PoCET: A Polynomial Chaos Expansion Toolbox for MATLAB

Felix Petzke

Ali Mesbah

Stefan Streif
Motivation / Problem Setup

Uncertainty propagation
• present in every (real) system at some point ➞ quantification & propagation, e.g. for robust/ stochastic control

\[
\dot{x} = f(x; a), \ x(0) = x_0, \ a \sim \mathcal{N}(\mu, \sigma)
\]

system of equations, initial distribution

sampling & simulations

information about final distribution (histogram, moments, PDF, …)

sampling & simulations

expanded system, initial PCE coefficients

\[
\hat{x} = \hat{f}(\hat{x}; \hat{a}), \ \hat{x}(0) = \hat{x}_0
\]

offline

online

moment recovery

final PCE coefficients \(\hat{x}(t_f)\)

Galerkin-projection PCE: Advantages
• shift (large) part of computational complexity offline ➞ very fast online simulations compared to sampling
• particularly useful for polynomial systems due to orthogonality

Drawbacks
• curse of dimensionality!
• series expansion ➞ might require high order to yield good approximation

PoCET: a Polynomial Chaos Expansion Toolbox for MATLAB

projection PCE provides means of quickly propagating stochastic uncertainties through ODE systems
Projection PCE: Brief Introduction

**Main idea**

- series expansion transforming probabilistic variables into deterministic ones

\[ a \sim \mathcal{N}(\mu, \sigma) \Rightarrow a \approx \sum_{i=0}^{\tilde{p}-1} \hat{a}_i \varphi_i(\xi) = \hat{a}^T \Phi(\xi), \quad \tilde{p} = \frac{(N_\xi + P)!}{N_\xi! P!}, \quad \hat{a}_i = \frac{\langle a(\xi), \varphi_i \rangle}{\langle \varphi_i, \varphi_i \rangle} \]

with \( a \) - original variable, \( \hat{a}_i \) - expansion coefficients, \( \Phi = [\varphi_1, \ldots, \varphi_\tilde{p}] \) - polynomial basis, \( \xi = \{\xi_1, \ldots, \xi_{N_\xi}\} \) - standard random variables, \( N_\xi \) - # of random variables, \( P \) - order of expansion

<table>
<thead>
<tr>
<th>( \tilde{p}(P, N_\xi) )</th>
<th>( N_\xi = 2 )</th>
<th>4</th>
<th>6</th>
<th>8</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P = 3 )</td>
<td>10</td>
<td>35</td>
<td>84</td>
<td>165</td>
<td>286</td>
</tr>
<tr>
<td>4</td>
<td>15</td>
<td>70</td>
<td>210</td>
<td>495</td>
<td>1001</td>
</tr>
<tr>
<td>5</td>
<td>21</td>
<td>126</td>
<td>462</td>
<td>1287</td>
<td>3003</td>
</tr>
<tr>
<td>6</td>
<td>28</td>
<td>210</td>
<td>924</td>
<td>3003</td>
<td>8008</td>
</tr>
</tbody>
</table>

- generally using orthogonal polynomials, i.e. \( \langle \varphi_i, \varphi_j \rangle = \int_\Omega \varphi_i(\xi)\varphi_j(\xi)\rho(\xi)d\xi = \lambda_i \delta_{ij} \)

with \( \lambda_i \in \mathbb{R} \) - coefficient value, \( \delta_{ij} := \{1 \text{ if } i = j, \text{ 0 else} \} \)

- increased applicability due to increases in computational power

---

1 Sudret (2014). Risk and Reliability in Geotechnical Engineering, chapter Polynomial chaos expansions and stochastic finite element methods. CRC Press.
Projection PCE: Application to ODEs

Example

- analogous procedure to static equations, but often more complicated outcome due to time dependency
- initial system: \[ \dot{x}(t) = \alpha(\xi)x^2(t), \quad \alpha(\xi) \sim \mathcal{N}(\mu, \sigma) \]
- PCE:
  \[ \sum_{n=0}^{\tilde{p}-1} \dot{x}_n \phi_n = \sum_{j=0}^{\tilde{p}-1} \hat{a}_j \phi_j \sum_{k=0}^{\tilde{p}-1} \dot{x}_k \phi_k \sum_{l=1}^{\tilde{p}-1} \dot{x}_l \phi_l \]
- projection:
  \[ \sum_{n=0}^{\tilde{p}-1} \dot{x}_n \langle \phi_n, \varphi_i \rangle = \sum_{j,k,l=0}^{\tilde{p}-1} \hat{a}_j \dot{x}_k \dot{x}_l \langle \phi_j \phi_k \phi_l, \varphi_i \rangle \]
- orthogonality:
  \[ \dot{x}_i = \frac{1}{\langle \varphi_i, \varphi_i \rangle} \sum_{j,k,l=0}^{\tilde{p}-1} \hat{a}_j \dot{x}_k \dot{x}_l \langle \phi_j \phi_k \phi_l, \varphi_i \rangle \Rightarrow \text{requires polynomial system} \]
- expanded system:
  \[ \dot{\hat{x}} = \sum_{j=0}^{\tilde{p}-1} \hat{a}_j E_j (\hat{x} \otimes \hat{x}) \Rightarrow \text{often } \alpha_j = 0 \text{ for } j \geq 2 \]

with \( E_j = \begin{bmatrix} e_{j000} & e_{j010} & \cdots & e_{j050} & \cdots & e_{j0s0} \\
e_{j001} & e_{j011} & \cdots & e_{j051} & \cdots & e_{j0s1} \\
\vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\
e_{j00s} & e_{j01s} & \cdots & e_{j0ss} & \cdots & e_{jss} \end{bmatrix} \in \mathbb{R}^{\tilde{p} \times \tilde{p}^2}, \quad e_{jkl} := \frac{\langle \varphi_j \varphi_k \varphi_l, \varphi_i \rangle}{\langle \varphi_i, \varphi_i \rangle} \]


\[ \Rightarrow \text{setting up a PCE can be challenging, especially for ODE systems} \]
Why PoCET?

Main contributions
- automatic generation of projection PCEs for dynamic systems in Matlab, including
  - choosing sufficiently large polynomial basis for several standard distributions
  - computation of coefficient matrices for simulation / moment recovery
  - writing .m-files for the extended system
  - routines for simulation, moment recovery, PDF fitting, sampling, ...
- particularly useful for polynomial dynamic systems

Features & comparison to UQLab [5]

PoCET
- UQ in nonlinear optimization problems (e.g. [6])
- strong focus on Galerkin projection PCE
  - “exact” for polynomial systems (cf. [7])
  - inherent curse of dimensionality

UQLab
- UQ in reliability analysis & surrogate modeling
- vast number of sparse regression methods
  - approximative in nature
  - circumvents curse of dimensionality


→ showcase capabilities via demonstration
**Problem setup (cf. [8])**

- two model candidates for a chemical reaction: **Henri kinetics** or **Michaelis-Menten kinetics**
  \[
  \begin{align*}
  \dot{x}_1^H &= \left(p_1^H + p_3^H\right) \left(x_2^H - 1\right) x_1^H + \left(p_2^H + u\right) x_2^H, \\
  \dot{x}_2^H &= p_1^H \left(1 - x_2^H\right) x_1^H - \left(p_2^H + u\right) x_2^H,
  \end{align*}
  \]
  \[
  \begin{align*}
  \dot{x}_1^M &= p_1^M \left(x_2^M - 1\right) x_1^M + \left(p_2^M + u\right) x_2^M, \\
  \dot{x}_2^M &= p_1^M \left(1 - x_2^M\right) x_1^M - \left(p_3^M + p_2^M + u\right) x_2^M,
  \end{align*}
  \]
  
  with \(x_1^*\) - substrate concentration, \(x_2^*\) - complex concentration, \(p_i^*\) - reaction rates, \(u\) - input

- uncertain initial conditions \(x_1^*(0) \sim \mathcal{B}_4(3, 3, 0.96, 0.98), \ x_2^*(0) \sim \mathcal{B}_4(3, 3, 0.01, 0.03)\)

- uncertain reaction rates \(p_1^H \sim \mathcal{U}(0.9, 1.1)\) and \(p_1^M \sim \mathcal{U}(0.9, 1.15)\)

- total of 5 independent uncertainties per system

- outputs \(y^* = x_2^*\) after 10s virtually **indistinguishable** due to large overlap of possible results

- goal: **find optimal input** \(u\) such that PDFs don’t overlap
  
  ![Hellinger distance](image)

  \[1 - \int_{\Omega} \sqrt{\mu_H(\xi)\mu_M(\xi)}d\xi\]


**Demo: Optimal Experimental Design**

solve task at hand with PoCET
Demo: Setup in PoCET

System definition: states and parameters

\[ \dot{x}_1^H = (p_1^H + p_3^H)(x_2^H - 1)x_1^H + (p_2^H + u)x_2^H, \quad x_1^H(0) \sim \mathcal{B}_4(3, 3, 0.96, 0.98) \]

\[ \dot{x}_2^H = p_1^H(1 - x_2^H)x_1^H - (p_2^H + u)x_2^H, \quad x_2^H(0) \sim \mathcal{B}_4(3, 3, 0.01, 0.03) \]

\[ p_i^H \sim U(0.9, 1.1) \]

for i = 1:3
parametersH(i).name = ['p_' num2str(i)]; % name
parametersH(i).dist = 'uniform'; % distribution
parametersH(i).data = [0.9, 1.1]; % distribution parameters
end

analogous for second system
Demo: Setup in PoCET

System definition: inputs

- here: assume piecewise constant input $u(t; u_V, u_t)$

$$u(t) = \begin{cases} 0 : t < u_t(1) \\ u_V(i) : u_t(i) \leq t < u_t(i+1), i = 1, \ldots, N_t - 1 \\ u_V(N_t) : t \geq u_t(N_t) \end{cases}$$

```matlab
61 - inputs(1).name = 'u'; % name
62 - inputs(1).rhs = 'piecewise(u_t,u_v,t)'; % right hand side
63 - inputs(1).u_t = [0 0.5 1 1.5 2 2.5 3 3.5]; % step times
64 - inputs(1).u_v = [0 0 0 0 0 0 0]; % initial step sizes
```

Expansion

- system defined in terms of structures ‘statesH’, ‘parametersH’, ‘inputs’

⇒ carry out the actual PCE of specified order

```matlab
67 - pce_order = 4;
68 - pcesysH = PoCETcompose(statesH,parametersH,inputs,[],pce_order); % calculate system expansion
```

- ‘PoCETcompose’ checks input for consistency, parses equations, chooses sufficiently large polynomial basis, and calculates coefficient matrices ⇒ all stored in output structure ‘pcesysH’

⇒ last steps before simulations: calculate matrices for moment calculation & write ODE function files
Demo: Function Files

Last steps

• calculate matrices for moment recovery from PCE coefficients: here up to 4\textsuperscript{th} order

```matlab
> pcesysH.MomMats = PoCETmomentMatrices(pcesysH,4); % calculate matrices for moment calculation
```

• write ODE function files in current directory

```matlab
> PoCETwriteFiles(pcesysH,'ex4_ODE_H'); % write .m-function files for simulation
```

• resulting .m-file:

```matlab
function dXdt = ex4_ODE_H(t,X,PoCETsys,u_t,u_v)
    M = PoCETsys.coeff_matrices; % coefficient matrices from system struct
    x_1 = X(0*PoCETsys.pce.options.n_phi+1:1*PoCETsys.pce.options.n_phi);
    x_2 = X(1*PoCETsys.pce.options.n_phi+1:2*PoCETsys.pce.options.n_phi); % initial conditions
    u1 = piecewise(u_t,u_v,t); % specified input function
    x_1x_2 = mykron(x_1,x_2); % precompute Kronecker product(s) for faster computations
    ddt_x_1 = M.p_1_02*x_1*x_2 - M.p_1_01*x_1 + M.p_2_01*x_2 + M.p_3_02*x_1*x_2 - M.p_3_01*x_1 + u1*M.one_01*x_2;
    ddt_x_2 = -M.p_1_02*x_1*x_2 + M.p_1_01*x_1 - M.p_2_01*x_2 - u1*M.one_01*x_2; % expanded ODEs
    dXdt = [ddt_x_1; ddt_x_2]; % output vector
end
```

ready for simulation
Demo: Optimize

Optimization problem
• here: using fmincon ➔ define cost function, initial conditions, and constraints

```matlab
77 - u0 = [1 .5 0 0 0 0 0 0]; % initial values
78 - cost = @(u) u*eye(8)*u'; % cost function
79 - u_min = zeros(8); % lower bound
80 - u_max = 5*ones(8); % upper bound
81 - constr = @(u)ex4_mycon(u,pcesysH,sys_M,simoptions); % nonlinear constraint
82 - u_opt = fmincon(cost,u_0,[],[],[],[],u_min,u_max,constr,ops); % optimize!
```

• nonlinear constraint: simulate systems ➔ calculate stochastic moments ➔ fit PDFs

```matlab
1 - function [c,ceq] = ex4_mycon(u,pcesysH,pcesysM,simoptions)
2 - simH = PoCETsimGalerkin(pcesysH,'ex4_ODE_H',[],simoptions,'u_v',u); % simulate system H
3 - momH = PoCETcalcMoments(pcesysH,pcesysH.MomMats,simH.x_2.pcvals(:,end)); % calc. moments
4 - betaH = calcBeta4(momH); % fit 4-parameter beta distribution
5 -
6 - simM = PoCETsimGalerkin(pcesysM,'ex4_ODE_M',[],simoptions,'u_v',u); % simulate system M
7 - momM = PoCETcalcMoments(pcesysM,pcesysM.MomMats,simM.x_2.pcvals(:,end)); % calc. moments
8 - betaM = calcBeta4(momM); % fit 4-parameter beta distribution
9 -
10 - c = calcMDCbeta(betaH,betaM); % use Hellinger distance as measure of PDF overlap
11 - ceq = []; % no equality constraints
```

➔ nonlinear constraints evaluated in every optimization step
Demo: Results & Computation Times

Simulation results

• no overlap between resulting PDFs ➞ single measurement sufficient to decide which model is correct

Computation times

• setting up PCE for both systems: 7.6s (parsing, calculation of coefficient & moment matrices)
• ‘online’ optimization using fmincon: 11.5s (185 constraint evaluations ➞ 370 simulations, PDF fits, …)
 ➞ less than 0.03s per simulation + moment calculation + PDF fitting
• 40% of computational load ‘offline’ compared to overall load ➞ >99% compared to one simulation

 ➞ very fast online computations allow for vast range of applications (MPC, FDI, …)
Concluding Remarks

Perks of PoCET
- **straight-forward** definition of dynamic systems
- automatic generation of **projection-based PCEs for dynamic systems**, in particular for polynomial systems
- **modular design** allows for easy use in conjunction with other tools (e.g. UQLab)
- updating exiting PoCET-systems (i.e. changing parameters of uncertainties) very **easy and fast**
- several simulation routines provided (Galerkin or collocation PCE, Monte-Carlo)
  - usable with any existing Matlab ODE solver
- **stand-alone**, apart from employing Symbolic Math toolbox for parsing

Outlook
- open source: modify it to your own needs!
- inclusion of **arbitrary PCE** → very straight-forward if quadrature rules for integration provided (cf. [9])
- more examples: stochastic MPC, static equations, ...
  → we’re curious to see what you’ll use it for!


Visit [www.tu-chemnitz.de/etit/control/research/PoCET/](http://www.tu-chemnitz.de/etit/control/research/PoCET/) for more details!