Chapter 1
Introduction

Young men should prove theorems, old men should write books.
—G.H. Hardy

Abstract This introductory chapter covers briefly the main idea of the book, how to code Proper Generalized Decomposition techniques with Matlab.

Proper Generalized Decomposition (PGD) has revolutionized many fields of applied sciences and engineering, and particularly the way we see parametric problems in a high dimensional setting. It has revealed how to obtain reduced-order models without the need for complex and costly computational experiments, typical of *a posteriori* model order reduction techniques such as the much better know Proper Orthogonal Decompositions (POD). These techniques, developed and re-discovered in many branches of science under different names such as Principal Component Analysis (PCA) [34], but also Karhunen-Loève transform [43, 48] in signal processing, also the Hotelling transform, Eckart-Young theorem, singular value decomposition (SVD), eigenvalue decomposition (EVD) in linear algebra, factor analysis, empirical orthogonal functions (EOF), empirical eigenfunction decomposition, empirical component analysis [49], quasiharmonic modes, and empirical modal analysis in structural dynamics. All these methods need for some snapshots, empirical realizations of the problem at hand under different values of the considered parameters. From these snapshots, the eigenmodes that contain most of the energy of their autocorrelation matrix are retained and employed as the best possible basis (given these realizations or snapshots) for subsequent simulations under different values of the parameters.

This approach (often referred to as *projection-based* methods, since the discrete equations are projected onto a reduced-order subspace in which the number of degrees of freedom of the problem is minimal) has, noteworthy, several disadvantages. In non-linear problems, for instance, the application of Newton algorithms for the linearization implies an update of the tangent stiffness matrix of the full problem, and

E. Cueto et al., *Proper Generalized Decompositions*, SpringerBriefs
in Applied Sciences and Technology, DOI 10.1007/978-3-319-29994-5_1

hence the loss of most of the time savings obtained through POD. Although several alternatives exist, such as the use of empirical interpolation methods, for instance, [14, 21, 53], or the use of perturbation techniques in conjunction with POD, [58], no definitive answer has been given to the model order reduction of non-linear problems.

The origin of Proper Generalized Decompositions, can be traced back to the so-called *radial loading* within the LATIN method [44] as a space-time separated representation in non-incremental structural mechanics solvers. Independently, Chinesta and coworkers in their seminal papers [6, 7] developed a method for the solution of non-Newtonian fluid models defined in high-dimensional phase spaces that were soon identified as a generalization of the work by P. Ladeveze. PGD is constructed indeed upon a very old idea, the method of separation of variables or Fourier method for partial differential equations. But the main novelty lies in the ability of PGD for the construction of sums of separated functions *a priori*, i.e., without any prior knowledge on the solution nor the need for costly computer experiments or snapshots. Indeed, a PGD approximation to the solution of a given PDE, say u, depending in principle of space, time and a number m of parameters, assumes a form

$$u(x, t, p_1, p_2, \ldots, p_m) \approx u^n = \sum_{i=1}^{n} F_x^i(x) \cdot F_t^i(t) \cdot F_1^i(p_1) \cdot F_2^i(p_2) \cdot \ldots \cdot F_m^i(p_m), \tag{1.1}$$

where the functions F_i^j are in principle unknown and p_i represent the parameters affecting the solution.

Briefly speaking, to determine these functions F_i^j a non-linear problem must be solved (independently of the character of the initial problem), since we look for one or more products of functions. Surprisingly or not, very simple techniques have demonstrated to provide very good results. Thus, for instance, in many references of the PGD bibliography (see [24, 29] for recents reviews on the field) greedy algorithms are employed such that one sum is computed at a time, and within each greedy step, naive linearization strategies such as fixed point iterations usually provide very good results.

The choice of an appropriate truncation level n deserves some comments. In fact, Eq. (1.1) would need, in the most general case, an infinite number of terms. To properly determine the number of terms n needed to obtain a certain level of accuracy, several possibilities exist. The most rigorous ones include the computation of error estimators, possibly based on engineering quantities of interest, see [3, 4, 19, 46, 52]. It is also worthy of mention that the PGD method, as stablished in Eq. (1.1) has proven convergence for elliptic problems [47].

This very simple approach has allowed to solve parametric problems in phase spaces of one hundred dimensions but, notably, it has provided new insights in the way we look at physical phenomena governed by partial differential equations. Thus, for instance, treating as parameters things that *a priori* are not parameters (such as loads, boundary conditions, initial conditions, …) allows for a very efficient solution

of different problems in engineering and applied sciences, that sometimes reaches real-time constraints by employing an off-line/on-line strategy.

Just to show some examples, PGD has allowed to construct efficient surgery simulators with haptic response, see Fig. 1.1, or to enable not-so-simple simulations on handheld, deployed devices such as smartphones or tablets, Fig. 1.2, to embed complex simulations on a simple web page (by employing simple java applets), see Fig. 1.3. The developed technique could even have important implications in augmented learning environments, opening the possibility to include real-time simulations on e-books, Fig. 1.4 [60].

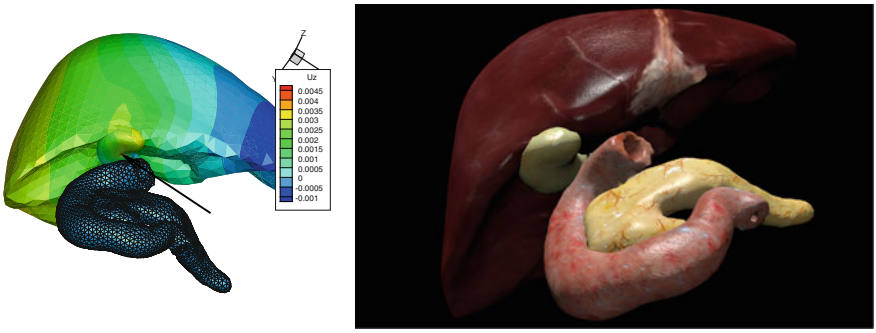

Fig. 1.1 An example of surgery simulator developed with the aid of PGD methods [57]

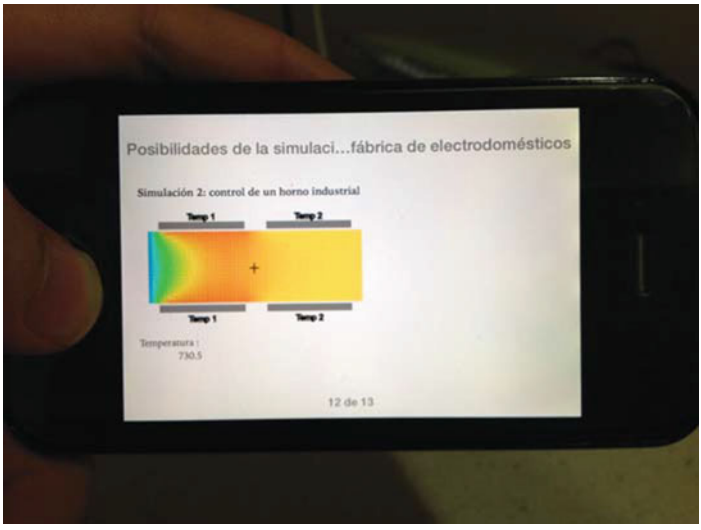

Fig. 1.2 Interactive simulation of an industrial furnace running on an iPhone [35]. The simulation could eventually be continuously fed by data streaming from sensors. It is what is known as dynamic, data-driven applications systems (DDDAS)

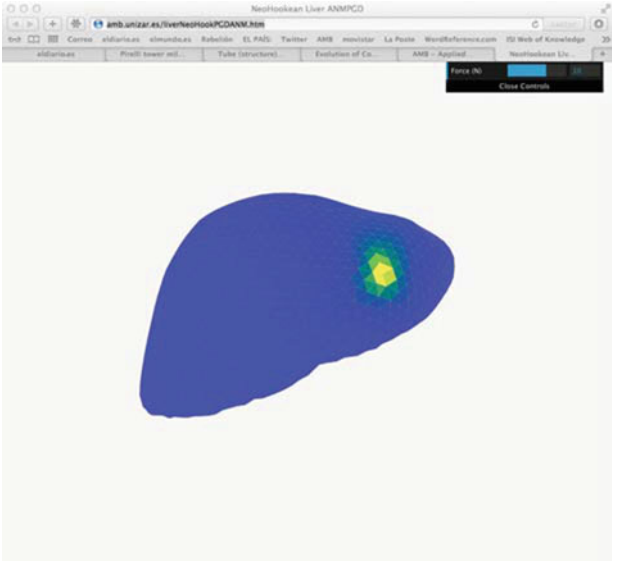

Fig. 1.3 Interactive palpation of a liver running a small java applet on a web page

Fig. 1.4 PGD techniques open the possibility to embed real-time, interactive simulations on e-books, thus opening unprecedented possibilities for augmented learning environments [60]

In recent times, PGD has been applied to a wide variety of problems, showing its impressive ability to give appropriate responses in very different fields. Thus, for instance, one can mention the solution of Helmholtz equations [13, 51], geophysical problems [63], magnetostatics [42], real-time monitoring [2], Boltzmann and Fokker-Planck equations [22], gene regulatory networks [5, 30], contact problems [37, 40], the construction of response surfaces, virtual charts or computational vademecums [26, 61], multiscale problems [9, 23, 25, 32, 45], shape optimization [10], dynamic, data driven application systems (DDDAS) [36, 39], or virtual surgery [3, 33, 56, 59], to cite but a few of the more than 300 references available in the World of Science as of December, 2015. In addition, PGD is a quite intrusive method that can not, in general, be applied with the help of commercial finite element codes. Nevertheless, it has been efficiently coupled to existing techniques in a number of references, see, for instance, [8, 11].

However, from the programmer point of view, PGD has a clear barrier to entry. Even for experienced finite element programmers, PGD could appear as something complex, obscure, with many tricks to know in advance and difficult to understand. It is not the purpose of this *brief* to cover all the theory related to PGD, something already done in previous monographs, see [27, 28], for instance. Instead, this book is devoted to an easy and friendly introduction to PGD programming with one of the most popular languages for applied scientists and engineers, Matlab.