A continuous analogue of the tensor-train decomposition

Yile Li

Introduction

- Functional tensor train: A multivariate function is represented as a particular sum of products of univariate functions

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

Low rank representation

- In this paper we provide a computational methodology for approximating multivariate functions and computing with them in this format.
- there are no existing methods for low-rank function approximation that adapt parameterizations (on a grid or otherwise) in addition to ranks.

Contribution

- 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.

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

- 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 tensortrain (TT) decomposition
- The FT represents functions using a tensor-train ansatz by replacing the three-dimensional TT cores with univariate matrix-valued functions.
- a framework to compute the FT that employs adaptive approximations of univariate fibers, and that is not tied to any tensorized discretization

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

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

Discrete tensor-trains and function approximation

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

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

Existing methods

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.

Low rank representation of a coefficient tensor

- representing multivariate functions via a tensor-train decomposition of coefficients

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 $k \in \{1,\dots,d\}$ 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:

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

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,

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

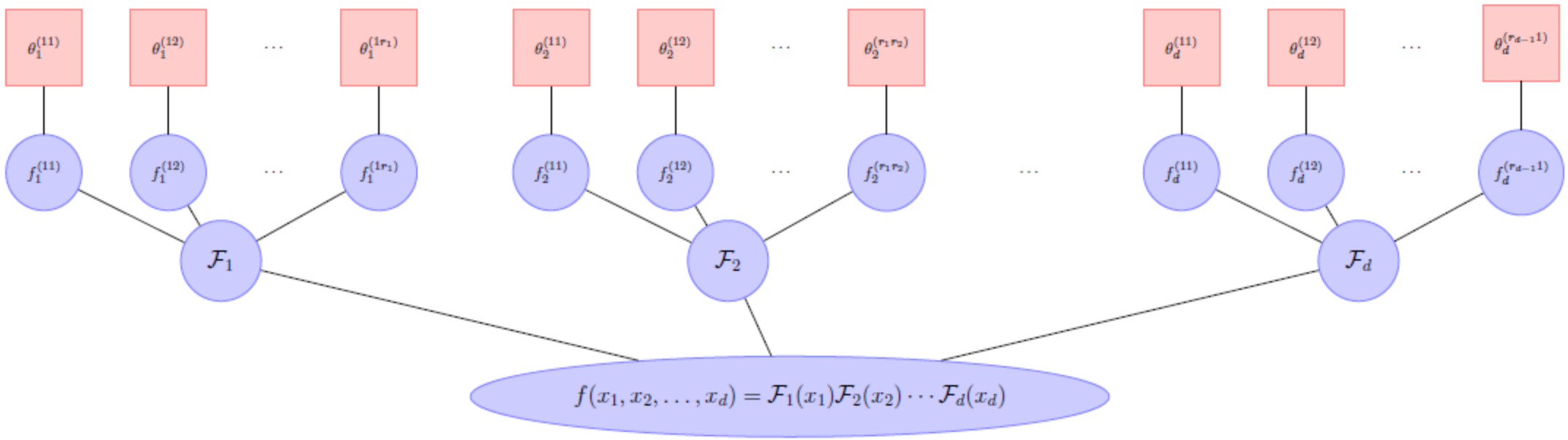
Continuous analogue of the tensor-train

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).

Diagram



- 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.

Conditions of equivalence

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

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

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 $\alpha_{k-1} = 1, \ldots, r_{k-1}$, $\alpha_k = 1, \ldots, r_k$, and $j = 1 \ldots, n_k$.

Conditions of equivalence

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:

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

Errors introduced through parameterization

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

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

Constructing low-rank approximations

- Cross approximation
- Approximate maxvol through dominant submatrices
- Rounding

- how to construct FT approximations with **continuous analogues of cross approximation and rounding algorithms**

- 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.

- rather than tensor fibers and univariate functions, we would have **tensor slices and bivariate functions**
- provide **an adapt-to-tolerance procedure for both ranks and fibers**.

Cross approximation

- 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.

$$f(x,y) = \mathcal{C}(x)\mathbf{F}^{\dagger}\mathcal{R}(y)$$

$$\mathcal{C}(x) = \left[f\left(x, y^{(1)}\right) \;\; f\left(x, y^{(2)}\right) \;\; \cdots \;\; f\left(x, y^{(r)}\right) \right] \qquad \mathcal{R}(y) = \begin{bmatrix} f\left(x^{(1)}, y\right) \\ f\left(x^{(2)}, y\right) \\ \vdots \\ f\left(x^{(r)}, y\right) \end{bmatrix}$$

$$\mathbf{F}[i,j] = f\left(x^{(i)}, y^{(j)}\right) \qquad i,j = 1,\ldots,r,$$

In two dimensions, cross approximation represents a low-rank function by its CUR decomposition

Cross approximation

- there **are fibers corresponding to each dimension of the tensor**
- fibers associated with each input coordinate are chosen sequentially to approximately **maximize the volume of the skeleton matrix F formed by their intersection**.
- choosing an actual **maximum volume submatrix**(discrete)
- generalizing the concept of tensor fibers to encompass the case of univariate functions obtained by fixing all but one input variable (continuous)

Cross approximation

- Using the notation of matrix-valued functions, pseudocode for our continuous cross approximation algorithm is provided in Algorithm 1. which uses the dominant algorithm to **select the next fiber locations**.

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 `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: $\mathcal{C}(x)[1,i] =$ `approx-fiber`$(f(x, y^{(i)}), \epsilon_{\text{approx}})$ for $i = 1 \ldots r$
6: **while** $\delta \leq \delta_{\text{cross}}$ **do**
7: $\quad \mathcal{Q}\mathbf{T} =$ `qr`$(\mathcal{C})$ # QR decomposition of a matrix-valued function
8: $\quad \boldsymbol{x} =$ `dominant`$(\mathcal{Q})$
9: $\quad$ Initialize rows $\mathcal{R} : [c,d] \to \mathbb{R}^{r \times 1}$
10: $\quad \mathcal{R}[j,1](y) =$ `approx-fiber`$(f(x^{(j)}, y), \epsilon_{\text{approx}})$ for $j = 1 \ldots r$
11: $\quad \mathcal{Q}\mathbf{T} =$ `qr`$(\mathcal{R}^T)$
12: $\quad \boldsymbol{y} =$ `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: $\quad \mathcal{C}(x)[1,i] =$ `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: $\quad k = k + 1$

in Lines 5, 10, and 14, where column and row quasimatrices are formed

we only generate an approximation of a tensor fiber when the cross approximation algorithm dictates that we need that particular fiber

we adapt the function at the (local) fiber level rather than the (global) dimension level

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

the dominant routine. This routine chooses the set of fibers to be used within a dimension during the next sweep across dimensions

Approximate maxvol through dominant submatrices

- Algorithm 1 uses the dominant algorithm to select the next fiber locations
- goal of dominant is to **find indices that correspond to a dominant submatrix**

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* $|\mathcal{B}[i, k](x)| \leq 1$.

Approximate maxvol through dominant submatrices

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

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

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 $\boldsymbol{\alpha} = \{(i_1, x_1), \dots, (i_r, x_r)\}$.

Rounding

- 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
- 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

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

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

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

Rounding

- we compress f1 via a truncated SVD. To this end, we first perform two QR decompositions of matrix-valued functions

Suppose that we start with the first unfolding of a function

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

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

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

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 $\hat{r}_1 \le r_1$ is chosen to obtain an approximation

$$g_1 = \underbrace{\mathcal{Q}_1\mathbf{U}_1}_{\hat{\mathcal{F}}_1(x_1)} \underbrace{\left[\mathbf{D}_1\mathbf{V}_1{}^\top\widetilde{\mathcal{Q}}_1^\top\right]}_{\hat{\mathcal{F}}_2(x_2)\hat{\mathcal{F}}_3(x_3)\ldots\hat{\mathcal{F}}_d(x_d)},$$

with error $\|f - g_1\| \le \delta$. $\hat{\mathcal{F}}_1(x_1)$ 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

Rounding

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

$$\mathcal{U}_2(x_{\leq 2}) = \mathcal{Q}_2(x_{\leq 2})\mathbf{R}_2, \quad \mathcal{V}_2^\top(x_{>2}) = \widetilde{\mathcal{Q}}_2(x_{>2})\widetilde{\mathbf{R}}_2$$

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 $\delta = \frac{\epsilon}{d-1}\|f\|$.

Rounding

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**
        $\boldsymbol{r}_k = \hat{\boldsymbol{r}}_k + \texttt{kickrank}$
        $\hat{f} = \texttt{cross-approx}(f, \boldsymbol{r}, \delta_{\text{cross}})$
    $\hat{f}_r = \texttt{ft-round}(\hat{f}, \epsilon_{\text{round}})$
    $\hat{\boldsymbol{r}} = \text{ranks}(\hat{f}_r)$
$\hat{f} = \hat{f}_r$

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.

Theory

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

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

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):

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

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

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

Theory

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

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

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 $d\epsilon < 1/e$. □

Theory

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

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

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.