Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Uncertainty Quantification in Engineering

Mini-Project (Assignment #3)

Start date:	April 15, 2021
Hand-in date:	May 27, 2021

Introduction

In this assignment, you will develop your own Polynomial Chaos Expansion tool and apply it to a case study of your choice. The exercise is divided into several sections, each dealing with a different aspect of the problem.

You are required to program the problem solutions from scratch using Matlab – do not use any already pre-existing code for uncertainty quantification. A set of helper codes to handle the most technically challenging sections is provided as an aid.

Important remarks

- This is a **mini-project**, not the usual exercise-based homework. It is graded and you are expected to interact with your assigned supervisor. Contact him/her whenever you have questions.
- It is recommended that you form groups of three, if possible from similar backgrounds, so that you can develop and report the mini-project together. The final report shall clearly indicate the contribution of each student in the group.

Deliverables

- The project report (electronically as PDF), max. 10 pages
- The Matlab code you wrote for the project

1 Overview

The focus of this mini-project is to develop a polynomial chaos expansion tool for metamodelling a model of your choice. For the sake of simplicity, some technical options are restricted to "simple" methods, but you are free to extend your tool once the minimum requirements for the mini-project are met.

1.1 Ingredients needed in PCE

The following bullets represent the various bricks that need to be coded in order to complete a polynomial chaos expansion tool.

- A full model that can be surrogated
- A probabilistic input model
- A truncated orthonormal polynomial basis
- An experimental design
- The corresponding polynomial coefficients
- The evaluation of the computed expansion for new values of the input parameters

2 Detailed project description

Reminder: the truncated polynomial chaos expansion of a model $Y = \mathcal{M}(X)$ is given by:

$$Y = \sum_{\boldsymbol{\alpha} \in \mathcal{A}} c_{\boldsymbol{\alpha}} \, \Psi_{\boldsymbol{\alpha}}(\boldsymbol{X}) \tag{1}$$

where $c_{\boldsymbol{\alpha}}$ are real numbers and $\Psi_{\boldsymbol{\alpha}}(\boldsymbol{X})$ are multivariate polynomials orthonormal with respect to the joint PDF of $\boldsymbol{X}$.

2.1 Choice of a model

Each group is allowed to create their own computational model to surrogate, as long as the following conditions are respected:

- The number of input parameters is comprised in the interval $2 \leq M \leq 5$.
- The model must be continuous and smooth for all the plausible values of the inputs

- The model is a *scalar-valued* function (i.e., for each sampling of the M input variables, only one output is produced).
- The model is *vectorized*. In detail this requirement means that, given an $n \times M$ matrix of n realizations of an input vector `X` it will return an $n \times 1$ vector of model responses $y \in \mathbb{R}$, *i.e.* with the following syntax:
  `Y = myCustomModel(X);`

It is suggested (but not compulsory) that you begin your study with a simple polynomial model for easier validation of your elementary code bricks.

Suggested computational models: amongst the possible choices, the dyke problem of Assignment 2 and the simply supported beam in Lecture 5 can be used.

2.2 The probabilistic input model

The probabilistic input model must be chosen according to the model selected in the previous step, and must include only independent and uniformly distributed variables in appropriate intervals $X_i \sim \mathcal{U}([a_i,\ b_i])$. The range of each input variable shall be selected according to reasonable values. You may ask for help and validation of your probabilistic model.

2.3 The experimental design

The method that shall be used for the calculation of the polynomial coefficients is **least-square analysis**. The experimental design can therefore be created independently from the remaining PCE components. In order to create an experimental design $\mathcal{X}_{\text{ED}} = \{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(n)}\}$, the following steps need to be implemented:

- **Choice of a sampling strategy:** standard Monte Carlo sampling or Latin Hypercube Sampling (`lhsdesign` in Matlab). Matlab offers simple options to create samples in the unit hypercube, see Section A.1 for a reference.
- **Sampling of the input variables according to their distributions:** the samplers in the previous point produce samples from the unit hypercube: $\boldsymbol{u} \sim \mathcal{U}([0,1]^M)$. Devise the correct isoprobabilistic transform from the latter to the input vector $\boldsymbol{X}$ defined earlier.
- **Evaluation of the full model on the experimental design:** the full model must be evaluated on the generated experimental design and the resulting responses must be stored in matrix format (response vector $\boldsymbol{y}_{\text{ED}} = (y_1, y_2, \dots, y_n)^T$ with $y_i = \mathcal{M}(\boldsymbol{x}^{(i)})$).

It is recommended to use at least $n = 100$ samples as a starting point.

2.4 The polynomial basis

This section is most likely the most complex to develop, due to the necessity of coding tensor products effectively (book-keeping problem). It is divided into several subsections, each one dealing with one aspect of the polynomial basis generation. Before proceeding, the students must select a maximum polynomial degree p for their PCE calculations. It is recommended to start from lower degree polynomials ($1 \leq p \leq 4$) and gradually increase, to keep complexity manageable.

2.4.1 Evaluate the ED on univariate polynomials

Due to the choice of uniform input distributions, the univariate polynomials in the PCE are Legendre polynomials. In order to perform least-square analysis, the univariate Legendre polynomials up to degree p need to be evaluated on the entire experimental design $\mathcal{X}_{\text{ED}}$.

Classical Legendre polynomials, however, are orthonormal only with respect to random variables uniformly distributed between -1 and 1:

$$\xi \sim \mathcal{U}([-1, 1]) \tag{2}$$

The experimental design calculated in Section 2.3 must then be transformed with a suitable isoprobabilistic transform:

$$\boldsymbol{\xi}^{(i)} = \boldsymbol{\mathcal{T}}(\boldsymbol{x}^{(i)}) \tag{3}$$

before the Legendre polynomials can be correctly evaluated on it.

Note: The provided `eval_legendre` function (see Section A.2) can be used to evaluate the corresponding univariate polynomials.

2.4.2 Polynomial degree and basis truncation

Given a set of univariate polynomials $\psi_k(x_i)$, with $0 < k \leq p$ and $1 < i \leq M$, it is possible to create a multivariate polynomial basis by tensor product of the univariate ones according to:

$$\Psi_{\boldsymbol{\alpha}}(\boldsymbol{x}) = \prod_{i=1}^{M} \psi_{\alpha_i}(x_i) \tag{4}$$

where $\boldsymbol{\alpha} = \{\alpha_1, \ldots, \alpha_M\}$, with $|\boldsymbol{\alpha}| \overset{\text{def}}{=} \alpha_1 + \ldots + \alpha_M \leq p$.

We have generated the univariate polynomials $\psi_k(x_i)$ in the previous step, so what remains is to create all the possible multi-indices $\boldsymbol{\alpha}$ that satisfy the maximum polynomial degree requirement $|\boldsymbol{\alpha}| \leq p$. The cardinality of such set of indices will be denoted by P.

Efficiently performing this operation is computationally and algorithmically intensive, and not particularly instructive in this context, hence you should make use of the provided function `create_alphas(M,p)` (see also Section A.3) to generate the set of $\boldsymbol{\alpha} \in \mathcal{A}$ needed to create the regression matrix $\boldsymbol{\Psi}$. This matrix has entries $\boldsymbol{\Psi}_{ij} = \Psi_j\left(\boldsymbol{x}^{(i)}\right)$, where $1 < j \leq P$ is the index over the P basis elements generated by the `create_alphas(M,p)` function, while $1 < i \leq n$ is the index over the elements of the experimental design. The full matrix is given by

$$\boldsymbol{\Psi} = \begin{pmatrix} \Psi_1\left(\boldsymbol{x}^{(1)}\right) & \cdots & \Psi_P\left(\boldsymbol{x}^{(1)}\right) \\ \vdots & \ddots & \vdots \\ \Psi_1\left(\boldsymbol{x}^{(n)}\right) & \cdots & \Psi_P\left(\boldsymbol{x}^{(n)}\right) \end{pmatrix}. \tag{5}$$

2.5 Calculation of the polynomial coefficients with Ordinary Least-Squares (OLS)

Now that the full set of basis elements $\{\Psi_{\boldsymbol{\alpha}}\}_{\boldsymbol{\alpha}\in\mathcal{A}}$ has been evaluated on the experimental design $\mathcal{X}_{\text{ED}}$, which yielded the regression matrix $\boldsymbol{\Psi}$, the calculation of the coefficients via least-square minimization is reduced to an algebraic problem that is easily solved in MATLAB.

Indeed, the vector $\boldsymbol{c}$ containing the PCE coefficients can be simply obtained from the matrix $\boldsymbol{\Psi}$ and the response vector $\boldsymbol{y}_{\text{ED}}$ as:

$$\boldsymbol{c} = \left(\boldsymbol{\Psi}^{\mathsf{T}}\boldsymbol{\Psi}\right)^{-1}\boldsymbol{\Psi}^{\mathsf{T}}\boldsymbol{y}_{\text{ED}} \tag{6}$$

You can use matlab's built-in function `Ainv = pinv(A)` to perform the matrix inversion, see A.4 for reference.

2.6 Validation: evaluation of the metamodel on a new set of points

According to Eq. (1), it is possible to evaluate the polynomial response on a new input vector $\boldsymbol{x}$ with the coefficients obtained in the previous step or on a validation set $\mathcal{X}_V = \{\boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(n_V)}\}$.

However, Legendre polynomials are not orthonormal w.r.t. the vector $\boldsymbol{X}$, but w.r.t. the vector $\boldsymbol{\Xi} = \mathcal{T}(\boldsymbol{X})$. Hence, to evaluate the metamodel on the validation set $\mathcal{X}_V = \{\boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(n_V)}\}$, it is necessary to pay attention to operating with the correct variable space.

- Create a validation set $\mathcal{X}_V$ of suitable size and evaluate the full model on it. Compare the model responses with the responses of the metamodel for increasing polynomial degrees. What do you observe?

- Compute the relative leave-one-out error (see Lecture 8) *using the experimental design* $\mathcal{X}_{ED}$, and compare it with the relative mean-squared error computed from the validation set $\mathcal{X}_V$ as follows:

$$\epsilon^{\text{val}} = \frac{\sum_{i=1}^{n_V} \left(\mathcal{M}(\boldsymbol{x}^{(i)}) - \mathcal{M}^{PC}(\boldsymbol{x}^{(i)})\right)^2}{\sum_{i=1}^{n_V} \left(\mathcal{M}(\boldsymbol{x}^{(i)}) - \mu_{\mathcal{M}}\right)^2} \tag{7}$$

where $\mu_{\mathcal{M}} = \frac{1}{n_V} \sum_{i=1}^{n_V} \mathcal{M}(\boldsymbol{x}^{(i)})$ is the sample mean of the responses on the validation set.

2.7 Validation: postprocessing of the coefficients

A first check for the accuracy of the obtained coefficients can be obtained rapidly by evaluating mean and variance resulting from the vector of coefficients $\boldsymbol{c}_{\boldsymbol{\alpha}}$ and comparing it to the sample mean and variance of a large validation set drawn from the original model. For PCE, the response moments are derived in the slides of Lecture 8.

Prepare convergence plots for mean and variance as a function of:

- The polynomial degree p;
- The number of samples in the experimental design $\mathcal{X}_{\text{ED}}$.

A Appendix: Matlab function reference

A.1 Random samples: `rand` and `lhsdesign`

The basic function to generate random numbers in MATLAB is `rand`. It can be used to generate a single random number when called without arguments, or it can be used to generate an $n \times M$ matrix of random numbers uniformly distributed in $(0, 1)$ by typing:

```
A = rand(n,M);
```

However, in the statistics toolbox, MATLAB also offers the possibility to create a Latin Hypercube Sampling with the function `lhsdesign`. The syntax is similar to the previous one:

```
X = lhsdesign(n, M);
```

Note Both `rand` and `lhsdesign` are MATLAB functions, so you can see a more detailed description of their usage by typing `help rand` and `help lhsdesign` respectively.

A.2 Evaluate polynomials: `eval_legendre`

The function `eval_legendre(X, p)` evaluates the *normalized* univariate Legendre polynomial of order p on the coordinates given on a column vector X. For example, let us evaluate the normalized second order Legendre polynomial, which is given by (see lecture 6):

$$\tilde{P}_2(x) = \sqrt{5} \cdot \frac{(3x^2 - 1)}{2},$$

on the points $x_1 = -0.5$ and $x_2 = 0.3$. This can be done with the following code:

```
>> X = [-0.5; 0.3];
>> Y = eval_legendre(X, 2)

Y =
   -0.2795
   -0.8162
```

A.3 Generate a set of indices: `create_alphas`

The function `create_alphas(M, p)` returns a matrix with M columns, that contains in its rows all the possible combinations of $M-$dimensional vectors $\boldsymbol{\alpha} = (\alpha_1, ..., \alpha_M)$ such that $|\boldsymbol{\alpha}| = \alpha_1 + \cdots + \alpha_M \leq p$. Here is an example of its usage in dimension $M = 2$, for finding the indices that generate a polynomial basis with elements up to degree $p = 3$:

```
>> idx = create_alphas(2, 3)

idx =
 0     1
 1     0
 0     2
 2     0
 0     3
 3     0
 1     1
 1     2
 2     1
```

A.4 Inverting a matrix

Matlab offers several options for inverting a matrix:

- `inv` for square, invertible, non-singular matrices
- `pinv` for the computation of the Moore-Penrose pseudoinverse (equal to `inv` if the input matrix is invertible)

If your goal is to solve a system of linear equations, Matlab also offers several options: you can use `inv/pinv`, `lsqminnorm`, or `mldivide` (backslash operator). Read the Matlab documentation to find out which function is suited best for your purpose.