Technische Universität München
Lehrstuhl für Informatik V
Dr. Tobias Neckel
M. Sc. Ionuț Farcaș

SS 2017

May 31, 2017

Algorithms for Uncertainty Quantification

Tutorial 6: Polynomial chaos approximation 1: the pseudo-spectral approach

In this worksheet, we focus on the generalized polynomial chaos approximation and the pseudo-spectral approach.

Orthogonal polynomials

In the lecture, we saw that the generalized polynomial chaos (gPC) approximation is defined in terms of orthogonal polynomials. Given a weight function ρ, two polynomials $\phi_i(x)$ and $\phi_j(x)$ are called orthogonal with respect to ρ if

$$< \phi_i(x), \phi_j(x) >_\rho = \int \phi_i(x)\, \phi_j(x)\, \rho(x)\, dx = \gamma_i\, \delta_{ij},$$

where $\gamma_i =< \phi_i(x), \phi_i(x) >_\rho \in \mathbb{R}$ and δ_{ij} is Kronecker's delta function.
Two polynomials $\phi_i(x)$ and $\phi_j(x)$ are called orthonormal with respect to ρ if

$$< \phi_i(x), \phi_j(x) >_\rho = \int \phi_i(x)\, \phi_j(x)\, \rho(x)\, dx = \delta_{ij}.$$

Note that orthogonal polynomials can be easily transformed into orthonormal polynomials via

$$\phi_i(x) := \frac{\phi_i(x)}{\sqrt{\gamma_i}}.$$

A scheme to compute orthogonal polynomials is Stieltjes' three-terms recursion relation

$$\begin{aligned}\phi_{-1}(x) &\equiv 0\\ \phi_0(x) &\equiv 1\\ \phi_{n+1}(x) &= (A_n x + B_n)\,\phi_n(x) \;-\; C_n\,\phi_{n-1}(x) \qquad n \geq 0,\end{aligned}$$

where A_n, B_n, C_n are some constants.
Fortunately, we do not need to implement this scheme from scratch. In `chaospy`, it is easy to generate orthogonal polynomials based on a given probability distribution ρ. In the following code snippet, we assume that the underlying distribution is uniform. To generate orthogonal polynomials up to degree N, the `python` + `chaospy` syntax is

```
import chaospy as cp

# create a uniform distribution object
distr = cp.Uniform()

# generate K orthonormal polynomials based on distr
orth_poly = cp.orth_ttr(N, distr, normed=True)
# when you want unnormalized polynomials, i.e. normed=False, to
    compute the normalization constants, the syntax is
orth_poly, norm_consts = cp.orth_ttr(N, distr, retall=True)

# to compute the inner product, i.e. the expectation of the
   product of two polynomials of degrees i and j, use
expect  = cp.E(orth_poly[i]*orth_poly[j], distr)
```

Assignment 1

Let $\rho_1 = \mathcal{U}(-1, 1)$ and $\rho_2 = \mathcal{N}(10, 1)$. Moreover, let $N = [2, 5, 8]$. Write a `python` program to generate orthonormal polynomials of degree up to N with respect to ρ_1 and ρ_2. Check their orthonormality by computing $< \phi_i(x), \phi_i(x) >_\rho$, $\quad i = 0, \ldots, N$, i.e. $\mathbb{E}[\phi_i^2(x)]$. What do you observe?

Probabilistic collocation: the pseudo-spectral approach

In the methodology that follows, the assumption is that the weight function ρ is a probability density function. In the lecture, we saw that, given a function $f(x, \omega)$, where x denotes the deterministic inputs and ω denotes the stochastic inputs, the degree N gPC approximation of f reads

$$f(x, \omega) \approx \sum_{i=0}^{N-1} \hat{f}_i(x)\ \phi_i(\omega), \tag{1}$$

where $\phi_i(\omega), \quad i = 0, \ldots, N-1$ are orthogonal polynomials with respect to the probability distribution ρ of ω and $\hat{f}_i(x)$ are the expansion's coefficients. In this tutorial, we asses $\hat{f}_i(x)$ via

$$\int_{\text{supp}(\rho)} f(x)\, \phi_i(x)\, \rho(x)\, dx \approx \sum_{k=0}^{K-1} f(x_k)\, \phi_i(x_k) w_k, \tag{2}$$

where $\{x_k, w_k\}_{k=0}^{K-1}$ denotes the set of nodes and weights associated to ρ.
For simplicity, we assume that the orthogonal polynomials are orthonormal. After the obtain the gPC expansion coefficients, statistics such as expectation and variance can be easily computed via

$$\mathbb{E}[f(x, \omega)] = \hat{f}_0(x), \quad \text{Var}[f(x, \omega)] = \sum_{i=1}^{N-1} \hat{f}_i(x). \tag{3}$$

It is straightforward to implement the gPC + pseudospectral approach in `chaospy`. The following code snippet assumes that the input probability distribution is uniform.

```
import chaospy as cp

# create a uniform distribution object
distr = cp.Uniform()

# generate K Gaussian nodes and weights based on distr
nodes, weights = cp.generate_quadrature(K, distr, rule='G')

# generate K orthonormal polynomials based on distr
orth_poly = cp.orth_ttr(N, distr, normed=True)
# when you want unnormalized polynomials, i.e. normed=False, to
    compute the normalization constants, the syntax is
```

```
orth_poly, norm_consts = cp.orth_ttr(N, distr, retall=True)

# assume that you evaluated your underlying model M at all
    quadrature nodes (see Eq. (2) and stored the values in a
    vector M_eval

# find the gPC approximation of your model M
gPC_M = cp.fit_quadrature(orth_poly, nodes, weights, M_eval)
# to return the gPC expantion coefficients, too
gPC_M, gPC_coeff = cp.fit_quadrature(orth_poly, nodes, weights,
     M_eval, retall=True)

# finally, compute the expectation and the variance of your
    model M
expect  = cp.E(gPC_M, distr)
var     = cp.Var(gpc_M, distr)
```

Assignment 2

Consider the model problem, the linear damped oscillator

$$\begin{cases} \frac{d^2y}{dt^2}(t) + c\frac{dy}{dt}(t) + ky(t) = f\cos(\omega t) \\ y(0) = y_0 \\ \frac{dy}{dt}(0) = y_1. \end{cases} \tag{4}$$

Considering $t \in [0, 20]$, $\Delta t = 0.01$, $c = 0.5$, $k = 2.0$, $f = 0.5$, $y_0 = 0.5$, $y_1 = 0.0$, use the `odeint`[1] function from `scipy.integrate` to discretize the model from Eq. (4). For the upcoming experiments, the output that we are interested in is $y(10)$. Assume that $\omega \sim \mathcal{U}(0.95, 1.05)$. Write a `python` program to propagate the uncertainty in ω through the model in Eq. (4) using the gPC expansion method from Eq. (1). Asses the expansion coefficients from Eq. (2) using the pseudospectral approach. Consider $K = [2, 4, 6]$ and $N = [1, 2, 3]$; the correspondence is one-to-one. Initially, write your own implementation of the pseudospectral approach. Afterwards, use `chaospy`'s `fit_quadrature` function. Based on these coefficients, compute the expectation and the variance of $y(10)$. Compare the results with the ones obtained via Monte Carlo sampling. What do you observe?

[1] see `https://docs.scipy.org/doc/scipy-0.17.0/reference/generated/scipy.integrate.odeint.html`