Chapter 3
Parametric Problems

We have a habit in writing articles published in scientific journals to make the work as finished as possible, to cover all the tracks, to not worry about the blind alleys or to describe how you had the wrong idea first, and so on. So there isn't any place to publish, in a dignified manner, what you actually did in order to get to do the work.

—Richard Feynmann

Abstract This chapter develops the application of PGD methods for parametric problems, their natural field of application.

Parametric problems constitute perhaps the most relevant application of Proper Generalized Decomposition methods. Initially aimed at solving problems defined within a high dimensional phase space [6], PGD soon revealed an impressive ability to solve in the same setting parametric problems by just considering parameters as new dimensions, a sort of parametric phase space [27].

In this chapter we explore precisely these capabilities. Since they have also been included in a number of previous references, notably in former books (see [28], Chap. 5), we focus here on a particular instance of parametric problems: that of moving loads.

3.1 A Particularly Challenging Problem: A Moving Load as a Parameter

An *influence line* is a graphical representation of a given magnitude (a bending moment, for instance) at a given point of a beam caused by a unit load moving along the span of that beam. This concept has helped engineers through the years to design beam structures in an efficient manner. One can imagine, however, an extension of the concept of influence line to general, three-dimensional solids. This

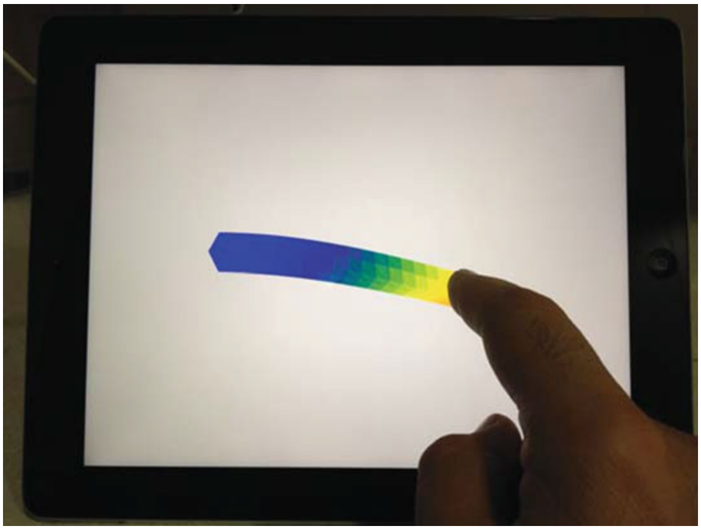

Fig. 3.1 An application of the influence line concept for a hyperelastic beam. You can play interactively with the beam by just placing the load with your finger on a tablet. A web-based version of this same app can be reached at http://amb.unizar.es/barra03.htm

would give rise to a sort of *response surface* in which particularizing the parameter (the position of the load) provides in an immediate way the response of the solid (its deformed configuration, in this case). This sort of influence line has potentially many applications in science and engineering for real-time simulation in fields such as computational surgery [55], decision taking, or even augmented learning [60], see Fig. 3.1.

In fact, this problem has been frequently thought of as *non separable*, i.e., that the number of modes needed to express the solution is so big, that no gain is obtained by applying any kind of model order reduction technique and therefore it is better to simply simulate it in a straightforward manner, by finite element methods or any other numerical technique of your choice.

However, it can be easily found that this is not true. Consider the influence line sketched in Fig. 3.2. We consider a clamped beam with a moving load and try to compute its deformed configuration for the load acting at any point. In fact, by applying Proper Orthogonal Decomposition techniques to the results, it can readily be seen that the number of modes or shape functions needed to express this solution $v = v(x, s)$ is in fact limited, see Fig. 3.3. Here, v denotes here the vertical displacement of the beam, x the particular point of the beam in which we want to know this displacement and s the position of the load,

As can be noticed from Fig. 3.3 (left), the eigenvalues decrease fast after a reasonable number of eigenmodes. Thus, even if we need to consider every possible load position, the number of functions needed to express the parametric solution is in fact reasonably limited.

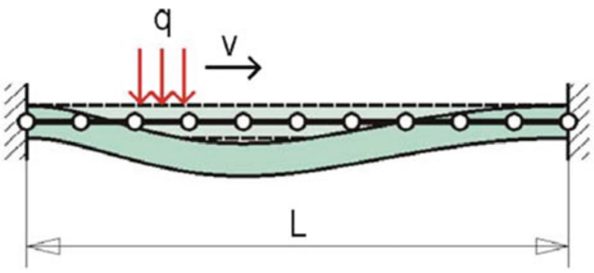

Fig. 3.2 A clamped beam with a moving load

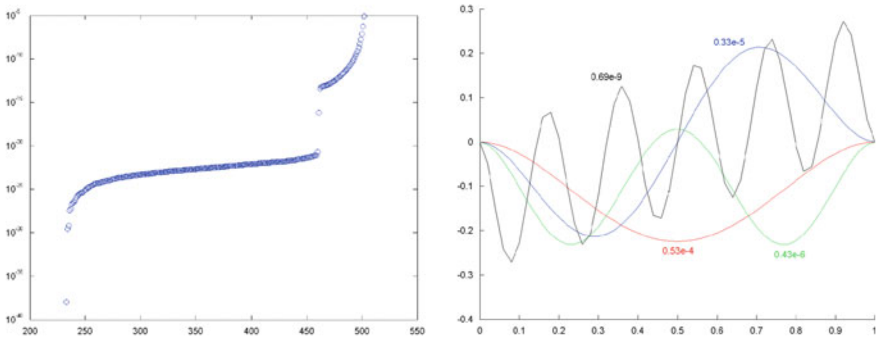

Fig. 3.3 (*Left*) resulting eigenvalues for the clamped beam problem and (*right*) first modes of the solution

3.2 The Problem Under the PGD Formalism

As will be noticed, under the PGD formalism, the *influence line* problem becomes simple. We generalize this problem so as to find the displacement field $\boldsymbol{u}(x, y, z)$ at any point of a three-dimensional solid Ω, for any load position $\boldsymbol{s} \in \bar{\Gamma} \subset \partial\Omega$.

Under these assumptions, the weak form of the problem will result after multiplying the strong form by an arbitrary test function $\boldsymbol{u}^*$ and integrating over the region occupied by the solid, Ω, and the portion of its boundary which is accessible to the load, $\bar{\Gamma}$. It therefore consists in finding the displacement $\boldsymbol{u} \in \mathcal{H}^1$ such that for all $\boldsymbol{u}^* \in \mathcal{H}^1_0$:

$$\int_{\bar{\Gamma}} \int_{\Omega} \nabla_s \boldsymbol{u}^* : \boldsymbol{\sigma} \, d\Omega d\bar{\Gamma} = \int_{\bar{\Gamma}} \int_{\Gamma_{t2}} \boldsymbol{u}^* \cdot \boldsymbol{t} \, d\Gamma d\bar{\Gamma} \tag{3.1}$$

where $\Gamma = \Gamma_u \cup \Gamma_t$ represents the essential and natural portions of the boundary, and where $\Gamma_t = \Gamma_{t1} \cup \Gamma_{t2}$, i.e., regions of homogeneous and non-homogeneous, respectively, natural boundary conditions. We assume, for simplicity of the exposition, that the load is of unity module and acts along the vertical axis: $\boldsymbol{t} = \boldsymbol{e}_k \cdot \delta(\boldsymbol{x} - \boldsymbol{s})$, where δ represents the Dirac-delta function and $\boldsymbol{e}_k$ the unit vector along the z-coordinate axis.

The Dirac-delta term needs to be approximated by a truncated series of separable functions in the spirit of the PGD method, i.e.,

$$t_j \approx \sum_{i=1}^{m} f_j^i(\boldsymbol{x}) g_j^i(\boldsymbol{s}) \tag{3.2}$$

where m represents the order of truncation and f_j^i, g_j^i represent the j-th component of vectorial functions in space and boundary position, respectively.

The PGD approach to the problem consists in finding, in a greedy way, a finite sum of separable functions to approach the solution. Assuming that, at iteration n of this procedure, we have converged to such an approximation, we have

$$u_j^n(\boldsymbol{x}, \boldsymbol{s}) = \sum_{k=1}^{n} F_j^k(\boldsymbol{x}) \cdot G_j^k(\boldsymbol{s}), \tag{3.3}$$

where the term u_j refers to the jth component of the displacement vector, $j = 1, 2, 3$ and functions $\boldsymbol{F}^k$ and $\boldsymbol{G}^k$ represent the separated functions used to approximate the unknown field, obtained in previous iterations of the PGD algorithm.

If this rank-n approximation does not give the desired accuracy, we can proceed further and look for the $(n + 1)$th term, that will look like

$$u_j^{n+1}(\boldsymbol{x}, \boldsymbol{s}) = u_j^n(\boldsymbol{x}, \boldsymbol{s}) + R_j(\boldsymbol{x}) \cdot S_j(\boldsymbol{s}),$$

where $\boldsymbol{R}(\boldsymbol{x})$ and $\boldsymbol{S}(\boldsymbol{s})$ are the sought functions that improve the approximation.

By just applying standard rules of variational calculus, the test function will be

$$u_j^*(\boldsymbol{x}, \boldsymbol{s}) = R_j^*(\boldsymbol{x}) \cdot S_j(\boldsymbol{s}) + R_j(\boldsymbol{x}) \cdot S_j^*(\boldsymbol{s}).$$

To determine the new functions $\boldsymbol{R}$ and $\boldsymbol{S}$ any linearization strategy could in principle be applied. In our experience, a fixed-point algorithm in which functions $\boldsymbol{R}$ and $\boldsymbol{S}$ are determined iteratively gives very good results, even if convergence is not guaranteed, as is well known. We provide details of this strategy in the next section.

3.2.1 Computation of $\boldsymbol{S}(\boldsymbol{s})$ Assuming $\boldsymbol{R}(\boldsymbol{x})$ Is Known

In this case, the admissible variation of the displacement will be given by

$$u_j^*(\boldsymbol{x}, \boldsymbol{s}) = R_j(\boldsymbol{x}) \cdot S_j^*(\boldsymbol{s}),$$

or, equivalently, $\boldsymbol{u}^*(\boldsymbol{x}, \boldsymbol{s}) = \boldsymbol{R} \circ \boldsymbol{S}^*$, where the symbol "$\circ$" denotes the so-called entry-wise, Hadamard or Schur multiplication for vectors. Once substituted into Eq. (3.1), gives

$$\int_{\bar{\Gamma}}\int_{\Omega} \nabla_s(\boldsymbol{R}\circ\boldsymbol{S}^*) : \mathbf{C} : \nabla_s\left(\sum_{k=1}^{n} \boldsymbol{F}^k\circ\boldsymbol{G}^k + \boldsymbol{R}\circ\boldsymbol{S}\right) d\Omega d\bar{\Gamma} = \int_{\bar{\Gamma}}\int_{\Gamma_{t2}} (\boldsymbol{R}\circ\boldsymbol{S}^*)\cdot\left(\sum_{k=1}^{m} \boldsymbol{f}^k\circ\boldsymbol{g}^k\right) d\Gamma d\bar{\Gamma}, \tag{3.4}$$

or, simply

$$\begin{aligned}&\int_{\bar{\Gamma}}\int_{\Omega} \nabla_s(\boldsymbol{R}\circ\boldsymbol{S}^*) : \mathbf{C} : \nabla_s(\boldsymbol{R}\circ\boldsymbol{S}) d\Omega d\bar{\Gamma}\\ &= \int_{\bar{\Gamma}}\int_{\Gamma_{t2}} (\boldsymbol{R}\circ\boldsymbol{S}^*)\cdot\left(\sum_{k=1}^{m} \boldsymbol{f}^k\circ\boldsymbol{g}^k\right) d\Gamma d\bar{\Gamma} - \int_{\bar{\Gamma}}\int_{\Omega} \nabla_s\left(\boldsymbol{R}\circ\boldsymbol{S}^*\right)\cdot\mathcal{R}^n d\Omega d\bar{\Gamma},\end{aligned}$$

where $\mathcal{R}^n$ represents:

$$\mathcal{R}^n = \mathbf{C} : \nabla_s \boldsymbol{u}^n.$$

Since the symmetric gradient operates on spatial variables only, we have:

$$\begin{aligned}&\int_{\bar{\Gamma}}\int_{\Omega} (\nabla_s\boldsymbol{R}\circ\boldsymbol{S}^*) : \mathbf{C} : (\nabla_s\boldsymbol{R}\circ\boldsymbol{S}) d\Omega d\bar{\Gamma}\\ &= \int_{\bar{\Gamma}}\int_{\Gamma_{t2}} (\boldsymbol{R}\circ\boldsymbol{S}^*)\cdot\left(\sum_{k=1}^{m} \boldsymbol{f}^k\circ\boldsymbol{g}^k\right) d\Gamma d\bar{\Gamma} - \int_{\bar{\Gamma}}\int_{\Omega} \left(\nabla_s\boldsymbol{R}\circ\boldsymbol{S}^*\right)\cdot\mathcal{R}^n d\Omega d\bar{\Gamma}\end{aligned}$$

where all the terms depending on $\boldsymbol{x}$ are known. Therefore, all integrals over Ω and Γ_{t2} (support of the regularization of the initially punctual load) can be computed to obtain an equation for $\boldsymbol{S}(\boldsymbol{s})$.

3.2.2 Computation of $\boldsymbol{R}(\boldsymbol{x})$ Assuming $\boldsymbol{S}(\boldsymbol{s})$ Is Known

By proceeding in the same way, we have

$$u_j^*(\boldsymbol{x},\boldsymbol{s}) = R_j^*(\boldsymbol{x})\cdot S_j(\boldsymbol{s}),$$

which, once substituted into Eq. (3.1), gives

$$\int_{\bar{\Gamma}}\int_{\Omega} \nabla_s(\boldsymbol{R}^*\circ\boldsymbol{S}) : \mathbf{C} : \nabla_s\left(\sum_{k=1}^{n} \boldsymbol{F}^k\circ\boldsymbol{G}^k + \boldsymbol{R}\circ\boldsymbol{S}\right) d\Omega d\bar{\Gamma} = \int_{\bar{\Gamma}}\int_{\Gamma_{t2}} (\boldsymbol{R}^*\circ\boldsymbol{S})\cdot\left(\sum_{k=1}^{m} \boldsymbol{f}^k\circ\boldsymbol{g}^k\right) d\Gamma d\bar{\Gamma}.$$

Conversely, all the terms depending on $\boldsymbol{s}$ (load position) can be integrated over $\bar{\Gamma}$, leading to a generalized elastic problem to compute function $\boldsymbol{R}(\boldsymbol{x})$.

3.3 Matrix Structure of the Problem

As stated before, see Eq. (3.3), the essential ingredient of PGD methods is to assume the variable (here, the displacement field) to be decomposed in the form of a finite sum of separable functions, i.e.,

$$\boldsymbol{u}(\boldsymbol{x}, \boldsymbol{s}) = \sum_{i=1}^{n} \boldsymbol{F}^i(\boldsymbol{x}) \circ \boldsymbol{G}^i(\boldsymbol{s}),$$

where a dependency on the physical position of the considered point, $\boldsymbol{x}$ and the position of the applied load, $\boldsymbol{s}$ is assumed.

In the implementation introduced in Sect. 3.4 below, we enforce functions $\boldsymbol{G}^i$ to have unity norm. By doing that, the weighting parameter α_i (see Eq. (2.2) is not computed explicitly, but assumed to multiply the $\boldsymbol{F}^i$ functions (which, therefore, do not have unity norm).

As in the previous chapter, $\boldsymbol{F}^i(\boldsymbol{x})$ and $\boldsymbol{G}^i(\boldsymbol{s})$ are approximated by employing (linear in this case) finite elements, so that, at iteration n, the i-th sum of the approximation will be given by

$$\boldsymbol{u}^h(\boldsymbol{x}, \boldsymbol{s}) = \sum_{i=1}^{n} \boldsymbol{F}^i(\boldsymbol{x}) \cdot \boldsymbol{G}^i(\boldsymbol{s}) = \sum_{i=1}^{n} \left[\boldsymbol{N}^T(\boldsymbol{x})\boldsymbol{F}^i \boldsymbol{M}^T(\boldsymbol{s})\boldsymbol{G}^i\right],$$

with $\boldsymbol{N}$ and $\boldsymbol{M}$ the vectors containing the finite element shape functions defined in each separated space and $\boldsymbol{F}^i$ and $\boldsymbol{G}^i$ the vectors of nodal values at the FE mesh for the functions $\boldsymbol{F}^i(\boldsymbol{x})$ and $\boldsymbol{G}^i(\boldsymbol{s})$, respectively. For simplicity and ease of reading, we have maintained the same notation employed in Eq. (2.8). The superscript h for the discrete, finite element approximation of a variable will no longer be employed, for clarity, if there is no risk of confusion.

The code included below solves the problem of a two-dimensional cantilever beam under a load placed at an arbitrary position along its upper face. Therefore, we assume a 2D spatial approximation on $\boldsymbol{x}$ (given by simple linear triangular elements) and 1D approximation for the $\boldsymbol{s}$ direction.

When we look for a new couple of functions in the approximation, we assume that

$$\boldsymbol{u}^{n+1}(\boldsymbol{x}, \boldsymbol{s}) = \boldsymbol{u}^n(\boldsymbol{x}, \boldsymbol{s}) + \boldsymbol{R}(\boldsymbol{x})\boldsymbol{S}(\boldsymbol{s}) = \sum_{i=1}^{n} \boldsymbol{F}^i(\boldsymbol{x})\boldsymbol{G}^i(\boldsymbol{s}) + \boldsymbol{R}(\boldsymbol{x})\boldsymbol{S}(\boldsymbol{s}), \tag{3.5}$$

while

$$\boldsymbol{u}^*(\boldsymbol{x}, \boldsymbol{s}) = \boldsymbol{R}^*(\boldsymbol{x})\boldsymbol{S}(\boldsymbol{s}) + \boldsymbol{R}(\boldsymbol{x})\boldsymbol{S}^*(\boldsymbol{s}).$$

Very often, in parametric problems (like in this example) the derivatives appearing in the weak form of the problem, given by Eq. (3.4), are acting only spatial coordinates and not on the parameters, despite that in the PGD framework they are considered as actual “extra” coordinates. Therefore,

$$\nabla_s \boldsymbol{u} = \nabla_s \left[\sum_{i=1}^{n} \boldsymbol{F}^i(\boldsymbol{x}) \cdot \boldsymbol{G}^i(\boldsymbol{s}) \right] + \nabla_s \left[\boldsymbol{R}(\boldsymbol{x}) \cdot \boldsymbol{S}(\boldsymbol{s}) \right] = \sum_{i=1}^{n} \left[\nabla_s \boldsymbol{F}^i(\boldsymbol{x}) \right] \cdot \boldsymbol{G}^i(\boldsymbol{s}) + \left[\nabla_s \boldsymbol{R}(\boldsymbol{x}) \right] \cdot \boldsymbol{S}(\boldsymbol{s})$$
$$= \sum_{i=1}^{n} \left[\nabla_s \boldsymbol{N}^T(\boldsymbol{x}) \boldsymbol{F}^i \right] \cdot \boldsymbol{M}^T(\boldsymbol{s}) \boldsymbol{G}^i + \left[\nabla_s \boldsymbol{N}^T(\boldsymbol{x}) \boldsymbol{R} \right] \cdot \boldsymbol{M}^T(\boldsymbol{s}) \boldsymbol{S}.$$

The terms defined before and depending on nodal values $\boldsymbol{F}_i$ and $\boldsymbol{G}_i$ (i.e., those corresponding to the already computed $\boldsymbol{u}^n$ approximation) are known, so they should be moved to the right-hand side of the final algebraic equation.

In a similar way,

$$\nabla_s \boldsymbol{u}^* = \nabla_s \boldsymbol{N}^T(\boldsymbol{x}) \boldsymbol{R} \cdot \boldsymbol{M}^T(\boldsymbol{s}) \boldsymbol{S}^* + \nabla_s \boldsymbol{N}^T(\boldsymbol{x}) \boldsymbol{R}^* \cdot \boldsymbol{M}^T(\boldsymbol{s}) \boldsymbol{S}.$$

The L.H.S. of the weak form, Eq. (3.4), can be easily expressed in separated form,

$$\begin{aligned} &\int_{\bar{\Gamma}} \int_{\Omega} \nabla_s \boldsymbol{u}^* : \mathbf{C} : \nabla_s \boldsymbol{R}(\boldsymbol{x}) \boldsymbol{S}(\boldsymbol{s}) d\Omega d\bar{\Gamma} \\ &= \int_{\Omega} \boldsymbol{R}^T \nabla_s \boldsymbol{N}(\boldsymbol{x}) : \mathbf{C} : \nabla_s \boldsymbol{N}^T(\boldsymbol{x}) \boldsymbol{R} d\Omega \cdot \int_{\bar{\Gamma}} \boldsymbol{S}^{*T} \boldsymbol{M}(\boldsymbol{s}) \boldsymbol{M}^T(\boldsymbol{s}) \boldsymbol{S} d\bar{\Gamma} \\ &\quad + \int_{\Omega} \boldsymbol{R}^{*T} \nabla_s \boldsymbol{N}(\boldsymbol{x}) : \mathbf{C} : \nabla_s \boldsymbol{N}^T(\boldsymbol{x}) \boldsymbol{R} d\Omega \cdot \int_{\bar{\Gamma}} \boldsymbol{S}^T \boldsymbol{M}(\boldsymbol{s}) \boldsymbol{M}^T(\boldsymbol{s}) \boldsymbol{S} d\bar{\Gamma} \end{aligned} \tag{3.6}$$

Since $\boldsymbol{R}$ and $\boldsymbol{S}$ represent vectors of nodal values for functions $\boldsymbol{R}(\boldsymbol{x})$ and $\boldsymbol{S}(\boldsymbol{s})$, respectively, they can be extracted from the integrals in Eq. (3.6), so as to give,

$$\begin{aligned} &\int_{\bar{\Gamma}} \int_{\Omega} \nabla_s \boldsymbol{u}^* : \mathbf{C} : \nabla_s \boldsymbol{R}(\boldsymbol{x}) \boldsymbol{S}(\boldsymbol{s}) d\Omega d\bar{\Gamma} \\ &= \boldsymbol{R}^T \left[\int_{\Omega} \nabla_s \boldsymbol{N}(\boldsymbol{x}) : \mathbf{C} : \nabla_s \boldsymbol{N}^T(\boldsymbol{x}) d\Omega \right] \boldsymbol{R} \cdot \boldsymbol{S}^{*T} \left[\int_{\bar{\Gamma}} \boldsymbol{M}(\boldsymbol{s}) \boldsymbol{M}^T(\boldsymbol{s}) \right] \boldsymbol{S} d\bar{\Gamma} \\ &\quad + \boldsymbol{R}^{*T} \left[\int_{\Omega} \nabla_s \boldsymbol{N}(\boldsymbol{x}) : \mathbf{C} : \nabla_s \boldsymbol{N}^T(\boldsymbol{x}) d\Omega \right] \boldsymbol{R} \cdot \boldsymbol{S}^T \left[\int_{\bar{\Gamma}} \boldsymbol{M}(\boldsymbol{s}) \boldsymbol{M}^T(\boldsymbol{s}) d\bar{\Gamma} \right] \boldsymbol{S} \\ &= \boldsymbol{R}^T \boldsymbol{K1}(\boldsymbol{x}) \boldsymbol{R} \cdot \boldsymbol{S}^{*T} \boldsymbol{M2}(\boldsymbol{s}) \boldsymbol{S} + \boldsymbol{R}^{*T} \boldsymbol{K1}(\boldsymbol{x}) \boldsymbol{R} \cdot \boldsymbol{S}^T \boldsymbol{M2}(\boldsymbol{s}) \boldsymbol{S} \end{aligned} \tag{3.7}$$

where $\boldsymbol{K1}(\boldsymbol{x})$ represents a sort of stiffness matrix on the spatial coordinates $\boldsymbol{x}$, while $\boldsymbol{M2}(\boldsymbol{s})$ represents a mass matrix for the 1D discretization problem in the $\boldsymbol{s}$ variable. Both of them are computed in routine `elemstiff` (see the call `[K1,M2] = elemstiff(coors)` in the main file of the code in Sect. 3.4).

We proceed similarly for the RHS term of the weak form, Eq. (3.4), and taking into account that the integrals involved in it do not depend on spatial coordinates, but only on the $\boldsymbol{s}$ coordinate, we arrive at

$$\begin{aligned}\int_{\bar{\Gamma}}\int_{\Gamma_{t2}} \boldsymbol{u}^* \cdot \boldsymbol{t} d\Gamma d\bar{\Gamma} &= \int_{\bar{\Gamma}}\int_{\Gamma_{t2}} \left[\boldsymbol{R}^*(\boldsymbol{x})\boldsymbol{S}(\boldsymbol{s}) + \boldsymbol{R}(\boldsymbol{x})\boldsymbol{S}^*(\boldsymbol{s})\right] \cdot \left[\sum_{i=1}^{m} \boldsymbol{f}^i(\boldsymbol{x})\boldsymbol{g}^i(\boldsymbol{s})\right] d\Gamma d\bar{\Gamma} \\ &= \int_{\Gamma_{t2}} \sum_{i=1}^{m} \boldsymbol{R}^*(\boldsymbol{x})\boldsymbol{f}^i(\boldsymbol{x}) \int_{\bar{\Gamma}} \boldsymbol{S}(\boldsymbol{s})\boldsymbol{g}^i(\boldsymbol{s}) d\bar{\Gamma} + \int_{\Gamma_{t2}} \sum_{i=1}^{m} \boldsymbol{R}(\boldsymbol{x})\boldsymbol{f}^i(\boldsymbol{x}) \int_{\bar{\Gamma}} \boldsymbol{S}^*(\boldsymbol{s})\boldsymbol{g}^i(\boldsymbol{s}) d\bar{\Gamma}.\end{aligned} \tag{3.8}$$

Denoting by $\boldsymbol{FR1}$ the nodal values of the source term decomposition for the spatial variables and by $\boldsymbol{FR2}$ for the load direction, Eq. (3.8) has the form,

$$\sum_{i=1}^{m} \boldsymbol{R^*}^T \boldsymbol{FR1} \cdot \boldsymbol{S}^T \left[\int_{\bar{\Gamma}} \boldsymbol{M}(\boldsymbol{s})\boldsymbol{M}^T(\boldsymbol{s}) d\bar{\Gamma}\right] \boldsymbol{FR2} + \sum_{i=1}^{m} \boldsymbol{R}^T \boldsymbol{FR1} \cdot \boldsymbol{S^*}^T \left[\int_{\bar{\Gamma}} \boldsymbol{M}(\boldsymbol{s})\boldsymbol{M}^T(\boldsymbol{s}) d\bar{\Gamma}\right] \boldsymbol{FR2}. \tag{3.9}$$

As discussed in the introduction to this chapter, the problem of expressing the moving load as a finite sum of separable functions, i.e., $\boldsymbol{t}(\boldsymbol{x}, \boldsymbol{s}) \approx \sum_{i=1}^{m} \boldsymbol{f}^i(\boldsymbol{x})\boldsymbol{g}^i(\boldsymbol{s})$ is not separable. In other words, it can only be done in the discrete setting by choosing $\boldsymbol{FR1}$ to be a matrix of zeros (of size [# `dof of the whole problem` × `dof along the loaded boundary`] and whose only non-vanishing entries are, for each column, the position of nodes that can receive a load. These entries will be equal to the value of the applied force. Respectively, $\boldsymbol{FR2}$ will be an identity matrix (`eye(# dof along the loaded boundary)` in Matlab terms). Each column in $\boldsymbol{FR1}$ thus includes the corresponding degree of freedom that is loaded. Note that, therefore,

$$\sum_{i=1}^{m} \boldsymbol{f}^i(\boldsymbol{x})\boldsymbol{g}^i(\boldsymbol{s}) = \sum_{i=1}^{m} \boldsymbol{N}(\boldsymbol{x})\boldsymbol{FR1}^i \cdot \boldsymbol{M}(\boldsymbol{s})\boldsymbol{FR2}^i.$$

To this R.H.S. vector we should add the terms related to the known solution up to iteration n, $\boldsymbol{u}^n(\boldsymbol{x}, \boldsymbol{s})$, Eq. (3.5),

$$\begin{aligned}
&\int_{\bar{\Gamma}} \int_{\Omega} \nabla_s \boldsymbol{u}^* : \mathbf{C} : \nabla_s \boldsymbol{u}^n d\Omega d\bar{\Gamma} \\
&= \sum_{i=1}^{n} \boldsymbol{R}^T \left[\int_{\Omega} \nabla_s \boldsymbol{N}(\boldsymbol{x}) : \mathbf{C} : \nabla_s \boldsymbol{N}^T(\boldsymbol{x}) d\Omega \right] \boldsymbol{F}^i \cdot \boldsymbol{S}^{*T} \left[\int_{\bar{\Gamma}} \boldsymbol{M}(\boldsymbol{s}) \boldsymbol{M}^T(\boldsymbol{s}) \right] \boldsymbol{G}^i d\bar{\Gamma} + \\
&\quad + \sum_{i=1}^{n} \boldsymbol{R}^{*T} \left[\int_{\Omega} \nabla_s \boldsymbol{N}(\boldsymbol{x}) : \mathbf{C} : \nabla_s \boldsymbol{N}^T(\boldsymbol{x}) d\Omega \right] \boldsymbol{F}^i \cdot \boldsymbol{S}^T \left[\int_{\bar{\Gamma}} \boldsymbol{M}(\boldsymbol{s}) \boldsymbol{M}^T(\boldsymbol{s}) d\bar{\Gamma} \right] \boldsymbol{G}^i \\
&= \sum_{i=1}^{n} \boldsymbol{R}^T \boldsymbol{K1}(\boldsymbol{x}) \boldsymbol{F}^i \cdot \boldsymbol{S}^{*T} \boldsymbol{M2}(\boldsymbol{s}) \boldsymbol{G}^i + \sum_{i=1}^{n} \boldsymbol{R}^{*T} \boldsymbol{K1}(\boldsymbol{x}) \boldsymbol{F}^i \cdot \boldsymbol{S}^T \boldsymbol{M2}(\boldsymbol{s}) \boldsymbol{G}^i .
\end{aligned} \quad (3.10)$$

In the Matlab code included in Sect. 3.4 we are employing the following notation: $\boldsymbol{V1}^i = \boldsymbol{FR1}^i$ and $\boldsymbol{V2}^i = \boldsymbol{M2}(\boldsymbol{s})\boldsymbol{FR2}^i$ for $i = 1, \ldots, m$, so that Eq. (3.9) is expressed in the form

$$\sum_{i=1}^{m} \boldsymbol{R}^{*T} \boldsymbol{V1} \cdot \boldsymbol{S}^T \boldsymbol{V2} + \sum_{i=1}^{m} \boldsymbol{R}^T \boldsymbol{V1} \cdot \boldsymbol{S}^{*T} \boldsymbol{V2}. \quad (3.11)$$

After some gymnastics, the weak form, Eq. (3.4), taking into account Eqs. (3.7), (3.10) and (3.11), will look in matrix form like

$$\begin{aligned}
&\boldsymbol{R}^T \boldsymbol{K1}(\boldsymbol{x}) \boldsymbol{R} \cdot \boldsymbol{S}^{*T} \boldsymbol{M2}(\boldsymbol{s}) \boldsymbol{S} + \boldsymbol{R}^{*T} \boldsymbol{K1}(\boldsymbol{x}) \boldsymbol{R} \cdot \boldsymbol{S}^T \boldsymbol{M2}(\boldsymbol{s}) \boldsymbol{S} = \\
&= \sum_{i=1}^{m} \boldsymbol{R}^{*T} \boldsymbol{V1} \cdot \boldsymbol{S}^T \boldsymbol{V2} + \sum_{i=1}^{m} \boldsymbol{R}^T \boldsymbol{V1} \cdot \boldsymbol{S}^{*T} \boldsymbol{V2} - \sum_{i=1}^{n} \boldsymbol{R}^T \boldsymbol{K1}(\boldsymbol{x}) \boldsymbol{F}^i \cdot \boldsymbol{S}^{*T} \boldsymbol{M2}(\boldsymbol{s}) \boldsymbol{G}^i \\
&\quad - \sum_{i=1}^{n} \boldsymbol{R}^{*T} \boldsymbol{K1}(\boldsymbol{x}) \boldsymbol{F}^i \cdot \boldsymbol{S}^T \boldsymbol{M2}(\boldsymbol{s}) \boldsymbol{G}^i .
\end{aligned}$$

As mentioned before, in order to determine the new functional pair $\boldsymbol{R}$ and $\boldsymbol{S}$, any linearization strategy could be envisaged. Remember that the fact that we look for a product of functions makes the problem of enriching the approximation non-linear. In the code of Sect. 3.4, a fixed-point algorithm in which functions $\boldsymbol{R}$ and $\boldsymbol{S}$ are sought iteratively is implemented. Other strategies, like Newton-Raphson, for instance, could work equally well.

The final matrix form of the fixed-point alternating directions algorithm, in which the computation of $\boldsymbol{S}(\boldsymbol{s})$ is performed, assuming that $\boldsymbol{R}(\boldsymbol{x})$ is known, will look like

$$\boldsymbol{R}^T\boldsymbol{K1}(\boldsymbol{x})\boldsymbol{R}\cdot\boldsymbol{S}^{*T}\boldsymbol{M2}(\boldsymbol{s})\boldsymbol{S}=\sum_{i=1}^{m}\boldsymbol{R}^T\boldsymbol{V1}\cdot\boldsymbol{S}^{*T}\boldsymbol{V2}-\sum_{i=1}^{n}\boldsymbol{R}^T\boldsymbol{K1}(\boldsymbol{x})\boldsymbol{F}^i\cdot\boldsymbol{S}^{*T}\boldsymbol{M2}(\boldsymbol{s})\boldsymbol{G}^i. \tag{3.12}$$

Equivalently, when we look for $\boldsymbol{R}(\boldsymbol{x})$ assuming $\boldsymbol{S}(\boldsymbol{s})$ is known, the resulting problem will have the following matrix form,

$$\boldsymbol{R}^{*T}\boldsymbol{K1}(\boldsymbol{x})\boldsymbol{R}\cdot\boldsymbol{S}^T\boldsymbol{M2}(\boldsymbol{s})\boldsymbol{S}=\sum_{i=1}^{m}\boldsymbol{R}^{*T}\boldsymbol{V1}\cdot\boldsymbol{S}^T\boldsymbol{V2}-\sum_{i=1}^{n}\boldsymbol{R}^{*T}\boldsymbol{K1}(\boldsymbol{x})\boldsymbol{F}^i\cdot\boldsymbol{S}^T\boldsymbol{M2}(\boldsymbol{s})\boldsymbol{G}^i. \tag{3.13}$$

In next section the detailed Matlab code implementing this strategy is provided.

3.4 Matlab Code for the Influence Line Problem

As always, the code begins at file `main.m`, whose content is reproduced below. It solves the problem of a cantilever beam under a load placed at an arbitrary location along its top boundary. Small strains assumption is made. The code provides the solution under plane stress or plane strain conditions.

```
%
%                      PGD Code for parametrized force
%                     D. Gonzalez, I. Alfaro, E. Cueto
%                         Universidad de Zaragoza
%                           AMB-I3A Dec 2015
%
clear all; close all; clc;
%
% VARIABLES
%
global coords triangles E nu behaviour % Global variables.
E = 1000; nu = 0.3; % Material (Young Modulus and Poisson Coef)
Modulus = 1; % Force Modulus.
behaviour = 1; % Plane Stress(1), Plane Strain(2).
TOL = 1.0E-03; % Tolerance.
num_max_iter = 11; % Max. # of functional pairs for the approximation.
%
% GEOMETRY
%
X0 = 0; Xf = 3; Y0 = 0; Yf = 1.0; % The beam dimensions, [0,3]x[0,1]
tamx = 0.1; tamy = 0.1; %  mesh size along each direction
nenY = numel(Y0:tamy:Yf); % # of elements in vertical direction
[X,Y] = meshgrid(X0:tamx:Xf,Y0:tamy:Yf);
coords = [X(:),Y(:)];
force = X0:tamx:Xf; force = force'; % force positions and 1D coordinates.
triangles = delaunayTriangulation(coords(:,1),coords(:,2)); % Mesh data.
%
% ALLOCATION OF MATRICES AND VECTORS
%
F = zeros(numel(coords),1); % Nodal values of spatial function F
```

```
G = zeros(numel(force),1); % Nodal values of force function G
FR1 = zeros(numel(coords),numel(force)); % Nodal values for force (spatial term
FR2 = eye(numel(force)); % Nodal values for force (force term)
%
% COMPUTING STIFFNESS AND MASS MATRIX FOR SPACE, ONLY MASS MATRIX FOR FORCE
%
[K1,M2] = elemstiff(force);
%
% SOURCE (FORCE) TERM IN SEPARATED FORM
%
DOFforceed = 2*nenY:2*nenY:numel(coords); % force on vertical d.o.f. on top.
for i1=1:numel(DOFforceed)
    FR1(DOFforceed(i1),i1) = -Modulus;
end
V1 = FR1; % Take into account that integration is done only in S direction
V2 = M2*FR2; % Mass matrix times nodal value of the source. R.H.S. of Eq.(3.9)
%
% BOUNDARY CONDITIONS
%
CC = 1:2*(nenY); % Left side of the beam fixed.
%
% ENRICHMENT OF THE APPROXIMATION, LOOKING FOR R AND S
%
num_iter = 0; iter = zeros(1); Aprt = 0; Error_iter = 1.0;
while Error_iter>TOL && num_iter<num_max_iter
    num_iter = num_iter + 1;
    S0 = rand(numel(force),1); % Initial guess for S.
    %
    % ENRICHMENT STEP
    %
    [R,S,iter(num_iter)] = enrichment(K1,M2,V1,V2,S0,F,G,num_iter,TOL,CC);
    F(:,num_iter) = R; G(:,num_iter) = S; % R and S are valid, new summand.
    %
    % STOPPING CRITERION
    %
    Error_iter = norm(F(:,num_iter)*G(:,num_iter)');
    Aprt = max(Aprt,sqrt(Error_iter));
    Error_iter = sqrt(Error_iter)/Aprt;
    fprintf(1,'%dst␣summand␣in␣%d␣iterations␣with␣a␣weight␣of␣%f\n',...
        num_iter,iter(num_iter),Error_iter);
end
num_iter = num_iter - 1; % The last sum was negligible, we discard it.
fprintf(1,'PGD␣off-line␣Process␣exited␣normally\n\n');
save('WorkSpacePGD_Parametricedforce.mat');
%
% POST-PROCESSING
%
fprintf(1,'Please␣select␣force␣position');
fprintf(1,'on␣the␣figure␣or␣pick␣out␣of␣the␣beam␣to␣exit');
h1 = figure(1); triplot(triangles);
axis equal;
[Cx,Cy] = ginput(1); % Waiting for a mouse click on the figure.
lim = 0.2/(Xf-X0); % Establishes an exit zone on the figure.
while X0-lim<=Cx && Cx<=Xf+lim && Y0-lim<=Cy && Cy<=Yf+lim
    h1 = figure(1); triplot(triangles);
    axis equal;
    Posforce = find(force<Cx,1,'last'); % Look for the closest loaded node.
    %
    % EVALUATING THE SOLUTION CHOOSING THE SELECTED NODE IN G VECTOR
    %
    desp = zeros(numel(coords),1);
    for i1=1:num_iter
        desp = desp + F(:,i1).*G(Posforce,i1); % Obtain the solution
    end
    %
    % PLOTTING THE SOLUTION
    %
    cdx = coords(:,1) + desp(1:2:end); % New X coordinates.
    cdy = coords(:,2) + desp(2:2:end); % New Y coordinates.
    trisurf(triangles.ConnectivityList,cdx,cdy,desp(2:2:end));
    title('Vertical␣Displacement'); view(2); colorbar;
    figure(1); axis equal; [Cx,Cy] = ginput(1); % Wait for a new force.

end
fprintf(1,'\n\n##########␣End␣of␣simulation␣##########\n\n');
```

As in Chap. 2, what we call stiffness and mass matrices are computed in function `elemstiff.m`:

```
function [K1,M2] = elemstiff(coor2)
% function [K1,M2] = elemstiff(coor2)
% For space compute stifness matrix, for load parameter compute mass matrix
% Universidad de Zaragoza - 2015

%
% SPACE MATRICES
%
[K1] = fem2D; % Standard 2D FEM code for Triangular Elements, computing
%
% LOAD MATRICES: 1D PARAMETRIC PROBLEM
%
sg = [-1.0/sqrt(3.0) 1.0/sqrt(3.0)]; wg = ones(2,1); % Gauss points
npg = numel(sg); nen2 = numel(coor2); M2 = zeros(nen2);
X2 = coor2(1:nen2-1)'; Y2 = coor2(2:nen2)'; % Coordinates of elements
L2 = Y2 - X2; % Longitude of each  element for parametriced variable
for i1=1:nen2-1
    c2 = zeros(1,npg); N2 = zeros(nen2,npg);
    c2(1,:) = 0.5.*(1.0-sg).*X2(i1) + 0.5.*(1.0+sg).*Y2(i1);
    N2(i1+1,:) = (c2(1,:)-X2(i1))./L2(i1);
    N2(i1,:) = (Y2(i1)-c2(1,:))./L2(i1);
    for j1=1:npg
        M2 = M2 + N2(:,j1)*N2(:,j1)'*0.5.*wg(j1).*L2(i1); %NůN
    end
end
return
```

The enrichment procedure, i.e., the computation of a new functional pair ***R***, ***S***, is detailed in function `enrichment.m`:

```
function  [R,S,iter] = enrichment(K1,M2,V1,V2,S0,F,G,num_iter,TOL,CC)
% function  [R,S,iter] = enrichment(K1,M2,V1,V2,S0,F,G,num_iter,TOL,CC)
% Computes a new sumand by fixed-point algorithm using PGD
% Universidad de Zaragoza - 2015
R = zeros(size(F,1),1); R0 = R; % Initial value R to compare in first loop.
h = size(V2,2); % Number functions of the source
ExitFlag = 1;
iter = 0;
mxit = 25; % #ă of possible iterations for the fixde point algorithm.
Free = setdiff(1:numel(F(:,1)),CC);
%
% FIXED POINT ALGORITHM
%
while ExitFlag>TOL
    %
    % LOOKING FOR R, KNOWNING S
    %
    matrixR = K1*(S0'*M2*S0);
    sourceR = zeros(size(F,1),1);
    for k1=1:h
        sourceR = sourceR + V1(:,k1)*(S0'*V2(:,k1));
    end
    for i1=1:num_iter-1
        sourceR = sourceR - K1*F(:,i1)*(S0'*M2*G(:,i1));
    end
    %
    % SOLVE R
    %
    R(Free) = matrixR(Free,Free)\sourceR(Free);
    %
    % LOOKING FOR S, KNOWNING R
    %
    matrixS = (R'*K1*R)*M2;
    sourceS = zeros(size(G,1),1);
    for k1=1:h
        sourceS = sourceS + V2(:,k1)*(R'*V1(:,k1));
    end
    for i1=1:num_iter-1
        sourceS = sourceS - R'*K1*F(:,i1)*(M2*G(:,i1));
    end
    %
    % SOLVE S
    %
    S = matrixS\sourceS;
    S = S./norm(S); % We normalize S. R takes care of alpha constant.
    %
    % COMPUTING STOP CRITERIA
    %
    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
```

The code makes use of a traditional, two-dimensional FEM code, whose structure is reproduced below. In fact, it returns the stiffness matrix $\boldsymbol{K}$ typical of these FEM programs.

```
function [K] = fem2D
% function [K] = fem2D
% A 2D FEM code for linear triangles. Return Stifness matrix
% Universidad de Zaragoza - 2015
global coords triangles E nu behaviour
dof = 2; % Degree of freedom per node
numNodes = size(coords,1); numTriang = size(triangles,1);
%
% ALLOCATE MEMORY
%
K = zeros(dof*numNodes);
%
% MATERIAL AND BEHAVIOUR
%
G = E/2/(1+nu);
if behaviour==2 % Plane Strain
    E1 = E*(1-nu)/(1+nu)/(1-2*nu);
    E2 = E*nu/(1+nu)/(1-2*nu);
elseif behaviour==1 % Plane Stress
    E1 = E/(1-nu^2);
    E2 = E*nu/(1-nu^2);
end
D = [E1 E2 0;E2 E1 0;0 0 G]; % Behaviour matrix
% Integration points: 3 Hammer Points
sg(1) = 1.0/6.0; sg(2) = 1.0/6.0; sg(3) = 2.0/3.0;
sg(4) = 1.0/6.0; sg(5) = 1.0/6.0; sg(6) = 2.0/3.0;
wg(1) = 1.0/6.0; wg(2) = 1.0/6.0; wg(3) = 1.0/6.0;
nph = numel(wg);
%
% ELEMENT LOOP
```

```
%
for j=1:numTriang
    tri = triangles.ConnectivityList(j,:); % Connectivity of each Element
    vertices = coords(tri,:); % Coordinates of the nodes
    Ind = [2*(tri-1)+1; 2*tri]; Ind = Ind(:);
    %
    % JACOBIAN
    %
    a = vertices(2,1)-vertices(1,1); b = vertices(3,1)-vertices(1,1);
    c = vertices(2,2)-vertices(1,2); d = vertices(3,2)-vertices(1,2);
    jcob = a*d - b*c;
    a1 = vertices(2,1)*vertices(3,2) - vertices(2,2)*vertices(3,1);
    a2 = vertices(3,1)*vertices(1,2) - vertices(3,2)*vertices(1,1);
    a3 = vertices(1,1)*vertices(2,2) - vertices(1,2)*vertices(2,1);
    b1 = vertices(2,2) - vertices(3,2); b2 = vertices(3,2) - vertices(1,2);
    b3 = vertices(1,2) - vertices(2,2);
    c1 = vertices(3,1) - vertices(2,1); c2 = vertices(1,1) - vertices(3,1);
    c3 = vertices(2,1) - vertices(1,1);
    %
    % INTEGRATION POINTS LOOP
    %
    chiG = 0.0; etaG = 0.0;
    for j1=1:nph
        chi = sg(2*(j1-1)+1); eta = sg(2*j1);
        %
        % GLOBAL GEOMETRICAL APPROXIMATION
        %
        SHPa(3) = eta; SHPa(2) = chi; SHPa(1) = 1.-chi -eta;
        for k1=1:3
            chiG = chiG + SHPa(k1)*vertices(k1,1);
            etaG = etaG + SHPa(k1)*vertices(k1,2);
        end
        %
        % COMPUTE SHAPE FUNCTIONS AND THEIR DERIVATIVES
        %
        SHP(1) = (a1+b1*chiG+c1*etaG)/jcob; dSHPx(1) = b1; dSHPy(1) = c1;
        SHP(2) = (a2+b2*chiG+c2*etaG)/jcob; dSHPx(2) = b2; dSHPy(2) = c2;
        SHP(3) = (a3+b3*chiG+c3*etaG)/jcob; dSHPx(3) = b3; dSHPy(3) = c3;
        %
        % N AND B MATRIX
        %
        B = [dSHPx(1) 0 dSHPx(2) 0 dSHPx(3) 0; ...
            0 dSHPy(1) 0 dSHPy(2) 0 dSHPy(3); ...
            dSHPy(1) dSHPx(1) dSHPy(2) dSHPx(2) dSHPy(3) dSHPx(3)];
        %
        % STIFNESS MATRIX
        %
        K(Ind,Ind) = K(Ind,Ind) + B'*D*B/jcob*wg(j1);
    end
end
return
```

Once executed, the code allows the user to choose interactively with the mouse the point in which the load is applied. It is implemented in an off-line/on-line approach, such that the modes (functional pairs) approximating the solution are first computed and (eventually) stored in memory. Then, in the on-line phase, the user can interactively play with the position of the load and see in real time the deformed configuration of the solid.

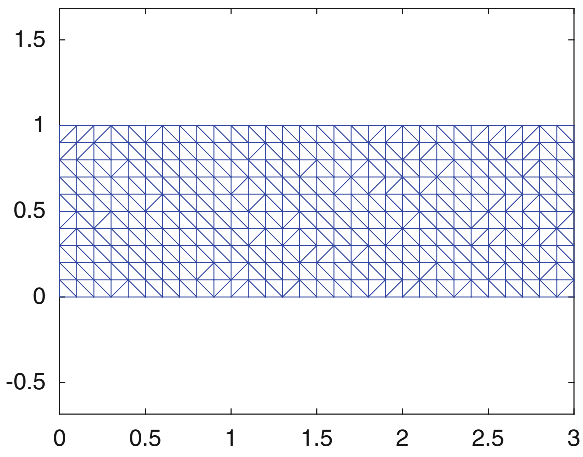

Fig. 3.4 Mesh for the moving load problem

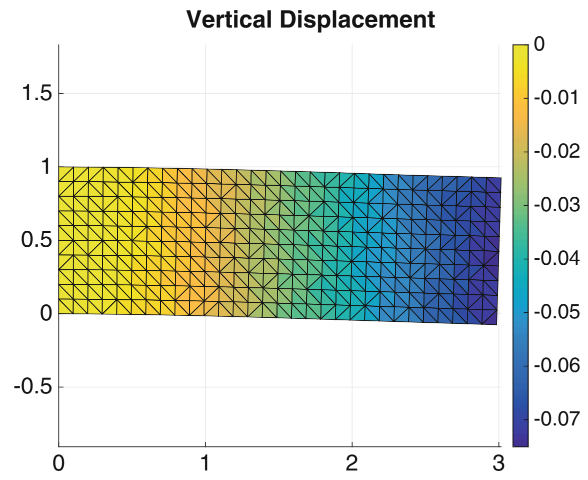

Fig. 3.5 Once a point along the upper side of the beam has been chosen, the problem depicts the deformed configuration of the beam

At a first instance, the mesh of the problem is shown, see Fig. 3.4. By clicking on it, the user can choose the particular placement of the load. The program gives immediately the deformed configuration of the beam, see Fig. 3.5.

Note that by simply typing on the Matlab command line the instruction `trisurf(triangles.ConnectivityList,coords(:,1),coords(:,2), F(1:2:end,1))`, the first spatial mode of the solution, namely $\boldsymbol{F}^1(\boldsymbol{x})$ is represented, see Fig. 3.6. Equivalently, by typing `plot(G(:,1))` the load modes $\boldsymbol{G}^i(\boldsymbol{s})$ can be plotted, see Fig. 3.7. Notice the increasing frequency content of the modes in the load variable.

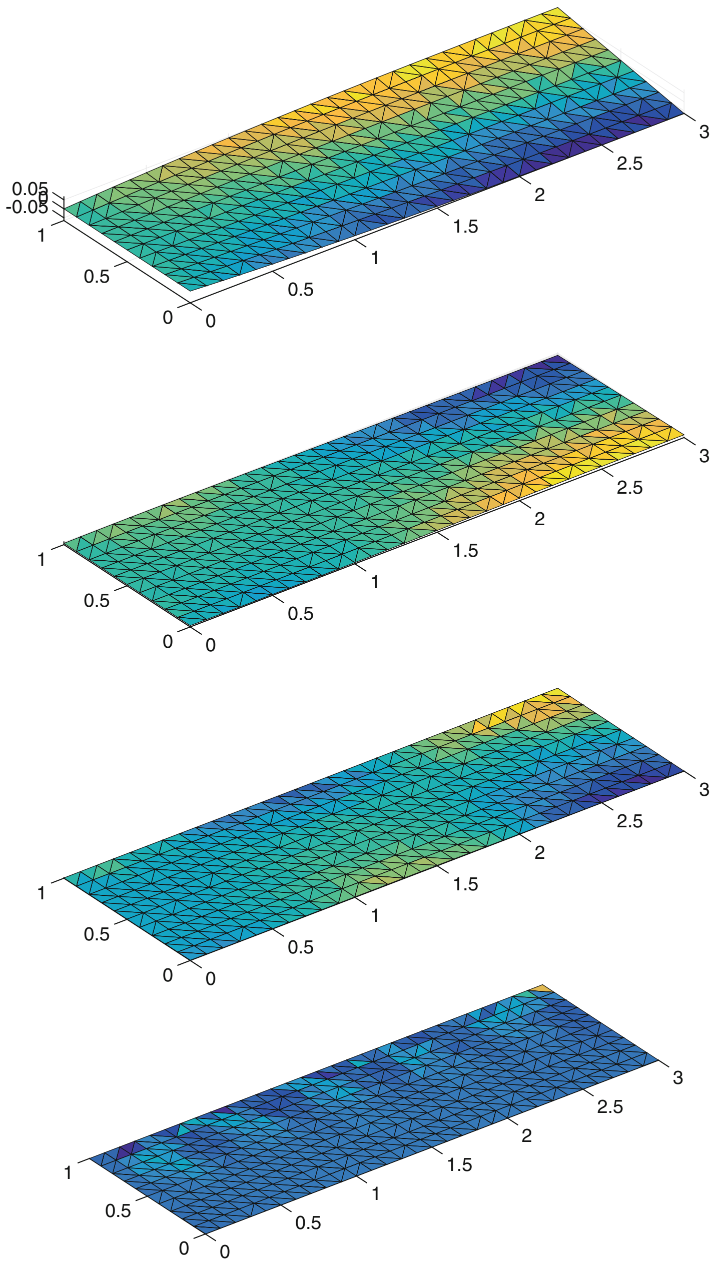

Fig. 3.6 Spatial modes $\boldsymbol{F}^i(\boldsymbol{x})$, $i = 1, 2, 3$ and 11. The magnitude of each mode is represented in the *vertical axis*

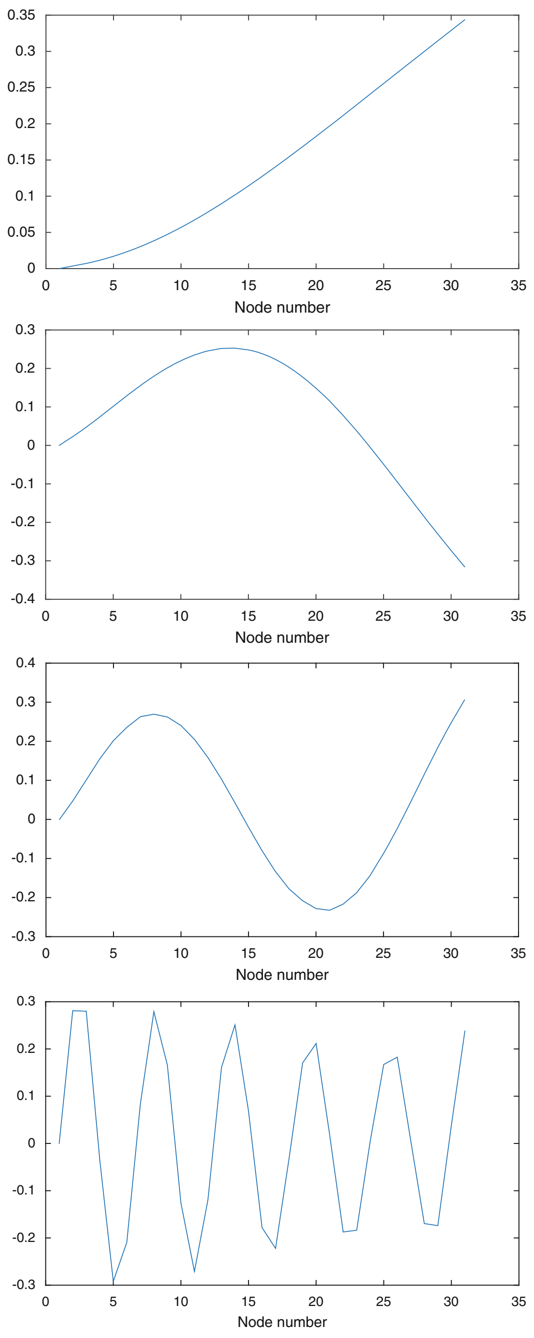

Fig. 3.7 Load modes $\boldsymbol{G}^i(\boldsymbol{s})$, $i = 1, 2, 3$ and 11. Node labels refer to the relative position along the upper boundary of the beam

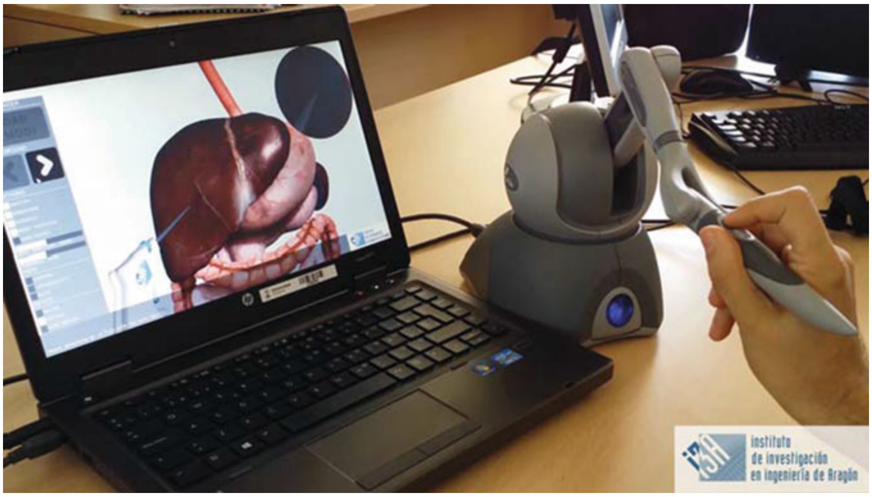

Fig. 3.8 A virtual surgery simulator based on the same algorithm explained in this chapter

This algorithm has revealed to be very powerful. In fact, it is essentially the same employed to construct our virtual surgery simulator, see Fig. 3.8, able to provide response feedback in the order of kHz, thus amenable to be employed in haptic environments [56, 57].