A continuous analogue of the tensor-train decomposition

Alex Gorodetsky[a,*], Sertac Karaman[b], Youssef Marzouk[b]

[a] *University of Michigan, Ann Arbor, MI 48109, USA*
[b] *Massachusetts Institute of Technology, Cambridge, MA 02139, USA*

Abstract

We develop new approximation algorithms and data structures for representing and computing with multivariate functions using the functional tensor-train (FT), a continuous extension of the tensor-train (TT) decomposition. The FT represents functions using a tensor-train ansatz by replacing the three-dimensional TT cores with univariate matrix-valued functions. The main contribution of this paper is a framework to compute the FT that employs adaptive approximations of univariate fibers, and that is not tied to any tensorized discretization. The algorithm can be coupled with any univariate linear or nonlinear approximation procedure. We demonstrate that this approach can generate multivariate function approximations that are several orders of magnitude more accurate, for the same cost, than those based on the conventional approach of compressing the coefficient tensor of a tensor-product basis. Our approach is in the spirit of other continuous computation packages such as Chebfun, and yields an algorithm which requires the computation of "continuous" matrix factorizations such as the LU and QR decompositions of vector-valued functions. To support these developments, we describe continuous versions of an approximate maximum-volume cross approximation algorithm and of a rounding algorithm that re-approximates an FT by one of lower ranks. We demonstrate that our technique improves accuracy and robustness, compared to TT and quantics-TT approaches with fixed parameterizations, of high-dimensional integration, differentiation, and approximation of functions with local features such as discontinuities and other nonlinearities.

Keywords: tensor decompositions, tensor-train, Chebfun, high-dimensional approximation

1. Introduction

Tensor decompositions are a popular tool for mitigating the curse of dimensionality in applications that involve large-scale multiway arrays. Examples are wide-ranging and include compressing the results of scientific simulations [1], neural networks [2], numerical solution of PDEs [3], stochastic optimal control [4, 5], and uncertainty quantification [6, 7, 8].

Yet many of the above problems are not naturally posed in the setting of discrete, structured grids. Instead, they are more naturally cast as questions of *function approximation*. Here, low-rank tensor decompositions have been applied by either (i) discretizing and evaluating a multivariate function on a tensor-product grid to form a multiway array [9, 10, 11], or (ii) representing the function in a tensor-product basis and decomposing the coefficient tensor [12, 13]. But these two methods are not flexible enough for many applications; more general "continuous" or functional low-rank representations are needed. In this paper we describe an *adaptive* low-rank function approximation framework that is not limited to particular discretizations or fixed parameterizations.

*Corresponding author

As an example, consider representing a function with localized features—e.g., a probability density function that concentrates around one or more modes or a function with jump discontinuities; doing so can require specialized (non tensor-product) discretizations to avoid excessive computation. Indeed, adaptive mesh refinement is an important area of research precisely because efficient and accurate solutions to real-world problems must often resolve local structure. Another limitation, specifically of the discretization approach, is that its utility diminishes outside of the tasks of compressed storage and evaluation. To illustrate, consider computing a derivative of a compressed function. If the compressed representation is built from pointwise evaluations on a tensor-product grid, derivative approximations are essentially limited to finite difference rules that can be defined on the same underlying grid. Yet scalable algorithms for uncertainty quantification or control might first require approximating a function, and then operating (e.g., performing differentiation) with the approximation. For instance, in the context of an iterative PDE solver, one might need to apply a Poisson operator with an inhomogeneous coefficient field onto a test function [14]. In this case, the grid representing the field may differ from the grid representing the test function, and computing inner products or elementwise products of arrays of different sizes is an ill-defined operation.

On the other hand, parameterizing a function using a prescribed tensor-product basis and then compressing the coefficients is often insufficiently expressive. Modern applications of machine learning suggest that tensor-product bases might not efficiently represent many phenomena; nonlinear parameterizations, e.g., neural networks, can be more successful. In uncertainty quantification and control theory, there has also been a realization that propagating uncertainty, performing inference, or representing feedback policies for complex models requires a certain degree of nonlinear parameterization, for example, through local approximations [15] or with neural networks [16].

Rather than compressing function evaluations on a fine grid or the coefficients of a prescribed tensor-product basis, we seek a more adaptive and expressive approach, suited to a wide range of computational operations. The central element of our approach is a continuous analogue of the tensor-train representation, called the *functional tensor-train*, described in [17]. In that work, a multivariate function is represented as a particular sum of products of univariate functions:

$$f(x_1, \ldots, x_d) = \sum_{\alpha_0=1}^{r_0} \sum_{\alpha_1=1}^{r_1} \cdots \sum_{\alpha_d=1}^{r_d} f_1^{(\alpha_0,\alpha_1)}(x_1) f_2^{(\alpha_1,\alpha_2)}(x_2) \cdots f_d^{(\alpha_{d-1},\alpha_d)}(x_d), \tag{1}$$

with $r_0 = r_d = 1$. Analytical examples in [17] demonstrate how certain functions can be represented in this continuous low-rank format.

In this paper we provide a *computational methodology* for approximating multivariate functions and computing with them in this format. We do not discretize functions on tensor-product grids and we do not *a priori* specify a tensor-product basis for approximation, and as a result our approach overcomes the limitations described above. Furthermore, our approach is adaptive and akin to incorporating adaptive grid refinement within the context of low-rank approximations. While some related approaches [18, 19, 20] represent low-rank functions without relying on tensor-product discretizations or coefficient tensors, the firs two [18, 19] employ the canonical polyadic format and all of them rely on fixed (non-adaptive) parameterizations to learn from scattered data. To the best of our knowledge, there are no existing methods for low-rank function approximaton that adapt parameterizations (on a grid or otherwise) in addition to ranks. Overall, our contributions include:

1. *locally and globally adaptive* algorithms for low-rank approximation of black-box functions;
2. algorithms for rounding multivariate functions already in FT format; and
3. a new *data structure* for representing (1) that is suited to both linear and nonlinear parameterizations of the univariate functions.

These contributions are enabled by integrating two lines of research: tensor decompositions and continuous linear algebra. We combine the idea of separation of variables, which underlies tensor decompositions, with the flexibility of linear and multilinear algebraic algorithms for "computing with

functions." Computing with functions is an emerging alternative to *discretize-then-solve* methodologies [21], and has been facilitated by continuous extensions to the common and widely used linear algebra techniques that underpin virtually all numerical simulations. For example, [22, 23, 24, 25, 26] introduce an extension of MATLAB, called Chebfun, for computing with functions of one, two [24], and three [27] variables. Chebfun implements continuous versions of the LU, QR, and singular value decompositions; enables approximation, optimization, and rootfinding; and provides methods for solving differential equations without discretization. Another example of computing with functions 'on-the-fly' is [28], where functions are adaptively built, as needed, to solve the Schrödinger equation. Our work extends this methodology and portions of Chebfun's capabilities to arbitrary dimensions by marrying them with tensor decompositions.

As demonstrated by Chebfun for almost a decade, the advantages of computing with functions include increased accuracy and stability, along with more natural modeling and algorithm development: complex operations on functions can be built through relatively simpler operations with great generality. Analogously, our approach provides similar building blocks for multivariate functions. These building blocks lead to new capabilities relative to existing tensor-based methods, including the quantics tensor-train (QTT) decomposition [3, 29]; capabilities include progressive *refinement of local features*, *adaptive integration and differentiation* rules that offer significant gains in efficiency and numerical stability, and the ability to apply binary operations to functions that have entirely *different parameterizations.*

The rest of the paper is organized in four sections. In Section 2, we detail how tensors appear in function approximation problems and describe the resulting limitations. In Section 3, we describe a data structure for representing (1) that stores parameters of the univariate functions *independently.* The structure does not store any tensor, either explicitly or implicitly. We also show how storing a tensor-train decomposition of a coefficient tensor is a specific realization of our framework and comment on the additional flexibility that we add. In Section 4 we describe continuous analogues of cross-approximation and rounding that are used to decompose a black box function into its low-rank representation. Finally, in Section 5 we present numerical experiments validating our approach that demonstrate significant improvements over previous low-rank functional representations.

2. Discrete tensor-trains and function approximation

In this section, we provide background on tensor decompositions, review their existing use for representing functions, and describe their limitations. Let $\mathbb{Z}^+$ denote the set of positive integers and $\mathbb{R}$ the set of reals. Let $d \in \mathbb{Z}^+$ and $\mathcal{X} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_d$ be a tensor-product space with $\mathcal{X}_k \subset \mathbb{R}$ for $k = 1, \ldots, d$. Let $n_k \in \mathbb{Z}^+$ for $k = 1, \ldots, d$ and $N = \prod_{k=1}^{d} n_k$. A tensor is a d-way array with n_k elements along *mode* k, and is denoted by uppercase bold calligraphic letters $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$. The special case of a 2-way array (matrix) is denoted by an uppercase bold letter, e.g., $\mathbf{A} \in \mathbb{R}^{m \times n}$, and a 1-way array (vector) is denoted by a lowercase bold letter, e.g., $\boldsymbol{a} \in \mathbb{R}^m$.

2.1. Tensor-train representation

A *tensor-train* (TT) representation of a tensor $\boldsymbol{\mathcal{A}}$ is defined by a list of 3-way arrays, $\mathbb{TT}(\boldsymbol{\mathcal{A}}) = (\boldsymbol{\mathcal{A}}_k \in \mathbb{R}^{r_{k-1} \times n_k \times r_k})_{k=1}^{d}$, with $r_0 = r_d = 1$ and $r_k \in \mathbb{Z}^+$ for $k = 2, \ldots, d-1$. Each $\boldsymbol{\mathcal{A}}_k$ is called the *TT-core*, and the sizes $(r_k)_{k=0}^{d}$ are called the *TT-ranks*. Computing with a tensor in TT format requires computing with its cores. For example, an element of a tensor is obtained through multiplication

$$\boldsymbol{\mathcal{A}}[i_1, i_2, \ldots, i_d] = \boldsymbol{\mathcal{A}}_1[1, i_1, :]\boldsymbol{\mathcal{A}}_2[:, i_2, :] \cdots \boldsymbol{\mathcal{A}}_d[:, i_d, 1], \quad 1 < i_k < n_k \text{ for all } k,$$

and tensor-vector contraction, with $(\boldsymbol{w}_k \in \mathbb{R}^{n_k})_{k=1}^{d}$, is obtained according to

$$\boldsymbol{\mathcal{A}} \times_1 \boldsymbol{w}_1 \cdots \times_d \boldsymbol{w}_d = \left(\sum_{i_1=1}^{n_k} \boldsymbol{\mathcal{A}}_1[1, i_1, :]\boldsymbol{w}[i_1] \right) \left(\sum_{i_2=1}^{n_2} \boldsymbol{\mathcal{A}}_2[:, i_2, :]\boldsymbol{w}[i_2] \right) \cdots \left(\sum_{i_d=1}^{n_d} \boldsymbol{\mathcal{A}}_d[:, i_d, 1]\boldsymbol{w}[i_d] \right).$$

The advantage of this representation of tensors is reduced storage and computational costs that scale *linearly* with dimension, rather than exponentially. If we let $r_k = r$ and $n_k = n$, then the cost of storing the list of cores is $\mathcal{O}(dnr^2)$, and the costs of evaluation and contraction are $\mathcal{O}(dr^2)$ and $\mathcal{O}(dnr^2)$, respectively.[1] The computational complexity becomes driven by the problem's rank.

In practice, tensors typically have low-rank representations that are *approximate* rather than exact. To make the concept of approximate low-rank representations more concrete, the TT-ranks are related to SVD ranks of various tensor reshapings. Let $\mathbf{A}^k$ represent a reshaping of the tensor $\boldsymbol{\mathcal{A}}$ along the kth mode into a matrix: in MATLAB notation we have

$$\mathbf{A}^k = \texttt{reshape}\left(\boldsymbol{\mathcal{A}}, \left[\prod_{i=1}^{k} n_i, \prod_{i=k+1}^{d} n_k\right]\right).$$

Now consider the SVD of this matrix. According to [30], if for each unfolding k we have $\mathbf{A}^k = \mathbf{G}^k + \mathbf{E}^k$ such that the matrices $\mathbf{G}^k$ have SVD rank equal to r_k and $\|\mathbf{E}^k\| = \varepsilon_k$, then a rank $\boldsymbol{r} = (1, r_1, \ldots, r_{d-1}, 1)$ approximation $\boldsymbol{\mathcal{A}}_{\boldsymbol{r}}$ exists with bounded error

$$\|\boldsymbol{\mathcal{A}} - \boldsymbol{\mathcal{A}}_{\boldsymbol{r}}\|^2 \leq \sum_{k=1}^{d-1} \varepsilon_k^2.$$

Essentially, this result implies that for problems with good separability, as defined by the matrix SVD, between groups of the first k and last $(d-k)$ variables, an approximate tensor-train representation with low ranks may attain good accuracy.

Finally, we comment on how existing literature converts function approximation problems into tensor decomposition problems. The first method is straightforward: each $\mathcal{X}_k$ is discretized into n_k elements so that $\mathcal{X}$ is a tensor product grid. The tensor arises from evaluating a multivariate function $f(x_1, \ldots, x_d)$ at each node in this grid. In this case we view $\boldsymbol{\mathcal{A}}$ as a mapping from a discrete set to the real numbers, $\boldsymbol{\mathcal{A}} : \mathcal{X} \to \mathbb{R}$.

The second method for obtaining a tensor is to represent a function in a tensor product basis, i.e., a basis produced by taking a full tensor product of univariate functions [13, 31, 6]. In particular, let

$$\{\phi_{ki_k} : \mathcal{X}_k \to \mathbb{R} : k = 1, \ldots, d, \text{ and } i_k = 1, \ldots, n_k\}$$

be a set of univariate basis functions. Then a function represented in this basis can be written as

$$f(x_1, x_2, \ldots, x_d) = \sum_{i_1}^{n_1} \sum_{i_2}^{n_2} \cdots \sum_{i_d}^{n_n} \boldsymbol{\mathcal{A}}[i_1, i_2, \ldots, i_d] \phi_{1i_1}(x_1) \phi_{2i_2}(x_2) \cdots \phi_{di_d}(x_d). \tag{2}$$

Now the coefficients of this expansion comprise the tensor of interest. This representation arises quite frequently in uncertainty quantification, e.g., polynomial chaos expansions where the univariate basis functions are orthonormal polynomials.[2]

Finally, we point out a simple extension to the tensor-train that is typically more efficient when the n_k are large. The *quantics tensor-train* (QTT) [29, 9] decomposition increases the dimension of a tensor by reshaping the standard modes into $2 \times 2 \times \cdots$ modes, $3 \times 3 \times \cdots$ modes, or larger-sized modes. Typically such a reshaping allows a further reduction in storage complexity so that it scales with $\log(n)$ rather than n.

---

[1] Contraction of the full tensor $\sum_{i_1=1}^{n_1} \sum_{i_2=1}^{n_2} \cdots \sum_{i_d=1}^{n_d} \boldsymbol{\mathcal{A}}[i_1, i_2, \ldots, i_d] \boldsymbol{w}[i_1] \boldsymbol{w}[i_2] \cdots \boldsymbol{w}[i_d]$ requires an exponentially growing, $N = \mathcal{O}(n^d)$, number of operations.

[2] While the grid discretization approach, i.e., pointwise evaluations of a function on a tensor product grid, could also be thought of as, for instance, a piecewise constant approximation, the latter involves the *additional* assumption of a particular basis or interpolation scheme. Thus we prefer to distinguish the grid discretization approach from the alternative of choosing a basis and storing its coefficients.

Desired Usage	Grid discretization	Tensor-of-coefficients
Represent local features	no adaptivity	only linear parameterizations and no adaptivity
Extract gradients	only finite differences	**no limitation**
Perform integration	only tensor-product quadrature	**no limitation**
Multilinear algebra with several functions	only identical grids	only identical parameterizations

Table 1: Limitations of grid discretization and tensor-of-coefficients formulations for computing with functions

2.2. Limitations of existing tensor-train approaches

In this section, we describe some limitations of the TT representation for function approximation, which follow from its discrete nature; these limitations will motivate continuous extensions. Among our broader goals is to use low rank approximations to mitigate the curse of dimensionality in uncertainty quantification and control. Both application areas share procedures that employ simple operations—addition, multiplication, integration, differentiation, and computing inner products—to construct more complex algorithms. However, the direct application of existing low-rank tensor approximations within such algorithms is faced with the following challenges:

1. Employing low-rank tensors in new contexts requires choosing specific discretizations or parameterizations (e.g., what grid to use, what quadrature rule to incorporate); the fact that these choices are not automatic limits generality and adaptivity.

2. Operating on functions with different discretizations requires ad hoc interpolation onto shared grids or a shared parameterization, before executing an operation.

Specific limitations associated with these challenges are summarized in Table 1. We illustrate these limitations with three examples:

(1) Approximating functions with discontinuities or local features. Multivariate functions with localized nonlinearities or discontinuities arise in, among other settings, stochastic optimal control problems with discontinuous cost functions [5]. In these cases, specifying basis functions *a priori* and approximating the resulting tensor of coefficients can lead to numerical issues: for instance, if the basis functions do not respect the discontinuity, the Gibbs phenomenon can ruin approximation quality. On the other hand, a fixed wavelet basis can potentially under-resolve or over-resolve the discontinuity. To the best of our knowledge, adaptive schemes to address these issues in the setting of high-dimensional low-rank function approximation do not yet exist.

(2) Computing gradients and integrals of multivariate functions. Gradients and integrals appear within a myriad of algorithms, and efficient ways to compute them in high dimensions are essential. For example, value and policy iteration in dynamic programming [32] involve updating a multivariate function through fixed point iterations, and these updates require the gradient of a multivariate function [33]. If we simply represent a multivariate function on a discretized grid, obtaining the gradient is an undefined operation. Numerical finite difference schemes to approximate gradients face the problem of choosing a good discretization. Dealing with such issues by, for example, discretizing each input coordinate with an extremely large number of points and performing a QTT decomposition does not solve this problem, as finite-difference roundoff errors will persist.

(3) Computing with several functions. Once a low-rank approximation of a multivariate function is constructed, there are enormous computational advantages to *computing* with the function in this low-rank format. For example, consider multiplying or adding two functions. In the discrete setting of a tensor obtained from pointwise evaluations of a function on a tensor product grid, these operations are undefined unless both functions have identical underlying discretizations. Otherwise, problem-specific interpolation techniques are needed in order to interpolate the functions onto the same grid. These interpolation procedures are potentially inefficient and can result in unnecessary

decompression of the underlying function. Similar procedures would be needed to compute identical parameterizations for the case of a tensor of coefficients. For example, suppose that two functions are parameterized by piecewise polynomials with different choices of knots: existing algorithms would require a projection or interpolation of both functions onto the same parameterization.

The limitations above arise from a *data structure* problem. When we represent a multiway array of function values in TT or QTT format, the connection between these values and the function itself is lost. In other words, one is left to create algorithms that work with discrete objects without any knowledge of where these objects came from. One must then inject additional knowledge in a post-processing stage to perform analysis. Consider, for example, the problem of integrating a multivariate function. Given function values on a tensor of input points, one could perform Monte Carlo integration, apply a Newton–Cotes rule, perform rational interpolation and then integrate, etc. Each of these choices involves assumptions about the properties of the underlying function that are not available in the discrete tensor representation itself.

Ideally, a data structure that combines *function values or parameters* and methods that *interpret* them, through both the compression and post-processing steps, are needed.

While each of the limitations and examples provided above can potentially be addressed independently, we seek an algorithmic framework that is general enough to overcome all of them. Such a framework requires both a new data structure that more closely follows the mathematical model (1), and an abstraction of discrete algorithms to the continuous case. Fortunately, Chebfun [23] and related efforts have formulated exactly such a computational paradigm. In fact, they have demonstrated the great success of such approaches for over ten years, for problems ranging from function approximation to solving PDEs. The rest of this paper describes how to apply this paradigm to low-rank functions of arbitrary dimension.

3. Continuous analogue of the tensor-train

Our proposed data structure to represent the functional tensor-train (1) stores each univariate function $f_k^{(\alpha_{k-1},\alpha_k)}$ *independently* and groups the functions associated with each input coordinate $k \in \{1,\ldots,d\}$ into matrix-valued functions $\mathcal{F}_k : \mathcal{X}_k \to \mathbb{R}^{r_{k-1}\times r_k}$, called cores. Note that no three-way arrays are required. The evaluation of a function in this format, using these continuous cores, involves multiplying matrix-valued functions:

$$f(x_1, x_2, \ldots, x_d) = \mathcal{F}_1(x_1)\mathcal{F}_2(x_2)\cdots\mathcal{F}_d(x_d). \tag{3}$$

In this section, we describe a new parameterization of this format and compare it to existing work.

Working with this mathematical object on a computer requires a finite parameterization. In our case, we parameterize (1) with a hierarchical structure. Beginning at the leaf level, each univariate function $f_k^{(\alpha_{k-1},\alpha_k)}$ is parameterized by $n_k^{(\alpha_{k-1},\alpha_k)} \in \mathbb{Z}^+$ parameters $\left(\theta_{kj}^{(\alpha_{k-1},\alpha_k)}\right)_{j=1}^{n_k^{(\alpha_{k-1},\alpha_k)}}$. Both linear parameterizations, e.g., a basis expansion

$$f_k^{(\alpha_{k-1},\alpha_k)}(x_k) = \sum_{j=1}^{n_k^{(\alpha_{k-1},\alpha_k)}} \theta_{kj}^{(\alpha_{k-1},\alpha_k)} \phi_{kj}^{(\alpha_{k-1},\alpha_k)}(x_k), \tag{4}$$

and nonlinear parameterizations are allowed. As an example of the latter, consider a piecewise constant approximation,

$$f_k^{(\alpha_{k-1},\alpha_k)}(x_k) = \begin{cases} \theta_{k(j+n/2)}^{(\alpha_{k-1},\alpha_k)} & \text{if } \theta_{kj}^{(\alpha_{k-1},\alpha_k)} \le x < \theta_{kj+1}^{(\alpha_{k-1},\alpha_k)} \\ 0 & \text{otherwise} \end{cases}; \tag{5}$$

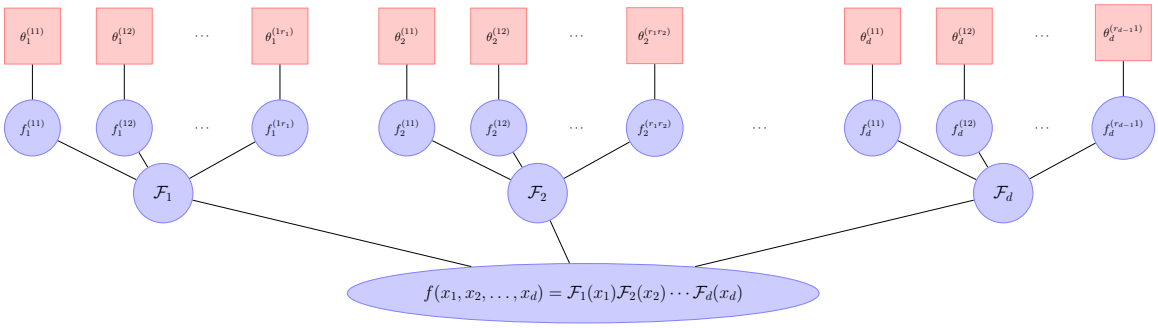


Figure 1: Hierarchical data structure for storing low-rank functions. Parameters are represented by red squares and functions are represented in blue. In contrast to previous approaches [31, 6], the parameters and parameterized form that represent each univariate function are stored separately, i.e., no tensor-product structure is imposed upon the **parameters** of the constituent univariate functions. Instead tensor-product structure is imposed upon the **univariate functions** themselves.

where the parameters are here a concatenation of the knot locations and the function value between any pair of knots. In practice, we employ even more adaptive nonlinear parameterizations—refining piecewise polynomial approximations, and the value of $n_k^{(\alpha_{k-1},\alpha_k)}$, independently for each univariate function. Similarly, the number of basis functions in the linear parameterization (4) can even vary among the univariate functions for a given x_k. In Section 5.1 we show a numerical example where both linear and nonlinear parameterizations can be used *simultaneously* for an input coordinate x_k.

Note that these parameterizations cannot be incorporated within the tensor-of-coefficients framework. As a result, our approach is more expressive precisely because it decouples the notion of "tensor products of functions" from the notion of "tensors of parameters." We also note that while any function *can* be represented using a complete tensor-product basis, this is seldom the *most efficient* choice. Such representations are often convenient for proving theoretical results, but they may be too computationally expensive in practice.

The second level of the hierarchy groups $f_k^{(\alpha_{k-1},\alpha_k)} : \mathcal{X}_k \to \mathbb{R}$ into a matrix-valued function $\mathcal{F}_k : \mathcal{X}_k \to \mathbb{R}^{r_{k-1}\times r_k}$, i.e.,

$$\mathcal{F}_k(x_k) = \begin{bmatrix} f_k^{(1,1)}(x_k) & \cdots & f_k^{(1,r_k)}(x_k) \\ \vdots & & \vdots \\ f_k^{(r_{k-1},1)}(x_k) & \cdots & f_k^{(r_{k-1},r_k)}(x_k) \end{bmatrix}. \tag{6}$$

The third level, analogous to the discrete setting, represents the continuous tensor-train with a list of matrix-valued functions, $\mathbb{FT}(f) = (\mathcal{F}_k : \mathcal{X}_k \to \mathbb{R}^{r_{k-1}\times r_k})_{k=1}^d$, with $r_0 = r_d = 1$ and $r_k \in \mathbb{Z}^+$ for $k = 2, \ldots, d-1$, as described by (3).

A graph representation of this hierarchy is shown in Figure 1. There are *no tensor data structures* in this representation; there is no underlying tensorized grid or tensor of coefficients, either explicitly or in a decomposed form. Next we demonstrate conditions under which our representation is equivalent to (2) and when it is advantageous.

3.1. Comparison with a tensor of coefficients

We now compare the above approach with the typical alternative of representing multivariate functions via a tensor-train decomposition of coefficients. In particular, we demonstrate conditions under which the low-rank representation (1) and a low-rank representation of a coefficient tensor in (2) lead to equivalent approximations. This comparison will serve to highlight the advantages of directly computing with (1) instead of (2).

3.1.1. Conditions of equivalence

Equivalence between the two models arises when the following two conditions are met:

1. Each $f_k^{(\alpha_{k-1},\alpha_k)}$ is parameterized linearly with a fixed number of basis functions in (4).
2. An *identical* basis is used for all $f_k^{(\alpha_{k-1},\alpha_k)}$ within a dimension k, i.e., $n_k^{(\alpha_{k-1},\alpha_k)} = n_k$ and $\phi_{kj}^{(\alpha_{k-1},\alpha_k)} = \phi_{kj}$ for all $\alpha_{k-1} = 1,\dots,r_{k-1}$, $\alpha_k = 1,\dots,r_k$, and $j = 1\dots,n_k$.

To demonstrate this fact, consider a tensor $\boldsymbol{\mathcal{A}}_k \in \mathbb{R}^{r_{k-1}\times n_k \times r_k}$ such that

$$\boldsymbol{\mathcal{A}}_k[:,j,:] = \begin{bmatrix} \theta_{kj}^{(1,1)} & \dots & \theta_{kj}^{(1,r_k)} \\ \vdots & & \vdots \\ \theta_{kj}^{(r_{k-1},1)} & \dots & \theta_{kj}^{(r_{k-1},r_k)} \end{bmatrix},$$

for $j = 1,\dots,n_k$. This tensor is the kth core of a TT decomposition of the coefficient tensor in (2). The relationship between the continuous core and the discrete TT core is now

$$\mathcal{F}_k(x_k) = \sum_{j=1}^{n_k} \boldsymbol{\mathcal{A}}_k[:,j,:]\phi_{kj}(x_k),$$

where the same basis function is used for every slice of $\boldsymbol{\mathcal{A}}_k$. In other words, one contracts the TT core with the basis functions along the second mode. This connection yields the equivalence:

$$f(x_1,\dots,x_2) = \mathcal{F}_1(x_1)\cdots\mathcal{F}_d(x_d) = \sum_{i_1=1}^{n_1}\cdots\sum_{i_d=1}^{n_d} \boldsymbol{\mathcal{A}}_1[:,i_1,:]\cdots\boldsymbol{\mathcal{A}}_d[:,i_d,:]\phi_{1i_1}(x_1)\cdots\phi_{di_d}(x_d)$$

3.1.2. FT as a sparse storage structure

One advantage of our approach is that, under the equivalence conditions described above, it can be viewed as a *sparse* storage scheme for the TT cores $\boldsymbol{\mathcal{A}}_k$. We demonstrate this property on two examples.

The first example involves representing additive functions. Additive functions are extremely common within high-dimensional modeling [34, 35, 36]. As such, it is useful to be able to represent these functions in low-rank format. An additive function has TT-ranks $r_1 = \cdots = r_{d-1} = 2$, which can be seen from the following decomposition:

$$f(x_1,x_2,\dots,x_d) = f_1(x_1) + f_2(x_2) + \cdots + f_d(x_d) = [f_1(x_1)\ 1]\begin{bmatrix} 1 & 0 \\ f_2(x_2) & 1 \end{bmatrix}\cdots\begin{bmatrix} 1 \\ f_d(x_d) \end{bmatrix}.$$

Assume that $n_k = n$ terms are required to parameterize each f_k in a basis expansion. Then the TT cores of the coefficient tensor store $4n + 4(d-2)n$ floating point numbers. The constants 1 and 0 are essentially represented with n parameters, when in reality they can be represented with a single parameter. Standard techniques in the literature do not exploit this structure of the TT cores. If we instead let $n_k^{(\alpha_{k-1},\alpha_k)}$ vary for each function in each core (violating the second equivalence condition), only $2(n+1) + (d-2)(n+3)$ floating point numbers would require storage. In other words, in the limit of large d, four times less memory is required for our low-rank format than for a TT decomposition of coefficients.

A more impressive example is a quadratic function. A quadratic function has d^2 parameters, and therefore one would expect any low rank storage format only to require storing $\mathcal{O}(d^2)$ parameters. However, this storage requirement can only be achieved by a sparse storage scheme for the TT cores in (4), which is naturally supported by the format that we propose. To be more clear, one parameterization of a quadratic function

$$f(x_1,x_2,\dots,x_d) = x^T\mathbf{A}x, \quad x = [x_1\ x_2\ \cdots\ x_d]$$

in low-rank format has continuous cores with the following structure:[3]

$$\mathcal{F}_1(x_1) = \begin{bmatrix} a_{1,1}x_1^2 & a_{1,2}x_1 & \dots & a_{1,d}x_1 & 1 \end{bmatrix}, \qquad \mathcal{F}_d(x_d) = \begin{bmatrix} 1 & x_d & a_{d,d}x_d^2 \end{bmatrix}$$

$$\mathcal{F}_k(x_k) = \begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 & 0 \\ x_k & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 & 0 \\ \vdots & & \ddots & \ddots & & & \vdots \\ \vdots & & & \ddots & \ddots & & \vdots \\ \vdots & \vdots & & & & \ddots & \ddots \\ 0 & 0 & 0 & 0 & \cdots & 1 & 0 \\ a_{k,k}x_k^2 & a_{k,k+1}x_k & a_{k,k+2}x_k & \cdots & a_{k,d-1}x_k & a_{k,d}x_k & 1 \end{bmatrix}.$$

The TT representation of each core requires storing $d \times 3 \times d$ floating point values, denoting the coefficients of constant, linear, and quadratic basis functions. The data structure that we propose instead stores each element of $\mathcal{F}_k(x_k)$ separately, and there are $\mathcal{O}(d)$ nonzero elements. Thus the total storage of our proposed structure is $\mathcal{O}(d^2)$, versus $\mathcal{O}(d^3)$ for the TT version.

Similar examples can be constructed for functions of higher polynomial degree. The main takeaway from these examples is that the data structure that we propose is flexible enough to take advantage of core sparsity. Thus, even when it is appropriate to model a function using (4), we can achieve better compression using (1). Secondly, we note that while it is true that one could potentially use a sparse storage mechanism for the TT cores to achieve similar benefits, our proposed framework naturally supports this structure with no additional modifications.

3.1.3. Nonlinear parameterizations for more accurate and efficient compression

This section provides an example of a function that, while representable in a low-rank format, cannot be readily described by a decomposition of a coefficient tensor. In particular, we demonstrate the additional expressivity afforded by modeling functions using (1).

Consider the rank-2 function

$$f(x_1, x_2) = \begin{cases} 0 & \text{if } \frac{1}{4} \le x_1 \le \frac{3}{4} \text{ and } \frac{1}{4} \le x_2 \le \frac{3}{4} \\ \sin(x_1)\cos(x_2) & \text{otherwise} \end{cases}, \quad (x_1, x_2) \in [0,1]^2.$$

The rank-2 representation of this function is

$$f(x_1, x_2) = \sin(x_1)\cos(x_2) - \chi_{1/4 \le x_1 \le 3/4} \sin(x_1) \chi_{1/4 \le x_2 \le 3/4} \cos(x_2), \tag{7}$$

where χ_A denotes an indicator function on the set A.

This function is a model problem for situations when there is a "hole" in an otherwise tensor-product space. For example, this might be a rectangle around which a fluid is flowing, an obstacle around which a robot must navigate in a motion planning problem, or a failure region for some black-box model. Sometimes, the location of this region is not known *a priori*, but discovered through interacting with the function. In either case, we would like our function to be zero for inputs that are inside of the "hole."

[3]For clarity of presentation we have written these cores as being $d \times d$, i.e., a rank d representation. A more compact representation is possible with cores of varying sizes and the maximum core size being $d/2 \times d/2$. The asymptotic results that we present would not change in this case, however.

When the boundary is exactly known, this case can easily be handled using the tensor-of-coefficients formulation (4). First one defines a set of basis functions for the "good" region; in this case suppose we use $\phi_{11} = \sin(x_1)$. Then for each basis function, one simply adds another basis function according to the rule $\phi_{1(2j)} = \chi_A \phi_{1j}$; in this case we would use $\phi_{12} = \chi_{1/4 \le x_1 \le 3/4} \sin(x_1)$. Thus, one simply doubles the number of basis functions in each dimension.

In practice, however, the exact location of such failure/zero regions may not be known. Instead, it must be found through the process of evaluating f. One way to model such regions is to adapt the knot locations of a piecewise polynomial; see (5). This situation immediately becomes one where the parameters of each representation are no longer coefficients of a tensor product basis. Section 5.1 will provide a numerical example of such a procedure.

3.2. Errors introduced through parameterization

In practice, approximation of each univariate function $f_k^{(ij)}$—via a finite-dimensional parameterization—introduces errors in the overall approximation of f. In our implementation, these errors are incurred during the cross approximation procedure (Section 4), when approximating selected univariate fibers of f. The following result relates the approximation error of each univariate function in (1) to the approximation error in f.

Theorem 1 (Parameterization error)**.** *Assume that each of the univariate functions in* (1) *is bounded according to* $\left|f_k^{(ij)}(x_k)\right| \le C$, *for some* $1 \le C < \infty$. *Let* $\hat{f}_k^{(ij)}(x_k)$ *be univariate functions such that* $|f_k^{(ij)}(x_k) - \hat{f}_k^{(ij)}(x_k)| \le \epsilon C$ *for* $\epsilon d < 1/e$. *Then the difference between the multivariate approximation*

$$\hat{f}(x_1, \ldots, x_d) = \sum_{\alpha_0=1}^{r_0} \sum_{\alpha_1=1}^{r_1} \cdots \sum_{\alpha_d=1}^{r_d} \hat{f}_1^{(\alpha_0,\alpha_1)}(x_1) \hat{f}_2^{(\alpha_1,\alpha_2)}(x_2) \cdots \hat{f}_d^{(\alpha_{d-1},\alpha_d)}(x_d) \tag{8}$$

and (1) *is bounded by*

$$\left|f(x) - \hat{f}(x)\right| \le ed^2 C^d r^{d-1} \epsilon. \tag{9}$$

Proof. Let $\Delta_k^{(\alpha_{k-1}\alpha_k)}(x) = f_k^{(\alpha_{k-1}\alpha_k)}(x) - \hat{f}_k^{(\alpha_{k-1}\alpha_k)}(x)$. First, substitute the definition of each $\hat{f}_k^{(ij)}$ into (8):

$$\begin{aligned}
\hat{f}(x) &= \sum_{\mathbf{1} \le \boldsymbol{\alpha} \le \boldsymbol{r}} \left(f_1^{(\alpha_0\alpha_1)} - \Delta_1^{(\alpha_0\alpha_1)}\right) \cdots \left(f_d^{(\alpha_{d-1}\alpha_d)} - \Delta_d^{(\alpha_{d-1}\alpha_d)}\right) \\
&= \sum_{\mathbf{1} \le \boldsymbol{\alpha} \le \boldsymbol{r}} \left[\sum_{\boldsymbol{k} \le \mathbf{1}_d} \left[f_1^{(\alpha_0\alpha_1)}\right]^{k_1} \left[-\Delta_1^{(\alpha_0\alpha_1)}\right]^{1-k_1} \cdots \left[f_d^{(\alpha_{d-1}\alpha_d)}\right]^{k_d} \left[-\Delta_d^{(\alpha_{d-1}\alpha_d)}\right]^{1-k_d} \right],
\end{aligned}$$

where the second equality follows from the binomial theorem.[4] Computing the difference between this expression and f, the term with $\boldsymbol{k} = \mathbf{1}_d$ drops out, leaving the rest of the expression

$$f(x) - \hat{f}(x) = - \sum_{\mathbf{1} \le \boldsymbol{\alpha} \le \boldsymbol{r}} \left[\sum_{\boldsymbol{k} \le \mathbf{1}_d, \boldsymbol{k} \ne \mathbf{1}_d} \left[f_1^{(\alpha_0\alpha_1)}\right]^{k_1} \left[-\Delta_1^{(\alpha_0\alpha_1)}\right]^{1-k_1} \cdots \left[f_d^{(\alpha_{d-1}\alpha_d)}\right]^{k_d} \left[-\Delta_d^{(\alpha_{d-1}\alpha_d)}\right]^{1-k_d} \right].$$

[4]Here we have used multi-index notation to simplify the expression. In multi-index notation we assume $\boldsymbol{\alpha} = (\alpha_0, \ldots, \alpha_d)$, $\boldsymbol{r} = (r_0, \ldots, r_d)$, $\boldsymbol{k} = (k_0, \ldots, k_d)$ and $\mathbf{1}_d = (1, \ldots, 1)$. The notation $\boldsymbol{\alpha} \le \boldsymbol{r}$ implies $\alpha_i \le r_i$ for every $i = 0, \ldots, d$.

Now using the bound C on the univariate functions and ϵC on the $\Delta_i^{(\alpha_{i-1}\alpha_i)}$ results in

$$\begin{aligned}
|f(x) - \hat{f}(x)| &\leq \left| \sum_{\mathbf{1} \leq \boldsymbol{\alpha} \leq \boldsymbol{r}} \left[\sum_{\boldsymbol{k} \leq \mathbf{1}_d, \boldsymbol{k} \neq \mathbf{1}_d} C^{k_1} \left[\epsilon C\right]^{1-k_1} \cdots C^{k_d} \left[\epsilon C\right]^{1-k_d} \right] \right| \\
&= \sum_{\mathbf{1} \leq \boldsymbol{\alpha} \leq \boldsymbol{r}} \sum_{\boldsymbol{k} \leq \mathbf{1}_d, \boldsymbol{k} \neq \mathbf{1}_d} \left[\prod_{l=1}^{d} \left(C^{k_l} C^{1-k_l} \right) \right] \epsilon^{d - \sum_l k_l} \\
&= \sum_{\mathbf{1} \leq \boldsymbol{\alpha} \leq \boldsymbol{r}} C^d \sum_{\boldsymbol{k} \leq \mathbf{1}_d, \boldsymbol{k} \neq \mathbf{1}_d} \epsilon^{d - \sum_l k_l},
\end{aligned}$$

where in the second equality we combined exponents with common bases. Using $\sum_{\mathbf{1} \leq \boldsymbol{\alpha} \leq \boldsymbol{r}} C^d \leq C^d r^{d-1}$, where $r = \max_{1 \leq i \leq d-1} r_i$, we obtain

$$|f(x) - \hat{f}(x)| \leq C^d r^{d-1} \sum_{\boldsymbol{k} \leq \mathbf{1}_d, \boldsymbol{k} \neq \mathbf{1}_d} \epsilon^{d - \sum_l k_l}.$$

The k_l terms only enter through a summation. Thus we can simplify the sum over $\boldsymbol{k}$ by summing over possible values for $\sum_{l=1}^{d} k_l$. This sum ranges from zero to $d-1$ and each particular sum is obtained when a certain number of k_l are non-zero. Thus we can convert this summation to a sum over d choose k combinations for $k = 0, \ldots, d-1$ to obtain

$$\begin{aligned}
|f(x) - \hat{f}(x)| &\leq C^d r^{d-1} \sum_{l=0}^{d-1} \binom{d}{l} \epsilon^{d-l} = C^d r^{d-1} \sum_{l=0}^{d-1} \binom{d}{d-l} \epsilon^{d-l} \leq C^d r^{d-1} \sum_{l=0}^{d-1} \left(ed\epsilon\right)^{d-l} \\
&\leq C^d r^{d-1} \sum_{k=1}^{d} \left(ed\epsilon\right)^{k} \leq d C^d r^{d-1} ed\epsilon \leq ed^2 C^d r^{d-1} \epsilon,
\end{aligned}$$

where in the second inequality we have used an upper bound for $\binom{d}{d-l}$ and in the fourth inequality we used the fact that $d\epsilon < 1/e$. □

This bound formalizes what may be intuitively obvious: the error in a single univariate function is multiplied against all combinations of all the univariate functions of the other input coordinates. Therefore, the impact of any error in a single univariate function can grow exponentially with dimension. At the same time, $|f| \leq C^d r^{d-1}$ is the only bound on f itself that we can obtain without further assumptions on the interactions among the univariate functions $f_k^{(\alpha_{k-1}\alpha_k)}$. This fact can be simply seen by setting $\Delta_k^{(\alpha_{k-1}\alpha_k)} = f_k^{(\alpha_{k-1}\alpha_k)}$ in the proof above. Our result can thus be interpreted, perhaps more naturally, as

$$\left| f(x) - \hat{f}(x) \right| \leq ed^2 \epsilon \max_x \left[f(x) \right]. \tag{10}$$

This bound suggests that, to maintain a fixed bound on the relative L^∞ error, a conservative approximation scheme should decrease the error threshold ϵ for the univariate functions comprising (1) quadratically with dimension. In practice, though, we usually see good approximation behavior with respect to dimension even with fixed thresholds. Additional assumptions on the interactions between each of the univariate functions can potentially lead to less restrictive requirements for ϵ.

4. Constructing low-rank approximations

Next we describe how to construct FT approximations with continuous analogues of cross approximation and rounding algorithms [37]. There are four important differences between the discrete and continuous versions:

- The continuous algorithms interpret tensor fibers as univariate functions obtained by fixing all but one input coordinate;
- Each tensor fiber/univariate function is approximated *adaptively* to a particular tolerance as it is encountered, introducing a new source of error;
- Optimization within an approximate maximum volume procedure for selecting fibers is performed over a *continuous* rather than *discrete* variable; and
- Rounding algorithms use *continuous* QR decompositions, implicitly defining a continuous, rather than a discrete, inner product.

Note that the algorithms we present are also applicable to low-rank approximation techniques such as DMRG-cross [38], in which case, rather than tensor fibers and univariate functions, we would have tensor slices and bivariate functions. In other words, our essential contributions lie in placing these existing algorithms in the context of Chebfun-style continuous computation. Furthermore, if interpreted in the tensor-of-coefficients context, they are the first to provide an *adapt-to-tolerance* procedure for both ranks *and* fibers.

4.1. Cross approximation

In this section, we describe a continuous analogue of a cross approximation algorithm for compressing multiway arrays. Cross approximation is a dimension-sweeping algorithm that seeks to interpolate a tensor using a basis consisting of finite set of its fibers. In the continuous context, we are approximating a multivariate *function* using certain univariate functions formed by fixing all but a single input variable.

To illustrate how continuous cross approximation differs from discrete cross approximation, it is sufficient to consider a bivariate function approximation problem. Extensions to higher dimensions are made using the same types of modifications we describe here to a multivariate cross approximation algorithm, for example [38, Algorithm 1]. In two dimensions, cross approximation represents a low-rank function by its CUR decomposition [39, 40, 41]:

$$
\begin{aligned}
f(x,y) &= \mathcal{C}(x)\mathbf{F}^{\dagger}\mathcal{R}(y) \\
\mathcal{C}(x) &= \begin{bmatrix} f\left(x, y^{(1)}\right) & f\left(x, y^{(2)}\right) & \cdots & f\left(x, y^{(r)}\right) \end{bmatrix} \quad \mathcal{R}(y) = \begin{bmatrix} f\left(x^{(1)}, y\right) \\ f\left(x^{(2)}, y\right) \\ \vdots \\ f\left(x^{(r)}, y\right) \end{bmatrix} \\
\mathbf{F}[i,j] &= f\left(x^{(i)}, y^{(j)}\right) \quad i,j = 1,\ldots,r,
\end{aligned}
$$

where $\mathbf{F}^{\dagger}$ indicates the pseudoinverse of $\mathbf{F}$. The difference between continuous and discrete CUR decompositions is that columns and rows define a "quasimatrix" (vector-valued function)[5] rather than a matrix. In this bivariate case, the fibers are rows and columns. In the more general multivariate case there are fibers corresponding to each dimension of the tensor. As a result, instead of the three-way arrays that form the discrete TT cores, we obtain matrix-valued functions (3).

In the TT (and DMRG-)cross approximation algorithms, fibers associated with each input coordinate are chosen sequentially to approximately maximize the volume of the skeleton matrix $\mathbf{F}$ formed by their intersection. We refer to [42, 43, 44] for results regarding the quasioptimality of choosing an actual maximum volume submatrix. This algorithm can be abstracted to the continuous case by generalizing the concept of tensor fibers to encompass the case of univariate functions

[5] Called a quasimatrix because it corresponds to a matrix of infinite rows and n columns; see, e.g., [22, 25].

obtained by fixing all but one input variable. With this abstraction, continuous cross approximation is almost identical (algebraically) to the discrete case.

Using the bivariate setting for simplicity, we now point out aspects of the algorithm that differ between the discrete and continuous cases. As mentioned above, these algorithmic changes are directly applicable to higher dimensional contexts as well. Using the notation of matrix-valued functions, pseudocode for our continuous cross approximation algorithm is provided in Algorithm 1. The first difference can be seen in Lines 5, 10, and 14, where column and row quasimatrices are formed. In the context of multivariate functions, we do not have access to a discrete vector representing a row. Instead we adaptively approximate each column and row via an *online* procedure. In other words, we only generate an approximation of a tensor fiber when the cross approximation algorithm dictates that we need that particular fiber.

The advantage in expressivity of our approach arises during this approximation step. In particular, we adapt the function at the (local) *fiber level* rather than the (global) *dimension* level. Therefore local features occuring along one fiber will not increase the computational effort to approximate another. In Section 5.2, we will provide an example wherein global polynomials are used to represent regions where a function is smooth and adaptive piecewise polynomials are used in a region where a function has discontinuities.

The second difference involves Lines 7 and 11. In these lines a QR decomposition of the rows and columns is performed. This step is used to increase the stability of the algorithm; see [37]. Because we are operating with quasimatrices, we use a *continuous* QR decomposition [24, 25, 14]. Essentially, the discrete and continuous QR decompositions differ with respect to the inner product used to define orthonormality. In the context of a tensor formed by discretizing a function, the discrete inner product is typically only an *approximation* of the continuous inner product. Here, we see that the continuous computation paradigm of Chebfun enables maintaining a notion of orthonormality that is consistent with the original function space, as compared to a discretized TT/QTT approach.

The final difference involves the `dominant` routine. This routine chooses the set of fibers to be used within a dimension during the next sweep across dimensions. We have developed a continuous analogue of `dominant` that operates on matrix-valued functions and that is applicable to both the bivariate and multivariate cases. This algorithm is described in the next section.

Finally, we point out the distinction between this row-column alternating algorithm and *adaptive* cross approximation algorithms as in [45]. Adaptive cross approximation algorithms for the approximation of bivariate functions build up a set of row and column indices by sequentially choosing points which maximize the volume. In other words, they do not require one to prescribe a rank, nor do they require one to evaluate entire function fibers. These algorithms, which are equivalent to LU decomposition with complete pivoting, can use two-dimensional optimization methods to seek the optimal locations for function evaluation (which become the pivots). This methodology is attractive for bivariate problems but difficult to extend to the multivariate case, because it would require us to optimize over locations in d dimensions.

4.2. Approximate maxvol through dominant submatrices

Algorithm 1 uses the `dominant` algorithm to select the next fiber locations. In particular, the goal of `dominant` is to find indices that correspond to a *dominant* submatrix. For the case of matrix-valued functions, the definitions of a submatrix and a dominant submatrix are below.

Definition 1 (Submatrix of a matrix-valued function)**.** *A submatrix of a matrix-valued function* $\mathcal{A} : \mathcal{X} \to \mathbb{R}^{n\times r}$ *is the matrix* $\bar{\mathbf{A}}_{\text{mat}} \in \mathbb{R}^{r\times r}$ *obtained by fixing a set of* r *tuples* $\{(i_1, x_1), (i_2, x_2), \ldots, (i_r, x_r)\}$*, where* $(i_k, x_k) \in \{1, \ldots, n\} \times \mathcal{X}$ *for* $k, l = 1 \ldots r$ *and* $(i_k, x_k) \neq (i_l, x_l)$ *for* $k \neq l$*, such that* $\bar{\mathbf{A}}_{\text{mat}}[k, j] = \mathcal{A}[i_k, j](x_k)$.

Definition 2 (Dominant submatrix)**.** *A dominant submatrix of a matrix-valued function* $\mathcal{A} : \mathcal{X} \to \mathbb{R}^{n\times r}$ *is any submatrix* $\bar{\mathbf{A}}_{\text{mat}}$ *such that for all values* $(i, x, k) \in \{1, \ldots, n\} \times \mathcal{X} \times \{1, \ldots, r\}$ *the matrix-valued function* $\mathcal{B} = \mathcal{A}\bar{\mathbf{A}}_{\text{mat}}^{-1}$ *is bounded as* $|\mathcal{B}[i, k](x)| \leq 1$.

Algorithm 1 Continuous cross approximation using fiber–adaptive approximations

Input: Bivariate function $f : [a,b] \times [c,d] \to \mathbb{R}$; rank estimate r; initial column fiber indices $\boldsymbol{y} = [y^{(1)}, y^{(2)}, \ldots, y^{(r)}]$; stopping tolerance $\delta_{\text{cross}} > 0$; adaptive approximation scheme $\texttt{approx-fiber}(f_k, \epsilon_{\text{approx}})$; fiber approximation tolerance ϵ_{approx}

Output: $\boldsymbol{x}, \boldsymbol{y}$ such that $\mathbf{F} \in \mathbb{R}^{r \times r}$, with $\mathbf{F}[i,j] = f(\boldsymbol{x}[i], \boldsymbol{y}[j])$, has "large" volume

1: $\delta = \delta_{\text{cross}} + 1$
2: $f^{(0)} = 0$
3: $k = 1$
4: Initialize columns $\mathcal{C} : [a,b] \to \mathbb{R}^{1 \times r}$
5: $\mathcal{C}(x)[1,i] = \texttt{approx-fiber}(f(x, y^{(i)}), \epsilon_{\text{approx}})$ for $i = 1 \ldots r$
6: **while** $\delta \leq \delta_{\text{cross}}$ **do**
7: $\quad \mathcal{Q}\mathbf{T} = \texttt{qr}(\mathcal{C})$ # QR decomposition of a matrix-valued function
8: $\quad \boldsymbol{x} = \texttt{dominant}(\mathcal{Q})$
9: $\quad$ Initialize rows $\mathcal{R} : [c,d] \to \mathbb{R}^{r \times 1}$
10: $\quad \mathcal{R}[j,1](y) = \texttt{approx-fiber}(f(x^{(j)}, y), \epsilon_{\text{approx}})$ for $j = 1 \ldots r$
11: $\quad \mathcal{Q}\mathbf{T} = \texttt{qr}(\mathcal{R}^T)$
12: $\quad \boldsymbol{y} = \texttt{dominant}(\mathcal{Q})$
13: $\quad \hat{\mathbf{Q}} = [\mathcal{Q}[1,1](y_1)\ \mathcal{Q}[1,2](y_2)\ \ldots\ \mathcal{Q}[1,r](y_r)]$
14: $\quad \mathcal{C}(x)[1,i] = \texttt{approx-fiber}(f(x, y^{(i)}), \epsilon_{\text{approx}})$ for $i = 1 \ldots r$
15: $\quad f^{(k)}(x,y) = \mathcal{C}(x)\hat{\mathbf{Q}}^{\dagger}\mathcal{Q}^T(y)$
16: $\quad \delta = \|f^{(k)} - f^{(k-1)}\| / \|f^{(k)}\|$
17: $\quad k = k + 1$

Pseudocode for computing a dominant submatrix of a matrix-valued function is given in Algorithm 2;[6] it mirrors the algorithm provided in [46] for "tall and skinny" matrices. The algorithm seeks a submatrix which is dominant, because a dominant submatrix has a volume that is not much smaller than that of the maximum volume submatrix, and finding the actual maximum volume submatrix is computationally intractable. For a further discussion of this approach we refer the reader to [42, 43, 46].

The algorithm works by swapping "rows" of $\mathcal{A}$ (these are now specified by (index, x-value) combinations) until all the elements of $\mathcal{B}$ are less than 1. This is exactly what the operations in Lines 5 and 8 of Algorithm 2 are doing. To find an initial set of linearly independent "rows" of $\mathcal{A}$, the algorithm first performs a continuous pivoted LU decomposition, denoted by `lu`, yielding pivots $\boldsymbol{\alpha} = \{(i_1, x_1), \ldots, (i_r, x_r)\}$.

There exist two differences between the continuous and discrete versions of these algorithms. The first is that we utilize a *continuous* pivoted LU decomposition [25, 14]. This approach allows the pivots to range freely over the input space, rather than being restricted to lie on a discretized set of locations. The second difference is the optimization problem specified in Line 5 of Algorithm 2. First, it is a *continuous* optimization problem in x, allowing us to search over the entire space $\mathcal{X}$; in a tensor arising through discretization and compression in TT/QTT format, this maximization can only occur over the discretized points. If the discretization missed a local feature, then the approximation could suffer. Secondly, when $\mathcal{A} : [a,b] \to \mathbb{R}^{n \times r}$, the continuous optimization problem involves a one-dimensional decision variable and can exploit the structure of the scalar-valued functions

[6] We construct the algorithm for a matrix-valued function in order to generalize to multivariate cross approximation. The bivariate case is a special case where the matrix-valued function has a single row. The matrix-valued function arises in [38, Lines 4, 7, 8, 21 in Alg. 1]. In that discrete algorithm, reshapings of TT cores are formed. In our continuous case, these reshapings need not be formed. Instead they require treating each continuous core (6) as a matrix whose rows are indexed by (i_k, x_k) pairs.

Algorithm 2 `dominant`: Dominant submatrix of a matrix-valued function

Input: $\mathcal{A} : \mathcal{X} \to \mathbb{R}^{n \times r}$, a matrix-valued function.
Output: $\boldsymbol{\alpha} = \{(i_1, x_1,) \dots (i_r, x_r)\}$ such that $\bar{\mathbf{A}}_{\text{mat}}$ is dominant

$\{L, \mathbf{U}, \boldsymbol{\alpha}\} = \texttt{lu}(A)$ # LU decomposition of the matrix-valued function
$\delta = 2$
while $\delta > 1$ **do**
    $\bar{\mathbf{A}}_{\text{mat}} \leftarrow$ submatrix defined by $\boldsymbol{\alpha}$
    $x^*, i^*, j^* = \arg\max_{(x,i,j)} \mathcal{A}[i, :](x)\bar{\mathbf{A}}^{\dagger}_{\text{mat}}[:, j]$
    $\delta = \mathcal{A}[i^*, :](x^*)\bar{\mathbf{A}}^{\dagger}_{\text{mat}}[:, j^*]$
    if $\delta > 1$ **then**
        $x_{j^*} = x^*$, $i_{j^*} = i^*$

that comprise $\mathcal{A}$. For example, if these scalar-valued functions are represented with an expansion of orthonormal polynomials, then the maximization reduces to an eigenvalue problem [47] and therefore does not require the use of optimization approaches that might only converge to local minima. If these functions are piecewise polynomials, then we search over each piece; if they are piecewise linear, then we search over each node. The benefit we gain is *exploration* of the continuous space.

4.3. Rounding

The cross approximation procedure discussed above relies on specifying the ranks r_k *a priori*. In this section, we discuss a procedure called *rounding* that will be used to find ranks adaptively and that is useful for reducing the rank of a function formed by the multiplication and addition operations. The idea is that if a representation with certain ranks can be well approximated by a representation of smaller ranks, then we have overestimated the ranks used in the cross approximation algorithm and can be confident about its results.

Rounding begins with a low-rank representation and aims to generate an approximation with the smallest ranks that is accurate to a specified relative error ϵ. This is useful not only as a way of verifying the ranks used in cross approximation, but also because the ranks of (1) may initially be higher than necessary, depending on how it was created. We now describe a continuous rounding procedure, similar to the procedure for discrete TT representations in [30, Algorithm 2], with the primary distinction involving the notions of left and right orthogonality. In the discrete case, these notions refer to orthogonality in certain reshapings of the three-dimensional TT cores. In our continuous case, these notions refer to orthogonality between vector-valued functions that are the rows or columns of the continuous cores.

Rounding follows directly from the definition of the ranks of the unfoldings of the function f. An unfolding, or $k-$*separated* representation, of f is a grouping of the first k variables and the last $d-k$ variables to form a bivariate representation of a multivariate function. This representation is denoted by the superscript k

$$f^k : \mathcal{X}_{\leq k} \times \mathcal{X}_{>k} \to \mathbb{R}, \quad \text{such that } f^k(\{x_1, \dots, x_k\}, \{x_{k+1}, \dots, x_d\}) = f(x_1, \dots, x_d), \tag{11}$$

where $\mathcal{X}_{\leq k} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_k$ and $\mathcal{X}_{>k} = \mathcal{X}_{k+1} \times \cdots \times \mathcal{X}_d$.

Suppose that we start with the first unfolding of a function

$$f^1(x_1, x_{>1}) = \mathcal{F}_1(x_1)\left[\mathcal{F}_2(x_2)\mathcal{F}_3(x_3)\dots\mathcal{F}_d(x_d)\right] = \mathcal{F}_1(x_1)\mathcal{V}_1(x_{>1}),$$

where $\mathcal{V}_1 : \mathcal{X}_{>1} \to \mathbb{R}^{r_1 \times 1}$. This equation expresses f^1 in a rank r_1 low rank representation. One can then use standard algorithms to compress this representation further [48]; for instance we compress

f^1 via a truncated SVD. To this end, we first perform two QR decompositions of matrix-valued functions

$$\mathcal{F}_1(x_1) = \mathcal{Q}_1(x_1)\mathbf{R}_1 \quad \text{and} \quad \mathcal{V}_1^\top(x_{>1}) = \widetilde{\mathcal{Q}}_1(x_{>1})\widetilde{\mathbf{R}}_1, \text{ such that } f^1 = \mathcal{Q}_1\mathbf{R}_1\widetilde{\mathbf{R}}_1^\top\widetilde{\mathcal{Q}}_1^\top. \tag{12}$$

Then, we calculate the truncated SVD of $\mathbf{R}_1\widetilde{\mathbf{R}}_1^\top \simeq \mathbf{U_1}\mathbf{D_1}\mathbf{V_1}^\top$, where $\mathbf{U}_1 \in \mathbb{R}^{r_1\times\hat{r}_1}$, $\mathbf{V}_1^\top \in \mathbb{R}^{\hat{r}_1\times r_1}$, $\mathbf{D_1} \in \mathbb{R}^{\hat{r}_1\times\hat{r}_1}$ is a diagonal matrix, and the truncation level $\hat{r}_1 \leq r_1$ is chosen to obtain an approximation

$$g_1 = \underbrace{\mathcal{Q}_1\mathbf{U_1}}_{\mathcal{F}_1(\hat{x}_1)} \underbrace{\left[\mathbf{D_1}\mathbf{V_1}^\top\widetilde{\mathcal{Q}}_1^\top\right]}_{\mathcal{F}_2(\hat{x}_2)\mathcal{F}_3(\hat{x}_3)\ldots\mathcal{F}_d(\hat{x}_d)},$$

with error $\|f - g_1\| \leq \delta$. $\mathcal{F}_1(\hat{x}_1)$ now has size $1 \times \hat{r}_1$. Now that we have reduced the number of columns of the first core and the number of rows of the second core from r_1 to $\hat{r}_1$, we move on to the second unfolding of g_1 using the updated cores. Again, a truncated SVD is performed on this unfolding to reduce the rank from r_2 to $\hat{r}_2$. In this case we have

$$g_1^2(x_{\leq 2}, x_{>2}) = \left[\hat{\mathcal{F}}_1(x_1)\hat{\mathcal{F}}_2(x_2)\right][\mathcal{F}_3(x_3)(x_3)\ldots\mathcal{F}_d(x_d)(x_d)] = \mathcal{U}_2\mathcal{V}_2,$$
$$\mathcal{U}_2(x_{\leq 2}) = \mathcal{Q}_2(x_{\leq 2})\mathbf{R}_2, \quad \mathcal{V}_2^\top(x_{>2}) = \widetilde{\mathcal{Q}}_2(x_{>2})\widetilde{\mathbf{R}}_2$$

where $\mathcal{U}_2 : \mathcal{X}_2 \to \mathbb{R}^{1\times r_2}$ is the matrix-valued QR of the left cores and $\mathcal{V}_2 : \mathcal{X}_2 \to \mathbb{R}^{r_2\times 1}$ and is the matrix-valued QR of the right cores. These QR decompositions are used to obtain a truncated SVD:

$$\mathbf{R}_2\widetilde{\mathbf{R}}_2^\top \simeq \mathbf{U_2}\mathbf{D_2}\mathbf{V_2}^\top, \qquad g_2 = \underbrace{\mathcal{Q}_2\mathbf{U_2}}_{\hat{\mathcal{F}}_1(x_1)\hat{\mathcal{F}}_2(x_2)} \underbrace{\left[\mathbf{D_2}\mathbf{V_2}^\top\widetilde{\mathcal{Q}}_2^\top\right]}_{\hat{\mathcal{F}}_3(x_3)\ldots\hat{\mathcal{F}}_d(x_d)}, \tag{13}$$

where the first equation is the truncated SVD and the second is the new approximation with $\|g_2 - g_1\| \leq \delta$. After the truncation associated with the first and second cores, we obtain a total error of $\|g_2 - f\| = \|g_2 - g_1 + g_1 - f\| \leq \delta + \delta = 2\delta$. We repeat this procedure $(d-1)$ times to obtain a final approximation $\hat{f} = g_{d-1}$ such that $\|\hat{f} - f\| \leq (d-1)\delta$ and a relative error ϵ by setting $\delta = \frac{\epsilon}{d-1}\|f\|$.

The main computational burden of the algorithm described above is calculating the QR decompositions of the matrix-valued functions $\mathcal{U}_k(x_{\leq k})$ and $\mathcal{V}_k(x_{>k})$, since these functions have potentially high-dimensional inputs. A computationally feasible rounding procedure requires that these QR decompositions be tractable. In the discrete setting, one can obtain an algorithm that only requires the QR decompositions of reshapings of single cores. In the continuous setting, we can similarly construct an algorithm that requires only the QR decomposition of each univariate core. This algorithm starts by assuming that all the cores $\mathcal{F}_2(x_2), \ldots, \mathcal{F}_d(x_d)$ have *orthonormal rows*. Then, we can show that $\mathcal{V}_1$ also has orthonormal rows and that we are not required to take its QR decomposition in (12). Thus, in the first step of the rounding procedure we only need to compute the QR decomposition of $\mathcal{F}_1$ (12). The notion of *orthonormal rows*, that $\langle \mathcal{F}_k(x_k)[i,:], \mathcal{F}_k(x_k)[j,:]\rangle = \delta_{i,j}$ for $i, j = 1\ldots r_{k-1}$, is analogous to left orthogonality [30]; while *orthonormal columns*, that $\langle \mathcal{F}_k(x_k)[:,i], \mathcal{F}_k(x_k)[:,j]\rangle = \delta_{i,j}$ for $i, j = 1\ldots r_{k\cdot}$, are analogous to right orthogonality. Now that the continuous notions of orthonormality is clear, Algorithm 2 from [30, Algorithm 2] can be applied by replacing the discrete QR decompositions with continuous QR decompositions of matrix-valued functions [14].

Overall, constructing a low-rank approximation of a black-box function via cross approximation and rank adaptation can be implemented by successively increasing or "kicking" the TT ranks until rounding leads to a reduction of all ranks. This approach ensures that the ranks are *overestimated* for the cross approximation procedure. Pseudocode for rank-adaptation is given in Algorithm 3.

5. Numerical experiments

This section describe several numerical experiments that explore the capabilities and performance of our computational approach. In Section 5.1 we demonstrate how our adaptive approach

Algorithm 3 `ft-rankadapt`: FT approximation with rank adaptation

Input: A multivariate function $f : \mathcal{X} \to \mathbb{R}$, with $\mathcal{X} \subset \mathbb{R}^d$; cross approximation tolerance δ_{cross}; size of rank increase `kickrank`; rounding accuracy ϵ_{round}; initial ranks $\boldsymbol{r} = (1, r_1, \ldots, r_{d-1}, 1)$

Output: Low-rank approximation $\hat{f}$ such that rank increase followed by rounding does not change ranks

$\hat{f} = \texttt{cross-approx}(f, \boldsymbol{r}, \delta_{\text{cross}})$
$\hat{f}_r = \texttt{ft-round}(\hat{f}, \epsilon_{\text{round}})$
$\hat{\boldsymbol{r}} = \text{ranks}(\hat{f}_r)$
while $\exists\, i$ s.t. $\hat{r}_i = r_i$ **do**
  for $k = 1$ **to** $d - 1$ **do**
    $\boldsymbol{r}_k = \hat{\boldsymbol{r}}_k + \texttt{kickrank}$
    $\hat{f} = \texttt{cross-approx}(f, \boldsymbol{r}, \delta_{\text{cross}})$
  $\hat{f}_r = \texttt{ft-round}(\hat{f}, \epsilon_{\text{round}})$
  $\hat{\boldsymbol{r}} = \text{ranks}(\hat{f}_r)$
$\hat{f} = \hat{f}_r$

can outperform, by several orders of magnitude, the QTT and the tensor-of-coefficients approaches for approximating functions with local features. In Section 5.2, we show two examples of how our fiber-adaptive continuous algorithms explore a function, i.e., *where* they place function evaluations in the domain. These examples offer an intuitive explanation of the observed performance gains. In Section 5.3 we demonstrate that the FT can be used for adaptive integration *and* differentiation, even with functions that contain a discontinuity. Furthermore, these operations use the functional representation to avoid constructing discretizations that are specific to either differentiation or integration.

In Section 5.5, using a prototypical application in uncertainty quantification, we study how the fiber adaptation tolerance interacts with rounding tolerance. All of the experiments described below are performed with the Compressed Continuous Computation (C^3) library available at `http://github.com/goroda/Compressed-Continuous-Computation`.

We briefly mention how to perform integration and differentation in low-rank FT format. Integration involves integrating all of the univariate functions in each core and then performing matrix-vector multiplication $d - 1$ times:

$$\begin{aligned}\int f(x)dx &= \int \mathcal{F}_1(x_1)\mathcal{F}_2(x_2)\cdots\mathcal{F}_d(x_d)dx_1\ldots dx_d \\ &= \left(\int \mathcal{F}_1(x_1)dx_1\right)\left(\int \mathcal{F}_2(x_2)dx_2\right)\cdots\left(\int \mathcal{F}_d(x_d)dx_d\right) = \mathbf{\Gamma_1\Gamma_2}\cdots\mathbf{\Gamma_d},\end{aligned}$$

where $\mathbf{\Gamma_k} = \int \mathcal{F}_k(x_k)dx_k$ contains entries $\mathbf{\Gamma_k}[i,j] = \int f_k^{(ij)}(x_k)dx_k$. Furthermore, since each univariate function is represented in a continuous format, this integral is uniquely defined and computationally inexpensive. Differentiation requires differentiating the scalar-valued functions that make up the corresponding core. For example, consider the partial derivative of a d–variate function:

$$\frac{\partial f}{\partial x_k} = \mathcal{F}_1\cdots\mathcal{F}_{k-1}\begin{bmatrix} \frac{df_k^{(11)}}{dx_k} & \cdots & \frac{df_k^{(1r_k)}}{dx_k} \\ \vdots & & \vdots \\ \frac{df_k^{(r_{k-1}1)}}{dx_k} & \cdots & \frac{df_k^{(r_{k-1}r_k)}}{dx_k}\end{bmatrix}\mathcal{F}_{k+1}\cdots\mathcal{F}_d.$$

Again, because each univariate function is represented in a continuous format (e.g., in its own orthogonal polynomial basis), this operation is uniquely defined and computationally inexpensive.

5.1. Approximation of local features in analytic functions

In this section we compare local fiber-based adaptivity to discrete tensor-based approaches. In particular, we compare our fiber-adaptive scheme with the spectral tensor-train (STT) algorithms found in [6]. The STT approach works by first discretizing a function on a tensor-product grid defined by Gaussian quadrature, performing a low-rank decomposition of this discretized function using the TT or QTT, and projecting this decomposition onto a polynomial basis in order to obtain *spectrally convergent* approximations for analytic functions. We will demonstrate that even for *analytic* functions, a fiber adaptive approach can be more efficient compared with the "discretized function" *and* "tensor-of-coefficient" approaches. Our demonstration repeats an experiment from [6][7] on an *analytic* function that can be well represented using a global polynomial basis. The goal is to integrate a Gaussian bump

$$f(x) = A \exp\left(-\frac{(x_1-c)^2 + (x_2-c)^2 + (x_3-c)^2}{2l^2}\right), \quad x \in [0,1]^3, \qquad \text{with } c = 0.2 \text{ and } l = 0.05, \tag{14}$$

with $A = 1$. As part of its construction, the STT uses a QTT to perform *rank-revealing* interpolation of the function on the tensor-product grid of Gaussian quadrature nodes using the `TT-DMRG-cross` algorithm with $\epsilon = 10^{-10}$. [6] notes that QTT is more efficient than TT because it is able to localize function evaluations around the bump.

For this comparison, our *continuous* algorithms employ a fiber-adaptive scheme based on piecewise polynomials with $\delta_{\text{cross}} = \epsilon_{\text{round}} = 10^{-10}$. In particular, for each fiber encountered through cross approximation in Algorithm 3, the following steps are taken:

1. Approximate the fiber with a degree–7 Legendre polynomial basis.
2. If the square of the coefficient of the highest-degree polynomial, normalized by the squared norm of the function, is greater than a tolerance ϵ_{approx}, then split the domain into three regions.
3. If a split occured, return to step 1 for each new subdomain; otherwise stop.

The resulting errors in the integral for a given number of function evaluations are shown in Figure 2. Since the STT is not an adaptive scheme, increasing numbers of function evaluations are chosen by increasing the number of quadrature nodes in each dimension. The polynomial orders for the STT are increased such that the number of quadrature nodes grows in powers of two, from 4 to 32, to be conducive to the QTT. In contrast, the continuous cross-approximation algorithms we describe here are adaptive, and therefore increasing numbers of evaluations are obtained by tightening the fiber approximation tolerance ϵ_{approx}; tolerance values corresponding to each data point are indicated on the plot. Both of these approximations are able to eventually achieve $\mathcal{O}(10^{-11})$ error, but the QTT-based scheme reaches this error with a larger number of evaluations. This suggests that the base-2 reshapings of the QTT do not exploit local structure as effectively as our chosen piecewise approximation. Second, note that an adaptive tolerance allows for more granular control of the number of evaluations rather than a global polynomial order. Finally, we are able to achieve a maximum of *3 orders of magnitude* reduction in error for a specified number of function evaluations.

Next we explore *how* the automated adaptation algorithm chooses to approximate fibers. In particular, we perform cross-approximation with a fixed rank-2 approximation. Since (14) is actually rank-1, an efficient rank-2 approximation should introduce *zero* functions that require little to no storage, as described in Section 3. In Figure 3, we see that this is indeed the case. To demonstrate this fact consider that (14) can be written as $f(x) = e_1(x_1)e_2(x_2)e_3(x_2)$ with e_s proportional to a

[7] We use the TensorToolbox package (`https://pypi.python.org/pypi/TensorToolbox/`) for the STT results.

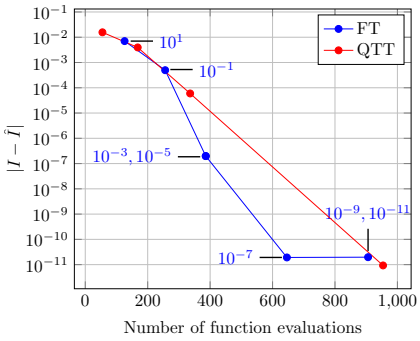
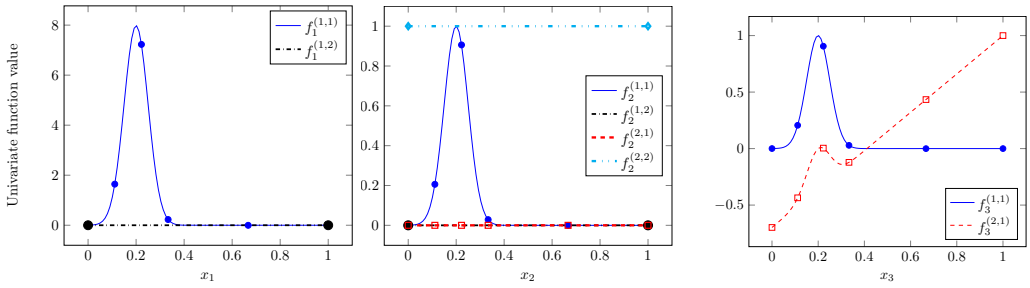


Figure 2: Convergence comparison between the spectral QTT [6] and the fiber-adaptive FT for a three-dimensional Gaussian bump. In the FT, each fiber is adapted using a breadth-first splitting of piecewise polynomials into three equal regions until the highest-degree coefficient in each region is below the tolerance indicated next to the data points. Rank adaptation uses `ft-rankadapt`.



(a) First core (b) Second core (c) Third core

Figure 3: Univariate functions for a fixed TT-rank=2 approximation. Marks indicate the location of knots found through adaptive piecewise polynomial procedures. Note that in the first and second core, there exist univariate functions with differing *numbers* of knots. In other words, the basis for each function was chosen *online*, *adaptively*, and *independently* for each fiber.

univariate Gaussian bump centered at $x_j = 0.2$. A TT-rank 2 function could then take the form[8]

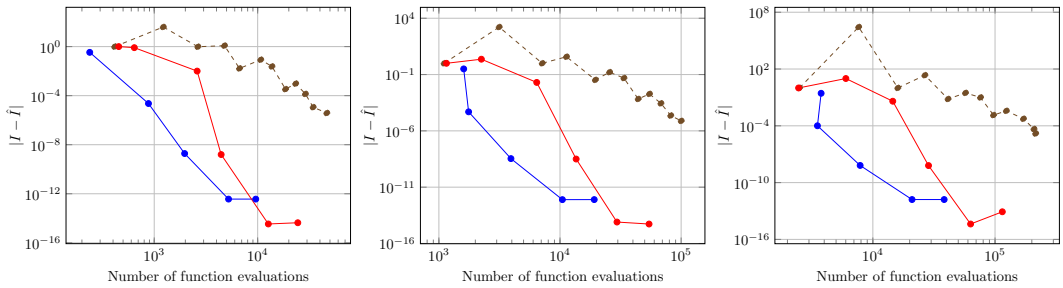
$$f(x_1, x_2, x_3) = [e_1(x_1)\ 0] \begin{bmatrix} e_2(x_2) & 0 \\ f_2^{(21)}(x_2) & f_2^{(22)}(x_2) \end{bmatrix} \begin{bmatrix} e_3(x_3) \\ f_3^{(21)}(x_3) \end{bmatrix}, \tag{15}$$

where $f_2^{(12)}$, $f_2^{(22)}$ and $f_3^{(21)}$ can be arbitrary functions since they are multiplied by the zero function in the first core.

Figure 3 indicates that this is exactly the structure found using continuous cross approximation. The first core essentially specifies $f_1^{(11)}(x_1) = e_1(x_1)$ and $f_2^{(12)} = 0$. Also notice that $f_1^{(11)}$ was refined twice, while $f_2^{(12)}$ was never refined. This adaptation happened in an *online* fashion; it would have been difficult *a priori* to create a basis for the first dimension that included this precise refinement. The second core also exhibits the structure we expect: $f_2^{(11)}(x_2) = e_2(x_2)$, $f_2^{(12)}(x_2) = 0$, $f_2^{(21)}(x_2) = 0$, and $f_2^{(22)}(x_2) = 1$. Note that the latter two functions could have been chosen *arbitrarily*. In this case, they were chosen to be constants. The third core chooses $f_3^{(11)} = e_3(x_3)$ and an arbitrary function for $f_3^{(21)}$.

As described above, in this example we chose a particular uniform refinement scheme for piecewise

[8] One can specify a different combination of zero, nonzero, and arbitrary functions by rearranging the order of the first core, for example. We specify this combination for illustrative purposes because it is the one chosen by the cross-approximation algorithm.



(a) $d = 8$ (b) $d = 16$ (c) $d = 32$

Figure 4: Comparison of convergence for FT (blue), QTT (red), and TT (brown) for the probability density functions of d-dimensional normal random variables. FT uses fiber-adaptive cross approximation, while the QTT and TT represent the coefficients of global polynomials and are computed using DMRG-cross.

polynomials. Other choices of number of regions or polynomial degree will naturally lead to different performance characteristics. Our goal was simply to show that the flexibility of *allowing* fiber adaptation *can* greatly improve performance. More complex non-uniform and function-adaptive refinement schemes [49] for piecewise polynomials can certainly be used. Our approach allows for *any* univariate function approximation scheme. In practice, this scheme can and should be tailored to any problem-specific knowledge.

Finally, we compare the fiber-adaptive FT with the QTT- and TT-based spectral decompositions for approximating the probability density function (PDF) of normal random variables of increasing dimension. Our tests are performed on d-dimensional versions of (14). We refine the spectral approximations by increasing polynomial degree, and, correspondingly, we tighten the univariate approximation tolerance for the FT. The results for the error in the integral are shown in Figure 4. Again, we observe excellent performance of the fiber-adaptive FT scheme.

5.2. Fiber adaptivity examples

We now explicitly demonstrate that the locations of function evaluations resulting from fiber adaptation do not lie on a tensor product grid; instead they are adapted to the local features.

These examples illustrate one of the primary ways that our approach differs from the discrete tensor-train and from other functional tensor-train approaches. Consider the approximation of two canonical rank-one functions:

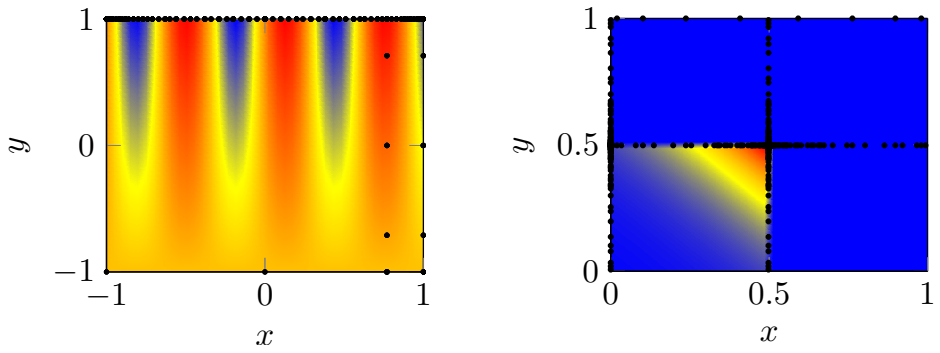
$$f_{\sin}(x_1, x_2) = \sin\left(10x_1 + 1/4\right)(x_2 + 1), \qquad (x_1, x_2) \in [-1, 1]^2 \tag{16}$$

$$f_{\text{genz2d}}(x_1, x_2) = \begin{cases} 0 & \text{if } x_1 > 0.5 \text{ or } x_2 > 0.5 \\ \exp(5x_1 + 5x_2) & \text{otherwise} \end{cases}, \qquad (x_1, x_2) \in [0, 1]^2 \tag{17}$$

where (17) is a bivariate Genz function of the 'discontinuous' family [50].

For (16) we use a global expansion of Legendre polynomials to represent each fiber. The coefficients are determined via projection using Clenshaw-Curtis quadrature to obtain nested rules; other quadrature rules, for instance Gaussian quadrature, can also be readily used. The degree $n - 1$ of each fiber approximation is progressively increased until four successive coefficients have magnitudes less than $\epsilon_{\text{approx}} = 10^{-10}$.

Figure 5(a) shows the function and the parameter values where it is evaluated by the cross approximation algorithm. Even though (16) has rank one, we deliberately construct a rank-two cross approximation to illuminate the pattern of function evaluations chosen by our method. We observe that fibers at different positions are approximated using differing numbers of function evaluations. The number of evaluations required in the oscillatory region (near $x_2 = 1$) is much greater than in the constant portion ($x_2 = -1$). Such an adaptation of the grid would be difficult to achieve with discrete TT, and highlights the flexibility of the continuous approximation framework.



(a) Sine function (16) and evaluation points. (b) Genz function (17) and evaluation points.

Figure 5: Contour plots and evaluations of f_{sin} and f_{genz2d}.

For (17) we can no longer use a global polynomial expansion due to the discontinuity, and therefore we employ piecewise polynomial approximations. We perform adaptation in a similar manner to Section 5.1. We use a breadth-first refinement where we split polynomials into three regions, approximate each region with a degree–6 polynomial, and further split the region if the leading coefficient of the expansion contributes more than $\epsilon_{\text{approx}} = 10^{-10}$ to the squared norm of the univariate function. We then approximate each smooth interval of the fiber using suitably scaled Legendre polynomials. In this example, we seek a rank-one approximation of the function. Function contours and evaluation locations are shown in Figure 5(b). This approximation, like that of (16), achieves machine accuracy. We see that the algorithm clusters evaluations points around the discontinuity, as desired, and that the evaluations again do not lie on a tensor-product grid.

Having constructed low-rank representation of these functions, we can perform computations directly in compressed format. For example, we can now integrate the discontinuous function f_{genz2d}. Such an integration could not be performed using array-based tensor-train algorithms, unless the entire domain were manually partitioned; otherwise, one would need specialized integration rules to deal with the discontinuity. By representing everything in functional form, we are able to perform integration and approximation automatically. In Section 5.3.2, we will evaluate integration performance for discontinuous Genz functions on higher-dimensional input spaces.

5.3. Adaptive integration and differentiation

In this section, we fix a target fiber approximation tolerance and fix the ranks to determine the resulting errors of approximate integrals and derivatives. We show that the error is well behaved and sufficient for most applications. Our second goal is to show that our representation of the FT is flexible enough to enable a variety of different computations. Specifically, we demonstrate that we simultaneously obtain adaptive integration and differentiation procedures by using an adaptive approximation approach. No specialized integration or finite difference algorithms or grids are required.

5.3.1. Integration of rank–two sine function

As in [37] we consider the rank-two function

$$f(x) = \sin(x_1 + x_2 + \ldots + x_d), \tag{18}$$

whose integral on the unit hypercube is known analytically for any d: $\int_{[0,1]^d} f(x)dx = \text{Im}\left[\left(\frac{e^i-1}{i}\right)^d\right]$.

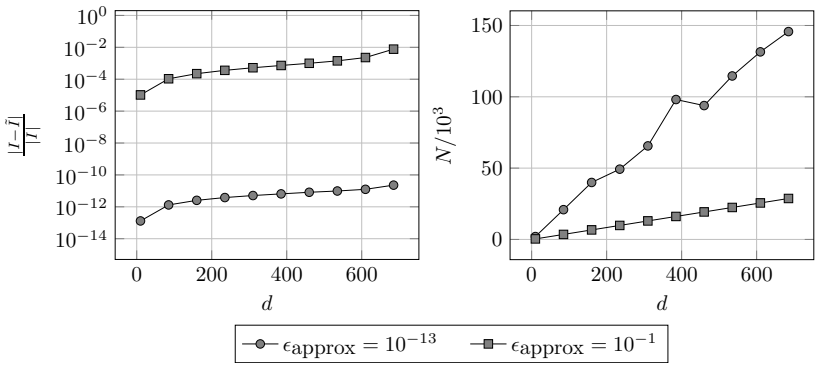


Figure 6: Relative errors (left panel) and growth in the number of evaluations (right panel) involved in the integration of (18), as a function of dimension d and fiber adaptation parameter ϵ_{approx}.

We study the performance of FT-based integration as a function of dimension and the fiber adaptation tolerance. Specifically, we approximate each univariate fiber with a Legendre polynomial expansion using pseudospectral projection with Gaussian quadrature. Each fiber is initially represented by an expansion of degree $k = 5$, and the degree is increased from k to $k + 7$ with each adaptation step. We stop adaptation after the last *two* coefficients contribute less than ϵ_{approx} to the squared norm of the function. Figure 6 shows the relative error in the integral as a function of ϵ_{approx} and the dimension d of the problem. The right panel also shows the number of function evaluations used to construct the FT approximation. As desired, the number of evaluations grows *linearly* with the input dimension, and the integration error is well behaved across dimensions, even for $d > 600$. (Note that the error axis is on a log scale and hence the variation of the errors for the tightest tolerance would be virtually invisible on the $\epsilon_{\text{approx}} = 10^{-1}$ curve.)

5.3.2. Integration and differentiation of rank–one discontinuous Genz function

We now demonstrate integration and differentiation of the discontinuous Genz function for varying dimensions. Specifically, we consider $f : [0, 1]^d \to \mathbb{R}$ defined as

$$f(x_1, x_2, \ldots, x_d) = \begin{cases} 0 & \text{if } x_i > \frac{1}{2} \text{ for any } i = 1 \ldots d \\ \exp\left(\sum_{i=1}^{d} 5x_i\right) & \text{otherwise} \end{cases} . \tag{19}$$

The exact integral of (19) is $I[f] = \left(\frac{\exp\left(\frac{5}{2}\right)-1}{5}\right)^d$; this problem is quite challenging because the integral grows exponentially with dimension. For example, for $d = 10$ the value of the integral is $I \approx 3.131 \times 10^3$, while for $d = 100$ the value is $I \approx 9.05455 \times 10^{34}$. In addition to computing the integral, we wish to evaluate numerically the following derivative: $D[f](x) = \nabla \cdot f(x_1, \ldots, x_d) = 5d \exp(\sum_{i=1}^{d} 5x_i)$ for $x_i = 0.2$ for all i.

Fiber adaptation is performed using the same scheme as for the two-dimensional case of Section 5.2. For this function, individual fiber adaptation is essential for approximating in the presence of discontinuities. Note that other discontinuity detection schemes [51, 52, 53] could also be readily used within our framework by embedding them within univariate function approximation routines.

The resulting relative errors for I and D, and the required number of function evaluations, are shown in Figure 7 as functions of the dimension d.

The results indicate stable approximations for all values of d for fixed tolerance levels, and that the number of function evaluations scales linearly with dimension. Furthermore, we are able to approximate extremely large values of the integral, suggesting a general robustness of the algorithm.

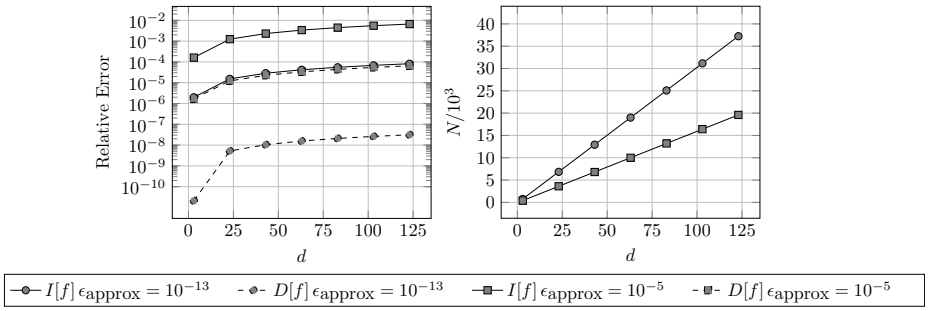


Figure 7: Relative errors (left panel) and # evaluations (right panel) when integrating or differentiating (19), versus dimension d, for different values of the fiber approximation tolerance ϵ_{approx}.

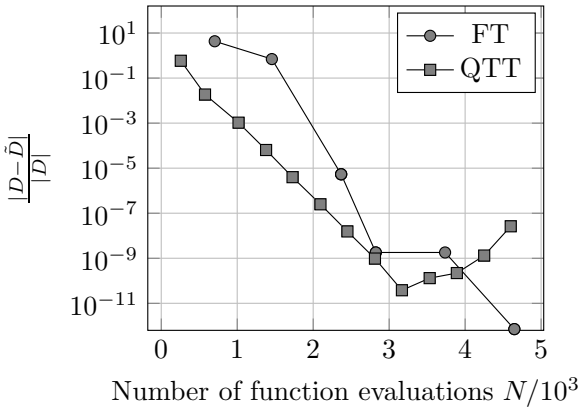
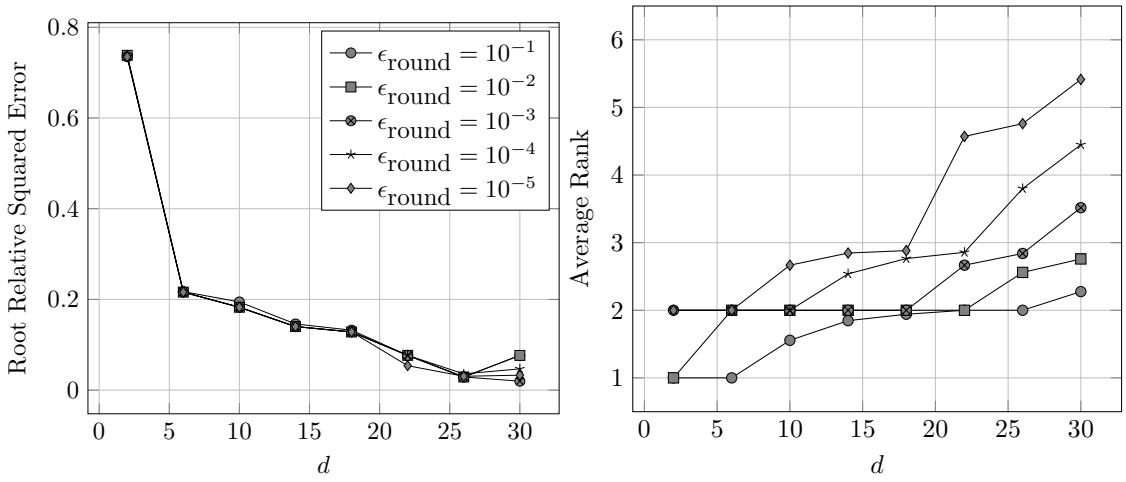


Figure 8: Relative error of differentiation with the continuous tensor-train using adaptive piecewise polynomials and the QTT using finite differences for a ten dimensional version of (19). FT refinement is obtained by diminishing ϵ_{approx}. QTT refinement is performed by quadrupling the size of the grid along each dimension. Refining functions proves more numericaly stable than refining grids.

Simultaneous integration and differentiation as reported here would be extremely difficult to perform using either the discrete tensor-train or the spectral tensor-train [6] techniques, because discontinuities pose problems for most integration rules. For example, we can compare our adaptive integration scheme to a scheme based on finite differences using a QTT decomposition of a function discretized on a regular grid. Practically, we can consider discretizations as fine as 2^{50} in each input coordinate, because the complexity of the QTT grows like the log of the discretization size. However, such discretizations can be difficult to employ in an adapt-to-tolerance setting, due to the difficulty of controlling roundoff errors.

Figure 8 shows, for $d = 10$, the error of the derivative with increasing numbers of function evaluations. The number of evaluations required by the QTT is governed by the size of discretization, while the number of evaluations for the FT is governed by the fiber adaptation tolerance. Errors with the QTT approach decrease rapidly, and then increase just as sharply. Our FT approach is more robust in the sense that it avoids such numerical instabilities (which are due to roundoff error). In effect, increasing the number of function evaluations by tightening a tolerance is safer than simply refining a tensor-product discretization. Practically, it is difficult to decide how finely to discretize a function and then to assess the accuracy of a fixed discretization scheme; our adaptive approach instead learns more about the structure of the problem in order to avoid such choices.



(a) Approximation Errors (b) Average ranks

Figure 9: Approximation errors and average rank for various ϵ_{round} on the function with a singularity (20).

5.4. Approximation of a function with a singularity

In this section we explore the properties of our algorithm on a function that does not readily admit low-rank structure and that has a singularity (which we soften by adding a small value to the denominator), namely

$$f(x_1, \ldots, x_d) = \frac{1}{\sqrt{\sum_{i=1}^{d} x_i^2 + 10^{-12}}}, \tag{20}$$

for $x \in [-1, 1]^d$. The quadratic in the denominator of this function is a sum of univariate functions and is therefore rank 2 regardless of dimension. However, taking its square root and then inverse causes the rank to grow with dimension. We seek to study how this rank grows with dimension and how the rank adaptation scheme performs with varying tolerances.

For this problem we again use piecewise-polynomials of maximum degree three, splitting the domain of a univariate fiber into four regions for adaptation. We initialize the rank to four, use a maximum of five cross-approximation iterations, and set `kickrank` $= 2$. We report the root relative mean squared error of the approximation on $10,000$ uniformly sampled inputs, and we provide the average FT rank found by the adaptive rouding scheme. These results are shown in Figure 9. The left panel shows the approximation error as a function of dimension for various rounding tolerances. First, the approximation error decreases with dimension but is still generally larger than the errors reported elsewhere in this paper. This fact reflects the challenging nature of this target function. The reduction in error with dimension can be attributed to the fact that the relative volume of the space that is affected by the discontinuity actually *decreases* with dimension. Furthermore, we see that over the given range of rounding tolerances, there is no significant difference in the approximation accuracy. This can be attributed to the fact that univariate approximation errors are the dominant source of inaccuracy, due to the difficulty of approximating the function near the origin. In Section 5.5 we will provide an example where the opposite occurs.

The right panel of Figure 9 shows the average ranks determined by the rank adaptation scheme. As expected the rank grows with dimension. Since the errors with dimension beyond five are relatively steady, this plot indicates the correct behavior that rank needs to grow to maintain a given level accuracy. Furthermore, the tighter the rounding tolerance, the larger the average rank. To demonstrate the process of adaptivity, in Table 2 we show the ranks following the rounding portion of each stage of Algorithm 3.

Iteration	Ranks
1	1 2 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 3 3 3 1
2	1 2 4 5 6 6 5 6 6 5 5 5 5 5 5 5 6 6 5 5 5 5 5 5 5 5 5 4 4 3 1
3	1 2 4 5 6 6 5 5 6 5 5 5 6 5 5 5 5 6 5 6 5 5 5 5 5 5 5 5 4 1
4	1 2 4 6 6 6 6 6 7 6 6 5 6 5 5 5 5 5 5 5 6 6 6 6 6 6 6 5 5 5 1
5	1 2 4 5 6 7 7 7 7 7 6 6 5 5 5 5 5 6 5 5 6 6 6 5 5 5 5 5 5 5 1
6	1 2 4 5 6 7 7 7 7 7 7 7 6 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 1

Table 2: Function ranks after rounding for given iteration of Algorithm 3 for Equation (20) with $\epsilon_{round} = 10^{-5}$ and $d = 30$.

5.5. Approximation of an elliptic PDE

We next explore the effects of various parameters of the FT approximation algorithm on a model of subsurface flow frequently encountered in UQ applications. Consider the following one-dimensional elliptic PDE: $\frac{\partial}{\partial s}\left(k(s,\omega)\frac{\partial u}{\partial s}\right) = s^2$, for $s \in [0,1]$ with $u(s)|_{s=0} = 0$ and $\frac{\partial u}{\partial s}|_{s=1} = 0$. We consider the effects of an uncertain permeability field k on a functional of the output pressure u. The permeability is modeled as a random process $k(s,\omega)$, endowed with a log-normal distribution: $\log\left[k(s,\omega) - a\right] \sim \mathcal{N}(0, c(s,s'))$. Here the covariance kernel is chosen to be $c(s,s') = \sigma^2 \exp\left(-\frac{|s-s'|}{l}\right)$. To obtain a finite dimensional representation of $k(s,\omega)$, we use the Karhunen-Loève expansion of $\log\left(k(s,\omega) - a\right)$ to express the random field using the eigenfunctions of an integral operator with kernel c. In particular, given eigenfunctions and eigenvalues obtained from $\int_{[0,1]} c(s,s')\phi_i(s')ds' = \lambda_i\phi_i(s)$, the log-normal process is represented as $k(s,\omega) = a + \exp\left(\sum_{i=1}^{\infty}\sqrt{\lambda_i}\phi_i(s)\xi_i(\omega)\right)$, where $\xi_i(\omega)$ are independent standard Gaussian random variables. We truncate this expansion after 24 modes.

The approximation objective of this problem is the solution of the PDE at a spatial location s for a realization of the permeability field parameterized by $(\xi_i)_{i=1}^{24}$. For simplicity, we will fix $s = 0.7$ to obtain a quantity of interest Q that is only a function of the ξ_i, i.e., $Q := u(0.7, \xi_1, \ldots, \xi_{24})$. We measure the relative L^2 error of our approximation with $n = 10^4$ Monte Carlo samples as

$$\text{error} = \sqrt{\frac{\sum_{i=1}^{n}(Q(\xi) - \widehat{f}(\xi))^2}{\sum_{i=1}^{n} Q(\xi)^2}}.$$

We will construct approximations of Q using parameter settings for (a, σ^2, l) that correspond to three different levels of difficulty for this problem. In particular, we will tackle an "easy" problem (P1) where $(a, \sigma^2, l) = (0.0, 0.1, 0.125)$, a "moderately" difficult problem (P2) where $(a, \sigma^2, l) = (0.5, 1.0, 0.045)$, and a "harder" problem (P3) where $(a, \sigma^2, l) = (0.0, 1.0, 0.045)$.

Before building low-rank representations of Q, we reparameterize the problem so that it maps from $[0,1]^{24}$ to the PDE solution value of interest. We do so by using the cumulative distribution function Φ of a standard Gaussian to define new input variables $\hat{\xi}_i := \Phi(\xi_i)$. Now $(\hat{\xi}_i)_{i=1}^{24}$ are uniformly distributed on the unit hypercube, and the function Q to be approximated is $(\hat{\xi}_i)_{i=1}^{24} \mapsto u\left(0.7, \Phi^{-1}(\hat{\xi}_1), \ldots, \Phi^{-1}(\hat{\xi}_{24})\right)$.

Our numerical experiments investigate how the final approximation error, number of function evaluations, and maximum rank change with the fiber approximation tolerance ϵ_{approx} and the rounding tolerance ϵ_{round}. We fix the cross approximation parameters to $\delta_{\text{cross}} = 10^{-3}$ and a maximum of 3 sweeps. The rank adaptation parameters are fixed to `kickrank` $= 5$ and a maximum of 4 rank adaptations. We use Legendre polynomials to approximate the fibers and initialize these approximations at degree two. Results for the three problem setups shown in Figure 10.

Several interesting patterns are apparent in the results. First, consider the ranks for each problem, shown in the rightmost column of Figure 10. We immediately see that the fiber approximation

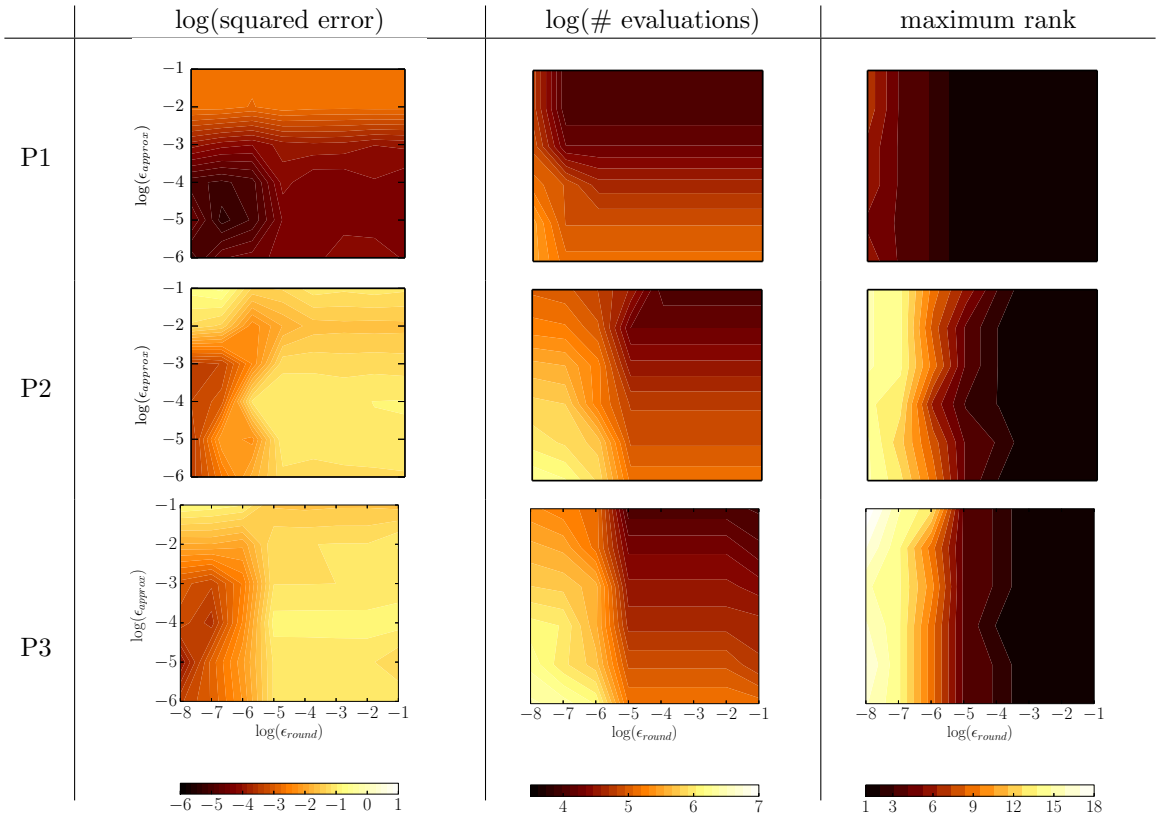


Figure 10: Squared error (left column), number of function evaluations (middle column), and maximum rank (right column) for three different configurations of the elliptic PDE problem, corresponding to different combinations of (a, σ^2, l). In particular, the top row corresponds to $(0.0, 0.1, 0.125)$, middle row corresponds to $(0.5, 1.0, 0.045)$, and the bottom row corresponds to $(0.0, 1.0, 0.045)$. Contours in the left two columns represent log-quantities.

accuracy has essentially no impact on the maximum rank found by rank adaptation. The maximum rank found by the adaptation procedure only changes as the rounding tolerance decreases. Furthermore, we see that the maximum rank increases as we move down the table. This trend corresponds to the increasing difficulty of each problem, and reflects slower decay of the singular values in decompositions of u. Higher ranks for P3 confirm that approximation is more difficult with $a = 0$ than with $a = 0.5$. We also note that unlike array-based tensor-train algorithms or the spectral tensor-train algorithm, the maximum rank attainable by the numerical procedure is *not bounded* by discretization level. Rounding is critical to restricting the growth of the rank.

The function evaluation count, shown in the middle column of Figure 10, also displays some interesting properties. In particular, for each of the three models, the contours exhibit two separate regimes. Before the rounding tolerance becomes tight enough to cause significant increases in the rank, the number of function evaluations is essentially unaffected by ϵ_{round}. The number of evaluations is only affected by the fiber approximation accuracy ϵ_{approx}. This behavior makes sense because the number of function evaluations is roughly proportional to $\mathcal{O}(ndr^2)$, where n can be thought of as an average number of function evaluations for each fiber—and in this first regime, the rank is constant. Once the rank starts increasing, the number of function evaluations grows with both tighter fiber approximation tolerance and tighter rounding tolerance. The number of evaluations increases more rapidly with tighter rounding tolerance, because reducing the latter produces a rapid increase in rank.

The left column of Figure 10 shows that the error exhibits a pattern similar to the number of function evaluations. In particular, the error is fairly constant until the rank starts increasing. Once the rank starts increasing, however, then both the fiber approximation tolerance and the rounding tolerance affect the error. Furthermore, if we fix the rounding tolerance and decrease the approximation tolerance, we see a rapid change in approximation error followed by a relatively large plateau area—suggesting that below a given ϵ_{approx}, either the accuracy of fiber approximations can no longer appreciably increase or further gains in accuracy have little effect on the overall approximation error. Overall, these results suggest that it is possible to find a reasonable value for the rank before increasing the fiber approximation accuracy. Future work will require investigating how to jointly adapt these two parameters in the best way.

Finally, comparing the error plots for P2 and P3, we observe that the contours look quite similar, but that the ranks and numbers of function evaluations are larger in the bottom row (P3) than in the middle row (P2). This characteristic is highly desirable for an "adapt-to-tolerance" scheme, as a given setting for the tolerances yields similar errors but appropriately larger computational effort.

6. Conclusions

We have developed new algorithms and data structures for representing and computing with the functional tensor-train decomposition. The algorithms are both locally and globally adaptive, and enable robust approximation, integration, and differentiation schemes. The primary way we achieve adaptation is by following the Chebfun paradigm of continuous computation: we consider continuous extensions of the CUR and QR matrix decompositions that naturally embed adaptivity within the resulting algorithms.

We have demonstrated that such schemes can enable several orders of magnitude more accurate approximation than the state-of-the-art STT or QTT approaches for the same computational expense. These savings arise because using continuous linear algebra provides a flexible way of incorporating and exploiting more than just low-rank structure. In particular, our algorithms can exploit the regularity of the target function and adapt to its features. Pivots for skeleton approximation are found via continuous optimization. Ranks can be adapted via a rounding procedure. And the FT decomposition offers an enormously flexible approach to univariate fiber approximation: different bases or approximation schemes may be chosen for different input coordinates, or even for parallel fibers in the same dimension; and these approximations can be adapted or refined on a fiber-by-fiber

basis. Function evaluations can thus be tailored to local features, without following any tensor product structure. Indeed, the FT scheme employs no *a priori* discretization of the target function and does not require specifying a tensor product set of candidate evaluation locations. This characteristic is particularly important for problems that are sensitive to the choice of discretization. The data structure that we have developed stores each univariate function independently, which allows for within-dimension differences in parameterization and increased compression rates when compared with the discrete TT cores.

Future algorithmic extensions will incorporate the `dmrg-cross` technique of [38] into a continuous rank-revealing algorithm, which should be more efficient than the rank-adaptation algorithm described here. Future work will also seek to exploit low-rank structure in inference and control. For example, we would like to extend the optimal stochastic control framework described in [5] to problems that are control-affine and avoid discretization of the state space altogether.

Acknowledgments

This work was supported by the National Science Foundation through grant IIS-1452019, and by the US Department of Energy, Office of Advanced Scientific Computing Research under award number DE-SC0007099.

References

[1] W. Austin, G. Ballard, T. G. Kolda, Parallel tensor compression for large-scale scientific data, in: Parallel and Distributed Processing Symposium, 2016 IEEE International, IEEE, 2016, pp. 912–922.

[2] A. Novikov, D. Podoprikhin, A. Osokin, D. P. Vetrov, Tensorizing neural networks, in: C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, R. Garnett (Eds.), Advances in Neural Information Processing Systems 28, Curran Associates, Inc., 2015, pp. 442–450.

[3] B. N. Khoromskij, I. V. Oseledets, Quantics-TT collocation approximation of parameter-dependent and stochastic elliptic PDEs, Computational Methods in Applied Mathematics Comput. Methods Appl. Math. 10 (2010) 376–394.

[4] A. A. Gorodetsky, S. Karaman, Y. M. Marzouk, Efficient high-dimensional stochastic optimal motion control using Tensor-Train decomposition, in: Proceedings of Robotics: Science and Systems, Rome, Italy, 2015.

[5] A. Gorodetsky, S. Karaman, Y. Marzouk, High-dimensional stochastic optimal control using continuous tensor decompositions, The International Journal of Robotics Research 37 (2018) 340–377.

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

[7] A. A. Gorodetsky, S. Karaman, Y. M. Marzouk, Low-rank tensor integration for Gaussian filtering of continuous time nonlinear systems, in: 2017 IEEE 56th Annual Conference on Decision and Control (CDC), 2017, pp. 2789–2794. doi:10.1109/CDC.2017.8264064.

[8] M. S. Eldred, G. Geraci, A. A. Gorodetsky, J. D. Jakeman, Multilevel-multifidelity approaches for forward UQ in the DARPA SEQUOIA project, in: 2018 AIAA Non-Deterministic Approaches Conference, AIAA SciTech Forum, Kissimmee, Florida, 2018. URL: https://alexgorodetsky.com/wp-content/uploads/2018/02/main-1.pdf. doi:10.2514/6.2018-1179.

[9] B. N. Khoromskij, I. V. Oseledets, QTT approximation of elliptic solution operators in higher dimensions, Russian Journal of Numerical Analysis and Mathematical Modelling 26 (2011) 303–322.

[10] S. V. Dolgov, TT-GMRES: solution to a linear system in the structured tensor format, Russian Journal of Numerical Analysis and Mathematical Modelling 28 (2013) 149–172.

[11] N. Lee, A. Cichocki, Regularized computation of approximate pseudoinverse of large matrices using low-rank tensor train decompositions, SIAM Journal on Matrix Analysis and Applications 37 (2016) 598–623.

[12] L. Mathelin, Quantification of uncertainty from high-dimensional scattered data via polynomial approximation, International Journal for Uncertainty Quantification 4 (2014) 243–271.

[13] P. Rai, Sparse low-rank approximation of multivariate functions – applications in uncertainty quantification, Ph.D. thesis, Ecole Centrale Nantes, 2014.

[14] A. Gorodetsky, Continuous low-rank tensor decompositions, with applications to stochastic optimal control and data assimilation, Ph.D. thesis, Massachusetts Institute of Technology, 2017.

[15] P. R. Conrad, Y. M. Marzouk, N. S. Pillai, A. Smith, Accelerating asymptotically exact mcmc for computationally intensive models via local approximations, Journal of the American Statistical Association 111 (2016) 1591–1607.

[16] D. P. Bertsekas, J. N. Tsitsiklis, Neuro-dynamic programming: an overview, in: Decision and Control, 1995., Proceedings of the 34th IEEE Conference on, volume 1, IEEE, 1995, pp. 560–564.

[17] I. V. Oseledets, Constructive representation of functions in low-rank tensor formats, Constructive Approximation 37 (2013) 1–18.

[18] G. Beylkin, J. Garcke, M. J. Mohlenkamp, Multivariate regression and machine learning with sums of separable functions, SIAM Journal on Scientific Computing 31 (2009) 1840–1857.

[19] A. Doostan, A. Validi, G. Iaccarino, Non-intrusive low-rank separated approximation of high-dimensional stochastic models, Computer Methods in Applied Mechanics and Engineering 263 (2013) 42–55.

[20] A. Gorodetsky, J. Jakeman, Gradient-based optimization for regression in the functional tensor-train format, Journal of Computational Physics 374 (2018) 1219–1238.

[21] S. Olver, A. Townsend, A practical framework for infinite-dimensional linear algebra, in: Proceedings of the 1st First Workshop for High Performance Technical Computing in Dynamic Languages, IEEE Press, 2014, pp. 57–62.

[22] Z. Battles, L. N. Trefethen, An extension of MATLAB to continuous functions and operators, SIAM Journal on Scientific Computing 25 (2004) 1743–1770.

[23] R. B. Platte, L. N. Trefethen, Chebfun: a new kind of numerical computing, in: Progress in Industrial Mathematics at ECMI 2008, Springer, 2010, pp. 69–87.

[24] A. Townsend, L. N. Trefethen, An extension of Chebfun to two dimensions, SIAM Journal on Scientific Computing 35 (2013) C495–C518.

[25] A. Townsend, L. N. Trefethen, Continuous analogues of matrix factorizations, Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences 471 (2014) 20140585–20140585.

[26] T. A. Driscoll, N. Hale, L. N. Trefethen, Chebfun guide, 2014.

[27] B. Hashemi, L. N. Trefethen, Chebfun in three dimensions, SIAM Journal on Scientific Computing 39 (2017) C341–C363.

[28] M. J. Mohlenkamp, Function space requirements for the single-electron functions within the multiparticle schrödinger equation, Journal of Mathematical Physics 54 (2013) 062105.

[29] B. N. Khoromskij, O(dlog n)-Quantics approximation of nd tensors in high-dimensional numerical modeling, Constructive Approximation 34 (2011) 257–280.

[30] I. V. Oseledets, Tensor-train decomposition, SIAM Journal on Scientific Computing 33 (2011) 2295–2317.

[31] M. Chevreuil, R. Lebrun, A. Nouy, P. Rai, A least-squares method for sparse low rank approximation of multivariate functions, SIAM/ASA Journal on Uncertainty Quantification 3 (2015) 897–921.

[32] D. Bertsekas, Dynamic programming and optimal control, Athena Scientific Belmont, 2007.

[33] W. H. Fleming, H. M. Soner, Controlled Markov processes and viscosity solutions, volume 25, Springer, 2006.

[34] T. J. Hastie, R. J. Tibshirani, Generalized additive models, Chapman and Hall, 1990.

[35] P. Ravikumar, H. Liu, J. D. Lafferty, L. Wasserman, SpAM: Sparse Additive Models, in: J. C. Platt, D. Koller, Y. Singer, S. T. Roweis (Eds.), Advances in Neural Information Processing Systems 20, Curran Associates, Inc., 2008, pp. 1201–1208.

[36] L. Meier, S. Van de Geer, P. Bühlmann, High-dimensional additive modeling, The Annals of Statistics 37 (2009) 3779–3821.

[37] I. V. Oseledets, E. E. Tyrtyshnikov, TT-cross approximation for multidimensional arrays, Linear Algebra and its Applications 432 (2010) 70–88.

[38] D. Savostyanov, I. V. Oseledets, Fast adaptive interpolation of multi-dimensional arrays in tensor train format, in: Multidimensional (nD) Systems (nDs), 2011 7th International Workshop on, IEEE, 2011, pp. 1–8.

[39] S. A. Goreinov, E. E. Tyrtyshnikov, N. L. Zamarashkin, A theory of pseudoskeleton approximations, Linear Algebra and its Applications 261 (1997) 1–21.

[40] M. W. Mahoney, P. Drineas, CUR matrix decompositions for improved data analysis, Proceedings of the National Academy of Sciences 106 (2009) 697–702.

[41] C. Boutsidis, D. P. Woodruff, Optimal CUR matrix decompositions, in: Proceedings of the 46th Annual ACM Symposium on Theory of Computing, ACM, 2014, pp. 353–362.

[42] S. A. Goreinov, N. L. Zamarashkin, E. E. Tyrtyshnikov, Pseudo-skeleton approximations by matrices of maximal volume, Mathematical Notes 62 (1997) 515–519.

[43] S. A. Goreinov, E. E. Tyrtyshnikov, The maximal-volume concept in approximation by low-rank matrices, Contemporary Mathematics 280 (2001) 47–52.

[44] D. V. Savostyanov, Quasioptimality of maximum-volume cross interpolation of tensors, Linear Algebra and its Applications 458 (2014) 217–244.

[45] M. Bebendorf, Approximation of boundary element matrices, Numerische Mathematik 86 (2000) 565–589.

[46] S. A. Goreinov, I. V. Oseledets, D. V. Savostyanov, E. E. Tyrtyshnikov, N. L. Zamarashkin, How to find a good submatrix, Matrix methods: theory, algorithms and applications (2010) 247.

[47] D. Day, L. Romero, Roots of Polynomials Expressed in Terms of Orthogonal Polynomials, SIAM Journal on Numerical Analysis 43 (2005) 1969.

[48] M. Bebendorf, Hierarchical Matrices: A means to efficiently solve elliptic boundary value problems, Springer, 2008.

[49] A. Cohen, J. Mirebeau, Adaptive and anisotropic piecewise polynomial approximation, in: Multiscale, Nonlinear and Adaptive Approximation, Springer, 2009, pp. 75–135.

[50] A. Genz, Testing multidimensional integration routines, in: Proc. of international conference on Tools, methods and languages for scientific and engineering computation, Elsevier North-Holland, Inc., 1984, pp. 81–94.

[51] R. Archibald, A. Gelb, J. Yoon, Polynomial fitting for edge detection in irregularly sampled signals and images, SIAM Journal on Numerical Analysis 43 (2005) 259–279.

[52] R. Archibald, A. Gelb, R. Saxena, D. Xiu, Discontinuity detection in multivariate space for stochastic simulations, Journal of Computational Physics 228 (2009) 2676–2689.

[53] A. A. Gorodetsky, Y. M. Marzouk, Efficient localization of discontinuities in complex computational simulations, SIAM Journal on Scientific Computing 36 (2014) A2584–A2610.