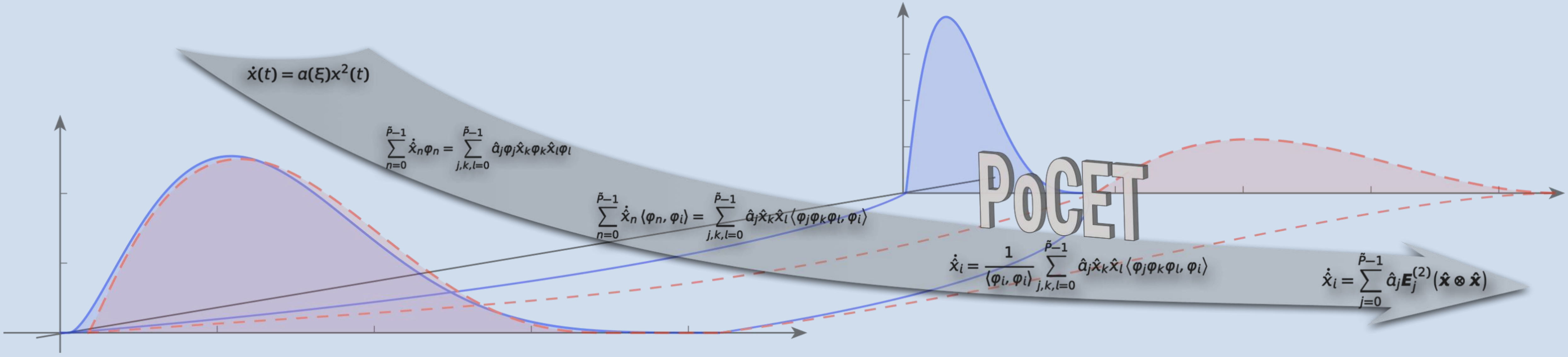
Chemnitz University of Technology
Faculty of Electrical Engineering and Information Technology
Automatic Control and System Dynamics Lab [1)]

University of California, Berkeley
Department of Chemical and Biomolecular Engineering
Mesbah Lab [2)]


# PoCET: A Polynomial Chaos Expansion Toolbox for MATLAB

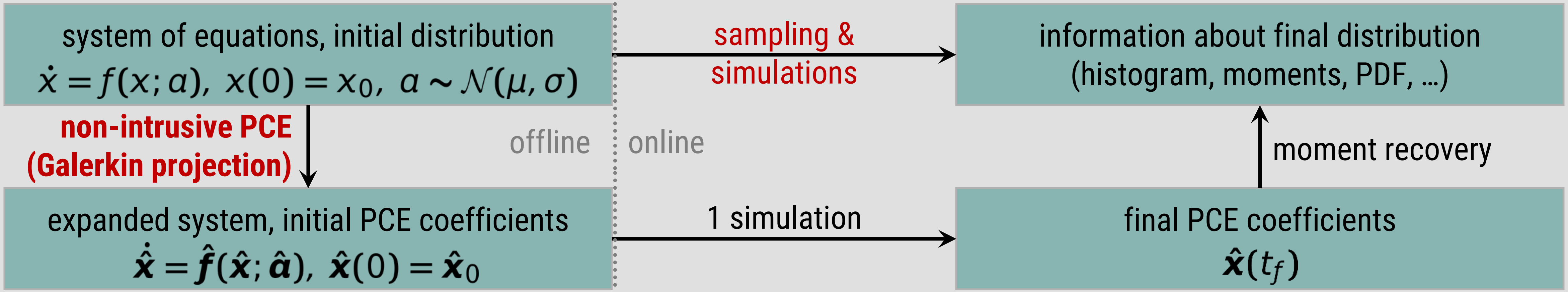
IFAC WC 2020


**Felix Petzke** [1)]
Ali Mesbah [2)]
Stefan Streif [1)]

# Motivation / Problem Setup

## Uncertainty propagation

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




## 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

➡ 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

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

with $a$ - original variable, $\hat{a}_i$ - expansion coefficients, $\boldsymbol{\Phi} = [\varphi_1, \ldots, \varphi_{\tilde{P}}]$ - polynomial basis,
$\hat{\boldsymbol{\xi}} = \{\hat{\xi}_1, \ldots, \hat{\xi}_{N_\xi}\}$ - standard random variables, $N_\xi$ - # of random variables, $P$ - order of expansion

| $\tilde{P}(P, N_\xi)$ | $N_\xi$ = 2 | 4 | 6 | 8 | 10 |
|---|---|---|---|---|---|
| $P$ = 3 | 10 | 35 | 84 | 165 | 286 |
| 4 | 15 | 70 | 210 | 495 | 1001 |
| 5 | 21 | 126 | 462 | 1287 | 3003 |
| 6 | 28 | 210 | 924 | 3003 | 8008 |

➡ curse of dimensionality

- 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, $\delta_{ij} := \{1 \text{ if } i = j, \; 0 \text{ else}\}$


*[1] Sudret (2014). Risk and Reliability in Geotechnical Engineering, chapter Polynomial chaos expansions and stochastic finite element methods. CRC Press.*
*[2] Eldred et al. (2008). Evaluation of non-intrusive approaches for Wiener-Askey generalized polynomial chaos. In 49th AIAA SSDM.*
*[3] Eldred, Burkardt (2009). Comparison of non-intrusive polynomial chaos and stochastic collocation methods for uncertainty quantification. In 47th AIAA ASM.*


➡ increased applicability due to increases in computational power

**Example**

- analogous procedure to static equations, but often more complicated outcome due to time dependency
- initial system: $\dot{x}(t) = a(\xi)x^2(t),\ a(\xi) \sim \mathcal{N}(\mu, \sigma)$
- PCE: $\sum_{n=0}^{\tilde{P}-1} \dot{\hat{x}}_n \varphi_n = \sum_{j=0}^{\tilde{P}-1} \hat{a}_j \varphi_j \sum_{k=0}^{\tilde{P}-1} \hat{x}_k \varphi_k \sum_{l=0}^{\tilde{P}-1} \hat{x}_l \varphi_l$
- projection: $\sum_{n=0}^{\tilde{P}-1} \dot{\hat{x}}_n \langle \varphi_n, \varphi_i \rangle = \sum_{j,k,l=0}^{\tilde{P}-1} \hat{a}_j \hat{x}_k \hat{x}_l \langle \varphi_j \varphi_k \varphi_l, \varphi_i \rangle$
- orthogonality: $\dot{\hat{x}}_i = \frac{1}{\langle \varphi_i, \varphi_i \rangle} \sum_{j,k,l=0}^{\tilde{P}-1} \hat{a}_j \hat{x}_k \hat{x}_l \langle \varphi_j \varphi_k \varphi_l, \varphi_i \rangle$ ➡ requires polynomial system
- expanded system: $\dot{\hat{\boldsymbol{x}}} = \sum_{j=0}^{\tilde{P}-1} \hat{a}_j \boldsymbol{E}_j (\hat{\boldsymbol{x}} \otimes \hat{\boldsymbol{x}})$ ➡ often $\alpha_j = 0$ for $j \geq 2$

$\langle \varphi_i, \varphi_j \rangle = \int_\Omega \varphi_i \varphi_j \rho d\xi$

with $\boldsymbol{E}_j = \begin{bmatrix} e_{j000} & e_{j010} & \cdots & e_{j0S0} & \cdots & e_{jSS0} \\ e_{j001} & e_{j011} & \cdots & e_{j0S1} & \cdots & e_{jSS1} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ e_{j00S} & e_{j01S} & \cdots & e_{j0SS} & \cdots & e_{jSSS} \end{bmatrix} \in \mathbb{R}^{\tilde{P} \times \tilde{P}^2},\ e_{jkli} := \frac{\langle \varphi_j \varphi_k \varphi_l, \varphi_i \rangle}{\langle \varphi_i, \varphi_i \rangle}$

*[4] Kim, Shen, Nagy, Braatz (2013). Wiener's polynomial chaos for the analysis and control of nonlinear dynamical systems with probabilistic uncertainties. IEEE CSM.*

➡ 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

*[5] Marelli, Sudret (2014). UQLab: A framework for uncertainty quantification in Matlab. In 2nd ICVRAM.*
*[6] Mesbah, Braatz (2014). Active Fault Diagnosis for nonlinear systems with probabilistic uncertainties. In 19th IFAC WC.*
*[7] Mühlpfordt, Findeisen, Hagenmeyer, Faulwasser (2017). Comments on quantifying truncation errors for polynomial chaos expansions. IEEE Control System Letters.*

➡ showcase capabilities via demonstration

**Problem setup (cf. [8])**

- two model candidates for a chemical reaction: **H**enri kinetics or **M**ichaelis-Menten kinetics

$$\dot{x}_1^H = \left(p_1^H + p_3^H\right)\left(x_2^H - 1\right)x_1^H + \left(p_2^H + u\right)x_2^H \qquad \dot{x}_1^M = p_1^M\left(x_2^M - 1\right)x_1^M + \left(p_2^M + u\right)x_2^M$$

$$\dot{x}_2^H = p_1^H\left(1 - x_2^H\right)x_1^H - \left(p_2^H + u\right)x_2^H, \qquad \dot{x}_2^M = p_1^M\left(1 - x_2^M\right)x_1^M - \left(p_3^M + p_2^M + u\right)x_2^M,$$

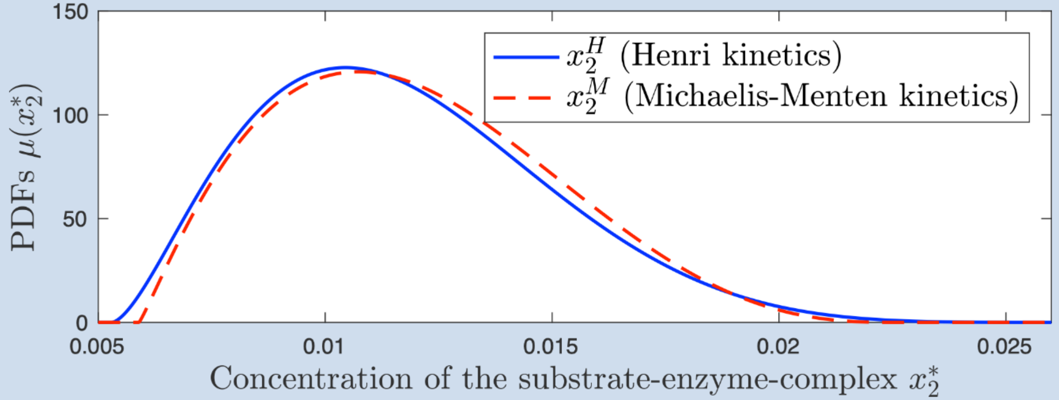
with $x_1^*$ - substrate concentration, $x_2^*$ - complex concentration, $p_i^*$ - reaction rates, $u$ - input

- 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 $p_i^H \sim \mathcal{U}(0.9, 1.1)$ and $p_i^M \sim \mathcal{U}(0.9, 1.15)$

➡ 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 $1 - \int_\Omega \sqrt{\mu_H(\xi)\mu_M(\xi)}d\xi$




*[8] Streif, Petzke, Mesbah, Findeisen, Braatz (2014). Optimal experimental design for probabilistic model discrimination using polynomial chaos. In 19th IFAC WC.*

➡ solve task at hand with PoCET

# Demo: Setup in PoCET

## System definition: states and parameters

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

```
statesH(1).name = 'x_1'; % name
statesH(1).rhs = '(p_1+p_3)*(x_2-1)*x_1+(p_2+u)*x_2'; % right hand side of ODE
statesH(1).dist = 'beta4'; % initial distribution
statesH(1).data = [3 3 0.96 0.98]; % initial distribution parameters
```

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

```
statesH(2).name = 'x_2'; % name
statesH(2).rhs = 'p_1*(1-x_2)*x_1-(p_2+u)*x_2'; % right hand side of ODE
statesH(2).dist = 'beta4'; % initial distribution
statesH(2).data = [3 3 0.01 0.03]; % initial distribution parameters
```

$$p_i^H \sim \mathcal{U}(0.9, 1.1)$$

```
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
```

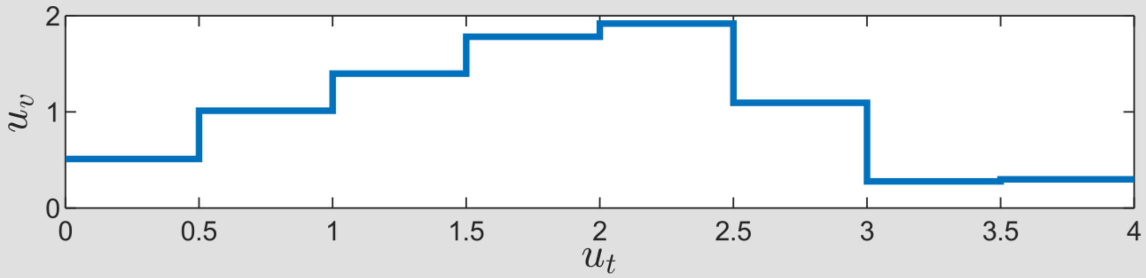
➡ analogous for second system

# Demo: Setup in PoCET

## System definition: inputs

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

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




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

```
pce_order = 4;
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$^{th}$ order

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

- write ODE function files in current directory

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

- resulting .m-file:

```
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_1x_2 - M.p_1_01*x_1 + M.p_2_01*x_2 + M.p_3_02*x_1x_2 - M.p_3_01*x_1 + u1*M.one_01*x_2;
 ddt_x_2 = - M.p_1_02*x_1x_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

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

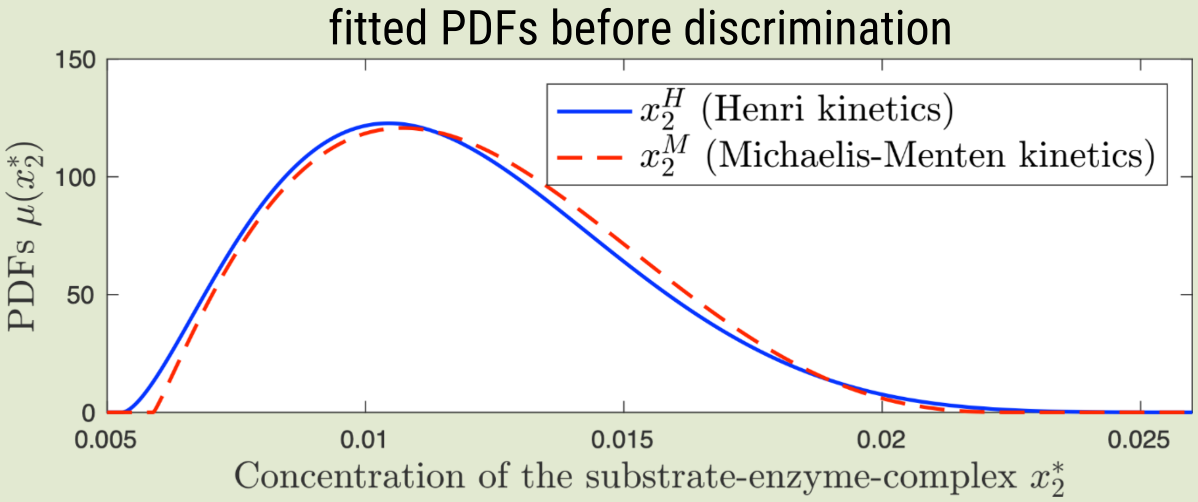
- nonlinear constraint: simulate systems ➡ calculate stochastic moments ➡ fit PDFs

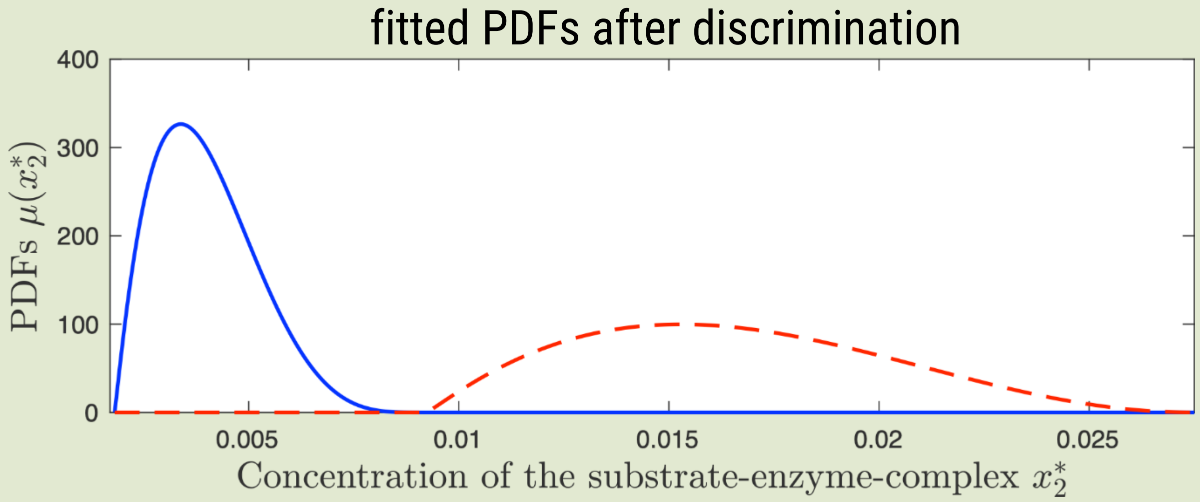
```
function [c,ceq] = ex4_mycon(u,pcesysH,pcesysM,simoptions)
  simH = PoCETsimGalerkin(pcesysH,'ex4_ODE_H',[],simoptions,'u_v',u); % simulate system H
  momH = PoCETcalcMoments(pcesysH,pcesysH.MomMats,simH.x_2.pcvals(:,end)); % calc. moments
  betaH = calcBeta4(momH); % fit 4-parameter beta distribution

  simM = PoCETsimGalerkin(pcesysM,'ex4_ODE_M',[],simoptions,'u_v',u); % simulate system M
  momM = PoCETcalcMoments(pcesysM,pcesysM.MomMats,simM.x_2.pcvals(:,end)); % calc. moments
  betaM = calcBeta4(momM); % fit 4-parameter beta distribution

  c = calcMDCbeta(betaH,betaM); % use Hellinger distance as measure of PDF overlap
  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!

*[9] Mühlpfordt, Zahn, Hagenmeyer, Faulwasser (2020). PolyChaos.jl – a Julia package for polynomial chaos in Systems and Control. In 21st IFAC WC.*

**Visit www.tu-chemnitz.de/etit/control/research/PoCET/ for more details!**