COMPUTING STATISTICAL MOMENTS VIA TENSORIZATION OF POLYNOMIAL CHAOS EXPANSIONS

RAFAEL BALLESTER-RIPOLL*

Abstract. We present an algorithm for estimating higher-order statistical moments of multidimensional functions expressed as polynomial chaos expansions (PCE). The algorithm starts by decomposing the PCE into a low-rank tensor network using a combination of tensor-train and Tucker decompositions. It then efficiently calculates the desired moments in the compressed tensor domain, leveraging the highly linear structure of the network. Using three benchmark engineering functions, we demonstrate that our approach offers substantial speed improvements over alternative algorithms while maintaining a minimal and adjustable approximation error. Additionally, our method can calculate moments even when the input variable distribution is altered, incurring only a small additional computational cost and without requiring retraining of the regressor.

Key words. polynomial chaos expansions, statistical moments, tensor networks, surrogate modeling, tensor train decomposition, Tucker decomposition

AMS subject classifications. 65C20, 49Q12, 15A69, 65D15

1. Introduction.

Polynomial chaos expansions (PCE) constitute one of the most successful techniques for surrogate modeling in engineering, risk assessment, analysis of dynamical systems, and more. PCE interpolators are often better than Gaussian processes at capturing the global structure of a multidimensional function, especially at higher numbers of dimensions. They typically outperform neural networks when the available training data is scarce, as is often the case in expensive experimental or simulation processes. In contrast to support vector regressors or random forests, PCEs are exceptionally well suited for many sensitivity analysis tasks that can be accomplished efficiently by direct manipulation of the expansion's coefficients. Last, since they are simply a combination of sums and products of input variables, PCEs largely behave like white-box models and hence are amenable to interpretation.

Despite these attractive features, PCE expansions suffer from the *curse of dimensionality*: the amount of degrees of freedom (number of expansion weights) depends exponentially on the number of input variables considered. This is typically countered via sparsity constraints, which limit the number of non-zero weights without compromising the regressor's predictive power. As a bonus, since polynomial bases can be chosen to be orthogonal, sparsity also allows for efficient computation of dot products and L^2 norms of learned regressors, which is useful to obtain statistics, perform uncertainty propagation, dimensionality reduction, etc.

Still, this framework comes with several limitations. PCE orthogonality allows for the mean and variance to be efficiently obtained, but this convenience does not extend to higher-order moments (which are useful in sensitivity analysis, for example for generalizing the method of Sobol and investigating function extremes [20]). In all but special cases (e.g. a Hermite basis), computing the skewness and kurtosis requires evaluating the expectation of products between all triplets and quadruplets of basis elements, respectively [26]. There are other post-processing operations, such as multidimensional convolution, for which no efficient algorithm (i.e. free of the curse of dimensionality) is available when using a PCE representation. Last, orthogonality is tightly coupled with basis choice. Certain transformations on a fitted PCE model,

*School of Science and Technology, IE University, Madrid (rafael.ballester@ie.edu).

such as updating its domain or input variables' distribution, will destroy its basis orthogonality.

In order to mitigate those limitations, we propose to convert a learned PCE regressor into a *low-rank tensor decomposition* and, thus, gain access to a complementary family of numerical methods. Such decompositions generalize matrix decompositions and offer flexible ways to store and handle multidimensional functions using a moderate number of coefficients. Tensor decompositions are increasingly connected to many diverse areas of computer science and applied mathematics including certain neural network architectures [15, 6], probabilistic graphical models [22], or polynomial regression [8]. Having only emerged in the last few years, these kinds of symbiotic interdisciplinary bridges are still far from maturity.

More specifically, we learn PCE regression coefficients via a state-of-the-art robust fitting method, namely *least angle regression* (LARS) [9] combined with *hyperbolic truncation* [27], and to compress the resulting weights into a *tensor network* (TN), more specifically a *tensor train* decomposition (or TT, for short). The key enabling idea for our compression approach is that, after hyperbolic truncation, the surviving PCE coefficients form a well-structured region in a hypercube that can be captured compactly by a low-rank tensor. By casting a sparse robust model in such a tensor form, one opens the door to a suite of metamodel post-processing algorithms: sensitivity analysis, computation of statistical moments, global optimization, etc. [8, 4, 2]. TNs transform these tasks into multilinear algebra problems, which can be then solved via Kronecker products, tensor contractions, QR factorizations, eigen- or singular value decompositions, etc.

Our contribution is two-fold:

- To the best of our knowledge, our method is the first to cast a sparse PCE expansion into a TN of any kind. The compression algorithm to convert the former into the latter has bounded L^2 error.
- We give a new algorithm for computation of higher-order moments in the extended TT format, and we show that the proposed format is still usable if the distribution of input parameters is updated, with no need to learn it anew.

Remark. Some authors use *sparse* PCE to simply mean they truncate the set of PCE coefficients instead of considering the full product grid set (which grows exponentially with the number of dimensions N). By *sparse* we adhere to the usual linear algebra sense that the number C of non-zero expansion coefficients does not depend on N; see e.g. [27].

Notation. Throughout the paper, we use:

- Capital letters to denote integer constants: N is the number of features, also known as number of *tensor dimensions*; S is the size of the polynomial basis, C is the amount of non-zero PCE coefficients learned, etc.
- Lowercase letters for integer or real variables.
- Bold lowercase letters for vectors (e.g. $\boldsymbol{x}$).
- Bold uppercase letters for matrices (e.g. $\mathbf{U}$).
- Calligraphic letters for tensors and sets (e.g. $\mathcal{X}$ or $\mathcal{H}$).
- Element-wise operations on tensors, denoted for example as $\mathcal{A} + \mathcal{B}$ and $\mathcal{A}\mathcal{B}$ for the sum and product, respectively.
- One-based indexing for tensors. For instance, $\mathcal{X}[:, :, 1]$ is the first slice of a 3D tensor $\mathcal{X}$ along its third dimension.
- When slicing a TT core $\mathcal{G}$, for brevity we will use $\mathcal{G}[i]$ to represent $\mathcal{G}[:, i, :]$.

- Quantities like I_n, R_n or S_n to denote tensor sizes, where n is the dimension number. When giving asymptotic complexities, we sometimes omit the subscript to indicate we take the largest over all n, for example $I := \max_n\{I_n\}$ and $R := \max_n\{R_n\}$.
- Our main accuracy metric is the *relative error*, defined as $\epsilon(\boldsymbol{u}, \tilde{\boldsymbol{u}}) = \|\boldsymbol{u} - \tilde{\boldsymbol{u}}\|/\|\boldsymbol{u}\|$. This metric also applies to general tensors, not just vectors.
- For any integer n, $[n]$ denotes $\{1, \ldots, n\}$.

2. Related Work. This work is part of an ongoing trend in tensor network research that aims to exploit informed priors such as smoothness or sparsity for high-dimensional regression and, in particular, to do so via *predefined* sets of bases, in this case polynomials.

2.1. Polynomial Chaos Expansions. Let $f : \mathbb{R}^N \to \mathbb{R}$ be an L^2-integrable function that we wish to approximate, and let $\boldsymbol{\psi}^{(1)}, \ldots, \boldsymbol{\psi}^{(N)}$ be families of orthogonal (in a sense discussed below) polynomials. A *truncated PCE* of f is a linear combination of multivariate polynomial-term products:

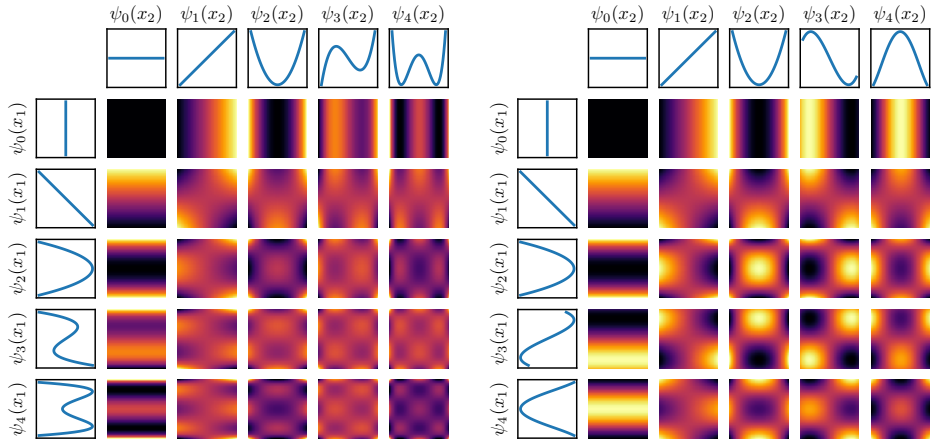
$$f(\boldsymbol{x}) \approx \hat{f}(\boldsymbol{x}) = \sum_{\boldsymbol{\alpha} \in \{0,\ldots,S-1\}^N} w_{\boldsymbol{\alpha}} \cdot \psi^{(1)}_{\alpha_1}(x_1) \cdots \psi^{(N)}_{\alpha_N}(x_N). \tag{2.1}$$

The $w_{\boldsymbol{\alpha}}$ are the *expansion weights* and are usually fitted via some available training data set $(\mathbf{X}, \boldsymbol{y})$ so that $\|\hat{f}(\mathbf{X}) - \boldsymbol{y}\|_2$ is minimized. They can be learned by solving the (usually overdetermined) linear system $\mathbf{M}(\mathbf{X}) \cdot \mathbf{w} = \mathbf{y}$ for $\mathbf{w}$, where $\mathbf{M}(\mathbf{X})$ is the so-called *design matrix* of the input feature matrix $\mathbf{X}$. See Fig. 1 for two examples of polynomial bases in $N = 2$ dimensions, where the product terms of Eq. 2.1 take the form of S^2 bivariate dictionary patches.

The main challenge with Eq. 2.1 is related to the number of coefficients to be learned, which grows exponentially as S^N. This is not only a computational problem: similarly to the so-called *Runge phenomenon* in 1D polynomial interpolation, PCE expansions can suffer from overfitting if too many non-zero coefficients are chosen.

Fortunately, coefficient sparsity can be imposed in order to prevent such an exponential blow-up of the number of degrees of freedom. For example, one may restrict the coefficient search to subregions of the hypercube $\{0, \ldots, S-1\}^N$. This is often done in the form of *total degree truncation*, which bounds the sum of degrees of each term $\boldsymbol{\alpha}$ by a threshold: $\|\boldsymbol{\alpha}\|_1 < p$. For instance, linear regression fixes $p = 2$. This technique decreases the number of degrees of freedom, namely to $\binom{N+p-1}{p-1}$ when p is an integer. When this reduction is not enough, the more general *hyperbolic truncation* enforces $\|\boldsymbol{\alpha}\|_q < p$ for q in $[0, 1]$. Good values for hyperparameters p and q that make the model generalize best can be chosen via k-fold or cross-validation strategies. One can also pursue sparsity in other ways, including *LASSO*, *ridge regression*, etc.

Parameter Distribution. The joint distribution $p(\boldsymbol{x})$ of the input parameters is another fundamental aspect of PCE modeling. In many settings, the N input variables are independent, i.e. their PDF is separable with $p(\boldsymbol{x}) = p_1(x_1) \cdot \cdots \cdot p_N(x_N)$. This is the case in experimental design, where the experimentalist is free to tune the input parameters at will within a prespecified domain, often a rectangle $\Omega \subset \mathbb{R}^N$. Independent variables are also a common assumption in variance-based sensitivity analysis, particularly in the so-called *method of Sobol* [24]. Under independence, choosing polynomial bases that are orthogonal to the parameters' marginal distributions makes it simple to compute useful statistics (mean, variance, etc.) that are important in uncertainty quantification, for example to compute Sobol indices [25] or



(a) Legendre (b) Hermite

Fig. 1: For $N = 2$ variables, bounding the degree by $S = 5$ along each dimension results in a dictionary of $S^N = 25$ 2D patches. When $N > 2$, truncation schemes are needed to prevent exponential growth. The polynomial families shown here, Legendre and Hermite, are orthogonal with respect to the uniform and Gaussian distributed measures, respectively.

derived quantities of interest such as the *dimension distribution* [21]. This simplicity no longer holds under parameter dependence but, even then, PCEs often converge well and are at least useful for regression purposes [27].

2.2. Tensor Networks: Tucker, Tensor Train, and Extended TT Decompositions. Tensor networks (TN) generalize low-rank matrix factorizations to higher-dimensional ($N > 2$) arrays, or *tensors*. Suppose $\mathcal{X}$ is an N-dimensional tensor indexed by integers $0 \le i_n \le I_n - 1$ for $n = 1, \dots, N$. Two of the most important TNs are:

- The *Tucker decomposition* [7] writes

$$\mathcal{X}[\boldsymbol{i}] \approx \sum_{\boldsymbol{r}=\mathbf{1}}^{\mathbf{R}} \mathcal{G}[\boldsymbol{r}] \mathbf{U}^{(1)}[i_1, r_1] \cdots \mathbf{U}^{(N)}[i_N, r_N] \tag{2.2}$$

  where $\boldsymbol{R} = (R_1, \dots, R_N)$ are the *Tucker ranks*, $\mathcal{G}$ is the *Tucker core*, and $\mathbf{U}^{(1)}, \dots, \mathbf{U}^{(N)}$ are the *Tucker factors*.
- The *tensor-train decomposition* (TT) [17] writes

$$\mathcal{X}[\boldsymbol{i}] \approx \sum_{\boldsymbol{r}=\mathbf{1}}^{\mathbf{R}} \mathcal{G}^{(1)}[r_0, i_1, r_1] \dots \mathcal{G}^{(N)}[r_{N-1}, i_N, r_N] \tag{2.3}$$

  where $\boldsymbol{R} = (R_1, \dots, R_{N-1})$ are the *TT ranks* ($R_0 = R_N = 1$ by convention) and the $\mathcal{G}^{(1)}, \dots, \mathcal{G}^{(N)}$ are the *TT cores*.

In this paper we make use of a combination of the Tucker and TT formats known as *extended TT* [19, 23] (ETT), whereby $\mathcal{X}$ is encoded as per the Tucker format, but the resulting Tucker core $\mathcal{G}$ is in turn represented in the TT scheme as N cores

Tensor network	Tensor algebra notation	Einstein's summation formula
Tucker	$\mathcal{C} \times_1 \mathbf{U}^{(1)} \times_2 \mathbf{U}^{(2)} \times_3 \mathbf{U}^{(3)}$	`ijk,ai,bj,ck->abc`
TT	$[[\mathcal{G}^{(1)}, \mathcal{G}^{(2)}, \mathcal{G}^{(3)}]]$	`iaj,jbk,kcl->abc`
ETT	$[[\mathcal{G}^{(1)} \times_2 \mathbf{U}^{(1)}, \mathcal{G}^{(2)} \times_2 \mathbf{U}^{(2)}, \mathcal{G}^{(3)} \times_2 \mathbf{U}^{(3)}]]$	`ixj,ax,jyk,by,kzl,cz->abc`

Table 1: Representations of a 3D tensor $\mathcal{X}_{abc}$ in three different tensor formats. The ETT format (used in this paper) combines the best properties of both TT and Tucker formats: (i) linear size growth w.r.t. N; and (ii) interpretation in terms of an expansion using N function bases.

$\mathcal{G}^{(1)}, \ldots, \mathcal{G}^{(N)}$. We write this compactly as $[[\mathcal{G}^{(1)} \times_2, \mathbf{U}^{(1)}, \ldots, \mathcal{G}^{(N)} \times_2 \mathbf{U}^{(N)}]]$, where $\times_2$ denotes the *tensor-times-matrix* product [12] of each TT core along its second dimension. In order to approximate a function $f(\boldsymbol{x})$ over a rectangular domain $\Omega = [a_1, b_1] \times \cdots \times [a_N, b_N]$ using the ETT format, we start out by discretizing coordinates $\boldsymbol{x}$ into a uniform tensor product grid indexed by $\boldsymbol{i}$: if $a_n \leq x_n \leq b_n$ for $n = 1, \ldots, N$, then $i_n := \text{round}\left(\frac{x_n - a_n}{b_n - a_n} \cdot (I_n - 1)\right) + 1$. This is a common choice in the literature; see e.g. [17]. Thus, $f \approx \mathcal{X} \approx [[\mathcal{G}^{(1)} \times_2, \mathbf{U}^{(1)}, \ldots, \mathcal{G}^{(N)} \times_2 \mathbf{U}^{(N)}]]$. Viewed from another angle, this is essentially a discrete version of the so-called *functional tensor train* (FTT) format [10].

The way tensor networks factorize an array can be written best by means of the *Einstein summation convention*, which uses subindices and implicitly sums over the indices that appear repeated. For example, a matrix product $\mathbf{A} = \mathbf{B} \cdot \mathbf{C}$ becomes $\mathbf{A}_{ab} = \mathbf{B}_{ai}\mathbf{C}_{ib}$, while a singular value decomposition $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ is $\mathbf{A}_{ab} = \mathbf{U}_{ai}\mathbf{\Sigma}_{ij}\mathbf{V}_{bj}$. The latter can be written even more compactly as `'ai,ij,bj->ab'` in many modern data analysis and learning packages (NumPy, TensorFlow, PyTorch, etc.). See Tab. 1 for example Einstein-like forms of ETT and its ingredients (Tucker, TT) in the 3D case. For a more in-depth discussion of TNs and their role in data analysis, signal processing and machine learning, we refer the reader to the survey by [5].

A method related to ours was proposed by [8], which also compresses the set of PCE coefficients in the TT format. However, they assume an adaptive sampling scenario where arbitrary coefficients can be acquired on demand, which is the case in their target application: approximating the solution of a stochastic partial differential equation (PDE) via the Galerkin method (a so-called *intrusive* method). A non-intrusive version that aims to approximate general black-box functions was considered by [4], but it still requires arbitrary sampling. Our approach, in contrast, pursues general black-box regression of a *fixed*, usually scarce set of points $\mathbf{X}$ and their corresponding groundtruth values $\boldsymbol{y}$. To cope with limited data availability, we induce sparsity via robust sparse regression techniques (hyperbolic truncation and LARS) *before* tensor compression.

3. Proposed Pipeline. We propose to estimate statistical moments using a three-step procedure:

1. Choose polynomial bases $\boldsymbol{\psi}^{(1)}, \ldots, \boldsymbol{\psi}^{(N)}$ that are orthogonal w.r.t. the marginal distributions of $\boldsymbol{x}$ and learn a sparse set of PCE weights $\boldsymbol{w}_{\boldsymbol{\alpha}}$ that regresses values $\boldsymbol{y}$ using features $\mathbf{X}$ over these bases. See Sec. 3.1.
2. Cast the sparse tensor $\boldsymbol{w}_{\boldsymbol{\alpha}}$ into a TT tensor representing the hypercube of

PCE coefficients, discretize the axes of $x_1, \ldots, x_N$, and sample the bases on these points to form Tucker factors. Together, they form an ETT regressor that approximates the target function f. See Sec. 3.2.

3. Compute the desired higher-order moments out of that TT tensor by applying the necessary operations (sum, Hadamard product) in the compressed domain. See Sec. 3.3.

3.1. Sparse PCE Fitting. Learning PCE weights is a standard and well-studied problem in the literature. First, for each dimension n, we employ the *Gram-Schmidt orthogonalization process* to find polynomials $\boldsymbol{\psi}^{(n)} = (\psi_0^{(n)}, \ldots, \psi_{S-1}^{(n)})$ that are orthogonal w.r.t. the n-th marginal distribution $p_n(x_n)$ [29]. Once all bases are found, we assemble the column-truncated design matrix $\mathbf{M}(\mathbf{X})[:, \mathcal{H}]$, where $\mathcal{H} := \{\boldsymbol{\alpha} \in \{0, \ldots, S_{N-1}\} \,|\, \|\boldsymbol{\alpha}\|_q < p\}$, and we find a least-squares solution for $\mathbf{M}(\mathbf{X})[:, \mathcal{H}] \cdot \boldsymbol{w} = \boldsymbol{y}$ using the LARS algorithm [27]. LARS efficiently generates a *coefficient path* that contains the optimal solution for different regularization strengths. We pick the solution that maximizes the model's performance over a validation data set. See Alg. 3.1.

3.2. Sparse PCE to ETT Conversion. The Tucker decomposition (Eq. 2.2) is a discretized analog of the PCE formula (Eq. 2.1) where the hypercube of S^N weights acts as Tucker core $\mathcal{G}$ while each polynomial basis $\psi^{(n)}$, evaluated over I_n points in the interval (a_n, b_n), becomes the factor matrix $\mathbf{U}^{(n)}$. By compressing the core $\mathcal{G}$ into a low-rank TT, we produce an ETT tensor $[[\mathcal{G}^{(1)} \times_2 \mathbf{U}^{(1)}, \ldots, \mathcal{G}^{(N)} \times_2 \mathbf{U}^{(N)}]]$ that approximates function f over the grid. If S_n is the number of polynomial basis elements for dimension n and I_n is the tensor grid size along that dimension, factor matrix $\mathbf{U}^{(n)}$ has shape $I_n \times S_n$ while core $\mathcal{G}^{(n)}$ has shape $R_{n-1} \times S_n \times R_n$. Therefore, the format needs to store a total of $O(NSR^2) + O(NIS)$ coefficients only, down from the initial I^N of the original tensor. Fig. 2 shows an example ETT representation for a (5^3)-sized input tensor using $S_n = 3$ basis functions per dimension n and TT ranks $R_n = 4$.

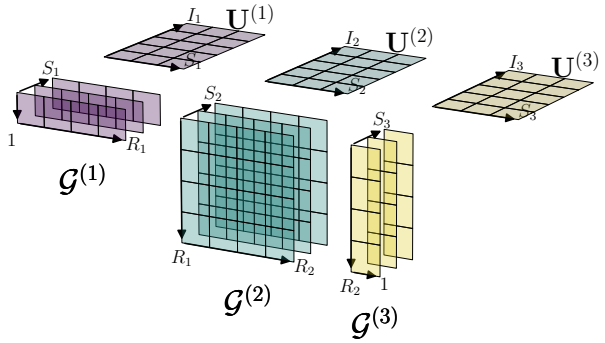


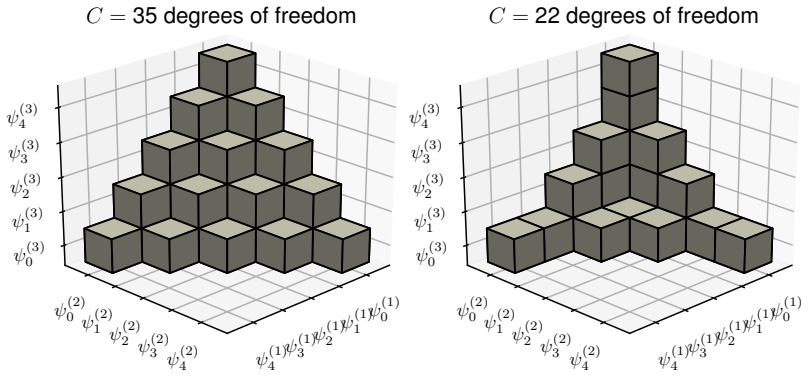
Fig. 2: When compressed in the ETT format, a 3D tensor has three cores $\mathcal{G}^{(1)}, \mathcal{G}^{(2)}$ and $\mathcal{G}^{(3)}$, each of which is a 3D tensor, and three factor matrices $\mathbf{U}^{(1)}, \mathbf{U}^{(2)}$ and $\mathbf{U}^{(3)}$. In this example, the tensor's spatial dimensions are $I_1 = I_2 = I_3 = 5$, its TT ranks are $R_1 = R_2 = 4$, and its Tucker ranks are $S_1 = S_2 = S_3 = 3$. We use the cores to encode sparse PCE weights, while the factors store discretized PCE basis functions.

Crucially, thanks to the hyperbolic sparsity constraints imposed on the PCE weights (step 2), modest TT ranks R_n are often enough to attain a high accuracy in step 3. The set $\mathcal{H}$ of valid hypercube entries restricts learning to weights that lie

	q						
	0.4	**0.5**	**0.6**	**0.7**	**0.8**	**0.9**	**1.0**
N. of non-zero weights C	51	96	186	441	801	1191	3003
Max. TT rank (ones)	2	3	4	5	6	7	6
Max. TT rank (random)	2	7	12	17	22	31	42

Table 2: Non-zero weights after hyperbolic truncation ($N = 10$ variables and varying q) and TT ranks after compression with threshold relative error $\epsilon = 10^{-4}$. The threshold is $p = 5.5$ in all cases. Even when weights are chosen at random (bottom row), the hyperbolic region structure keeps the required number of ranks low.

close to corner $(0, \dots, 0)$ of the hypercube, and fixes all other weights to 0; see Fig. 3 for two examples in 3D. In turn, the low TT ranks enable efficient post-processing, in particular the method for computing order three and four moments.



(a) Total degree truncation ($q = 1, p = 5$)

(b) Hyperbolic truncation ($q = 0.6, p = 6$)

Fig. 3: Two weight truncation examples for $N = 3$ variables. The set of surviving coefficients is well structured and always contains the origin.

Note that, in general, sparsity alone does not automatically mean a good low-rank approximation exists; for instance, a superdiagonal tensor (having, say, C ones along the superdiagonal and zeros elsewhere) has TT rank equal to C. Such a high rank drives the total number of TT elements to $O(NSC^2) + O(NIS)$, which can become already problematic if, say, $C > 100$ (in the applications we will show in Sec. 4, the best values of C often exceed 500). In contrast, we show that hyperbolic truncation imposes a highly structured arrangement of the set $\mathcal{H}$ of non-zero coefficients, leading to tensor ranks that grow very mildly with respect to C. See for example Tab. 2, which shows moderate resulting ranks for $N = 10$ input variables and varying levels of q, even when the weight values to compress are chosen at random.

We tackle the actual compression step with a sparse variant of the *TT-SVD* [17], the most popular algorithm for the TT decomposition. The original algorithm relies on computing the left singular values of a sequence of matrices $\{\mathbf{V}_n\}_{n=1}^{N-1}$, each of which has shape $SR_{n-1} \times S^{N-n}$. These matrices can be prohibitively wide if stored

in a dense form, but in fact they are column-sparse in our case (that is, most of their columns are zero). Our algorithm is identical to standard TT-SVD, with just two modifications:

- Instead of finding the SVD of $\mathbf{V}_n$, we compute the leading eigenvectors of $\mathbf{V}_n\mathbf{V}_n^T$, which has shape $SR_{n-1} \times SR_{n-1}$ (i.e., much smaller than $\mathbf{V}_n$). Although this form of computing left singular vectors can be less numerically stable than the SVD, the effect is negligible at the accuracy levels employed in our applications ($\epsilon \geq 10^{-4}$). The asymptotic cost of the eigendecomposition is $O(S^3R^3)$.
- We store $\mathbf{V}$ in a sparse matrix format; this way, $\mathbf{V}_n\mathbf{V}_n^T$ can be assembled efficiently. If $Z = |\mathcal{H}|$ is the number of non-zero PCE weights to be compressed, then $\mathbf{V}_n$ has at most Z non-zero columns, and the cost of computing the sparse matrix product is $O(S^2R^2Z)$.

All in all, after processing the full sequence $\mathbf{V}_n$ for $n = 1, \ldots, N-1$, the overall cost for this sparse TT-SVD is $O(NS^3R^3) + O(NS^2R^2Z)$.

The complete pipeline details are shown in Alg. 3.1.

Algorithm 3.1 Proposed pipeline to build a sparse PCE interpolator and convert it to the ETT format.

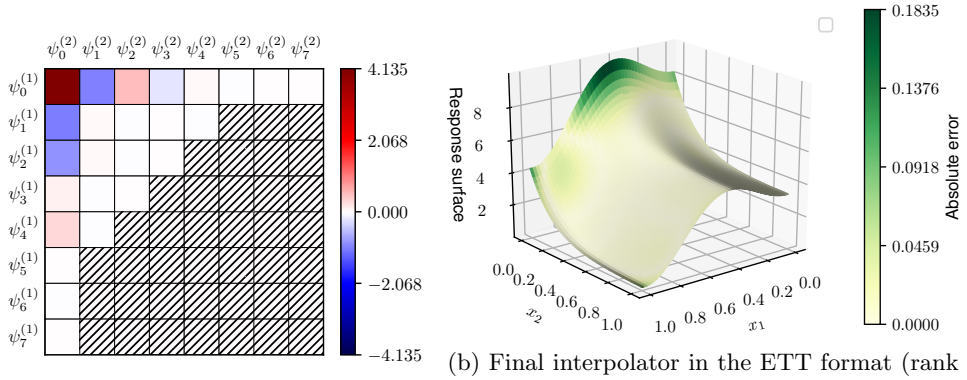
Input: Data $(\mathbf{X}, \boldsymbol{y})$, hyperparameters p, q, relative error ϵ, parameter bounds $(a_1, b_1), \ldots, (a_N, b_N)$, and grid sizes $I_1, \ldots, I_N$
Output: An ETT that regresses the target function on a discretized grid
for n $= 1, \ldots, N$ **do**
  ▶ $\boldsymbol{\psi}^{(n)}$:= polynomial basis that is orthogonal w.r.t. marginal distribution $p_n(x_n)$, built using the Gram-Schmidt orthogonalization process
  // Evaluate $\boldsymbol{\psi}^{(n)}$'s functions on the n-th dimensional tensor grid points to form the n-th factor matrix:
  for i $= 0, \ldots, I_n$ **do**
    for j $= 0, \ldots, \lfloor p \rfloor + 1$ **do**
      ▶ $\mathbf{U}^{(n)}[i, j] := \psi_j^{(n)}\left(a_n + \frac{i-1}{I_n - 1} \cdot (b_n - a_n)\right)$
    end for
  end for
end for
▶ Split the available data into training and validation: $\mathbf{X} = \mathbf{X}_{\text{train}} \cup \mathbf{X}_{\text{val}}$
▶ Select coordinates that satisfy the hyperbolic condition $\mathcal{H} = \{\boldsymbol{\alpha} : \|\boldsymbol{\alpha}\|_q < p\}$
▶ Assemble training and validation design matrices with columns restricted to $\mathcal{H}$: $\mathbf{M}(\mathbf{X}_{\text{train}})[:, \mathcal{H}]$ and $\mathbf{M}(\mathbf{X}_{\text{val}})[:, \mathcal{H}]$
▶ Find the LARS coefficient path for the LASSO minimization problem: $\boldsymbol{w}_\alpha := \arg\min_{\boldsymbol{w}} \left\{1/(2 \cdot \text{nsamples})\|\boldsymbol{y}_{\text{train}} - \mathbf{M}(\mathbf{X}_{\text{train}})[:, \mathcal{H}] \cdot \boldsymbol{w}\|_2^2 + \alpha\|\boldsymbol{w}\|_1\right\}$ for many values of α
▶ Select an α that minimizes validation error: $\arg\min_\alpha \|\mathbf{M}(\mathbf{X}_{\text{val}})[:, \mathcal{H}] \cdot \boldsymbol{w}_\alpha - \boldsymbol{y}_{\text{val}}\|$
▶ Retrain at regularization strength α over the entire data set $\mathbf{X}$ to find the final weights $\boldsymbol{w}$
▶ Let $\mathcal{S}$ be a sparse tensor with non-zero entries $\mathcal{H}$ and values $\boldsymbol{w}$
▶ Compress $\mathcal{S}$ into a TT tensor $[[\mathcal{G}^{(1)}, \ldots, \mathcal{G}^{(N)}]]$ at error ϵ using the sparse TT-SVD algorithm
return ETT tensor $\mathcal{T} = [[\mathcal{G}^{(1)} \times_2 \mathbf{U}^{(1)}, \ldots, \mathcal{G}^{(N)} \times_2 \mathbf{U}^{(N)}]]$

For illustrative purposes, Fig. 4 displays the learned PCE coefficients for the

synthetic 2D function proposed by [14], which is defined as

$$f(x_1, x_2) = \frac{1}{6}\left((30 + 5x_1 \sin(5x_1)) \cdot (4 + e^{-5x_2}) - 100\right)$$

with inputs uniformly distributed on the square $[0, 1]^2$. We also show the resulting response surface after running the entire Alg. 3.1.



(a) Sparse PCE coefficients

(b) Final interpolator in the ETT format (rank 3)

Fig. 4: Left: PCE coefficients learned by least angle regression (LARS) for the 2D test function proposed by [14] with 100 training samples. Grayed-out entries were discarded by the hyperbolic truncation step ($p = 8, q = 0.6$). Right: learned surface $\hat{f}$ after tensor conversion, color-coded with its point-wise absolute error. The learned PCE coefficients tend to be highly structured and therefore well compressible; in this example, a rank $R = 3$ made the tensor conversion incur only $\epsilon = 0.0005$ error.

3.2.1. Hyperparameters. Both p and q can have a significant effect on the accuracy of the final regressor. Higher values of p and/or q carve larger hypercube regions of allowed non-zeros, which in turn encourages LARS to increase the regularization strength and makes the algorithm more computationally expensive. In practice, it is best to fit p and q using a hyperparameter tuning algorithm. Regarding ϵ and $I_1, \ldots, I_N$, they have little influence as long as they are low (resp. high) enough. We used $\epsilon = 10^{-4}$ and $I_1 = \ldots I_N = 512$ in all our applications, which seem to be good default values (increasing the resolution beyond 512 did not lead to noticeable differences in accuracy).

3.3. Moment Computation. Suppose we want to find the K-th raw moment of any tensor $\mathcal{T}$ over a distribution $\mathcal{P}$. We have that $\mathbb{E}[\mathcal{T}^K] = \sum(\mathcal{P}\mathcal{T}^K)$. If we define

$$\begin{cases} \mathcal{T}_1 := \mathcal{T}\mathcal{P} \\ \mathcal{T}_2 := \cdots := \mathcal{T}_K := \mathcal{T}, \end{cases} \tag{3.1}$$

then

$$\mathbb{E}[\mathcal{T}^K] = \sum \mathcal{M} \text{ where } \mathcal{M} := \prod_{k=1}^{K} \mathcal{T}_k. \tag{3.2}$$

Next, we will give an algorithm to compute Eq. 3.2 for any general sequence of TT tensors $\mathcal{T}_1, \ldots, \mathcal{T}_K$ of the same shape.

Let each $\mathcal{T}_k$ have TT cores $[[\mathcal{T}_k^{(1)}, \ldots, \mathcal{T}_k^{(N)}]]$ and TT ranks $R_1^{(k)}, \ldots, \mathcal{R}_{N-1}^{(k)} \geq 1, R_0^{(k)} = R_N^{(k)} = 1$. To compute $\sum \mathcal{M}$ we will first use the observation [17] that the product $\mathcal{M}$ is given by the Kronecker product of the respective core slices for any entry $\boldsymbol{i}$:

$$\mathcal{M}[\boldsymbol{i}] = \left(\bigotimes_{k=1}^{K} \mathcal{T}_k^{(1)}[i_1]\right) \ldots \left(\bigotimes_{k=1}^{K} \mathcal{T}_k^{(N)}[i_N]\right). \tag{3.3}$$

Second, note that for any TT tensor $\mathcal{X} = [[\mathcal{X}^{(1)}, \ldots, \mathcal{X}^{(N)}]]$, the sum of its elements can be obtained [17] as

$$\sum \mathcal{X} = \prod_{n=1}^{N} \left(\sum_{i=1}^{I_n} \mathcal{X}^{(n)}[i]\right). \tag{3.4}$$

Putting Eqs. 3.3 and 3.4 together, we have

$$\sum \mathcal{M} = \prod_{n=1}^{N} \sum_{i=1}^{I_n} \bigotimes_{k=1}^{K} \mathcal{T}_k^{(n)}[i]. \tag{3.5}$$

By computing the product of Eq. 3.5 from right to left, we can write $\sum \mathcal{M} = \boldsymbol{u}^{(0)}$ with the following recurrence:

$$\boldsymbol{u}^{(n)} := \left(\sum_{i=1}^{I_n} \bigotimes_{k=1}^{K} \mathcal{T}_k^{(n)}[i]\right) \boldsymbol{u}^{(n+1)} = \sum_{i=1}^{I_n} \left[\left(\bigotimes_{k=1}^{K} \mathcal{T}_k^{(n)}[i]\right)\right] \boldsymbol{u}^{(n+1)}, \text{ for } n = N, \ldots, 0 \tag{3.6}$$

and $\boldsymbol{u}^{(N+1)} := (1)$.

Last, we speed up the computation of each n-th summand in the r.h.s. of Eq. 3.6 by rewriting it as follows:

$$\left(\bigotimes_{k=1}^{K} \mathcal{T}_k^{(n)}[i]\right) \boldsymbol{u}^{(n+1)} = \text{vec}\left(\mathcal{U}^{(n+1)} \times_1 \mathcal{T}_1^{(n)}[i] \times_2 \cdots \times_K \mathcal{T}_K^{(n)}[i]\right), \tag{3.7}$$

where $\mathcal{U}^{(n+1)}$ is the result of reshaping $\boldsymbol{u}^{(n+1)}$ into a tensor of shape $R_n^{(1)} \times \cdots \times R_n^{(K)}$. To see why Eq. 3.7 holds, consider the matrix identity

$$(\mathbf{B} \otimes \mathbf{A})\text{vec}(\mathbf{X}) = \text{vec}(\mathbf{A}\mathbf{X}\mathbf{B}^T), \tag{3.8}$$

which generalizes to the tensor case as

$$(\mathbf{A}_K \otimes \cdots \otimes \mathbf{A}_1)\text{vec}(\mathcal{X}) = \text{vec}(\mathcal{X} \times_1 \mathbf{A}_1 \times_2 \cdots \times_K \mathbf{A}_K). \tag{3.9}$$

See the survey by Kolda and Bader [12] for a closely related formula.

Thanks to Eq. 3.7 we can avoid the expensive Kronecker product of the l.h.s., which requires $O(R^{2K})$ operations, and instead use a TTM with cost $O(KR^{K+1})$. This yields a critical speed-up for the proposed method. After iterating over all slices $1, \ldots, I_n$ for all dimensions $1, \ldots, N$ and accounting for the cost of computing $\mathbf{A}_1, \ldots, \mathbf{A}_K$, the total asymptotic cost for our moment computation of order K becomes $O(NIKR^{K+1}) + O(NIR^2S)$.

See Alg. 3.2 for a summary of the algorithm just presented, applied to compute the raw moment of an ETT tensor under independently distributed variables.

Algorithm 3.2 Computing the K-th raw moment from an ETT tensor where the variables are independently distributed

Input: An ETT tensor $\mathcal{T}$ with cores $[[\mathcal{G}^{(1)} \times_2 \mathbf{U}^{(1)}, \dots, \mathcal{G}^{(N)} \times_2 \mathbf{U}^{(N)}]]$ approximating $f(\boldsymbol{x})$; a separable TT tensor $\mathcal{P}$ with cores $[[\mathcal{P}^{(1)}, \dots, \mathcal{P}^{(N)}]]$ approximating the distribution $p(\boldsymbol{x})$
Output: the K-th raw moment $\mathbb{E}[\mathcal{T}^K] = \sum(\mathcal{P}\mathcal{T}^K)$
▶ $\mathcal{U} = \text{ones}(1 \times \cdots \times 1)$ // K dimensions
for $n = N, \dots, 1$ **do**
  ▶ $\mathcal{V} = \text{zeros}(R_{n-1} \times \cdots \times R_{n-1})$ // K dimensions
  for $i = 1, \dots, I_n$ **do**
    ▶ $\mathbf{M} = \mathcal{G}^{(n)} \times_2 (\mathbf{U}^{(n)}[i, :])$ // Has shape $R_{n-1} \times R_n$
    ▶ $\mathcal{V} = \mathcal{V} + \mathcal{P}^{(n)}[i] \cdot (\mathcal{U} \times_1 \mathbf{M} \times_2 \cdots \times_K \mathbf{M})$ // $\mathcal{P}$ has rank 1, so $\mathcal{P}^{(n)}[i]$ is a scalar
  end for
  ▶ $\mathcal{U} = \mathcal{V}$
end for
▶ // $\mathcal{U}$ has now shape $(R_0)^K = 1^K$
return scalar($\mathcal{U}$)

4. Experiments. In this section, we demonstrate numerically the usefulness of the proposed algorithms as compared against several alternatives.

4.1. Software and Models. We have implemented and released the proposed method as part of the Python/PyTorch tensor network library *tntorch*. We use the library *scikit-learn* for the LARS regression step, and *Optuna* [1] with its default Tree-structured Parzen Estimator for hyperparameter optimization. We set a memory limit of 20MB for the PCE fitting step when optimizing hyperparameters, i.e. we discard combinations of p and q that yield a design matrix above that size. We considered three models that are well established in the uncertainty quantification literature and have independently distributed variables:

- *Welch 1992*: a 20-dimensional function [28] defined as:

$$\begin{aligned} f(\boldsymbol{x}) = {} & \frac{5x_{12}}{1+x_1} + 5(x_4 - x_{20})^2 + x_5 + 40x_{19}^3 - 5x_{19} + 0.05x_2 \\ & + 0.08x_3 - 0.03x_6 + 0.03x_7 - 0.09x_9 - 0.01x_{10} - 0.07x_{11} \\ & + 0.25x_{13}^2 - 0.04x_{14} + 0.06x_{15} - 0.01x_{17} - 0.03x_{18} \end{aligned} \tag{4.1}$$

  where all inputs are independently distributed over the rectangle $[-0.5, 0.5]^{20}$.
- *Piston*: a 7-dimensional function modeling the cycle time C of a piston within a cylinder [3]:

$$\begin{aligned} C(M, S, V_0, k, P_0, T_a, T_0) &= 2\pi \sqrt{\frac{M}{k + S^2 \frac{P_0 V_0}{T_0} \frac{T_0}{V^2}}}, \\ \text{where } & \begin{cases} V = \frac{S}{2k} \left(\sqrt{A^2 + 4k \frac{P_0 V_0}{T_0} T_a} - A \right) \\ A = P_0 S + 19.62M - \frac{KV_0}{S} \end{cases} \end{aligned} \tag{4.2}$$

  with uniformly distributed inputs over a rectangle in $\mathbb{R}^7$; see [3] for all details.

- *Dike*: an 8-dimensional function modeling the cost C (in millions of euros) caused by a flood overflowing a dike [13]:

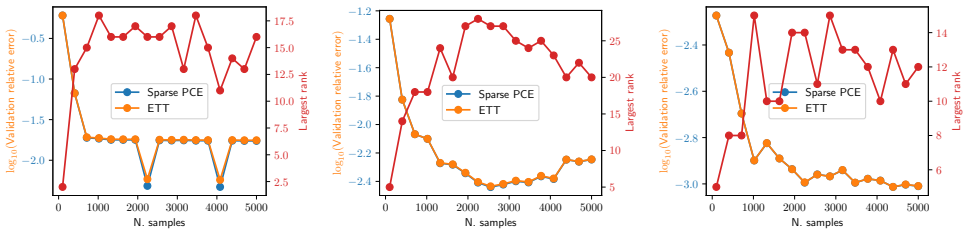
$$C(Q, K_s, Z_v, Z_m, H_d, C_b, L, B) = \mathbb{1}_{S>0} + \mathbb{1}_{S\leq 0}\left(0.2 + 0.8(1 - e^{-1000m^4/S^4})\right) + 0.05\min(\frac{H_d}{m}, 8) \tag{4.3}$$
$$\text{where } S = Z_v + H - H_d - C_b \text{ and } H = \left(\frac{Q}{BK_s\sqrt{(Z_m - Z_v)/L}}\right)^{3/5},$$

  Unlike the previous examples, the Dike's inputs are not all uniform but also include triangular, Gumbel, and normal marginal distributions [13].

For each model and number of samples, we used Bayesian optimization to select the best hyperparameters p and q (out of 100 trials) over ranges $1 \leq p \leq 10$ and $0.25 \leq q \leq 1$. The LARS regularization strength was automatically chosen in each case as the best performing value among the entire coefficient path, as measured over a 30% validation split. Unless stated otherwise, we use TT-SVD compression relative error $\epsilon = 10^{-4}$ and tensor grid size $I = 512$ in all experiments.

4.2. Regression Accuracy and Speed. We first report the relative error $\|\mathcal{T}[\mathbf{X}_{\text{val}}] - \boldsymbol{y}_{\text{val}}\|/\|\boldsymbol{y}_{\text{val}}\|$, where $\mathcal{T}$ is the tensor we learned, $\mathbf{X}_{\text{val}}$ is a matrix of $n_{\text{val}} = 10^4$ validation samples, and $\boldsymbol{y}_{\text{val}} = f(\mathbf{X}_{\text{val}})$ are the corresponding groundtruth values. Results are shown in Fig. 5, where we see that the conversion to ETT format leads to a small (often negligible) increase in error over the underlying sparse PCE expansion.

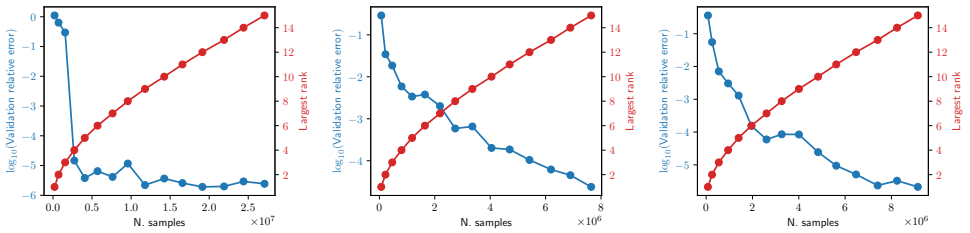
$$\mathbf{A}_{12} = 170$$



(a) *Welch 1992* model (b) *Piston* model (c) *Dike* model

Fig. 5: Regression relative error for each model and varying number of samples, computed over a validation set of size $n_{\text{val}} = 10'000$ for both the PCE regressor (blue line) and its conversion to the ETT format (orange line). In each case, the validation samples were drawn at random according to the joint PDF $p(\boldsymbol{x})$. The largest tensor rank is shown in red.

For comparison, we consider *cross approximation* (CA) [18], an algorithm that can learn a TT tensor out of a black-box function by evaluating it on a set of adaptively selected samples. Fig. 6 reports error results for CA-learned regressors of our three target functions. Although the number of ranks is often less than that of sparse PCE (red line in Fig. 5), CA requires many more samples to attain an equivalent validation

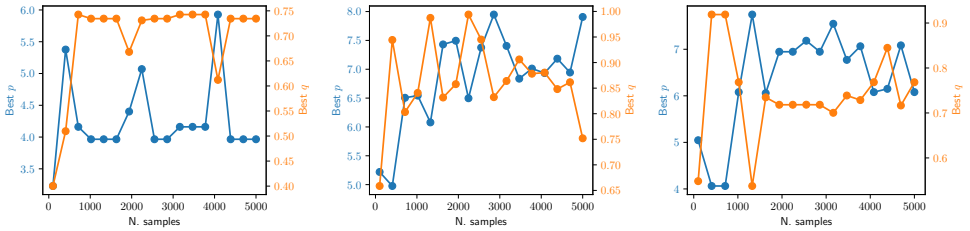


(a) *Welch 1992* model (b) *Piston* model (c) *Dike* model

Fig. 6: We show the cross-approximation regression validation error (estimated with $n_{\text{val}} = 10'000$ samples) and largest resulting rank using different numbers of training samples. Note that CA requires much more training data than sparse PCE to attain a comparable error.

error. In addition, it needs to select its training samples, whereas PCE can be learned from any given training set.

Fig. 7 depicts the best p and q values found by the hyperparameter tuning step in our method. Note that, in general, they are inversely correlated.



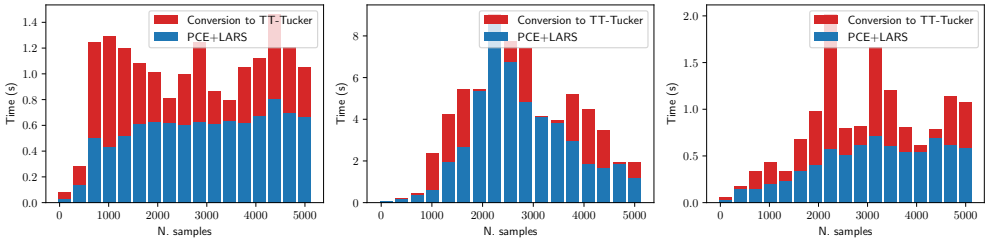
(a) *Welch 1992* model (b) *Piston* model (c) *Dike* model

Fig. 7: Best values for p and q in the sparse PCE learning step (Sec. 3.1) for varying number of samples, found using Bayesian hyperparameter optimization with 100 trials.

We also conducted time measurements (Fig. 8). The proposed tensor compression step consistently took under two seconds across our experiments. This was in all cases half or less of the total processing time (not accounting for PCE hyperparameter optimization), which is mostly dominated by assembling the design matrix and, especially, the LARS regression step.

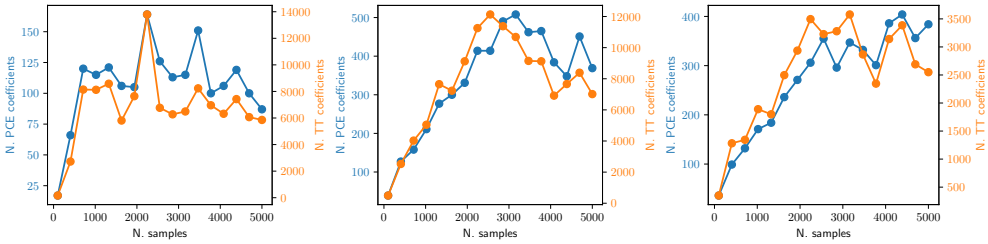
Next, we report in Fig. 9 the number of PCE and tensor-compressed coefficients. Note that the latter is sometimes larger than the number of training samples. This does not mean overfitting occurred: the learned PCE weights inhabit a large sparse tensor, and their positions, not just their values, carry relevant information. For example, storing 100 non-zero weights in an 8D sparse tensor in a non-compressed form would require $100 + 100 \cdot 8 = 900$ elements. The compressed tensor needs to spend coefficients to capture that structure.

4.3. Statistical Moments.



(a) *Welch 1992* model (b) *Piston* model (c) *Dike* model

Fig. 8: Training time broken down into PCE fitting time (in blue) plus conversion time into the ETT format (red). The times in blue do not include the fine-tuning of the PCE hyperparameters p and q. Each result is the median of 5 independent runs.



(a) *Welch 1992* model (b) *Piston* model (c) *Dike* model

Fig. 9: Number of non-zero PCE (in blue) and TT core (in orange) coefficients learned in each case. Each result is the median of 5 independent runs.

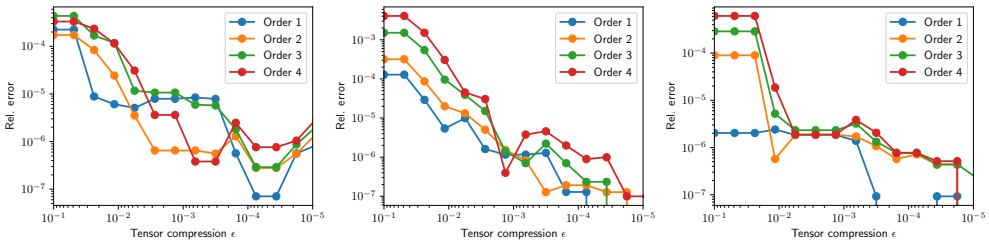
4.3.1. Sensitivity to ϵ and I. Figs. 10 and 11 show the accuracy dependence on the TT-SVD error tolerance ϵ and grid resolution I, respectively. We used $n_{\text{train}} = 500$. Values $\epsilon := 10^{-4}$ and $I := 512$ serve as good default choices as they already yield estimation relative errors in the order of 10^{-5}.

4.3.2. Comparison with Monte Carlo. Estimations for raw moments one through four are reported in Fig. 12 for both our method and MC as a baseline.

4.3.3. Comparison to Direct PCE Moment Computation. Given a representation of a function, one may exactly obtain its K-th order moment by directly expanding the expansion terms into a sum of K-tuple products; for example, for $K = 3$:

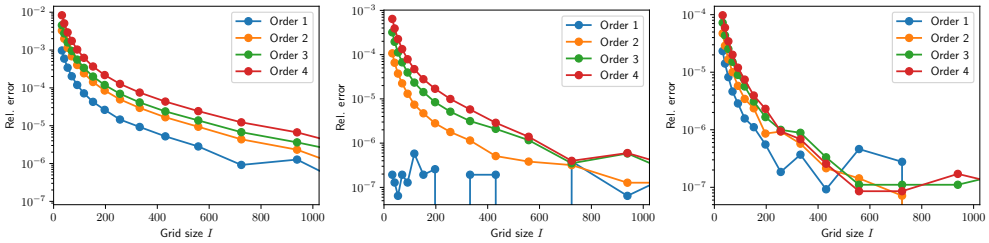
$$\mathbb{E}[\tilde{f}^3] = \mathbb{E}\left[\left(\sum_{\boldsymbol{\alpha}\in\mathcal{H}} w_{\boldsymbol{\alpha}}\psi_{\boldsymbol{\alpha}}\right)^3\right] = \sum_{\boldsymbol{\alpha},\boldsymbol{\beta},\boldsymbol{\gamma}\in\mathcal{H}} w_{\boldsymbol{\alpha}} w_{\boldsymbol{\beta}} w_{\boldsymbol{\gamma}} \mathbb{E}\left[\psi_{\boldsymbol{\alpha}}\psi_{\boldsymbol{\beta}}\psi_{\boldsymbol{\gamma}}\right]. \tag{4.4}$$

This approach is problematic for $K \geq 3$ because of the fast increase in the number of terms and because basis orthogonality does not simplify the expansion (as happens for $K = 2$). For example, finding the fourth moment of a 100-coefficient expansion would require computing $100^4 = 10^8$ expectations. The PCE expansions for the



(a) *Welch 1992* model (b) *Piston* model (c) *Dike* model

Fig. 10: Relative accuracy of moments of order up to 4 as we decrease the error tolerance ϵ for the sparse TT-SVD step. We keep $I = 512$ and $n_{\text{train}} = 500$ constant. As a baseline we use the moments obtained at $\epsilon = 10^{-6}$.



(a) *Welch 1992* model (b) *Piston* model (c) *Dike* model

Fig. 11: Relative accuracy of moments of order up to 4 as we increase the grid resolution I. We keep $\epsilon = 10^{-4}$ and $n_{\text{train}} = 500$ constant. As a baseline we use the moments obtained at $I = 2048$.

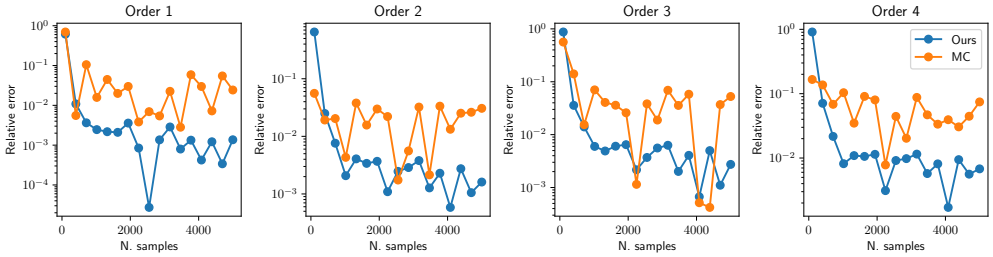
models considered in this paper are typically well above 100 or even 200 coefficients (Fig. 8), which renders this direct method non-viable for all but a minority of cases.

4.3.4. Comparison to Other Tensor Decomposition Methods. Given an ETT tensor $\mathcal{T}$, we support the value of Sec. 3.3's method for computing its K-order raw moment by comparing it to two alternatives:

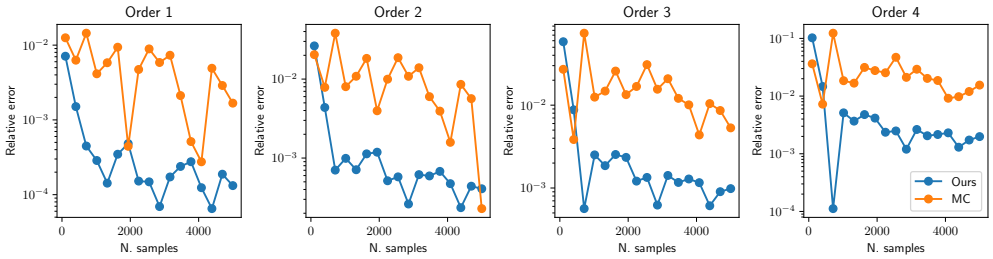
- We use cross-approximation to learn a new TT tensor that approximates $\mathcal{T}^K$ elementwise. Then, the raw moment equals $\sum(\mathcal{P} \cdot \mathcal{T}^K)$.
- Novikov et al. [16] give an algorithm to approximately compute $\sum \mathcal{T}_1, \cdots \mathcal{T}_K$ for any list of TT tensors. It relies on a sequence of products between a tensor-times-matrix (TTM) and a TT tensor and it rank-truncates the intermediate result at each step. We apply it to the tensors $(\mathcal{P}, \mathcal{T}, \cdots, \mathcal{T})$, where $\mathcal{T}$ appears K times, and use a relative error threshold of 10^{-4} for the rank truncations.

Results are reported in Tab. 3. While all three methods are approximate, they all agree with each other to multiple digits of precision; ours is consistently the fastest.

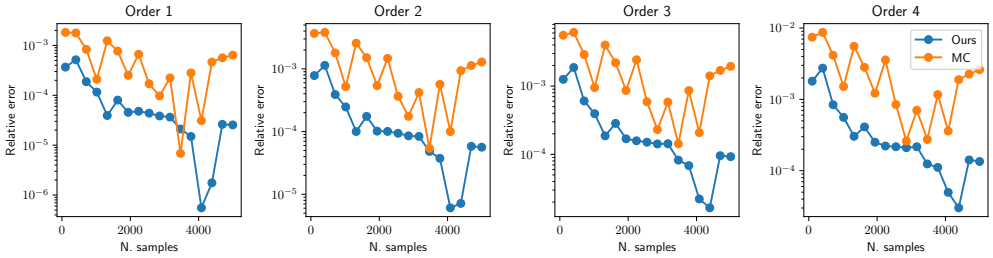
4.4. Updating Variables' Distribution. Having polynomial bases that are orthogonal with respect to the inputs' distribution is a paramount requirement of several PCE post-processing algorithms –for instance, those that efficiently obtain the Sobol indices of a PCE model [25]. If two functions f and g admit expansion



(a) Welch 1992



(b) Piston



(c) Dike

Fig. 12: Relative accuracy of moments of order up to 4 and increasing number of samples for both MC and the proposed method. As baseline we use MC at 10^7 samples.

weights $\{u_{\boldsymbol{\alpha}}\}_{\boldsymbol{\alpha}\in\mathcal{F}}$ and $\{v_{\boldsymbol{\alpha}}\}_{\boldsymbol{\alpha}\in\mathcal{G}}$ with the same orthogonal bases $\boldsymbol{\psi}^{(1)},\dots,\boldsymbol{\psi}^{(N)}$ w.r.t. a distribution $p(\boldsymbol{x})$, then their dot product over Ω and measure p equals simply $\langle f,g\rangle = \sum_{\boldsymbol{\alpha}\in\mathcal{F}\cap\mathcal{G}} u_{\boldsymbol{\alpha}} v_{\boldsymbol{\alpha}}$. In particular, we have $\|f\| = \|\{u_{\boldsymbol{\alpha}}\}_{\boldsymbol{\alpha}\in\mathcal{F}}\|$ and $\|g\| = \|\{v_{\boldsymbol{\alpha}}\}_{\boldsymbol{\alpha}\in\mathcal{G}}\|$. When the weights are sparse, this allows for efficient computation of two important quantities of interest:

- The variance of any function $f(\boldsymbol{x})$.
- The covariance and L^2 distance between a pair of functions $f(\boldsymbol{x}), g(\boldsymbol{x})$.

As attested in the literature, these operations allow for several useful applications in sensitivity analysis, dimensionality/model reduction, visualization, etc. Unfortunately, as argued in Sec. 1, orthogonality is lost as soon as one changes the input distribution (as it is desirable in e.g. importance sampling) or domain –as is the case if one wishes to, say, find a model's variance over some rectangular subregion Ω' within the original domain Ω.

	Moment			Time		
Order	Cross	TTM	Ours	Cross	TTM	Ours
2	5.135093	5.135339	5.135584	34.735191	2.490366	0.082438
3	13.199483	13.141907	13.141766	35.014597	5.761072	0.162104
4	78.631409	80.084427	80.087517	39.572971	21.888931	0.874509

(a) Welch1992. PCE hyperparameter tuning and learning took 19.165s, $p = 4.386$, $q = 0.566$, low-rank compression took 0.354s, largest rank = 13.

	Moment			Time		
Order	Cross	TTM	Ours	Cross	TTM	Ours
2	0.233647	0.233646	0.233647	10.274744	0.851440	0.089289
3	0.127622	0.127620	0.127623	14.466701	3.188387	0.132181
4	0.074896	0.074891	0.074897	13.483672	7.380891	0.480626

(b) Piston. PCE hyperparameter tuning and learning took 16.132s, $p = 5.419$, $q = 0.910$, low-rank compression took 0.270s, largest rank = 14.

	Moment			Time		
Order	Cross	TTM	Ours	Cross	TTM	Ours
2	0.416517	0.416518	0.416518	5.030414	0.642504	0.109994
3	0.269206	0.269204	0.269206	7.515483	0.762552	0.047809
4	0.174170	0.174146	0.174170	12.848080	3.509272	0.126049

(c) Dike. PCE hyperparameter tuning and learning took 20.154s, $p = 6.542$, $q = 0.647$, low-rank compression took 0.054s, largest rank = 9.

Table 3: We learn a sparse PCE for each model and convert it to low-rank ETT (more details in each subcaption). Then, we obtain moments of order two to four with our method (Alg. 3.2) compared against two alternatives: cross-approximation [18] and an algorithm based on tensor-train matrices (TTM) [16] using $\epsilon = 10^{-4}$ for its rank truncation steps.

In contrast, any distribution change $p' \neq p$ can be handled easily using the proposed representation. Let $\mathcal{T}_1, \mathcal{T}_2$, and $\mathcal{P}$ be TT tensors approximating two functions and their inputs' distribution, respectively, over a grid domain Ω. Let $\mathcal{P}'$ represent an updated distribution. Then,

$$\mathbb{E}_{p'}[\mathcal{T}_1\mathcal{T}_2] = \langle \mathcal{P}' \cdot \mathcal{T}_1, \mathcal{T}_2 \rangle \tag{4.5}$$

where $\langle \cdot, \cdot \rangle$ and $\cdot$ denote the dot and element-wise product of two compressed tensors, respectively. Both operations can be efficiently and exactly accomplished in the compressed domain: the dot product is computed by successive tensor contractions, while the element-wise product arises from Kronecker products of TT core slices, as we used in Sec. 3.3. Assuming the input variables $\boldsymbol{x}$ are independently distributed,

	$\mathrm{Var}_{p'}[f]$	Time	N. coef.
MC	0.902	152.639	N/A
Retraining	1.119	0.157	53
Ours	0.904	0.00888	$3.323 \cdot 10^4$

(a) Welch 1992

	$\mathrm{Var}_{p'}[f]$	Time	N. coef.
MC	0.00466	46.974	N/A
Retraining	0.00322	0.228	55
Ours	0.00462	0.0183	$2.075 \cdot 10^4$

(b) Piston

	$\mathrm{Var}_{p'}[f]$	Time	N. coef.
MC	$1.539 \cdot 10^{-4}$	55.104	N/A
Retraining	$1.113 \cdot 10^{-4}$	0.0272	16
Ours	$1.545 \cdot 10^{-4}$	0.00298	$1.54 \cdot 10^4$

(c) Dike

Table 4: Once an ETT tensor is learned for input variables' distribution $p(\boldsymbol{x})$, we can efficiently compute the variance on any updated distribution $p'(\boldsymbol{x})$. This is both faster and more accurate than retraining the regressor. The MC variances were estimated using 10^6 samples.

the PDF is separable and, therefore, $\mathcal{P}'$ can be represented exactly with tensor rank 1. In this case, computing Eq. 4.5 takes $O(NIS) + O(NSR^2)$ operations only.

Tab. 4 shows the variance on an updated region Ω' which is a subregion of Ω: for $n = 1, \ldots, N$, we choose the new bounds as $(a'_n, b'_n) := (a_n + (b_n - a_n)/8, b_n - (b_n - a_n)/8)$. We compare three methods: Monte Carlo; retraining a new PCE expansion that is orthogonal w.r.t. the new distribution; and Eq. 4.5 on the tensor originally learned with our method. The training dataset consisted of 10^3 samples. Not only is the latter method more efficient, but it also leads to more accurate results.

5. Conclusions. We proposed a moment computation algorithm that deepens the connection between two related fields: polynomial chaos and tensor networks. Instead of explicitly working with the sparse set of PCE coefficients found after hyperbolic truncation and LARS regression, we first organized them in a compressed form, namely the ETT decomposition. This step exploits the structure of the PCE weights found by the truncated LARS scheme and takes around half (often less) of the overall learning time. The proposed representation is amenable to various post-processing operations; in this paper we showed how to calculate higher-order statistical moments from it by performing all relevant operations (summation, elementwise products) efficiently in the tensor compressed domain. We argued that such operations are applicable for any distribution of the model variables, as long as the distribution can be encoded in a low-rank tensor of its own.

Note that the tensor regression step could be replaced for a different one – one could use methods such as Gorodetsky et al.'s [11] for a fixed training set, or cross-approximation if one is free to sample new function evaluations at will. In any case, the proposed pipeline is a useful addition to the toolbox of PCE practitioners that wish to estimate moments based on their models.

An obvious limitation of our method is its dependence on successful and affordable PCE interpolation, which places a fundamental constraint on hyperparameters p, q, and especially on the dimensionality N of the target function. In other words, the larger the N, the lower p and/or q need to be for the design matrix $\mathbf{M}(\mathbf{X})$ not to exceed available computer memory. In turn, these constraints may negatively affect the accuracy of the resulting model. In the future, we will explore further ways to bake the sparsity ansatzes exploited in this paper (hyperbolic truncation, LARS feature selection and regression) into the ETT computation, in the hope of bypassing such dimensionality limitations.

REFERENCES

[1] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, *Optuna: A next-generation hyperparameter optimization framework*, in Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2019, pp. 2623–2631.

[2] R. Ballester-Ripoll, E. G. Paredes, and R. Pajarola, *Sobol tensor trains for global sensitivity analysis*, Reliability Engineering and System Safety, 183 (2019), pp. 311–322.

[3] E. N. Ben-Ari and D. M. Steinberg, *Modeling data from computer experiments: An empirical comparison of kriging with mars and projection pursuit regression*, Quality Engineering, 19 (2007), pp. 327–338.

[4] D. Bigoni, A. Engsig-Karup, and Y. Marzouk, *Spectral tensor-train decomposition*, SIAM Journal on Scientific Computing, 38 (2016), pp. A2405–A2439.

[5] A. Cichocki, N. Lee, I. Oseledets, A.-H. Phan, Q. Zhao, and D. P. Mandic, *Tensor networks for dimensionality reduction and large-scale optimization: Part 1 low-rank tensor decompositions*, Foundations and Trends in Machine Learning, 9 (2016), pp. 249–429.

[6] N. Cohen, O. Sharir, and A. Shashua, *On the expressive power of deep learning: A tensor analysis*, in Journal of Machine Learning Research, 2016.

[7] L. de Lathauwer, B. de Moor, and J. Vandewalle, *On the best rank-1 and rank-$(R_1, R_2, ..., R_N)$ approximation of higher-order tensors*, SIAM Journal of Matrix Analysis and Applications, 21 (2000), pp. 1324–1342.

[8] S. Dolgov, B. N. Khoromskij, A. Litvinenko, and H. G. Matthies, *Polynomial chaos expansion of random coefficients and the solution of stochastic partial differential equations in the tensor train format*, SIAM-ASA Journal on Uncertainty Quantification, (2015), https://arxiv.org/abs/1503.03210.

[9] B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani, *Least angle regression*, Annals of Statistics, 32 (2004), pp. 407–499.

[10] A. Gorodetsky, S. Karaman, and Y. Marzouk, *A continuous analogue of the tensor-train decomposition*, Computer Methods in Applied Mechanics and Engineering, 347 (2019), pp. 59–84.

[11] A. A. Gorodetsky and J. D. Jakeman, *Gradient-based optimization for regression in the functional tensor-train format*, ArXiv e-print 1801.00885, (2018), https://arxiv.org/abs/1801.00885.

[12] T. G. Kolda and B. W. Bader, *Tensor decompositions and applications*, SIAM Review, 51 (2009), pp. 455–500.

[13] M. Lamboni, B. Iooss, A.-L. Popelin, and F. Gamboa, *Derivative-based global sensitivity measures: General links with sobol' indices and numerical tests*, Mathematics and Computers in Simulation, 87 (2013), pp. 45–54.

[14] Y. B. Lim, J. Sacks, W. J. Studden, and W. J. Welch, *Design and analysis of computer experiments when the output is highly correlated over the input space*, Canadian Journal of Statistics, 30 (2002), pp. 109–126.

[15] A. Novikov, D. Podoprikhin, A. Osokin, and D. Vetrov, *Tensorizing neural networks*, in Advances in Neural Information Processing Systems, 2015, https://arxiv.org/abs/1509.06569.

[16] A. Novikov, A. Rodomanov, A. Osokin, and D. Vetrov, *Putting MRFs on a tensor train*, in Proceedings International Conference on Machine Learning, 2014, pp. 811–819.
[17] I. V. Oseledets, *Tensor-train decomposition*, SIAM Journal on Scientific Computing, 33 (2011), pp. 2295–2317.
[18] I. V. Oseledets and E. Tyrtyshnikov, *TT-cross approximation for multidimensional arrays*, Linear Algebra Applications, 432 (2010), pp. 70–88.
[19] I. V. Oseledets and E. E. Tyrtyshnikov, *Tensor tree decomposition does not need a tree*, Preprint (Submitted to Linear Algebra Appl) 2009-04, INM RAS, Moscow, 2009, http://pub.inm.ras.ru/pub/inmras2009-08.pdf.
[20] A. Owen, J. Dick, and S. Chen, *Higher order Sobol' indices*, Information and Inference, 3 (2014), pp. 59–81.
[21] A. B. Owen, *The dimension distribution and quadrature test functions*, Statistica Sinica, 13 (2003), pp. 1–17.
[22] E. Robeva and A. Seigal, *Duality of graphical models and tensor networks*, Information and Inference, 8 (2018), pp. 273–288.
[23] R. Schneider and A. Uschmajew, *Approximation rates for the hierarchical tensor format in periodic Sobolev spaces*, Journal of Complexity, 30 (2014), pp. 56–71, https://doi.org/https://doi.org/10.1016/j.jco.2013.10.001, https://www.sciencedirect.com/science/article/pii/S0885064X13000782. Dagstuhl 2012.
[24] I. M. Sobol, *Global sensitivity indices for nonlinear mathematical models and their monte carlo estimates*, Mathematics and Computers in Simulation, (2001).
[25] B. Sudret, *Global sensitivity analysis using polynomial chaos expansions*, Reliability Engineering and System Safety, 93 (2008), pp. 964 – 979.
[26] B. Sudret, M. Berveiller, and M. Lemaire, *A stochastic finite element procedure for moment and reliability analysis*, European Journal of Computational Mechanics, 15 (2006), pp. 825–866.
[27] E. Torre, S. Marelli, P. Embrechts, and B. Sudret, *Data-driven polynomial chaos expansion for machine learning regression*, Journal of Computational Physics, (2019), https://arxiv.org/abs/1808.03216.
[28] W. J. Welch, R. J. Buck, J. Sacks, H. P. Wynn, T. J. Mitchell, and M. D. Morris, *Screening, predicting, and computer experiments*, Technometrics, 34 (1992), pp. 15–25.
[29] J. A. Witteveen and H. Bijl, *Modeling arbitrary uncertainties using Gram-Schmidt polynomial chaos*, in AIAA Aerospace Sciences Meeting and Exhibit, January 2006, pp. 1–17.