The Pennsylvania State University

The Graduate School

Department of Mechanical and Nuclear Engineering

COMPUTATION OF NATURAL FREQUENCIES AND MODE SHAPES OF A PERIODIC STRUCTURE VIA PERTURBATION ANALYSIS AND POLYNOMIAL CHAOS

A Thesis in

Mechanical Engineering

by

Jonathan Barés

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

May 2010

The thesis of Jonathan Barés was reviewed and approved* by the following:

Alok Sinha
Professor of Mechanical Engineering
Thesis Advisor

Asok Ray
Distinguished Professor of Mechanical Engineering

Karen A. Thole
Professor of Mechanical Engineering
Head of the Department of Mechanical and Nuclear Engineering

*Signatures are on file in the Graduate School

ABSTRACT

The goal of this paper is to analyze the modal characteristics of a mistuned bladed disk assembly. We compute the expected value and standard deviation of eigenvalues and eigenvectors of such a system by using a polynomial chaos technique. The model of the bladed disk assembly considers only one mode of vibration of each blade and the mistuning phenomenon has been simulated by treating the modal stiffness of each blade as a stochastic variable. A Monte Carlo simulation and a Taylor series expansion are used to validate the result of this method.

RESUME

L'objectif de cet article est d'analyser les caractéristiques modales d'un assemblage cyclique d'hélices avec pour chacune d'entre-elles un paramètre incertain. Nous calculons la valeur moyenne et l'écart type des valeurs propres et vecteurs propres d'un tel système and utilisant une méthode de polynômes de chaos. Le modèle utilisé pour l'assemblage cyclique d'hélice ne prend en considération que le premier mode de vibration de chaque hélice et l'incertitude sur un paramètre a été choisie pour être la raideur modale de chaque hélice. Celle-ci suit une loi gaussienne. Une simulation de Monte Carlo et un développement en série de Taylor sont utilisés pour vérifier les résultats obtenus avec cette méthode.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

NOMENCLATURE

- General notations

$\{\bullet\}$: Vector

${}^{T}\{\bullet\}$: Transposed vector

$(\{\bullet\})_q$: q^{th} element of the vector

$[\bullet]$: Matrix

${}^{T}[\bullet]$: Transposed matrix

$[I_d]$: Identity matrix of dimension d

y_{ij} : Elements of the matrix $[Y]$ (general formulation)

δ_{ij} : Kronecker delta symbol

$\langle \bullet | \bullet \rangle$: Scalar product associated with the Hilbert space

$\langle \bullet \rangle$: Expected value operator

$\sigma_{\bullet}$: Standard deviation operator

- For the mechanical problem:

ma : Mass f the oscillators

k_0 : Stiffness of the tuned oscillators

k_c : Coupling stiffness

n : Number of oscillators

ω : Frequency of the oscillators

$k_i = k_0(1+\xi_i)$: Stiffness of oscillators

$u_i(t) = U_i \cdot e^{j\omega t}$: Displacement of the oscillators

$f_i = f_0 \cdot e^{j\varphi_i} \cdot e^{j\omega t}$: Excitation forces on oscillators

$\varphi_i = \dfrac{2\pi r(i+1)}{n}$: Difference of phase between excitation

$R = \dfrac{k_c}{k_0}$: Coupling ratio

$\omega_0 = \sqrt{\dfrac{k_0}{ma}}$: Natural frequency of the oscillators

$\lambda = \left(\dfrac{\omega}{\omega_0}\right)^2$: Eigenvalue of the system

$\xi_0 = \dfrac{c_0}{ma \cdot \omega_0}$: Viscous damping coefficient forces

$[K]$: Deterministic part of the normalized stiffness matrix

$[\tilde{K}(\bar{\xi})]$: Stochastic part of the normalized stiffness matrix

$\bar{\xi}$: Vector of mistuned parameters

$(\lambda_k, \{U_k\})$: Couples of eigenvalues and eigenvectors for the tuned system

$\left(\tilde{\lambda}_k(\bar{\xi}), \{\tilde{U}_k(\bar{\xi})\}\right)$: Stochastic couples of eigenvalues and eigenvectors for the random system

$\{\xi_i\}_i$: Mistuned parameters

- For the explanation of the polynomial chaos method:

(Ω, A, P) : Measurable space of random events

ω : Random event

$(\Psi_k)_{k\in\aleph}$: Family of Polynomial chaos

$\{\phi_i(\bar{\xi}(\omega))\}_{i\in\aleph}$: Family of Hermite polynomial chaos

Θ : Hilbert space of elementary random process

X, Y : Random processes

Ψ : Subspace of Θ

m: Number of used Hermite polynomial chaos

p: Maximum degree of used Hermite polynomial chaos

$Equa_k\left(\bar{\xi}\right)$: Stochastic equation

- For the application of polynomial chaos method:

$\left\lfloor J_q \right\rfloor$: Zero matrix of dimension n with the q^{th} diagonal element equal to 1

$\left\{ {}^k\beta_i^{\,j} \right\}_{\substack{j\in\{1,\ldots,n\}-\{k\} \\ i\in\{1,\ldots,m\}}}$: Coefficients of the truncated expended form of the stochastic eigenvectors

$\left\{\alpha_i^k\right\}_{i\in\{1,\ldots,m\}}$: Coefficients of the truncated expended form of the stochastic eigenvalues

$\left.\begin{array}{l} [A]\in M_{n\times m}(\Re) \\ [B]\in M_{n\times m\times d\times m}(\Re) \\ [C]\in M_{n\times m}(\Re) \\ [E]\in M_{n\times 1}(\Re) \\ [D]\in M_{n\times m\times n}(\Re) \\ [M]\in M_{m\times m\times m}(\Re) \end{array}\right\}$ Coefficients of the first system of equations

$\left[\overline{A}\right]\in M_{n\times m}(\Re)$: Storage matrix of $\left\{\alpha_i^k\right\}$

$\left[\overline{B}\right]\in M_{n\times m\times(n-1)}(\Re)$: Storage matrix of $\left\{ {}^k\beta_i^{\,j} \right\}$

$\left[\Gamma_k\right] = \left[{}^k\gamma_i^{\,j} \right]_{\substack{j\in\{1,\ldots,n\} \\ i\in\{1,\ldots,m\}}}$: Matrix of unknowns of the system

$\left\langle \phi_i \middle| \phi_j \middle| \phi_k \right\rangle$: Average value of the product of three polynomial chaos

$\{\chi\}\in \Re^n$: Chosen radial direction in the mistuned parameter space

$\left[\overline{K}\right]\in M_{n\times 2}(\Re)$: Diagonal deterministic matrix which linearizes $\left[\tilde{\tilde{K}}\right]$

ρ: New unique normalized stochastic parameter

$(\{v\},\{w\})$: Family of two repeated eigenvalues

$[X]\in M_{n\times 2}(\Re)$: Matrix of two eigenvectors associated with a repeated eigenvalues

$[Z]\in M_{n\times 2}(\Re)$: Matrix of two new eigenvectors associated with a repeated eigenvalues

$[\Lambda]$: Zero matrix with diagonal elements equal to repeated eigenvalues

$[G]\in M_{2\times 2}(\Re)$: Convenient matrix to solve our problem

$\left.\begin{array}{l} \left[\hat{A}\right]\in M_{m\times n}(\Re) \\ \left[\hat{B}\right]\in M_{n\times m\times d\times m}(\Re) \\ \left[\hat{C}\right]\in M_{m\times n}(\Re) \\ \left[\hat{D}\right]\in M_{n\times 1}(\Re) \\ \left[\hat{E}\right]\in M_{m\times m\times m}(\Re) \end{array}\right\}$ Coefficients of the first system of equations

Chapter 1

Introduction

1.1 Motivation of this study

Any manufacturing process of a bladed disk assembly implies variations in modal parameters of blades. This problem has received wide attention in the existing literature ([2], [4], [9]-[12]) because these deviations can lead to large deviations in the amplitudes of blade vibration. Furthermore, deviations in modal parameters due to manufacturing tolerances are random variables. Hence, there exists a large number of sets of blades' parameters that a bladed disk assembly may have. Since the maximum amplitude of blade vibration will be different for each bladed disk assembly, it is important to know the statistics of eigenvectors, particularly, the probability that an eigenvector corresponding to a certain frequency exceeds a certain critical value. Technologically, a good knowledge of this phenomenon could permit to improve the efficiency of hydro-electric turbines, aircraft engine …etc

1.2 Existing works on the subject

Because of the technological interest in this problem, there exists a lot of papers which deal with it. To begin with, a classical approach to these kinds of problems is to use a Monte Carlo simulation. We compute the responses for a large set of random entries and we statistically treat them. But such a method is time consuming and needs lots of computer time. Hence, perturbation methods have been developed to obtain required results in a computationally efficient manner. The most common is to expand, via Taylor series, all the stochastic variables about the value of the parameter in the tuned case, up to the second order. Then we can get approximations of the variations of the parameters as functions of the mistuned parameter and we can

also get statistic parameters of the system. This method has been explained by Dessombz et al. [1] and Sinha [2]. It is much faster than the Monte Carlo simulation, but suffers from limitations in terms of variance of the uncertain parameters. That's why a more complete perturbation method has been developed. It is based on the Karhunen-Loeve decomposition coupled with a polynomial chaos expansion and a Galerkin projection. This method has been widely developed by Ghanem and Spanos [6]. Sinha [4] applied it to compute the statistics of a forced response of a mistuned bladed disk assembly. Dessombz et al. [1] also used it to compute the statistics of the modal parameters of a solid mechanical system in case of finite elements study. Nevertheless, to our knowledge it has never been used to compute the statistics of the eigenvalues and eigenvectors of a mistuned bladed disk assembly.

1.3 Work carried out in this thesis

Hence, in this thesis, we have applied the polynomial chaos technique to compute the statistics of the eigenvalues and eigenvectors of a mistuned bladed disk assembly. This could seem to be a direct application of the work by Dessombz et al. [1]. However, one of main particularities of this system and one of the main problems to treat it with the classical method is the eigenvalue coalescences. Actually, the modal system in the tuned case has number repeated (two times) eigenvalues. This brings discontinuity problems when several mistuned parameters vary. These difficulties have been discussed by Sinha [2], Andrew and Tan [5] and Zhang and Wang [7] who have developed an algorithm along with a change of variable to avoid the discontinuity of eigenvalues and eigenvectors. Thus, we have also applied it to develop a computer code which leads to a numerical application of this polynomial chaos method. Then, we have discussed the difficulties in checking the results we have got because of the coalescence of eigenvalues. Indeed for repeated eigenvalues and associated eigenvectors it is impossible to use a Monte Carlo simulation to check our results. So we have used a Taylor series method. Finally, we deal with the validity of our numerical results.

Chapter 2

Presentation of the mechanical problem

In this chapter we want to present briefly how we have chosen to model a bladed disk assembly with uncertain parameters. We also derive the equations of motion of such a system and we write them in a consistent form to treat the problem in a general way.

2.1 Model of mistuned bladed disk assembly

The bladed disk assembly is modeled as a close chain of n linear oscillators whose one of the parameter is a random variable (Figure 1). The stiffness of the spring which links the mass to the ground is chosen to be this random parameter. This choice represents the fact that the modal stiffness of a blade is a random variable due to manufacturing tolerance. It should be noted that there is no loss of generality, we could take any other physical parameter as a random variable.

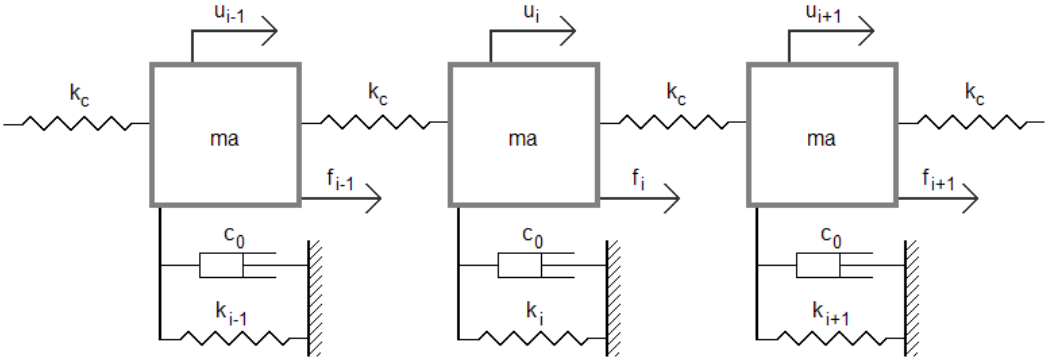


Figure 1: Model of bladed disk assembly

Modelization by a cyclic chain of oscillators

In Figure 1 $i \in \{1,\dots, n\}$ with $\begin{cases} i=1 \Rightarrow i-1=n \\ i=n \Rightarrow i+1=1 \end{cases}$ since we have a geometrically (or rotationally) periodic system.

We note that the minimum value for n is 3. Indeed, if we take $n = 2$, the physics of the problem is slightly changed. Moreover a case where n is small is not interesting technologically.

2.2 Equations of motion of the system

We want to write the equations of motion of this system in a matrix to simplify the mathematical study. Hence, we insulate one part of the system, that is to say, we choose one $i \in \{1,\dots, n\}$ and we write the equation of the dynamic for this part:

$\forall i \in \{1,\dots, n\}$

$$ma \cdot \ddot{u}_i = -k_i \cdot u_i - c_0 \cdot \dot{u}_i - k_c(u_i - u_{i-1}) + k_c(u_{i+1} - u_i) + f_i \qquad (1)$$

$$ma \cdot \ddot{u}_i + (k_i + 2k_c) \cdot u_i + c_0 \cdot \dot{u}_i - k_c \cdot u_{i-1} - k_c \cdot u_{i+1} = f_i \qquad (2)$$

Then, assuming a harmonic excitation and a resulting harmonic motion, we get:

$$(k_i + 2k_c - ma \cdot \omega^2 + jc_0 \cdot \omega) \cdot U_i - k_c \cdot U_{i-1} - k_c \cdot U_{i+1} = f_0 \cdot e^{j\varphi_i} \qquad (3)$$

We divide by k_0 and use the notation given in the nomenclature to get:

$$(1 + \xi_i + 2R - \lambda + j\xi_0\sqrt{\lambda}) \cdot U_i - R \cdot U_{i-1} - R \cdot U_{i+1} = f_0 \cdot e^{j\varphi_i} \qquad (4)$$

Therefore, we can write these n equations in the matrix form:

$$\left(\underbrace{\begin{bmatrix} 1+2R & -R & 0 & 0 & -R \\ -R & 1+2R & -R & 0 & 0 \\ 0 & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & -R & 1+2R & -R \\ -R & 0 & 0 & -R & 1+2R \end{bmatrix}}_{[K]} + \underbrace{\begin{bmatrix} \xi_1 & & & & \\ & \xi_2 & & 0 & \\ & & \ddots & & \\ & 0 & & \ddots & \\ & & & & \xi_n \end{bmatrix}}_{[\tilde{K}(\bar{\xi})]} + (\lambda + j\xi_0\sqrt{\lambda}) \cdot I_n \right) \cdot \begin{Bmatrix} U_1 \\ U_2 \\ \vdots \\ \vdots \\ U_n \end{Bmatrix} = f_0 \begin{Bmatrix} e^{\varphi_1} \\ e^{\varphi_2} \\ \vdots \\ \vdots \\ e^{\varphi_n} \end{Bmatrix} \qquad (5)$$

We will call $[K]$ the deterministic part of the stiffness matrix and $\left[\tilde{K}(\bar{\xi})\right]$ the random one, with $\bar{\xi}=(\xi_1,...,\xi_n)$ the random variables of the system. To simplify the matrix equation we could assume a hysteretic damping. Nevertheless, since we are interested in the eigenvalues and eigenvectors, for the time being, we will say that $\xi_0=0$. Similarly, since the left part of the left hand side is not deterministic, the eigenvalues and associated eigenvectors of the system are random functions, so we introduce random eigenvalues ($\tilde{\lambda}_k(\bar{\xi})$) and random eigenvectors ($\{\tilde{U}_k(\bar{\xi})\}$) which will be the unknown of the system. That is to say $(\lambda_k,\{U_k\})_{k\in\{1,...,n\}}$ for the tuned system becomes $(\tilde{\lambda}_k(\bar{\xi}),\{\tilde{U}_k(\bar{\xi})\})_{k\in\{1,...,n\}}$ for the random one.

Finally, the equation (5) becomes:

$$\left([K]+\left[\tilde{K}(\bar{\xi})\right]-\tilde{\lambda}_k(\bar{\xi})\cdot[I_n]\right)\{\tilde{U}_k(\bar{\xi})\}=\{0\} \qquad \forall k\in\{1,...,n\} \qquad (6)$$

Chapter 3

Explanation of the polynomial chaos method

In this chapter, we give an explanation of the polynomial chaos method and we detail it in an accurate way. To do so, we begin with introduction of the different mathematical operators and spaces in which the variables we manipulate exist. Then, we show how to use these operators and variables to simplify the problem.

3.1 General presentation of the polynomial chaos method

We recall that the goal of our study would be to fully solve the equations (6). Nevertheless the unknowns $\left(\tilde{\lambda}_k\left(\bar{\xi}\right), \left\{\tilde{U}_k\left(\bar{\xi}\right)\right\}\right)_{k\in\{1,\dots,n\}}$ belong to a very complex mathematical space. So it is not reasonable to fully solve it. We will just compute an approximated solution. To understand what we will do to find this approximated solution it could be fruitful to compare with what we do in finite element methods. When we have a complex mechanical problem to solve for example, first we determine a basis (most often orthogonal) of a subspace of the space of the exact solution. Then, we assume the approximated solution we are looking for belongs to this subspace. Thus, to find it we just have to minimize the error between the approximated and exact solutions using an orthogonal projection onto this subspace. This is called a Galerkin projection. In the case of polynomial chaos, we follow exactly the same process using probability spaces. Ghanem and Spanos [3]&[6] are among the firsts to solve these kinds of problems in this way.

3.2 Definition of the spaces and operators of the problem

First, we need to define the space in of each random eigenvalue and coefficient of eigenvector. But before that, we have to define the measurable space of random events which contains the variables of random process functions [3]. We call it (Ω, A, P) with:

- Ω : the event space
- A : the sigma-algebra of Ω
- P : the probability measure

Then, the space of each random eigenvalue and coefficient of eigenvector is a Hilbert space called Θ, the space of elementary random processes, such that:

$$\forall X \in \Theta \qquad X : \begin{matrix} \Omega \to \Re \\ \omega \mapsto X(\omega) \end{matrix} \qquad (7)$$

The scalar product of this Hilbert space is of course defined with the measure of (Ω, A, P) as follow:

$$\forall X, Y \in \Theta \qquad \langle X | Y \rangle = E(X \cdot Y) = \int_{\Omega} X(\omega) \cdot Y(\omega) dP(\omega) \qquad (8)$$

We note that this scalar product is just the expected value of the product of the two random processes.

Then, according to Karhunen-Loeve theorem [14] & [15], for a stochastic process of second order $X \in \Theta$, where Θ corresponds with our physical problem, we can express $(\forall \omega \in \Omega)\ X(\omega)$ as a function of $(\xi_j(\omega))_{j \in \{1,\ldots,n\}}$ such that:

$$X(\omega) = a_0 \cdot \Psi_0 + \sum_{i_1=1}^{+\infty} a_{i_1} \cdot \Psi_1(\xi_{i_1}(\omega)) + \sum_{i_1=1}^{+\infty} \sum_{i_2=1}^{i_1} a_{i_1 i_2} \cdot \Psi_2(\xi_{i_1}(\omega), \xi_{i_2}(\omega))$$

$$+ \sum_{i_1=1}^{+\infty} \sum_{i_2=1}^{i_1} \sum_{i_3=1}^{i_2} a_{i_1 i_2 i_3} \cdot \Psi_3(\xi_{i_1}(\omega), \xi_{i_2}(\omega), \xi_{i_3}(\omega)) + \sum_{i_1=1}^{+\infty} \sum_{i_2=1}^{i_1} \sum_{i_3=1}^{i_2} \sum_{i_4=1}^{i_3} a_{i_1 i_2 i_3 i_4} \cdot \Psi_4(\xi_{i_1}(\omega), \xi_{i_2}(\omega), \xi_{i_3}(\omega), \xi_{i_4}(\omega)) + \ldots \qquad (9)$$

Where $(\Psi_k)_{k \in \aleph}$ are polynomial chaos.

Nevertheless this formula is not very convenient to use except theoretically. We prefer to expand the random process as a spectral expansion on orthogonal Hermite polynomials [6] as follow:

$$X(\omega)=\sum_{i=1}^{+\infty} a_i \cdot \phi_i\left(\overline{\xi}(\omega)\right) \qquad \text{with} \quad \overline{\xi}(\omega)=\left(\xi_1(\omega),\ldots,\xi_n(\omega)\right) \qquad (10)$$

In this case we use $\left\{\phi_i\left(\overline{\xi}(\omega)\right)\right\}_{i\in\aleph}$, the Hermite polynomial basis which is an orthogonal basis of Ω (more details about these polynomials and the way to build them are given in Appendix A) . However, we note that we could use any other uncorrelated or statistically independent basis.

Besides, as it is close to most of the physical cases and because it is numerically convenient, we choose $\overline{\xi}(\omega)$ to be a random vector of independent (uncorrelated) Gaussian random variables of the random event $\omega\in\Omega$. This is important and very convenient because from now, we can forget to write the variable ω .Hence, we are just interested in the statistical behavior of $\overline{\xi}$ and we already know it since $\forall j\in\{1,\ldots,n\}$ $\xi_j=\aleph(\mu_j,{\sigma_j}^2)$.

We have managed to reduce the complexity of the Ω space. But we need to be precise about what the scalar product becomes in this case. Actually, we are just interested in the scalar product of Hermite polynomials because any element of Θ we will manipulate is a linear combination of these polynomials. So we just compute:

$$\left\langle \Phi_i\left(\overline{\xi}(\bullet)\right)\middle|\,\phi_j\left(\overline{\xi}(\bullet)\right)\right\rangle=\int_\Omega \phi_i\left(\overline{\xi}(\omega)\right)\cdot\phi_j\left(\overline{\xi}(\omega)\right)dP(\omega)=\int_\Omega \phi_i\left(\overline{\xi}\right)\cdot\phi_j\left(\overline{\xi}\right)w\left(\overline{\xi}\right)d\overline{\xi}=\left\langle\phi_i\middle|\,\phi_j\right\rangle \qquad (11)$$

With $w\left(\overline{\xi}\right)=\dfrac{dP(\omega)}{d\overline{\xi}(\omega)}$ the weight function of the Radon-Nikodym theorem.

We note that this vanishes if $i\neq j$ because of orthogonality.

3.3 Polynomial chaos method

In our case we recall that we are looking for an approximation of the exact solution. Hence we are looking for a truncated version of the expansion (10):

$$X\left(\overline{\xi}\right)=\sum_{i=1}^{m} a_i \cdot \phi_i\left(\overline{\xi}\right) \qquad (12)$$

Nevertheless because we actually truncate the sum (9), m must follow:

$$m=\sum_{k=0}^{p}\binom{n+k-1}{k}=\frac{p+1}{n}\binom{n+p}{1+p} \qquad (13)$$

Where n is the number of Gaussian variables and p is the value of the highest polynomial chaos order. The way to find this formula is explained in Appendix A.

Now, we note that we can see this last operation from another point of view. Indeed, let us consider the following subset of Θ : $\Psi = Span\left\{\phi_i / i \in \{1,..., m\}\right\}$

Truncating the sum (10) turns out to be the projection of the element $X \in \Theta$ onto the subspace Ψ. We know that the exact solutions of the equations (6) belong to Θ. We will try to get the projection of these solutions onto the subspace Ψ. So we want to minimize the error between the exact solutions in Θ and the approximated ones in Ψ. To do so, we will plug the new form of our unknown (12) and we will project the equation onto each vector of an orthogonal basis of Ψ. That will gives us $m \times n$ equations:

$$\begin{matrix}\forall i \in \{1,..., m\} \\ \forall k \in \{1,..., n\}\end{matrix} \qquad \left\langle Equa_k\left(\overline{\xi}\right) \middle| \phi_i\left(\overline{\xi}\right)\right\rangle \qquad (14)$$

Therefore, now we have $n \times m$ equations to solve.

But if we look at (6) carefully, we can see that we have now $n \times m$ matrix equations. So we need to project these equations onto an orthogonal basis of $\Re^n$. Of course, we will use the modal basis vectors of the tuned system ($\{U_j\}_{j\in[1,\ldots,n]}$):

$$\begin{array}{l} \forall i \in \{1,\ldots, m\} \\ \forall k \in \{1,\ldots, n\} \\ \forall j \in \{1,\ldots, n\} \end{array} \qquad \left\langle {}^T\{U_j\} \cdot Equa_k\left(\bar{\xi}\right) \middle| \phi_i\left(\bar{\xi}\right) \right\rangle \qquad (15)$$

Finally, we get $n^2 \times m$ scalar equations that we can solve numerically. Actually, we will see later that these equations are non linear. Hence, we use a Newton-Raphson's algorithm to solve it.

Chapter 4

Application of polynomial chaos method to the problem

In this chapter we apply the polynomial chaos method we have explained in chapter 3 to the model of bladed disk assembly we have built in chapter 2. We also explain the formalism we have used to implement this algorithm in MatLab [13]. And finally, we show the problem of using several random variables.

4.1 Application of the polynomial chaos to the bladed disk assembly

4.1.1 Expression of the elements of the problem with the mathematical formalism

Now we want to apply what we explained in the previous chapter to the equations (6). First, we note that the eigenvalues and coefficients of eigenvectors are random processes, they belong to Θ. So as we did in (12), we write the random eigenvalues and eigenvectors using the Karhunen-Loeve truncated expansion with Hermite polynomial chaos:

$$\tilde{\lambda}_k(\bar{\xi}) \approx \lambda_k \cdot \sum_{b=1}^{m} \alpha_b^k \cdot \phi_b(\bar{\xi}) \qquad \forall k \in \{1,\ldots,n\} \qquad (16)$$

$$\{\tilde{U}_k(\bar{\xi})\} \approx \sum_{r=1}^{n} \tilde{\beta}_r^k(\bar{\xi}) \cdot \{U_r\} \quad \text{with} \quad \tilde{\beta}_r^k(\bar{\xi}) \approx \sum_{l=1}^{m} {}^k\beta_l^r \cdot \phi_l(\bar{\xi}) \qquad \forall k \in \{1,\ldots,n\} \qquad (17)$$

Introducing a classical normalization condition ${}^T\{U_k\}[I_n]\{\tilde{U}_k(\bar{\xi})\} = 1 \quad \forall k \in \{1,\ldots,n\}$ we get:

$$\{\tilde{U}_k(\bar{\xi})\} \approx \{U_k\} + \sum_{\substack{r=1 \\ r \neq k}}^{n} \sum_{l=1}^{m} {}^k\beta_l^r \cdot \phi_l(\bar{\xi}) \cdot \{U_r\} \qquad (18)$$

We also need to express the input random element of the problem with the same approximation on Ψ. So, we express the random stiffness matrix onto this polynomial basis. This just gives:

$$\left[\tilde{K}\left(\bar{\xi}\right)\right]=\begin{bmatrix}\xi_1 & & & \\ & \xi_2 & & 0 \\ & & \ddots & \\ & 0 & & \ddots \\ & & & & \xi_n\end{bmatrix}=\sum_{q=1}^{n}\left[J_q\right]\cdot\xi_q \qquad (19)$$

With $\left[J_q\right]=\begin{bmatrix}0 & & & \\ & \ddots & & 0 \\ & & 1 & \\ & 0 & & \ddots \\ & & & & 0\end{bmatrix}$ (q^{th} row) $\quad \forall q\in\{1,\ldots,n\}$ (20)

Then, we plug (16), (18), and (19) into (6) and we obtain:

$\forall k\in\{1,\ldots,n\}$

$$\left(\left[K\right]+\sum_{q=1}^{n}\left[J_q\right]\cdot\xi_q-\lambda_k\cdot\sum_{b=1}^{m}\alpha_b^k\cdot\phi_b\cdot\left[I_n\right]\right)\left(\left\{U_k\right\}+\sum_{\substack{r=1\\ r\neq k}}^{n}\sum_{l=1}^{m}{}^{k}\beta_l^r\cdot\phi_l\left(\bar{\xi}\right)\cdot\left\{U_r\right\}\right)=\left\{0\right\} \qquad (21)$$

4.1.2 Derivation of the scalar equation

Then, we carry out the Galerkin projection. That is to say, we project these n equations onto each subspace spanned by the eigenvectors of the tuned system and by each polynomial chaos. And we get something similar to (15):

$$\left\langle {}^{T}\left\{U_j\right\}\cdot(13)_k\,\middle|\,\phi_i\right\rangle \quad \begin{cases}\forall k\in\{1,\ldots,n\}\\ \forall j\in\{1,\ldots,n\}\\ \forall i\in\{1,\ldots,m\}\end{cases} \qquad (22)$$

We recall that the scalar product of the Hilbert space of random parameters, noted $\langle \ | \ \rangle$ is just the expected value of the product of the elements. Then, plugging (21) into (22), we get:

$$\begin{cases} \forall k \in \{1,\ldots,n\} \\ \forall j \in \{1,\ldots,n\} \\ \forall i \in \{1,\ldots,m\} \end{cases}$$

$$\begin{aligned} & {}^{t}\{U_j\}[K]\{U_k\}\cdot\langle\phi_i\rangle + \sum_{\substack{r=1 \\ r\neq k}}^{n}\sum_{l=1}^{m} {}^{k}\beta_l^r \cdot {}^{T}\{U_j\}[K]\{U_r\}\cdot\langle\phi_l|\phi_i\rangle + \sum_{q=1}^{n} {}^{T}\{U_j\}[J_q]\{U_k\}\cdot\langle\xi_q|\phi_i\rangle \\ & -\lambda_k\cdot\sum_{b=1}^{m}\alpha_b^k\,{}^{T}\{U_j\}[I_n]\{U_k\}\cdot\langle\phi_b|\phi_i\rangle + \sum_{q=1}^{n}\sum_{\substack{r=1 \\ r\neq k}}^{n}\sum_{l=1}^{m} {}^{k}\beta_l^r \cdot {}^{T}\{U_j\}[J_q]\{U_r\}\cdot\langle\xi_q|\phi_l|\phi_i\rangle \qquad (23) \\ & -\lambda_k\cdot\sum_{b=1}^{m}\sum_{\substack{r=1 \\ r\neq k}}^{n}\sum_{l=1}^{m}\alpha_b^k\cdot{}^{k}\beta_l^r\cdot{}^{T}\{U_j\}[I_n]\{U_r\}\cdot\langle\phi_b|\phi_l|\phi_i\rangle = \{0\} \end{aligned}$$

4.1.3 Simplification of the system of equations

For the moment we still did not used the particular properties of the basis of $\Re^n$ and Ψ we have used. We need to use the orthogonality between these elements to simplify (23). Hence, we use the orthogonality properties of the modal and polynomial basis vectors:

$$ {}^{T}\{U_i\}[I_n]\{U_j\} = \delta_{ij} \quad (24)$$

$$ {}^{T}\{U_i\}[K]\{U_j\} = \lambda_i\delta_{ij} \quad (25)$$

$$ \langle\phi_i|\phi_j\rangle = 0 \quad \text{if} \quad i \neq j \quad (26)$$

And we note that :

$$ {}^{T}\{U_i\}[J_q]\{U_j\} = (\{U_i\})_q(\{U_j\})_q \quad (27)$$

These simplifications give :

$$\begin{cases} \forall k \in \{1,...,n\} \\ \forall j \in \{1,...,n\} \\ \forall i \in \{1,...,m\} \end{cases}$$

$$\begin{cases} \bullet\ j = k: \\ \lambda_k \cdot \langle \phi_i \rangle + \sum_{q=1}^{n} (\{U_k\})_q^2 \cdot \langle \xi_q | \phi_i \rangle - \lambda_k \cdot \alpha_i^k \langle \phi_i^2 \rangle + \sum_{q=1}^{n} \sum_{\substack{r=1 \\ r \neq k}}^{n} \sum_{l=1}^{m} {}^k\beta_l^r \cdot (\{U_k\})_q (\{U_r\})_q \cdot \langle \xi_q | \phi_l | \phi_i \rangle = \{0\} \\ \bullet\ j \neq k: \\ {}^k\beta_i^j \cdot \lambda_j \cdot \langle \phi_i^2 \rangle + \sum_{q=1}^{n} (\{U_j\})_q (\{U_k\})_q \cdot \langle \xi_q | \phi_i \rangle + \sum_{q=1}^{n} \sum_{\substack{r=1 \\ r \neq k}}^{n} \sum_{l=1}^{m} {}^k\beta_l^r \cdot (\{U_j\})_q (\{U_r\})_q \cdot \langle \xi_q | \phi_l | \phi_i \rangle \\ - \lambda_k \cdot \sum_{b=1}^{m} \sum_{l=1}^{m} \alpha_b^k \cdot {}^k\beta_l^j \cdot \langle \phi_b | \phi_l | \phi_i \rangle = \{0\} \end{cases} \quad (28)$$

4.2 Unification of the formalism to simplify the code

4.2.1 Unification of the known parameters

As we have explained in the previous chapter, we can check that the system of equations (28) has got $m \times n^2$ unknowns and $m \times n^2$ equations separated in n independent systems. So this is a quite large system to solve and it is also a quite large system to build if we want to implement the algorithm. To simplify it, since the known coefficients of the system are used in each of the n in dependent system, we can compute them in a preprocessing part of the code and store them in matrices. Moreover, we can also note here that we get a non-linear system which will be solved by a Newton-Raphson's method.

Next, we simplify the computation of (16) by introducing the following matrices:

- $[A] \in M_{n \times m}(\Re)$ such that:

$$A_{ji} = \lambda_j \cdot \langle \phi_i \rangle + \sum_{q=1}^{n} (\{U_j\})_q^2 \cdot \langle \xi_q | \phi_i \rangle \quad \begin{cases} \forall j \in \{1,...,n\} \\ \forall i \in \{1,...,m\} \end{cases} \quad (29)$$

- $[B] \in M_{n \times m \times d \times m}(\Re)$ such that:

$$B_{jlri} = \sum_{q=1}^{n} \left(\{U_j\}\right)_q \left(\{U_r\}\right)_q \cdot \left\langle \xi_q \middle| \phi_l \middle| \phi_i \right\rangle \quad \begin{cases} \forall j \in \{1,...,n\}, \forall l \in \{1,...,m\} \\ \forall r \in \{1,...,d\}, \forall i \in \{1,...,m\} \end{cases} \qquad (30)$$

- $[C] \in M_{n\times m}(\Re)$ such that:

$$C_{ji} = -\lambda_j \cdot \left\langle \phi_i^2 \right\rangle \quad \begin{cases} \forall j \in \{1,...,n\} \\ \forall i \in \{1,...,m\} \end{cases} \qquad (31)$$

- $[E] \in M_{n\times 1}(\Re)$ such that:

$$E_j = -\lambda_j \quad \forall j \in \{1,...,n\} \qquad (32)$$

- $[D] \in M_{n\times m\times n}(\Re)$ such that:

$$D_{kij} = \sum_{q=1}^{n} \left(\{U_j\}\right)_q \left(\{U_k\}\right)_q \cdot \left\langle \xi_q \middle| \phi_i \right\rangle \quad \begin{cases} \forall k \in \{1,...,n\} \\ \forall i \in \{1,...,m\} \\ \forall j \in \{1,...,n\} \end{cases} \qquad (33)$$

- $[M] \in M_{m\times m\times m}(\Re)$ such that:

$$M_{bli} = \left\langle \phi_b \middle| \phi_l \middle| \phi_i \right\rangle \quad \begin{cases} \forall b \in \{1,...,m\} \\ \forall l \in \{1,...,m\} \\ \forall i \in \{1,...,m\} \end{cases} \qquad (34)$$

Thus, using (29), (30), (31), (32), (33), (34), we can simplify (28) into:

$$\begin{cases} \forall k \in \{1,...,n\} \\ \forall j \in \{1,...,n\} \\ \forall i \in \{1,...,m\} \end{cases} \quad \begin{cases} \bullet\ j = k: \\ A_{ki} + \alpha_i^k \cdot C_{ki} + \sum_{\substack{r=1 \\ r\neq k}}^{n} \sum_{l=1}^{m} {}^k\beta_l^r \cdot B_{klri} = \{0\} \\ \bullet\ j \neq k: \\ -{}^k\beta_i^j \cdot C_{ji} + D_{kij} + \sum_{\substack{r=1 \\ r\neq k}}^{n} \sum_{l=1}^{m} {}^k\beta_l^r \cdot B_{jlri} + E_k \cdot \sum_{b=1}^{m} \sum_{l=1}^{m} \alpha_b^k \cdot {}^k\beta_l^j \cdot F_{bli} = \{0\} \end{cases} \qquad (35)$$

4.2.2 Unification of the unknowns

Numerically, the system (35) is more convenient to solve. Nevertheless we still need to change the formalism of the unknowns to unify it and properly solve the equations using for example the routine 'fsolve' in MatLab [13]. We recall that the unknowns are for each $k \in \{1,..., n\}$:

$$\begin{cases} \{\alpha_i^k\}_{i\in\{1,...,m\}} \\ \{{}^k\beta_i^j\}_{\substack{j\in\{1,...,n\}-\{k\} \\ i\in\{1,...,m\}}} \end{cases} \qquad (36)$$

We unify these unknowns into $\forall k \in \{1,..., n\}$ $[\Gamma_k] \in M_{m\times n}(\Re)$ such that:

$$[\Gamma_k] = \begin{bmatrix} {}^k\beta_1^1 & \cdots & {}^k\beta_1^{k-1} & {}^k\beta_1^{k+1} & \cdots & {}^k\beta_1^n & \alpha_1^k \\ \vdots & \ddots & & & \iddots & \vdots & \vdots \\ \vdots & & \ddots & \iddots & & \vdots & \vdots \\ \vdots & & {}^k\beta_i^{k-1} & {}^k\beta_i^{k+1} & & \vdots & \alpha_i^k \\ \vdots & & \iddots & \ddots & & \vdots & \vdots \\ \vdots & \iddots & & & \ddots & \vdots & \vdots \\ {}^k\beta_m^1 & \cdots & {}^k\beta_m^{k-1} & {}^k\beta_m^{k+1} & \cdots & {}^k\beta_m^n & \alpha_m^k \end{bmatrix} = \left[{}^k\gamma_i^j\right]_{\substack{j\in\{1,...,n\} \\ i\in\{1,...,m\}}} \qquad (37)$$

With this new unknown, the system (35) becomes:

$\forall k \in \{1,..., n\}$

$$\begin{cases} \bullet\ j = k, \forall i \in \{1,..., m\}: \\ A_{ki} + {}^k\gamma_i^n \cdot C_{ki} + \sum_{r=1}^{k-1}\sum_{l=1}^{m} {}^k\gamma_l^r \cdot B_{klri} + \sum_{r=k+1}^{n}\sum_{l=1}^{m} {}^k\gamma_l^{r-1} \cdot B_{klri} = \{0\} \\ \bullet\ j \in \{1,..., k-1\}, \forall i \in \{1,..., m\}: \\ -{}^k\gamma_i^j \cdot C_{ji} + D_{kij} + \sum_{r=1}^{k-1}\sum_{l=1}^{m} {}^k\gamma_l^r \cdot B_{jlri} + \sum_{r=k+1}^{n}\sum_{l=1}^{m} {}^k\gamma_l^{r-1} \cdot B_{jlri} + E_k \cdot \sum_{b=1}^{m}\sum_{l=1}^{m} {}^k\gamma_b^n \cdot {}^k\gamma_l^j \cdot F_{bli} = \{0\} \\ \bullet\ j \in \{k+1,..., n\}, \forall i \in \{1,..., m\}: \\ -{}^k\gamma_i^{j-1} \cdot C_{ji} + D_{kij} + \sum_{r=1}^{k-1}\sum_{l=1}^{m} {}^k\gamma_l^r \cdot B_{jlri} + \sum_{r=k+1}^{n}\sum_{l=1}^{m} {}^k\gamma_l^{r-1} \cdot B_{jlri} + E_k \cdot \sum_{b=1}^{m}\sum_{l=1}^{m} {}^k\gamma_b^n \cdot {}^k\gamma_l^{j-1} \cdot F_{bli} = \{0\} \end{cases} \qquad (38)$$

Then, if the $[K]$ matrix wasn't so special, if some of these eigenvalues was not repeated, we could solve this system for each $k \in \{1,..., n\}$ and we would store the solution into two different matrices. We note that it is possible just to solve for a certain number of k, since we have n independent systems. Then, we define matrices to store the results:

- $[\overline{A}] \in M_{n\times m}(\Re)$ such that: $\overline{a}_{ij} = \alpha_i^j = {}^j\gamma_i^n \quad \forall i \in \{1,..., m\} \quad \forall j \in \{1,..., n\}$ (39)
- $[\overline{B}] \in M_{n\times m\times(n-1)}(\Re)$ such that: $\overline{b}_{kij} = {}^j\beta_i^n = {}^j\gamma_i^n \quad \forall k \in \{1,..., n\} \forall i \in \{1,..., m\} \quad \forall j \in \{1,..., n-1\}$ (40)

4.3 Issues in the implementation of algorithm

4.3.1 Complexity problem

At first sight, it seems that we have all the tools to solve the problem properly. Nevertheless, while solving the system of equations (38) numerically, two problems arise. The first one is the complexity of the computation of the coefficient M_{bli} defined in (34). Indeed, the algorithm to compute one element of this three dimensional matrix is extremely time consuming and this matrix could be really huge for a big system. Actually by using symmetry properties of scalar product, we can show that this complexity is of the form:

$$\frac{1}{6}m^3 + \frac{1}{2}m^2 + \frac{1}{3}m \qquad (41)$$

Where m is defined in (13).

Hence, computational efforts turn out to be huge if we want a good accuracy or if the system is not small. For a middle sized system and a suitable accuracy, it could take weeks to get the M_{bli}coefficients on a typical PC. We note here that we have tried several kinds of algorithms, they are all very computationally demanding. Another solution is to compute by hand the coefficients of this matrix using the symmetry properties. To the order three it is a large amount of algebra but it is still doable. Nevertheless it could easily be the reason of lots of implementation mistakes and for a better accuracy it seems almost impossible

4.3.2 Discontinuity problem

The second problem is a theoretical problem which has already been explained by Sinha [2]. Actually, because of the circular nature of the normalized stiffness matrix of the tuned system ($[K]$), we have several repeated tuned eigenvalues. More precisely, we can say that:

- If n is odd λ_1 is unrepeated but $\lambda_{2i} = \lambda_{2i+1} \quad \forall i \in \left\{1,..., \frac{n-1}{2}\right\}$ (42)
- If n is even λ_1 and λ_n are unrepeated but $\lambda_{2i} = \lambda_{2i+1} \quad \forall i \in \left\{1,..., \frac{n}{2}-1\right\}$ (43)

Sinha [2] explains that there is a discontinuity of these repeated eigenvalues when you vary them as functions of several parameters $\{\xi_i\}_i$. Indeed, as we can see in Figures 2a and 2b, when we vary just one parameter (ξ_1), there is no discontinuity problem, but when two parameters vary (ξ_1 and ξ_2) repeated eigenvalues are not continuous functions any more.

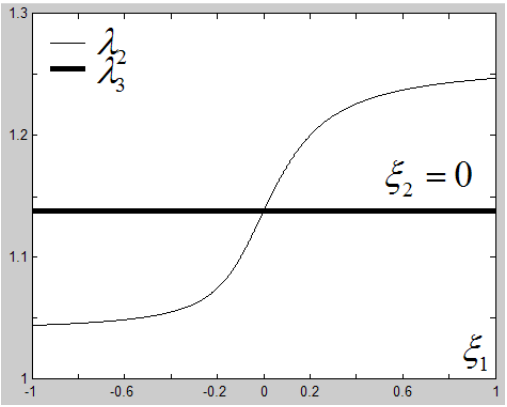


Figure 2a: Variation of two eigenvalues, continuity

Variation of both repeated eigenvalues as a function of just one mistuning parameter

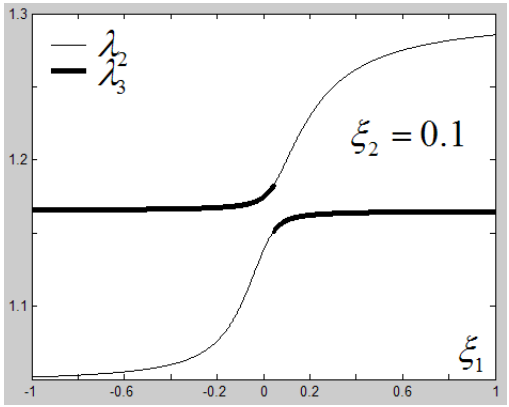


Figure 2b: Variation of two eigenvalues, discontinuity

Variation of both repeated eigenvalues as a function of one mistuning parameter another being constant non zero

As we will see later this becomes a problem in application of a Monte Carlo method. But most importantly it is a problem because it changes the meaning of the definition eigenvalue we have used to derive

equations (38). More precisely, with this definition we cannot write the expansion (16) and (18) any more. Besides, it is also easy to show that this discontinuity problem appears for the eigenvectors.

Chapter 5

One-dimensional algorithm with polynomial chaos

In this chapter, we begin by giving a solution to the complexity and discontinuity problems we have described in chapter 4. Then, we derive the new system of equations for the model of a bladed disk assembly. Finally, we present the formulae to compute the expected values and the standard deviations of the eigenvalues and eigenvectors.

5.1 Resolution of the complexity and discontinuity problems

5.1.1 Continuity of eigenvalues and complexity problems

To solve the discontinuity problem, Sinha [2] explains that we can generalize what we observed in Figure 2a. That is to say, each eigenvalue, irrespective of being repeated or unrepeated, and the corresponding eigenvector (up to certain conditions) is continuous along any radial direction in the mistuning parameter space.

Therefore, to take advantage of this property, we will change this problem from a n mistuning parameters to a one mistuning parameter which belongs to a radial direction of the mistuning parameter space. Thus, even if we lose some generality, this solves the discontinuity problem and it significantly reduces computational efforts. Moreover, we can bring back some generality by using a Monte Carlo simulation where the input random parameter would be the radial direction in the mistuning parameter.

Hence now, we will show how to change algorithm to a unidirectional case. First we choose a mistuning direction into $\Re^n$ for $\overline{\overline{\xi}} = \left(\xi_1, \ldots, \xi_n\right)$. Let us call $\{\chi\} \in \Re^n$ the unit vector which gives this direction. An example of graphical representation for this radial direction in a three dimensional mistuning parameter space is given

in Figure 3. As a remark, we note here that this vector can't be $\{\chi\}=\frac{1}{\sqrt{n}}^{T}\{1...1\}$ otherwise, the stiffness matrix of the random system would remain proportional to the one of the tuned system. Moreover, we choose to introduce a new unique mistuned parameter which is normalized to simplify the computation. We call ρ this new unique variable, such that:

$$\forall q \in \{1,...,n\} \quad \xi_q(\rho)=\chi_q \cdot \sigma\rho \qquad (44)$$

And because of normalization, we write: $\rho = \aleph(0,1)$ (45)

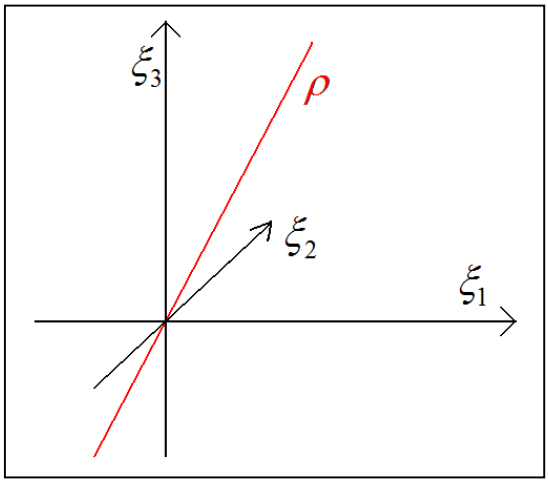


Figure 3: Example of new parameter
Instance of new mistuned parameter in the 3D case

This new normalized random parameter permits to simplify the computation and the manipulation of Hermite polynomial chaos as we can see in Appendix A. Moreover, the number of polynomial chaos is decreased because in formula (8) which gives $m, n=1$. Hence the complexity of the computation of the matrix $[M]$ also decreases a lot.

Besides, it also simplifies the writing of the random stiffness matrix. Thus, with this new notation, the random stiffness matrix becomes:

$$\left[\widetilde{K}(\rho)\right]=\left(\sum_{q=1}^{n}\left[J_q\right]\cdot\chi_q\right)\cdot\sigma\rho=\left[\overline{K}\right]\cdot\sigma\rho \qquad (46)$$

with $$\left[\overline{K}\right]=\begin{bmatrix}\chi_1 & & 0\\ & \ddots & \\ 0 & & \chi_n\end{bmatrix} \qquad (47)$$

5.1.2 Continuity of eigenvectors problem

Nevertheless, we note that the change of mistuning parameter is not enough to have continuous eigenvectors. Indeed, for each repeated eigenvalue, we have two corresponding independent eigenvectors. It is well known and easy to show that any non trivial linear combination of this eigenvectors is also an eigenvector. Actually choosing a pair of eigenvectors turns out to be the same thing as choosing a basis for the two dimensional eigenspace corresponding to the repeated eigenvalue. Andrew and Tan [5] explain that a certain choice for this basis permits to make the eigenvectors continuous along any radial direction of mistuning parameter space. Sinha [2] and they [5] give an algorithm to compute these eigenvectors from any pair of independent eigenvectors.

Let λ_0 be a two times repeated eigenvalue and $(\{v\},\{w\})$ an independent family of corresponding eigenvectors numerically computed. Then we define:

$$\left[X\right]=\left[\{v\},\{w\}\right]\in M_{n\times 2}(\Re) \qquad (48)$$

And

$$\left[Z\right]=\left[X\right]\cdot\left[\Gamma\right]\in M_{n\times 2}(\Re) \quad \text{with} \quad \left[\Gamma\right]\in Orth_2(\Re) \qquad (49)$$

We note that $\left[Z\right]$ is the new family of eigenvectors as a linear combination of the formers eigenvectors. Substituting (49) into (6), we get:

$$\left([K]+\left[\widetilde{K}\right]\right)[Z]=[I_n]\cdot[Z]\cdot[\Lambda] \quad \text{with} \quad [\Lambda]=\lambda_0\cdot[I_2] \qquad (50)$$

Following the process explained in the Sinha's paper [2], we obtain:

$$\underbrace{\left[{}^{T}[X]\frac{d\left[\widetilde{K}\right]}{d\rho}[X]\right]}_{[G]}\cdot[\Gamma]=[\Gamma]\cdot\frac{d[\Lambda]}{d\rho} \qquad (51)$$

To find $[Z]$ the family of eigenvectors, we just have to compute the matrix of eigenvalues $[\Gamma]$ of the eigenproblem (51). Then, we plug it back into (49), and we find the new eigenvectors which are a linear combination of the previous ones. To do so, we note that $[G]$ can be computed in a very simple way using (46):

$$[G]=\sigma^T[X][\overline{K}][X] \qquad (52)$$

5.2 Algorithm for the new one dimensional problem

5.2.1 New system of equations

Now we have solved the two problems. We have obtained a new variable and a new family of eigenvectors for the tuned system. Hence, using these new elements, plugging (44) and (46) into (6) and using the orthogonality properties, we compute a new system of equations similar to (28):

$\forall k \in \{1,\dots,n\}$

$$\begin{cases} \bullet\ j=k: \\ \lambda_k\cdot\langle\phi_i\rangle+\sigma^T\{U_k\}[\overline{K}]\{U_k\}\cdot\langle\rho|\phi_i\rangle-\lambda_k\cdot\alpha_i^k\langle\phi_i^{\,2}\rangle+\sum_{\substack{r=1\\r\neq k}}^{n}\sum_{l=1}^{m}\sigma^k\beta_l^r\cdot{}^T\{U_k\}[\overline{K}]\{U_r\}\cdot\langle\rho|\phi_l|\phi_i\rangle=\{0\} \\ \bullet\ j\neq k: \\ {}^k\beta_i^j\cdot\lambda_j\cdot\langle\phi_i^{\,2}\rangle+\sigma^T\{U_j\}[\overline{K}]\{U_k\}\cdot\langle\rho|\phi_i\rangle+\sum_{\substack{r=1\\r\neq k}}^{n}\sum_{l=1}^{m}\sigma^k\beta_l^r\cdot{}^T\{U_j\}[\overline{K}]\{U_r\}\cdot\langle\rho|\phi_l|\phi_i\rangle \\ -\lambda_k\cdot\sum_{b=1}^{m}\sum_{l=1}^{m}\alpha_b^k\cdot{}^k\beta_l^j\cdot\langle\phi_b|\phi_l|\phi_i\rangle=\{0\} \end{cases} \qquad (53)$$

5.2.2 New formalism for the known parameters

As we have done for the system of equations (28), we introduce new quantities to simplify the previous system and to implement the computation more easily:

- $\left[\widehat{A}\right] \in M_{m \times n}(\Re)$ such that:

$$\widehat{A}_{ij} = \lambda_j \cdot \delta_{1i} \quad \begin{cases} \forall i \in \{1,...,m\} \\ \forall j \in \{1,...,n\} \end{cases} \qquad (54)$$

- $\left[\widehat{B}\right] \in M_{n \times m \times d \times m}(\Re)$ such that:

$$\widehat{B}_{jlri} = \sigma^T \left\{U_j\right\}\left[\overline{K}\right]\left\{U_r\right\} \cdot \left\langle \rho \middle| \phi_i \middle| \phi_l \right\rangle \quad \begin{cases} \forall j \in \{1,...,n\}, \forall l \in \{1,...,m\} \\ \forall r \in \{1,...,d\}, \forall i \in \{1,...,m\} \end{cases} \qquad (55)$$

- $\left[\widehat{C}\right] \in M_{m \times n}(\Re)$ such that:

$$\widehat{C}_{ij} = \lambda_j \cdot \left\langle \phi_i^2 \right\rangle \quad \begin{cases} \forall i \in \{1,...,m\} \\ \forall j \in \{1,...,n\} \end{cases} \qquad (56)$$

- $\left[\widehat{D}\right] \in M_{n \times 1}(\Re)$ such that:

$$\widehat{D}_j = -\lambda_j \quad \forall j \in \{1,..., n\} \qquad (57)$$

- $\left[\widehat{E}\right] \in M_{m \times m \times m}(\Re)$ such that:

$$\widehat{E}_{ibl} = \left\langle \phi_i \middle| \phi_b \middle| \phi_l \right\rangle \quad \begin{cases} \forall i \in \{1,...,m\} \\ \forall b \in \{1,...,m\} \\ \forall l \in \{1,...,m\} \end{cases} \qquad (58)$$

Thus, using the unified unknown $\left[\Gamma_k\right]$ that we have defined in (37) and plugging (54), (55), (56), (57) and (58) into (53), we finally get:

$\forall k \in \{1,..., n\}$

$$\begin{cases} \bullet\ j = k, \forall i \in \{1,\dots,m\}: \\ \widehat{A}_{ik} + \widehat{B}_{k1ki} - {}^{k}\gamma_i^n \cdot \widehat{C}_{ik} + \sum_{r=1}^{k-1}\sum_{l=1}^{m} {}^{k}\gamma_l^r \cdot \widehat{B}_{klri} + \sum_{r=k+1}^{n}\sum_{l=1}^{m} {}^{k}\gamma_l^{r-1} \cdot \widehat{B}_{klri} = \{0\} \\ \bullet\ j \in \{1,\dots,k-1\}, \forall i \in \{1,\dots,m\}: \\ {}^{k}\gamma_i^j \cdot \widehat{C}_{ij} + \widehat{B}_{j1ki} + \sum_{r=1}^{k-1}\sum_{l=1}^{m} {}^{k}\gamma_l^r \cdot \widehat{B}_{jlri} + \sum_{r=k+1}^{n}\sum_{l=1}^{m} {}^{k}\gamma_l^{r-1} \cdot \widehat{B}_{jlri} + \widehat{D}_k \cdot \sum_{b=1}^{m}\sum_{l=1}^{m} {}^{k}\gamma_b^n \cdot {}^{k}\gamma_l^j \cdot \widehat{E}_{bli} = \{0\} \\ \bullet\ j \in \{k+1,\dots,n\}, \forall i \in \{1,\dots,m\}: \\ {}^{k}\gamma_i^{j-1} \cdot \widehat{C}_{ij} + \widehat{B}_{j1ki} + \sum_{r=1}^{k-1}\sum_{l=1}^{m} {}^{k}\gamma_l^r \cdot \widehat{B}_{jlri} + \sum_{r=k+1}^{n}\sum_{l=1}^{m} {}^{k}\gamma_l^{r-1} \cdot \widehat{B}_{jlri} + \widehat{D}_k \cdot \sum_{b=1}^{m}\sum_{l=1}^{m} {}^{k}\gamma_b^n \cdot {}^{k}\gamma_l^{j-1} \cdot \widehat{E}_{bli} = \{0\} \end{cases} \quad (59)$$

Then as we explained before it is possible to solve this non linear system of $m \times n^2$ equations and $m \times n^2$ unknowns by a Newton-Raphson method. It is still a heavy computation but it is doable in a reasonable amount of time. One more time we note that we have n independent systems to solve. For each of them, we solve the results we have got into the matices $[\overline{A}]$ and $[\overline{B}]$ we have defined in (39) and (40).

5.3 Computation of the final results: expected values and standard deviations

Once we have solved the n systems numerically, that is to say when we have got the matrices $[\overline{A}]$ and $[\overline{B}]$, then we plug back the results into (16) and (18) to get an approximation of the probability density functions of the random eigenvalues and eigenvectors. As we have seen before this is the solution to the full problem with just one chosen mistuning parameter projected onto the Ψ subspace. Nevertheless we are not directly interested in this solution. Indeed, practically what is useful is the average value and standard deviation of each eigenvalue and eigenvector of the mistuned systems. But this is easy to get from the approximations of the probability density functions we have got. They are given by the following formulae which are derived in Appendix B:

$\forall k \in \{1,\dots,n\}$

$$\left\langle \tilde{\lambda}_k \right\rangle = \lambda_k \cdot \alpha_1^k \qquad (60)$$

$$\sigma_{\tilde{\lambda}_k} = \left|\lambda_k\right| \cdot \sqrt{\sum_{b=2}^{m} \left(\alpha_b^k\right)^2 \cdot \left\langle \phi_b{}^2 \right\rangle} \qquad (61)$$

$$\left\langle \left\{\tilde{U}_k\right\} \right\rangle = \left\{U_k\right\} + \sum_{\substack{r=1 \\ r\neq k}}^{n} {}^k\beta_l^r \cdot \left\{U_r\right\} \qquad (62)$$

$$\sigma_{\left\{\tilde{U}_k\right\}_q} = \sqrt{\sum_{\substack{r=1 \\ r\neq k}}^{n} \sum_{l=2}^{m} \left({}^k\beta_l^r\right)^2 \cdot \left\langle \phi_l{}^2 \right\rangle \cdot \left\{U_r\right\}_q{}^2 + 2 \cdot \sum_{\substack{r_1=1, r_2<r_1 \\ r_1, r_2 \neq k}}^{n} \sum_{l=2}^{m} {}^k\beta_l^{r_1}\, {}^k\beta_l^{r_2} \cdot \left\langle \phi_l{}^2 \right\rangle \cdot \left\{U_{r_1}\right\}_q \left\{U_{r_2}\right\}_q} \qquad (63)$$

Chapter 6

Processes to check numerical results

In this chapter we want to determine a method to check the results we have got. As it is absolutely impossible to set an experiment to check these results for these problems we often use Monte Carlo simulation. But we will see that it is not always possible because of the peculiarities of the stiffness matrix. Therefore, we also need to use a Taylor series method

6.1 Convergence of the solution

Actually, the first check if our numerical results are acceptable is to see that the process converges when we increase the degree of polynomial chaos. As we will see in the chapter 7, we have very good results for $p = 3$ or $p = 4$, but for higher degrees the solutions seem to diverge. This may imply that the method is wrong. But, even if Karhunen-Loeve expansion assures that the truncated series converge, nothing is said about the way of convergence. More precisely, it is not obvious that the convergence is uniform. We have tried to find something about this way of convergence but nothing is said in the literature we have found. Moreover, the papers we have read that deal with these kinds of methods never use an order higher than four. We could also think of a numerical saturation to explain this problem but even with lots of digits, we observe the same problem at the same level. Hence, this will be a main point of our conclusion. It could be interesting to study the way of convergence of this process.

6.2 Monte Carlo simulation

Nevertheless, we have a process to check our results at least for $p = 3$ or $p = 4$. As we wrote before the more common process to check the results of stochastic problems is to use a Monte Carlo simulation. This is what we have done here. But we have to be aware that the results we get with this method can be meaningless. For example, let us consider Figure 2a where we drawn the variation of repeated eigenvalues as functions of one mistuning parameter. Arbitrarily, we have chosen to call one curve λ_2 and the other λ_3. But, the Monte Carlo simulation is a numerical process that chooses λ_2 and λ_3 arbitrarily, in a different way as we did. For each step, that is to say for each value of the mistuning parameter it chooses λ_2 higher than λ_3. Hence Figure 2a becomes as Figure 4:

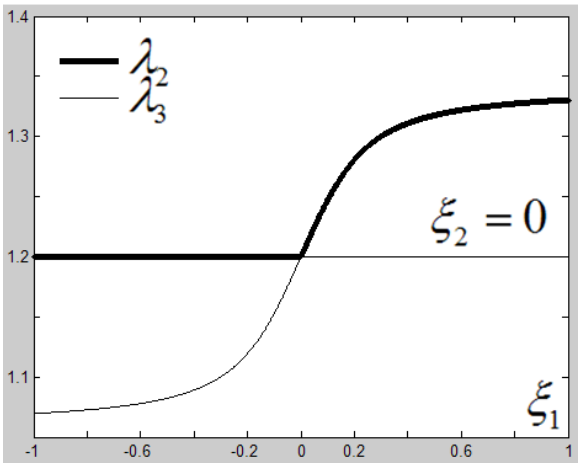


Figure 4: <u>Variation of two eigenvalues, one parameter</u>

Variation of both repeated eigenvalues as a function of just one mistuning Parameter in the case of a numerical computation

Thus, we understand that we cannot compare the results we get with a method which uses the formalism of the Figure 2a and a method which uses the one of the Figure 4. We don't compare the same thing and results from the Monte Carlo simulation will not have any physical meaning in this case. The problem is exactly the same for eigenvectors; we give the same name to two different physical realities.

Nevertheless, we can apply this method for the unrepeated eigenvalues and associated eigenvectors because in this case, we do not have the same problem as long as standard deviation of mistuning parameter σ is small enough not to cause the eigenvalues coalescence. In this case, the method is good to check the expected values and standard deviations of the eigenvalues and eigenvectors.

6.3 Taylor series method

But we still cannot check our results for repeated eigenvalues and associated eigenvectors. Hence to verify the answers we get with the polynomial chaos method, we have adapted the work done by Sinha [2]. In this paper, Sinha has applied a work done by Zhang and Wang [7] to compute the first and/or second order derivatives of the repeated and unrepeated eigenvalues and their corresponding eigenvectors. Then we can expand each eigenvalue and eigenvector as a Taylor series with respect to the normalized mistuning parameter to the second order. Finally, we treat the mistuning parameter as a random variable and we compute an approximation of the expected value and the standard deviation.

The details and the results of this method are given in Appendix D. Theoretically, this is not supposed to give a good result because the variable is not very small. However, numerically we can see that we get something quite accurate by comparing for the unrepeated eigenvalues with what we get with a Monte Carlo simulation. Nevertheless, the algorithm presented by Andrew and Tan [5] just permits to get the first and second derivative of repeated eigenvalues and only the first derivative of corresponding eigenvectors. For the unrepeated eigenvalues we can just compute the second derivative of the eigenvalue and the first derivative of the corresponding eigenvector. This is a problem because it permits to get accurate results for the expected values and standard deviations of the repeated eigenvalues but we do not get any result for the expected values of the corresponding eigenvectors and just a crude approximation of their standard deviations.

6.4 Summary

What we wrote in this chapter, the ways to check the numerical results, is summed up in Table 1 which presents the means we have used to check our results for each case. By the way we note that one of the main advantages of the polynomial chaos method is that we can solve in the same way for each of these cases.

Table 1: Summary of the verification methods
Techniques used to check the results of the polynomial chaos method for each case

	Unrepeated		Repeated	
	$\tilde{\lambda}$	$\{\tilde{U}\}$	$\tilde{\lambda}$	$\{\tilde{U}\}$
$\langle\ \rangle$	Monte Carlo	Monte Carlo	Taylor series	?
σ	Monte Carlo	Monte Carlo	Taylor series	?

Thus, as we can see in Table 1 the main weak point of our verification is that we cannot get a way to check the results we find with the polynomial chaos method for the eigenvectors associated with repeated eigenvalues. And we will see later that we really miss this verification because even for eigenvectors associated with unrepeated eigenvalues the Monte Carlo simulation gives standard deviations far from what we get with the polynomial chaos method.

Chapter 7

Numerical application of the polynomial chaos method

In this chapter we present the numerical applications of the polynomial chaos we have done for a particular example. We plot the results and we deal with the validity of the polynomial chaos technique.

7.1 Parameters values

To begin with, we have chosen to take a system of four oscillators ($n=4$) (Figure 1) so we can observe the results for two repeated and two unrepeated eigenvalues. The mass of each oscillator is $ma = 0.1kg$. The two stiffness are $k_0 = 10000N \cdot m^{-1}$ and $k_c = 100N \cdot m^{-1}$. That is, the coupling parameter is $R = 0.01$ and the natural frequency is $\omega_0 = 316rad \cdot s^{-1}$. The polynomial chaos method is carried out in the mistuning parameter direction $\{\chi\} = \frac{1}{\sqrt{18}}^T \{1,2,3,2\}$, to the order $p = 3$ and with a standard deviation of the Gaussian parameter of 1% ($\sigma = 0.01$). For the Monte Carlo simulation, we have used 10000 iterations for a good accuracy.

7.2 Eigenvalues

The results we have got for the eigenvalues for this example are presented in absolute value and in relative value in Figure 5.

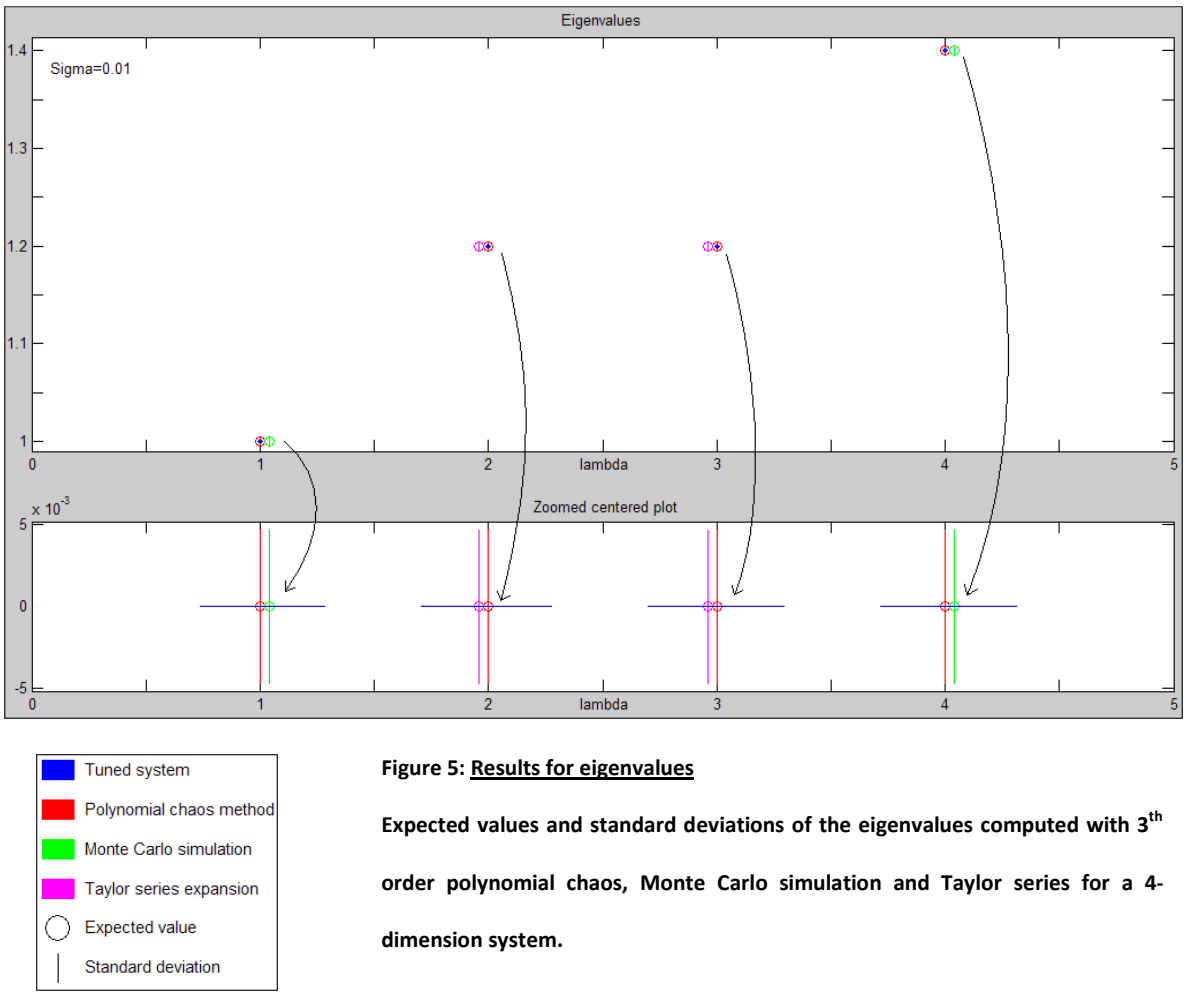


Figure 5: <u>Results for eigenvalues</u>

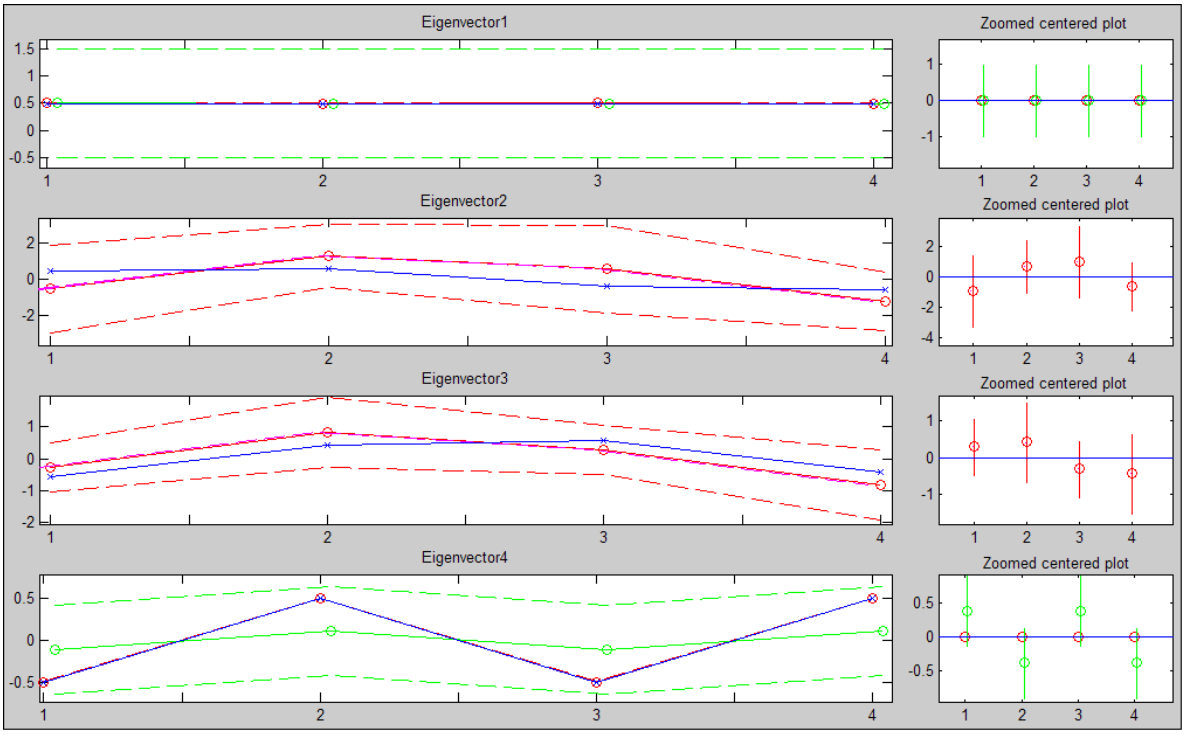
Expected values and standard deviations of the eigenvalues computed with 3th order polynomial chaos, Monte Carlo simulation and Taylor series for a 4-dimension system.

We can see that the result we get with the different methods match perfectly. The standard deviations are almost equal and the average values are pretty close to each other. We can note that even the Taylor series method which was not expected to give good results is very close to the ones we have got with the polynomial chaos method. Moreover, for unrepeated eigenvalues, the polynomial chaos method gives results very close to the Monte Carlo simulation's values.

Nevertheless we have also seen that if we increase the number of polynomial chaos, if we increase the degree of the higher polynomial chaos, the solution diverge as we wrote before.

7.3 Eigenvectors

Then Figure 6 presents the results for the eigenvectors in absolute and relative value.



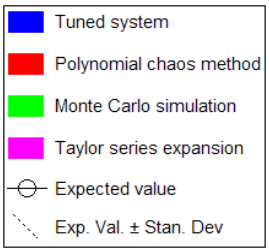


Figure 6: <u>Results for eigenvectors</u>

Expected values and standard deviations of the corresponding eigenvectors computed with 3th order polynomial chaos, Monte Carlo simulation and Taylor series for a 4-dimension system.

In the case of the eigenvectors we see that the results are not as good as the ones we have got for the eigenvalues. Actually, it is difficult to say if the results for the eigenvectors associated with the repeated eigenvalue are good or bad because the method we use to check these results has a very poor accuracy. Nevertheless, we can say that polynomial chaos results are not wrong when we look at the shape of the

eigenvectors of the tuned system. In the case of the eigenvectors associated with the unrepeated eigenvalues, we see first that the expected values given by both methods are pretty similar (up to a multiplicative coefficient). Nevertheless, for the standard deviation, we note a really huge difference between both results. We donot know how to explain this difference.

Conclusion

The polynomial chaos method has been used to compute the expected values and standard deviations of the eigenvalues and associated eigenvectors of a mistuned bladed disk assembly. We have explained the model we have used to represent this mechanical system and we have also given mathematical details of the polynomial chaos method. Applying this technique to the situation we have seen that a problem of complexity and a problem of discontinuity rose. We have used an algorithm to change the eigenvector basis of the repeated eigenvalues and we have transformed our multi-dimensional problem into a one dimensional one to solve these problems. Then we have seen that a Monte Carlo simulation was not enough to check all the results we get with the polynomial chaos method, so we have adapted a Taylor series technique to get results comparable to the ones we wanted to check. Finally, we have been able to implement these techniques in a MatLab [13] (Appendix E) code and have computed the expected values and standard deviations of the eigenvalues and associated eigenvectors in a system with four blades. For a low polynomial order results seem to be correct. Nevertheless, we have noted a divergence of the results when the order of the polynomial chaos increases. This phenomenon is contradictory but we have found no way to explain it.

References

This work has been carried out form the following documents:

[1] - Dessombz O., Diniz A., Thouverez F., Jézéquel L., 2000, *Analysis of stochastic structures: perturbation method and projection on homogeneous chaos*, June 20 2000, Fr CISTI ICIST, pp. 03-09

[2] - Sinha, A., 2006, *Computation of Eigenvalues and eigenvectors of a mistuned bladed disk*, ASME turbo expo 2006: Power for land, sea and air, GT2006-90087

[3] - Ghanem, R. G. and Spanos, P. D., 1990, *Polynomial chaos in stochastic finite element*, Journal of applied mechanics, March 1990, Vol. 57, pp. 197-202

[4] - Sinha A., 2006, *Computation of the statistics of Forced Response of a mistuned bladed disk assembly via polynomial chaos*, Journal of vibration and acoustics, August 2006, Vol. 128, pp. 449-457

[5] - Andrew, A. L. and Tan, R. C. E., 1998, *Computation of the derivatives of repeated eigenvalues and the corresponding eigenvectors of symmetric matrix pencil*, SIAM Journal on Matrix Analysis and Applications, Vol.20, No. 1, pp. 78-100

[6] - Ghanem, R. G. and Spanos, P. D., 1991, *Stochastic Finite Elements: A spectral approach*, Springer-Verlag

[7] - Zhang, Y.-Q. and Wang, W.-L., 1995, *Eigenvector derivatives of generalized nondefective eigenproblems with repeated eigenvalues*, ASME Journal of Engineering for Gas Turbines and Power, Vol. 117, pp. 207-212

[8] - Griffin, J.H., and Hoosac, T. M., 1984, *Model development and statistical investigation of turbine blade mistiming*, ASME Journal of acoustics stress reliability in design, 106, pp. 204-210

[9] - Sinha, A., 1986, *Calculating the statistics of forced response of a mistuned bladed disk assembly*, AIAA J., 24(11), pp. 1797-1801.

[10] - Sinha, A. and Chen, S., 1989, *A higher order technique to calculate the statistics of forced response of a mistuned bladed disk assembly*, Journal of sound and vibrations, 130(2), pp. 207-221

[11] - Pierre, C. and Wei, S.-T., 1988, *Localization phenomena in mistuned assemblies with cyclic symmetry: free vibration / forced vibrations*, Journal of vibration, acoustics, stress, and reliability in design, October 1988, Vol. 110, pp.429-449

[12] - Pierre, C., 1987, Localization *free and forced vibrations of disordered nearly periodic structures*, AIAA paper 87-0774, Proceedings of the 28th AIAA/ASME/ASCE/AHS structures, structural dynamics and materials conference, Monterey, CA

[13] - MATLAB 7.4.0 (R2007a), The Mathwork inc., copyright 1984-2007

[14] - Karhunen, K. ,1947, Kari, Über lineare Methoden in der Wahrscheinlichkeitsrechnung, Ann. Acad. Sci. Fennicae. Ser. A. I. Math.-Phys., 1947, No. 37, 1–79

[15] - Loève, M., 1963, Probability theory, Vol. II, 4th ed., Graduate Texts in Mathematics,

Appendix A

Computation of orthogonal homogeneous normalized Hermite polynomial chaos

In this appendix we present the way to compute the Hermite polynomial chaos of n normalized Gaussian variables to the order p . We also deal with the way to count the number of such polynomials.

To begin, we call $\{\xi_k\}_{k\in\{1,...,n\}}$ the n normalized Gaussian random variables. That is to say, $\forall k \in \{1,..., n\}$ $\xi_k = \aleph(0,1)$. Then a formula to easily get the polynomial chaos of order p is:

$$\phi(\xi_1,..., \xi_n) = (-1)^p e^{\frac{1}{2}\{\xi\}^T\{\xi\}} \frac{\partial^p}{\partial\xi_{i_1}...\partial\xi_{i_p}}\left(e^{-\frac{1}{2}\{\xi\}^T\{\xi\}}\right) \quad \text{with} \quad i_1,...,i_p \in \{1,...,n\} \qquad \text{(a1)}$$

We note that it is possible to find two different similar formulae in [1] and [6] (non-normalized expression of the polynomials), only this one is complete and accurate for normalized polynomial chaos.

We see here that there are as many possible polynomial chaos of order p as possible combination $(i_1,...,i_p)$ without being aware of the order. Hence, to know the number of possible polynomials of order p we can simply solve this basic stochastic problem:

- We have n distinct balls in a bag: $\xi_1,..., \xi_n$
- We take p balls in any order and possibly several time: $\partial\xi_{i_1}..\partial\xi_{i_p}$

  So we get $m' = \begin{pmatrix} n+p-1 \\ p \end{pmatrix}$ possible combinations and also m' possible Hermite polynomial chaos of n variables at the order p .

Then it is easier to find these polynomials. We use a random process to find all the different possible combinations $\partial\xi_{i_1}..\partial\xi_{i_p}$. And then we just use m' times the formula (a1).

Then we can compute the number of polynomial chaos from the order 0 to the order p with n random variables:

$$m = \sum_{k=0}^{p} \binom{n+k-1}{k} = \frac{p+1}{n} \binom{n+p}{1+p} \qquad \text{(a2)}$$

As an example we give here the polynomial chaos to the order p :

- Order 0:

$$1 \qquad \text{(a3)}$$

- Order 1:

$$\xi_1, \ldots, \xi_i, \ldots, \xi_n \;\mid\; i = 1 \to n \qquad \text{(a4)}$$

- Order 2:

$$\xi_1^2 - 1, \ldots, \xi_i^2 - 1, \ldots, \xi_n^2 - 1 \;\mid\; i = 1 \to n \qquad \text{(a5)}$$

$$\xi_1\xi_2, \ldots \xi_i\xi_j, \ldots \xi_{n-1}\xi_n \;\mid\; i = 1 \to n-1 \;;\; j = i+1 \to n \qquad \text{(a6)}$$

- Order 3:

$$\xi_1^3 - 3\xi_1, \ldots, \xi_i^3 - 3\xi_i, \ldots, \xi_n^3 - 3\xi_n \;\mid\; i = 1 \to n \qquad \text{(a7)}$$

$$\xi_1\xi_2\xi_3, \ldots \xi_i\xi_j\xi_k, \ldots, \xi_{n-2}\xi_{n-1}\xi_n \;\mid\; i = 1 \to n-2 \;;\; j = i+1 \to n-1 \;;\; k = j+1 \to n \qquad \text{(a8)}$$

$$-\xi_1 + \xi_1\xi_2^2, \ldots, -\xi_i + \xi_i\xi_j^2, \ldots, -\xi_n + \xi_n\xi_{n-1}^2 \;\mid\; i = 1 \to n \;;\; j = 1 \to n \quad j \neq i \qquad \text{(a9)}$$

Appendix B

Computation of expected values and standard deviations of eigenvalues and eigenvectors

In this appendix we derive the formulae to get the expected value and standard deviation of eigenvalues and eigenvectors from the approximation of the probability density functions we have computed. We recall that these approximations are (equations (16) and (18)):

$$\forall k \in \{1,\ldots, n\} \qquad \tilde{\lambda}_k(\bar{\xi}) = \lambda_k \cdot \sum_{b=1}^{m} \alpha_b^k \cdot \phi_b(\bar{\xi})$$

$$\{\tilde{U}_k(\bar{\xi})\} = \{U_k\} + \sum_{\substack{r=1 \\ r \neq k}}^{n} \sum_{l=1}^{m} {}^k\beta_l^r \cdot \phi_l(\bar{\xi}) \cdot \{U_r\}$$

Since we have computed the coefficients of these both Karhunen-Loeve expansions, we can now calculate an approximation of the expected values and of the standard deviations.

Expected values of the eigenvalues

We directly use the linearity of the operator $\langle\ \rangle$ and the orthogonality of polynomial chaos on (16) to get:

$\forall k \in \{1,\ldots, n\}$

$$\left\langle \tilde{\lambda}_k \right\rangle = \left\langle \lambda_k \cdot \sum_{b=1}^{m} \alpha_b^k \cdot \phi_b \right\rangle = \lambda_k \cdot \sum_{b=1}^{m} \alpha_b^k \cdot \delta_{1b} = \lambda_k \cdot \alpha_1^k \qquad \text{(b1)}$$

Standard deviations of the eigenvalues

From (b1) we get:

$$\left\langle \tilde{\lambda}_k \right\rangle^2 = \left(\lambda_k \cdot \alpha_1^k\right)^2 \qquad \text{(b2)}$$

Then we compute:

$$\tilde{\lambda}_k^{\,2} = \lambda_k^{\,2}\left[\sum_{b=1}^{m}\left(\alpha_b^k\right)^2\cdot\phi_b^{\,2} + 2\cdot\sum_{\substack{b_1=1\\ b_2<b_1}}^{m}\alpha_{b_1}^k\alpha_{b_2}^k\cdot\phi_{b_1}\phi_{b_2}\right] \quad \text{(b3)}$$

Hence using the linearity and orthogonality properties as previously, we get:

$$\left\langle\tilde{\lambda}_k^{\,2}\right\rangle = \left(\lambda_k\cdot\alpha_1^k\right)^2 + \lambda_k^{\,2}\cdot\sum_{b=2}^{m}\left(\alpha_b^k\right)^2\cdot\left\langle\phi_b^{\,2}\right\rangle \quad \text{(b4)}$$

Finally using (b2) and (b4), we directly get the standard deviation:

$\forall k \in \{1,\ldots,n\}$

$$\sigma_{\tilde{\lambda}_k} = \sqrt{\left\langle\tilde{\lambda}_k^{\,2}\right\rangle - \left\langle\tilde{\lambda}_k\right\rangle^2} = \left|\lambda_k\right|\cdot\sqrt{\sum_{b=2}^{m}\left(\alpha_b^k\right)^2\cdot\left\langle\phi_b^{\,2}\right\rangle} \quad \text{(b5)}$$

Expected values of the eigenvectors

We do exactly the same thing for eigenvectors and we directly get the expected value from the linearity and orthogonality properties applied on (18):

$\forall k \in \{1,\ldots,n\}$

$$\left\langle\left\{\tilde{U}_k\right\}\right\rangle = \left\langle\left\{U_k\right\} + \sum_{\substack{r=1\\ r\neq k}}^{n}\sum_{l=1}^{m}{}^k\beta_l^r\cdot\phi_l\cdot\left\{U_r\right\}\right\rangle = \left\{U_k\right\} + \sum_{\substack{r=1\\ r\neq k}}^{n}\sum_{l=1}^{m}{}^k\beta_l^r\cdot\delta_{1l}\cdot\left\{U_r\right\} = \left\{U_k\right\} + \sum_{\substack{r=1\\ r\neq k}}^{n}{}^k\beta_l^r\cdot\left\{U_r\right\} \quad \text{(b6)}$$

Standard deviations of the eigenvalues

We note here that we compute the standard deviation for each element ($q \in \{1,\ldots,n\}$) of each eigenvector. From (b6), we get:

$$\left\langle\left\{\tilde{U}_k\right\}_q\right\rangle^2 = \left\{U_k\right\}^2 + \sum_{\substack{r=1\\ r\neq k}}^{n}\left({}^k\beta_1^r\right)^2\cdot\left\{U_r\right\}_q^{\,2} + 2\cdot\sum_{\substack{r_1=1,r_2<r_2\\ r_1,r_2\neq k}}^{n}{}^k\beta_1^{r_1}\,{}^k\beta_1^{r_2}\cdot\left\{U_{r_1}\right\}_q\left\{U_{r_2}\right\}_q + 2\cdot\sum_{\substack{r=1\\ r\neq k}}^{n}{}^k\beta_1^r\cdot\left\{U_r\right\}_q\left\{U_k\right\}_q \quad \text{(b7)}$$

Then, we calculate:

$$\left\langle \left\{\tilde{U}_k\right\}_q^{\,2} \right\rangle = \left\langle \left\{U_k\right\}_q^{\,2} + \sum_{\substack{r=1 \\ r\neq k}}^{n} \left[\sum_{l=1}^{m} \left({}^{k}\beta_l^{r}\right)^2 \cdot \phi_l^{\,2} + 2 \cdot \sum_{\substack{l_1=1 \\ l_2<l_1}}^{m} {}^{k}\beta_{l_1}^{r}\, {}^{k}\beta_{l_2}^{r} \cdot \phi_{l_1}\phi_{l_2} \right] \cdot \left\{U_r\right\}_q^{\,2} \right.$$

$$\left. + 2 \cdot \sum_{\substack{r_1=1, r_2<r_1 \\ r_1, r_2 \neq k}}^{n} \left(\sum_{l=1}^{m} {}^{k}\beta_l^{r_1} \cdot \phi_l \right) \left(\sum_{l=1}^{m} {}^{k}\beta_l^{r_2} \cdot \phi_l \right) \cdot \left\{U_{r_1}\right\}_q \left\{U_{r_2}\right\}_q + 2 \cdot \sum_{\substack{r=1 \\ r\neq k}}^{n} \sum_{l=1}^{m} {}^{k}\beta_l^{r} \cdot \phi_l \cdot \left\{U_r\right\}_q \left\{U_k\right\}_q \right\rangle \qquad \text{(b8)}$$

$$= \left\{U_k\right\}_q^{\,2} + \sum_{\substack{r=1 \\ r\neq k}}^{n} \sum_{l=1}^{m} \left({}^{k}\beta_l^{r}\right)^2 \cdot \left\langle \phi_l^{\,2} \right\rangle \cdot \left\{U_r\right\}_q^{\,2} + 2 \cdot \sum_{\substack{r_1=1, r_2<r_1 \\ r_1, r_2 \neq k}}^{n} \sum_{l=1}^{m} {}^{k}\beta_l^{r_1}\, {}^{k}\beta_l^{r_2} \cdot \left\langle \phi_l^{\,2} \right\rangle \cdot \left\{U_{r_1}\right\}_q \left\{U_{r_2}\right\}_q$$

$$+ 2 \cdot \sum_{\substack{r=1 \\ r\neq k}}^{n} {}^{k}\beta_1^{r} \cdot \left\{U_r\right\}_q \left\{U_k\right\}_q$$

Finally we get from (b7) and (b8):

$$\sigma_{\left\{\tilde{U}_k\right\}_q} = \sqrt{\sum_{\substack{r=1 \\ r\neq k}}^{n} \sum_{l=2}^{m} \left({}^{k}\beta_l^{r}\right)^2 \cdot \left\langle \phi_l^{\,2} \right\rangle \cdot \left\{U_r\right\}_q^{\,2} + 2 \cdot \sum_{\substack{r_1=1, r_2<r_1 \\ r_1, r_2 \neq k}}^{n} \sum_{l=2}^{m} {}^{k}\beta_l^{r_1}\, {}^{k}\beta_l^{r_2} \cdot \left\langle \phi_l^{\,2} \right\rangle \cdot \left\{U_{r_1}\right\}_q \left\{U_{r_2}\right\}_q} \qquad \text{(b9)}$$

Actually this last formula is not very convenient to implement. It could be equally efficient to directly use the algorithm to compute the expected value of any polynomial of multi random variables. This algorithm is explained in Appendix C.

Appendix C

Computation of expected values of product of polynomial chaos

In this appendix we give an algorithm to compute the expected value of a product of polynomial chaos. This is particularly useful in preprocessing to calculate $[F]$ defined in (22) or $[\widehat{E}]$ defined in (58). The goal in this is case to get the value of $\langle \phi_i | \phi_j | \phi_k \rangle$. It also could be useful in postprocessing to calculate the expected value and standard deviation of the eigenvectors and eigenvalues.

The algorithm given here permits to compute the expected value of any polynomial of multi random normalized variables in a general way. Hence, we first assume that the random variables of the polynomials are normalized and Gaussian. That is to say, $\forall k \in \{1,\ldots, n\}\ \ \xi_k = \aleph(0,1)$. Moreover, we assume that ξ_k and $\xi_{k'}$ are independent stochastic variables if $k \neq k'$. Then we can write the scalar product of three polynomial chaos in a general way:

$$\langle \phi_i | \phi_j | \phi_k \rangle = \left\langle \sum_a g_a \prod_{b_a=1}^{n} \xi_{b_a}{}^{n_{b_a}} \right\rangle \qquad \text{(c1)}$$

We use the linearity of the scalar product and the property of expected value about decoupled stochastic variables to change (c1) into:

$$\langle \phi_i | \phi_j | \phi_k \rangle = \sum_a g_a \prod_{b_a=1}^{n} \left\langle \xi_{b_a}{}^{n_{b_a}} \right\rangle \qquad \text{(c2)}$$

Thus to find the value of (c2), we need to compute the expected value of a normalized Gaussian variable to a certain power. To do so, we use the moments of the normal distribution and we get the following classical result:

$$\left\langle \xi_\alpha{}^\beta \right\rangle = \begin{cases} 0 & \text{if } \beta \text{is odd} \\ \dfrac{\beta!}{2^{\frac{\beta}{2}}\left(\frac{\beta}{2}\right)!} & \text{if } \beta \text{is even} \end{cases} \qquad \text{(c3)}$$

Then we plug the results of (c3) into (c2) and we get the expected values we are looking for. We note that for the special case of the computation of $\left\langle \phi_i \middle| \phi_j \middle| \phi_k \right\rangle$, we do not have m^3 different cases to compute. Actually because of symmetry properties, we just have $\frac{1}{6}m^3 + \frac{1}{2}m^2 + \frac{1}{3}m$ distinct cases. This reduces the computational burden to obtain $[F]$(22) and $\left[\widehat{E}\right]$(58).

Appendix D

Taylor series method to compute expected value and standard deviation of eigenvalues

This appendix gives the algorithm to get an approximation of the expected value and the standard deviation of certain eigenvalues and eigenvectors. It may not give a good approximation but it permits us to have an idea of the value of the true results for cases for which Monte Carlo simulation does not work. First, we note that study in this case considers just one random parameter. Specifically, we study the following random system:

$$\left(\underbrace{[K] + \overbrace{\left[\widetilde{K}(\rho)\right]}^{=\sigma[\bar{K}]\rho}}_{=[\wp(\rho)]} - \tilde{\lambda}(\rho)\cdot[I_n] \right)\left\{\widetilde{U}(\rho)\right\} = \{0\} \qquad \text{(d1)}$$

Where ρ is a normalized and centered Gaussian random parameter, that is to say, $\rho = \aleph(0,1)$. Here again we call $(\lambda_k, \{U_k\})_{k\in\{1,\ldots,n\}}$ the eigenvalues and eigenvectors of the tuned system. We can show that in the special case of our physical system (equations (42) and (43)):

- If n is odd λ_1 is unrepeated but $\lambda_{2i} = \lambda_{2i+1} \ \ \forall i \in \left\{1,\ldots,\frac{n-1}{2}\right\}$
- If n is even λ_1 and λ_n are unrepeated but $\lambda_{2i} = \lambda_{2i+1} \ \ \forall i \in \left\{1,\ldots,\frac{n}{2}-1\right\}$

We will treat ρ as a linear parameter of the eigenvalues and eigenvectors. So we can expand both as a Taylor series as follows:

$$\tilde{\lambda}_k(\rho) = \lambda_k + \left.\frac{d\tilde{\lambda}_k}{d\rho}\right|_{\rho=0}\rho + \frac{1}{2}\left.\frac{d^2\tilde{\lambda}_k}{d\rho^2}\right|_{\rho=0}\rho^2 + o\left(\rho^2\right) \qquad \text{(d2)}$$

$\forall k \in \{1,\ldots,n\}$

$$\left\{\widetilde{U}_k(\rho)\right\} = \{U_k\} + \left.\frac{d\left\{\widetilde{U}_k\right\}}{d\rho}\right|_{\rho=0}\rho + \frac{1}{2}\left.\frac{d^2\left\{\widetilde{U}_k\right\}}{d\rho^2}\right|_{\rho=0}\rho^2 + o\left(\rho^2\right) \qquad \text{(d3)}$$

Then, we will treat equations (d2) and (d3) statistically. It may not give very good results because ρ is not always very small but it can provide a check on the results obtained with the polynomial chaos method. To compute the first and second derivatives of the eigenvectors and eigenvalues, we will use the method given by Andrew and Tan [5] and Zhang and Wang [7]. We note that a simplification of this method applied for two time repeated eigenvalues is given by Sinha [2]. In our case we can simplify further.

First, we want to compute first derivatives of repeated eigenvalues. Hence, for each $i \in \left\{1,..., \frac{n-1}{2}\right\}$ or $i \in \left\{1,..., \frac{n}{2}-1\right\}$, it depends if n is odd or even, we build:

$$[X_i]=[\{U_{2i}\}\{U_{2i+1}\}] \qquad \text{(d4)}$$

$$\text{and} \quad [Z_i]=[X_i][\Gamma_i] \quad \text{with} \quad [\Gamma_i] \in Orth_2(\Re) \qquad \text{(d5)}$$

Actually $[Z_i]$ is just a matrix of two vectors which are a non singular linear combination of eigenvectors of the tuned system; e.i. it is still a matrix of two eigenvectors of this system.

Then we build:

$$[G_i]=\sigma^{\cdot T}[X_i][\overline{K}][X_i] \qquad \text{(d6)}$$

And Sinha [2] shows that

$$[G_i][\Gamma_i]=[\Gamma_i][d\Lambda_i] \qquad \text{(d7)}$$

$$\text{Where } [d\Lambda_i]=\begin{bmatrix} \left.\frac{d\tilde{\lambda}_{2i}}{d\rho}\right|_{\rho=0} & 0 \\ 0 & \left.\frac{d\tilde{\lambda}_{2i+1}}{d\rho}\right|_{\rho=0} \end{bmatrix} \qquad \text{(d8)}$$

Thus, finding the first derivatives of repeated eigenvalues turns out to find the eigenvalues of the system (d7). We can easily numerically solve this with MatLab [13]. Then, we want to derive the second derivatives of these eigenvalues. One more time, Sinha [2] shows that:

$$\left[d^2\Lambda_i\right]=2^T\left[Z_i\right]\left(\sigma\left[\overline{K}\right]\right)\left[d\Lambda_i\right]-2^T\left[Z_i\right]\left[dZ_i\right]\left[d\Lambda_i\right] \qquad \text{(d9)}$$

Hence, now to compute the second derivatives, we need to know the first derivatives of the eigenvectors ($\left[dZ_i\right]$), which are obtained from this equation:

$$\underbrace{\left(\left[K\right]-\lambda_i\left[I_n\right]\right)}_{\left[V_i\right]}\left[dZ_i\right]=\underbrace{\left(\left[X_i\right]^T\left[X_i\right]-\left[I_n\right]\right)}_{\left[T_i\right]}\left(\sigma\left[\overline{K}\right]\right)\left[Z_i\right] \qquad \text{(d10)}$$

Then, to solve (d10), for $\left[dZ_i\right]$, we need to find the homogeneous solution ($\left[Z_i\right]\left[S_i\right]$) and a particular one ($\left[W_i\right]$) as follow:

$$\left[dZ_i\right]=\left[W_i\right]+\left[Z_i\right]\left[S_i\right] \qquad \text{(d11)}$$

We also build:

$$\left[U_i\right]=2^T\left[Z_i\right]\left(\sigma\left[\overline{K}\right]\right)\left[W_i\right]-2^T\left[Z_i\right]\left[W_i\right]\left[d\Lambda_i\right] \qquad \text{(d12)}$$

Sinha [2] gives us:

$$s_{12}=\frac{u_{12}}{2\left(\left.\frac{d\tilde{\lambda}_{i+1}}{d\rho}\right|_{\rho=0}-\left.\frac{d\tilde{\lambda}_i}{d\rho}\right|_{\rho=0}\right)},\ s_{21}=\frac{u_{21}}{2\left(\left.\frac{d\tilde{\lambda}_i}{d\rho}\right|_{\rho=0}-\left.\frac{d\tilde{\lambda}_{i+1}}{d\rho}\right|_{\rho=0}\right)} \text{ and } s_{jj}=-^T\left\{z_j\right\}\left\{w_j\right\} \text{ with } j\in\{1,2\} \qquad \text{(d13)}$$

Such that $\left[Z_i\right]=\left[\left\{z_1\right\}\left\{z_2\right\}\right]$, $\left[W_i\right]=\left[\left\{w_1\right\}\left\{w_2\right\}\right]$ and s_{jk} and u_{jk} are the coefficients of the matrices $\left[S_i\right]$ and $\left[U_i\right]$. Hence, numerically we can easily find a particular solution for (d10). (d13) gives us the homogeneous solution. And finally (d11) gives the full solution for $\left[dZ_i\right]$. Then, from (d9) we can get the second derivatives of the repeated eigenvalues.

We need now to compute the derivative of the unrepeated eigenvalues(s). Nevertheless, we can show that we can just get the second derivative. We will see later that this is not a problem for the computation of an approximation of the expected value of the random eigenvalues. But without this value we cannot get a suitable approximation for the standard deviation of this parameter.

From Sinha [2], we get:

$$\frac{d^2\tilde{\lambda}_i}{d\rho^2} = \frac{d^T\{U_i\}}{d\rho}\left(\sigma[\overline{K}]\right)\{U_i\} + {}^T\{U_i\}\left(\sigma[\overline{K}]\right)\frac{d\{U_i\}}{d\rho} \qquad \text{(d14)}$$

Thus to get the second derivative of the eigenvalues we need first the first derivative of the corresponding eigenvector. This value is given by:

$$\frac{d\{U_i\}}{d\rho} = [P_i]^{-1}\{Q_i\} \qquad \text{(d15)}$$

where $[P_i] = {}^T[\{U_1\},\cdots,\{U_{i-1}\},2\cdot\{U_i\},\{U_{i+1}\},\cdots,\{U_n\}]$ and $\{Q_1\} = {}^T\{Q_1,\cdots,Q_{i-1},0,Q_{i+1}\cdots,Q_n\}$ (d16)

with $Q_j = \frac{-1}{\lambda_j - \lambda_i}\left({}^T\{U_j\}\left(\sigma[\overline{K}]\right)\{U_i\}\right) \quad j \in \{1,\ldots,i-1,i+1,\ldots,n\}$ (d17)

Now we plug (d15) into (d14) and we get the second derivative of an unrepeated eigenvalues.

Then we can derive the expected value and standard deviation for the eigenvalues from (d2) and (d3). Indeed, we can directly write:

$$\left\langle\tilde{\lambda}_k\right\rangle \approx \lambda_k + \frac{1}{2}\left.\frac{d^2\tilde{\lambda}_k}{d\rho^2}\right|_{\rho=0} \qquad \text{(d18)}$$

Then, we compute the standard deviation:

$$\left\langle\tilde{\lambda}_k\right\rangle^2 \approx {\lambda_k}^2 + \frac{1}{4}\left(\left.\frac{d^2\tilde{\lambda}_k}{d\rho^2}\right|_{\rho=0}\right)^2 + \lambda_k\left.\frac{d^2\tilde{\lambda}_k}{d\rho^2}\right|_{\rho=0} \qquad \text{(d19)}$$

$$\tilde{\lambda}_k{}^2(\rho) = \lambda_k{}^2 + \left(\left. \frac{d\tilde{\lambda}_k}{d\rho} \right|_{\rho=0} \right)^2 \rho^2 + 2\lambda_k \left. \frac{d\tilde{\lambda}_k}{d\rho} \right|_{\rho=0} \rho + \lambda_k \left. \frac{d^2\tilde{\lambda}_k}{d\rho^2} \right|_{\rho=0} \rho^2 + o(\rho^2) \qquad \text{(d20)}$$

Hence,

$$\left\langle \tilde{\lambda}_k{}^2 \right\rangle \approx \lambda_k{}^2 + \left(\left. \frac{d\tilde{\lambda}_k}{d\rho} \right|_{\rho=0} \right)^2 + \lambda_k \left. \frac{d^2\tilde{\lambda}_k}{d\rho^2} \right|_{\rho=0} \qquad \text{(d21)}$$

Finally,

$$\sigma_{\tilde{\lambda}_k} \approx \sqrt{\left(\left. \frac{d\tilde{\lambda}_k}{d\rho} \right|_{\rho=0} \right)^2 - \frac{1}{4}\left(\left. \frac{d^2\tilde{\lambda}_k}{d\rho^2} \right|_{\rho=0} \right)^2} \qquad \text{(d22)}$$

We can do exactly the same thing for each element of the eigenvectors and we get:

$\forall k \in \{1,\ldots, n\}$

$$\left\langle \{\tilde{U}_k\}_q \right\rangle \approx \{U_k\}_q + \frac{1}{2} \left. \frac{d^2\{\tilde{U}_k\}_q}{d\rho^2} \right|_{\rho=0} \qquad \text{(d23)}$$

$$\sigma_{\{\tilde{U}_k\}_q} \approx \sqrt{\left(\left. \frac{d\{\tilde{U}_k\}_q}{d\rho} \right|_{\rho=0} \right)^2 - \frac{1}{4}\left(\left. \frac{d^2\{\tilde{U}_k\}_q}{d\rho^2} \right|_{\rho=0} \right)^2} \qquad \text{(d24)}$$

Finally we can again simplify the expressions (d18), (d22), (d23) and (d24) by vanishing the two order term. In the case of the expected value we see that we get a very bad result because the approximation is just the value of the tuned mode. Nevertheless, in the case of the standard deviation this approximation is not so bad and could at least gives an idea of the value of the true solution when we can use a Monte Carlo simulation.

Appendix E

MatLab code for the one dimension case

This appendix presents the following codes:

- Main.m
- FuncNewEigVec.m
- FuncPolynomialChaoses.m
- FuncExpectedVal.m
- FuncGetExpected.m
- FuncGetB.m
- FuncGetC.m
- FuncEquationk.m
- FuncStandardDev.m
- FuncStandardDevVal.m
- FuncEigenValMC.m
- FuncRandEigVec.m
- FuncEigenValVecTS.m
- FuncExpStanDevVec.m
- FuncEigenVecMC.m
- ParticSol.m
- sym2strModif.m

```
clear all
close all

%Definition of the global parameters and symbolic parameters:
global n p m k sigma K Kbar Chi Lambda U PolCha Expected A B C D E L Bbar
syms Polcha LambdaTild

%%%%%%%%%%%%%%%%%%%

%Parameters for the problem (%c means changeable):
%%Physical parameters:
n=4;%c                  %Number of oscillators (n belongs to |[3,99]|)
ma=0.1;%c               %Mass of each oscillator
kc=100;%c               %Compling stiffness
k0=10000;%c             %Average stiffness of each oscillator
sigma=0.01;%c           %Standard deviation of the stiffness mistake (sigma belongs to [0,1])

%%Resolution parameters:
p=3;%c                          %Order of the polynomial chaoses (p>0)
acMC=10000;%c                   %Number of iteration in the Monte Carlo resolution
digits(30);%c                   %Number of digits we want to use for the resolution (32 otherwise)
Chiprim=[1 2 3 2];%c            %Direction of the resolution (n by 1 vector)

%%Built parameters:
R=kc/k0;                        %Coupling parameter
omega0=sqrt(k0/ma);             %Natural frequency of each oscillator
Chi=1/norm(Chiprim)*Chiprim;    %Normalized direction of resolution
Kbar=diag(Chi);                 %Perturbation matrix (n by n diagonal matrix)

%%%%%%%%%%%%%%%%%%%

%Computation for the tuned system:
%%Matrix of the tuned system:
K1=(2*R+1)*ones(1,n);                   %Convenient vector
K2=-R*ones(1,n-1);                      %Convenient vector
K=diag(K1)+diag(K2,1)+diag(K2,-1);      %Building of the tuned system matrix
K(1,n)=-R;                              %Building of the tuned system matrix
K(n,1)=-R;                              %Building of the tuned system matrix

%%Solution for the tuned system:
tic
[Vec,Val]=eig(K);               %Solution of the eigen problem with the tuned matrix
U=FuncNewEigVec(Vec);           %Matrix of the eigenvectors for the tuned system
Lambda=diag(Val);               %Vector of the eigenvalues for the tuned system
t0=toc;
display(strcat('Tuned system, eigenvalues and eigenvectors built: ',num2str(t0),'s'))

%Building of the m' polynomial chaoses
tic
m=1+p;                                      %Number of different polynomials chaoses up to p order
PolCha=FuncPolynomialChaoses(p);            %Function which gives the m' polynomials stored in Polcha
t1=toc;                                     %Polcha is the vector to store the polynomials
display(strcat('Polynomial chaos built:',num2str(t1),'s'))

%%%%%%%%%%%%%%%%%%%

%Building of the different matrix coefficients:

%%Expected matrix:
tic
x=1:1:floor(3*p/2);                         %First we build the matrix of the values of <rho^(2*k)>
L=factorial(2.*x)./2.^(x)./factorial(x);
clear('x');
Expected=FuncGetExpected(m);                %Function which gives the Expected matrix
t2=toc;
display(strcat('Matrix Expected built: ',num2str(t2),'s'))

%%A matrix:
A=zeros(m,n);
A(1,:)=Lambda';                             %We directly get the matrix coefficient A
display(strcat('Matrix A built:~0s'))
```

```
%%B matrix:
tic
B=FuncGetB(n,m);                            %Function which gives the matrix coefficient B
t3=toc;
display(strcat('Matrix B built: ',num2str(t3),'s'))

%%C matrix:
tic
C=FuncGetC(n,m);                            %Function which gives the matrix coefficient C
t4=toc;
display(strcat('Matrix C built: ',num2str(t4),'s'))

%%D matrix:
D=-Lambda;                                  %We get it directly
display('Matrix D built:~0S ')

%%E matrix:
E=Expected;                                 %We note in this case that E is simplified as Expected
display('Matrix E built:~0s')

%%%%%%%%%%%%%%%%%%%

%Building and resolution of each k system:
Abar=zeros(m,n);                                        %Matrix to store the alpha's values (coefficients for eigenvalues)
Bbar=zeros(n,m,n-1);                                    %Matrix to store the beta's values (coefficients for eigenvectors)
for k=1:n
    Gammak0=ones(m,n);                                  %Beginning solution for the Newton Raphson algorithm
    options.MaxFunEvals=10000;                          %Control the maximum number of functions evaluated in fsolve
    options.TolFun=1e-20;                               %Control the accuracy of the result given by fsolve
    options.MaxIter=1000;                               %Control the maximum number of iterations in the fsolve algorithm
    tic
    Gammak = fsolve(@FuncEquationk,Gammak0,options);    %Building of the kth system of equation with FuncEquationk
    tk=toc;                                             %and resolution via the fsolve function
    display(strcat('System solved for k=',num2str(k),':',num2str(tk),'s'))
    Abar(:,k)=double(Gammak(:,n));                      %Storage of the solution for the kth eigenvalue
    Bbar(k,:,:)=double(Gammak(:,1:n-1));                %Storage of the solution for the kth eigenvector
    clear('Gammak');
end

%%%%%%%%%%%%%%%%%%%

%Building and analysis of the random eigenvalues:

%%Building of the random eigenvalue functions:
for k=1:n
    LambdaTild(1,k)=vpa(Lambda(k)*dot(Abar(:,k),PolCha));   %Approximation of the probability density function of eigenvalues
end                                                         %it is not directly used here

%%Analysis of the random eigenvalue functions:
LambdaMu=zeros(n,1);                                        %Expected values of random eigenvalues
LambdaSigma=zeros(n,1);                                     %Standard deviations of random eigenvalues
for k=1:n
    LambdaMu(k)=Lambda(k)*Abar(1,k);                        %Computation of the expected value for random eigenvalue functions
    LambdaSigma(k)=FuncStandardDevVal(Abar(:,k),Lambda(k)); %Computation of the standard deviation for each
end                                                         %random eigenvalue function

%%Analysis of the system's eigenvalues by Monte Carlo method:
tic
[LambdaMuMC,LambdaSigmaMC]=FuncEigenValMC(acMC,sigma);      %Computation of the averaged value and the standard deviation
t5=toc;                                                     %for the eigenvalues of the random system with a
display(strcat('MonteCarlo simulation for eigenvalues acheived:',num2str(t5),'s'))  %Monte Carlo simulation

%%Analysis of the system's eigenvalues by Taylor Series method:
tic
[LambdaMuTS,LambdaSigmaTS,USigmaTS]=FuncEigenValVecTS(K);   %Computation of the expected value and the standard deviation for
t6=toc;                                                     %eigenvalues (and vectors) of the random system from a Taylor series
display(strcat('Taylor Series for eigenvalues acheived:',num2str(t6),'s'))

%%Visualization of the results for eigenvalues:
FuncVisualizeEigVal(LambdaMu,LambdaSigma,LambdaMuMC,LambdaSigmaMC,LambdaMuTS,LambdaSigmaTS,Lambda);
display('PAUSE, press a key to continue')                   %Function to graphicaly compare the eigenvalues
pause                                                       %for the tuned and random system (3 methods)
```

```
%%%%%%%%%%%%%%%%%%

%Building and analysis of the random eigenvalues:

%%Building of the random eigenvector functions:
UTild=FuncRandEigVec(n,m);                                  %Computation of the approximated probability density function

%%Analysis of the random eigenvector functions:
[UMu,USigma]=FuncExpStanDevVec(UTild);                      %Computation of expected values and standard deviation

%%Analysis of the random eigenvector functions by Monte Carlo method:
tic
[UMuMC,USigmaMC]=FuncEigenVecMC(acMC,sigma);                %Computation of the averaged value and the standard deviationthe
t7=toc;                                                     %eigenvalues of the random system with a Monte Carlo simulation
display(strcat('MonteCarlo simulation for eigenvectors acheived:',num2str(t7),'s'))

%%Visualization of the results for eigenvectors:
FuncVisualizeEigVec(UMu,USigma,UMuMC,USigmaMC,U,USigmaTS);  %Function to graphically compare the eignevectors for the
                                                            %tuned and random system
```

```
function [U]= FuncNewEigVec(Vec)
%This function gives a new set of eigenvectors associated with
%multiple eigenvalues to get a smooth variation wrt rho

%Definition of the global parameters:
global n Kbar sigma

%We look for the number of repeated eigenvalues:
if rem(n,2)==1
    l=(n-1)/2;      %Number of repeated eigenvalues (odd case)
else
    l=n/2-1;        %Number of repeated eigenvalues (even case)
end

%Step by step changement of eignevectors for each double pair:
for it1=1:l
    G=sigma*Vec(:,2*it1:2*it1+1)'*Kbar*Vec(:,2*it1:2*it1+1);    %Convenient matrix to build
    [Gamma,Useless]=eig(G);                                     %Computatio of its eigenvectors
    Z=Vec(:,2*it1:2*it1+1)*Gamma;                               %Building of the Z matrix
    Vec(:,2*it1:2*it1+1)=Z;                                     %Changement of eigenvectors
end
U=Vec;
```

```
function [PolCha]= FuncPolynomialChaoses(p)
%Function which permits to get the polynomial
%chaoses to the pth order (with p>0)

%Definition of the global parameters:
syms rho PolCha;

%Polynomial chaos at the order 0:
PolCha(1,1)=1;

%Computation of the polynomial chaoses by derivation to the order p:
temp=exp(-1/2*rho^2);                                               %Convenient function to build
for it2=1:p
    temp=diff(temp,'rho');                                          %Derivation it2 times
    PolCha(1,it2+1)=expand(simplify((-1)^it2*temp/exp(-1/2*rho^2)));    %Storage in PolCha
end
```

```
function [Poly]= FuncExpectedVal(Poly1,Poly2,Poly3)
%This function permits to get the expected value
%of the product of 3 polynomial chaoses

%Definition of the global parameters:
global n L

%Polynomial product:
syms Poly
Poly=sym2strModif(expand(Poly1*Poly2*Poly3));       %We multiply and expand the 3 polynomials and
                                                    %we convert it into a string expression
%Constants, storage matrix and initialization of the iterators:
l=numel(Poly);              %Number of chars into the expression of the polynomial
Exponents=zeros(10*n,1);    %Matrix to store each different exponent which appears in the expression
it1=1;                         %of Poly, such that: rho^j
it3=1;

%Filling in of the matrix Exponents:
while it1<l                 %We go through the string and we detect the different exponent to store
    if Poly(it1)=='r'
        if it1+3<l
            if Poly(it1+3)=='^'
                temp1a='0';
                temp1b=Poly(it1+4);
                if it1+5<=l
                    if Poly(it1+5)~='-'&&Poly(it1+5)~='+'&&Poly(it1+5)~='*'
                        temp1a=temp1b;
                        temp1b=Poly(it1+5);
                        it1=it1+6;
                    else
                        it1=it1+5;
                    end
                end
            else
                temp1a='0';
                temp1b='1';
                it1=it1+3;
            end
        else
            temp1a='0';
            temp1b='1';
            it1=it1+3;
        end

        %Storage of the exponent if it isn't allready in the list:
        temp1=vpa(strcat(temp1a,temp1b));
        it2=1;
        while temp1~=Exponents(it2)&&Exponents(it2)~=0
            it2=it2+1;
        end
        if Exponents(it2)==0
            Exponents(it2)=temp1;
        end

    end
    it1=it1+1;
end
Exponents=sort(Exponents,'descend');

%Substitution of the xi^j's and computation of the expected value:
while Exponents(it3)~=0
    if rem(Exponents(it3),2)==1                 %We go throught the Exponents matrix and each time
        temp2='0';                              %it is even we substitute xi^j by L(j/2). Otherway, we substitute it
    else                                        %by 0 (that are the expected values of xi^j for j odd)
        temp2=num2str(L(Exponents(it3)/2));
    end
    if Exponents(it3)~=1
        Poly=strrep(Poly,strcat('rho','^',num2str(Exponents(it3))),temp2);
    else
        Poly=strrep(Poly,'rho',temp2);
    end
    it3=it3+1;
end
Poly=vpa(Poly);     %Finally, we convert it into a number.
```

```
function [Expected]= FuncGetExpected(m)
%This function permits to get the matrix of all
%the possible values for the expected value

%Definition of the global parameters:
global PolCha

%Computation of the expected value for each coordinate:
Expected=zeros(m,m,m);                      %Matrix in which we store the expected values
for it1=1:m
    for it2=it1:m
        for it3=it2:m
            Expected(it1,it2,it3)=FuncExpectedVal(PolCha(it1),PolCha(it2),PolCha(it3)); %Computation using the built function
        end
    end
end
%We just store the different values using the symmetry properties

%Then we use these same symmetry properties to fill the rest of the Expected matrix:
for it4=1:m
    for it5=1:m
        for it6=1:m
            List=sort([it4 it5 it6]);       %We sort in ascendent order
            Expected(it4,it5,it6)=Expected(List(1),List(2),List(3));
        end
    end
end
```

```
function [B]= FuncGetB(n,m)
%This function permits to get the coefficient matrix B

%Definition of the global parameters:
global U Kbar sigma Expected

%Computation for each coefficient:
B=zeros(n,m,n,m);
for it1=1:n   %it1 correspond to j
    for it2=1:m   %it2 correspond to l
        for it3=1:n   %it3 correspond to r
            for it4=1:m   %it4 correspond to i
                B(it1,it2,it3,it4)=sigma*U(:,it1)'*Kbar*U(:,it3)*Expected(2,it4,it2);
            end
        end
    end
end
```

```
function [C]= FuncGetC(n,m)
%This function permits to get the coefficient matrix C

%Definition of the global parameters:
global Lambda Expected

%Computation for each coefficient:
C=zeros(m,n);
for it1=1:m   %it1 correspond to i
    for it2=1:n   %it2 correspond to j
        C(it1,it2)=Lambda(it2)*Expected(1,it1,it1);
    end
end
```

```
function equationk = FuncEquationk(Gammak,k)
%System of equations with k fixed
%to find the alpha's and beta's values

%Definition of the global parameters:
global m n k A B C D E

equationk=zeros(1,m*n);     %Vector where we store the system of equation

%Building of the system of equation according to the found formulae:
%%First m equations (j=k):
for it1=1:m   %it1 correspond to i
    flag1=0;
    for it2=1:n   %it2 correspond to r
        if it2~=k   %case r~=k
            for it3=1:m   %it3 correspond to l
                equationk(1,it1)=equationk(1,it1)+B(k,it3,it2,it1)*Gammak(it3,it2-flag1);
            end
        else
            flag1=1;
        end
    end
    equationk(1,it1)=equationk(1,it1)+A(it1,k)+B(k,1,k,it1)-C(it1,k)*Gammak(it1,n);
end

%%m*(n-1) remaining equations (j~=k):
it0=m+1;
for it4=1:m   %it4 correspond to i
    flag2=0;
    for it5=1:n   %it5 correspond to j
        if it5~=k   %case j~=k
            flag3=0;
            for it6=1:n   %it6 correspond to r
                if it6~=k   %case r~=k
                    for it7=1:m   %it7 correspond to l
                        equationk(1,it0)=equationk(1,it0)+B(it5,it7,it6,it4)*Gammak(it7,it6-flag3);
                    end
                else
                    flag3=1;
                end
            end
            for it8=1:m   %it8 correspond to b
                for it9=1:m   %it9 correspond to l
                    equationk(1,it0)=equationk(1,it0)+D(k)*E(it8,it9,it4)*Gammak(it8,n)*Gammak(it9,it5-flag2);
                end
            end
            equationk(1,it0)=equationk(1,it0)+C(it4,it5)*Gammak(it4,it5-flag2)+B(it5,1,k,it4);
            it0=it0+1;
        else
            flag2=1;
        end
    end
end
```

```
function [sigma]= FuncStandardDev(Poly)
%This function permits to get the standard déviation of a polynomial pdf

%The following method works in any case but is not optimized:
mu=double(FuncExpectedVal(Poly,1,1));           %Expected value of the random function
epsilon=double(FuncExpectedVal(Poly,Poly,1));   %Expected value of the squared random function
sigma=sqrt(epsilon-mu^2);                       %Formula of the standard deviation
```

```
function [sigma]= FuncStandardDevVal(LambdaTild,lambda)
%This function permits to get the standard déviation of an eigenvalue

%Definition of the global parameters:
global m Expected

sigma=0;                                                    %We initialize sigma

for it1=2:m
    sigma=sigma+LambdaTild(it1)^2*Expected(1,it1,it1);      %We use the sum given in the appendix
end
sigma=sqrt(sigma)*abs(lambda);
```

```
function [LambdaMuMC,LambdaSigmaMC]= FuncEigenValMC(acMC,sigma)
%Function which gives the averaged value and the standard
%deviation of the eigenvalues of the random system

%Definition of the global parameters:
global n K Kbar

Rho=sigma*randn(1,acMC);                                %Mistakes values for each iteration

%Storage vectors for the Monte Carlo iterations:
LambdaMuMC=zeros(n,1);                                  %Storage of average eigenvalues
LambdaSigmaMC=zeros(n,1);                               %Storage of the standard derivation of the eigenvalues
EigVals=zeros(n,acMC);                                  %Storage of the random eigenvalues

%Monte Carlo iterations:
%%Computation of the averaged values:
for it1=1:acMC
    KTild=K+Rho(it1)*Kbar;                              %Building of the random stiffness matrix
    EigVals(:,it1)=eig(KTild);                          %Computation of the eigenvalues
    LambdaMuMC=LambdaMuMC+EigVals(:,it1);               %Addition of random eigenvalues
end
LambdaMuMC=LambdaMuMC./acMC;                            %Linear averaging

%%Computation of the standard deviations:
for it2=1:acMC
    LambdaSigmaMC=LambdaSigmaMC+(EigVals(:,it2)-LambdaMuMC).^2;
end
LambdaSigmaMC=sqrt(LambdaSigmaMC./acMC);                %Geometrical averaging
```

```
function [UTild]= FuncRandEigVec(n,m)
%This function permits to get the random functions for each random eigenvector

%Definition of the global parameters:
global U Bbar PolCha

UTild=sym(zeros(n));

%Sum for each random vector:
for it1=1:n   %it1 correspond to k
    flag1=0;
    for it2=1:n   %it2 correspond to r
        if it2==it1
            flag1=1;
        else
            for it3=1:m   %it3 correspond to l
                UTild(:,it1)=UTild(:,it1)+Bbar(it1,it3,it2-flag1)*PolCha(it3)*U(:,it2);
            end
        end
    end
    UTild(:,it1)=UTild(:,it1)+U(:,it1);
end
UTild=vpa(UTild);
```

```
function [LambdaMuTS,LambdaSigmaTS,USigmaTS]= FuncEigenValVecTS(K)
%This function gives the averaged value of eigenvalues computed from
%the Taylor Series of the eigenvalues and eigenvectors

%Definition of the global parameters:
global n Kbar sigma

%Vectors where we store the results:
LambdaMuTS=zeros(n,1);
LambdaSigmaTS=zeros(n,1);
USigmaTS=zeros(n);

%Building of the eigenvalues ad eigenvectors of the problem:
[Vec,Val]=eig(K);
Lambda=diag(Val);

%We look for the number of repeated eigenvalues:
if rem(n,2)==1
    l=(n-1)/2;      %Number of repeated eigenvalues (odd case)
    odd=1;
else
    l=n/2-1;        %Number of repeated eigenvalues (even case)
    odd=0;
end

%Computation of the mean and standard deviation of eigenvalues for each double pairs:
for it1=1:l
    %%Computation of first and second derivatives:
    X=Vec(:,2*it1:2*it1+1);
    G=sigma*X'*Kbar*X;
    [Gamma,dLambda]=eig(G);                         %First derivatives
    Z=X*Gamma;
    V=K-Lambda(2*it1)*eye(n);
    T=sigma*(X*X'-eye(n))*Kbar*Z;
    W=zeros(n,2);
    W(:,1)=ParticSol(V,T(:,1));
    W(:,2)=ParticSol(V,T(:,2));
    S=zeros(2);
    U=2*sigma*Z'*Kbar*W-2*Z'*W*dLambda;
    S(1,2)=U(1,2)/(2*(dLambda(2,2)-dLambda(1,1)));
    S(2,1)=U(2,1)/(2*(dLambda(1,1)-dLambda(2,2)));
    S(1,1)=-Z(:,1)'*W(:,1);
    S(2,2)=-Z(:,2)'*W(:,2);
    dZ=W+Z*S;
    d2Lambda=2*sigma*Z'*Kbar*dZ-2*Z'*dZ*dLambda;    %Second derivatives
    %%Computation of approximated expected values and standard deviations:
    LambdaMuTS(2*it1)=Lambda(2*it1)+d2Lambda(1,1);
    LambdaMuTS(2*it1+1)=Lambda(2*it1+1)+d2Lambda(2,2);
    LambdaSigmaTS(2*it1)=sqrt(dLambda(1,1)^2-1/4*d2Lambda(1,1)^2);
    LambdaSigmaTS(2*it1+1)=sqrt(dLambda(2,2)^2-1/4*d2Lambda(2,2)^2);
    USigmaTS(:,2*it1)=abs(dZ(:,1));
    USigmaTS(:,2*it1+1)=abs(dZ(:,2));
end

%Computation of the average of single eigenvalue(s):
%%For the first eigenvalue:
%%%Computation of the second derivative:
P1=Vec';
P1(1,:)=2*P1(1,:);
Q1=zeros(n,1);
for it2=2:n
    Q1(it2)=-1/(Lambda(it2)-Lambda(1))*sigma*(Vec(:,it2)'*Kbar*Vec(:,1));
end
dV1=inv(P1)*Q1;
d2Lambda1=sigma*dV1'*Kbar*Vec(:,1)+sigma*Vec(:,1)'*Kbar*dV1;        %Second derivative
%%%Computation of the expected value (eigenvalue) and standard deviation (eigenvector):
LambdaMuTS(1)=Lambda(1)+d2Lambda1;
USigmaTS(:,1)=abs(dV1);
```

```
%%For the last eigenvalue if n is even:
if odd==0
    %%%Computation of the second derivative:
    Pn=Vec';
    Pn(n,:)=2*Pn(n,:);
    Qn=zeros(n,1);
    for it3=1:n-1
        Qn(it3)=-1/(Lambda(it3)-Lambda(n))*sigma*(Vec(:,it3)'*Kbar*Vec(:,n));
    end
    dVn=inv(Pn)*Qn;
    d2Lambdan=sigma*dVn'*Kbar*Vec(:,n)+sigma*Vec(:,n)'*Kbar*dVn;    %Second derivative
    %%%Computation of the expected value (eigenvalue) and standard deviation (eigenvector):
    LambdaMuTS(n)=Lambda(n)+d2Lambdan;
    USigmaTS(:,n)=abs(dVn);
end
```

```
function [UMu,USigma]=FuncExpStanDevVec(UTild)
%This function permits to get the coefficient expected value and
%the standard deviation of the eigenvectors

%Definition of the global parameters:
global U n Bbar

%Definition of the storage matrices:
UMu=zeros(n);                                   %Matrix to store the averaged eigenvectors
USigma=zeros(n);                                %Matrix to store the standard deviations of eigenvectors

%Computation of the standard deviation and expected value vector by vector:
for it1=1:n
    %%Computation of the expected value
    for it2=1:it1-1
        UMu(:,it1)=UMu(:,it1)+Bbar(it1,1,it2)*U(:,it2);     %Computation of the expected values using the formula
    end                                                     %given in appendix
    for it3=it1+1:n
        UMu(:,it1)=UMu(:,it1)+Bbar(it1,1,it3-1)*U(:,it3);
    end
    UMu(:,it1)=UMu(:,it1)+U(:,it1);
    %%Computation of the standard deviation
    for it4=1:n
        USigma(it4,it1)=FuncStandardDev(UTild(it4,it1));    %Computation of standard deviation of each term
    end
end
```

```
function [UMuMC,USigmaMC]= FuncEigenVecMC(acMC,sigma)
%Function which gives the averaged value and the standard deviation
%of the eigenvectors of the random system

%Definition of the global parameters:
global n K Chi

Rho=sigma*randn(1,acMC);                        %Mistakes values for each iteration

%Storage vectors for the Monte Carlo iterations:
UMuMC=zeros(n);                                 %Storage of average eigenvalues
USigmaMC=zeros(n);                              %Storage of the standard derivation of the eigenvalues
EigVecs=zeros(n,n,acMC);                        %Storage of the random eigenvalues
```

```
%Monte Carlo iterations:
%%Computation of the averaged values:
for it1=1:acMC
    KTild=K+diag(Rho(it1)*Chi);                    %Building of the random stiffness matrix
    [EigVec,Useless]=eig(KTild);                   %Computation of the eigenvalues
    UMuMC=UMuMC+EigVec;                            %Addition of random eigenvalues
    EigVecs(:,:,it1)=EigVec;                       %We store the random eigenvectors
end
UMuMC=-UMuMC./acMC;                                 %Linear averaging

%%Computation of the standard deviations:
for it2=1:acMC
    USigmaMC=USigmaMC+(EigVecs(:,:,it2)-UMuMC).^2;
end
USigmaMC=sqrt(USigmaMC./acMC);                     %Geometrical averaging
```

```
function x = ParticSol(A, b)

% x = ParticSol(A, b) returns a particular solution to Ax=b.
% This particular solution has all free variables set to zero.
% An empty vector is returned if Ax=b is not solvable.

[m, n] = size(A);
[Rd, pivcol] = rref([A b]);
r = length(pivcol);

% If the last column of the augmented matrix [A b]
% is a pivot column, then Ax=b has no solution.

if max(pivcol) == n+1
  x = [];
else

% The values of the pivot variables are in the
% last column (which is called d) of Rd.
% The free variables are zero in this particular solution.

  x = zeros(n, 1);
  d = Rd(:, n+1);
  x(pivcol) = d(1:r);
end
```

```
function [in] = sym2strModif(sy)

%This version is a modified (simplified to the case of polynomials) version
%of the initial one called "sym2str" downloadable online.

%updated:  02-03-2009
%author: Marty Lawson
%
%converts symbolic variables to a matlab equation string insuring that
%only array opps are used.

    sy = sym(sy); %insure input is symbolic
    siz = numel(sy);   %find the number of elements in "sy"
    for i = 1:siz   %dump it into a cell array with the same number of elements
        in{i} = char(sy(i)); %convert to char
    end
    if siz == 1
        in = char(in);      %revert back to a 'char' array for single answers
    end
```