UNCERTAINTY QUANTIFICATION FOR AUTONOMOUS ODES VIA POLYNOMIAL CHAOS EXPANSIONS

DIOGO CORREIA NETTO[*]

Abstract. We survey a method for computing global sensitivity indices in systems described by autonomous ordinary differential equations parameterized by random variables. The method of Polynomial Chaos Expansions (PCE) is based on projecting the differential equation's solution onto families of orthogonal polynomials of the random parameters. We benchmark PCE's accuracy using a linear autonomous ODE (whose closed-form solution is known) and compare PCE's performance and accuracy with those of a Monte-Carlo method in the SEIR contagion model and in the Lotka-Volterra equations. Finally, we analyze PCE's performance as a function of the problem's dimension, discussing how the curse of dimensionality for tensor product approximations imposes a barrier for using PCE in high-dimensional problems, and we propose an alternative based on limiting the total degree of the expansion.

1. Introduction.

With the recent advances in the field of Scientific Machine Learning and the improvement of computational tools such as differential equation solvers, nonlinear optimizers and numerical integrators, the automatic discovery of mechanistic models through data analysis is becoming an increasingly popular technique, with applications ranging from Fluid Mechanics [1] to Chemical Kinetics [2].

Even though the governing equations of certain physical phenomena can be automatically and accurately discovered, methods of uncertainty quantification are still necessary for studying how each of the uncertain parameters in the model contribute to the output variance, and in a more practical sense, to allow researchers to identify and target which inputs have higher contribution to the uncertainty of an observed quantity, allowing them to systematically improve the quality of measured data.

Simple approaches such as computing the variation of the quantity of interest with respect to a single input often fail to capture interactions between the uncertainty effects of groups of variables, and may also need some form of ad hoc consideration (such as taking a derivative-based quantity at the average of the parameters) to enable their computation.

Global Sensitivity Analysis is often the desired solution for such problems. It can be informally defined as the process of quantifying the output uncertainty in a model parameterized by random variables, and also the process of partitioning the contributions to the variance of a quantity of interest among its unknown parameters, while also taking into account potential interactions between them.

Although Monte-Carlo methods are generally chosen to perform Global Sensitivity Analysis due to their extensive applicability, relatively straightforward implementation and easiness of parallelization, they may fail to explore some inherent structure in the problem and, not surprisingly, may have a higher computational cost and lower accuracy than methods that use the mathematical form of the problem more effectively.

The discussion presented in the previous paragraph motivates us to analyze in this paper the use of Polynomial Chaos Expansions (PCE) to perform Global Sensitivity Analysis in certain dynamical systems described by autonomous ordinary differential equations (ODEs), exploring the time-independence in the dynamics equations to produce more computationally efficient and more accurate estimates than Monte-Carlo based approaches in cases where the time-evolution of the so-called sensitivity

[*]Massachusetts Institute of Technology, Cambridge, MA (netto@mit.edu)

indices (which quantify the contribution of each parameter to the variance of the quantity of interest) is desired.

The paper is divided in the following manner: in section 2, we introduce the Law of Total Variance and define the sensitivity indices, a way to quantify the effect of the uncertainty of a parameter into the uncertainty of a quantity of interest. In section 3 we discuss one of the Monte-Carlo approaches for Global Sensitivity Analysis. In section 4, we outline the mathematical results which motivate the use of PCE for not only approximating the solution of a dynamical system described by an autonomous random ODE but also for reducing the problem of uncertainty quantification into the task of solving another autonomous and deterministic ODE. In section 5, we first use a linear first-order ODE, a problem with a closed-form solution, to benchmark the accuracy of PCE. We proceed by using the the SEIR contagion model and the Lotka-Volterra equations from evolutionary ecology to provide additional benchmarks of accuracy and performance. We then compare our implementation (in Julia) with an existing Julia Package for PCE, discussing how performance engineering techniques such as the extensive use of mutation to eliminate unnecessary heap-allocations produce significant performance improvements in terms of memory usage and runtime. Finally, we finish section 5 discussing the limitation of PCE for high-dimensional problems with a large number of random variables and briefly analyze how truncating the total degree of the expansion and exploring the sparsity of the resulting indices can help to alleviate the so-called curse of dimensionality. Conclusion and acknowledgements then follow in 6 and 7, respectively.

2. Law of Total Variance and Sensitivity Indices.

This section presents the Law of Total Variance, a result which enables one to calculate the variance of a function of multiple random variables by conditioning it on a subset of variables. This law led us to introduce the notion of the variance explained by a set of random variables, from which the first-order and total sensitivity indices are defined.

From Explained Variance to Sensitivity Indices. Let's consider random variables $X_1, X_2, \dots, X_n$, with a joint distribution $p_X(x_1, x_2, \dots, x_n)$. Let's assume that the distribution of X_i, $p_i(x_i)$ is known and that the distribution

$$p_{X\sim i}(x_1, x_2, \dots, x_{i-1}, x_{i+1}, x_n) \coloneqq \int_{\mathcal{D}_i} p_X(x_1, x_2, \dots, x_n)\mathrm{d}x_i,$$

can be computed, where $X_{\sim i} = (X_1, X_2, \dots, X_{i-1}, X_{i+1}, X_n)$ (the integration extends over the support $\mathcal{D}_i$ of X_i).

Under the assumptions above, a simple application of the Law of Total Expectation gives us the Law of Total Variance

$$\mathrm{Var}(Y) = \mathrm{Var}_{X_i}(\mathrm{E}_{X\sim i}(Y|X_i)) + \mathrm{E}_{X_i}(\mathrm{Var}_{X\sim i}(Y|X_i)).$$

Furthermore, we denote

$$\mathrm{Var}_{X_i}(\mathrm{E}_{X\sim i}(Y|X_i))$$

as the variance explained by X_i.

The first-order sensitivity index S_i is defined as the fraction of the output variance explained by X_i only. That's it,

$$S_i \coloneqq \frac{\mathrm{Var}_{X_i}(\mathrm{E}_{X\sim i}(Y|X_i))}{\mathrm{Var}(Y)}.$$

Additionally, the total sensitivity index S_{T_i} is defined as the complement of the variance explained by $X_{\sim i}$. That's it, defining $S_{\sim i}$

$$S_{\sim i} \coloneqq \frac{\mathrm{Var}_{X_{\sim i}}(\mathrm{E}_{X_i}(Y|X_{\sim i}))}{\mathrm{Var}(Y)}$$

as the variance explained by $X_{\sim i}$, we write

$$S_{T_i} \coloneqq 1 - S_{\sim i} = 1 - \frac{\mathrm{Var}_{X_{\sim i}}(\mathrm{E}_{X_i}(Y|X_{\sim i}))}{\mathrm{Var}(Y)}.$$

Alternatively, from the Law of Total Variance

$$S_{T_i} = 1 - S_{\sim i} = \frac{\mathrm{E}_{X_{\sim i}}(\mathrm{Var}_{X_i}(Y|X_{\sim i}))}{\mathrm{Var}(Y)}.$$

For the case of independent random variables, the so-called High-Dimensional Model Representation (HDMR) introduced below provides an intuition for the nomenclature and meaning of these indices.

For every function $Y = f(X_1, X_2, \dots, X_n)$, where $X_1, X_2, \dots X_n$ are now assumed to be independent, there exists a decomposition of the form

$$f(X_1, X_2, \dots, X_n) = g_0 + \sum_i g_i(X_i) + \sum_{i<j} g_{i,j}(X_i, X_j) + \dots + g_{1,2,\dots,n}(X_1, X_2, \dots, X_n),$$

where for $\{i_1, i_2, \dots, i_{n_1}\} \neq \{j_1, j_2, \dots j_{n_2}\}$, we have

$$\mathrm{Cov}(g_{i_1,i_2,\dots,i_{n_1}}(\cdot), g_{j_1,j_2,\dots j_{n_2}}(\cdot)) = 0.$$

The random variables dependency is clear from context and has been omitted for conciseness.

By setting

$$g_{1,2\dots,n}(\cdot) = f(\cdot),$$

and the remaining terms to be zero, it becomes clear that a trivial case of such decomposition exists. However, as shown in [4], for cases of low coupling between the parameters, the decomposition may be restricted to lower order terms.

From the fact that terms with different sets of indices are uncorrelated, we conclude that the variance of Y may be written as

$$\mathrm{Var}(Y) = \sum_i \mathrm{Var}(g_i(\cdot)) + \sum_{i<j} \mathrm{Var}(g_{i,j}(\cdot)) + \dots \mathrm{Var}(g_{1,2,\dots,n}(\cdot)),$$

It follows that the first-order sensitivity indices are given by

$$S_i = \frac{\mathrm{Var}(g_i(\cdot))}{\mathrm{Var}(Y)}.$$

Additionally, S_{T_i} collects the variance contributions from all terms in the expansion which contain the ith index.

It's immediate that

$$\sum_i S_i \leq 1,$$

with equality for functions with no 2nd or higher order terms. Finally, due to overlapping variance contributions in the expansion,

$$\sum_i S_{T_i} \geq 1,$$

with equality holding in the same case as above.

3. Monte-Carlo Method.

The following section introduces a Monte-Carlo estimator described in [3] for first-order and total sensitivity indices.

Such method's accuracy and performance will be compared with those of PCE in the next sections.

A Monte-Carlo Estimator for Sensitivity Indices. Let $X_1, X_2, \ldots, X_n$ be independent real-valued random variables and $Y = f(X_1, X_2, \ldots, X_n) \in \mathbb{R}$ be a real-valued random variable obtained via the application of a deterministic function $f(\cdot)$.

The Monte-Carlo estimators for the first-order and total sensitivity indices can be obtained by following the procedure below:

1. Sample M random vectors $R_A^{(i)} = (X_1^{(i)}, X_2^{(i)}, \ldots, X_n^{(i)}), 1 \leq i \leq M$ by sampling each coordinate independently. Construct a matrix

$$A = \begin{pmatrix} \text{---} R_A^{(1)} \text{---} \\ \vdots \\ \text{---} R_A^{(M)} \text{---} \end{pmatrix}.$$

   Repeat the process constructing a matrix

$$B = \begin{pmatrix} \text{---} R_B^{(1)} \text{---} \\ \vdots \\ \text{---} R_B^{(M)} \text{---} \end{pmatrix}.$$

2. Let $A_B^{(i)}, 1 \leq i \leq n$ be the matrix obtained by inserting the ith column of B into A. Denote $Y_A = f(A), Y_B = f(B), Y_1 = f(A_B^{(1)}), Y_2 = f(A_B^{(2)}), \ldots, Y_n = f(A_B^{(n)})$ as the $M \times 1$ vectors obtained via a row-wise application of f.
3. Compute $f_0 = \frac{Y_A^T \mathbf{1}}{M}$, where $\mathbf{1}$ is the $M \times 1$ vector of all ones.

4. The first-order sensitivity indices can be estimated by

$$\hat{S}_i = \frac{\frac{Y_B^T(Y_i - Y_A)}{M}}{\frac{\|Y_A\|^2}{M} - f_0^2}.$$

and for the total sensitivity indices we use

$$\hat{S}_{T_i} = \frac{\frac{\|Y_A - Y_i\|^2}{2M}}{\frac{\|Y_A\|^2}{M} - f_0^2}.$$

The convergence properties of such algorithm are discussed in [7].

4. Function Approximation via Polynomial Chaos Expansions.

This section discusses fundamental results of functional analysis from which the theory of function approximation via PCE is derived.

Background. Throughout this paper, we will be discussing the case of dynamical systems parameterized by independent random variables with known distributions.

With this constraint in mind, let $X_1, X_2, \dots, X_n$ be independent random variables (not necessarily identically distributed) with joint probability density function

$$p_X(x_1, x_2, \dots, x_n) = p_1(x_1)p_2(x_2)\dots p_n(x_n).$$

Given real-valued functions $Y(x_1, x_2, \dots, x_n)$ and $Z(x_1, x_2, \dots, x_n)$, we define the inner product with respect to this density function as

$$\langle Y(\cdot), Z(\cdot)\rangle_{p_X} = \int Y(x_1, x_2, \dots, x_n)Z(x_1, x_2, \dots, x_n)p_X(x_1, x_2, \dots, x_n)\mathrm{d}x_1\mathrm{d}x_2\dots\mathrm{d}x_n.$$

Furthermore, we define $L^2_{p_X}$ as the space of real-valued functions $Y(x_1, x_2, \dots, x_n)$ such that

$$\|Y(\cdot)\|^2_{p_X} := \langle Y, Y\rangle_{p_X} < \infty.$$

It's possible to show that $\|Y(\cdot)\|^2_{p_X}$ defines a norm in $L^2_{p_X}$ [4], which is a Hilbert space.

We say that two functions Y, Z are orthogonal with respect to p_X if

$$\langle Y(\cdot), Z(\cdot)\rangle_{p_X} = 0.$$

For the univariate case, given a random variable $X \in \mathbb{R}$ and a probability density function $p(\cdot)$ our initial goal is to find a family of polynomials $\Psi = \{\Psi_0, \Psi_1, \dots\}$ pairwise orthogonal with respect to p such that

$$\operatorname{span}(\Psi) = L^2_p.$$

It's clear that a orthogonal Ψ family satisfying the condition above is also a basis of L^2_p, so that an expansion in terms of polynomials in this family is unique.

As an example, the uniform distribution on $[-1,1]$ is associated with the orthonormal family of Legendre polynomials and the unit univariate normal distribution yields the orthonormal family of (probabilist's, as opposed to physicist's) Hermite polynomials [4].

Having such family of orthogonal polynomials allows us to approximate a real-valued function $f(x)$ by truncating the expansion in terms of this basis, as we show below.

Theorem 4.1. *If $f(\cdot) \in L_p^2$ is such that*

$$f(x) = \sum_{i=0}^{\infty} C_i \Psi_i(x),$$

then

$$\hat{f}_n(x) = \sum_{i=0}^{n-1} C_i \Psi_i(x) \in \underset{\hat{f}\in \text{span}\{\Psi_0,\Psi_1,\dots\,\Psi_{n-1}\}}{\text{argmin}} \|f(x) - \hat{f}(x)\|_p^2.$$

That's it, the truncated expansion of $f(\cdot)$ is also the best approximation in $\text{span}\{\Psi_0,\dots\,\Psi_{n-1}\}$ *under the mean-squared-error.*

Proof. For any $\hat{f}(x) = \sum_{i=0}^{n-1} \hat{C}_i \Psi_i(x) \in \text{span}\{\Psi_0,\Psi_1,\dots\,\Psi_{n-1}\}$

$$\|f(x) - \hat{f}(x)\|_p^2 = \|\sum_{i=0}^{n-1}(C_i - \hat{C}_i)\Psi_i(x) + \sum_{i=n}^{\infty} C_i\Psi_i(x)\|_p^2.$$

From the triangle inequality,

□

$$\|f(x) - \hat{f}(x)\|_p^2 \le \sum_{i=0}^{n-1}(C_i - \hat{C}_i)^2\|\Psi_i(x)\|_p^2 + \|\sum_{i=n}^{\infty} C_i\Psi_i(x)\|_p^2,$$

which is clearly minimized for $\hat{C}_i = C_i$, $0 \le i \le n-1$, since the norm is non-negative.

Additionally, the orthogonality of the polynomials yields

$$C_i = \frac{\langle f(\cdot), \Psi_i(\cdot)\rangle}{\langle \Psi_i(\cdot), \Psi_i(\cdot)\rangle}.$$

For the multivariate case, a fundamental result from functional analysis [4] is that if $\Psi^{(1)}, \Psi^{(2)}, \dots, \Psi^{(n)}$ are families of orthogonal polynomials with respect to the densities

$$p_1(\cdot), p_2(\cdot), \dots p_n(\cdot)$$

of the independent random variables $X_1, X_2, \dots, X_n$, and such families also satisfy

$$\text{span}(\Psi^{(i)}) = L^2_{p_i}, 1 \leq i \leq n$$

then the tensor product product family defined as

$$\Psi^{(1:n)} = \Psi^{(1)} \otimes \Psi^{(2)} \otimes \ldots \otimes \Psi^{(n)} := \text{span}\,\{\prod_{i=1}^{n} \Psi^{(i)}_{\alpha_i}(x_i) | \vec{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_n) \in \mathbb{N}^n\}$$

is such that for the joint probability density function

$$p_X(x_1, x_2, \ldots, x_n) = p_1(x_1)p_2(x_2)\ldots p_n(x_n),$$

we have

$$\text{span}(\Psi^{(1:n)}) = L^2_{p_X}.$$

These function approximation results can also be extended for dynamical systems parameterized by random variables. Let's consider the autonomous ODE

$$\frac{\mathrm{d}Y(t; X_1, X_2, \ldots, X_n)}{\mathrm{d}t} = f(Y, X_1, X_2, \ldots, X_n),$$

with $Y \in \mathbb{R}^d$ and $X_1, X_2, \ldots, X_n$ real-valued independent random variables with known distributions (not necessarily all equal) and families of orthogonal polynomials $\Psi^{(1)}, \Psi^{(2)}, \ldots, \Psi^{(n)}$.

As proved in [4], $Y(t; X_1, X_2, \ldots, X_n)$ can be written in the functional form

$$Y(t; X) = \sum_{\vec{\alpha} \in \mathbb{N}^n} C_{\vec{\alpha}}(t) \Psi_{\vec{\alpha}}(X),$$

where $C_{\vec{\alpha}}(t) \in \mathbb{R}^d$ and

$$\Psi_{\vec{\alpha}}(X) = \prod_{i=1}^{n} \Psi^{(i)}_{\alpha_i}(X_i).$$

In order to obtain a multivariate function approximation, we may truncate our expansion to the first m polynomials in each dimension, obtaining

$$Y(t; X) \approx \hat{Y}(t; X) = \sum_{\vec{\alpha} \in [0:m-1]^n} C_{\vec{\alpha}}(t) \Psi_{\vec{\alpha}}(X),$$

where $[0 : m - 1] = \{0, 1, \ldots, m - 1\}$. Substituting the approximation into the ODE yields

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\sum_{\vec{\alpha}\in[0:m-1]^n} C_{\vec{\alpha}}(t)\Psi_{\vec{\alpha}}(X)\right) = f(\hat{Y}(t;X), X_1, X_2, \dots, X_n).$$

Due to orthogonality,

$$\frac{\mathrm{d}C_{\vec{\alpha}}}{\mathrm{d}t} = \frac{\left\langle f\left(\sum_{\vec{\alpha}\in[0:m-1]^n} C_{\vec{\alpha}}(t)\Psi_{\vec{\alpha}}(\cdot), \cdot\right), \Psi_{\vec{\alpha}}(\cdot)\right\rangle_{p_X}}{\|\Psi_{\vec{\alpha}}(\cdot)\|_{p_X}^2},$$

and we transform the random autonomous ODE into a deterministic autonomous ODE.

Since the orthogonal polynomials induced by the univariate distribution functions are uncorrelated [4], we may approximate the covariance matrix of the solution as

$$\mathrm{Cov}(Y(t;X)) \approx \sum_{\vec{\alpha}\in[0:m-1]^n} C_{\vec{\alpha}}(t)C_{\vec{\alpha}}(t)^T \mathrm{Var}(\Psi_{\vec{\alpha}}(X)),$$

which, for instance, allows us to estimate the total sensitivity index of any $X_i, 1 \leq i \leq n$ on any of the $Y_j(t;X), 1 \leq j \leq d$ as as a function of time, since the contributions of X_i to $\mathrm{Var}(Y_j(t;X)) = \mathrm{Cov}(Y(t;X))_{jj}$ can be easily computed using orthogonality.

It should be noted that in order to integrate a linear combination of the first m orthogonal polynomials of some family, we require m points in its Gaussian quadrature [4], so that for typical approximation orders ($m \leq 10$) the inner product computation in the ODE for $C_{\vec{\alpha}}$ is computationally feasible.

5. PCE Showcase.

In this section, we provide benchmarks for our PCE Global Sensitivity Analysis implementation. We start by analyzing in 5.1 PCE's accuracy for a problem with closed-form solution. In 5.2 and 5.3 we compare PCE's performance with those of the Monte-Carlo estimators from 3 for the case where the time-evolution of the sensitivity indices is desired. We then proceed by comparing in 5.4 our implementation's performance with PolyChaos.jl. Finally, in 5.5 we present some of PCE's limitations due to the curse of dimensionality.

5.1. Linear Autonomous ODEs.

Let's start by constructing a random autonomous ODE with closed-form solution and closed-form output variance to be used in our benchmarks. Initially, consider the linear ODE

$$\frac{\mathrm{d}Y}{\mathrm{d}t} = AY,$$

where $Y \in \mathbb{R}^n$ and $A \in \mathbb{R}^{n\times n}$ is a diagonalizable matrix given by

$$A = Q^T \operatorname{diag}(\lambda_1, \lambda_2, \dots, \lambda_n)Q,$$

with $Q \in \mathbb{R}^{n\times n}$ orthonormal. Setting $Z = QY$ allows us to derive the closed form solution

$$Z(t) = \operatorname{diag}(e^{\lambda_1 t}, e^{\lambda_2 t}, \ldots, e^{\lambda_n t}) Z(0),$$

from which $Y(t) = Q^T Z(t)$ can be recovered. We can transform the linear ODE above into a random autonomous ODE with the desired closed-form solution properties by considering a modified matrix A' obtained by introducing additive Gaussian noise to the diagonal matrix of eigenvalues and conjugating it with respect to Q.

More specifically, consider the modified random ODE

$$\frac{\mathrm{d}Y}{\mathrm{d}t} = Q^T \operatorname{diag}(\lambda_1 + \varepsilon_1 X_1, \lambda_2 + \varepsilon_2 X_2, \ldots, \lambda_n + \varepsilon_n X_n) QY,$$

with $X_1, X_2, \ldots, X_n \sim \mathcal{N}(0, \sigma^2)$ independent. In this case, $Z = QY$ may be written as

$$Z(t) = \operatorname{diag}(e^{\tilde{X}_1(t)}, e^{\tilde{X}_2(t)}, \ldots, e^{\tilde{X}_n(t)}) Z(0),$$

with $\tilde{X}_i(t) \sim \mathcal{N}(\lambda_i t, (\sigma \varepsilon_i t)^2)$. It follows that each component $Z_i(t)$ is distributed according to a multiple of a log-normal random variable and has, therefore, a closed-form variance. Since $Y(t)$ is recovered from $Z(t)$ via a linear transformation, we conclude that the variances of each component of Y can be computed in closed-form.

With these results in mind, we perform the following experiment:

1. Sample a random matrix $P = QR \in \mathbb{R}^{n \times n}$ (with Q orthonormal and R right-triangular) whose elements are distributed according to a unit normal.
2. Construct an error vector $\vec{\varepsilon} = (\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_n)$, whose elements are sampled from $\mathcal{U}(l_1, r_1)$.
3. Construct an eigenvalue vector $\vec{\lambda} = (\lambda_1, \lambda_2, \ldots, \lambda_n)$, whose elements are sampled from $\mathcal{U}(l_2, r_2)$, with $l_2 < r_2 < 0$ to guarantee the stability of the ODE solution.
4. Apply the tensor-product decomposition described in 4 to the ODE

$$\frac{\mathrm{d}Y(t; X)}{\mathrm{d}t} = Q^T \operatorname{diag}(\lambda_1 + \varepsilon_1 X_1, \lambda_2 + \varepsilon_2 X_2, \ldots, \lambda_n + \varepsilon_n X_n) QY(t; X),$$

   with $X_i \sim \mathcal{N}(0, \frac{1}{2})$ (so that the corresponding orthogonal polynomial is the Physicist's Hermite polynomial, whose quadrature has already been implemented in FastGaussQuadrature.jl [6]).

We have chosen $n = 4, l_1 = 0.05, r_1 = 0.1, l_2 = -0.2, r_2 = -0.1$ in our implementation. The initial value for Y was randomly sampled from a normal distribution.

The evolution of the absolute error in the standard deviation of each component is shown below:

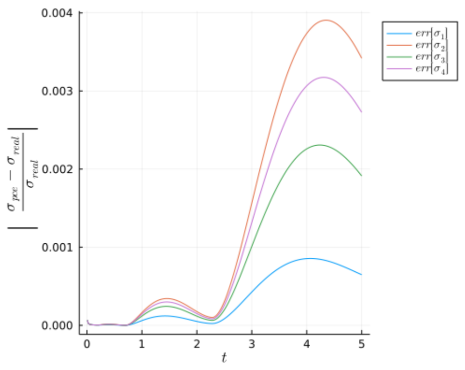


Fig. 1: Evolution of the absolute error in σ_Y.

The average error of $\sim 0.2\%$ and maximum error of $\sim 0.4\%$ over the time interval under consideration showcase the method's accuracy for the linear ODE described above.

5.2. SEIR Contagion Model. The SEIR model [8] is a set of non-linear differential equations that describes the propagation of a disease in a population where individuals are in one of the states S (succetible), E (exposed), I (infectious) and R (recovered) and the only possible transitions between states are $S \to E, E \to I, I \to R$.

Let's denote s, e, i, r as the fraction of the population in the states S, E, I, R, respectively (where $s + e + i + r = 1$).

The dynamics of the model is given by

$$\frac{\mathrm{d}s(t;X)}{\mathrm{d}t} = -\beta(X)s(t;X)i(t;X)$$

$$\frac{\mathrm{d}e(t;X)}{\mathrm{d}t} = \beta(X)s(t;X)i(t;X) - \sigma(X)e(t;X)$$

$$\frac{\mathrm{d}i(t;X)}{\mathrm{d}t} = \sigma(X)e(t;X) - \gamma(X)i(t;X)$$

$$\frac{\mathrm{d}r(t;X)}{\mathrm{d}t} = \gamma(X)i(t;X)$$

where X is the vector of random variables that introduces uncertainty in the parameters and $\beta(X), \sigma(X), \gamma(X)$ are all positive parameters.

To analyze the accuracy and performance of PCE compared to the Monte-Carlo method from 3, we perform the following benchmark:

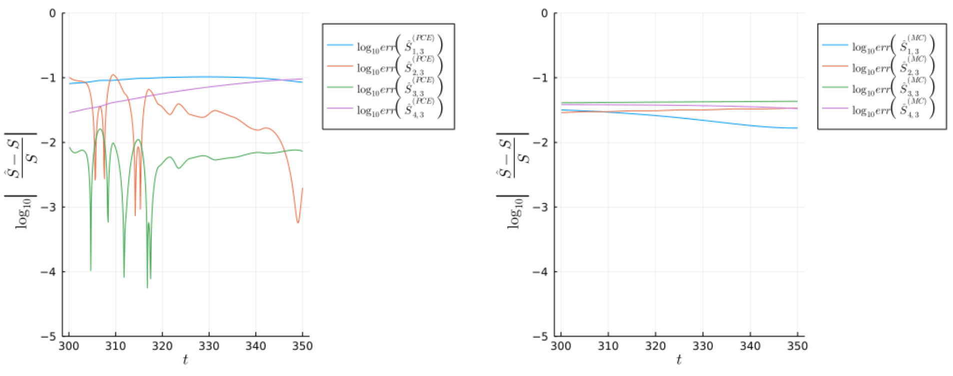
1. Start with a vector $(\beta_0, \sigma_0, \gamma_0)$, which will be the mean of $(\beta(X), \sigma(X), \gamma(X))$.

2. Construct an error vector $\vec{\varepsilon} = (\varepsilon_1, \varepsilon_2, \varepsilon_3)$, whose elements are sampled from $\mathcal{U}(l_3, r_3)$.
3. Apply the tensor-product decomposition described in 4 to the ODE obtained by replacing $\beta(X) = \beta_0 + \varepsilon_1 X_1, \sigma(X) = \sigma_0 + \varepsilon_2 X_2, \gamma(X) = \gamma_0 + \varepsilon_3 X_3$ into the system's dynamics, with $X_i \sim \mathcal{N}(0, \frac{1}{2}), 1 \leq i \leq 3$ (for the same reason as in 5.1).

Following [8], we have chosen $\beta_0 = \frac{1}{18}, \sigma_0 = \frac{1}{5.2}, \gamma_0 = 3.0, l_3 = 0.0005, r_3 = 0.001$ and $s(0) = 1 - 5 \times 10^{-7}, e(0) = 4 \times 10^{-7}, i(0) = 10^{-7}, r(0) = 0$ in our implementation.

The absolute error in the sensitivity indices were compared with those of a Monte-Carlo method with $1,000$ samples. We have taken the results from a Monte-Carlo method with $20,000$ samples as our baseline (that's it, we considered the results of this execution as the true values, from which errors were computed).

The evolution in the log of absolute errors of the sensitivity indices for each method are shown in figure 2 (where $\hat{S}_{1,3}$ denotes the estimated sensitivity index of the third random variable in s, $\hat{S}_{2,3}$ denotes the sensitivity index of the same random variable as measured in e, etc.)



(a) PCE. (b) Monte-Carlo.

Fig. 2: Evolution in the log of absolute error in the total sensitivity indices for each method in the SEIR model (lower is better).

It should be noted that, since the fraction of infected individuals remains close to 0 in the first ~ 200 time periods, the denominator in the method presented in 3 is close to 0 and causes the Monte-Carlo method to diverge in the first time periods. Because of this, only samples for $t \geq 300$ were taken for comparison.

The runtime and memory usage data are summarized in table 1.

Method	Total Runtime (s)	Total Memory Allocation (MB)
PCE	0.1	9.5
MC	2.3	2600

Table 1: Runtime and memory usage for both methods in the SEIR model.

We observed a better accuracy in PCE (as measured in average absolute error in the global sensitivity indices) for most dimensions and also a significant improvement in performance (both in runtime and memory usage) from PCE.

5.3. Lotka-Volterra Equations. The Lotka-Volterra equations [9], also known as predator-prey equations, are used to describe the population dynamics in biological systems in which two species interact (one being a prey and the other being a predator). The non-linear ODE is given by

$$\frac{\mathrm{d}w(t;X)}{\mathrm{d}t} = \alpha(X)w(t;X) - \beta(X)w(t;X)v(t;X)$$

$$\frac{\mathrm{d}v(t;X)}{\mathrm{d}t} = \delta(X)w(t;X)v(t;X) - \gamma(X)v(t;X)$$

where w is the number of preys, v is the number of predators, X is the vector of random variables that introduces uncertainty in the parameters and $\alpha(X), \beta(X), \delta(X), \gamma(X)$ are all positive parameters.

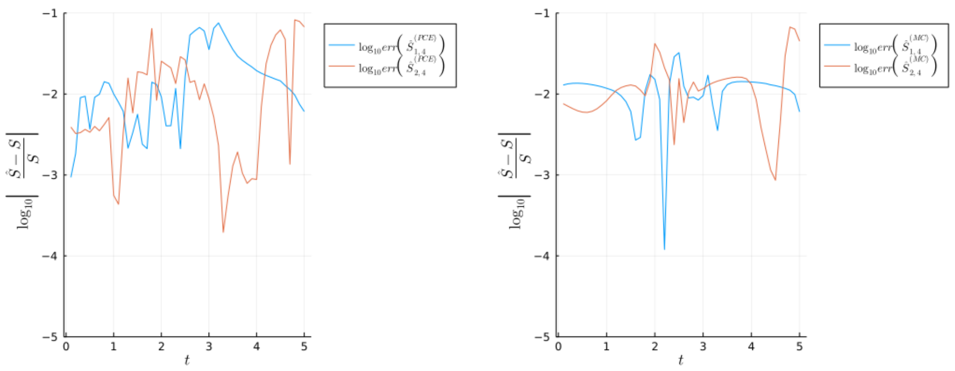
To analyze the accuracy and performance of PCE compared to the Monte-Carlo method from 3, we perform the following benchmark:

1. Start with a vector $(\alpha_0, \beta_0, \delta_0, \gamma_0)$, which will be the mean of $(\alpha(X), \beta(X), \delta(X), \gamma(X))$.
2. Construct an error vector $\vec{\varepsilon} = (\varepsilon_1, \varepsilon_2, \varepsilon_3, \varepsilon_4)$, whose elements are sampled from $\mathcal{U}(l_4, r_4)$.
3. Apply the tensor-product decomposition described in 4 to the ODE obtained by replacing $\alpha(X) = \alpha_0 + \varepsilon_1 X_1, \beta(X) = \beta_0 + \varepsilon_2 X_2, \delta(X) = \delta_0 + \varepsilon_3 X_3, \gamma(X) = \gamma_0 + \varepsilon_4 X_4$ into the system's dynamics, with $X_i \sim \mathcal{N}(0, \frac{1}{2}), 1 \leq i \leq 4$ (for the same reason as in 5.1).

We have chosen $\alpha_0 = 1.5, \beta_0 = 1.0, \gamma_0 = 3.0, \delta_0 = 1.0, l_4 = 0.05, r_4 = 0.1$ and $w(0) = v(0) = 1.0$ in our implementation.

The absolute error in the sensitivity indices were compared with those of a Monte-Carlo method with $10,000$ samples. We have taken the results from a Monte-Carlo method with $100,000$ samples as our baseline (that's it, we considered the results of this execution as the true values, from which errors were computed).

The evolution in the log of absolute errors of the sensitivity indices for each method are shown in figure 3 (where $\hat{S}_{1,4}$ denotes the estimated sensitivity index of the fourth random variable in w and $\hat{S}_{2,4}$ denotes the sensitivity index of the same random variable as measured in v).



(a) PCE. (b) Monte-Carlo.

Fig. 3: Evolution in the log of absolute error in the total sensitivity indices for each method in the Lotka-Volterra equations (lower is better).

The runtime and memory usage data are summarized in table 2.

Method	Total Runtime (s)	Total Memory Allocation (MB)
PCE	0.94	2.8
MC	4.77	3300

Table 2: Runtime and memory usage for both methods in the Lotka-Volterra equations

We observed a marginally better accuracy in PCE (as measured in average absolute error in the global sensitivity indices) and also a significant improvement in performance (both in runtime and memory usage) from PCE.

5.4. Comparison with PolyChaos.jl. PolyChaos.jl [10] is one of the existing packages in Julia, which among other functionalities, provides PCE support to compute uncertainties in the solution of an autonomous ODE.

As in example provided in [11], we solve the simple linear ODE

$$\frac{\mathrm{d}w(t;X)}{\mathrm{d}t} = Xw,$$

with $X \sim \mathcal{N}(\mu, \sigma^2)$, where for purposes of implementation $\mu = -0.5, \sigma = 0.05$.

In our comparison, both packages showed similar accuracy in the computation of variances. For this same task, however, we observed a significant performance difference in our implementation compared to PolyChaos.jl, as summarized in table 3.

Implementation	Total Runtime (s)	Total Memory Allocation (MB)
Current	0.003	1.6
PolyChaos.jl	0.044	57.1

Table 3: Runtime and memory usage for the current implementation and PolyChaos.jl

This performance difference is mainly due to the reduced number of heap allocations of small arrays in our implementation caused by the extensive mutation of cache arrays. Furthermore, the type-stability in our implementation also contributed to performance, since type-instability in Julia can often produce non-specialized methods and thus unnecessary heap-allocations.

5.5. Curse of Dimensionality: How far can PCE go?. As one may note, for a fixed number of polynomials in each dimension, the size of PCE's solution arrays grows exponentially with the number of random variables. Because of this, an exponential or faster than exponential growth in runtime and memory usage is expected, which may impose a barrier for using PCE in high-dimensional problems with a large number of random variables.

To confirm this hypothesis, timing and memory usage data for the linear ODE setup in 5.1 with varying dimension (and number of random variables) are summarized in table 4.

n	Total Runtime (s)	Total Memory Allocation (MB)
1	0.001	1.7
2	0.004	3.0
3	0.017	9.8
4	0.77	49.6
5	12.8	550.3

Table 4: Runtime and memory usage as a function of the problem dimension (and number of random variables) for a linear ODE.

We observed that for $n \geq 7$ random variables the slowdown due to the large memory usage makes the method infeasible, since (among other reasons) arrays in the solution may no longer fit in cache and the number cache misses grows significantly.

Such problem may be alleviated by truncating the total degree (maximum among all terms of the sum of degrees of each variable) of the expansion. It should be noted that the size of PCE's solution arrays would have a polynomial growth in this case due to the sparsity in the indices which yield a sufficiently small total degree.

6. Conclusions. We presented a technique for reducing the task of uncertainty estimation in an autonomous ODE with random parameters to the task of solving another autonomous deterministic ODE. Our implementation of such technique was benchmarked for a linear first-order ODE with closed-form solution and compared with Monte-Carlo estimators for global sensitivity indices in the SEIR model and in the Lotka-Volterra equations. We compared our implementation with another Julia package, showing how a reduced number of heap-allocations due to re-use of cache arrays and due to type stability can produce significant performance enhancements. Finally, we analyzed the feasibility of using our implementation for high-dimensional problems with a large number of random variables, showing how cache misses due to large solution arrays may produce a significant slowdown in large problems and mentioned how total degree truncation in the expansion may help to alleviate this curse of dimensionality.

7. Acknowledgments. The current implementation relies on the Julia packages DifferentialEquations.jl and FastGaussQuadrature.jl developed by Rackauckas et al [5], [6]. The motivation for the topic of this paper came from a Summer UROP (Undergraduate Research Opportunities) project at MIT supervised by Chris Rackauckas on the use of tensor product expansions as universal approximators and from the MIT course 16.940 (Numerical Methods for Stochastic Modelling and Inference, taught by Youssef Marzouk) that I took as a listener in the Fall of 2020.

REFERENCES

[1] Chen Z. et al. Physics-informed learning of governing equations from scarce data, 2021. `https://arxiv.org/pdf/2005.03448.pdf`
[2] Yank W. et al. When machine learning meets multiscale modeling in chemical reactions, 2020. *https://aip.scitation.org/doi/10.1063/5.0015779*
[3] Variance-based sensitivity analysis. *https://en.wikipedia.org/wiki/Variance-based_sensitivity_analysis*
[4] Smith R. Uncertainty Quantification: Theory, Implementation and Applications. Society of Industrial and Applied Mathematics, 2013.
[5] Rackauckas C. et al. Differentialequations.jl–a performant and feature-rich ecosystem for solving differential equations in julia. Journal of Open Research Software, 2017.
[6] Rackauckas C. et al. FastGaussQuadrature.jl. *https://github.com/JuliaApproximation/FastGaussQuadrature.jl*
[7] Sobol I. Global sensitivity indices for nonlinear mathematical models and their Monte Carlo estimates. Mathematics and Computer Simulation, 2001.
[8] Perla et al. Modeling COVID 19 with Differential Equations. *https://julia.quantecon.org/continuous_time/seir_model.html*
[9] Blaszak T. Lotka-volterra equations. *https://web.mst.edu/~huwen/teaching_Predator_Prey_Tyler_Blaszak.pdf*
[10] Muhlpfordt et al. PolyChaos.jl. *https://timueh.github.io/PolyChaos.jl/stable/*
[11] Muhlpfordt et al. Galerkin-based Solution of Random Differential Equation. *https://timueh.github.io/PolyChaos.jl/stable/random_ode/*