deforms:

$$\epsilon_{xx} = \frac{\partial u_x}{\partial x} \tag{2.17}$$

$$\epsilon_{yy} = \frac{\partial u_y}{\partial y} \tag{2.18}$$

$$\epsilon_{zz} = \frac{\partial u_z}{\partial z} \tag{2.19}$$

$$\epsilon_{xy} = \frac{1}{2}\left(\frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x}\right) \tag{2.20}$$

$$\epsilon_{yz} = \frac{1}{2}\left(\frac{\partial u_y}{\partial z} + \frac{\partial u_z}{\partial y}\right) \tag{2.21}$$

$$\epsilon_{zx} = \frac{1}{2}\left(\frac{\partial u_z}{\partial x} + \frac{\partial u_x}{\partial z}\right) \tag{2.22}$$

$$\gamma_{xy} = 2\epsilon_{xy} \qquad \gamma_{yz} = 2\epsilon_{yz} \qquad \gamma_{zx} = 2\epsilon_{zx} \tag{2.23}$$

In matrix-vector notation, the strain-displacement relations are:

$$\begin{Bmatrix} \epsilon_{xx} \\ \epsilon_{yy} \\ \epsilon_{zz} \\ \gamma_{xy} \\ \gamma_{yz} \\ \gamma_{zx} \end{Bmatrix} = \begin{bmatrix} \frac{\partial}{\partial x} & 0 & 0 & \frac{\partial}{\partial y} & 0 & \frac{\partial}{\partial z} \\ 0 & \frac{\partial}{\partial y} & 0 & \frac{\partial}{\partial x} & \frac{\partial}{\partial z} & 0 \\ 0 & 0 & \frac{\partial}{\partial z} & 0 & \frac{\partial}{\partial y} & \frac{\partial}{\partial x} \end{bmatrix}^{\mathrm{T}} \begin{Bmatrix} u_x \\ u_y \\ u_z \end{Bmatrix} \tag{2.24}$$

or:

$$\boldsymbol{\epsilon} = \mathbf{L}\mathbf{u} \tag{2.25}$$

The constitutive relation is defined as:

$$\boldsymbol{\sigma} = \mathbf{D}\boldsymbol{\epsilon} \tag{2.26}$$

where the constitutive matrix for a 3D solid is:

$$\mathbf{D} = \begin{bmatrix} 2\mu+\lambda & \lambda & \lambda & 0 & 0 & 0 \\ \lambda & 2\mu+\lambda & \lambda & 0 & 0 & 0 \\ \lambda & \lambda & 2\mu+\lambda & 0 & 0 & 0 \\ 0 & 0 & 0 & \mu & 0 & 0 \\ 0 & 0 & 0 & 0 & \mu & 0 \\ 0 & 0 & 0 & 0 & 0 & \mu \end{bmatrix} \tag{2.27}$$

where the Lamé constants are:

$$\mu = \frac{E}{2(1+\nu)} \tag{2.28}$$

$$\lambda = \frac{\nu E}{(1+\nu)(1-2\nu)} \tag{2.29}$$

In case of *2D plane strain* the constitutive matrix $\mathbf{D}$ is:

$$\mathbf{D} = \frac{E}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1-\nu & \nu & 0 \\ \nu & 1-\nu & 0 \\ 0 & 0 & \frac{1-2\nu}{2} \end{bmatrix} \tag{2.30}$$

In case of *2D plane stress* the constitutive matrix $\mathbf{D}$ is:

$$\mathbf{D} = \frac{E}{1-\nu^2} \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1-\nu}{2} \end{bmatrix} \tag{2.31}$$

where E is the Young's modulus and ν is the Poisson's ratio.

2.3.1 Weighted residual method applied to solid mechanics problems

In order to derive the strong form of the problem, the equilibrium equations and mechanical boundary conditions can be written in the weighted residual form. The principle of virtual work is used in this section to derive the stiffness matrix and the load vector associated to the problem. The virtual displacements δu_x , δu_y and δu_z are any displacement fields taht satisfies kinematic boundary conditions. The displacement field is admissible and for this reason does not violate compatibility boundary conditions. The weighted residual form is:

$$\begin{aligned} &\int_\Omega \left[\left(\frac{\partial \sigma_{xx}}{\partial x} + \frac{\partial \sigma_{yx}}{\partial y} + \frac{\partial \sigma_{zx}}{\partial z} + \rho b_x \right) \delta u_x + \left(\frac{\partial \sigma_{xy}}{\partial x} + \frac{\partial \sigma_{yy}}{\partial y} + \frac{\partial \sigma_{zy}}{\partial z} + \rho b_y \right) \delta u_y \right. \\ &\left. + \left(\frac{\partial \sigma_{xz}}{\partial x} + \frac{\partial \sigma_{yz}}{\partial y} + \frac{\partial \sigma_{zz}}{\partial z} + \rho b_z \right) \delta u_z \right] \mathrm{d}\Omega - \int_{\Gamma_t} \left[(t_x - \bar{t}_x)\, \delta u_x + (t_y - \bar{t}_y)\, \delta u_y \right. \\ &\left. + (t_z - \bar{t}_z)\, \delta u_z \right] \mathrm{d}\Gamma = 0 \end{aligned} \tag{2.32}$$

In matrix-vector notation:

$$\int_{\Omega} \delta\mathbf{u}^{\mathrm{T}}(\mathbf{L}^{\mathrm{T}}\boldsymbol{\sigma} + \rho\mathbf{b})\mathrm{d}\Omega + \int_{\Gamma_t} \delta\mathbf{u}^{\mathrm{T}}(\mathbf{t} - \bar{\mathbf{t}})\mathrm{d}\Gamma = 0 \tag{2.33}$$

where:

$$\delta\mathbf{u} = \begin{Bmatrix} \delta u_x & \delta u_y & \delta u_z \end{Bmatrix}^{\mathrm{T}} \tag{2.34}$$

Integrating by part the first term and applying Gauss's theorem:

$$\begin{aligned} \int_{\Omega} \frac{\partial \sigma_{xx}}{\partial x}\delta u_x \mathrm{d}\Omega &= \int_{\Omega} \frac{\partial(\sigma_{xx}\delta u_x)}{\partial x}\mathrm{d}\Omega - \int_{\Omega} \sigma_{xx}\frac{\partial \delta u_x}{\partial x}\mathrm{d}\Omega \\ &= \int_{\Gamma_t} \sigma_{xx}\delta u_x n_x \mathrm{d}\Gamma - \int_{\Omega} \sigma_{xx}\frac{\partial \delta u_x}{\partial x}\mathrm{d}\Omega \end{aligned} \tag{2.35}$$

For all terms, this results in:

$$\begin{aligned} &\int_{\Omega} \left[\sigma_{xx}\delta\epsilon_{xx} + \sigma_{yy}\delta\epsilon_{yy} + \sigma_{zz}\delta\epsilon_{zz} + \sigma_{yx}\delta\gamma_{xy} + \sigma_{zy}\delta\gamma_{yz} + \sigma_{xz}\delta\gamma_{zx}\right]\mathrm{d}\Omega \\ &- \int_{\Omega} \rho\left[b_x\delta u_x + b_y\delta u_y + b_z\delta u_z\right]\mathrm{d}\Omega - \int_{\Gamma_t} \left[\left(\sigma_{xx}n_x + \sigma_{xy}n_y + \sigma_{xz}n_z\right)\delta u_x \right. \\ &+ \left.\left(\sigma_{xy}n_x + \sigma_{yy}n_y + \sigma_{zy}n_z\right)\delta u_y + \left(\sigma_{xz}n_x + \sigma_{yz}n_y + \sigma_{zz}n_z\right)\delta u_z\right]\mathrm{d}\Gamma \\ &+ \int_{\Gamma_t} \left[\left(t_x - \bar{t}_x\right)\delta u_x + \left(t_y - \bar{t}_y\right)\delta u_y + \left(t_z - \bar{t}_z\right)\delta u_z\right]\mathrm{d}\Gamma = 0 \end{aligned} \tag{2.36}$$

In matrix notation this is written as:

$$\int_{\Omega} \delta\boldsymbol{\epsilon}^{\mathrm{T}}\boldsymbol{\sigma}\mathrm{d}\Omega = \int_{\Omega} \delta\mathbf{u}^{\mathrm{T}}\rho\mathbf{b}\mathrm{d}\Omega + \int_{\Gamma_t} \delta\mathbf{u}^{\mathrm{T}}\bar{\mathbf{t}}\mathrm{d}\Gamma \tag{2.37}$$

Equation (2.37) represents the virtual work equation.
Let the displacement field $\mathbf{u}$ be interpolated over an element by:

$$\mathbf{u} = \mathbf{N}\underline{\mathbf{u}} \tag{2.38}$$

where $\mathbf{N}$ contains the *shape functions* related to spatial coordinates. The nodal values of the displacement field are denoted as $\underline{\mathbf{u}}$.

The constitutive law leads to:

$$\boldsymbol{\epsilon} = \mathbf{B}\underline{\mathbf{u}} \tag{2.39}$$

where $\mathbf{B}$ is called the *strain-displacement matrix* that includes the first derivative of the shape functions.
Replacing the equations (2.38) and (2.39) in equation (2.37) leads to:

$$\delta\underline{\mathbf{u}}^{\mathrm{T}} \int_{\Omega} \left(\mathbf{LN}\right)^{\mathrm{T}}\mathbf{DLN}\mathrm{d}\Omega\underline{\mathbf{u}} = \delta\underline{\mathbf{u}}^{\mathrm{T}} \int_{\Omega} \mathbf{N}^{\mathrm{T}}\rho\mathbf{b}\mathrm{d}\Omega + \delta\underline{\mathbf{u}}^{\mathrm{T}} \int_{\Gamma_t} \mathbf{N}^{\mathrm{T}}\bar{\mathbf{t}}\mathrm{d}\Gamma \tag{2.40}$$

$$\delta\underline{\mathbf{u}}^{\mathrm{T}} \int_{\Omega} \mathbf{B}^{\mathrm{T}}\mathbf{D}\mathbf{B}\mathrm{d}\Omega\underline{\mathbf{u}} = \delta\underline{\mathbf{u}}^{\mathrm{T}} \int_{\Omega} \mathbf{N}^{\mathrm{T}}\rho\mathbf{b}\mathrm{d}\Omega + \delta\underline{\mathbf{u}}^{\mathrm{T}} \int_{\Gamma_t} \mathbf{N}^{\mathrm{T}}\bar{\mathbf{t}}\mathrm{d}\Gamma \tag{2.41}$$

Equation (2.41) must hold for *any* admissible virtual displacement $\delta\underline{\mathbf{u}}$. Therefore:

$$\left[\int_{\Omega} \mathbf{B}^{\mathrm{T}}\mathbf{D}\mathbf{B}\mathrm{d}\Omega\right] \underline{\mathbf{u}} = \int_{\Omega} \mathbf{N}^{\mathrm{T}}\rho\mathbf{b}\mathrm{d}\Omega + \int_{\Gamma_t} \mathbf{N}^{\mathrm{T}}\bar{\mathbf{t}}\mathrm{d}\Gamma \tag{2.42}$$

Equation (2.42) yields:

$$\mathbf{K}\underline{\mathbf{u}} = \mathbf{f} \tag{2.43}$$

where the stiffness matrix is:

$$\mathbf{K} = \int_{\Omega} \mathbf{B}^{\mathrm{T}}\mathbf{D}\mathbf{B}\mathrm{d}\Omega \tag{2.44}$$

and the external load vector $\mathbf{f}$ includes the body forces and the external load:

$$\mathbf{f} = \int_{\Omega} \mathbf{N}^{\mathrm{T}}\rho\mathbf{b}\mathrm{d}\Omega + \int_{\Gamma_t} \mathbf{N}^{\mathrm{T}}\bar{\mathbf{t}}\mathrm{d}\Gamma \tag{2.45}$$

The stiffness matrix $\mathbf{K}$ is a matrix with dimensions $n_{dof} \times n_{dof}$, the external load vector is a vector $n_{dof} \times 1$ and the nodal values $\underline{\mathbf{u}}$ is a vector $n_{dof} \times 1$, where n_{dof} is the *number of the degree of freedom* of a mechanical system is the number of indipendent parameters that define its configuration, thus considering solid mechanics problems the degrees of freedom of the problem are the unrestrained displacements of each node.

Chapter

3 Boundary conditions as extra problem dimension

3.1 Introduction

This chapter introduces a parametric problem where the load position is considered as an extra parameter of the problem. The load position domain is assumed linear and so it is discretized assuming 1D elements. The PGD method revealed an impressive ability to solve problems just considering parameters as new dimension of the considered problem, thus it leads a sort of parametric phase space [3]. Therefore the parametric problems constitute the most interesting application of the PGD method.

3.2 The load position as extra parameter of the problem

The problem of a moving unit load along a beam leads to the influence line problem that is a graphical representation of a field at a given point of a considered structure when the load is applied at different positions. In order to study the problem for varying load position by applying the PGD method a cantilever beam has been considered. The goal is to obtain a sort of response surface in which by varying the load position, the response of the beam is provided in real time. In this case the response is the deformed configuration of the beam [5].
The PGD authors first had considered the problem as non separable. So that the number of modes needed to express the solution is so big, that no gain is obtained by applying any kind of model reduction technique and therefore it is better to simply simulate it in a straighforward manner, by finite element methods or any other numerical technique [2].
In this chapter, a clamped beam is considered with a moving load acting on the beam surface. Using the PGD method and considering the load position s as extra dimension in the problem, the solution is expressed as:

$$\mathbf{u}\left(\mathbf{x}, s\right) = \sum_{i=1}^{n} \mathbf{F}_i(\mathbf{x}) G_i(s) \tag{3.1}$$

where s represents the load position, $\mathbf{x}$ the spatial coordinates and $\mathbf{u}$ contains three displacement components. $\mathbf{F}_i(\mathbf{x})$ is the spatial vector and $G_i(s)$ is the load vector.

Considering a general solid mechanics problem and neglecting the body forces, the weak form of the problem can be written as:

$$\int_{\Omega\times\overline{\Gamma}} (\mathbf{L}\delta\mathbf{u})^{\mathrm{T}} : \mathbf{D} : \mathbf{L}\mathbf{u}\mathrm{d}\Omega\mathrm{d}\overline{\Gamma} = \int_{\Gamma_t\times\overline{\Gamma}} \delta\mathbf{u}^{\mathrm{T}}\bar{\mathbf{t}}\mathrm{d}\Gamma\mathrm{d}\overline{\Gamma} \tag{3.2}$$

Therefore the solution $\mathbf{u}(\mathbf{x}, s)$ is defined in $\Omega \times \overline{\Gamma}$. In particular Ω is the domain for the spatial coordinate, $\overline{\Gamma}$ is the domain where the load can be applied and Γ_t is the part of the boundary on which Neumann boundary conditions are applied.
The load position for simplicity is considered with unit value and it can be expressed with the Dirac-delta like:

$$\bar{\mathbf{t}} = \mathbf{e}_t \cdot \delta(x - s) \tag{3.3}$$

where $\mathbf{e}_t$ is the force magnitude. The load is expressed in a linear way.
In the spirit of the PGD method the Dirac-delta needs to be approximated with a series of separable functions as:

$$\bar{\mathbf{t}} = \sum_{j=1}^{m} \mathbf{f}_j(\mathbf{x})\, g_j(s) \tag{3.4}$$

where m is the number of functions of the source term. The first function $\mathbf{f}_j(\mathbf{x})$ is the spatial term and g_j is the load term to approximate the load $\bar{\mathbf{t}}$. The goal of the PGD method is to find, in an impressive way, a finite sum of separable functions to approach the solution. Assuming that we have convergence at iteration n of this iterative procedure, the solution can be written as:

$$\mathbf{u}^n(\mathbf{x}, s) = \sum_{i=1}^{n} \mathbf{F}_i(\mathbf{x})\, G_i(s) \tag{3.5}$$

where the solution of the problem $\mathbf{u}(\mathbf{x}, s)$ includes the j-th component of the displacement vector with $j = 1, 2, 3$. Using the FE formulation, equation (3.5) is expressed as:

$$\mathbf{u}^n(\mathbf{x}, s) = \sum_{i=1}^{n} \mathbf{N}(\mathbf{x})\, \underline{\mathbf{F}}_i \mathbf{M}(s)\, \underline{\mathbf{G}}_i \tag{3.6}$$

where $\underline{\mathbf{F}}_i$ and $\underline{\mathbf{G}}_i$ are the nodal values of the functions $\mathbf{F}_i(\mathbf{x})$ and $G_i(s)$ while $\mathbf{N}(\mathbf{x})$ is the shape function for the spatial coordinates and $\mathbf{M}(s)$ is the shape fuction for the load position and it is expressed using 1D elements.
If the rank-n of the approximation does not give the desired accuracy, we look for the $(n+1)$-th term, so that the approximation becomes:

$$\mathbf{u}^{n+1}(\mathbf{x}, s) = \mathbf{u}^n(\mathbf{x}, s) + \mathbf{R}(\mathbf{x})\, S(s) \tag{3.7}$$

where $\mathbf{R}(\mathbf{x})$ and $S(s)$ are the enrichment functions that improve the approximation. The enrichment functions $\mathbf{R}(\mathbf{x})$ and $S(s)$ are determined iteratively with a fixed-point algorithm which gives good results and converges quickly.
The *virtual displacement field* $\delta\mathbf{u}$ is expressed as the following formulation:

$$\delta\mathbf{u}(\mathbf{x}, s) = \delta\mathbf{R}(\mathbf{x})\, S(s) + \mathbf{R}(\mathbf{x})\, \delta S(s) \tag{3.8}$$

In the PGD framework in order to compute the enrichment functions, two steps should be repeated until convergence:

1. The computation of $\mathbf{R}(\mathbf{x})$ assuming $S(s)$ is known
   In this case the virtual displacement field becomes:

$$\delta\mathbf{u}\left(\mathbf{x},s\right)=\delta\mathbf{R}(\mathbf{x})S(s) \tag{3.9}$$

2. The computation of $S(s)$ assuming $\mathbf{R}(\mathbf{x})$ is known
   In this case the virtual displacement field becomes:

$$\delta\mathbf{u}\left(\mathbf{x},s\right)=\mathbf{R}(\mathbf{x})\delta S\left(s\right) \tag{3.10}$$

These two steps should be repeated until convergence and in order to compute them a fixed point method has been adopted. The iterative procedure stops when the norm of the solution at iteration n and $n-1$ is less than the TOL. Equation (1.2) shows the stopping criterion formula.

3.2.1 Computation of $\mathbf{R}\left(\mathbf{x}\right)$ assuming $S\left(s\right)$ is known

Using equations (3.4), (3.5) and (3.9), the weak form of the problem (equation (3.2)) can be written as:

$$\begin{aligned}&\int_{\Omega\times\overline{\Gamma}}\left[\mathbf{L}\delta\mathbf{R}\left(\mathbf{x}\right)S\left(s\right)\right]^{\mathrm{T}}:\mathbf{D}:\left[\sum_{i=1}^{n}\mathbf{L}\mathbf{F}_{i}\left(\mathbf{x}\right)G_{i}\left(s\right)+\mathbf{L}\mathbf{R}\left(\mathbf{x}\right)S\left(s\right)\right]\mathrm{d}\Omega\mathrm{d}\overline{\Gamma}\\&=\int_{\Gamma_{t}\times\overline{\Gamma}}\left[\delta\mathbf{R}\left(\mathbf{x}\right)S\left(s\right)\right]^{\mathrm{T}}\left[\sum_{j=1}^{m}\mathbf{f}_{j}\left(\mathbf{x}\right)g_{j}\left(s\right)\right]\mathrm{d}\Gamma\mathrm{d}\overline{\Gamma}\end{aligned} \tag{3.11}$$

where the operator $\mathbf{L}$ affects functions of the spatial coordinates and $\mathbf{D}$ is the constitutive matrix.
Moving all known terms to the RHS, equation (3.11) becomes:

$$\begin{aligned}&\int_{\Omega\times\overline{\Gamma}}\left[\mathbf{L}\delta\mathbf{R}\left(\mathbf{x}\right)S\left(s\right)\right]^{\mathrm{T}}:\mathbf{D}:\mathbf{L}\mathbf{R}\left(\mathbf{x}\right)S\left(s\right)\mathrm{d}\Omega\mathrm{d}\overline{\Gamma}\\&=\int_{\Gamma_{t}\times\overline{\Gamma}}\left[\mathbf{R}\left(\mathbf{x}\right)\delta S\left(s\right)\right]^{\mathrm{T}}\left[\sum_{j=1}^{m}\mathbf{f}_{j}\left(\mathbf{x}\right)g_{j}\left(s\right)\right]\mathrm{d}\Gamma\mathrm{d}\overline{\Gamma}\\&-\int_{\Omega\times\overline{\Gamma}}\left[\mathbf{L}\delta\mathbf{R}\left(\mathbf{x}\right)S\left(s\right)\right]^{\mathrm{T}}:\mathbf{D}:\sum_{i=1}^{n}\mathbf{L}\mathbf{F}_{i}\left(\mathbf{x}\right)G_{i}\left(s\right)\mathrm{d}\Omega\mathrm{d}\overline{\Gamma}\end{aligned} \tag{3.12}$$

The FE discretization of the enrichment terms is:

$$\mathbf{u}^{n+1}\left(\mathbf{x},s\right)=\mathbf{u}^{n}\left(\mathbf{x},s\right)+\mathbf{N}\left(\mathbf{x}\right)\underline{\mathbf{R}}\mathbf{M}\left(s\right)\underline{\mathbf{S}} \tag{3.13}$$

where $\underline{\mathbf{R}}$ and $\underline{\mathbf{S}}$ are the nodal values of the functions $\mathbf{R}(\mathbf{x})$ and $S(s)$ relative to the enrichment step. Take into account that the operator $\mathbf{L}$ is applied to the spatial coordinates, the first derivative of the solution at iteration $n+1$ is like:

$$\begin{aligned}\mathbf{L}\mathbf{u}^{n+1} &= \mathbf{L}\mathbf{u}^{n} + \mathbf{L}\mathbf{R}\left(\mathbf{x}\right) S\left(s\right) \\ &= \sum_{i=1}^{n} \mathbf{L}\mathbf{N}\left(\mathbf{x}\right) \underline{\mathbf{F}}_i \mathbf{M}\left(s\right) \underline{\mathbf{G}}_i + \mathbf{L}\mathbf{N}\left(\mathbf{x}\right) \underline{\mathbf{R}}\mathbf{M}\left(s\right) \underline{\mathbf{S}} \\ &= \sum_{i=1}^{n} \mathbf{B}\left(\mathbf{x}\right) \underline{\mathbf{F}}_i \mathbf{M}\left(s\right) \underline{\mathbf{G}}_i + \mathbf{B}\left(\mathbf{x}\right) \underline{\mathbf{R}}\mathbf{M}\left(s\right) \underline{\mathbf{S}}\end{aligned} \tag{3.14}$$

And in a similar way for the virtual displacement field:

$$\begin{aligned}\mathbf{L}\delta\mathbf{u} &= \mathbf{L}\delta\mathbf{R}\left(\mathbf{x}\right) S\left(s\right) + \mathbf{L}\mathbf{R}\left(\mathbf{x}\right) \delta S\left(s\right) \\ &= \mathbf{L}\mathbf{N}\left(\mathbf{x}\right) \delta\underline{\mathbf{R}}\mathbf{M}\left(s\right) \underline{\mathbf{S}} + \mathbf{L}\mathbf{N}\left(\mathbf{x}\right) \underline{\mathbf{R}}\mathbf{M}\left(s\right) \delta\underline{\mathbf{S}} \\ &= \mathbf{B}\left(\mathbf{x}\right) \delta\underline{\mathbf{R}}\mathbf{M}\left(s\right) \underline{\mathbf{S}} + \mathbf{B}\left(\mathbf{x}\right) \underline{\mathbf{R}}\mathbf{M}\left(s\right) \delta\underline{\mathbf{S}}\end{aligned} \tag{3.15}$$

The extra parameter of the problem, the load $\bar{\mathbf{t}}$, can be discretized with the following equation by introducing the shape functions respectively for the spatial coordinate and the load position:

$$\bar{\mathbf{t}} = \sum_{j=1}^{m} \mathbf{N}\left(\mathbf{x}\right) \underline{\mathbf{f}}_j \mathbf{M}\left(s\right) \underline{\mathbf{g}}_j \tag{3.16}$$

Introducing the discretizations and taking into account equations (3.13), (3.14), (3.15) and (3.16), equation (3.12) can be written as:

$$\begin{aligned}&\int_{\Omega\times\overline{\Gamma}} \left[\mathbf{L}\mathbf{N}\left(\mathbf{x}\right) \delta\underline{\mathbf{R}}\mathbf{M}\left(s\right) \underline{\mathbf{S}}\right]^{\mathrm{T}} \mathbf{D}\mathbf{L}\mathbf{N}\left(\mathbf{x}\right) \underline{\mathbf{R}}\mathbf{M}\left(s\right) \underline{\mathbf{S}} \mathrm{d}\Omega\mathrm{d}\overline{\Gamma} \\ &= \int_{\Gamma_t\times\overline{\Gamma}} \left[\mathbf{N}\left(\mathbf{x}\right) \delta\underline{\mathbf{R}}\mathbf{M}\left(s\right) \underline{\mathbf{S}}\right]^{\mathrm{T}} \left[\sum_{j=1}^{m} \mathbf{N}\left(\mathbf{x}\right) \underline{\mathbf{f}}_j \mathbf{M}\left(s\right) \underline{\mathbf{g}}_j\right] \mathrm{d}\Gamma\mathrm{d}\overline{\Gamma} \\ &- \int_{\Omega\times\overline{\Gamma}} \left[\mathbf{L}\mathbf{N}\left(\mathbf{x}\right) \delta\underline{\mathbf{R}}\mathbf{M}\left(s\right) \underline{\mathbf{S}}\right]^{\mathrm{T}} \mathbf{D} \sum_{i=1}^{n} \mathbf{L}\mathbf{N}\left(\mathbf{x}\right) \underline{\mathbf{F}}_i \mathbf{M}\left(s\right) \underline{\mathbf{G}}_i \mathrm{d}\Omega\mathrm{d}\overline{\Gamma}\end{aligned} \tag{3.17}$$

Substituting the strain-displacement matrix $\mathbf{B}\left(\mathbf{x}\right) = \mathbf{L}\mathbf{N}(\mathbf{x})$ the previous equation becomes:

$$\begin{aligned}&\int_{\Omega\times\overline{\Gamma}} \left[\mathbf{B}\left(\mathbf{x}\right) \delta\underline{\mathbf{R}}\mathbf{M}\left(s\right) \underline{\mathbf{S}}\right]^{\mathrm{T}} \mathbf{D}\mathbf{B}\left(\mathbf{x}\right) \underline{\mathbf{R}}\mathbf{M}\left(s\right) \underline{\mathbf{S}} \mathrm{d}\Omega\mathrm{d}\overline{\Gamma} \\ &= \int_{\Gamma_t\times\overline{\Gamma}} \left[\mathbf{N}\left(\mathbf{x}\right) \delta\underline{\mathbf{R}}\mathbf{M}\left(s\right) \underline{\mathbf{S}}\right]^{\mathrm{T}} \left[\sum_{j=1}^{m} \mathbf{N}\left(\mathbf{x}\right) \underline{\mathbf{f}}_j \mathbf{M}\left(s\right) \underline{\mathbf{g}}_j\right] \mathrm{d}\Gamma\mathrm{d}\overline{\Gamma} \\ &- \int_{\Omega\times\overline{\Gamma}} \left[\mathbf{B}\left(\mathbf{x}\right) \delta\underline{\mathbf{R}}\mathbf{M}\left(s\right) \underline{\mathbf{S}}\right]^{\mathrm{T}} \mathbf{D} \sum_{i=1}^{n} \mathbf{B}\left(\mathbf{x}\right) \underline{\mathbf{F}}_i \mathbf{M}\left(s\right) \underline{\mathbf{G}}_i \mathrm{d}\Omega\mathrm{d}\overline{\Gamma}\end{aligned} \tag{3.18}$$

The spirit of the PGD method is to integrate each function on its proper domain, so that equation (3.18) becomes:

$$\begin{aligned}
&\mathbf{S}^{\mathrm{T}}\left[\int_{\overline{\Gamma}}\mathbf{M}^{\mathrm{T}}(s)\,\mathbf{M}(s)\,\mathrm{d}\overline{\Gamma}\right]\underline{\mathbf{S}}\delta\mathbf{R}^{\mathrm{T}}\left[\int_{\Omega}\mathbf{B}^{\mathrm{T}}(\mathbf{x})\,\mathbf{DB}(\mathbf{x})\,\mathrm{d}\Omega\right]\underline{\mathbf{R}}\\
&=\delta\underline{\mathbf{R}}^{\mathrm{T}}\sum_{j=1}^{m}\left[\int_{\Gamma_t}\mathbf{N}^{\mathrm{T}}(\mathbf{x})\,\mathbf{N}(\mathbf{x})\,\mathrm{d}\Gamma\right]\underline{\mathbf{f}}_j\underline{\mathbf{S}}^{\mathrm{T}}\left[\int_{\overline{\Gamma}}\mathbf{M}^{\mathrm{T}}(s)\,\mathbf{M}(s)\,\mathrm{d}\overline{\Gamma}\right]\underline{\mathbf{g}}_j\\
&-\delta\underline{\mathbf{R}}^{\mathrm{T}}\sum_{i=1}^{n}\left[\int_{\Omega}\mathbf{B}^{\mathrm{T}}(\mathbf{x})\,\mathbf{DB}(\mathbf{x})\,\mathrm{d}\Omega\right]\underline{\mathbf{F}}_i\underline{\mathbf{S}}^{\mathrm{T}}\left[\int_{\overline{\Gamma}}\mathbf{M}^{\mathrm{T}}(s)\,\mathbf{M}(s)\,\mathrm{d}\Gamma\right]\underline{\mathbf{G}}_i
\end{aligned}\tag{3.19}$$

The previous equation must hold for any $\delta\mathbf{R}$:

$$\begin{aligned}
&\underline{\mathbf{S}}^{\mathrm{T}}\left[\int_{\overline{\Gamma}}\mathbf{M}^{\mathrm{T}}(s)\,\mathbf{M}(s)\,\mathrm{d}\overline{\Gamma}\right]\underline{\mathbf{S}}\left[\int_{\Omega}\mathbf{B}^{\mathrm{T}}(\mathbf{x})\,\mathbf{DB}(\mathbf{x})\,\mathrm{d}\Omega\right]\underline{\mathbf{R}}\\
&=\sum_{j=1}^{m}\left[\int_{\Gamma_t}\mathbf{N}^{\mathrm{T}}(\mathbf{x})\,\mathbf{N}(\mathbf{x})\,\mathrm{d}\Gamma\right]\underline{\mathbf{f}}_j\underline{\mathbf{S}}^{\mathrm{T}}\left[\int_{\overline{\Gamma}}\mathbf{M}^{\mathrm{T}}(s)\,\mathbf{M}(s)\,\mathrm{d}\overline{\Gamma}\right]\underline{\mathbf{g}}_j\\
&-\sum_{i=1}^{n}\left[\int_{\Omega}\mathbf{B}^{\mathrm{T}}(\mathbf{x})\,\mathbf{DB}(\mathbf{x})\,\mathrm{d}\Omega\right]\underline{\mathbf{F}}_i\underline{\mathbf{S}}^{\mathrm{T}}\left[\int_{\overline{\Gamma}}\mathbf{M}^{\mathrm{T}}(s)\,\mathbf{M}(s)\,\mathrm{d}\Gamma\right]\underline{\mathbf{G}}_i
\end{aligned}\tag{3.20}$$

An equivalent expression of equation (3.20) is reproduced below:

$$\begin{aligned}
\underline{\mathbf{S}}^{\mathrm{T}}\mathbf{M}_2(s)\,\underline{\mathbf{S}}\mathbf{K}(\mathbf{x})\,\underline{\mathbf{R}} &= \sum_{j=1}^{m}\mathbf{N}_2(\mathbf{x})\,\underline{\mathbf{f}}_j\underline{\mathbf{S}}^{\mathrm{T}}\mathbf{M}_2(s)\,\underline{\mathbf{g}}_j\\
&-\sum_{i=1}^{n}\mathbf{K}(\mathbf{x})\,\underline{\mathbf{F}}_i\underline{\mathbf{S}}^{\mathrm{T}}\mathbf{M}_2(s)\,\underline{\mathbf{G}}_i
\end{aligned}\tag{3.21}$$

Equation (3.21) shows an interesting similarity with the classical FE formulation $\mathbf{KU}=\mathbf{P}$. In this case, the LHS shows a new stiffness matrix obtained from the classical stiffness matrix $\mathbf{K}$, using FEM, multiplied by a scalar value $\underline{\mathbf{S}}^{\mathrm{T}}\mathbf{M}_2(s)\,\underline{\mathbf{S}}$. The RHS shows a new external load vector, which is different from the vector $\mathbf{P}$ in the FEM, since it is obtained from two addends. In particular the second addend consists of previously computed known terms of the PGD approximation. Note that the PGD method applied to this particular parametric problem involves in integrals depending on just one variable. For that the gain of less computational cost is obtained.

3.2.2 Computation of $S(s)$ assuming $\mathbf{R}(\mathbf{x})$ is known

Taking into account of equations (3.4), (3.5) and (3.10), the weak form of the problem (equation (3.2)) becomes:

$$\int_{\Omega\times\overline{\Gamma}} \left[\mathbf{LR}(\mathbf{x})\,\delta S(s)\right]^{\mathrm{T}} : \mathbf{D} : \left[\sum_{i=1}^{n} \mathbf{LF}_i(\mathbf{x})\,G_i(s) + \mathbf{LR}(\mathbf{x})\,S(s)\right] \mathrm{d}\Omega\mathrm{d}\overline{\Gamma}$$
$$= \int_{\Gamma_t\times\overline{\Gamma}} \left[\mathbf{R}(\mathbf{x})\,\delta S(s)\right]^{\mathrm{T}} \left[\sum_{j=1}^{m} \mathbf{f}_j(\mathbf{x})\,g_j(s)\right] \mathrm{d}\Gamma\mathrm{d}\overline{\Gamma} \tag{3.22}$$

Moving all known terms to the RHS, equation (3.22) becomes:

$$\int_{\Omega\times\overline{\Gamma}} \left[\mathbf{LR}(\mathbf{x})\,\delta S(s)\right]^{\mathrm{T}} : \mathbf{D} : \mathbf{LR}(\mathbf{x})\,S(s)\,\mathrm{d}\Omega\mathrm{d}\overline{\Gamma} = \int_{\Gamma_t\times\overline{\Gamma}} \left[\mathbf{R}(\mathbf{x})\delta S(s)\right]^{\mathrm{T}} \left[\sum_{j=1}^{n} \mathbf{f}_j(\mathbf{x})\,g_j(s)\right] \mathrm{d}\Gamma\mathrm{d}\overline{\Gamma}$$
$$- \int_{\Omega\times\overline{\Gamma}} \left[\mathbf{LR}(\mathbf{x})\,\delta S(s)\right]^{\mathrm{T}} : \mathbf{D} : \sum_{i=1}^{n} \mathbf{LF}_i(\mathbf{x})\,G_i(s)\,\mathrm{d}\Omega\mathrm{d}\overline{\Gamma} \tag{3.23}$$

Taking into account of equations (3.13), (3.14), (3.15) and (3.16), equation (3.23) becomes:

$$\int_{\Omega\times\overline{\Gamma}} \left[\mathbf{LN}(\mathbf{x})\,\underline{\mathbf{R}}\mathbf{M}(s)\,\delta\underline{\mathbf{S}}\right]^{\mathrm{T}} \mathbf{DLN}(\mathbf{x})\,\underline{\mathbf{R}}\mathbf{M}(s)\,\underline{\mathbf{S}}\mathrm{d}\Omega\mathrm{d}\overline{\Gamma}$$
$$= \int_{\Gamma_t\times\overline{\Gamma}} \left[\mathbf{N}(\mathbf{x})\,\underline{\mathbf{R}}\mathbf{M}(s)\,\delta\underline{\mathbf{S}}\right]^{\mathrm{T}} \left[\sum_{j=1}^{m} \mathbf{N}(\mathbf{x})\,\underline{\mathbf{f}}_j\mathbf{M}(s)\,\underline{\mathbf{g}}_j\right] \mathrm{d}\Gamma\mathrm{d}\overline{\Gamma}$$
$$- \int_{\Omega\times\overline{\Gamma}} \left[\mathbf{LN}(\mathbf{x})\,\underline{\mathbf{R}}\mathbf{M}(s)\,\delta\underline{\mathbf{S}}\right]^{\mathrm{T}} \mathbf{D} \sum_{i=1}^{n} \mathbf{LN}(\mathbf{x})\,\underline{\mathbf{F}}_i\mathbf{M}(s)\,\underline{\mathbf{G}}_i\mathrm{d}\Omega\mathrm{d}\overline{\Gamma} \tag{3.24}$$

Introducing the strain-displacement matrix $\mathbf{B}(\mathbf{x})$, equation (3.24) becomes:

$$\int_{\Omega\times\overline{\Gamma}} \left[\mathbf{B}(\mathbf{x})\,\underline{\mathbf{R}}\mathbf{M}(s)\,\delta\underline{\mathbf{S}}\right]^{\mathrm{T}} \mathbf{DB}(\mathbf{x})\,\underline{\mathbf{R}}\mathbf{M}(s)\,\underline{\mathbf{S}}\mathrm{d}\Omega\mathrm{d}\overline{\Gamma}$$
$$= \int_{\Gamma_t\times\overline{\Gamma}} \left[\mathbf{N}(\mathbf{x})\,\underline{\mathbf{R}}\mathbf{M}(s)\,\delta\underline{\mathbf{S}}\right]^{\mathrm{T}} \left[\sum_{j=1}^{m} \mathbf{N}(\mathbf{x})\,\underline{\mathbf{f}}_j\mathbf{M}(s)\,\underline{\mathbf{g}}_j\right] \mathrm{d}\Gamma\mathrm{d}\overline{\Gamma}$$
$$- \int_{\Omega\times\overline{\Gamma}} \left[\mathbf{B}(\mathbf{x})\,\underline{\mathbf{R}}\mathbf{M}(s)\,\delta\underline{\mathbf{S}}\right]^{\mathrm{T}} \mathbf{D} \sum_{i=1}^{n} \mathbf{B}(\mathbf{x})\,\underline{\mathbf{F}}_i\mathbf{M}(s)\,\underline{\mathbf{G}}_i\mathrm{d}\Omega\mathrm{d}\overline{\Gamma} \tag{3.25}$$

Splitting the integrals in equation (3.25) results in:

$$\begin{aligned}
&\underline{\mathbf{R}}^{\mathrm{T}}\left[\int_{\Omega}\mathbf{B}^{\mathrm{T}}(\mathbf{x})\,\mathbf{D}\mathbf{B}(\mathbf{x})\,\mathrm{d}\Omega\right]\underline{\mathbf{R}}\delta\underline{\mathbf{S}}^{\mathrm{T}}\left[\int_{\overline{\Gamma}}\mathbf{M}^{\mathrm{T}}(s)\,\mathbf{M}(s)\,\mathrm{d}\overline{\Gamma}\right]\underline{\mathbf{S}}\\
&=\sum_{j=1}^{m}\delta\underline{\mathbf{S}}^{\mathrm{T}}\left[\int_{\overline{\Gamma}}\mathbf{M}^{\mathrm{T}}(s)\,\mathbf{M}(s)\,\mathrm{d}\overline{\Gamma}\right]\underline{\mathbf{g}}_j\underline{\mathbf{R}}^{\mathrm{T}}\left[\int_{\Gamma_t}\mathbf{N}^{\mathrm{T}}(\mathbf{x})\,\mathbf{N}(\mathbf{x})\,\mathrm{d}\Gamma\right]\underline{\mathbf{f}}_j\\
&-\sum_{i=1}^{n}\underline{\mathbf{R}}^{\mathrm{T}}\left[\int_{\Omega}\mathbf{B}^{\mathrm{T}}(\mathbf{x})\,\mathbf{D}\mathbf{B}(\mathbf{x})\,\mathrm{d}\Omega\right]\underline{\mathbf{F}}_i\delta\underline{\mathbf{S}}^{\mathrm{T}}\left[\int_{\overline{\Gamma}}\mathbf{M}^{\mathrm{T}}(s)\,\mathbf{M}(s)\,\mathrm{d}\Gamma\right]\underline{\mathbf{G}}_i
\end{aligned}\tag{3.26}$$

The previous equation must hold for any $\delta\mathbf{S}$, so that we can write:

$$\begin{aligned}
&\underline{\mathbf{R}}^{\mathrm{T}}\left[\int_{\Omega}\mathbf{B}^{\mathrm{T}}(\mathbf{x})\,\mathbf{D}\mathbf{B}(\mathbf{x})\,\mathrm{d}\Omega\right]\underline{\mathbf{R}}\left[\int_{\overline{\Gamma}}\mathbf{M}^{\mathrm{T}}(s)\,\mathbf{M}(s)\,\mathrm{d}\overline{\Gamma}\right]\underline{\mathbf{S}}\\
&=\sum_{j=1}^{m}\left[\int_{\overline{\Gamma}}\mathbf{M}^{\mathrm{T}}(s)\,\mathbf{M}(s)\,\mathrm{d}\overline{\Gamma}\right]\underline{\mathbf{g}}_j\underline{\mathbf{R}}^{\mathrm{T}}\left[\int_{\Gamma_t}\mathbf{N}^{\mathrm{T}}(\mathbf{x})\,\mathbf{N}(\mathbf{x})\,\mathrm{d}\Gamma\right]\underline{\mathbf{f}}_j\\
&-\sum_{i=1}^{n}\underline{\mathbf{R}}^{\mathrm{T}}\left[\int_{\Omega}\mathbf{B}^{\mathrm{T}}(\mathbf{x})\,\mathbf{D}\mathbf{B}(\mathbf{x})\,\mathrm{d}\Omega\right]\underline{\mathbf{F}}_i\left[\int_{\overline{\Gamma}}\mathbf{M}^{\mathrm{T}}(s)\,\mathbf{M}(s)\,\mathrm{d}\Gamma\right]\underline{\mathbf{G}}_i
\end{aligned}\tag{3.27}$$

Equation (3.27) can be rewritten in a short form like:

$$\begin{aligned}
\underline{\mathbf{R}}^{\mathrm{T}}\mathbf{K}(\mathbf{x})\,\underline{\mathbf{R}}\mathbf{M}_2(s)\,\underline{\mathbf{S}}&=\sum_{j=1}^{m}\mathbf{M}_2(s)\,\underline{\mathbf{g}}_j\underline{\mathbf{R}}^{\mathrm{T}}\mathbf{N}_2(\mathbf{x})\,\underline{\mathbf{f}}_j\\
&-\sum_{i=1}^{n}\underline{\mathbf{R}}^{\mathrm{T}}\mathbf{K}(\mathbf{x})\,\underline{\mathbf{F}}_i\cdot\mathbf{M}_2(s)\,\underline{\mathbf{G}}_i
\end{aligned}\tag{3.28}$$

The previous equation needs some considerations. In the LHS the classic stiffness matrix in the FEM $\mathbf{K}$, with size $n_{dof}\times n_{dof}$, is obtained. Then, we have to take into account of another integral over $\overline{\Gamma}$ of the shape functions for the extra coordinate called $\mathbf{M}_2$ of size $n_{dofs}\times n_{dofs}$. The RHS has a similar expression of the LHS plus a second addend takes into account of the source term in separated form.
The matrix $\mathbf{N_2}(\mathbf{x})\,\underline{\mathbf{f}}_j$ has been built by considering a matrix $n_{dof}\times n_{dofs}$ in which the no-vanishing entries are the corresponding degrees of freedom where the load can be applied and their values are equal to the force magnitude. Note that in the Matlab code shown in Appendix A, this matrix is called directly $\mathbf{f}$.

3.3 Numerical integration: Gaussian quadrature rules

In order to perform the numerical integrations, the Gaussian quadrature rule has been adopted. In particular, for the elements defining the load position domain,

a linear integration scheme is used with two Gauss points $s_{g,i}$. Let considere a general domain of integration $[a, b]$. The domain of integration for such a rule is conventionally taken as $[-1, +1]$. Thus the interval of integration change in the following way:

$$\int_a^b f(x)\,\mathrm{dx} = \frac{b-a}{2}\int_{-1}^{1} f\left(\frac{b-a}{2}x + \frac{a+b}{2}\right)\mathrm{dx} \tag{3.29}$$

Applying the Gaussian quadrature rule the integral involves in the following approximation:

$$\int_a^b f(x)\,\mathrm{dx} \approx \frac{b-a}{2}\sum_{i=1}^{n} w_g f\left(\frac{b-a}{2}s_g + \frac{a+b}{2}\right) \tag{3.30}$$

where the Gauss points s_g and the weights w_g are:

$$s_g = [-0.57735, 0.57735] \tag{3.31}$$
$$w_g = [1, 1] \tag{3.32}$$

3.4 Geometry

The cantilever beam, with heigth 1 m and lenght 3 m, is clamped at its left edge. The finite element model has been built using GMSH is a free 3D finite element generator with a built-in CAD engine and post-processor. Its goal is to provide a parametric output which nodes and elements [7].

(a) (b)

Figure 3.1: (a) Linear triangular element, (b) Shape function N_1^e of a triangular element.

The beam has been discretized using linear triangular elements. Figure 3.1 shows an element and one of its shape functions. Each element has three nodes and each node has two degrees of freedom, u_{xi} in x direction and u_{yi} in y direction with $i = 1, 2, 3$. The shape functions used to discretize the solution are linear shape functions and they are described using the Cartesian coordinates, an example of the first shape function N_1^e is given in figure (3.1,b).
Figure (3.2) shows the discretized beam with 435 nodes, 788 elements and 848 degrees of freedom.

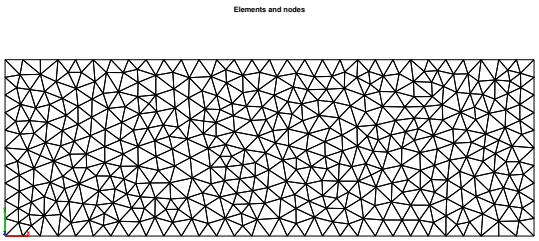


FIGURE 3.2: FE mesh of the beam.

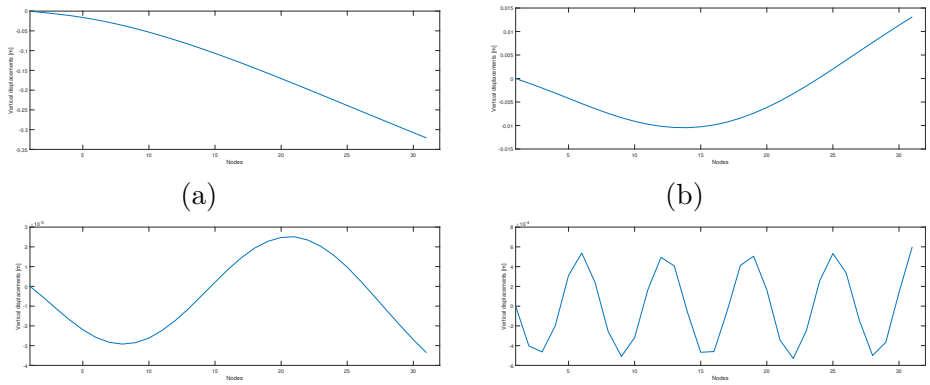
3.5 Results

In this section, the results for both the off-line and on-line phase are presented. First the outputs, the CPU time and the PGD convergence will be discussed. Second the diplacements for varying load positions and comparisons with the FE solution will be discussed.

3.5.1 Off-line phase results

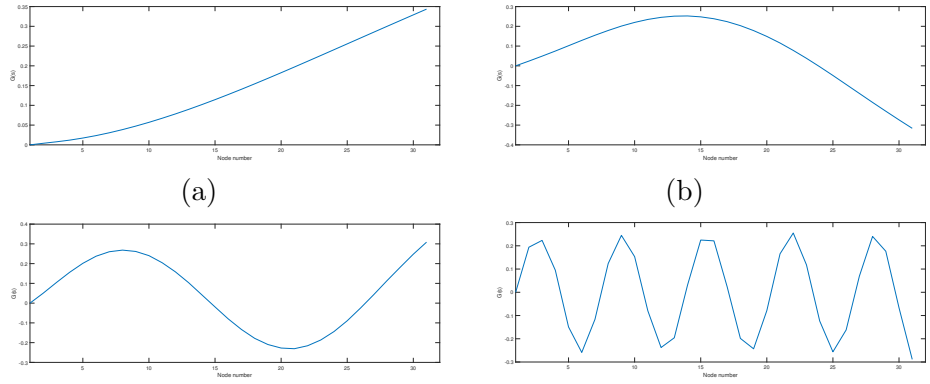
Once the *off-line phase* code has been executed, the vademecum is obtained. The load is applied at the vertical degrees of fredoom of the beam. The *off-line phase* code is reproduced in Appendix A.
The PGD solution is given by equation (3.5). So that the PGD outputs in order to obtain the meta model are two matrices. The first one is the matrix $\mathbf{F}$ that includes all the spatial modes. The size of $\mathbf{F}$ is $n_{dof} \times n_{iter}$. The second one is the matrix $\mathbf{G}$ that includes all the load modes. The size of $\mathbf{G}$ is $n_{dofs} \times n_{iter}$.
Figures 3.3 and 3.4 display some mechanical and load modes. In particular $\mathbf{F}_i$ and $\mathbf{G}_i$ with $i = 1, 2, 3, 10$.
Table 3.1 shows the CPU times for the PGD method and the FE method using StaBIL. The CPU time for the PGD method is referred to the time to find the solution for one particular extra coordinate. The CPU time for the FE method has been evaluated on the time to get the solutions $\mathbf{u}$ for any possible load positions. The offline phase, for the PGD method, takes a bigger time than the online phase. The interesting fact is that the CPU time for the FE implementation takes a bigger time than the time requested for the PGD implementation. For that, the PGD method demostrates its powerful, since the CPU time is reduced and in less time the user is able to obtain several responses of the model. The laptop used is an ASUS F552C with Processor Intel Core i7 3537U, memory of 4.0 GB and HDD 500 GB.

The load position problem has been studied considering an initialized vector $\mathbf{S_0}$ as a random vector and as a vector with constant values. Considering $\mathbf{S_0}$ as a random vector the number of iterations to reach the convergence is around 33-36 iterations while considering a unit vector the number of iteration for the convergence is 31.



(a) (b) (c) (d)

Figure 3.3: Spatial modes: (a) Spatial mode $\mathbf{F}_1$, (b) Spatial mode $\mathbf{F}_2$, (c) Spatial mode $\mathbf{F}_3$, (d) Spatial mode $\mathbf{F}_{10}$.



(a) (b) (c) (d)

Figure 3.4: Load modes: (a) Load mode $\mathbf{G}_1$, (b) Load mode $\mathbf{G}_2$, (c) Load mode $\mathbf{G}_3$, (d) Load mode $\mathbf{G}_{10}$.

Table 3.1: CPU time: PGD method VS FE method

PGD CPU time	**FE CPU time**
Offline 1.52 s	-
Online 0.0003 s	0.04 s

The stopping criteria used for the enrichment step is defined by equation (1.2). Figure 3.5 shows that it gives very good results in the load position problem since the convergence is reached quickly.

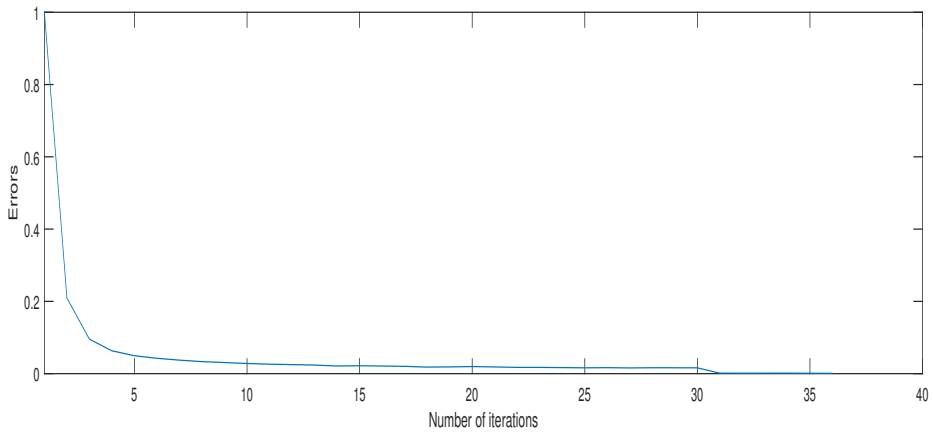


FIGURE 3.5: Relative errors for varying iterations.

3.5.2 On-line phase results

Once the vademecum is obtained, the user in the *on-line phase* code is able to choose the desidered position of the load clicking in any point of the beam surface. Interactively, the user can play with the load position and see in real time the beam deflection under a unit vertical load. The *on-line phase* code is reproduced in Appendix A.
Figure 3.6 shows the two points on the top of the beam chosen by the user. The chosen points are node 68 near the clamped end and node 46 near the free end. The displacement has been evaluated in the node 2 at the bottom right corner.
The two deformed shapes releated to the two chosen points are represented in figure 3.7, the red one represents the beam configuration when the load is applied in node 68 while the blue one is the beam configuration when the load is applied in node 46.
Table 3.2 gives the vertical displacements in node 2 of two loaded points using the PGD method and the FE method. These displacements for the PGD method are obtained at convergence of the procedure. The table shows that the vertical

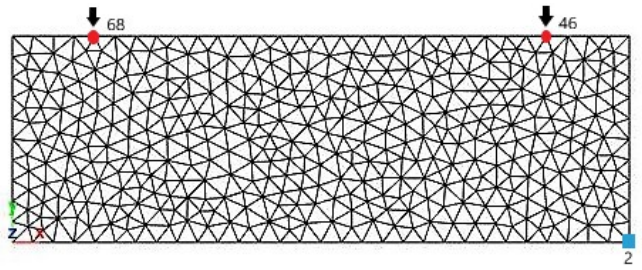


FIGURE 3.6: Example of two loaded points and evalutaion of the displacement in the node 2.

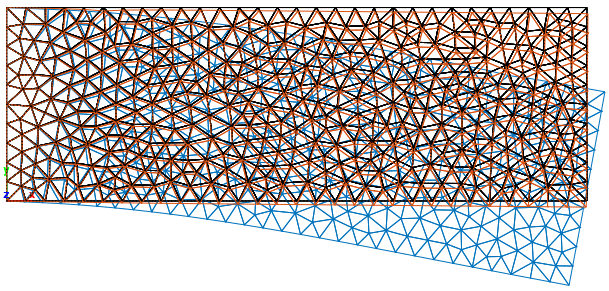


Figure 3.7: Deformed shapes due to the unit load. Note that the scaling is equal for both deformations.

displacements in node 46 match in both methods. While there is a little difference between the vertical displacements in node 68 and so far from the point where the load is applied.

Table 3.2: Displacement of node 2 due the load applied in nodes 68 and 46.

	Displacements [m]			
	PGD method		**FE method**	
	Load position			
Node	*Node 68*	*Node 46*	*Node 68*	*Node 46*
2	-0.0058	-0.0927	-0.0061	-0.0927

Interesting considerations about the accuracy of the displacements can be done considering the unit load applied at the *node 3*. The displacements are evaluated for nodes 3 and 71 for varying iterations. Figure 3.8 shows the FE model considering the loaded node (red indicator) and the two chosen nodes (blue) for the evaluation of the displacements.

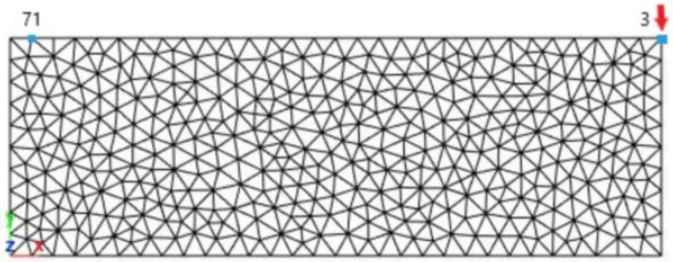


Figure 3.8: Beam loaded in node 3. Displacement evaluated in node 3 and 71.

Figure 3.8 shows some particular points chosen to check the validity of the PGD method for varying iterations. For that, table 3.3 shows other interesting results. As

expected, the accuracy of the displacements depends on the number of iterations. In particular considering the load applied at the free end, we need more iterations to get a better approximation of the displacement near the free end and so where the load is applied and not near the clamped end. Indeed the solution by FEM is reached at iteration 5 for the node 71 and at iteration 30 for the node 3. Obviously similar considerations are valid if the load is applied in node 71. In that case, we need 30 iterations in node 71 to get a better displacements accuracy, while in node 3, the FE solution is reached at iteration 5. The table shows only the vertical components of the displacements in m.

TABLE 3.3: Accuracy displacements: PGD method VS FE method

	Displacements PGD		**Displacements FEM**	
n. max iteration	*Node3*	*Node 71*	*Node 3*	*Node 71*
3	-0.1145	-0.00095	-0.1193	-0.0011
5	-0.1161	-0.00098		
10	-0.1177	-0.0011		
20	-0.1188	-0.0011		
25	-0.1191	-0.0011		
30	-0.1193	-0.0011		
50	-0.1193	-0.0011		

Chapter

4 Material parameter as extra problem dimension

4.1 Introduction

In this chapter the proper generalized decomposition method is applied to another solid mechanics problem where the Poisson's ratio is assumed as an extra parameter of the problem. The Poisson's ratio domain is assumed linear and discretized using 1D elements. After implementation the user is able to choose any Poisson's value digiting it on the keyboard and obtain in real time the deformed shape of the beam considering the load acting on the top right corner (node 3). The Young's modulus is assumed with a value of $E = 1000$ Mpa.

4.2 The Poisson's ratio as extra parameter of the problem

For this particular problem, the weak form of a linear elastic problem in continuum solid mechanics is represented by the following formulation:

$$\int_{\Omega \times I_\nu} (\mathbf{L}\delta\mathbf{u})^{\mathrm{T}} : \mathbf{D} : \mathbf{L}\mathbf{u} \mathrm{d}\Omega \mathrm{d}I_\nu = \int_{\Gamma_t \times I_\nu} \delta\mathbf{u}^{\mathrm{T}} \bar{\mathbf{t}} \mathrm{d}\Gamma \mathrm{d}I_\nu \tag{4.1}$$

where Ω is the domain of the problem to be solved, subjected to Dirichlet boundary contidions on a region Γ_u and Neumann boundary conditions with surface tractions $\bar{\mathbf{t}}$ on Γ_t, $\mathbf{D}$ the elastic constitutive matrix, ρ the material density and $\delta\mathbf{u}$ is any virtual dispacement of the displacement field $\mathbf{u}$ that is compatible with the prescribed boundary consitions.
Under the basic assumptions of the PGD method, assuming the Poisson's ratio as extra dimension in the problem, the solution of equation (4.1) depends on the spatial variable $\mathbf{x}$ and Poisson's ratio ν:

$$\mathbf{u}(\mathbf{x}, \nu) = \sum_{i=1}^{n} \mathbf{F}_i(\mathbf{x}) G_i(\nu) \tag{4.2}$$

Now the problem is defined in the domain $\Omega \times I_\nu$. Where I_ν is the range of values for the Poisson's ratio $I_\nu \in [0; 0.5[$.

If the rank-n approximation does not give the desired accuracy, we look for the $(n+1)$-th term like:

$$\mathbf{u}^{n+1}(\mathbf{x},\nu) = \mathbf{u}^{n}(\mathbf{x},\nu) + \mathbf{R}(\mathbf{x})\,P(\nu) \tag{4.3}$$

where $\mathbf{R}(\mathbf{x})$ and $P(\nu)$ are the enrichment functions that improve the approximation. The *virtual displacement field* $\delta\mathbf{u}$ is expressed with the following formulation:

$$\delta\mathbf{u}(\mathbf{x},\nu) = \delta\mathbf{R}(\mathbf{x})\,P(\nu) + \mathbf{R}(\mathbf{x})\,\delta P(\nu) \tag{4.4}$$

The PGD method, applied to this particular problem, involves in two main steps where the enrichment functions $\mathbf{R}(\mathbf{x})$ and $P(\nu)$ are determined iteratively with a fixed-point algorithm:

1. The computation of $\mathbf{R}(\mathbf{x})$ assuming $P(\nu)$ is known
   In this case the virtual displacement field becomes:

$$\delta\mathbf{u}(\mathbf{x},\nu) = \delta\mathbf{R}(\mathbf{x})P(\nu) \tag{4.5}$$

2. The computation of $P(\nu)$ assuming $\mathbf{R}(\mathbf{x})$ is known
   In this case the virtual displacement field becomes:

$$\delta\mathbf{u}(\mathbf{x},\nu) = \mathbf{R}(\mathbf{x})\delta P(\nu) \tag{4.6}$$

These two steps should be repeated until convergence. The fixed algorithm point has been used as stopping criterion. The iterative procedure stops when the norm of the solution at iteration n and $n-1$ is less than the TOL. Equation (1.2) shows the stopping criterion formula.

4.2.1 Computation of $\mathbf{R}(\mathbf{x})$ assuming $P(\nu)$ is known

Using equations (4.2) and (4.5), the weak form of the problem (equation (4.1)) can be written as:

$$\begin{aligned}&\int_{\Omega\times I_\nu} \left[\mathbf{L}\delta\mathbf{R}(\mathbf{x})\,P(\nu)\right]^{\mathrm{T}} : \mathbf{D}(\nu) : \sum_{i=1}^{n} \mathbf{L}\mathbf{F}_i(\mathbf{x})\,G_i(\nu) + \mathbf{L}\mathbf{R}(\mathbf{x})\,P(\nu)\,\mathrm{d}\Omega\mathrm{d}I_\nu \\ &= \int_{\Gamma_t\times I_\nu} \left[\delta\mathbf{R}(\mathbf{x})\,P(\nu)\right]^{\mathrm{T}} \bar{\mathbf{t}}\,\mathrm{d}\Gamma\mathrm{d}I_\nu\end{aligned} \tag{4.7}$$

where operator $\mathbf{L}$ only affects functions of the spatial coordinates $\mathbf{x}$ and $\mathbf{D}$ is the constitutive matrix.
Since the contribution of the first n modes are already available at this step, the

known terms are moved to the RHS and equation (4.7) becomes:

$$\begin{aligned}
&\int_{\Omega\times I_\nu} \left[\mathbf{L}\delta\mathbf{R}\left(\mathbf{x}\right)P\left(\nu\right)\right]^{\mathrm{T}} : \mathbf{D} : \mathbf{L}\mathbf{R}\left(\mathbf{x}\right)P\left(\nu\right)\mathrm{d}\Omega\mathrm{d}I_\nu \\
&= \int_{\Gamma\times I_\nu} \left[\delta\mathbf{R}\left(\mathbf{x}\right)P\left(\nu\right)\right]^{\mathrm{T}}\bar{\mathbf{t}}\mathrm{d}\Gamma\mathrm{d}I_\nu \\
&- \int_{\Omega\times I_\nu} \left[\mathbf{L}\delta\mathbf{R}\left(\mathbf{x}\right)P\left(\nu\right)\right]^{\mathrm{T}} : \mathbf{D} : \sum_{i=1}^{n}\mathbf{L}\mathbf{F}_i\left(\mathbf{x}\right)G_i\left(\nu\right)\mathrm{d}\Omega\mathrm{d}I_\nu
\end{aligned} \tag{4.8}$$

The FE discretization of the PGD solution is:

$$\mathbf{u}^n\left(\mathbf{x},\nu\right) = \sum_{i=1}^{n}\mathbf{N}\left(\mathbf{x}\right)\underline{\mathbf{F}}_i\mathbf{M}\left(\nu\right)\underline{\mathbf{G}}_i \tag{4.9}$$

where $\underline{\mathbf{F}}_i$ and $\underline{\mathbf{G}}_i$ are the nodal values of the fuctions $\mathbf{F}_i\left(\mathbf{x}\right)$ and $G_i\left(\nu\right)$ while $\mathbf{N}\left(\mathbf{x}\right)$ and $\mathbf{M}\left(\nu\right)$ are the shape functions respectively of the spatial coordinates $\mathbf{x}$ and Poisson's ratio ν.
Instead the FE discretization for the enrichment terms is:

$$\mathbf{u}^{n+1}\left(\mathbf{x},\nu\right) = \mathbf{u}^n\left(\mathbf{x},\nu\right) + \mathbf{N}\left(\mathbf{x}\right)\underline{\mathbf{R}}\mathbf{M}\left(\nu\right)\underline{\mathbf{P}} \tag{4.10}$$

Take into account that the operator $\mathbf{L}$ only affects the spatial coordinates, the displacement field at iteration $n+1$ is like:

$$\begin{aligned}
\mathbf{L}\mathbf{u}^{n+1} &= \mathbf{L}\mathbf{u}^n + \mathbf{L}\mathbf{R}\left(\mathbf{x}\right)P\left(\nu\right) \\
&= \sum_{i=1}^{n}\mathbf{L}\mathbf{N}\left(\mathbf{x}\right)\underline{\mathbf{F}}_i\mathbf{M}\left(\nu\right)\underline{\mathbf{G}}_i + \mathbf{L}\mathbf{N}\left(\mathbf{x}\right)\underline{\mathbf{R}}\mathbf{M}\left(\nu\right)\underline{\mathbf{P}} \\
&= \sum_{i=1}^{n}\mathbf{B}\left(\mathbf{x}\right)\underline{\mathbf{F}}_i\mathbf{M}\left(\nu\right)\underline{\mathbf{G}}_i + \mathbf{B}\left(\mathbf{x}\right)\underline{\mathbf{R}}\mathbf{M}\left(\nu\right)\underline{\mathbf{P}}
\end{aligned} \tag{4.11}$$

In a similar way:

$$\begin{aligned}
\mathbf{L}\delta\mathbf{u} &= \mathbf{L}\delta\mathbf{R}\left(\mathbf{x}\right)P\left(\nu\right) + \mathbf{L}\mathbf{R}\left(\mathbf{x}\right)\delta P\left(\nu\right) \\
&= \mathbf{L}\mathbf{N}\left(\mathbf{x}\right)\delta\underline{\mathbf{R}}\mathbf{M}\left(\nu\right)\underline{\mathbf{P}} + \mathbf{L}\mathbf{N}\left(\mathbf{x}\right)\underline{\mathbf{R}}\mathbf{M}\left(\nu\right)\delta\underline{\mathbf{P}} \\
&= \mathbf{B}\left(\mathbf{x}\right)\delta\underline{\mathbf{R}}\mathbf{M}\left(\nu\right)\underline{\mathbf{P}} + \mathbf{B}\left(\mathbf{x}\right)\underline{\mathbf{R}}\mathbf{M}\left(\nu\right)\delta\underline{\mathbf{P}}
\end{aligned} \tag{4.12}$$

The matrix structure of the problem is obtained considering equations (4.9), (4.10), (4.11) and (4.12), so that equation (4.8) can be written as:

$$\begin{aligned}
&\int_{\Omega\times I_\nu} \left[\mathbf{L}\mathbf{N}\left(\mathbf{x}\right)\delta\underline{\mathbf{R}}\mathbf{M}\left(\nu\right)\underline{\mathbf{P}}\right]^{\mathrm{T}}\mathbf{D}\left(\nu\right)\mathbf{L}\mathbf{N}\left(\mathbf{x}\right)\underline{\mathbf{R}}\mathbf{M}\left(\nu\right)\underline{\mathbf{P}}\mathrm{d}\Omega\mathrm{d}I_\nu \\
&= \int_{\Gamma_t\times I_\nu} \left[\mathbf{N}\left(\mathbf{x}\right)\delta\underline{\mathbf{R}}\mathbf{M}\left(\nu\right)\underline{\mathbf{P}}\right]^{\mathrm{T}}\bar{\mathbf{t}}\mathrm{d}\Gamma\mathrm{d}I_\nu \\
&- \int_{\Omega\times I_\nu} \left[\mathbf{L}\mathbf{N}\left(\mathbf{x}\right)\delta\underline{\mathbf{R}}\mathbf{M}\left(\nu\right)\underline{\mathbf{P}}\right]^{\mathrm{T}}\mathbf{D}\left(\nu\right)\sum_{i=1}^{n}\mathbf{L}\mathbf{N}\left(\mathbf{x}\right)\underline{\mathbf{F}}_i\mathbf{M}\left(\nu\right)\underline{\mathbf{G}}_i\mathrm{d}\Omega\mathrm{d}I_\nu
\end{aligned} \tag{4.13}$$

Substituing the strain-displacement matrix $\mathbf{B}(\mathbf{x}) = \mathbf{LN}(\mathbf{x})$ the previous equation becomes:

$$\begin{aligned}
&\int_{\Omega\times I_\nu} \left[\mathbf{B}\left(\mathbf{x}\right)\delta\underline{\mathbf{R}}\mathbf{M}\left(\nu\right)\underline{\mathbf{P}}\right]^{\mathrm{T}}\mathbf{D}\left(\nu\right)\mathbf{B}\left(\mathbf{x}\right)\underline{\mathbf{R}}\mathbf{M}\left(\nu\right)\underline{\mathbf{P}}\mathrm{d}\Omega\mathrm{d}I_\nu \\
&= \int_{\Gamma_t\times I_\nu} \left[\mathbf{N}\left(\mathbf{x}\right)\delta\underline{\mathbf{R}}\mathbf{M}\left(\nu\right)\underline{\mathbf{P}}\right]^{\mathrm{T}}\bar{\mathbf{t}}\mathrm{d}\Gamma\mathrm{d}I_\nu \\
&- \int_{\Omega\times I_\nu} \left[\mathbf{B}\left(\mathbf{x}\right)\delta\underline{\mathbf{R}}\mathbf{M}\left(\nu\right)\underline{\mathbf{P}}\right]^{\mathrm{T}}\mathbf{D}\left(\nu\right)\sum_{i=1}^{n}\mathbf{B}\left(\mathbf{x}\right)\underline{\mathbf{F}}_i\mathbf{M}\left(\nu\right)\underline{\mathbf{G}}_i\mathrm{d}\Omega\mathrm{d}I_\nu
\end{aligned} \tag{4.14}$$

In the spirit of the PGD method, each function should be integrated over its proper domain. In this particular PGD application, the function depending on ν should be integrated over I_ν and the functions depending on the spatial variable $\mathbf{x}$ over Ω. Keeping in mind this, equation (4.14) becomes:

$$\begin{aligned}
&\delta\underline{\mathbf{R}}^{\mathrm{T}}\int_{\Omega}\mathbf{B}^{\mathrm{T}}\left(\mathbf{x}\right)\underline{\mathbf{P}}^{\mathrm{T}}\left[\int_{I_\nu}\mathbf{M}^{\mathrm{T}}\left(\nu\right)\mathbf{D}\left(\nu\right)\mathbf{M}\left(\nu\right)\mathrm{d}I_\nu\right]\underline{\mathbf{P}}\mathbf{B}\left(\mathbf{x}\right)\mathrm{d}\Omega\underline{\mathbf{R}} \\
&= \delta\underline{\mathbf{R}}^{\mathrm{T}}\left[\underline{\mathbf{P}}^{\mathrm{T}}\int_{I_\nu}\mathbf{M}^{\mathrm{T}}\left(\nu\right)\mathrm{d}I_\nu\right]\left[\int_{\Gamma_t}\mathbf{N}^{\mathrm{T}}\left(\mathbf{x}\right)\bar{\mathbf{t}}\mathrm{d}\Gamma\right] \\
&- \delta\underline{\mathbf{R}}^{\mathrm{T}}\sum_{i=1}^{n}\int_{\Omega}\mathbf{B}^{\mathrm{T}}\left(\mathbf{x}\right)\left[\int_{I_\nu}\underline{\mathbf{P}}^{\mathrm{T}}\mathbf{M}^{\mathrm{T}}\left(\nu\right)\mathbf{D}\left(\nu\right)\mathbf{M}\left(\nu\right)\underline{\mathbf{G}}_i\mathrm{d}I_\nu\right]\mathbf{B}\left(\mathbf{x}\right)\underline{\mathbf{F}}_i\mathrm{d}\Omega
\end{aligned} \tag{4.15}$$

Equation (4.15) must hold for any $\delta\underline{\mathbf{R}}$:

$$\begin{aligned}
&\int_{\Omega}\mathbf{B}^{\mathrm{T}}\left(\mathbf{x}\right)\left[\int_{I_\nu}\underline{\mathbf{P}}^{\mathrm{T}}\mathbf{M}^{\mathrm{T}}\left(\nu\right)\mathbf{D}\left(\nu\right)\mathbf{M}\left(\nu\right)\underline{\mathbf{P}}\mathrm{d}I_\nu\right]\mathbf{B}\left(\mathbf{x}\right)\mathrm{d}\Omega\underline{\mathbf{R}} \\
&= \underline{\mathbf{P}}^{\mathrm{T}}\left[\int_{I_\nu}\mathbf{M}^{\mathrm{T}}\left(\nu\right)\mathrm{d}I_\nu\right]\left[\int_{\Gamma_t}\mathbf{N}^{\mathrm{T}}\left(\mathbf{x}\right)\bar{\mathbf{t}}\mathrm{d}\Gamma\right] \\
&- \sum_{i=1}^{n}\int_{\Omega}\mathbf{B}^{\mathrm{T}}\left(\mathbf{x}\right)\left[\int_{I_\nu}\underline{\mathbf{P}}^{\mathrm{T}}\mathbf{M}^{\mathrm{T}}\left(\nu\right)\mathbf{D}\left(\nu\right)\mathbf{M}\left(\nu\right)\underline{\mathbf{G}}_i\mathrm{d}I_\nu\right]\mathbf{B}\left(\mathbf{x}\right)\underline{\mathbf{F}}_i\mathrm{d}\Omega
\end{aligned} \tag{4.16}$$

Equation (4.16) has a structure very similar to the one of the FE method. Indeed the expression is similar to the FE equation $\mathbf{KU} = \mathbf{P}$ but in this case the stiffness matrix $\mathbf{K}$ is computed from a new and modified constitutive matrix $\mathbf{D}'$ obtained pre-multiplying and post-multiplying the original constitutive matrix $\mathbf{D}$ by the scalar values $\mathbf{P}^{\mathrm{T}}\mathbf{M}^{\mathrm{T}}(\nu)$ and $\mathbf{M}(\nu)\mathbf{P}$ and integrating over I_ν. The RHS has two terms. The first one is derived from the numerical integration of the shape function $\mathbf{M}(\nu)$ over

I_ν and the external load vector evaluated with the function of StaBil. The second one has an expression similar to the LHS where $\underline{\mathbf{F}}_i$ and $\underline{\mathbf{G}}_i$ are determined at the step $n-1$. Equation (4.16) permits to solve the spatial vector $\underline{\mathbf{R}}$.

4.2.2 Computation of $P(\nu)$ assuming $\mathbf{R}(\mathbf{x})$ is known

Substituting equations (4.2) and (4.6) the weak form (equation (4.1)) becomes:

$$\begin{aligned}&\int_{\Omega\times I_\nu} \left[\mathbf{LR}\left(\mathbf{x}\right)\delta P\left(\nu\right)\right]^{\mathrm{T}} : \mathbf{D}\left(\nu\right) : \sum_{i=1}^{n} \mathbf{LF}_i\left(\mathbf{x}\right) G_i\left(\nu\right) + \mathbf{LR}\left(\mathbf{x}\right) P\left(\nu\right) \mathrm{d}\Omega \mathrm{d}I_\nu \\ &= \int_{\Gamma\times I_\nu} \left[\mathbf{R}(\mathbf{x})\delta P\left(\nu\right)\right]^{\mathrm{T}} \bar{\mathbf{t}} \mathrm{d}\Gamma \mathrm{d}I_\nu \end{aligned} \tag{4.17}$$

Moving all known terms to the RHS:

$$\begin{aligned}&\int_{\Omega\times I_\nu} \left[\mathbf{LR}\left(\mathbf{x}\right)\delta P\left(\nu\right)\right]^{\mathrm{T}} : \mathbf{D}\left(\nu\right) : \mathbf{LR}\left(\mathbf{x}\right) P\left(\nu\right) \mathrm{d}\Omega \mathrm{d}I_\nu \\ &= \int_{\Gamma_t\times I_\nu} \left[\mathbf{R}\left(\mathbf{x}\right)\delta P\left(\nu\right)\right]^{\mathrm{T}} \bar{\mathbf{t}} \mathrm{d}\Gamma \mathrm{d}I_\nu \\ &- \int_{\Omega\times I_\nu} \left[\mathbf{LR}\left(\mathbf{x}\right)\delta P\left(\nu\right)\right]^{\mathrm{T}} : \mathbf{D}\left(\nu\right) : \sum_{i=1}^{n} \mathbf{LF}_i\left(\mathbf{x}\right) G_i\left(\nu\right) \mathrm{d}\Omega \mathrm{d}I_\nu \end{aligned} \tag{4.18}$$

The matrix structure of the problem is obtained considering equations (4.9), (4.10), (4.11) and (4.12), so that equation (4.18) becomes:

$$\begin{aligned}&\int_{\Omega\times I_\nu} \left[\mathbf{LN}\left(\mathbf{x}\right)\underline{\mathbf{R}}\mathbf{M}\left(\nu\right)\delta\underline{\mathbf{P}}\right]^{\mathrm{T}} \mathbf{D}\left(\nu\right) \mathbf{LN}\left(\mathbf{x}\right)\underline{\mathbf{R}}\mathbf{M}\left(\nu\right)\underline{\mathbf{P}} \mathrm{d}\Omega \mathrm{d}I_\nu \\ &= \int_{\Gamma_t\times I_\nu} \left[\mathbf{N}\left(\mathbf{x}\right)\underline{\mathbf{R}}\mathbf{M}\left(\nu\right)\delta\underline{\mathbf{P}}\right]^{\mathrm{T}} \bar{\mathbf{t}} \mathrm{d}\Gamma \mathrm{d}I_\nu \\ &- \int_{\Omega\times I_\nu} \left[\mathbf{LN}\left(\mathbf{x}\right)\underline{\mathbf{R}}\mathbf{M}\left(\nu\right)\delta\underline{\mathbf{P}}\right]^{\mathrm{T}} \mathbf{D}\left(\nu\right) \sum_{i=1}^{n} \mathbf{LN}\left(\mathbf{x}\right)\underline{\mathbf{F}}_i\mathbf{M}\left(\nu\right)\underline{\mathbf{G}}_i \mathrm{d}\Omega \mathrm{d}I_\nu \end{aligned} \tag{4.19}$$

Introducing the strain-displacement matrix $\mathbf{B}(\mathbf{x})$ that includes the first derivative of the shape function $\mathbf{N}(\mathbf{x})$, the previous equation becomes:

$$\begin{aligned}&\int_{\Omega\times I_\nu} \left[\mathbf{B}\left(\mathbf{x}\right)\underline{\mathbf{R}}\mathbf{M}\left(\nu\right)\delta\underline{\mathbf{P}}\right]^{\mathrm{T}} \mathbf{D}\left(\nu\right) \mathbf{B}\left(\mathbf{x}\right)\underline{\mathbf{R}}\mathbf{M}\left(\nu\right)\underline{\mathbf{P}} \mathrm{d}\Omega \mathrm{d}I_\nu \\ &= \int_{\Gamma_t\times I_\nu} \left[\mathbf{N}\left(\mathbf{x}\right)\underline{\mathbf{R}}\mathbf{M}\left(\nu\right)\delta\underline{\mathbf{P}}\right]^{\mathrm{T}} \bar{\mathbf{t}} \mathrm{d}\Gamma \mathrm{d}I_\nu \\ &- \int_{\Omega\times I_\nu} \left[\mathbf{B}\left(\mathbf{x}\right)\underline{\mathbf{R}}\mathbf{M}\left(\nu\right)\delta\underline{\mathbf{P}}\right]^{\mathrm{T}} \mathbf{D}\left(\nu\right) \sum_{i=1}^{n} \mathbf{B}\left(\mathbf{x}\right)\underline{\mathbf{F}}_i\mathbf{M}\left(\nu\right)\underline{\mathbf{G}}_i \mathrm{d}\Omega \mathrm{d}I_\nu \end{aligned} \tag{4.20}$$

Splitting the integrals, equation (4.20) becomes:

$$\begin{aligned}
&\delta\underline{\mathbf{P}}^{\mathrm{T}} \int_{I_\nu} \mathbf{M}^{\mathrm{T}}(\nu)\,\underline{\mathbf{R}}^{\mathrm{T}} \left[\int_{\Omega} \mathbf{B}^{\mathrm{T}}(\mathbf{x})\,\mathbf{D}(\nu)\,\mathbf{B}(\mathbf{x})\,\mathrm{d}\Omega\right] \underline{\mathbf{R}}\mathbf{M}(\nu)\,\mathrm{d}I_\nu\underline{\mathbf{P}} \\
&= \delta\underline{\mathbf{P}}^{\mathrm{T}} \left[\int_{I_\nu} \mathbf{M}^{\mathrm{T}}(\nu)\,\mathrm{d}I_\nu\right] \underline{\mathbf{R}}^{\mathrm{T}} \left[\int_{\Gamma_t} \mathbf{N}^{\mathrm{T}}(\mathbf{x})\,\bar{\mathbf{t}}\mathrm{d}\Gamma\right] \\
&- \delta\underline{\mathbf{P}}^{\mathrm{T}} \sum_{i=1}^{n} \int_{I_\nu} \mathbf{M}^{\mathrm{T}}(\nu)\,\underline{\mathbf{R}}^{\mathrm{T}} \left[\int_{\Omega} \mathbf{B}^{\mathrm{T}}(\mathbf{x})\,\mathbf{D}(\nu)\,\mathbf{B}(\mathbf{x})\,\mathrm{d}\Omega\right] \underline{\mathbf{F}}_i\mathbf{M}(\nu)\,\underline{\mathbf{G}}_i\mathrm{d}I_\nu
\end{aligned} \tag{4.21}$$

Equation (4.21) must hold for any $\delta\underline{\mathbf{P}}$:

$$\begin{aligned}
&\int_{I_\nu} \mathbf{M}^{\mathrm{T}}(\nu)\,\underline{\mathbf{R}}^{\mathrm{T}} \left[\int_{\Omega} \mathbf{B}^{\mathrm{T}}(\mathbf{x})\,\mathbf{D}(\nu)\,\mathbf{B}(\mathbf{x})\,\mathrm{d}\Omega\right] \underline{\mathbf{R}}\mathbf{M}(\nu)\,\mathrm{d}I_\nu\underline{\mathbf{P}} \\
&= \left[\int_{I_\nu} \mathbf{M}^{\mathrm{T}}(\nu)\,\mathrm{d}I_\nu\right] \underline{\mathbf{R}}^{\mathrm{T}} \left[\int_{\Gamma_t} \mathbf{N}^{\mathrm{T}}(\mathbf{x})\,\bar{\mathbf{t}}\mathrm{d}\Gamma\right] \\
&- \sum_{i=1}^{n} \int_{I_\nu} \mathbf{M}^{\mathrm{T}}(\nu)\,\underline{\mathbf{R}}^{\mathrm{T}} \left[\int_{\Omega} \mathbf{B}^{\mathrm{T}}(\mathbf{x})\,\mathbf{D}(\nu)\,\mathbf{B}(\mathbf{x})\,\mathrm{d}\Omega\right] \underline{\mathbf{F}}_i\mathbf{M}(\nu)\,\underline{\mathbf{G}}_i\mathrm{d}I_\nu
\end{aligned} \tag{4.22}$$

In a more elegant way, the previous equation can be written like:

$$\begin{aligned}
&\int_{I_\nu} \mathbf{M}^{\mathrm{T}}(\nu)\,\underline{\mathbf{R}}^{\mathrm{T}}\mathbf{K}(\nu)\,\underline{\mathbf{R}}\mathbf{M}(\nu)\,\mathrm{d}I_\nu\underline{\mathbf{P}} \\
&= \left[\int_{I_\nu} \mathbf{M}^{\mathrm{T}}(\nu)\,\mathrm{d}I_\nu\right] \underline{\mathbf{R}}^{\mathrm{T}} \left[\int_{\Gamma_t} \mathbf{N}^{\mathrm{T}}(\mathbf{x})\,\bar{\mathbf{t}}\mathrm{d}\Gamma\right] \\
&- \sum_{i=1}^{n} \int_{I_\nu} \mathbf{M}^{\mathrm{T}}(\nu)\,\underline{\mathbf{R}}^{\mathrm{T}}\mathbf{K}(\nu)\,\underline{\mathbf{F}}_i\mathbf{M}(\nu)\,\underline{\mathbf{G}}_i\mathrm{d}I_\nu
\end{aligned} \tag{4.23}$$

where the stiffness matrix $\mathbf{K}(\nu)$ still depends on the Poisson's ratio and for that, it should be integrated numerically over the domain I_ν. The RHS has two terms: the first one derives from the numerical integration of the shape function $\mathbf{M}(\nu)$ over I_ν and the external load vector evaluated with the function of StaBIL and the second one has an expression similar to the LHS, with $\underline{\mathbf{F}}_i$ and $\underline{\mathbf{G}}_i$ determined at the step $n-1$. Note that these vectors are initialized as null vectors at the beginning and determined iteratively step by step until convergence of the method. Finally, equation (4.23) permits to solve the Poisson's vector $\underline{\mathbf{P}}$. Note that the integral formulations for the Poisson's ratio problem are no longer separated as was the case for the load position problem.

4.3 Implentation in Matlab: Gaussian quadrature rules

The implementation in Matlab has been performed with the Gaussian quadrature rule. For more details refer to section 3.3.
In this section, the implentation in Matlab is presented. In particular the two procedures adopted to perform the numerical integrations of equations (4.16) and (4.22). The first intuitive procedure of integration leads to an extra approximation of the numerical result since it uses a linear interpolations to evaluate functions at Gauss points. The second procedure follows an interesting way to approach the implementation. In particular the solution is given by separeted integrals where each integral depends just on one variable. In order to do this, the constitutive matrix $\mathbf{D}(\nu)$ has been divided in a symmetric and anti-symmetric part [18]. Note that the only difference between the two approaches is the computation of the LHS and the second term of the RHS. Indeed the first term of the RHS is equal for both approaches.

4.3.1 First approach of implementation

The first approach used, leads to the implementation of the two steps governing the Poisson's problem with a bigger CPU time. Basically, it is due to the principle in which the numerical integrations of equations (4.16) and (4.23) are performed.

Computation of $\mathbf{R}(\mathbf{x})$ assuming $P(\nu)$ is known

Equation (4.16) should be implemented in order to find the spatial vector $\underline{\mathbf{R}}$.
The LHS of equation (4.16) leads to the computation of the stiffness matrix $\mathbf{K}$ that comes from a new and modified constitutive matrix $\mathbf{D}'$. The new constitutive matrix is given by pre-multiplication and post-multiplication of each term $\mathbf{D}_{ij}$ by a scalar values obtained from $\underline{\mathbf{P}}^{\mathrm{T}}\mathbf{M}^{\mathrm{T}}(\nu)$ and $\mathbf{M}(\nu)\,\underline{\mathbf{P}}$. So, its numerical integration, term by term, involves in the following integration:

$$\int_{I_\nu} \underline{\mathbf{P}}^{\mathrm{T}}\mathbf{M}^{\mathrm{T}}(\nu)\,\mathbf{D}_{ij}(\nu)\,\mathbf{M}(\nu)\,\underline{\mathbf{P}}\mathrm{d}I_\nu \tag{4.24}$$

where $\mathbf{D}_{ij}$ is the constitutive matrix element with $i = 1:3$ and $j = 1:3$.
In this spirit, the RHS of equation (4.16) has been performed as well. In this case the first scalar value is given by $\underline{\mathbf{P}}^{\mathrm{T}}\mathbf{M}^{\mathrm{T}}(\nu)$ and second by $\mathbf{M}(\nu)\,\underline{\mathbf{G}}_i$. The first addend of the RHS is given by an integral of the shape function $\mathbf{M}(\nu)$ over I_ν and the external load vector already available in StaBIL. In Appendix B, the Matlab functions *ke.plane3.m* and *ke.plane32.m* permit to perform the described stiffness matrix at element level.

Computation of $P(\nu)$ assuming $\mathbf{R}(\mathbf{x})$ is known

In order to find the Poisson's vector $\underline{\mathbf{P}}$, equation (4.23) should be numerically integrated. The LHS of equation (4.23) shows that the stiffness matrix $\mathbf{K}(\nu)$ still

depends on the Poisson's ratio. For that it should be integrated over the domain I_ν. As first attempt, the stiffnness matrix $\mathbf{K}(\nu)$ has been evaluated for every Poisson's ratio. This operation leads to a 3D matrix, whose entries are $n_{dof\nu}$ stiffness matrices. After that, the term $\underline{\mathbf{R}}^{\mathrm{T}}\mathbf{K}(\nu)\underline{\mathbf{R}}$ leads to a function of ν. It is described in figure 4.1, where the spatial vector $\underline{\mathbf{R}}$ is referred to the first iteration.

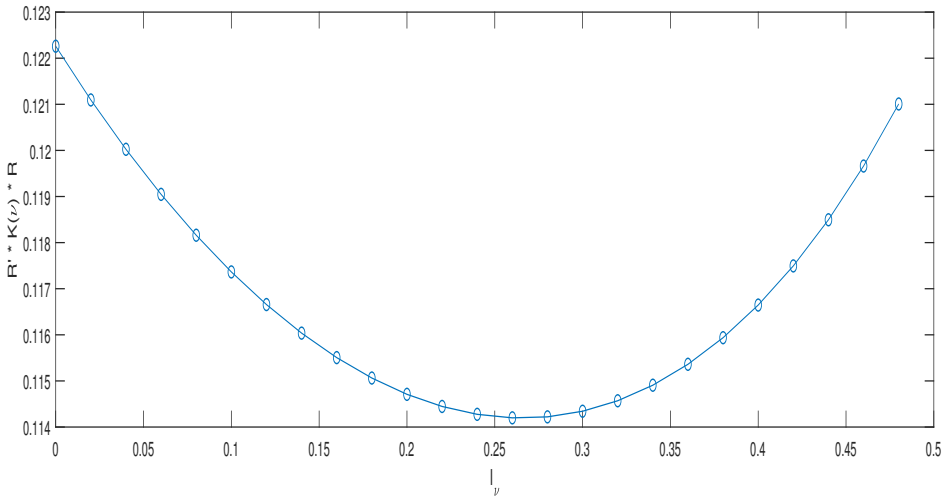


Figure 4.1: Function $\underline{\mathbf{R}}^{\mathrm{T}}\mathbf{K}(\nu)\underline{\mathbf{R}}$

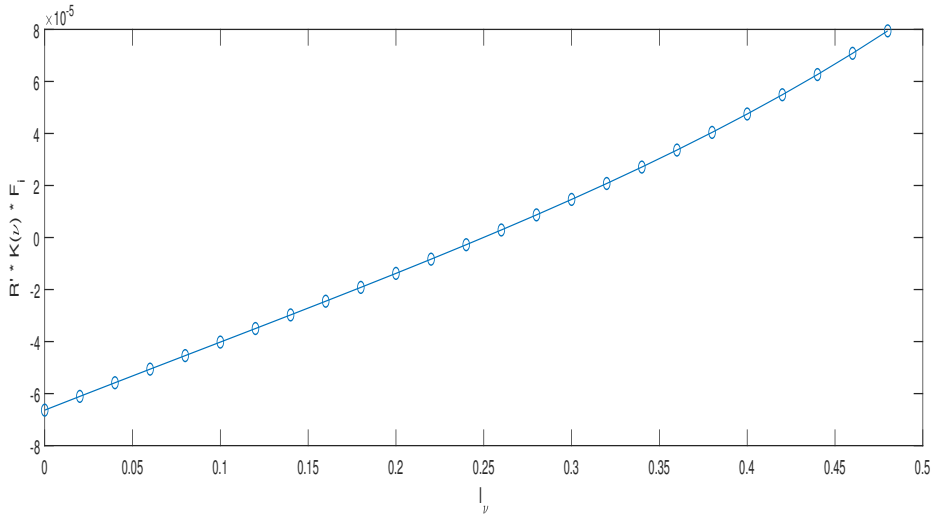


Figure 4.2: Function $\underline{\mathbf{R}}^{\mathrm{T}}\mathbf{K}(\nu)\underline{\mathbf{F}}_1$

However the Gaussian quadrature rule leads to the evaluation of a function in the Gauss points. For that, the function reproduced in figure 4.1, should be interpolated in the Gauss points. A linear interpolation has been adopted. After interpolation, the integral of the shape functions $\mathbf{M}(\nu)$ over I_ν is easy to compute.
In a similar way, the RHS of equation (4.23) has been approached. In this case, the function obtained from the 3D stiffness matrix $\mathbf{K}(\nu)$ is represented in figure 4.2. In Appendix B, the Matlab functions used to perform the numerical integration with the first approach are reproduced.

4.3.2 Second approach of implementation

The second approach to implement the PGD problem for the Poisson's ratio is more efficient and accurated than the first approach of integration. It respects the basic idea of the PGD method that the integrals over each domain should be separated. For that the computational cost is of the order of seconds instead of minutes.

Computation of $\mathbf{R}(\mathbf{x})$ assuming $P(\nu)$ is known

First the LHS of equation (4.16) should be integrated. It is a double integral where the inner integral leads to a new constitutive matrix $\mathbf{D}'$ while the outer integral is over Ω and it leads to the stiffness matrix using a particular constitutive matrix.
In order to perfom this integral, the original constitutive matrix $\mathbf{D}$ is divided in two new constitutive matrices $\mathbf{D}_1$ and $\mathbf{D}_2$, so that:

$$\mathbf{D} = \frac{E}{2(1-\nu)}\mathbf{D}_1 + \frac{E}{2(1+\nu)}\mathbf{D}_2 \tag{4.25}$$

where E is the Young's modulus and ν the Poisson's ratio. These two new matrices $\mathbf{D}_1$ and $\mathbf{D}_2$ are written like:

$$\mathbf{D}_1 = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{4.26}$$

$$\mathbf{D}_2 = \begin{bmatrix} 1 & -1 & 0 \\ -1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.27}$$

In this way the two constitutive matrices $\mathbf{D}_1$ and $\mathbf{D}_2$ do not depend more on the Poisson's ratio ν. Taking into account of this separated constitutive matrix, the stiffness matrix $\mathbf{K}$ can be derived automatically and it equals:

$$\mathbf{K} = \int_\Omega \mathbf{B}(\mathbf{x})^{\mathrm{T}}\mathbf{D}(\nu)\mathbf{B}(\mathbf{x})\mathrm{d}\Omega = \frac{E}{2(1-\nu)}\mathbf{K}_1 + \frac{E}{2(1+\nu)}\mathbf{K}_2 \tag{4.28}$$

where:

$$\mathbf{K}_1 = \int_\Omega \mathbf{B}(\mathbf{x})^{\mathrm{T}}\mathbf{D}_1\mathbf{B}(\mathbf{x})\mathrm{d}\Omega \tag{4.29}$$

$$\mathbf{K}_2 = \int_\Omega \mathbf{B}(\mathbf{x})^{\mathrm{T}}\mathbf{D}_2\mathbf{B}(\mathbf{x})\mathrm{d}\Omega \tag{4.30}$$

Under these considerations, equation (4.16) can be solved. In particular the inner integral over I_ν, that is the new constitutive matrix $\mathbf{D}'$, becomes like:

$$\begin{aligned}
&\int_{I_\nu} \underline{\mathbf{P}}^{\mathrm{T}}\mathbf{M}^{\mathrm{T}}(\nu)\,\mathbf{D}(\nu)\,\mathbf{M}(\nu)\,\underline{\mathbf{P}}\mathrm{d}I_\nu \\
&= \mathbf{D}_1 \int_{I_\nu} \frac{E}{2(1-\nu)}\underline{\mathbf{P}}^{\mathrm{T}}\mathbf{M}^{\mathrm{T}}(\nu)\,\mathbf{M}(\nu)\,\underline{\mathbf{P}}\mathrm{d}I_\nu + \mathbf{D}_2 \int_{I_\nu} \frac{E}{2(1+\nu)}\underline{\mathbf{P}}^{\mathrm{T}}\mathbf{M}^{\mathrm{T}}(\nu)\,\mathbf{M}(\nu)\,\underline{\mathbf{P}}\mathrm{d}I_\nu \\
&= \mathbf{D}_1\nu_1^* + \mathbf{D}_2\nu_2^* \qquad (4.31)
\end{aligned}$$

where the scalar values ν_1^* and ν_2^* are defined from the integral over I_ν and they are:

$$\nu_1^* = \int_{I_\nu} \frac{E}{2(1-\nu)}\underline{\mathbf{P}}^{\mathrm{T}}\mathbf{M}^{\mathrm{T}}(\nu)\,\mathbf{M}(\nu)\,\underline{\mathbf{P}}\mathrm{d}I_\nu \qquad (4.32)$$

$$\nu_2^* = \int_{I_\nu} \frac{E}{2(1+\nu)}\underline{\mathbf{P}}^{\mathrm{T}}\mathbf{M}^{\mathrm{T}}(\nu)\,\mathbf{M}(\nu)\,\underline{\mathbf{P}}\mathrm{d}I_\nu \qquad (4.33)$$

Introducing the new resulting constitutive matrix (4.31) into the LHS of equation (4.16), the global stiffness matrix $\mathbf{K}$ is written like:

$$\begin{aligned}
&\int_\Omega \mathbf{B}^{\mathrm{T}}(\mathbf{x}) \left[\int_{I_\nu} \underline{\mathbf{P}}^{\mathrm{T}}\mathbf{M}^{\mathrm{T}}(\nu)\,\mathbf{D}(\nu)\,\mathbf{M}(\nu)\,\underline{\mathbf{P}}\mathrm{d}I_\nu\right]\mathbf{B}(\mathbf{x})\,\mathrm{d}\Omega \\
&= \nu_1^*\int_\Omega \mathbf{B}(\mathbf{x})^{\mathrm{T}}\mathbf{D}_1\mathbf{B}(\mathbf{x}) + \nu_2^*\int_\Omega \mathbf{B}(\mathbf{x})^{\mathrm{T}}\mathbf{D}_2\mathbf{B}(\mathbf{x}) \\
&= \nu_1^*\mathbf{K}_1 + \nu_2^*\mathbf{K}_2 \qquad (4.34)
\end{aligned}$$

In a similar way the numerical integration of the RHS of equation (4.16) has been solved. In particular the second term looks like:

$$\begin{aligned}
&\sum_{i=1}^{n}\int_\Omega \mathbf{B}^{\mathrm{T}}(\mathbf{x}) \left[\int_{I_\nu} \underline{\mathbf{P}}^{\mathrm{T}}\mathbf{M}^{\mathrm{T}}(\nu)\,\mathbf{D}(\nu)\,\mathbf{M}(\nu)\,\underline{\mathbf{G}}_i\mathrm{d}I_\nu\right]\mathbf{B}(\mathbf{x})\,\underline{\mathbf{F}}_i\mathrm{d}\Omega \\
&= \sum_{i=1}^{n}\left[\nu_{1,i}^*\mathbf{K}_1 + \nu_{2,i}^*\mathbf{K}_2\right]\underline{\mathbf{F}}_i \qquad (4.35)
\end{aligned}$$

where, in this case, the scalar values $\nu_{1,i}^*$ and $\nu_{2,i}^*$ are defined like:

$$\nu_{1,i}^* = \int_{I_\nu} \frac{E}{2(1-\nu)}\underline{\mathbf{P}}^{\mathrm{T}}\mathbf{M}^{\mathrm{T}}(\nu)\,\mathbf{M}(\nu)\,\underline{\mathbf{G}}_i\mathrm{d}I_\nu \qquad (4.36)$$

$$\nu_{2,i}^* = \int_{I_\nu} \frac{E}{2(1+\nu)}\underline{\mathbf{P}}^{\mathrm{T}}\mathbf{M}^{\mathrm{T}}(\nu)\,\mathbf{M}(\nu)\,\underline{\mathbf{G}}_i\mathrm{d}I_\nu \qquad (4.37)$$

Note that with this interesting approach, the numerical integrations involve in integrals depending on just one variable. In particular the stiffness matrices $\mathbf{K}_1$ and $\mathbf{K}_2$ can be computed out from the enrichment step since they do not change during the simulation. Indeed, it leads to a further reduced computational cost.

Computation of $P(\nu)$ assuming $\mathbf{R}(\mathbf{x})$ is known

The computation of $P(\nu)$ has been perfomed taking into account the same considerations regard the divided constitutive matrix $\mathbf{D}(\nu)$. In particular the LHS of equation (4.23) shows that the stiffness matrix $\mathbf{K}(\nu)$ depends on the Poisson's ratio ν, so that it can be written in a separated form like:

$$\mathbf{K}(\nu) = \int_{\Omega} \mathbf{B}(\mathbf{x})^{\mathrm{T}} \mathbf{D}(\nu) \mathbf{B}(\mathbf{x}) \mathrm{d}\Omega = \frac{E}{2(1-\nu)} \mathbf{K}_1 + \frac{E}{2(1+\nu)} \mathbf{K}_2 \tag{4.38}$$

Considering equation (4.38), the LHS of equation (4.23) becomes:

$$\int_{I_\nu} \mathbf{M}^{\mathrm{T}}(\nu) \underline{\mathbf{R}}^{\mathrm{T}} \frac{E}{2(1-\nu)} \mathbf{K}_1 \underline{\mathbf{R}} \mathbf{M}(\nu) \, \mathrm{d}I_\nu + \int_{I_\nu} \mathbf{M}^{\mathrm{T}}(\nu) \underline{\mathbf{R}}^{\mathrm{T}} \frac{E}{2(1+\nu)} \mathbf{K}_2 \underline{\mathbf{R}} \mathbf{M}(\nu) \, \mathrm{d}I_\nu \tag{4.39}$$

Including into integral over I_ν only the variable depending on ν, the previous formulation can be written like:

$$(\underline{\mathbf{R}}^{\mathrm{T}} \mathbf{K}_1 \underline{\mathbf{R}}) \int_{I_\nu} \frac{E}{2(1-\nu)} \mathbf{M}^{\mathrm{T}}(\nu) \mathbf{M}(\nu) \, \mathrm{d}I_\nu + (\underline{\mathbf{R}}^{\mathrm{T}} \mathbf{K}_2 \underline{\mathbf{R}}) \int_{I_\nu} \frac{E}{2(1+\nu)} \mathbf{M}^{\mathrm{T}}(\nu) \mathbf{M}(\nu) \, \mathrm{d}I_\nu \tag{4.40}$$

The size of the LHS is $n_{dof\nu} \times n_{dof\nu}$. Note that the terms $\underline{\mathbf{R}}^{\mathrm{T}} \mathbf{K}_1 \underline{\mathbf{R}}$ and $\underline{\mathbf{R}}^{\mathrm{T}} \mathbf{K}_2 \underline{\mathbf{R}}$ give a scalar values.
In a similar way the RHS of equation (4.23) can be derived. Taking into account of equation (4.38), the RHS becomes:

$$\sum_{i=1}^{n} \left\{ (\underline{\mathbf{R}}^{\mathrm{T}} \mathbf{K}_1 \underline{\mathbf{F}}_i) \int_{I_\nu} \frac{E}{2(1-\nu)} \mathbf{M}^{\mathrm{T}}(\nu) \mathbf{M}(\nu) \, \mathrm{d}I_\nu + (\underline{\mathbf{R}}^{\mathrm{T}} \mathbf{K}_2 \underline{\mathbf{F}}_i) \int_{I_\nu} \frac{E}{2(1+\nu)} \mathbf{M}^{\mathrm{T}}(\nu) \mathbf{M}(\nu) \, \mathrm{d}I_\nu \right\} \underline{\mathbf{G}}_i \tag{4.41}$$

Again, the terms $\underline{\mathbf{R}}^{\mathrm{T}} \mathbf{K}_1 \underline{\mathbf{F}}_i$ and $\underline{\mathbf{R}}^{\mathrm{T}} \mathbf{K}_2 \underline{\mathbf{F}}_i$ are scalar values. The size of the RHS is $n_{dof\nu} \times 1$.
In Appendix B the Matlab codes for the second approach of implementation are shown.

4.4 Geometry

The FE model for the Poisson's ratio problem is the same of the load position problem, for more details refer to section 3.4.

4.5 Results

In this section the off-line phase results and the on-line phase results are illustrated. First the outputs, the CPU time and the PGD convergence will be discussed. Second the PGD diplacements for the Poisson's ratio problem and comparisons with the FE solution using StaBIL will be discussed.

4.5.1 Off-line phase results

The PGD solution is given by equation (4.2). So that the PGD outputs in order to obtain the meta model are two matrices. The first one is the matrix $\mathbf{F}$ that includes all the spatial modes. The size of $\mathbf{F}$ is $n_{dof} \times n_{iter}$. The second one is the matrix $\mathbf{G}$ that includes all the Poisson's modes. The size of $\mathbf{G}$ is $n_{dof\nu} \times n_{iter}$. The *off-line phase* code is reproduced in Appendix B.
Figure 4.3 shows the Poisson's modes for $i = 1, 2, 3, 8$: in x- axis there is the FE discretization of the Poisson's ratio domain I_ν and in y-axis the scalar values $G(\nu)$. The PGD solution is given by multiplication of the spatial mode $\mathbf{F}(\mathbf{x})$ and the scalar value $G(\nu)$ for each iteration. In particular the scalar value $G(\nu)$ at each iteration depends on the chosen Poisson's ratio. Figure 4.3 is referred to a Poisson's ratio domain discretized with 25 1D elements of length 0.02.
Particularly interesting is figure 4.4. It shows the convergence of the PGD method for the Poisson's ratio for varying iterations. The red line describes the convergence using the first approach and the blue line the convergence using the second approach.

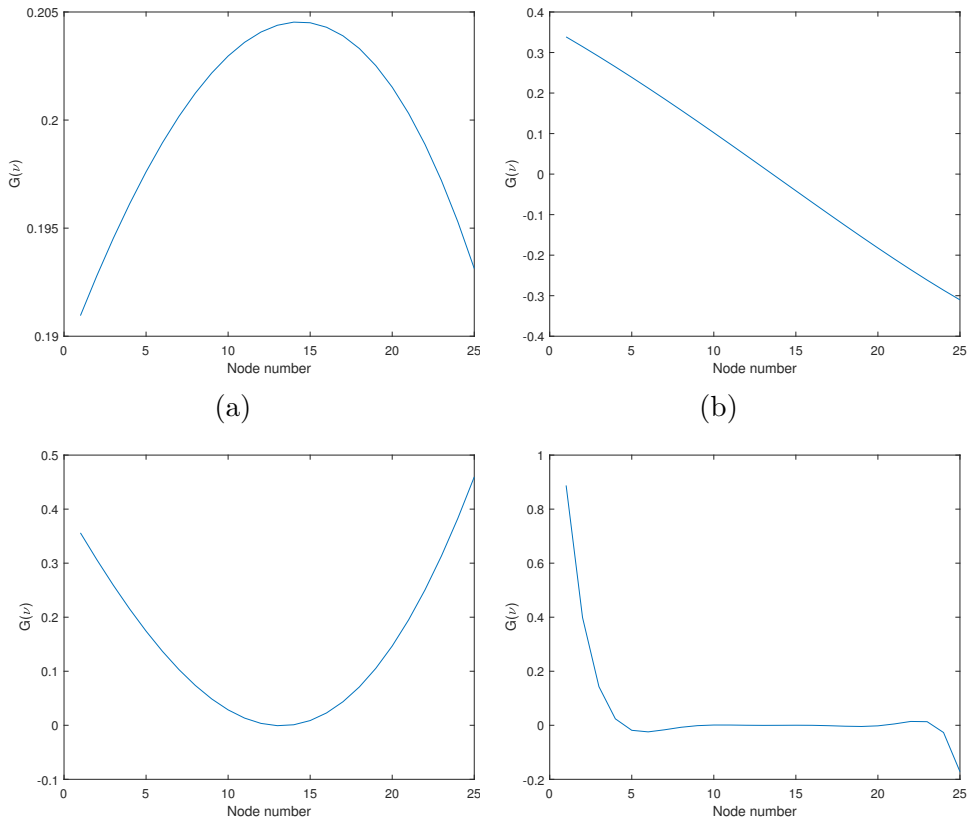


Figure 4.3: Poisson's modes: (a) Poisson's mode $\mathbf{G}_1$, (b) Poisson's mode $\mathbf{G}_2$, (c) Poisson's mode $\mathbf{G}_3$, (d) Poisson's mode $\mathbf{G}_8$.

The convergence is reached at iteration 9 for the first approach and at iteration 8 for the second approach. It could depend on the scheme of integration adopted and further also on the accuracy of the solution. Indeed, the inner integrals over I_ν of equation (4.16) have to be computed at each iteration, and so the PGD method using the first approach needs more iterations to reach the convergence. Note that the maximum number of iterations for the second approach does not depend on the discretization of the Poisson's ratio domain, in other words the convergence is always reached with 8 iterations. In both cases, the convergence is reached quickly, for that the fixed algorithm point is a good method to implement the PGD method. The maximum number of iterations to reach the convergence does not depend on the initial vector $\mathbf{P_0}$ if it is assumed as random or unit vector. There is a little difference of error between the two approaches due to the linear interpolation adopted to implement the formulations. Note that the number of elements used to discretized the Poisson's domain is 25, thus it is a quite dense mesh.
In order to check the convergence of the method a finer mesh has been adopted. In particular the Poisson's ratio domain has been discretized with 50 1D elements of length 0.01. Considering this discretization, the method behaves in different way as expected. The second approach of implementation leads to the same number of iterations and the same global trend. Instead the first approach of implementation, using a finer mesh, leads to 12 iterations to reach the convergence.

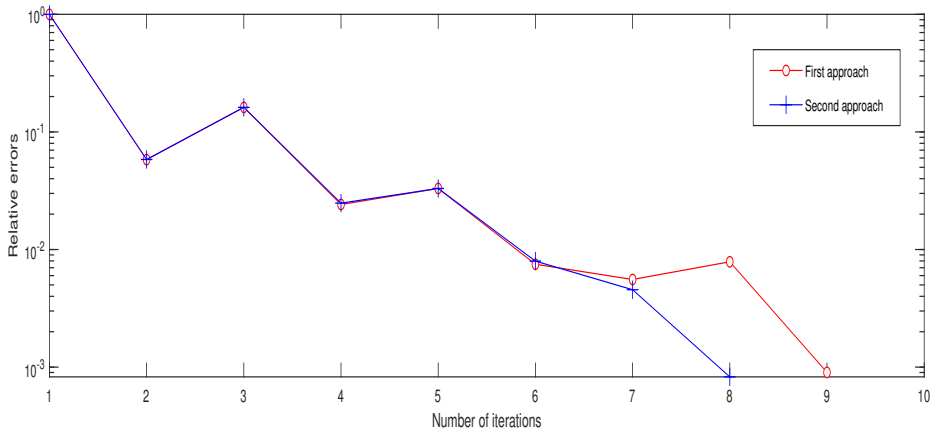


FIGURE 4.4: Relative errors using the first and the second approach of implementation.

Figure 4.5 shows the CPU time requested for each iteration using the first and the second approach. In particular, it is possible to remark that the computational cost, using the first approach, increases with the number of iterations (blue line) like the second approach (red line) since at each subsequent iteration there are more terms to improve the solution. The second approach leads to a lower computational cost for varying iterations. Basically it is due to the principle in which the numerical

integration of the first approach is performed. In particular, in the first approach, the stiffness matrix $\mathbf{K}$ computed from the modified constitutive matrix $\mathbf{D}'$ should be found at each iteration. Further the constitutive matrix $\mathbf{D}'$ is computed term by term and it is computationally expensive. Indeed the biggest computational time is due to the computation of the spatial vector $\underline{\mathbf{R}}$, where the stiffness matrix computed from the modified constitutive matrix takes several minutes. In particular the LHS of equation (4.16) takes 2 minutes while the RHS takes 20 minutes in total. As expected the RHS takes a bigger time since it takes into account of the enrichment terms. Instead, the second approach computes the stiffness matrices $\mathbf{K}_1$ and $\mathbf{K}_2$ out from the enrichment step. Further it solves just one dimensional integrals which is computationally cheap. Finally, the second approach of implementation respects completely the PGD idea.

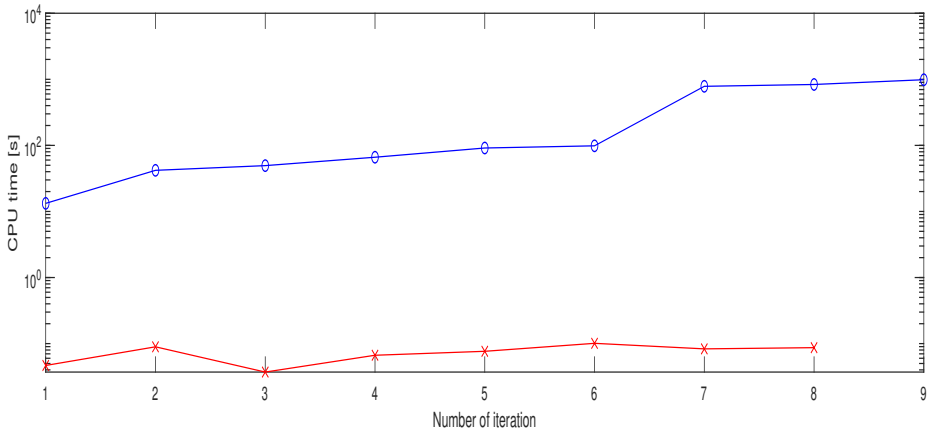


Figure 4.5: CPU time per iterations using the first and the second approach.

Table 4.1 shows some interesting results regarding the CPU time of the PGD method and FE method. The CPU time has been evaluated considering the time that the algorithm takes to find the solution for one particular Poisson's value. Of course, the CPU time for the FEM has been evaluated considering the time requested for all possible Poisson's ratio. For that, the CPU time for the FE program can be seen as mix between the off-line and on-line phase. The time requested for the computation of the off-line phase is higher than the time requested for the on-line phase. Note that the only difference between the first and the second approach is the computation

Table 4.1: CPU time: PGD method VS FE method

	PGD CPU time		**FEM CPU time**
	First approach	***Second approach***	-
Offline phase	45 min	0.58 s	-
Online phase	0.0005 s	0.0005 s	0.025 s

time of the off-line phase since the on-line phase code is the same for both approaches. Further, there is a big difference between the CPU time requested for the on-line phase and the CPU time of the FE program. Again important benefits, in term of computation time, are remarked. The laptop used is an ASUS F552C with Processor Intel Core i7 3537U, memory of 4.0 GB and HDD 500 GB.

4.5.2 On-line phase results

Once the *off-line phase* is executed, in the *on-line phase* the user is able to choose and digit on the keyboard the desidered Poisson's ratio ν within a range $[0, 0.5[$. The *on-line phase* code is reproduced in Appendix B.
In this section the results of two different approaches of implementation are shown. Table 4.2 shows the maximum displacements for several Poisson's values obtained using the PGD method and the FE method (StaBIL). Regard the PGD method, the table shows the displacements found using the two approaches. The accuracy of the displacements using the first approach is less than the accuracy of the displacements using the second approach. This is due to the linear interpolation adopted to perfom the numerical integration. Both procedures could be adopted as valid.

TABLE 4.2: Displacements for varying Poisson's ratio: PGD method VS FE method.

	Displacments [m]		
	PGD method		**FE method**
Poisson's ratio \nu	***First approach***	***Second approach***	-
0.1	0.121584	0.121589	0.121732
0.15	0.121989	0.121997	0.122056
0.2	0.122218	0.122226	0.122328
0.25	0.122501	0.122509	0.122549
0.3	0.122649	0.122658	0.122719
0.35	0.122813	0.12282	0.12284
0.4	0.122907	0.122911	0.12291
0.45	0.122931	0.122929	0.122928

Chapter

5 Conclusions and future works

This thesis studied static structural mechanics problems applying reduced order modelling techniques. In particular, the goal was to see the validity of the PGD method and compare it with the FE method. The work has led to very interesting results in terms of computational cost for instance. The extra dimensions of the problem considered are the load position and Poisson's ratio.

5.1 Conclusive remarks

The PGD approach introduces big advantage in terms of computational costs also if the PGD scheme involves in an iterative procedure. Further, at the end of this thesis, it has been seen that a coarse discretizations influences the computational costs, as expected. In particular a finer mesh for the extra dimension requires higher CPU time. For that the physical space discretization affects the results. Finally, a finer mesh influences the accuracy of the displacements.
Thanks to the separated nature of the method, it is possible to solve a general parametric problem for a large number of parameters as well, obtaining a global reduced computational cost - offline and online phases - which is less than the computational cost of the FE software. The results obtained underline the very low computational costs required by the PGD method, which is a fundamental requirement in the reduce order modelling (ROM) orientated to an offline-online splitting of the algorithm. It allows the user, in the online phase, to choose any extra dimension problem. Further, the low computational cost and the low storage requirements to use the offline phase results on low performance devices.
In general, from the mathematical point of view, a lot of work needs to be done in order to develop a tool to implement in a better way the PGD method. Since the main goal of the PGD method is to reduce the computational costs and solve the problem once, the mathematical formulation should be seen from a particular point of view. In order to reduce the CPU time, the goal is to obtain just 1D integrals which are much less expensive than the 2D integral and so on. It has been demostrated in this thesis, where the second approach of implementation for the Poisson's problem, shows a very low computational cost. For that the authors should find a way to derive formulations with integrals depending on one parameter. In other words, this

idea respects the separated representation of the solution using the PGD method. Thus, the solution is given by the numerical integration of several integrals which are depending on just one variables.
The PGD method for the load position (chapter 3) leads just to integrals depending on just one variable and so further mathematical treatments are not necessary. Instead the final formulations of the PGD method for the Poisson's ratio (chapter 4) leads to a double integral which need to be splitted in order to obtain gain using the PGD method. Finally, the more modes are obtained, the higher computational cost will be.

5.2 Recommendations for further research

Both PGD applications presented in this thesis form a foundation for further research developments, in particular for projects towards more complex scenario in structural mechanics problems.
In this thesis the PGD method has been applied to static solid mechanics problems. It could be interesting to implement the PGD method for static solid mechanics problems considering as extra dimensions problem the material parameters such as the Poisson's ratio ν, the Young's modulus E or the material density ρ. Further it is possible to combine them with the extra mechanical parameters like the load position or a prescribed settlement of the struture for instance. Obviously assuming all these variables as extra parameters, the problem becomes more difficult from the computational point of view. But at the same time, solving it with 1D integrals, could be a very interesting application since every possible input defining the problem can be particularized. Future work could be the PGD method applied to dynamics problems where, for instance, the frequency ω or the initial condition $\mathbf{u}_0$ are the extra dimension problem. In this case there is another additional variable that is the time t.
The numerical integration has been performed with the standard Gaussian quadrature rule and it could be more expensive than a more efficient quadrature rule like Gauss-Lobatto. Further, the PGD scheme could be implemented using other high-efficiency programming languages such as C/C++.

Bibliography

[1] Ammar, A., Mokdad, B., Chinesta, F., and Keunings, R. *A new family of solvers for some classes of multidimensional partial differential equations encountered in kinetic theory modeling of complex fluids. Part II: transient simulation using space-time separated representations*, vol. 144. J. Non Newton. Fluid mech, 2007.

[2] Chinesta, F., Ammar, A., and Cueto, E. Recent advances in the use of the proper generalized for solving multidimensional models. *Archives of Computational Methods in Engineering 17(4)* (2010), 327–350.

[3] Chinesta, F., and Cueto, E. *PGD-based modeling of materials, structures and processes.* Springer, Switzerland, 2014.

[4] Chinesta, F., Ladeveze, P., and Cueto, E. A short review on model order reduction based on proper generalized decomposition. *Archives of Computational Methods in Engineering 18* (2011), 395–404.

[5] Cueto, E., Gonzlez, D., and Alfaro, I. *Proper Generalized Decompositions: an introduction to computer implementation with Matlab.* Springer, 2015.

[6] De Roeck, G. *The finite element method.* KU Leuven, 2013.

[7] Geuzaine, C., and Remacle, J. A three-dimensional finite element mesh generator, http://gmsh.info.

[8] Ghnations, C., Masson, F., Huerta, A., Leygue, A., Cueto, E., and Chinesta, F. Proper generalized decomposition based dynamic data-driven control of thermal processes. *Computational Methods Applied Mechanical Engineering* (2011).

[9] Giner, E., Bognet, B., Rodenas, J., Leygue, A., Fuenmayor, F., and Chinesta, F. The proper generalized decomposition as a numerical procedure to solve 3d cracked plates in linear elastic fracture mechanics. *Internatonal Journal of Solids and Structures* (2012).

[10] Laughlin, R., and Pines, D. *The theory of everything.*, vol. 97. Proc. Natl. Acad. Sci.), 2000.

[11] Modestp, D., Zlotnik, S., and Huerta, A. Proper generalized decomposition for parameterized helmholtz problems in heterogeneous and unbounded domains: application to harbour agitation. *Laboratori de Clcul Numerc (LaCn), Universitat Politcnica de Catalunya, Barcelona, Spain*.

[12] Niroomandi, S., Gonzalez, D., Alfaro, I., Bordeu Weldt, B., and Leygue, A. Real time simulation of biological soft tissues: a proper generalized decomposition approach. *International Journal for Numerical Methods in Biomedical Engineering.* (2017), 586–600.

[13] of Colorado, U. https://www.colorado.edu/engineering/cas/courses.d/ifem.

[14] of Zaragoza, U. http://amb.unizar.es/projects.

[15] Pric, S., and Rogers, Y. *Let's get physical: the learning benefits of interacting in digitally augmented physical spaces.*, vol. 43. Comput. Educ., 2004.

[16] Quesada, C., Gonzlez, D., Alfaro, I., Cuedo, E., Huerta, A., and Chinesta, F. Final report. dddas workshop at arlington. *National Science Foundation* (2006).

[17] Talbot, D. Given tablets but no teachers, ethiopian children teach themselves (2012), http://www.technologyreview.com/news/506466/given-tablets-but-no-teachers-ethiopian-children-teach-themselves.

[18] Xi, Z. *Computational Mechanics and Advanced Materials.* PhD thesis, Universitat Politécnica de Catalunya, 10 2017.

Appendix

A PGD code for load position

The **off-line phase code** is reproduced below. It solves the off-line phase for the problem of a cantilever beam for varying load positions. The code provides the solution under plane stress condition.

Listing A.1: Off-line phase code for the load position problem.

```
%  MSc - KU Leuven - Faculty of Engineering Science
%
%                  Giuseppe D'Ettorre
%                       May 2018
%
%           Proper Generalized Method applied
%             to solid mechanincs problems
%
%    Boundary conditions as extra problem dimension
%                     OFF-LINE PHASE
%
clear all; close all; clc;
%% 1. VARIABLES
E=1000;
Modulus=1;
TOL=1.0E-03;
num_max_iter=50;

%% 2. Load mesh file
[Nodes,Elements] = GMSHread('beam');
nNodes=size(Nodes);
nelem = size(Elements,1);
triangles=[Elements(:,2) Elements(:,3) Elements(:,4)];

%% 3. Define element types, sections and materials
% Plane stress or plane strain conditions
Options.problem='stress'; % Options.problem='strain';
```

```
Types = {1 'plane3', Options};
Sections = [1];
% Materials = [MatID E nu rho];
Materials = [1 E 0.3 1];
ncoords=2*size(Nodes,1); %spatial coordinates X-Y
Elements = [Elements(:,1) ones(nelem,3) Elements(:,2:end)
   ];
Elements1 = [Elements(:,5) Elements(:,6) Elements(:,7)];

%% 4. Plot FE mesh with node and element numbers
figure;
plotnodes(Nodes,'r','numbering','off');
hold on
plotelem(Nodes,Elements,Types,'numbering','off');
title('Elements and nodes');
axis equal

%% 5. Obtain degrees of freedom and specify boundary
   conditions
DOF = getdof(Elements,Types);
% Clamped boundary at x = 0
seldof = find(Nodes(:,2)==0);
DOF = removedof(DOF,seldof);
nDOF=length(DOF);

%% 6.Domain where the load can be applied : \I_s
X0=0;
X1=3;
tamX=0.1;
force=X0:tamX:X1;
coords2=force'; % vector with the coordinates of the load
    domain
ncoords2=numel(coords2);

%% 7. Allocation of matrices and vectors
F=zeros(nDOF,1);          % Spatial vector
G=zeros(numel(force),1); % Load position vector
f=zeros(nDOF,ncoords2);  % Nodal values for spatial
   coords
g=eye(numel(force));      % Nodal values for load

%% 8. Assemble stiffness matrix, M2 matrix and N2 matrix
% Stiffness matrix K using the FEM: StaBil
[K,M] = asmkm(Nodes,Elements,Types,Sections,Materials,DOF
   );
```

```
% Matrix of linear shape functions to discretize the
   extra-coordinate 's'
[M2] = M2(coords2);

%% 9. Force term in separated form
Node_top=[];
for i1=1:length(Nodes)
    if Nodes(i1,3)==1
    node_top=[i1];
    Node_top=[Node_top;node_top];  %Node where the load
       can be applied: TOP of the beam
    end
end

dof_top=[];
for i1=1:numel(Node_top)
    if i1==2
        dof_top=[dof_top;5];   % (The number 5 is a
           simbolic number! Just to identify that node!)
    else
        dof_top=[dof_top;find(DOF(:,1)==(Node_top(i1,1)
           +0.02))]; % Considering only the .02 dof for
           nodes top
    end
    % Not taking into account of node 4 (clamped node)
       indeed 30x1 and not 31x1 !
end

for i1=1:numel(Node_top)
    if i1==2
        f(dof_top(i1),i1)=0;

    else
        f(dof_top(i1),i1)= - Modulus;  % I have -1 just
           where the load can be applied (thus DOF_top)
    end
end
 f3=f(:,1); % get the first column of 'f'
 f4=f(:,2);

 f2=[];     % get columns from 3 to 31
 for i1=3:numel(dof_top)
     f2=[f2 , f(:,i1)];
 end
 f2=fliplr(f2);
```

```
 f=[f4 ,f2 ,f3];

%% 10. Enrichment step: looking for R and S
num_iter=0;
iter=zeros(1);
Aprt=0;
Error_iter=1;

while Error_iter>TOL && num_iter<num_max_iter
    num_iter = num_iter + 1;
    S0=rand(numel(force),1); %random values for S

    % Enrichment step
    [ R,S,iter(num_iter)] = enrichment_load(K,M2,f,g,S0,F
       ,G,num_iter,TOL,DOF);

    F(:,num_iter)=R;
    G(:,num_iter)=S;

    % Stopping criterion
    Error_iter=norm(F(:,num_iter)*G(:,num_iter)'); %
       Evaluate the norm of the solution u=F*G
    Aprt=max(Aprt,sqrt(Error_iter));
    Error_iter = sqrt(Error_iter)/Aprt;

    % Plot PGD convergence:
    Errors(:,num_iter)=Error_iter;

    fprintf(1,'%dst iteration with an error of %f\n',
       num_iter,Error_iter);
end
num_iter=num_iter-1;
fprintf(1,'PGD off-line phase excuted normally\n\n');
save('Workspace_PGD_load_position_force.mat');
```

The following code shows the **on-line phase**. Running it, the user is able to click any point of the beam and obtain in real time the deformed shape of the beam under an unit vertical load.

LISTING A.2: On-line phase code for the load position problem.

```
%  MSc - KU Leuven - Faculty of Engineering Science
%
%                  Giuseppe D'Ettorre
%                       May 2018
%
```

```
%           Proper Generalized Method applied
%             to solid mechanincs problems
%
%    Boundary conditions as extra problem dimension
%                     ON-LINE PHASE
%
clear all; close all; clc;
load('Workspace_PGD_load_position_force.mat');

%% 1. Online phase : choose the load position on the beam
    surface with a click
fprintf(1,'PGD on-line phase...')
fprintf(1,'\n\nSelect the load position on the beam
   surface\n\n ');
fprintf(1,'\n\n otherwise pick out of the beam to exit!\n
   \n');
h1 = figure(1);
figure;
plotnodes(Nodes,'r','numbering','off')
hold on
plotelem(Nodes,Elements,Types,'numbering','off');
axis equal;

[Cx,Cy]=ginput(1); % Waiting for a click on the beam
lim= 0.1/(X1-X0); % Exit zone out of the beam
Y0=0;
Y1=1;

while X0-lim<=Cx && Cx<=X1+lim && Y0-lim<=Cy && Cy<=Y1+
   lim
    h1=figure(1);
    plotelem(Nodes,Elements,Types,'numbering','off');
    axis equal;
    Posforce=find(force<Cx,1,'last'); %Look for the
       closest loaded node

    % Evaluation of the solution choosing the selected
       node in G vector
    desp=zeros(numel(DOF),1);
    desp=F*G(Posforce,:).';

    % Plotting the solution
    plotdisp(Nodes,Elements,Types,DOF,desp,'dispscal',5)
         % StaBil!

```

```
    title('Vertical displacement [m]: Select the force
       position on the top of the beam or pick out to
       exit...');
    view(2);
    figure(1); axis equal;
    [Cx,Cy]=ginput(1); % Waiting for new load position...
end
fprintf(1,'\n\n _____End of simulation_____ \n\n');
```

The **enrichment function** is illustrated below. It permits to find iteratively the two enrichment functions **R** and **S**.

LISTING A.3: Matlab code for the enrichment function.

```
function [ R,S,iter] = enrichment_load(K,M2,f,g,S0,F,G,
   num_iter,TOL,DOF)
%
%   Computes a new sumand by a fixed point algorithm
   using PGD
%   function [ R,S,iter ] = enrichment( K,M2,S0,F,G,f,g,
   num_iter,TOL,DOF,nDOF,ncoords2)
%   returns the enrichment functions R and S
%
%   K           Global stiffness matrix (DOF * DOF)
%   M2          Integral over the load domain of M(s)'M(s
   ) (DOF_s * DOF_s)
%   f           Nodal values for the spatial coordinates
   (DOF * 1)
%   g           Nodal values for the extra coordinates (
   DOF_s * 1)
%   S0          Random vector to initialise the procedure
    (DOF_s * 1)
%   F           Spatial or mechanical vector (DOF * 1)
%   G           Load vector (DOF_s * 1)
%   num_iter    Number of iteration (1 * 1)
%   TOL         Tolerance (1 * 1)
%   DOF         Degrees of freedom (nDOF * 1)
%
%   Giuseppe D'Ettorre 2018

R=zeros(size(F,1),1);
R0=R;
h=size(M2,2);
ExitFlag=1;
iter=0;
mxit=25;
```

```
nDOF=size(DOF,1);
ncoords2=size(G,1);

while ExitFlag>TOL
    %% Looking for R(x) , assuming S(s) known

    % Computation of LHS equation (3.21)
    matrixR = (S0'*M2*S0)*K;

    % Computation of RHS equation (3.21)
    sourceR=zeros(nDOF,1);
    for k1=1:h
        sourceR = sourceR + (f(:,k1))*(S0'*M2*g(:,k1));
    end
    for i1=1:num_iter-1
        sourceR = sourceR - K*F(:,i1)*(S0'*M2*G(:,i1));
    end

    % Now I can solve R:
    R  = matrixR\sourceR;

    %% Looking for S assuming R known

    % Computation of LHS equation (3.28)
    matrixS=(R'*K*R)*M2;

    % Computation of RHS equation (3.28)
    sourceS=zeros(ncoords2,1);
    for k1=1:h
        sourceS = sourceS + (M2*g(:,k1))*(R'*f(:,k1));
    end
    for i1=1:num_iter-1
        sourceS = sourceS - (R'*K*F(:,i1))*(M2*G(:,i1));
    end

    % Now I can solve S:
    S = matrixS\sourceS;
    S = S./norm(S);

    %% Stopping criterion:
    error=max(abs(sum(R0-R)),abs(sum(S0-S)));
    R0=R;
    S0=S;
    iter=iter+1;
    if iter>mxit || abs(error)<TOL
```

```
        return
    end
end
return
```

In the load position problem, the only extra function nedeed is the function **M2** reproduced below. It is the integral over $\overline{\Gamma}$ of the product of the shape functions $\mathbf{M}(s)$.

LISTING A.4: Matlab code for the function M2.

```
function [M2] = M2(coords2)
%
%   function [M2] = M2(coords2) performs the numerical
   integration of M(s)'M(s)
%
%   M2          Integral over the load domain of M(s)'M(s
   ) (DOF_s * DOF_s)
%   coords2     Load domain (DOF_s * 1)
%
%   Giuseppe D'Ettorre 2018

sg = [-0.57735 0.57735];
wg = ones(2,1);
npg = numel(sg);
ncoords_2 = numel(coords2); % # coords of 's'

M2 = zeros(ncoords_2);

% Coordinates of elements
X2 = coords2(1:ncoords_2-1)';
Y2 = coords2(2:ncoords_2)';
L2 = Y2 - X2;

for i1=1:ncoords_2-1
    w=zeros(1,npg);
    M=zeros(ncoords_2,npg);
    w(1,:) = 0.5.*(1.0-sg).*X2(i1) + 0.5.*(1.0+sg).*Y2(i1
       );

    M(i1+1,:) = (w(1,:)-X2(i1))./L2(i1);
    M(i1,:) = (Y2(i1)-w(1,:))./L2(i1);

    for j1=1:npg
        M2 = M2 + M(:,j1)*M(:,j1)'*0.5.*wg(j1).*L2(i1); %
            int M'(s)M(s) \d_I_s
```

```
    end
end
return
```

Appendix

B PGD code for Poisson's ratio

In this appendix, the two approaches described in the section 4.3 are reproduced.

B.1 First approach of implementation

The **off-line phase code** is reproduced below. It leads to the vademecuum that includes all the possible solution of the problem.

Listing B.1: Off-line phase code for the Poisson's ratio problem (first approach).

```
%       MSc - KU Leuven - Faculty of Engineering Science
%
%                   Giuseppe D'Ettorre
%                        May 2018
%
% Proper Generalized Method applied to solid mechanincs
   problems
%
%  Material parameters as extra problem dimension
%                     OFF-LINE PHASE
%           First approach of implementation

clear all; close all; clc;

%% 1. VARIABLES
E=1000;
TOL=1.0E-03;
num_max_iter=10;

%% 2. Load mesh file
[Nodes,Elements] = GMSHread('beam');
nNodes=size(Nodes);
nelem = size(Elements,1);
triangles=[Elements(:,2) Elements(:,3) Elements(:,4)];
```

```
%% 3. Define element types, sections and materials
% Plane stress or plane strain conditions
Options.problem='stress'; % Options.problem='strain';
Types = {1 'plane3', Options     % Element to compute
   K_nu LHS equation (4.16)
         2 'plane32', Options};  % Element to compute K2
              equation (4.16)

Sections = [1];

% Poisson's ratio domain
I_nu=0:0.02:0.499;
ncoords_nu=numel(I_nu);
ncoords=2*size(Nodes,1); %spatial coordinates X-Y
Elements = [Elements(:,1) ones(nelem,3) Elements(:,2:end)
   ];
Elements1 = [Elements(:,5) Elements(:,6) Elements(:,7)];
   % Only vertices

Materials1=[];
for i1=1:ncoords_nu
    Materials1=[Materials1; 1 E I_nu(i1) 1];
end


%% 4. Plot FE mesh with node and element numbers
figure;
plotnodes(Nodes,'r','numbering','off')
hold on
plotelem(Nodes,Elements,Types,'numbering','off');
title('Elements and nodes');
axis equal

%% 5. Obtain degrees of freedom and specify boundary
   conditions
DOF = getdof(Elements,Types);
% Clamped boundary at x = 0
seldof = find(Nodes(:,2)==0);
DOF = removedof(DOF,seldof);
nDOF = length(DOF);

%% 6. Define loading
P_load = nodalvalues(DOF,3.02,-1); %vector with 1 at node
    where the load is applied
```

```
64 %% 7. Allocation of vectors
   F=zeros(nDOF,1); % vector nodal values spatial coords
   G=zeros(numel(I_nu),1); % vector nodal values Poisson's
      ratio
67
   %% 8. Matrix M1
   % Matrix of linear shape functions M and (int M)=M1
70 [M1] = M1(I_nu);
   % Sizes matrices M and M1 : 1 x Ndof_nu
   M1=M1';
73

   %% 10. Enrichment step: looking for R and P
76 num_iter=0;
   iter=zeros(1);
   Aprt=0;
79 Error_iter=1;

   Types1 = {1 'plane3', Options     % Element to compute
      K_nu LHS equation (4.16)
82           1 'plane32', Options    % Element to compute K2
                RHS equation (4.16)
             1 'plane33', Options};  % Element to compute
                matrixP equation (4.23)

85
    % Computation LHS equation (4.23) --> I get a 3D matrix
   K3=zeros(nDOF,nDOF,ncoords_nu);
88
       for i1=1:ncoords_nu
       Materials=Materials1(i1,:);
91     Types=Types1(3,:); % plane33_element
       K3(:,:,i1) = asmK3(Nodes,Elements,Types,Sections,
          Materials,DOF,I_nu);
       end
94
   while Error_iter>TOL && num_iter<num_max_iter
       num_iter = num_iter + 1;
97
       P0=ones(ncoords_nu,1);

100    % Enrichment step
       [ R,P,iter(num_iter)] = enrichment(K3,P_load,M1,P0,F,
          G,num_iter,TOL,DOF,I_nu,Nodes,Elements,Types1,
```

```
        Sections,Materials,Materials1);

    F(:,num_iter)=R;  % F=R  - spatial coord
    G(:,num_iter)=P;  % G=P  - extra coord - Poisson's
        ratio

    % Stopping criterion
    Error_iter=norm(F(:,num_iter)*G(:,num_iter)'); %
        Evaluate the norm of the solution u=F*G
    Aprt=max(Aprt,sqrt(Error_iter));
    Error_iter = sqrt(Error_iter)/Aprt;

    % Plot PGD convergence:
    Errors(:,num_iter)=Error_iter;

    fprintf(1,'%dst iteration with an error of %f\n',
        num_iter,Error_iter);

end
num_iter=num_iter-1;
fprintf(1,'PGD off-line phase exited normally\n\n');
save('Workspace_PGD_Poisson_ratio_first_approach.mat');
```

The following code shows the **on-line phase**. Running it, the user is able to digit on the keyboard the desidered Poisson's ratio and get in real time the beam configuration under unit load applied at node 3.

Listing B.2: On-line phase code for the Poisson's ratio problem (first approach).

```
%       MSc - KU Leuven - Faculty of Engineering Science
%
%                  Giuseppe D'Ettorre
%                       May 2018
%
% Proper Generalized Method applied to solid mechanincs
    problems
%
%  Material parameter as extra problem dimension
%                    ON-LINE PHASE
%           First approach of implementation
tic
clear all; close all; clc;

%% 1. Online phase: digit the Poisson's ratio on the
    keyboard. . .
load('Workspace_PGD_Poisson_ratio_first_approach');
```

```
fprintf(1,'PGD on-line phase...\n\n');
fprintf(1,'\n\nDigit on the keyboard the Poissons ratio
   between the range [0;0.5[ \n\n');
nu=input('\n\n nu = \n\n');

lim=0.005; % Exit zone out of the Poisson's ratio range
nu0=I_nu(1,1);
nu1=I_nu(1,ncoords_nu);

while nu0-lim<=nu && nu<=nu1+lim
     nu_chosen=find(I_nu<nu,1,'last'); %Look for the
        closest Poisson value
     % Evaluation of the solution choosing the selected
        node in G vector
     desp=zeros(nDOF,1);
     desp=F*G(nu_chosen,:).';

%    Plotting the solution
    plotdisp(Nodes,Elements,{1 'plane3'},DOF,desp,'
       dispscal',1)   % StaBil!
    title('Vertical displacement [m]');

    nu=input('\n\n nu = \n\n'); % New Poisson ratio...
end
  toc
 fprintf(1,'\n\n-------- End of simulation --------\n\n')
    ;
```

The LHS of equation (4.16) is computed using a linear triangular elements, in StaBIL *ke.plane3.m*. The stiffness matrix at element level, is computed using a modified constitutive matrix called in the Matlab code **D1**. The element stiffness matrices are assembled in the function of StaBIL $asmK_{\nu}.m$.

Listing B.3: Element stiffness matrix used to compute the LHS of equation (4.16).

```
function [Ke,Me]=ke_plane3(Node,Section,Material,Options,
   I_nu,P0)
%
%   KE_PLANE3   Plane 3 element (CST) stiffness matrix
%               computes from a modified constitutive
   matrix D1.
%
%   [Ke]=ke_plane3(Node,Section,Material,Options,I_nu,P0)
    returns the element
```

```
%   stiffness in the global coordinate system for a 3-node plane CST element.
%
%   Node        Node definitions            [x y z] (3 * 3)
%   Material    Material definition         [E nu rho]
%   Options     Element options             {Option1 Option2 ...}:
%                                           'planestress' (default)
%                                           'axisym'
%                                           'planestrain'
%   I_nu        Poisson's domain            (1 * DOF_nu)
%   P0          Random vector to initialise the procedure (DOF_nu * 1)

% MATERIAL PROPERTIES
E=Material(1);

% Constitutive matrix
switch lower(Options.problem)
case 'stress'  % PLANE STRESS
sg = [-0.57735 0.57735];
wg = [1 1];
ngp=numel(sg);
ncoords_nu=numel(I_nu);
nu1=I_nu(1:ncoords_nu-1)';
nu2=I_nu(2:ncoords_nu)';
L=nu2-nu1;
D1=zeros(3,3);

for iD = 1:9
    T=zeros(ncoords_nu);
    D=zeros(1,1);
    for i1=1:ncoords_nu-1
        w=zeros(1,ngp);
        w(1,:)=0.5.*(1-sg).*nu1(i1) + 0.5.*(1+sg).*nu2(i1);
        M=zeros(2,ngp);
        M(1,:)=(nu2(i1)-w(1,:))./L(i1);
        M(2,:)=(w(1,:)-nu1(i1))./L(i1);
        for j1=1:ngp
            Dt = E/(1-w(:,j1)^2)*[1   w(:,j1) 0;
                                  w(:,j1) 1   0;
                                  0   0 (1-w(:,j1))/2];
           D(j1,j1) = Dt(iD);
```

```
        end
      T(i1:i1+1,i1:i1+1) = T(i1:i1+1,i1:i1+1) + M.'*(D*
         diag(wg))*M*0.5*L(i1);
    end
    D1(iD) = P0.'*T*P0;
end
end

% Triangle shape function
X=Node(:,1);
Y=Node(:,2);
b1=Y(2)-Y(3); b2=Y(3)-Y(1); b3=Y(1)-Y(2);
c1=X(3)-X(2); c2=X(1)-X(3); c3=X(2)-X(1);

% Element area
Delta=0.5*det([1 X(1) Y(1)
               1 X(2) Y(2)
               1 X(3) Y(3)]);

% Shape function derivatives
Be=1/(2*Delta)*[b1  0  b2   0  b3   0
                0  c1  0  c2   0  c3
                c1  b1 c2  b2  c3  b3];

% Stiffness matrix
Ke=Delta*Be.'*D1*Be;

% Mass matrix
if nargout>1
  Me=1/3*rho*Delta*eye(6);
end
return
```

In a similar way, the RHS of equation(4.16) is computed. In this case the triangular element used is called in StaBIL *ke.plane32.m*. The element stiffness matrices are assembled with the function of StaBIL *asmK2.m*.

Listing B.4: Element stiffness matrix used to compute the RHS of equation (4.16).

```
function [Ke,Me]=ke_plane32(Node,Section,Material,Options
   ,I_nu,P0,G)
%
%   KE_PLANE32  Plane 32 element (CST) stiffness matrix
%               computes from a modified constitutive
   matrix D1.
%
```

```
%   [Ke]=ke_plane32(Node,Section,Material,Options,I_nu,P0
    ) returns the element
%   stiffness in the global coordinate system for a 3-
    node plane CST element.
%
%   Node        Node definitions            [x y z] (3 * 3)
%   Material    Material definition         [E nu rho]
%   Options     Element options             {Option1
    Option2 ...}:
%                                           'planestress' (
    default)
%                                           'axisym'
%                                           'planestrain'
%   I_nu        Poisson's domain            (1 * DOF_nu)
%   P0          Random vector to initialise the procedure
    (DOF_nu * 1)
%   G           Poisson vector              (DOF_nu * 1)

% MATERIAL PROPERTIES
E=Material(1);
if (nargout>1), rho=Material(1,3); end


% Computation of the new constitutive matrix
switch lower(Options.problem)
case 'stress'  % PLANE STRESS
sg=[-0.57735 0.57735];
wg=[1 1];
ngp=numel(sg);
ncoords_nu=numel(I_nu);
nu1=I_nu(1:ncoords_nu-1)';
nu2=I_nu(2:ncoords_nu)';
L=nu2-nu1;
D1=zeros(3,3);

D1=zeros(3,3);
for iD = 1:9
    T=zeros(ncoords_nu);
    D=zeros(1,1);
    for i1=1:ncoords_nu-1
        w=zeros(1,ngp);
        w(1,:)=0.5.*(1-sg).*nu1(i1) + 0.5.*(1+sg).*nu2(i1
           );
        M=zeros(2,ngp);
        M(1,:)=(nu2(i1)-w(1,:))./L(i1);
```

```
        M(2,:)=(w(1,:)-nu1(i1))./L(i1);
        for j1=1:ngp
            Dt = E/(1-w(:,j1)^2)*[1   w(:,j1) 0;
                                  w(:,j1) 1   0;
                                  0   0 (1-w(:,j1))/2];
             D(j1,j1) = Dt(iD);
        end
           T(i1:i1+1,i1:i1+1) = T(i1:i1+1,i1:i1+1) + M
              .'*(D*diag(wg))*M*0.5*L(i1);
    end
    D1(iD) = P0.'*T*G;
end
end

% Triangle shape function
X=Node(:,1);
Y=Node(:,2);
b1=Y(2)-Y(3); b2=Y(3)-Y(1); b3=Y(1)-Y(2);
c1=X(3)-X(2); c2=X(1)-X(3); c3=X(2)-X(1);

% Element area
Delta=0.5*det([1 X(1) Y(1)
               1 X(2) Y(2)
               1 X(3) Y(3)]);

% Shape function derivatives
Be=1/(2*Delta)*[b1   0   b2    0   b3    0
                0   c1   0   c2    0   c3
                c1   b1  c2   b2   c3   b3];

% Stiffness matrix
Ke=Delta*Be.'*D1*Be;

% Mass matrix
if nargout>1
  Me=1/3*rho*Delta*eye(6);
end
return
```

The LHS of equation (4.23), called in Matlab *matrixP1.m*, is reproduced below.

Listing B.5: LHS of equation (4.23).

```
function [matrixP]=matrixP1(K3,R,I_nu)
%
```

```
% function [matrixP]=matrixP1(K3,R,I_nu) computes the LHS
    of equation
% (4.23)
%
%   K3            Global 3D stiffness matrix for every
   Poisson's value of I_nu (DOF * DOF)
%   R             Spatial or mechanical vector (DOF * 1)
%   I_nu          Poisson's domain (1 * DOF_nu)
%
%   matrixP        (DOF_nu * DOF_nu)


sg=[-0.57735 0.57735];
wg=[1 1];
ngp=numel(sg);
ncoords_nu=numel(I_nu);
nu1=I_nu(1:ncoords_nu-1)';
nu2=I_nu(2:ncoords_nu)';
L=nu2-nu1;
matrixP=zeros(ncoords_nu,ncoords_nu);
matrix=zeros(ncoords_nu,1);

%% Computation of R'*K(\nu)*R
for j1=1:ncoords_nu
    matrix(j1,:)=R'*K3(:,:,j1)*R;
end

for i1=1:ncoords_nu-1
    w=zeros(1,ngp);
    y=zeros(1,ngp);
    M=zeros(ncoords_nu,ngp);
    w(1,:)=0.5.*(1-sg).*nu1(i1) + 0.5.*(1+sg).*nu2(i1);
    M(i1+1,:)=(w(1,:)-nu1(i1))./L(i1);
    M(i1,:)=(nu2(i1)-w(1,:))./L(i1);

    % Linear interpolation to find the value of R'*K(\nu)
       *R in the Gauss points
    y(1,:) = matrix(i1) + (w(1,:)-nu1(i1))*((matrix(i1+1)
       -matrix(i1))/(nu2(i1)-nu1(i1)));

    for j1=1:ngp
        matrixP = matrixP + M(:,j1)*y(:,j1)*M(:,j1)'*wg(
           j1)*0.5*L(i1); % y is a single value!
    end
end
```

```
43 return
```

The RHS of equation (4.23), called in Matlab *sourceP1.m*, is reproduced below.

Listing B.6: RHS of equation (4.23).

```
function [sourceP]=sourceP1(I_nu,F,K3,R)
%
% function [sourceP]=sourceP1(I_nu,F,K3,R) computes the RHS of equation
% (4.23)
%
%   K3            Global 3D stiffness matrix for every Poisson's value of I_nu (DOF * DOF)
%   R             Enrichment function for the spatial or mechanical vector (DOF * 1)
%   F             Spatial or mechanical vector (DOF * 1)
%   I_nu          Poisson's domain (1 * DOF_nu)
%
%   soruceP       (DOF_nu * 1)


sg=[-0.57735 0.57735];
wg=[1 1];
ngp=numel(sg);
ncoords_nu=numel(I_nu);
nu1=I_nu(1:ncoords_nu-1)';
nu2=I_nu(2:ncoords_nu)';
L=nu2-nu1;
sourceP=zeros(ncoords_nu,ncoords_nu);
source=zeros(1,ncoords_nu);

%% Computation of R'*K(\nu)*F
for j1=1:ncoords_nu
    source(:,j1)=R'*K3(:,:,j1)*F;
end

for i1=1:ncoords_nu-1
    w=zeros(1,ngp);
    y=zeros(1,ngp);
    M=zeros(ncoords_nu,ngp);
    w(1,:)=0.5.*(1-sg).*nu1(i1) + 0.5.*(1+sg).*nu2(i1);

    % Linear interpolation to find the value of R'*K(\nu)*R in the Gauss points
```

```
    y(1,:) = source(i1) + (w(1,:)-nu1(i1))*((source(i1+1)
       -source(i1))/(nu2(i1)-nu1(i1)));
    M(i1+1,:)=(w(1,:)-nu1(i1))./L(i1);
    M(i1,:)=(nu2(i1)-w(1,:))./L(i1);

    for j1=1:ngp
    sourceP = sourceP + M(:,j1)*y(:,j1)*M(:,j1)'.*wg(j1)
       .*0.5*L(i1);
    end
end
return
```

B.2 Second approach of implementation

The **off-line phase code** is reproduced below. It leads to the vademecum that includes all the possible solution of the problem.

Listing B.7: Off-line phase code for the Poisson's ratio problem (second approach).

```
%  MSc - KU Leuven - Faculty of Engineering Science
%
%                  Giuseppe D'Ettorre
%                       May 2018
%
%           Proper Generalized Method applied
%             to solid mechanincs problems
%
% Material parameters as extra problem dimension
%                     OFF-LINE PHASE
%
clear all; close all; clc;
%% 1. VARIABLES
E=1000;
TOL=1.0E-03;
num_max_iter=50;

%% 2. Load mesh file
[Nodes,Elements] = GMSHread('beam');
nNodes=size(Nodes);
nelem = size(Elements,1);
triangles=[Elements(:,2) Elements(:,3) Elements(:,4)];

%% 3. Define element types, sections and materials
% Plane stress or plane strain conditions
Options.problem='stress'; % Options.problem='strain';
```

```
Types = {1 'plane1', Options      % Triangular element to
   compute K1 using D1
28          2 'plane2', Options};   % Triangular element to
               compute K2 using D2
Sections = [1];

31 % Poisson's ratio domain
I_nu=0:0.02:0.499;
ncoords_nu=numel(I_nu);
34 ncoords=2*size(Nodes,1); %spatial coordinates X-Y
Elements = [Elements(:,1) ones(nelem,3) Elements(:,2:end)
   ];
Elements1 = [Elements(:,5) Elements(:,6) Elements(:,7)];
   % Only vertices
37 Materials=[1 E 0 1];

%% 4. Plot FE mesh with node and element numbers
40 figure;
plotnodes(Nodes,'r','numbering','off')
hold on
43 plotelem(Nodes,Elements,Types,'numbering','off');
title('Elements and nodes');
axis equal
46

%% 5. Obtain degrees of freedom and specify boundary
   conditions
DOF = getdof(Elements,Types);
49 % Clamped boundary at x = 0
seldof = find(Nodes(:,2)==0);
DOF = removedof(DOF,seldof);
52 nDOF = length(DOF);

%% 6. Define loading
55 P_load = nodalvalues(DOF,3.02,-1); %vector with 1 at node
    where the load is applied

%% 7. Allocation of vectors
58 F=zeros(nDOF,1); % Spatial vector
G=zeros(numel(I_nu),1); % Poisson vector

61 %% 8. Matrix M1, K1 and K2
% Matrix of linear shape functions M and (int M)=M1
[M1] = M1(I_nu);
64 % Sizes matrices M and M1 : 1 x Ndof_nu
M1=M1';
```

```
% Computation of K1 and K2 using the constitutive
   matrices D1 and D2
Types1 = {1 'plane1', Options     % Triangular element to
    compute K1 using D1
          1 'plane2', Options};   % Triangular element to
              compute K2 using D2
Types=Types1(1,:); % plane1 element
K1=asmK1(Nodes,Elements,Types,Sections,Materials,DOF);
Types=Types1(2,:); % plane2 element
K2=asmK2(Nodes,Elements,Types,Sections,Materials,DOF);

%% 10. Enrichment step: looking for R and P
num_iter=0;
iter=zeros(1);
Aprt=0;
Error_iter=1;

while Error_iter>TOL && num_iter<num_max_iter
    num_iter = num_iter + 1;

    P0=ones(ncoords_nu,1);

    % Enrichment step
    [ R,P,iter(num_iter)] = enrichment(K1,K2,P_load,M1,P0
       ,F,G,num_iter,TOL,DOF,I_nu,Materials);

    F(:,num_iter)=R;  % F=R  - spatial mode
    G(:,num_iter)=P;  % G=P  - Poisson's mode

    % Stopping criterion : Fixed algo. point
    Error_iter=norm(F(:,num_iter)*G(:,num_iter)');
    Aprt=max(Aprt,sqrt(Error_iter));
    Error_iter = sqrt(Error_iter)/Aprt;

    % Plot PGD convergence:
    Errors(:,num_iter)=Error_iter;

    fprintf(1,'%dst iteration with an error of %f\n',
       num_iter,Error_iter);
end
num_iter=num_iter-1;
fprintf(1,'PGD off-line phase exited normally\n\n');
save('Workspace_PGD_Poisson_ratio.mat');
```

The following code shows the **on-line phase**. Running it, the user is able to digit on the keyboard the desidered Poisson's ratio and get in real time the beam configuration under unit vertical load applied at node 3.

LISTING B.8: On-line phase code for the Poisson's ratio problem (second approach).

```
%  MSc - KU Leuven - Faculty of Engineering Science
%
%                  Giuseppe D'Ettorre
%                       May 2018
%
%           Proper Generalized Method applied
%             to solid mechanincs problems
%
% Material parameters as extra problem dimension
%                     ON-LINE PHASE
%
clear all; close all; clc;
load('Workspace_PGD_Poisson_ratio'); % Loading the
   vademecum

%% 10. Post processing : text the Poisson's ratio on the
   keyboard. . .
fprintf(1,'PGD on-line phase...\n\n');
fprintf(1,'\n\nDigit on the keyboard the Poissons ratio
   between the range [0;0.5[ \n\n');
nu=input('\n\n nu = \n\n');
lim=0.005;
nu0=I_nu(1,1);
nu1=I_nu(1,ncoords_nu);

while nu0-lim<=nu && nu<=nu1+lim
     nu_chosen=find(I_nu<nu,1,'last'); % Look for the
        closest Poisson's value

     % Evaluation of the solution choosing the selected
        node in G vector
     desp=zeros(nDOF,1);
     desp=F*G(nu_chosen,:).';

    % Plotting the solution
    plotdisp(Nodes,Elements,{1 'plane3'},DOF,desp,'
       dispscal',1)
    title('Vertical displacement [m]');

    nu=input('\n\n nu = \n\n'); % Waiting for a new
```

```
        Poisson's value...
end
fprintf(1,'\n\n-------- End of simulation --------\n\n');
```

The **enrichment function** is illustrated below. It permits to find iteratively the enrichment functions **R** and **P**.

Listing B.9: Matlab code for the enrichment function.

```
function [ R,P,iter] = enrichment(K1,K2,P_load,M1,P0,F,G,
    num_iter,TOL,DOF,I_nu,Materials)
%
%   Compute a new sumand by a fixed point algorithm using
    PGD
%   function [ R,P,iter ] = enrichment(K1,K2,P_load,M1,P0
    ,F,G,num_iter,TOL,DOF,I_nu,Nodes,Elements,Types,
    Sections,Materials,Materials1)
%   returns the enrichment functions R and P
%
%   K1            Global stiffness matrix using the
    constitutive matrix D1 (DOF * DOF)
%   K2            Global stiffness matrix using the
    constitutive matrix D2 (DOF * DOF)
%   P_load        External load vector (DOF * 1)
%   M1            Integral over the load domain of M(s) (
    DOF_nu * 1)
%   P0            Random vector to initialise the
    procedure (DOF_nu * 1)
%   F             Spatial or mechanical vector (DOF * 1)
%   G             Poisson's ratio vector (DOF_nu * 1)
%   num_iter      Number of iteration (1 * 1)
%   TOL           Tolerance (1 * 1)
%   DOF           Degrees of freedom (nDOF * 1)
%   I_nu          Poisson's domain (1 * DOF_nu)
%   Materials     Material definitions [MatID MatProp1
    MatProp2 ... ]
%
%   R             Spatial or mechanical vector (DOF * 1)
%   P             Poisson's ratio vector (DOF_nu * 1)
%
%   Giuseppe D'Ettorre 2018

nDOF=size(DOF,1);
ncoords_nu=length(I_nu);
R=zeros(nDOF,1);
R0=R;
```

```
ExitFlag=1;
iter=0;
mxit=25;

while ExitFlag>TOL
    %
    % Looking for R(x) , assuming P(nu) known
    %

    % Computation LHS of equation (4.16)
    matrixR=zeros(nDOF);
    matrixR = matrixR1(K1,K2,Materials,I_nu,P0,nDOF);

    % First term RHS of equation (4.16)
    sourceR=zeros(nDOF,1);
    sourceR = P0'*M1'*P_load;

    % Second term RHS of equation (4.16)
    for i1=1:num_iter-1
        sourceR = sourceR - sourceR1(K1,K2,Materials,I_nu
           ,P0,G(:,i1),F(:,i1),nDOF);
    end

    % Now I can solve R
    R = matrixR\sourceR;

    %
    % Looking for P assuming R known
    %

    % Computation LHS of equation (4.23)
    matrixP = matrixP1(K1,K2,I_nu,R,Materials);

    sourceP=zeros(ncoords_nu,1);
    sourceP = M1'*R'*P_load;

    % Computation RHS of equation (4.23)
    for i1 = 1:num_iter-1
        sourceP = sourceP - sourceP1(K1,K2,F(:,i1),G(:,i1
           ),I_nu,R,Materials);

    end

    % Now I can solve S
    P = matrixP\sourceP;
```

```
72      P = P./norm(P);


75      %% Stopping criterion
        error=max(abs(sum(R0-R)),abs(sum(P0-P)));
        R0=R;
78      P0=P;
        iter = iter + 1;
        if iter>mxit || abs(error)<TOL,
81          return
        end
    end
84  return
```

The numerical integration of $M(\nu)$ over I_ν is performed into the following function:

Listing B.10: Matlab code for the function M1.

```
    function [M1] = M1(I_nu)
    %
3   %   function [M1] = M1(I_nu) Computes the integral of M(
        nu) over I_nu
    %
    %   I_nu     Poisson's domain (1 * DOF_nu)
6   %
    %   M1      (DOF_nu * 1)
    %
9   %   Giuseppe D'Ettorre 2018

    sg = [-0.57735 0.57735];
12  wg = [1 1];
    ngp = numel(sg);
    ncoords_nu=numel(I_nu);
15  nu1 = I_nu(1:ncoords_nu-1)';
    nu2 = I_nu(2:ncoords_nu)';
    L = nu2-nu1;
18  M1=zeros(ncoords_nu,1);

    for i1=1:ncoords_nu-1
21      w=zeros(1,ngp);
        M=zeros(ncoords_nu,ngp);
        w(1,:) = 0.5.*(1-sg).*nu1(i1) + 0.5.*(1+sg).*nu2(i1);
24      M(i1,:) = (nu2(i1)-w(1,:))./L(i1);
        M(i1+1,:) = (w(1,:)-nu1(i1))./L(i1);

27      for j1=1:ngp
```

```
        M1 = M1 + M(:,j1)*0.5.*wg(j1).*L(i1);
    end
end
return
```

Into the **enrichment function** there are four sub-functions in order to perform the numerical integrations of the equations of the two iterative steps represented by equations (4.16) and (4.23).

Listing B.11: Matlab code for the function matrixR1.

```
function [matrixR] = matrixR1(K1,K2,Materials,I_nu,P0,
   nDOF)
%
%   Compute the LHS equation (4.16)
%   function [matrixR] = matrixR1(K1,K2,Materials,I_nu,P0
   ,nDOF) returns the
%   matrixR
%
%   K1            Global stiffness matrix using the
   constitutive matrix D1 (DOF * DOF)
%   K2            Global stiffness matrix using the
   constitutive matrix D2 (DOF * DOF)
%   Materials     Material definitions [MatID MatProp1
   MatProp2 ... ]
%   I_nu          Poisson's domain (1 * DOF_nu)
%   P0            Random vector to initialise the
   procedure (DOF_nu * 1)
%   nDOF          Number degrees of freedom (1 * 1)
%
%   matrixR       (DOF * DOF)
%
%   Giuseppe D'Ettorre 2018

E=Materials(2);

sg = [-0.57735 0.57735];
wg = [1 1];
ngp=numel(sg);
ncoords_nu=numel(I_nu);
nu1=I_nu(1:ncoords_nu-1)';
nu2=I_nu(2:ncoords_nu)';
L=nu2-nu1;
nu_1=zeros(1);
nu_2=zeros(1);
matrixR=zeros(nDOF);
```

```
for i1=1:ncoords_nu-1
    w=zeros(1,ngp);
    M=zeros(ncoords_nu,ngp);
    w(1,:) = 0.5.*(1-sg).*nu1(i1) + 0.5.*(1+sg).*nu2(i1);
    M(i1,:) = (nu2(i1)-w(1,:))./L(i1);
    M(i1+1,:) = (w(1,:)-nu1(i1))./L(i1);

    for j1=1:ngp
        nu_1 = nu_1 + 0.5*(P0.'* M(:,j1)).*(E/(1-w(:,j1))
           )*(M(:,j1)'*P0)*0.5.*wg(j1).*L(i1);
        nu_2 = nu_2 + 0.5*(P0.'* M(:,j1)).*(E/(1+w(:,j1))
           )*(M(:,j1)'*P0)*0.5.*wg(j1).*L(i1);
    end
end
matrixR = K1*nu_1 + K2*nu_2;
return
```

Listing B.12: Matlab code for the function sourceR1.

```
function [sourceR] = sourceR1(K1,K2,Materials,I_nu,P0,G,F
   ,nDOF)
%
%   Compute the second term of RHS of equation (4.16)
%   function [sourceR] = sourceR1(K1,K2,Materials,I_nu,P0
   ,G,F,nDOF)
%
%   K1            Global stiffness matrix using the
   constitutive matrix D1 (DOF * DOF)
%   K2            Global stiffness matrix using the
   constitutive matrix D2 (DOF * DOF)
%   Materials     Material definitions [MatID MatProp1
   MatProp2 ... ]
%   I_nu          Poisson's domain (1 * DOF_nu)
%   P0            Random vector to initialise the
   procedure (DOF_nu * 1)
%   F             Spatial or mechanical vector (DOF * 1)
%   G             Poisson's ratio vector (DOF_nu * 1)
%   nDOF          Number degrees of freedom (1 * 1)
%
%   sourceR       (DOF * 1)
%
%   Giuseppe D'Ettorre 2018

E=Materials(2);
```

```
sg = [-0.57735 0.57735];
wg = [1 1];
ngp=numel(sg);
ncoords_nu=numel(I_nu);
nu1=I_nu(1:ncoords_nu-1)';
nu2=I_nu(2:ncoords_nu)';
L=nu2-nu1;
nu_1=zeros(1);
nu_2=zeros(1);
sourceR=zeros(nDOF);

for i1=1:ncoords_nu-1
    w=zeros(1,ngp);
    M=zeros(ncoords_nu,ngp);
    w(1,:) = 0.5.*(1-sg).*nu1(i1) + 0.5.*(1+sg).*nu2(i1);
    M(i1,:) = (nu2(i1)-w(1,:))./L(i1);
    M(i1+1,:) = (w(1,:)-nu1(i1))./L(i1);

    for j1=1:ngp
        nu_1 = nu_1 + (P0.'* M(:,j1)).*(E/(2*(1-w(:,j1)))
           )*(M(:,j1)'*G)*0.5.*wg(j1).*L(i1);
        nu_2 = nu_2 + (P0.'* M(:,j1)).*(E/(2*(1+w(:,j1)))
           )*(M(:,j1)'*G)*0.5.*wg(j1).*L(i1);
    end
end
sourceR = (K1.*nu_1 + K2.*nu_2)*F;
return
```

Listing B.13: Matlab code for the function matrixP1.

```
function [matrixP]=matrixP1(K1,K2,I_nu,R,Materials)
%
%   Compute the LHS of equation (4.23)
%   [matrixP]=matrixP1(K1,K2,I_nu,R,Materials)
%
%   K1            Global stiffness matrix using the
   constitutive matrix D1 (DOF * DOF)
%   K2            Global stiffness matrix using the
   constitutive matrix D2 (DOF * DOF)
%   Materials     Material definitions [MatID MatProp1
   MatProp2 ... ]
%   I_nu          Poisson's domain (1 * DOF_nu)
%   R             Spatial or mechanical vector (DOF * 1)
%
%   matrixP       (DOF_nu * DOF_nu)
```

```
%
%    Giuseppe D'Ettorre 2018

E=Materials(2);
sg=[-0.57735  0.57735];
wg=[1  1];
ngp=numel(sg);
ncoords_nu=numel(I_nu);
nu1=I_nu(1:ncoords_nu-1)';
nu2=I_nu(2:ncoords_nu)';
L=nu2-nu1;
matrixP=zeros(ncoords_nu,ncoords_nu);
M1=zeros(ncoords_nu);
M2=zeros(ncoords_nu);

for i1=1:ncoords_nu-1
    w=zeros(1,ngp);
    M=zeros(ncoords_nu,ngp);
    w(1,:) = 0.5.*(1-sg).*nu1(i1) + 0.5.*(1+sg).*nu2(i1);
    M(i1,:) = (nu2(i1)-w(1,:))./L(i1);
    M(i1+1,:) = (w(1,:)-nu1(i1))./L(i1);

    for j1=1:ngp
        M1 = M1 + 0.5*M(:,j1)*(E/(1-w(:,j1)))*M(:,j1)
            '*0.5.*wg(j1).*L(i1);
        M2 = M2 + 0.5*M(:,j1)*(E/(1+w(:,j1)))*M(:,j1)
            '*0.5.*wg(j1).*L(i1);
    end
end
matrixP = R'*K1*R*M1 + R'*K2*R*M2;
return
```

Listing B.14: Matlab code for the function sourceP1.

```
function [sourceP1]=sourceP1(K1,K2,F,G,I_nu,R,Materials)
%
%    Compute the second term of RHS equation (4.23)
%    function [sourceP1]=sourceP1(K1,K2,F,G,I_nu,R,
    Materials)
%
%    K1            Global stiffness matrix using the
    constitutive matrix D1 (DOF * DOF)
%    K2            Global stiffness matrix using the
    constitutive matrix D2 (DOF * DOF)
%    Materials     Material definitions [MatID MatProp1
```

```
   MatProp2 ... ]
%   I_nu          Poisson's domain (1 * DOF_nu)
%   R             Spatial or mechanical vector (DOF * 1)
%   F             Spatial or mechanical vector (DOF * 1)
%   G             Poisson's ratio vector (DOF_nu * 1)
%
%   sourceP1       (DOF_nu * 1)
%
%   Giuseppe D'Ettorre 2018

E=Materials(2);
sg=[-0.57735 0.57735];
wg=[1 1];
ngp=numel(sg);
ncoords_nu=numel(I_nu);
nu1=I_nu(1:ncoords_nu-1)';
nu2=I_nu(2:ncoords_nu)';
L=nu2-nu1;
sourceP1=zeros(ncoords_nu,ncoords_nu);
M1=zeros(ncoords_nu);
M2=zeros(ncoords_nu);

for i1=1:ncoords_nu-1
    w=zeros(1,ngp);
    M=zeros(ncoords_nu,ngp);
    w(1,:) = 0.5.*(1-sg).*nu1(i1) + 0.5.*(1+sg).*nu2(i1);
    M(i1,:) = (nu2(i1)-w(1,:))./L(i1);
    M(i1+1,:) = (w(1,:)-nu1(i1))./L(i1);

    for j1=1:ngp
        M1 = M1 + M(:,j1)*(E/(2*(1-w(:,j1))))*M(:,j1)
           '*0.5.*wg(j1).*L(i1);
        M2 = M2 + M(:,j1)*(E/(2*(1+w(:,j1))))*M(:,j1)
           '*0.5.*wg(j1).*L(i1);
    end
end
sourceP1 = (R'*K1*F*M1 + R'*K2*F*M2)*G;
return
```