Preprints of the 21st IFAC World Congress (Virtual)
Berlin, Germany, July 12-17, 2020

PolyChaos.jl – A Julia Package for Polynomial Chaos in Systems and Control

Tillmann Mühlpfordt * **Frederik Zahn** * **Veit Hagenmeyer** *
Timm Faulwasser *

* *Institute for Automation and Applied Informatics, Karlsruhe Institute of Technology, Karlsruhe, Germany (e-mail: tillmann.muehlpfordt, frederik.zahn, veit.hagenmeyer@kit.edu; timm.faulwasser@ieee.org).*

Abstract: Polynomial chaos expansion (PCE) is an increasingly popular technique for uncertainty propagation and quantification in systems and control. Based on the theory of Hilbert spaces and orthogonal polynomials, PCE allows for a unifying mathematical framework to study systems under arbitrary uncertainties of finite variance; we introduce this problem as a so-called mapping under uncertainty. For practical PCE-based applications we require orthogonal polynomials relative to given probability densities, and their quadrature rules. With *PolyChaos.jl* we provide a Julia software package that delivers the desired functionality: given a probability density function, *PolyChaos.jl* offers several numerical routines to construct the respective orthogonal polynomials, and the quadrature rules together with tensorized scalar products. *PolyChaos.jl* is the first PCE-related software written in Julia, a scientific programming language that combines the readability of scripted languages with the speed of compiled languages. We provide illustrating numerical examples that show both PCE and *PolyChaos.jl* in action.

Keywords: polynomial chaos expansion, uncertainties, stochastic optimal control, Julia

1. INTRODUCTION

George Box's celebrated assessment that "all models are wrong, but some are useful," see (Box, 1979), may be read as an allusion to the importance of uncertainties for mathematical models: it is not just that mathematical models may be wrong qualitatively—e.g. failing to account for nonlinear phenomena—but the mathematical surrogates may be wrong also quantitatively—e.g. not being able to assign a precise numerical value to a chemical reaction rate. In the present paper we deal with the latter case: we are aware that uncertainties are present, and we are aiming for computational methods to account for them explicitly.

The traditional approach to dealing with these kinds of uncertainties is to sample and to simulate the system for each realization—the so-called Monte Carlo method. There exists a myriad of sampling-based methods that differ mostly with respect to how samples are generated and how many of them are required to capture the statistics, see (Xiu, 2010; Le Maître and Knio, 2010; Sullivan, 2015). However, sampling-based methods scale too poorly for online optimization and control applications. Polynomial chaos expansion (PCE) is a viable alternative to facilitate uncertainty propagation and uncertainty quantification. Dating back to (Wiener, 1938) PCE is a Hilbert space technique that expands random variables in terms of polynomials that are orthogonal relative to the underlying probability measure. Xiu and Karniadakis (2002) extended Wiener's work to beyond the Gaussian measure. The advantage of PCE for applications in systems and control is that it renders infinite-dimensional problems finite-dimensional. This facilitates the simulation of ordinary differential equations under uncertainty, stochastic optimal control problems, or optimization problems under uncertainty, see for instance (Hover and Triantafyllou, 2006; Kim et al., 2013; Paulson et al., 2017, 2015; Fagiano and Khammash, 2012; Mühlpfordt et al., 2016; Bradford et al., 2019; Mühlpfordt et al., 2018; Mühlpfordt et al., 2019).

Whenever PCE is applied, however, we need to know the orthogonal bases for given probability densities, and we need to solve integrals efficiently. This computational overhead calls for efficient, easy-to-use, and well-documented software. With *PolyChaos.jl*, which is available open source (Mühlpfordt et al., 2019), we deliver the first PCE-related software in the Julia programming language: given an arbitrary probability density function, *PolyChaos.jl* allows to construct the orthogonal bases via several routines from Gautschi (2004). Furthermore, quadrature rules to solve integrals or tensorized scalar products of the basis functions are available easily. Although there exist several software packages that provide PCE functionality, e.g. *UQLab* (Matlab, (Marelli and Sudret, 2014)), *Chaospy* (Python, (Feinberg and Langtangen, 2015)), or *OpenTURNS* (Python, (Baudin et al., 2017)), none previously existed in the Julia programming language, see (Bezanson et al., 2017). Julia is dedicated to scientific computing, and it aims to combine the readability of scripted languages with the performance of compiled languages.

⋆ This work was supported by the Helmholtz Association under the joint initiative "Energy System 2050 – A Contribution of the Research Field Energy."
TF is now with: Department of Electrical Engineering and Information Technology, TU Dortmund University, Dortmund, Germany.

Three main parts make up the present paper: Section 2 covers the theoretical framework: Given an input random variable, and given a suitable mapping, what is the image random variable? Undeniably, we provide but a glance at the theory of PCE; we refer to the rich literature for more details, e.g. (Xiu, 2010; Le Maître and Knio, 2010; Sullivan, 2015; Streif et al., 2013). The gist of Section 2 is to show the PCE-practitioner that the applicability of the method hinges on two key items: the orthogonal bases must be known, and integrals must be solved efficiently. Section 3 introduces the concepts permeating *PolyChaos.jl*. We focus on the concepts and omit implementational details as the code is available open source, see (Mühlpfordt et al., 2019). Finally, in Section 4 we show how to apply *PolyChaos.jl* to three numerical problems.

2. PROBLEM FORMULATION

2.1 Setting

We study mappings under uncertainty: given some input random variable, we are interested in the image random variable stemming from a known mapping.

Problem 1. (Mapping under uncertainty). Let $(\Omega, \mathfrak{F}, \mu)$ be a probability space with sample space Ω, sigma algebra $\mathfrak{F}$, and an absolutely continuous, non-negative probability measure μ. Also, let $L^2(\Omega, \mu; \mathbb{R})$ be the Hilbert space of all (equivalence classes of) real-valued random variables of finite variance. Given the random variable $\mathsf{x} \in L^2(\Omega, \mu; \mathbb{R})$ find the random variable $\mathsf{y} \in L^2(\Omega, \mu; \mathbb{R})$ that is defined implicitly via

$$0 = F(\mathsf{y}, \mathsf{x}), \tag{1}$$

where $F(\cdot)$ is a suitable implicit mapping.

The mapping $F(\cdot)$ in Problem 1 may stand for the solution to a (discretized) ordinary differential equation, for the solution to a discrete-time system, for a (system) of nonlinear algebraic equations, or for an argmin operator.

Remark 1. (Several sources of uncertainty).
For sake of readability we restrict our presentation to single sources of uncertainty, hence univariate polynomial bases. For m independent sources of uncertainty we can construct the m-variate orthogonal basis from the product of the m respective univariate bases, see (Xiu, 2010; Sullivan, 2015).

Remark 2. (Several random variables).
For sake of notation both Problem 1 consider a single input uncertainty x that is mapped to a single image random variable y. The extension to several uncertainties $\mathsf{x}_i \in L^2(\Omega, \mu; \mathbb{R})$ for $i \in \{1, \dots, n_x\}$ and several image random variables y_i for $i \in \{1, \dots, n_y\}$ would lead to substituting (1) with

$$0 = F(\mathsf{y}_1, \dots, \mathsf{y}_{n_y}, \mathsf{x}_1, \dots, \mathsf{x}_{n_x}).$$

Problem 1 is infinite-dimensional, hence intrinsically challenging. A popular method to render Problem 1 tractable is polynomial chaos expansion (PCE): a Hilbert space technique for random variables that is mathematically equivalent to Fourier series for periodic functions.

2.2 Polynomial chaos expansion

The PCE of the random variable $\mathsf{x} \in L^2(\Omega, \mu; \mathbb{R})$ is

$$\mathsf{x} = \sum_{k \in \mathcal{K}} x_k \phi_k, \tag{2}$$

with $\mathcal{K} \subseteq \mathbb{N}_0$, and where $\{\phi_k\}_{k \in \mathbb{N}_0}$ is an ordered set of monic orthogonal polynomials that forms a complete orthogonal sequence in $L^2(\Omega, \mu; \mathbb{R})$, see (Xiu and Karniadakis, 2002; Sullivan, 2015). The polynomials ϕ_k satisfy

$$\phi_0 = 1, \tag{3a}$$

$$\phi_k(\tau) = \tau^k + a_{k-1}\tau^{k-1} + \ldots + a_0, \qquad \forall k \in \mathbb{N}, \tag{3b}$$

$$\langle \phi_i, \phi_j \rangle = \int_{\mathbb{R}} \phi_i(\tau)\phi_j(\tau) \mathrm{d}\mu(\tau) = \gamma_i \delta_{ij}, \quad \forall i, j \in \mathbb{N}_0, \tag{3c}$$

where $\gamma_i > 0$, and δ_{ij} is the Kronecker-delta. The PCE from (2) is, generally speaking, exact whenever the index set $\mathcal{K}$ is equal to $\mathbb{N}_0$. In case we truncate the PCE from (2) to finitely many terms $\mathcal{K} = \{0, \dots, \hat{k}\}$ there may be a truncation error—which is minimal with respect to the induced norm of the space $L^2(\Omega, \mu; \mathbb{R})$, cf. (Xiu and Karniadakis, 2002; Sullivan, 2015).

Given Problem 1 and given the PCE (2) of x, what are the PCE coefficients y_k of $\mathsf{y} = \sum_{k \in \mathcal{K}} y_k \phi_k$ such that

$$0 = F\left(\sum_{k \in \mathcal{K}} y_k \phi_k, \sum_{k \in \mathcal{K}} x_k \phi_k\right) \tag{4}$$

holds, and how can we compute them? Galerkin projection is an option. The idea of Galerkin projection is to project the PCE-overloaded model (4) onto every basis element of $\{\phi_k\}_{k \in \mathcal{K}}$, see (Sullivan, 2015). This leads to a set of deterministic equations in the form of integrals

$$0 = \left\langle F\left(\sum_{k \in \mathcal{K}} y_k \phi_k, \sum_{k \in \mathcal{K}} x_k \phi_k\right), \phi_m \right\rangle \tag{5a}$$

$$= \int_\Omega F\left(\sum_{k \in \mathcal{K}} y_k \phi_k(\tau), \sum_{k \in \mathcal{K}} x_k \phi_k(\tau)\right) \phi_m(\tau) \mathrm{d}\mu(\tau) \tag{5b}$$

for all $m \in \mathcal{K}$. There are two main approaches to solve the integral in (5): non-intrusive and intrusive approaches. Non-intrusive approaches tackle the integral by quadrature, Monte Carlo integration, or least-squares. Intrusive approaches modify the expression (5) to derive integrals that are either simpler to evaluate or that admit an exact Gauss quadrature.

Remark 3. (Other advantages of PCE). PCE offers other advantages than facilitating uncertainty propagation for Problem 1: it neither relies on sampling nor is it restricted to a specific family of distributions such as Gaussian distributions. Also, moments of random variables are functions of the PCE coefficients, see (Xiu, 2010; Sullivan, 2015).

2.3 Revised setting

Let us revisit Problem 1—mappings under uncertainty—in light of PCE.

Problem 2. (Mapping under uncertainty using PCE). Consider the setup from Problem 1. Additionally, let the PCE of the random variable x be given by $\mathsf{x} = \sum_{k \in \mathcal{K}} x_k \phi_k$ for a known orthogonal basis $\{\phi_k\}_{k \in \mathcal{K}}$ and a known probability measure μ. Then, find the PCE coefficients y_k of $\mathsf{y} = \sum_{k \in \mathcal{K}} y_k \phi_k$ such that (5) holds.

Example 1. (Van de Vusse reaction under uncertainty).
Consider a van de Vusse reaction with two uncertain

Table 1. Askey scheme for "classical" distributions (Xiu and Karniadakis (2002)).

Type	Support	$\phi_k(\tau)$	Polynomial basis
Beta	$(0,1)$	$\mathrm{P}_k^{(\beta-1,\alpha-1)}(2\tau-1)$	Jacobi
Gamma	$(0,\infty)$	$\mathrm{L}_k^{(\alpha-1)}(\beta\tau)$	Gen. Laguerre
Gaussian	$(-\infty,\infty)$	$\mathrm{He}_k(\tau)$	Hermite
Uniform	$[0,1]$	$\mathrm{P}_k^{(0,0)}(2\tau-1)$	Legendre

reaction rates, see (Paulson et al., 2015; Scokaert and Rawlings, 1998). The dynamics of the concentrations can be modeled as

$$\dot{\mathsf{c}}_A = -\mathsf{c}_A u - \mathsf{r}_1 \mathsf{c}_A - r_3 \mathsf{c}_A^2, \qquad \mathsf{c}_A(0) = \mathsf{c}_{A,0}, \tag{6a}$$
$$\dot{\mathsf{c}}_B = -\mathsf{c}_B u + \mathsf{r}_1 \mathsf{c}_A - \mathsf{r}_2 \mathsf{c}_B, \qquad \mathsf{c}_B(0) = \mathsf{c}_{B,0}, \tag{6b}$$

with uncertain reaction rates $\mathsf{r}_1, \mathsf{r}_2$, a certain reaction rate r_3, and a fixed dilution rate u. In general, the initial conditions may be uncertain too. Using the notation of Problem 2 and Remark 2 we have $(\mathsf{x}_1, \mathsf{x}_2, \mathsf{x}_3, \mathsf{x}_4) = (\mathsf{r}_1, \mathsf{r}_2, \mathsf{c}_{A,0}, \mathsf{c}_{B,0})$ and $(\mathsf{y}_1, \mathsf{y}_2) = (\mathsf{c}_A, \mathsf{c}_B)$. Inserting the PCE for all x_i with $i \in \{1,2,3,4\}$ and y_i with $i \in \{1,2\}$, the Galerkin projection (5) for the system (6) becomes [1]

$$\dot{c}_{A,k_1} = -c_{A,k_1} u - \sum_{k_2,k_3 \in \mathcal{K}} (r_{1,k_2} c_{A,k_3} + r_3 c_{A,k_2} c_{A,k_3}) \nu_{k_1 k_2 k_3} \tag{7a}$$

$$\dot{c}_{B,k_1} = -c_{B,i} u + \sum_{k_2,k_3 \in \mathcal{K}} (r_{1,k_2} c_{A,k_3} - r_{2,k_2} c_{B,k_3}) \nu_{k_1 k_2 k_3} \tag{7b}$$

for all $k_1 \in \mathcal{K}$, and

$$\nu_{k_1 k_2 k_3} = \langle \phi_{k_1} \phi_{k_2}, \phi_{k_3} \rangle / \langle \phi_{k_1}, \phi_{k_1} \rangle. \tag{7c}$$

The Galerkin projection of the initial conditions leads to

$$c_{A,k}(0) = c_{A,0,k}, \; c_{B,k}(0) = c_{B,0,k} \tag{7d}$$

for all $k \in \mathcal{K}$. The Galerkin-projected system (7) is a system of ordinary differential equations in terms of the PCE coefficients of the concentrations. As a consequence of applying PCE to (6) we need to know the basis polynomials $\{\phi_k\}_{k\in\mathcal{K}}$ and the numbers $\nu_{k_1 k_2 k_3}$.

So far, the significance of Problem 2 is more theoretical than practical. To apply Problem 2 in practice we have to address the assumptions it hinges on, namely

- A1 the orthogonal basis polynomials are known, and
- A2 the integrals from (5)—respectively the scalars

$$\langle \phi_{k_1} \cdots \phi_{k_{n-1}}, \phi_{k_n} \rangle \tag{8}$$

for some $n \in \mathbb{N}$—can be computed efficiently.

For several well-known and often-employed uncertainties the Askey scheme provides orthogonal polynomials, see Table 1; accompanying quadrature rules to solve the integrals (5) follow from the Golub-Welsch algorithm, see (Golub and Welsch, 1969). In case of arbitrary probability densities we can utilize the Stieltjes procedure or the Lanczos procedure to construct the orthogonal polynomials—and then compute the quadrature rule according to Golub and Welsch (1969), or similar Gauss-like quadratures such as Gauss-Radau or Gauss-Lobatto, see (Gautschi, 2002).

[1] The observant reader noticed that, strictly speaking, we are leaving the setting from Problem 1, i.e. the realm of mere Hilbert spaces, with the setting from (6). The rigorous introduction of random ordinary differential equations of the form (6) is beyond the scope of this paper. Our focus is on the Galerkin-based reformulation and the quantities introduced by PCE.

2.4 Construction of orthogonal polynomials

Before we explain numerical procedures we need to introduce a core theorem related to orthogonal polynomials.

Theorem 3. (Recurrence relation). Let ϕ_k with $k \in \mathbb{N}_0$ be the monic orthogonal polynomials with respect to the measure μ. Then,

$$\phi_{k+1}(\tau) = (\tau - \alpha_k)\phi_k(\tau) - \beta_k \phi_{k-1}(\tau), \quad k \in \mathbb{N}_0, \tag{9a}$$
$$\phi_{-1}(\tau) = 0, \quad \phi_0(\tau) = 1 \tag{9b}$$
$$\alpha_k = \frac{\langle \tau\phi_k, \phi_k \rangle}{\langle \phi_k, \phi_k \rangle}, \quad \beta_k = \frac{\langle \phi_k, \phi_k \rangle}{\langle \phi_{k-1}, \phi_{k-1} \rangle} \tag{9c}$$

Proof: See (Gautschi, 2004, 1.3.1). □

We hence identify orthogonal polynomials by their sequence of recurrence coefficients $\{(\alpha_k, \beta_k \in)\}_{k\in\mathbb{N}_0}$—which we seek to compute for a given measure μ. Moment-based methods are one possibility: based on the moments of the underlying measure, the recurrence coefficients can be computed. This is the idea of Paulson et al. (2017). Unfortunately, this often leads to ill-conditioned problems, see (Gautschi, 2004). The Stieltjes procedure and the Lanczos procedure are numerically stable alternatives.

Stieltjes procedure Theorem 3 suggests a simple iterative procedure to compute the recurrence coefficents α_k, β_k, the so-called Stieltjes procedure: For $k = 0$ we define $\beta_0 = \int \mathrm{d}\mu(\tau) = 1$, from which we can compute $\alpha_0 = \langle \tau \rangle$, cf. (9b). Knowing (α_0, β_0) we obtain ϕ_1 from (9a). For $k = 1$ we compute $\beta_1 = \langle \phi_1, \phi_1 \rangle$, from which we get $\alpha_1 = \langle \tau\phi_1, \phi_1 \rangle / \langle \phi_1, \phi_1 \rangle$. Knowing (α_1, β_1) we can construct ϕ_2 from (9a). We can repeat this procedure until a desired degree k is reached.

The Stieltjes procedure is straightfoward to implement. All occurring integrals can be solved efficiently using quadrature rules. In case of numerical issues such as over- or underflow Gautschi (2004) suggests to scale the weights and polynomials.

Lanczos procedure The Lanczos algorithm allows to tri-diagonalize a given symmetric matrix A, see (Golub and Van Loan, 1983). More specifically, a real symmetric matrix A allows the transformation $Q^\top A Q = T$, where Q is orthogonal and T is symmetric and tridiagonal. Given the matrix A, Lanczos' algorithm produces the matrices Q and T. In light of orthogonal polynomials, the Lanczos *procedure* means to construct A such that the output of the Lanczos algorithm is the Jacobi matrix from which the recurrence coefficients can be read off. Gautschi (2004) shows how to construct the matrix A from the quadrature rule employed to solve the integrals. [2]

Multiple discretization Sometimes the underlying absolutely continuous probability measure μ allows to decompose integrals of some function f into $m \in \mathbb{N}$ parts according to

$$\int_{[a,b]} f(\tau)\mathrm{d}\mu(\tau) = \sum_{i=1}^{m} \int_{[a_i,b_i]} f_i(\tau)\mathrm{d}\mu_i(\tau), \tag{10}$$

[2] For details on the Lanczos algorithm itself we refer to (Golub and Van Loan, 1983) and/or (Gragg and Harrod, 1984).

where $[a, b] = \cup_{i=1}^{m}[a_i, b_i]$ with $-\infty \leq a < b \leq \infty$. For instance, this may be the case for mixture models or densities defined on disjoint intervals. Assuming we can apply either the Stieltjes procedure or the Lanczos procedure to the m measures μ_i, Gautschi (2004) proposes—in the spirit of *divide et impera*—a heuristic algorithm to construct the orthogonal polynomials relative to the measure μ.

2.5 Intermediate summary

Given an input uncertainty in terms of a continuous random variable of finite variance we are interested in image random variables stemming from a known mapping, see Problem 1. To apply the procedure in practice we require tools that compute orthogonal polynomials given a probability density, and that compute quadrature rules to solve the integrals. The main contribution of the present paper is to introduce the Julia package *PolyChaos.jl*—built for this very purpose.

3. POLYCHAOS.JL

With *PolyChaos.jl* we deliver a software package in the Julia programming language that provides numerical routines for PCE-related computations, specifically addressing items A1 and A2 from Section 2.3.

3.1 Existing software

Table 2 lists existing software packages for PCE. Except for *Chaospy* and *PolyChaos.jl* these packages are full-fledged libraries for uncertainty quantification; PCE comprises just one module of many, and it is used mostly for non-intrusive applications. Amongst the software from Table 2 *UQLab* and *Dakota* provide the richest functionality, each coming with a superb documentation. While the core functions of *UQLab* are closed source, the scientific methods surrounding PCE are all available under the BSD 3-clause license. Furthermore, *UQLab* provides methods for basis-adaptive PCE based on (Blatman and Sudret, 2010). *Dakota* is a mature framework: currently at version 6.0, version 3.0 beta, for instance, dates back to 2001. The functionality of *MUQ* and *UQToolkit* is comparable; unfortunately they do not allow to compute orthogonal polynomials for arbitrary probability densities. *OpenTURNS* is a full-fledged uncertainty quantification framework that comes with rich and mathematically detailed documentation. The solely PCE-centered Python package *Chaospy* comes with the least restrictive MIT-license whilst providing the core PCE functionality that includes the computation of orthogonal polynomials for arbitrary probability densities.

Table 2 positions *PolyChaos.jl* in the landscape of software packages for PCE: its premise is to support arbitrary probability densities, for which it provides not just the Stieltjes but also the Lanczos procedure based on (Gautschi, 2004). In case the density can be composed as a sum of individual densities—as is common for (Gaussian) mixture models—*PolyChaos.jl* provides a specific method for multiple discretization based on (Gautschi, 2004). Moreover, *PolyChaos.jl* provides several quadrature rules.

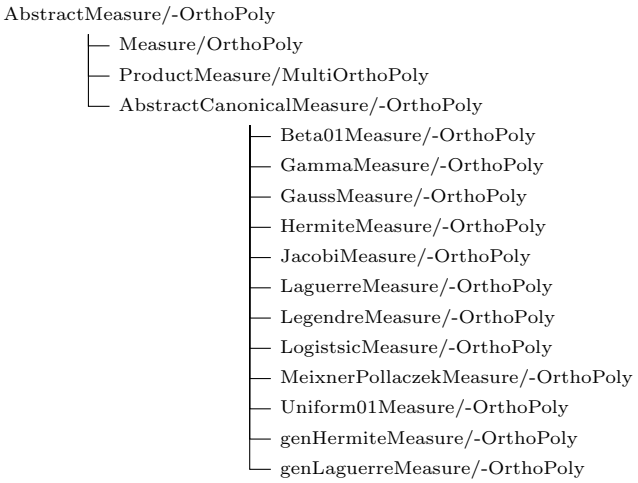

Fig. 1. Type hierarchy for measures and orth. polynomials.

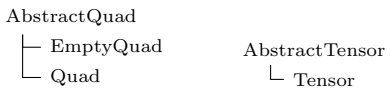

Fig. 2. Type hierarchy for quadrature rules and tensors.

3.2 Why Julia?

Julia is a just-in-time compiled programming language for scientific computing, see (Bezanson et al., 2017). Julia solves the so-called two-language problem, the undesirable situation in which a programmer creates prototypes in one language (often based on easy-to-read scripts or notebooks, e.g. Matlab or Python), then having to switch to a different language (often compiled, e.g. C/C++) to achieve fast execution times. Julia is a platform for both: rapid prototypes with intuitive code design can be morphed into type-specific, high-performance code. The fact that types of values need not be declared explicitly is one reason why Julia solves the aforementioned two-language problem: users may never feel the need to declare types, yielding code reminiscent of scripted languages, yet users *can* leverage the full power and expressiveness of Julia's type system to write cleaner and fast code. With *PolyChaos.jl* we provide the first Julia package dedicated to orthogonal polynomials, quadrature rules, and PCE.

3.3 Type hierarchy

Every value in Julia has a type. The conceptual foundation of the type system relies on *abstract* types. Abstract types serve but a single purpose: to form a type hierarchy. It is neither desired nor possible to instantiate abstract types. The type hierarchy remains independent from functions that operate on types. To get what in other languages is called a struct or an object Julia provides *composite* types. A composite type has fields, [3] it can be instantiated, and it can be declared a subtype of abstract types.

For *PolyChaos.jl* we devise our own type hierarchy. Figure 1 shows the two bread-and-butter type trees we need; abstract types carry the prefix "Abstract." Take the abstract type *AbstractMeasure*: it has two composite sub-

[3] Methods can be fields too, the type of the field being *Function*.

Table 2. Existing software packages for polynomial chaos expansion.

Name	Language	Features for polynomial chaos expansion	License	Reference
UQLab	Matlab	- Classic and arbitrary distributions - Stieltjes procedure - Gauss and sparse quadrature - Basis-adaptive sparse PCE - Least-angle regression	BSD 3-clause	Marelli and Sudret (2014)
Chaospy	Python	- Classic and arbitrary distributions - Gram-Schmidt, Stieltjes procedure - Gauss quadrature, Clenshaw-Curtis	MIT	Feinberg and Langtangen (2015)
OpenTURNS	Python	- Classic and arbitrary distributions - Stieltjes procedure - Gauss quadrature	GNU LPGL	Baudin et al. (2017)
Dakota	C++	- Classic and arbitrary distributions - Stieltjes, Gram-Schmidt, Chebyshev - Gauss and sparse quadrature - Stochastic collocation	GNU LPGL	Adams et al. (2014)
MUQ	C++, Python	- Classic distributions - Gauss quadrature	n/a	Conrad and Marzouk (2013)
UQToolkit	C++, Python	- Classic distributions - Gauss quadrature	GNU LGPL	Debusschere et al. (2016)
PolyChaos.jl	Julia	- Classic and arbitrary distributions - Stieltjes and Lanczos procedure - Multiple discretization - Gauss quadrature, Fejér, Clenshaw-Curtis - Tensorized scalar products	MIT	Mühlpfordt et al. (2019)

types: *Measure* and *ProductMeasure* with obvious meanings. There exist, however, well-studied canonical measures such as Gaussian or uniform measures for which we introduce the abstract subtype *AbstractCanonicalMeasure*. All subtypes of *AbstractCanonicalMeasure* are shown in Figure 1. The type hierarchy for orthogonal polynomials mirrors that of measures: there are generic composite types for univariate polynomials, namely *OrthoPoly*, and multivariate polynomials, namely *MultiOrthoPoly*, and there are canonical orthogonal polynomials. Figure 2 adds to the overall *PolyChaos.jl* type system quadrature rules via *AbstractQuad* and tensors of scalar products via *AbstractTensor*. We emphasize once more that the type hierarchies from Figure 1 and Figure 2 describe a *concept* and no implementation. The implementation details of all routines are beyond the scope of the present paper; we acknowledge that several routines are inspired by the Matlab code accompanying Gautschi (2004).

4. NUMERICAL EXAMPLES

We consider three numerical examples that use the features of *PolyChaos.jl*. The first example is about constructing the orthogonal basis for a non-trivial probability density; the second example studies uncertainty propagation for Example 1; the third example demonstrates optimal control for discrete-time systems under uncertainty.

The code for all numerical examples is available online. [4]

4.1 Beta mixture – Basis construction

Consider a continuous random variable $\mathsf{z} \in L^2(\Omega, \mu; \mathbb{R})$ with the absolutely continuous measure $\mathrm{d}\mu(\tau) = \rho(\tau)\mathrm{d}(\tau)$ being a Beta mixture with two components, hence the density for all $\tau \in (0,1)$ is given by

$$\rho(\tau) = w_1 \rho_B(\tau; \alpha_1, \beta_1) + w_2 \rho_B(\tau; \alpha_2, \beta_2), \qquad (11)$$

[4] See https://github.com/timueh/VanDeVusseUnderUncertainty .

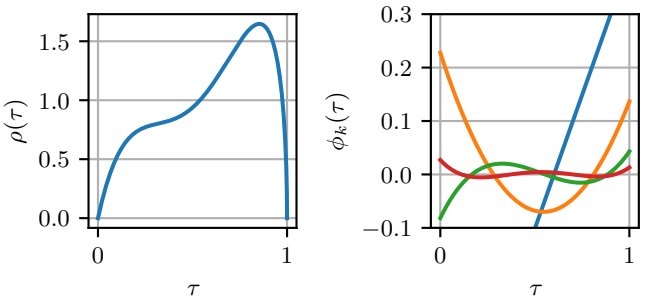

Fig. 3. Beta mixture density $\rho(\tau)$ from (11) and respective orthogonal polynomials ϕ_k for degrees $k \in \{1,2,3,4\}$.

with $w_1 + w_2 = 1$, and where $\rho_B(\tau; \alpha, \beta) = \tau^{\alpha-1}(1-\tau)^{\beta-1}/B(\alpha,\beta)$ for all $\tau \in (0,1)$ is a standard Beta density with positive shape parameters α, β. Figure 3 shows the density (11) for the specific values

w_1	α_1	β_1	w_2	α_2	β_2
0.3	2.0	4.5	0.7	4.0	1.5

.

We employ multiple discretization from Section 2.4 to construct the first 5 polynomials that are orthogonal relative to (11). This first requires to construct the orthogonal basis for each individual Beta distribution—for which we can use the built-in type *Beta01OrthoPoly*, see Figure 1. Applying multiple discretization using the *PolyChaos.jl* function *mcdisretization()*, we obtain the following orthogonal polynomials for degrees 1 to 4

$$\phi_1(\tau) = \tau - 0.6 \qquad (12a)$$
$$\phi_2(\tau) = \tau^2 - 1.09\tau + 0.23 \qquad (12b)$$
$$\phi_3(\tau) = \tau^3 - 1.6\tau^2 + 0.73\tau - 0.08 \qquad (12c)$$
$$\phi_4(\tau) = \tau^4 - 2.11\tau^3 + 1.47\tau^2 - 0.38\tau + 0.03, \qquad (12d)$$

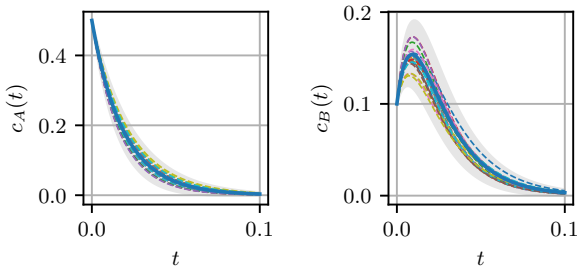

Fig. 4. Total of 20 realizations of the Van de Vusse reaction from Section 4.2; shaded area denotes the $\mathbb{E}(\mathsf{c}_i(t)) \pm 3\sigma(\mathsf{c}_i(t))$ interval with $i \in \{A, B\}$.

which are plotted in Figure 3. [5]

4.2 Van de Vusse reaction – Uncertainty propagation

Recall Example 1 which results in the Galerkin-projected ordinary differential equations for the Van de Vusse reaction with uncertainties. We choose the uncertain reaction ratees r_1 and r_2 each to follow an independent uniform distribution. The parameters are

$\mathbb{E}(\mathsf{r}_1)$	$\sigma(\mathsf{r}_1)$	$\mathbb{E}(\mathsf{r}_2)$	$\sigma(\mathsf{r}_2)$	r_3
50	$0.1\,\mathbb{E}(\mathsf{r}_1)$	100	$0.1\,\mathbb{E}(\mathsf{r}_2)$	10

,

where the values for $\mathbb{E}(\mathsf{r}_1), \mathbb{E}(\mathsf{r}_2), r_3$ are taken from Scokaert and Rawlings (1998). For simplicity we choose deterministic initial conditions with

$$\mathsf{c}_{A,0} \equiv c_{A,0} = 0.5, \quad \mathsf{c}_{B,0} \equiv c_{B,0} = 0.1,$$

and a constant dilution rate of $u = 0.1$. We construct the basis using the built-in type *Uniform01OrthoPoly* and compute the tensorized scalar products $\nu_{k_1 k_2 k_3}$ from (7) using the type *Tensor*, see Figures 1 and 2. The set of differential equations (7) is integrated using the Julia package *DifferentialEquations.jl*, see (Rackauckas and Nie, 2017). [6] From the solution of the PCE coefficients we can compute immediately moments, allowing to plot the $\mathbb{E}(\mathsf{c}_i(t)) \pm 3\sigma(\mathsf{c}_i(t))$ interval with $i \in \{A, B\}$ without having to sample, cf. Remark 3. This is shown in Figure 4, along with the solution trajectories for 20 realizations of the uncertainties that validate our findings. The PCE-based mean $\mathbb{E}(\mathsf{c}_i(t))$ for $i \in \{A, B\}$ of the random-variable solutions corresponds to the solid line.

4.3 Van de Vusse reaction – Stochastic optimal control

We consider the linearized and discretized van de Vusse reaction according to Paulson et al. (2015)

$$A = \begin{pmatrix} \mathsf{k} & 0 \\ 0.088 & 0.819 \end{pmatrix}, \quad B = \begin{pmatrix} -0.005 \\ -0.002 \end{pmatrix}, \tag{13a}$$

where the parameter k is uncertain according to $\mathsf{k} = \underline{k} + (\overline{k} - \underline{k})\mathsf{z}$, where z has the probability density from Section 4.1, and $(\underline{k}, \overline{k}) = (0.923, 0.926)$ is the support of k. [7] The initial conditions $\mathsf{x}_{1,0}, \mathsf{x}_{2,0}$ of the system (13a) are independent Gaussian random variables with

$\mathbb{E}(\mathsf{x}_{1,0})$	$\sigma(\mathsf{x}_{1,0})$	$\mathbb{E}(\mathsf{x}_{2,0})$	$\sigma(\mathsf{x}_{2,0})$
1/2	1/60	1/10	1/100

.

We wish to solve the following stochastic optimal control problem over the horizon of $\mathcal{T} = \{0, 1, \ldots, 74\}$

$$\min_{\substack{u(t) \\ \forall t \in \mathcal{T}}} \sum_{t \in \mathcal{T}} \mathbb{E}\left(\mathsf{x}(t+1)^\top Q \mathsf{x}(t+1)\right) + u(t)^\top R u(t) \tag{14a}$$

s.t.

$$\mathsf{x}(t+1) = A\mathsf{x}(t) + Bu(t), \quad \forall t \in \mathcal{T}, \tag{14b}$$

$$\mathsf{x}(0) = \begin{pmatrix} \mathsf{x}_{1,0} \\ \mathsf{x}_{2,0} \end{pmatrix}, \tag{14c}$$

$$\mathbb{E}(\mathsf{x}_2(\mathsf{t})) + \lambda\sigma(\mathsf{x}_2(t)) \leq \overline{x}_2, \quad \forall t \in \mathcal{T} \cup \{75\}, \tag{14d}$$

with positive definite weights $Q = I_2, R = 1$. [8] The inequality constraint (14d) is a reformulated chance constraint for $\lambda = 1.618$ and the upper limit $\overline{x}_2 = 0.17$, see (Paulson et al., 2015). We construct the basis using the built-in type *GaussOrthoPoly* twice together with the numerically computed basis from Section 4.1. Galerkin projection can be applied to Problem (14), which again leads to tensorized scalar products (Paulson et al., 2015)

$$\langle \phi_{k_1} \phi_{k_2}, \phi_{k_3} \rangle, \langle \phi_{k_1}, \phi_{k_2} \rangle \tag{15}$$

for all $k_1, k_2, k_3 \in \mathcal{K}$, for which we use the type *Tensor*. Figure 5 shows the trajectories of the second state for a total of 20 realizations of the uncertainties. In Figure 5 the solid line denotes the PCE-obtained trajectory of the mean. Owing to the chance constraint reformulation (14d), the bound $\overline{x}_2 = 0.17$ may be violated. For a total of 100,000 realizations (not shown) we empirically find that about 6 % of the trajectories violate the constraint.

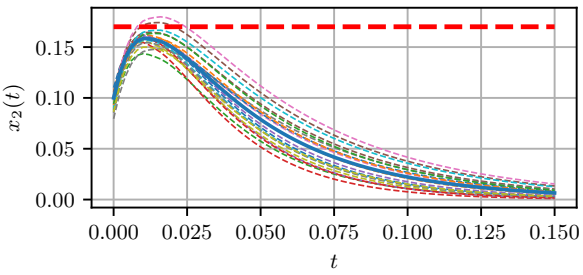

Fig. 5. Total of 20 realizations of the second state of the linearized and discretized Van de Vusse reaction from Section 4.3.

4.4 Computation times

So far we showed the numerical results of three different examples. We commented how to obtain the PCE-related quantities using *PolyChaos.jl*. To assess the performance of *PolyChaos.jl* within each example we measured the times to construct the basis, and to compute the scalars (15) for 10,000 consecutive runs, including the first run to compile the code; Table 3 shows the mean times

[5] Coefficients in (12) are rounded to two decimals; Figure 3 is based on all decimals.

[6] In light of Footnote 1 we remark that, generally speaking, we solve differential equations numerically by some form of discretization. In that case we can view the task at hand as Problem 1 together with Remark 2. This is true because random vectors can be viewed equivalently as discrete-time stochastic processes, see Sullivan (2015).

[7] The support is chosen equivalent to the support from Paulson et al. (2015), where k is modeled as a single Beta distribution.

[8] In light of Footnotes 1 and 6 we remark that the random-variable discrete-time system from (14) may be viewed either as a random vector or as a discrete-time stochastic process.

Table 3. Computation times with *PolyChaos.jl*; N_{unc} is the number of uncertainties.

		Mean time in μs for 10,000 runs		
Example	N_{unc}	Basis	$\langle \phi_{k_1}, \phi_{k_2} \rangle$	$\langle \phi_{k_1} \phi_{k_2}, \phi_{k_3} \rangle$
Beta mixture	1	120	20	100
Propagation	2	47	131	1,089
Optimization	3	171	707	12,888

obtained on a desktop computer with an Intel® Core™ i7-8700 CPU 3.20GHz processor and 31 GiB of RAM. The maximum degree of the basis polynomials is 4 for all cases. As we can see from Table 3 the computation time overhead stemming from *PolyChaos.jl* is negligible for the considered examples.

5. SUMMARY AND OUTLOOK

We introduce mappings under uncertainty and demonstrate how to solve them using polynomial chaos expansion, a Hilbert space technique for random variables of finite variance. Polynomial chaos imposes mainly two computational burdens: finding orthogonal polynomials given a probability density, and determining their quadrature rules. To facilitate these computations we introduce *PolyChaos.jl*, a software package written in the Julia programming language. Three numerical examples demonstrate how to use *PolyChaos.jl* for specific mappings under uncertainty. The computation time overhead stemming from *PolyChaos.jl* is negligible for the studied examples. Future work will focus on comparing the computational speed and accuracy to other existing software packages. Also, additional functionalities such as stochastic collocation and/or sparse basis construction are desirable to add.

REFERENCES

Adams, B., Bauman, L., Bohnhoff, W., Dalbey, K., Ebeida, M., Eddy, J., Eldred, M., Hough, P., Hu, K., Jakeman, J., Stephens, J., Swiler, L., Vigil, D., and Wildey, T. (2014). Dakota, a multilevel parallel object-oriented framework for design optimization, parameter estimation, uncertainty quantification, and sensitivity analysis: Version 6.0 user's manual. Technical report, Sandia National Lab SAND2014-4633.

Baudin, M., Dutfoy, A., Iooss, B., and Popelin, A.L. (2017). *OpenTURNS: An Industrial Software for Uncertainty Quantification in Simulation*. Springer International Publishing, Cham.

Bezanson, J., Edelman, A., Karpinski, S., and Shah, V. (2017). Julia: A fresh approach to numerical computing. *SIAM Review*, 59(1), 65–98.

Blatman, G. and Sudret, B. (2010). An adaptive algorithm to build up sparse polynomial chaos expansions for stochastic finite element analysis. *Prob. Eng. Mech.*, 25(2), 183–197.

Box, G. (1979). Robustness in the strategy of scientific model building. In R. Launer and G. Wilkinson (eds.), *Robustness in Statistics*, 201–236. Academic Press.

Bradford, E., Reble, M., and Imsland, L. (2019). Output feedback stochastic nonlinear model predictive control of a polymerization batch process. In *European Control Conference*, 3144–3151.

Conrad, P. and Marzouk, Y. (2013). Adaptive Smolyak pseudospectral approximations. *SIAM Journal on Scientific Computing*, 35(6), A2643–A2670.

Debusschere, B., Sargsyan, K., Safta, C., and Chowdhary, K. (2016). *Uncertainty Quantification Toolkit (UQTk)*, 1–21. Springer International Publishing, Cham.

Fagiano, L. and Khammash, M. (2012). Nonlinear stochastic model predictive control via regularized polynomial chaos expansions. In *Conf. on Dec. and Contr. (CDC)*, 142–147.

Feinberg, J. and Langtangen, H. (2015). Chaospy: An open source tool for designing methods of uncertainty quantification. *Journal of Computational Science*, 11, 46– 57.

Gautschi, W. (2002). The interplay between classical analysis and (numerical) linear algebra – A tribute to Gene H. Golub. *Electronic Transactions on Numerical Analysis*, 13, 119–147.

Gautschi, W. (2004). *Orthogonal Polynomials: Computation and Approximation*. Oxford University Press, Oxford.

Golub, G. and Van Loan, C. (1983). *Matrix Computations*. The Johns Hopkins University Press, Baltimore, MD, 4th edition.

Golub, G. and Welsch, J. (1969). Calculation of Gauss quadrature rules. *Mathematics of Computation*, 23, 221–230.

Gragg, W. and Harrod, W. (1984). The numerically stable reconstruction of Jacobi matrices from spectral data. *Numerische Mathematik*, 44(3), 317–335.

Hover, F. and Triantafyllou, M. (2006). Application of polynomial chaos in stability and control. *Automatica*, 42(5), 789–795.

Kim, K., Shen, D., Nagy, Z., and Braatz, R. (2013). Wiener's polynomial chaos for the analysis and control of nonlinear dynamical systems with probabilistic uncertainties [historical perspectives]. *IEEE Control Systems Magazine*, 33(5), 58–67.

Le Maître, O. and Knio, O. (2010). *Spectral Methods for Uncertainty Quantification with Applications to Computational Fluid Dynamics*. Scientific Computation. Springer Science+Business Media, Dordrecht Heidelberg London New York.

Marelli, S. and Sudret, B. (2014). *Vulnerability, Uncertainty, and Risk*, chapter UQLab: A Framework for Uncertainty Quantification in Matlab, 2554–2563.

Mühlpfordt, T., Faulwasser, T., and Hagenmeyer, V. (2018). A generalized framework for chance-constrained optimal power flow. *Sustainable Energy, Grids and Networks*, 16, 231–242.

Mühlpfordt, T., Paulson, J., Braatz, R., and Findeisen, R. (2016). Output feedback model predictive control with probabilistic uncertainties for linear systems. In *American Control Conference (ACC)*, 2035–2040.

Mühlpfordt, T., Roald, L., Hagenmeyer, V., Faulwasser, T., and Misra, S. (2019). Chance-constrained AC optimal power flow – A polynomial chaos approach. *IEEE Transactions on Power Systems*, 34(6), 4806–4816.

Mühlpfordt, T., Zahn, F., Becker, F., Faulwasser, T., and Hagenmeyer, V. (2019). github.com/timueh/PolyChaos.jl: v0.2.2.

Paulson, J., Buehler, E., and Mesbah, A. (2017). Arbitrary polynomial chaos for uncertainty propagation of correlated random variables in dynamic systems. *IFAC-PapersOnLine*, 50(1), 3548–3553. 20th IFAC World Congress.

Paulson, J., Streif, S., and Mesbah, A. (2015). Stability for receding-horizon stochastic model predictive control. In *American Control Conference (ACC)*, 937–943.

Rackauckas, C. and Nie, Q. (2017). DifferentialEquations.jl – A performant and feature-rich ecosystem for solving differential equations in Julia. *J. of Open Research Software*, 5(15), 1–10.

Scokaert, P. and Rawlings, J. (1998). Constrained linear quadratic regulation. *IEEE Trans. on Aut. Contr.*, 43(8), 1163–1169.

Streif, S., Kim, K., Rumschinski, P., Kishida, M., Shen, D., Findeisen, R., and Braatz, R. (2013). Robustness analysis, prediction and estimation for uncertain biochemical networks. *IFAC Proceedings Volumes*, 46(32), 1–20. 10th IFAC International Symposium on Dynamics and Control of Process Systems.

Sullivan, T. (2015). *Introduction to Uncertainty Quantification*, volume 63. Springer International Publishing, CH, 1st edition.

Wiener, N. (1938). The homogeneous chaos. *American Journal of Mathematics*, 60(4), 897–936.

Xiu, D. (2010). *Numerical Methods for Stochastic Computations*. Princeton University Press, Princeton, New Jersey.

Xiu, D. and Karniadakis, G.E. (2002). The Wiener-Askey polynomial chaos for stochastic differential equations. *SIAM Journal on Scientific Computing*, 24(2), 619–644.