Abstract

Recently, there has been a growing interest in efficient numerical algorithms based on tensor networks and low-rank techniques to approximate high-dimensional functions and solutions to high-dimensional PDEs. In this paper, we propose a new tensor rank reduction method based on coordinate transformations that can greatly increase the efficiency of high-dimensional tensor approximation algorithms. The idea is simple: given a multivariate function, determine a coordinate transformation so that the function in the new coordinate system has smaller tensor rank. We restrict our analysis to linear coordinate transformations, which gives rise to a new class of functions that we refer to as tensor ridge functions. Leveraging Riemannian gradient descent on matrix manifolds we develop an algorithm that determines a quasi-optimal linear coordinate transformation for tensor rank reduction. The results we present for rank reduction via linear coordinate transformations open the possibility for generalizations to larger classes of nonlinear transformations. Numerical applications are presented and discussed for linear and nonlinear PDEs.

1. Introduction

There has been a growing interest in efficient numerical algorithms based on tensor networks and low-rank techniques to approximate high-dimensional functions and solutions to high-dimensional PDEs [25, 8, 22, 23, 45, 47]. A tensor network is a factorization of an entangled object such as a multivariate function or an operator, into a set of simpler objects (e.g., low-dimensional functions or operators) which are amenable to efficient representation and computation. The process of building a tensor network relies on a hierarchical decomposition of the entangled object, which can be visualized in terms of trees [43, 14, 4]. Such a decomposition is rooted in the spectral theory for linear operators [21], and it opens the possibility to approximate high-dimensional functions and compute the solution of high-dimensional PDEs at a cost that scales linearly with respect to the dimension of the object and polynomially with respect to the tensor rank.

Given the fundamental importance of tensor rank in computations and its non-favorable scaling, in this paper we propose a new tensor rank reduction method based on coordinate flows that can greatly increase the efficiency of high-dimensional tensor approximation algorithms. To describe the method, consider the scalar field \( u(x) \), where \( x \in \Omega \subseteq \mathbb{R}^d, d > 1 \). The idea is very simple: determine an invertible coordinate transformation \( H : \mathbb{R}^d \rightarrow \mathbb{R}^d \) so that the function

\[
v(x) = u(H(x))
\]

has smaller tensor rank than \( u(x) \). Representing a function on a transformed coordinate system has proven to be a successful technique for a wide range of applications [29, 44, 19], including tensor rank reduction in quantum many-body problems [26]. To illustrate the effects of coordinate transformations on tensor rank, in...
Figure 1: (a) Two-dimensional Gaussian stretched along the $x_2$-axis; (b) Two-dimensional rotated Gaussian (clockwise rotation by $\epsilon = \pi/4$ radians); (c) Rank of the rotated Gaussian versus the clockwise rotation angle $\epsilon$.

Figure 1, we show that a simple two-dimensional rotation can increase the rank of fully separated (i.e., rank one) Gaussian function significantly. Vice versa, the inverse rotation can transform a rotated Gaussian with high tensor rank into a rank one function. Similar results hold of course in higher dimensions.

Under mild assumptions on the function $u(x)$, one may argue, e.g., using nonlinear dynamics or the theory of optimal mass transport, that there always exists a transformation $H$ such that $v(x) = u(H(x))$ possesses the smallest possible multilinear rank among all tensors in a given format. However, developing a computationally tractable algorithm for obtaining the transformation $H$ given the function $u(x)$ is not an easy task. The main objective of this paper is to develop a mathematical framework and computationally efficient algorithms for obtaining quasi-optimal tensor rank-reducing coordinate transformations $H$. In particular, we restrict our analysis to linear coordinate transformations. In this setting, $H$ can be represented by a matrix $A$, which allows us to write (1) in the simplified form

$$v(x) = u(Ax).$$

The function $v(x)$ is known as a generalized ridge function [34]. If $u(x)$ is represented in a tensor format, i.e., a series of products of one-dimensional functions $\psi(x_i)$ called tensor modes, then $v(x)$ inherits a similar series expansion. However, under the action of the linear map $Ax$ the tensor modes are no longer univariate. Instead, they take the form of ridge functions $\psi(a_i \cdot x)$, where $a_i$ is the $i$-th row of the matrix $A$. Since these ridge tensor modes are now $d$-dimensional, the tensor compression which we had for $u(x)$ may be lost.

Our contributions are as follows. First, we introduce a new class of functions that we refer to as ridge tensors, and provide a new method for evaluating $v(x)$ without losing the tensor compression of $u(x)$. Second, we develop a new Riemannian optimization algorithm for determining a linear map $A$ so that $v(x)$ has smaller tensor rank than $u(x)$. Our approach is to parameterize the transformation $A$ with a parameter $\epsilon \geq 0$, i.e., set $A = A(\epsilon)$, and consider $A(\epsilon)$ as the flow map of an appropriate linear dynamical system. From this dynamical system perspective we derive a partial differential equation (PDE) for the $\epsilon$-dependent function $v(x; \epsilon) = u(A(\epsilon)x)$. We then integrate the PDE for $v(x; \epsilon)$ using a step-truncation tensor integration scheme [36, 38, 23] to obtain $v(x; \epsilon)$ in a low-rank tensor format for all $\epsilon$. The quasi-optimal rank-reducing coordinate map $A$ is obtained by minimizing a rank-related cost function using an appropriate path of transformations $A(\epsilon)$. We construct such a path by giving the collection of linear transformations $A(\epsilon)$ a Riemannian manifold structure and performing Riemannian gradient descent optimization, i.e., assigning to the derivative of the path $dA(\epsilon)/d\epsilon$ a descent direction for the cost at each $\epsilon$ with respect to

$^{1}$If we allow for nonlinear coordinate flows [17], then we can of course map any multivariate probability density function (PDF) into a target distribution that has rank-one. Similar results can be obtained via optimal mass transport, e.g., by suitable approximations of the Knöthe-Rosenblatt rearrangement [39][42].

2
the chosen geometry. Our theoretical contributions provide a new framework that be generalized to larger classes of nonlinear transformations, and open new pathways for tensor rank reduction that are complementary to classical rank reduction methods, e.g., based on hierarchical SVD [15, 16]. Building upon our recent work on rank-adaptive tensor integrators [11], we also apply the new rank-reduction algorithm based on coordinate flows to linear PDEs. Specifically, we study advection problems in dimensions two, three and five and a reaction-diffusion equation in dimension two.

This paper is organized as follows. In section 2 we briefly review the functional tensor train (FTT) expansion, which will be the main tensor format for demonstrating the proposed rank-reduction method based on coordinate flows. In section 3 we introduce ridge tensor trains and provide a simple example to illustrate the effects of coordinate flows on tensor rank. In section 4 we formulate the tensor rank reduction problem as a Riemannian optimization problem on the manifold of volume-preserving invertible coordinate maps. We also provide an efficient algorithm for computing the numerical solution to such optimization problem. In section 5 we demonstrate the proposed Riemannian gradient descent algorithm for computing quasi-optimal linear coordinate maps on a three-dimensional prototype function. In section 6 we apply the proposed rank reduction methods to PDEs in dimensions two, three and five and a reaction-diffusion equation in dimension two. We also include two appendices in which we describe the Riemannian manifold of coordinate flows, and provide theoretical results supporting the proposed mathematical framework for ridge tensors.

2. Functional tensor train (FTT) expansion

The coordinate flow rank reduction technique we propose in this paper can be applied to any tensor format, in particular to the functional tensor train (FTT) format [33, 6, 13]. In this section we briefly review the construction of FTT expansions for multivariate function belonging to the weighted Hilbert space

$$H = L^2_\mu(\Omega).$$

Here, $\Omega \subseteq \mathbb{R}^d$ is a separable domain such as a $d$-dimensional flat torus $\mathbb{T}^d$ or a Cartesian product of $d$ subsets of the real line

$$\Omega = \prod_{i=1}^d \Omega_i, \quad \Omega_i \subseteq \mathbb{R},$$

and $\mu$ is a finite product measure on $\Omega$

$$\mu(x) = \prod_{i=1}^d \mu_i(x_i).$$

As is well-known (e.g., [33, 6, 13]), each element $u \in L^2_\mu(\Omega)$ admits a FTT expansion of the form

$$u(x) = \sum_{\alpha_1, \ldots, \alpha_d=1}^\infty \sqrt{\lambda(\alpha_d-1)} \psi_1(1; x_1; \alpha_1) \psi_2(\alpha_1; x_2; \alpha_2) \cdots \psi_d(\alpha_d-1; x_d; 1),$$

where $\{\psi_i(\alpha_i-1; x_i; \alpha_i)\}_{\alpha_i}$ are orthonormal eigenfunctions of a hierarchy of compact self-adjoint operators. Orthonormality of $\{\psi_i(\alpha_i-1; x_i; \alpha_i)\}_{\alpha_i}$ is relative to the inner products

$$\int_{\Omega_i} \psi_i(\alpha_i-1; x_i; \alpha_i) \psi_i(\alpha_i-1; x_i; \beta_i) d\mu_i(x_i) = \delta_{\alpha_i, \beta_i}, \quad i = 1, \ldots, d-1, \quad \alpha_0 = 1,$$

$$\int_{\Omega_d} \psi_d(\alpha_d-1; x_d; 1) \psi_d(\beta_d-1; x_d; 1) d\mu_d(x_d) = \delta_{\alpha_d-1, \beta_d-1}.$$
The sequence of positive real numbers \( \left\{ \sqrt{\lambda(\alpha_{d-1})} \right\}_{\alpha_{d-1}=1}^{\infty} \) appearing in (6) represents the product of all spectra of the compact self-adjoint operators mentioned above, and it has a single accumulation point at zero. By truncating such spectra so that only the largest eigenvalues and corresponding eigenfunctions are retained, we obtain the following FTT approximation of spectra of the compact self-adjoint operators mentioned above, and it has a single accumulation point at \( \Lambda \).

\[
\psi_i(\alpha_{i-1}; x; \alpha_i) = \psi_i(1, \alpha_{i-1}; x; \alpha_i),
\]

where \( r = (r_0, r_1, \ldots, r_{d-1}, r_d) \) is the tensor rank, \( r_0 = 1 \) and \( r_d = 1 \). At this point it is convenient to define the following matrix-valued functions \( \Psi_i(x_i) \) (known as tensor cores)

\[
\Psi_1(x_1) = \begin{bmatrix} \psi_1(1; x_1; 1) & \cdots & \psi_1(1; x_1; r_1) \end{bmatrix},
\]

\[
\Psi_i(x_i) = \begin{bmatrix} \psi_i(1; x_i; 1) & \cdots & \psi_i(1; x_i; r_i) \\ \vdots & \ddots & \vdots \\ \psi_i(r_{i-1}; x_i; 1) & \cdots & \psi_i(r_{i-1}; x_i; r_i) \end{bmatrix}, \quad i = 2, \ldots, d - 1,
\]

\[
\Psi_d(x_d) = \begin{bmatrix} \psi_d(1; x_d; 1) \\ \vdots \\ \psi_1(r_{d-1}; x_d; 1) \end{bmatrix},
\]

and write (8) in a compact matrix product form

\[
u_{\text{TT}}(x) = \Psi_1(x_1) \Psi_2(x_2) \cdots \sqrt{\Lambda} \Psi_d(x_d),
\]

where \( \Lambda \) is a diagonal matrix with entries \( \lambda(\alpha_{d-1}) \) \( (\alpha_{d-1} = 1, \ldots, r_{d-1}) \). To simplify notation further, we will often suppress explicit tensor core dependence on the spatial variable \( x_i \), allowing us to write \( \Psi_i = \Psi_i(x_i) \) and \( \psi_i(\alpha_{i-1}; \alpha_i) = \psi_i(\alpha_{i-1}; x_i; \alpha_i) \) as the spatial dependence is indicated by the tensor core subscript “\( i \)”.

### 3. Tensor ridge functions

Let us introduce a new class of functions, which we call tensor ridge functions, that will be used in subsequent sections to build a tensor rank-reduction theory via linear coordinate mappings. To this end, consider the following invertible linear coordinate transformation

\[
y = Ax, \quad A : \mathbb{R}^d \to \mathbb{R}^d.
\]

If we evaluate a function \( u \in L_p^2(\Omega) \) at \( y \), we obtain the generalized ridge function \( u(Ax) \) (e.g., [34]). Although the coordinate transformation \( A \) is linear, the evaluation of \( u \) on \( Ax \) defines a nonlinear map of functions

\[
u(x) \mapsto v(x) = u(Ax).
\]

The image of a FTT tensor (8) under such a map has the form

\[
u_{\text{TT}}(Ax) = \sum_{\alpha_0=1}^{r_0} \sum_{\alpha_1=1}^{r_1} \cdots \sum_{\alpha_d=1}^{r_d} \sqrt{\lambda(\alpha_{d-1})} \psi_1(\alpha_0; a_1 \cdot x; \alpha_1) \psi_2(\alpha_1; a_2 \cdot x; \alpha_2) \cdots \psi_d(\alpha_{d-1}; a_d \cdot x; \alpha_d),
\]

which we call a tensor ridge function. In (12), \( a_i \) denotes the \( i \)-th row of the matrix \( A \) and “\( \cdot \)” is the standard dot product on \( \mathbb{R}^d \), and \( r_0 = r_d = 1 \). When we consider \( u_{\text{TT}}(Ax) = u_{\text{TT}}(y) \) in coordinates \( y \), equation (12)
is the FTT expansion for \( u_{\text{TT}}(y) \). However, when we consider \( v(x) = u_{\text{TT}}(Ax) \) in coordinates \( x \) we have that each mode \( \psi_i(\alpha_i; a_i \cdot x; \alpha_i) \) in the tensor ridge function (12) is no longer a univariate function of \( x_i \) as in (8), but rather a \( d \)-variate ridge function, which, has the property of being constant in all directions orthogonal to the vector \( a_i \) (e.g., [7, 34]). An important problem is determining the FTT expansion (13) from (12) based on coordinate flows.

### 3.1. Computing tensor ridge functions via coordinate flows

Consider the non-autonomous linear dynamical system

\[
\begin{align*}
\frac{dy(\epsilon)}{d\epsilon} &= B(\epsilon)y(\epsilon), \\
y(0) &= x,
\end{align*}
\]

where \( y(\epsilon) \in \mathbb{R}^d \), and \( B(\epsilon) \) is a given \( d \times d \) matrix with real entries for all \( \epsilon \geq 0 \). It is well-known that the solution to (14) can be written as

\[
y(\epsilon) = \Phi(\epsilon)x,
\]

where

\[
\Phi(\epsilon) = e^{M(\epsilon)}
\]

is an invertible linear mapping on \( \mathbb{R}^d \) for each \( \epsilon \geq 0 \). The matrix \( M(\epsilon) \) can be represented by the Magnus series (e.g., [7])

\[
M(\epsilon) = \int_0^\epsilon B(\epsilon_1)d\epsilon_1 - \frac{1}{2} \int_0^\epsilon \int_0^{\epsilon_1} B(\epsilon_2)d\epsilon_2 B(\epsilon_1) d\epsilon_1 + \cdots,
\]

where \( \{\cdot, \cdot\} \) denotes the matrix commutator

\[
\{P, Q\} = PQ - QP.
\]

Now that we have introduced coordinate flows and their connection to linear coordinate transformations, let us consider the problem of determining the FTT expansion of \( u_{\text{TT}}(Ax) \) when \( A \) is generated by a coordinate flow, i.e., \( A = \Phi(\epsilon) \) for some \( \Phi \) and some \( \epsilon \geq 0 \). Differentiating \( v(x; \epsilon) = u_{\text{TT}}(\Phi(\epsilon)x) \) with respect to \( \epsilon \) yields the hyperbolic PDE

\[
\frac{\partial v(x; \epsilon)}{\partial \epsilon} = \frac{\partial u_{\text{TT}}(\Phi(\epsilon)x)}{\partial \epsilon} = \nabla u_{\text{TT}}(\Phi(\epsilon)x) \cdot \left( \frac{\partial \Phi(\epsilon)}{\partial \epsilon} x \right) = \nabla u_{\text{TT}}(\Phi(\epsilon)x) \cdot (B(\epsilon)\Phi(\epsilon)x),
\]
where in the second line we used the chain rule and in the third line we used equations (14) and (15). Recalling \( \Phi(0) = I_{d \times d} \) (identity matrix), we see that the initial state \( v(x; 0) = u_{TT}(x) \) is in FTT format. Thus, we have derived the following hyperbolic initial value problem for the tensor ridge function \( v(x; \epsilon) \)

\[
\begin{align*}
\frac{\partial v(x; \epsilon)}{\partial \epsilon} &= \nabla v(x; \epsilon) \cdot (B(\epsilon) \Phi(\epsilon) x), \\
v(x; 0) &= u_{TT}(x),
\end{align*}
\tag{20}
\]

with an initial condition that is given in an FTT format.

Integrating the PDE (20) forward in \( \epsilon \) on a FTT tensor manifold, e.g. using rank-adaptive step-truncation methods [27, 36, 23] or dynamic tensor approximation methods [13, 12, 11, 28, 24], results in a FTT approximation of \( v_{TT}(x; \epsilon) \) of the function \( v(x; \epsilon) \) for all \( \epsilon \geq 0 \). The computational cost of this approach for computing the FTT expansion of a FTT-ridge function is precisely the same cost as solving the hyperbolic PDE (20) in the FTT format, which in the case of step-truncation or dynamic approximation has computational complexity that scales linearly with \( d \). Note that the accuracy of \( v_{TT}(x; \epsilon) \) as an approximation of \( v(x; \epsilon) \) depends on the \( \epsilon \) step-size and order of integration scheme used to solve the PDE (20).

Using coordinate flows, it is straightforward to compute the FTT expansion of a tensor ridge function \( u_{TT}(Ax) \) when the matrix \( A \) admits a real matrix logarithm \( L \). In this case, setting \( B(\epsilon) = L \) yields \( \Phi(\epsilon) = e^{\epsilon L} \), and therefore \( A = \Phi(1) \). This means that we need to integrate (20) with \( B(\epsilon) = L \) up to \( \epsilon = 1 \) to obtain the FTT approximation of the tensor ridge function \( u_{TT}(Ax) \). Let us provide a simple example.

**An example.** Consider the two-dimensional Gaussian function depicted in Figure 1(a), i.e.,

\[
u_{TT}(x) = e^{-x_1^2 - x_2^2/10},\tag{21}\]

Clearly, (21) is the product of two univariate functions and therefore the FTT tensor representation coincides with (21) and has rank equal to one. Next, consider a simple linear coordinate transformation \( \Phi(\epsilon) \) which rotates the \( (x_1, x_2) \)-plane by an angle of \( \epsilon \) radians. It is well-known that

\[
\Phi(\epsilon) = e^{\epsilon L},\tag{22}\]

where

\[
L = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix},\tag{23}\]

is the infinitesimal generator of the two-dimensional rotation. The dynamical system (14) defining the coordinate flow \( y(\epsilon) = \Phi(\epsilon)x \) can be written as

\[
\begin{align*}
\frac{dy(\epsilon)}{d\epsilon} &= \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} y(\epsilon), \\
y(0) &= x.
\end{align*}\tag{24}\]

The tensor ridge function corresponding to the coordinate map \( \Phi(\epsilon) \) is given analytically by

\[
v(x; \epsilon) = u_{TT}(\Phi(\epsilon)x) = e^{-(\Phi_{11}(\epsilon)x_1 + \Phi_{12}(\epsilon)x_2)^2} e^{-(\Phi_{21}(\epsilon)x_1 + \Phi_{22}(\epsilon)x_2)^2/10}.	ag{25}\]

The hyperbolic PDE (20) for \( v(x; \epsilon) \) in this case is given by

\[
\begin{align*}
\frac{\partial v(x; \epsilon)}{\partial \epsilon} &= -x_2 \frac{\partial v(x; \epsilon)}{\partial x_1} + x_1 \frac{\partial v(x; \epsilon)}{\partial x_2}, \\
v(x; 0) &= u_{TT}(x),
\end{align*}\tag{26}\]
It is straightforward to verify that (25) satisfies (26). Note that $v(x; \epsilon)$ in (25) is not an FTT tensor if $\epsilon \neq \pi k/2$ and $k \in \mathbb{N}$. To compute the FTT representation of $v_{TT}(x; \epsilon)$ we can solve the PDE (26) on a tensor manifold using step-truncation or dynamic tensor approximation methods [11] [13] [36] [38]. Given the low dimensionality of the spatial domain in this example ($d = 2$), we can also evaluate (25) directly, and compute its FTT decomposition by solving an eigenvalue problem. In Figure 1(b) we provide a contour plot of $v(x; \epsilon)$ versus $\epsilon$ for all $\epsilon \in [0, \pi/4]$. Of course such a plot can be mirrored to obtain the rank for $\epsilon \in [\pi/4, \pi/2]$, $[\pi/2, 3\pi/4]$, and $[3\pi/4, \pi]$.

4. Tensor rank reduction via coordinate flows

The coordinate flows we introduced in the previous section can be used to morph a given function into another one that has a faster decay rate of FTT singular values, i.e., a FTT tensor with lower rank (after truncation). A simple example is the coordinate flow that rotates the Gaussian function in Figure 1(b) back to the rank-one state depicted in Figure 1(a). This example and the examples documented in subsequent sections indicate that symmetry of the function relative to the new coordinate system plays an important role in reducing the tensor rank.

The problem of tensor rank reduction via linear coordinate flows can be formulated as follows: how do we choose an invertible linear coordinate transformation $A$ so that $v_{TT}(x) \approx u_{TT}(Ax)$ has smaller rank than $u_{TT}(x)$ (eventually minimum rank)? The mathematical statement of this optimization problem is

$$A = \text{argmin}_{A \in \text{GL}_d(\mathbb{R})} \text{rank} \left[ v_{TT}(x) \right],$$

(27)

where $\text{GL}_d(\mathbb{R})$ denotes the set of $d \times d$ real invertible matrices, $\text{rank}[]$ is a metric related to the FTT rank, and $v_{TT}(x)$ is a FTT approximation of $u_{TT}(Ax)$. In equation (27) we have purposely left the cost function unspecified, as some care must be taken in its definition to ensure that the optimization problem is both feasible and computationally effective in reducing rank. One possibility is to define $\text{rank}[v_{TT}(x)]$ to return the sum of all $d$ entries of the multilinear rank vector $r$ corresponding to the FTT tensor $v_{TT}(x)$. While such a cost function is effective in measuring tensor rank it yields a NP-hard rank optimization problem [27].

4.1. Non-convex relaxation for the rank minimization problem

A common relaxation for rank minimization problems is to replace the rank cost function with the sum of the singular values. To describe this relaxation in the context of FTT tensors we first recall that any FTT tensor $u_{TT}(x)$ can be orthogonalized in the $i$-th variable as (e.g., [11])

$$u_{TT}(x) = Q_{\leq i} \Sigma_i Q_{>i},$$

(28)

where

$$\Sigma_i = \text{diag}(\sigma_i(1), \sigma_i(2), \ldots, \sigma_i(r_i)),$$

(29)

is a diagonal matrix with real entries (singular values of $u_{TT}$). The matrices $Q_{\leq i}$ and $Q_{>i}$ are defined as partial products

$$Q_{\leq i} = Q_1 Q_2 \cdots Q_{i-1}, \quad Q_{>i} = Q_{i+1} Q_{i+2} \cdots Q_d,$$

(30)

and they satisfy the orthogonality conditions

$$\langle Q_{\leq i}^T Q_{\leq i} \rangle_{\leq i} = I_{r_i \times r_i}, \quad \langle Q_{>i}^T Q_{>i} \rangle_{>i} = I_{r_i \times r_i}.$$

(31)

Here, $\langle \cdot \rangle_{\leq i}$ and $\langle \cdot \rangle_{>i}$ are the averaging operators

$$\langle W \rangle_{\leq i} (j, k) = \int_{\Omega_{\leq i}} w(j; x; k) d\mu_{\leq i}(x), \quad \langle W \rangle_{>i} (j, k) = \int_{\Omega_{>i}} w(j; x; k) d\mu_{>i}(x),$$

(32)
which map an arbitrary \( r_i \times r_i \) matrix-valued function \( W(x) \) with entries \( w(j; x; k) \) into another \( r_i \times r_i \) matrix-valued function depending on a smaller number of variables. Using the orthogonalization (28) for each \( i = 1, 2, \ldots, d - 1 \), we define the functions

\[
S_i : L^2_\mu(\Omega) \to \mathbb{R}
\]

\[
u_{TT}(x) \mapsto \sum_{\alpha_i = 1}^{r_i} \sigma_i(\alpha_i),
\]

which returns the sum of the singular values corresponding to the \( i \)-th component of the multilinear rank vector. Using these functions we define

\[
S : L^2_\mu(\Omega) \to \mathbb{R}
\]

\[
u_{TT}(x) \mapsto \sum_{i=1}^{d-1} \sum_{\alpha_i = 1}^{r_i} \sigma_i(\alpha_i),
\]

which is a relaxation of the rank cost function appearing in (27). An analogous relaxation of the rank cost function called the matrix nuclear norm has been studied extensively for matrix rank minimization problems [35, 27]. The function (34) has also been used as a relaxation of rank \([\cdot] \) in tensor completion [5]. Next, we proceed by selecting an appropriate search space for the rank cost function. The largest search space we may choose is \( GL_d(\mathbb{R}) \) as we have done in (27). This, however, is not a good choice since transformations in \( GL_d(\mathbb{R}) \) are not volume-preserving and hence do not preserve \( L^2 \) norm, i.e., \( \|u_{TT}(Ax)\|_{L^2_\mu(\Omega)} \neq \|u_{TT}(x)\|_{L^2_\mu(\Omega)} \).

Transformations which do not preserve the norm of \( u_{TT}(x) \) can reduce the rank cost function while having no impact on the tensor rank relative to the tensor’s norm. Therefore we choose \( SL_d(\mathbb{R}) \subset GL_d(\mathbb{R}) \) as the search space, i.e., the collection of invertible matrices with determinant equal to one. These transformations are obviously volume-preserving, and therefore they preserve the \( L^2 \) norm of \( u_{TT} \).

Since the domain of \( S \) in (34) is \( L^2_\mu(\Omega) \) and the cost function must be defined on the search space \( SL_d(\mathbb{R}) \), we define an evaluation map \( E \) corresponding to \( u_{TT}(x) \)

\[
E : SL_d(\mathbb{R}) \to L^2_\mu(\Omega)
\]

\[
A \mapsto u_{TT}(Ax).
\]

Composing \( E \) with \( S \) yields the following (non-convex) relaxation of the cost function rank \([\cdot] \) in (27)

\[
C = S \circ E : SL_d(\mathbb{R}) \to \mathbb{R}.
\]

The function \( C(A) \) returns the sum of the singular values of the tensor ridge function \( u_{TT}(Ax) \), where \( A \) is an invertible matrix with determinant equal to one. The optimization problem corresponding to the cost function and search space discussed above is

\[
A = \arg\min_{A \in SL_d(\mathbb{R})} C(A).
\]

While (38) is simpler than (27), it is still a computationally challenging problem for a number of reasons: First, it is non-convex. Second, when considered as a subset of all \( d \times d \) matrices, the search space \( SL_d(\mathbb{R}) \) is

\[
\|u_{TT}(Ax)\|_{L^p_\mu(A^{-1}\Omega)} = \|u_{TT}(x)\|_{L^p_\mu(\Omega)}, \quad p = 1, 2.
\]

---

\(^2\)It is straightforward to show with a simple change of variables that volume-preserving transformations preserve many quantities that are defined via an integral. For example, for any \( u_{TT}(x) \in L^p_\mu(\Omega) \) and \( A \in SL_d(\mathbb{R}) \) we have

\[
\|u_{TT}(Ax)\|_{L^p_\mu(A^{-1}\Omega)} = \|u_{TT}(x)\|_{L^p_\mu(\Omega)},
\]
subject to a non-trivial set of constraints, e.g., \( \det(A) = 1 \). Third, we note that the set \( \text{SL}_d(\mathbb{R}) \) is unbounded (matrices with determinant 1 can have entries that are arbitrarily large). This can yield convergence issues when looking for the minimizer \((38)\). In order to overcome this problem one may introduce linear inequality constraints on the entries of the matrix \( A \), i.e., optimize over bounded subsets of \( \text{SL}_d(\mathbb{R}) \) such as the set of rotation matrices with determinant 1.

Hereafter we develop a new method for obtaining a local minimum of \((38)\). To handle the search space constraints, we give \( \text{SL}_d(\mathbb{R}) \) a Riemannian manifold structure and perform gradient descent on this manifold \( \Pi \). To obtain the gradient of \( C(A) \) efficiently we build its computation into the FTT truncation procedure at a negligible additional computational cost.

### 4.2. Riemannian gradient descent path

In Lemma \[B4\] we show that if the FTT singular values of \( u_{TT}(Ax) \) are simple (i.e., distinct) then the cost function \( C(A) \) defined in \((37)\) is differentiable in \( A \). With this sufficient condition for the smoothness of the cost function established, we can use the machinery of Riemannian geometry summarized in Appendix A to construct a gradient descent path for the minimization of \((38)\) on the search space \( \text{SL}_d(\mathbb{R}) \). To this end, let us denote by \( \Gamma(\epsilon) \) such gradient descent path. To build \( \Gamma(\epsilon) \), we start at the identity \( \Gamma(0) = I \) and consider the matrix ordinary differential equation

\[
\begin{aligned}
\frac{d\Gamma(\epsilon)}{d\epsilon} &= -\text{grad} [C(\Gamma(\epsilon))], \\
\Gamma(0) &= I,
\end{aligned}
\]

where \( \text{grad} [C(\Gamma(\epsilon))] \) is the Riemannian gradient of the cost function \( C \) defined in \((37)\). By construction, the vector \( -\text{grad} [C(\Gamma(\epsilon))] \) is tangent to the manifold \( \text{SL}_d(\mathbb{R}) \) at each point \( \Gamma(\epsilon) \), and thus \( \Gamma(\epsilon) \in \text{SL}_d(\mathbb{R}) \) for all \( \epsilon \geq 0 \). Since \( -\text{grad} [C(\Gamma(\epsilon))] \) points in the direction of steepest descent of the cost function \( C \) at the point \( \Gamma(\epsilon) \), the cost function is guaranteed to decrease along the path \( \Gamma(\epsilon) \) (or remain constant which implies we have obtained a local minimum). In Figure 2 we provide an illustration of the Riemannian gradient descent path \( \Gamma(\epsilon) \). For computational efficiency, it is essential to have a fast method for computing the Riemannian gradient of the cost function \( C \) at an arbitrary point \( A \in \text{SL}_d(\mathbb{R}) \). The following Proposition provides an expression for such Riemannian gradient in terms of orthogonal FTT cores which we will use to efficiently compute the descent direction. The proof is given in Appendix B.

**Proposition 4.1.** The Riemannian gradient of the cost function \((37)\) at the point \( A \in \text{SL}_d(\mathbb{R}) \) is given by

\[
\text{grad} [C(A)] = DA,
\]

where

\[
D = \sum_{i=1}^{d-1} \int_{\Omega} Q_{\leq i} Q_{>i} \left( \nabla v(x) (Ax)^T - \frac{\nabla v(x)^T Ax}{d} I_{d\times d} \right) d\mu(x),
\]

\[
v(x) = u_{TT}(Ax) \text{ and } Q_{\leq i}, Q_{>i} \text{ are tensor cores of the orthogonalized FTT}
\]

\[
v(x) = Q_{\leq i} \Sigma_i Q_{>i}.
\]

Using the gradient descent path defined by \((39)\) we differentiate the coordinate transformation

\[
y(\epsilon) = \Gamma(\epsilon)x
\]

with respect to \( \epsilon \) and use \((39)-(40)\) to obtain

\[
\begin{aligned}
\frac{dy(\epsilon)}{d\epsilon} &= -D(\epsilon)y(\epsilon), \\
y(0) &= x.
\end{aligned}
\]
Figure 2: An illustration of a descent path $\Gamma(\epsilon)$ for the cost function $C$ defined in (37). At each point of the path $\Gamma(\epsilon) \in \text{SL}_d(\mathbb{R})$ we assign the tangent vector $-\nabla C(\Gamma(\epsilon)) \in T_{\Gamma(\epsilon)}\text{SL}_d(\mathbb{R})$ which points in the direction of steepest descent $-\langle d_{\Gamma(\epsilon)}C \rangle \nabla C(\Gamma(\epsilon)) \in T_{C(\Gamma(\epsilon))}\mathcal{M}$ of the function $C$ at the point $C(\Gamma(\epsilon)) \in \mathcal{M}$.

Note that (44) has the same form as the ODE (14) we used to define coordinate flows. Hence, by construction, the flow map generated by (44) is the descent path $\Gamma(\epsilon)$ on the manifold $\text{SL}_d(\mathbb{R})$, which converges to a local minimum of $C$ as $\epsilon$ increases. By defining the function $v(x; \epsilon) = u^{TT}(\Gamma(\epsilon)x)$ and differentiating it with respect to $\epsilon$ we obtain the hyperbolic PDE

$$
\begin{cases}
\frac{\partial v(x; \epsilon)}{\partial \epsilon} = -\nabla v(x; \epsilon) \cdot [D(\epsilon)\Gamma(\epsilon)x], \\
v(x; 0) = u^{TT}(x).
\end{cases}
$$

Note that the evolution of $y(\epsilon) = \Gamma(\epsilon)x$ at the right hand side of (45) is defined by (44). Integrating (44)-(45) forward in $\epsilon$ yields a rank-reducing linear coordinate transformation $\Gamma(\epsilon)$ and the reduced rank function $v(x; \epsilon) = u^{TT}(\Gamma(\epsilon)x)$.

4.3. Numerical integration of the gradient descent equations

It is convenient to use a step-truncation method \cite{36,38,23} to integrate the initial value problem (45) on a FTT tensor manifold. This is because applying the FTT truncation operation to $v(x; \epsilon)$ requires computing the orthogonalized tensor cores $Q_{\leq i}(\epsilon), Q_{> i}(\epsilon)$ ($i = 1, 2, \ldots, d - 1$) which can then be readily used to evaluate the matrix $D(\epsilon)$ defining the Riemannian gradient (41). Moreover, the FTT truncation operation applied to $v(x; \epsilon)$ yields an expansion of the form (13), which is the desired tensor format. To describe the integration algorithm in more detail, let us discretize the interval $[0, \epsilon_f]$ into $N + 1$ evenly-spaced points

$$
\epsilon_i = i\Delta\epsilon, \quad \Delta\epsilon = \frac{\epsilon_f}{N}, \quad i = 0, 1, \ldots, N,
$$

and let $v_i, \Gamma_i, D_i$ denote $v(x; \epsilon_i), \Gamma(\epsilon_i), D(\epsilon_i)$, respectively. Let

$$
v_{i+1} = v_i + \Delta\epsilon \Phi(v_i, D_i, \Delta\epsilon),
$$

$$
\Gamma_{i+1} = \Gamma_i + \Delta\epsilon \hat{\Phi}(\Gamma_i, D_i, \Delta\epsilon)
$$

The right end-point $\epsilon_f$ will ultimately be determined by the stopping criterion for gradient descent.
be one-step explicit integration schemes approximating the solution to the initial value problem (45) and the matrix ODE (39), respectively. In order to guarantee that the solution \( v_{i+1} \) is a low-rank FTT tensor, we apply a truncation operator to the right hand side of (47). This yields the step-truncation method \([36]\)

\[
v_{i+1} = \Xi_{\delta} \left( v_i + \Delta \epsilon \Phi \left( v_i, D_i, \Delta \epsilon \right) \right).
\]

(49)

Here, \( \Xi_{\delta} \) denotes the standard FTT truncation operator with relative accuracy \( \delta \) proposed in \([33]\) modified to return the matrix \( D_{i+1} \). A detailed description of the modified tensor truncation algorithm is provided in section 4.4.

At this point, a few remarks regarding the integration scheme (47)-(49) are in order. First, we notice that at each step of the gradient descent algorithm we are computing the gradient (40) at the identity matrix since the current tensor \( v_i \) is the tensor ridge function \( v_i(x) = u_{TT}(\Gamma_i, x) \). Second, the accuracy of the tensor \( v_i \) approximating \( u_{TT}(\Gamma, x) \) is determined by the chosen integration scheme and its relevant parameters (i.e., the function \( \Phi \), the step size \( \Delta \epsilon \), and the accuracy of \( \Xi_{\delta} \)). In a standard gradient descent algorithm, convergence can be expedited with a line-search routine that determines an appropriate step-size to take in the descent direction. However, in the proposed integration scheme the step-size determines the accuracy of the final tensor, thus we keep the step-size \( \Delta \epsilon \) fixed during gradient descent. We set a stopping criterion for the integration of (47)-(49) based on the empirical observation that the cost function \( C \) does not decrease substantially along the descent path \( \Gamma(\epsilon) \) when \( \Gamma \) is close to a local minimum. Mathematically, this translates into the condition

\[
\frac{dS(v_i)}{d\epsilon} > -\eta,
\]

(50)

where \( \eta \) is some predetermined tolerance. Since the solution history \( v_i, v_{i-1}, \ldots \) is available during gradient descent, a simple method for approximating \( dS(v_i)/d\epsilon \) is a \( p \)-point backwards difference stencil

\[
\frac{dS(v_i)}{d\epsilon} \approx BD(p)(S(v_i), S(v_{i-1}), \ldots, S(v_{i-p})).
\]

(51)

In addition to the stopping criterion (50), we also set a maximum number of iterations \( M_{iter} \) to ensure that the Riemannian gradient descent algorithm halts within a reasonable amount of time. We summarize the proposed Riemannian gradient descent method to compute a local minimum of (38) in Algorithm 1.

4.4. Modified tensor truncation algorithm

To speed up numerical integration of the gradient descent equations (48)-(49) it is convenient to compute the matrix \( D_i \) defined in (41) during FTT truncation. The reason being that standard FTT truncation requires the computation of all orthogonal FTT cores \( Q_{\leq j} \) and \( Q_{> j} \), which, can be readily used to compute \( D_i \). To describe the modified tensor truncation algorithm, let \( v_{TT}(x) = \Psi_1 \cdots \Psi_d \) be an FTT tensor with non-optimized rank, e.g., \( v_{TT}(x) \) is the result of adding two FTT tensors together. Denote by \( \Xi_{\delta} \) the modified truncation operator, where \( \delta \) is the required relative accuracy, i.e.,

\[
\|v_{TT}(x) - \Xi_{\delta}(v_{TT}(x))\|_{L_2^p(\Omega)} \leq \delta \|v_{TT}(x)\|_{L_2^p(\Omega)}.
\]

(52)

We also define

\[
\hat{\delta} = \frac{\delta}{\sqrt{d-1}} \|v_{TT}(x)\|_{L_2^p(\Omega)},
\]

(53)

which is the required accuracy for each SVD in the FTT truncation algorithm. As described in [11], we may perform a functional analogue of the QR decomposition on the FTT tensor cores of \( v_{TT}(x) \)

\[
\Psi_i = Q_i R_i.
\]

(54)
Algorithm 1: Riemannian gradient descent for computing rank-reducing linear coordinate maps.

Input:
$u_{TT}$ → initial FTT tensor,
$\Delta \epsilon$ → step-size for gradient descent,
$\eta$ → stopping tolerance,
$M_{\text{iter}}$ → maximum number of iterations.

Output:
$\Gamma$ → rank-reducing linear coordinate transformation,
$v_{TT}$ → reduced rank FTT tensor on transformed coordinates $v_{TT}(x) = u_{TT}(\Gamma x)$.

Runtime:
\[
[v_0, S(v_0), D_0] = \mathcal{F}(u_{TT}),
\Gamma_0 = I, 
\dot{S}(v_0) = -\infty, 
i = 0.
\]

while $\dot{S}(v_i) < -\eta$ and $i \leq M_{\text{iter}}$
\[
\begin{align*}
    v_{i+1} &= v_i + \Delta \epsilon \Phi(v_i, D_i, \Delta \epsilon), \\
    [v_{i+1}, S(v_{i+1}), D_{i+1}] &= \mathcal{F}(v_{i+1}), \\
    \Gamma_{i+1} &= \Gamma_i + \Delta \epsilon \mathcal{F}(\Gamma_i, D_i, \Delta \epsilon), \\
    \dot{S}(v_{i+1}) &= B D^{(p)}(S(v_{i+1}), S(v_i), \ldots, S(v_{i+1-p})), \\
    i &= i + 1, \\
    \Gamma &= \Gamma_i, \\
    v_{TT} &= v_i.
\end{align*}
\]
end

Figure 3: A summary of the modified FTT truncation algorithm for computing the Riemannian gradient.\[\]
Algorithm 2: Modified FTT truncation algorithm.

**Input:**
$v \rightarrow$ FTT tensor with cores $\Psi_1, \Psi_2, \ldots, \Psi_d$, 
$\delta \rightarrow$ desired accuracy.

**Output:**
$v_{TT} \rightarrow$ truncated FTT tensor satisfying $\|v - v_{TT}\|_{L^2(\Omega)} \leq \delta \|v\|_{L^2(\Omega)}$, 
$S(v_{TT}) \rightarrow$ sum of all multilinear singular values of $v_{TT}$,
$D \rightarrow$ left factor of the Riemannian gradient (40).

**Runtime:**
$$\hat{\delta} = \frac{\delta}{\sqrt{d-1}} \|v\|_{L^2} \quad \text{(Set truncation parameter)}$$

$$S(v) = 0 \quad \text{(Initialize $S(v)$)}$$

for $i = 1$ to $d - 1$ 
Left-to-right orthogonalization
$$[Q_i, R_i] = QR(\Psi_i)$$
$$\Psi_{i+1} = R_i \Psi_{i+1}$$
end

for $i = d$ to $2$ 
Right-to-left truncation and gradient computation
$$[L_i, Q_i] = LQ(\Psi_i)$$
$$[U_i, \Sigma_i, V_i] = \text{SVD}_\delta(L_i)$$
$$Q_i = V_i^T Q_i$$
$$Q_{i-1} = Q_{i-1} U_i,$$
$$S(v) = S(v) + \text{sum} (\Sigma_i),$$
$$D^{(i-1)} = \int Q_{\preceq i-1} Q_{\succ i-1} (\nabla v(x)x^T) d\mu(x)$$
end

$$D = \sum_{i=1}^{d-1} D^{(i)}.$$ 

$$v_{TT} = Q_1 \Sigma_2 Q_2 \ldots Q_d.$$ 

where $Q_i$ is a $r_{i-1} \times r_i$ matrix with elements in $L^2_{\mu_i}(\Omega_i)$ satisfying $\langle Q_i^T Q_i \rangle_i = I_{r_i \times r_i}$, and $R_i$ is an upper triangular $r_i \times r_i$ matrix with real entries. Similarly, we can perform an LQ-factorization
$$\Psi_i = L_i Q_i,$$  (55)

where $Q_i$ is a $r_{i-1} \times r_i$ matrix with elements in $L^2_{\mu_i}(\Omega_i)$ satisfying $\langle Q_i^T Q_i \rangle_i = I_{r_i \times r_i}$, and $L_i$ is a lower-triangular $r_i \times r_i$ matrix with real entries. The first procedure in the modified truncation routine is a left-to-right orthogonalization sweep in which we first perform the QR decomposition
$$\Psi_1 = Q_1 R_1,$$  (56)
and then update the core $\Psi_2$

$$\Psi_2 = R_1 \Psi_2.$$  \hfill (57)

This process is repeated recursively

$$\Psi_i = Q_i R_i, \quad \Psi_{i+1} = R_i \Psi_{i+1}, \quad i = 2, \ldots, d - 1,$$  \hfill (58)

resulting in the orthogonalization

$$v_{\text{FTT}}(x) = Q_1 Q_2 \cdots Q_{d-1} \Psi_d.$$  \hfill (59)

Next, we perform a right-to-left sweep which compresses $v_{\text{FTT}}(x)$ and simultaneously computes each term appearing in the summation of $D$ in equation (41). The first step of this procedure is to compute a LQ decomposition of $\Psi_d$

$$\Psi_d = L_d Q_d,$$  \hfill (60)

and then perform a truncated singular value decomposition of $L_d$ with threshold $\hat{\delta}$

$$L_d = U_d \Sigma_d V_d^T.$$  \hfill (61)

Substituting (60) and (61) into (59) yields

$$v_{\text{FTT}}(x) = Q_1 Q_2 \cdots Q_{d-1} \Sigma_d Q_d,$$  \hfill (62)

where we re-defined

$$Q_{d-1} = Q_{d-1} U_d, \quad Q_d = V_d^T Q_d.$$  \hfill (63)

At this point we have truncated the FTT tensor $v_{\text{FTT}}(x)$ in the $d$-th variable. The expansion (63) provides the FTT orthogonalization needed to compute the $(d-1)$-th term in the sum (41)

$$D^{(d-1)} = \int_{\Omega} Q_{\leq d-1} Q_{>d} (\nabla v_{\text{FTT}}(x) x^T) d\mu(x),$$  \hfill (64)

which, can be computed efficiently by applying one-dimensional differentiation matrices and quadrature rules to the FTT cores. We proceed in a recursive manner ($i = d - 1, \ldots, 2$) with the same steps described above for $\Psi_d$. First compute the LQ decomposition

$$(Q_i \Sigma_{i+1}) = L_i Q_i,$$  \hfill (65)

and then perform a singular value decomposition with threshold $\hat{\delta}$

$$L_i = U_i \Sigma_i V_i^T.$$  \hfill (66)

Then rewrite $v_{\text{FTT}}(x)$ as

$$v_{\text{FTT}}(x) = Q_1 Q_2 \cdots Q_{i-1} \Sigma_i Q_i \cdots Q_d,$$  \hfill (67)

where we re-defined

$$Q_{i-1} = Q_{i-1} U_i, \quad Q_i = V_i^T Q_i.$$  \hfill (68)

The expansion (67) provides the FTT orthogonalization required to compute the $(i-1)$-th term in the sum (41)

$$D^{(i-1)} = \int_{\Omega} Q_{\leq i-1} Q_{>i} (\nabla v_{\text{FTT}}(x) x^T) d\mu(x),$$  \hfill (69)
which, can be computed efficiently by applying one-dimensional differentiation matrices and quadrature rules to the FTT cores. Finally with all of the terms in (41) computed we simply sum them to obtain the matrix
\[ D = \sum_{i=1}^{d-1} D^{(i)}. \] (70)

We summarize the main steps of the modified tensor truncation in Figure 3 and in Algorithm 2. Clearly, the matrix \( D \) needs to be recomputed at each step \( \epsilon_i \), resulting in the matrix \( D_i \) appearing in the gradient descent equations (48)-(49).

4.5. Computational cost

One step of a first-order step-truncation integrator of the form (49) (e.g., Euler forward) without computing the left factor of the Riemannian gradient (41) requires one multiplication between a scalar and a TT tensor (\( O(nr^2) \) FLOPS), one addition between two TT tensors, and one FTT truncation (\( O(dnr^3) \) FLOPS) for a total computational complexity \( O(dnr^3) \). In the above estimate we assumed that each entry of the rank of the FTT tensor \( v_i + \Delta \Phi(v_i, D_i, \Delta \epsilon) \) is bounded by \( r \) and \( v_i \) is discretized on a grid with \( n \) points in each variable. In addition to the cost of step-truncation, we must also compute the matrix \( D_i \) defined in (41) at each step. The orthogonal FTT cores \( Q_{\leq j}, Q_{> j} \) needed for the computation of \( D_i \) are readily available during the tensor truncation procedure. For the computation of \( D_i \) we must compute the gradient of \( \nabla v_i \), which requires \( d \) matrix multiplications between a differentiation matrix of size \( n \times n \) and a FTT core \( \Psi_i \) of size \( n \times rn \) for total complexity of \( O(dnr^3) \). We also need to compute the outer product \( (\nabla v_i) x^T \) where each entry of \( \nabla v_i \) is in FTT format, and, for each of the \( d^2 \) entries in the matrix \( (\nabla v_i) x^T \) compute an integral of FTT tensors requiring \( O(dnr^3) \) FLOPS. The final estimate for computing the matrix \( D_i \) is \( O(d^2nr^3) \), which dominates the cost of performing one-step of (49). We point out that the computation of \( D_i \) may be incorporated into high performance computing algorithms for tensor train rounding, e.g., [2][41].

5. Application to multivariate functions

We now demonstrate the Riemannian gradient descent algorithm for generating rank-reducing linear coordinate transformations. Consider the Gaussian mixture
\[ u(x) = \sum_{i=1}^{N_g} w_i \exp \left( -\sum_{j=1}^{d} \frac{1}{\beta_{ij}} (R_{ij}^{(i)} \cdot x + t_{ij})^2 \right), \] (71)
where \( N_g \) is the number of Gaussians, \( \beta_{ij} \) are positive real numbers, \( w_i \) are positive weights satisfying
\[ \sum_{i=1}^{d} w_i = 1, \]
\( R_{ij}^{(i)} \) is the \( j \)-th row of a \( d \times d \) rotation matrix \( R^{(i)} \), and \( t_{ij} \) are translations. For our demonstration we consider three spatial dimensions (\( d = 3 \)) and set \( N_g = 3 \), \( w_i = 1/3 \),
\[
\begin{align*}
\beta_{11} &= 2, &\beta_{12} &= 1/3, &\beta_{13} &= 1/2, & t_{11} &= 0, & t_{12} &= 0, & t_{13} &= 0, \\
\beta_{21} &= 3, &\beta_{22} &= 4, &\beta_{23} &= 1/6, & t_{21} &= -1, & t_{22} &= 1/2, & t_{23} &= -1/3, \\
\beta_{31} &= 1, &\beta_{32} &= 1/5, &\beta_{33} &= 5, & t_{31} &= 1/2, & t_{32} &= -1/4, & t_{33} &= 1,
\end{align*}
\] (72)
and the rotation matrices
\[ R^{(i)} = \exp \begin{bmatrix}
0 & \theta_i(1) & \theta_i(2) \\
-\theta_i(1) & 0 & \theta_i(3) \\
-\theta_i(2) & -\theta_i(3) & 0
\end{bmatrix}, \] (73)
Figure 4: (a) Volumetric plot of the three-dimensional Gaussian mixture (71). (b) Volumetric plot of the corresponding reduced rank ridge tensor $v(x; \epsilon_f)$.

Figure 5: Rank reduction problem via coordinate flow for the three-dimensional Gaussian mixture (71). (a) Cost function (38) evaluated along a steepest descent mapping $\Gamma(\epsilon)$ versus $\epsilon$. (b) Absolute value of the derivative of the cost function in (38) versus $\epsilon$. The derivative is computed with a second-order backwards finite difference stencil (51).

with

$$
\begin{align*}
\theta_1 &= \begin{bmatrix} \pi/4 \\ \pi/3 \\ \pi/5 \end{bmatrix}, & \theta_2 &= \begin{bmatrix} \pi/3 \\ \pi/6 \\ \pi/4 \end{bmatrix}, & \theta_3 &= \begin{bmatrix} \pi/3 \\ \pi/3 \\ \pi/7 \end{bmatrix}.
\end{align*}
$$

(74)

We discretize the Gaussian mixture (71) on the computational domain $[-12, 12]^3$ (which is large enough to enclose the numerical support of (71)) using 200 evenly-spaced points in each variable. From the discretization of (71) we compute the FTT decomposition $u_{TT}(x)$ using recursive SVDs. To integrate the PDE (45) for the reduced rank ridge tensor $v(x; \epsilon)$, we use the explicit two-step Adams-Bashforth step-truncation method with $\Delta \epsilon = 10^{-3}$ and relative FTT truncation accuracy $\delta = 10^{-6}$. All spatial derivatives and integrals are computed by applying one-dimensional pseudo-spectral Fourier differentiation matrices and quadrature weights [18] to the tensor modes. We integrate up to $\epsilon_f = 10$ which is sufficient to demonstrate convergence of the gradient descent method. In Figure 4 we provide volumetric plots of the of the function $v(x; \epsilon)$ for $\epsilon = 0$ and $\epsilon = \epsilon_f$. We observe that the reduced rank ridge tensor $v(x; \epsilon_f)$ appears to be more symmetrical with respect to the $(x_1, x_2, x_3)$-axes than the higher rank function $v(x; 0)$. In Figure 5a we plot the cost
Figure 6: Multilinear spectra of the 3-dimensional Gaussian mixture $v(x; 0)$ defined in (71) (Cartesian coordinates) and the corresponding reduced-rank ridge tensor $v(x; \epsilon_f)$ (transformed coordinates). (a) Spectra $\sigma_1$ corresponding to multilinear rank $r_1$. (b) Spectra $\sigma_2$ corresponding to multilinear rank $r_2$. (c) 1-norm of the multilinear rank vector of $v_{TT}(x; \epsilon)$ versus $\epsilon$. It is seen that the coordinate flow $\Gamma(\epsilon)x$ reduces the multilinear rank of the Gaussian mixture (71) from about 63 (Cartesian coordinates) to 31 (transformed coordinates).

function $C(\Gamma(\epsilon))$ versus $\epsilon$ and in Figure 5(b) we plot the absolute value of the derivative of the cost function versus $\epsilon$. Observe in Figure 5 that $\Gamma(\epsilon)$ appears to be converging to a local minimum of $C$, and, the majority of the decrease in the cost function occurs in the interval $\epsilon \in [0, 2]$. Correspondingly, we notice that in Figure 6(c) the majority of rank increase occurs in the interval $\epsilon \in [0, 2]$. Finally, in Figure 6(a)-(b) we plot the multilinear spectra of the function $v_{TT}(x; \epsilon)$ for $\epsilon = 0$ and $\epsilon = \epsilon_f$. The decay rate of the multilinear spectra corresponding to $v(x; \epsilon_f)$ is significantly faster than the decay rate of the multilinear corresponding to the initial function $v(x; 0)$, resulting in a multilinear rank of about half the one in Cartesian coordinates. Thus, for any truncation tolerance $\delta$, the FTT-ridge tensor $v(x; \epsilon_f)$ can be stored at a significantly lower cost than the original function. Intuitively, the savings that can be obtained by the coordinate flow in higher-dimensions are even more pronounced, since hierarchical SVDs with steeper spectra yield a much smaller number of tensor modes.
6. Application to PDEs

We now apply the proposed rank tensor reduction method to initial-value problems of the form

\[
\begin{aligned}
\frac{\partial u(x, t)}{\partial t} &= G(u(x, t), x), \\
u(x, 0) &= u_0(x),
\end{aligned}
\]

(75)

where \(G\) is a nonlinear operator that may incorporate boundary conditions. As is well-known, solving (75) numerically involves repeated application of \(G\). In particular, if the approximate solution of the given PDE is represented as a FTT tensor \(u \approx u_{TT}\) then the operator \(G\) must be represented in a form that can take \(u_{TT}\) as an input and output another FTT tensor. If \(G\) is a linear operator then such a representation is given by the rank \(g\) FTT-operator (or TT-matrix after discretization [33])

\[
G(\cdot, x) \approx G_{TT}(\cdot, x) = \sum_{\alpha_1=1}^{g_1} \sum_{\alpha_2=1}^{g_2} \cdots \sum_{\alpha_{d-1}=1}^{g_{d-1}} A_1(x_1; \alpha_1) \otimes A_2(\alpha_1; x_2; \alpha_2) \otimes \cdots \otimes A_d(\alpha_{d-1}; x_d),
\]

(76)

where, for fixed \(\alpha_{i-1}\) and \(\alpha_i\), \(A_j\) is a one-dimensional operator acting only on functions of \(x_j\). The representation (76) is also known as matrix product operator (MPO) [31]. After applying \(G_{TT}\) to a FTT tensor \(u_{TT}\) with rank \(r\), the new FTT tensor \(G_{TT}(u_{TT}, x)\) has rank given by the element-wise (Hadamard) product of the two ranks \(g \circ r\), which then has to be truncated. The computational cost of such a truncation scales cubically in the new FTT rank. If the product rank \(g \circ r\) is prohibitively large then the FTT operator \(G_{TT}\) can be split into sums of low rank operators

\[
G_{TT} = \sum_{k=1}^{n} G^{(k)}_{TT},
\]

(77)

where each \(G^{(k)}_{TT}\) has FTT operator rank \(g^{(k)}\), which is less than \(g\). Hence, instead of applying \(G_{TT}\) directly to the solution tensor, we can apply each \(G^{(k)}_{TT}\) \((k = 1, 2, \ldots, n)\) to \(u_{TT}\), truncate each \(G^{(k)}_{TT} (u_{TT}, x)\), and then add them together. After the addition, one more truncation procedure must be performed to ensure the result of the FTT addition has optimal ranks. This procedure can be written mathematically as

\[
G_{TT}(u_{TT}, x) \approx \Xi_\delta \left[ \sum_{k=1}^{n} \Xi_\delta \left( G^{(k)}_{TT} (u_{TT}, x) \right) \right],
\]

(78)

where \(\Xi_\delta\) is a truncation (or rounding) operator for FTT tensors with relative accuracy \(\delta\). Alternatively, one can use randomized algorithms, e.g., based on tensor sketching, for computing sums of many TT tensors [10]. This can increase efficiency of applying high rank FTT operators to FTT tensors. For the PDEs considered hereafter we employ the truncation algorithm (78) to mitigate the cost of applying high rank operators.

6.1. Coordinate transformation

For a given PDE operator \(G(\cdot, x)\) and linear coordinate transformation \(y = \Gamma x\), it is always possible to write \(G\) in coordinates \(y\) resulting in a new operator \(G_{\Gamma}\). If \(G\) acts on \(u_{TT}(x, t_k)\) then \(G_{\Gamma}\) acts on the transformed tensor \(u_{TT}(y, t_k) = v_{TT}(x, t_k)\). Such an operator can be constructed using standard tools of differential geometry [3] [44], and usually has different FTT-operator rank than \(G\). For example, consider the variable coefficient advection operator

\[
G(u(x, t), x) = \sum_{i=1}^{d} f_i(x) \frac{\partial u}{\partial x_i}.
\]

(79)
The scalar field \( u(x, t_k) \) can be written in the new coordinate system \( y \) as
\[
u(x, t_k) = u(\Gamma^{-1}y, t_k) = U(y, t_k) = U(\Gamma x, t_k)
\] (80)
which implies that
\[
\frac{\partial u(x, t)}{\partial x_j} = \sum_{k=1}^{d} \Gamma_{kj} \frac{\partial U(y, t_k)}{\partial y_k}.
\] (81)
In this way, we can rewrite the operator (79) in coordinates \( y = \Gamma x \) as
\[
G_\Gamma (U(y, t), y) = d \sum_{i,j=1}^{d} \Gamma_{ij} f_j (\Gamma^{-1}y) \frac{\partial U(y, t)}{\partial y_i},
\] (82)
where
\[
h_i (y) = d \sum_{j=1}^{d} \Gamma_{ij} f_j (\Gamma^{-1}y).
\] (83)
Note that \( G_\Gamma \) has a relatively simple form due to the linearity of the coordinate transformation. In this case, the rank of the operators \( G \) and \( G_\Gamma \) are determined by the FTT ranks of the variable coefficients \( f_i(x) \) and \( h_i(y) \) \((i = 1, 2, \ldots, d)\), respectively.

### 6.2. Time integration

For the time integration of (75) using low-rank tensors we discretize the temporal domain of interest \([0, T]\) into \(N + 1\) evenly-spaced time instants,
\[
t_k = k \Delta t, \quad \Delta t = \frac{T}{N}, \quad k = 0, 1, \ldots, N,
\] (84)
and consider the rank-adaptive step-truncation scheme \([11, 36, 38]\)
\[
u_{TT}(x, t_{k+1}) = \mathcal{T}_\delta \left( u_{TT}(x, t_k) + \Delta t \Phi (G_{TT}(u_{TT}(x, t_k), \Delta t)) \right),
\] (85)
where \( \Phi \) is an iteration function associated with a temporal discretization scheme and \( G_{TT} \) is a FTT-operator approximation \([76]\) of the given operator \( G \). For example, a step-truncation Euler forward scheme is
\[
u_{TT}(x, t_{k+1}) = \mathcal{T}_\delta \left[ u_{TT}(x, t_k) + \Delta t \mathcal{T}_\delta \left( G_{TT}(u_{TT}(x, t_k), x) \right) \right],
\] (86)
while a step-truncation Adams-Bashforth 2 (AB2) scheme \([7]\) can be written as
\[
u_{TT}(x, t_{k+1}) = \mathcal{T}_\delta \left[ u_{TT}(x, t_k) + \Delta t \left( \frac{3}{2} \mathcal{T}_\delta \left[ G_{TT}(u_{TT}(x, t_k), x) \right] - \frac{1}{2} \mathcal{T}_\delta \left[ G_{TT}(u_{TT}(x, t_{k-1}), x) \right] \right) \right].
\] (87)

\[\text{For more general nonlinear coordinate transformations } y = H(x), \text{ the operator } G_{\Gamma} \text{ includes the metric tensor of the coordinate change, which can significantly complicate the form of } G_{\Gamma} \text{ (e.g., [3, 29]).}\]
\[\text{Variants of these step-truncation schemes can be obtained by inserting or removing truncation operations between summations, changing truncation tolerances } \delta \text{ in each of the truncation operators, or by using operator splitting \([77]\).}\]
Algorithm 3: PDE integrator with adaptive rank-reducing coordinate transformations

**Input:**
- $u_0 \rightarrow$ initial condition in FTT tensor format,
- $\Delta t \rightarrow$ temporal step size,
- $N_t \rightarrow$ total number of time steps,
- $\text{max rank} \rightarrow$ maximum rank during FTT integration before attempting rank reduction,
- $\text{max time} \rightarrow$ maximum computational time for one time step before attempting rank reduction,
- $k_r \rightarrow$ increase for maximum rank after performing coordinate transformation,
- $k_t \rightarrow$ increase for maximum time after performing coordinate transformation,
- $\Delta \epsilon \rightarrow$ gradient descent step-size,
- $\eta \rightarrow$ tolerance for coordinate gradient descent,
- $M_{\text{iter}} \rightarrow$ maximum number of iterations for gradient descent routine.

**Output:**
- $\Gamma \rightarrow$ rank-reducing linear coordinate transformation for PDE solution,
- $v_{\text{TT}}(x, t_f) = u_{\text{TT}}(\Gamma x, t_f) \rightarrow$ FTT solution tensor at time $t_f$ on rank-reducing coordinate system.

**Runtime:**

\[
\begin{align*}
\Gamma &= I, \\
v_0 &= u_{\text{TT}}, \\
\text{for } k = 0 \text{ to } N_t \\
\text{if } \text{time} > \text{max time or rank} > \text{max rank} \\
[v_k, \Gamma_{\text{new}}] &= \text{gradient descent}(v_k, \Delta \epsilon, \eta, M_{\text{iter}}) \\
\Gamma &= \Gamma_{\text{new}} \Gamma \\
\text{max rank} &= \text{max rank} + k_r \\
\text{max time} &= \text{max time} + k_t \\
\end{align*}
\]

\[
[v_{k+1}, \text{time}, \text{rank}] = \mathcal{X}_\delta (v_k + \Delta t \Phi(v_k, G_{\text{TT}, \Gamma}, \Delta \epsilon))
\]

At any time step $t_k$ during temporal integration of (75) we may compute a rank-reducing coordinate transformation $y = \Gamma x$ and obtain a tensor ridge representation of the solution at time $t_k$. To integrate the initial boundary value problem (75) using the tensor ridge representation of the solution at time $t_k$, the operator $G_{\text{TT}}$ may be rewritten as a new (FTT) operator $G_{\text{TT}, \Gamma}$ acting in the transformed coordinate system. With the operator $G_{\text{TT}, \Gamma}$ available, we can write the following PDE for $U(y, t)$ corresponding to (75)

\[
\begin{align*}
\frac{\partial U(y, t)}{\partial t} &= G_{\text{TT}, \Gamma}(U(y, t), y), \quad t \geq t_k, \\
U(y, t_k) &= v_{\text{TT}}(y, t_k),
\end{align*}
\]

with initial condition given at time $t_k$. Time integration can proceed in the transformed coordinate system by applying a step-truncation scheme (85)-(87) to the transformed PDE (88) resulting in a step-truncation...
Algorithm 4: PDE integrator with coordinate corrections at each time step.

**Input:**
- $u_0 \rightarrow$ initial condition in FTT tensor format,
- $\Delta t \rightarrow$ temporal step size,
- $N_t \rightarrow$ total number of time steps,
- $\Delta \epsilon \rightarrow$ gradient descent step-size,
- $\eta \rightarrow$ tolerance for coordinate gradient descent,
- $M_{\text{iter}} \rightarrow$ maximum number of iterations for gradient descent routine.

**Output:**
- $\Gamma \rightarrow$ rank-reducing linear coordinate transformation for PDE solution,
- $v_{TT}(x, t_f) = u_{TT}(\Gamma x, t_f) \rightarrow$ FTT solution tensor at time $t_f$ on rank-reducing coordinate system.

**Runtime:**
- $\Gamma = I,$
- $v_0 = u_{TT},$
- for $k = 0$ to $N_t$
  - $[v_k, \Gamma_{\text{new}}] = \text{gradient descent}(v_k, \Delta \epsilon, \eta, M_{\text{iter}})$
  - $\Gamma = \Gamma_{\text{new}} \Gamma$
  - $v_{k+1} = \mathcal{T}_\delta (v_k + \Delta t \Phi(v_k, G_{TT}, \Gamma), \Delta t)$
- end

scheme in coordinates $y = \Gamma x$

$$v_{TT}(y, t_{k+1}) = \mathcal{T}_\delta [v_{TT}(y, t_k) + \Delta t \Phi(G_{TT}, v_{TT}(y, t_k), \Delta t)].$$  \hspace{1cm} (89)

It is well-known that the computational cost of the scheme (85) (or (89)) scales linearly in the problem dimension $d$ and polynomially in the tensor rank of the solution and the operator. To determine an optimal coordinate transformation for reducing the overall cost of temporal integration it is necessary to have more precise estimates on the computational cost of one time step. Such computational cost depends on many factors, e.g., the increment function $\Phi$, the separation rank of the PDE operator $G_{\Gamma}$, the operator splitting (77) used, the FTT rank of the PDE solution $U(y, t_k)$ at time $t_k$, the rank of the operator applied to the solution after truncation $\mathcal{T}_\delta (G_{\Gamma}(U(y, t_k)))$, etc. From this observation it is clear that in order to obtain a optimal coordinate transformation for reducing the overall computational cost of temporal integration with step-truncation, we must take into consideration all these factors, in particular the solution rank and the operator rank. We emphasize that determining a coordinate transformation $\Gamma$ that controls the separation rank of a general nonlinear operator $G_{TT, \Gamma}$ is a non-trivial problem that we do not address in the present paper.

6.3. Coordinate-adaptive time integration

Next we develop coordinate-adaptive time integration schemes for PDEs on FTT manifolds that are designed to control the solution rank, the PDE operator rank, or the rank of the right hand side of the PDE.
Figure 7: Solution to the linear advection equation (90) with coefficients (92) at time $t = 30$. (a) Cartesian coordinates, (b) tensor ridge simulation 1 using Algorithm 3 with $\Delta \epsilon = 10^{-4}$, $M_{iter} = 2000$, and $\eta = 10^{-3}$, (c) tensor ridge simulation 2 using Algorithm 4 with $M_{iter} = 1$ and $\Delta \epsilon = 10^{-3}$. In all simulations the truncation tolerance is set to $10^{-6}$.

The first coordinate-adaptive algorithm (Algorithm 3) is designed to attempt a rank reducing coordinate transformation if the computational cost of time integration in the current coordinate system exceeds a predetermined threshold. The computational cost of one time step may be measured in different ways, e.g., by the CPU-time it takes to perform one time step, by the rank of the solution, or by the rank of the right hand side of the PDE (operator applied to the solution). In the second to last line of Algorithm 3 “time” denotes the computational time it takes to compute one time step and “rank” denotes either the solution rank, the rank of the PDE right hand side, or the maximum of the two.

The second algorithm (Algorithm 4) we propose for coordinate-adaptive tensor integration of PDEs is based on computing a small correction of the coordinate system at every time step. In practice, we compute one $\epsilon$-step of (45) at every time step during temporal integration of the given PDE. This yields a PDE in which the operator (which depends on the coordinate system) changes at every time step, i.e., a time-dependent operator induced by the time-dependent coordinate change.

Hereafter we apply these coordinate-adaptive algorithms to five different PDEs and compare the results with conventional FTT integrators in fixed Cartesian coordinates. All numerical simulations were run in Matlab 2022a on a 2021 MacBook Pro with M1 chip and 16GB RAM, spatial derivatives and integrals were approximated with one-dimensional Fourier pseudo-spectral differentiation matrices and quadrature rules [18], and various explicit step-truncation time integration schemes were used.

6.4. 2D linear advection equations

First we apply coordinate-adaptive tensor integration to the 2D linear advection equation

\[
\begin{align*}
\partial u(x, t) / \partial t & = f_1(x) \partial u(x, t) / \partial x_1 + f_2(x) \partial u(x, t) / \partial x_2, \\
\partial x_1, \partial x_2 & = u_0(x),
\end{align*}
\]

(90)

with two different sets of coefficients $f_i(x)$ specified hereafter. Each example is designed to demonstrate different features of the proposed coordinate-adaptive algorithms (Algorithm 3 and Algorithm 4).

In the first example, we generate the vector field $f(x) = (f_1(x), f_2(x))$ via the two-dimensional stream function [46]

\[
\psi(x_1, x_2) = \Theta(x_1)\Theta(x_2)
\]

(91)

with

\[
\Theta(x) = \frac{\cos(\alpha x / L)}{\cos(\alpha / 2)} - \frac{\cosh(\alpha x / L)}{\cosh(\alpha / 2)},
\]

(92)
L = 30 and \( \alpha = 4.73 \). Such a stream function generates the divergence-free vector field (see Figure 9(a))

\[
f_1(x) = \frac{\partial \psi}{\partial x_2}, \quad f_2(x) = -\frac{\partial \psi}{\partial x_1}.
\]

We set the initial condition

\[
u_0(x) = \frac{1}{m} \exp\left(-4(x_1 - 2)^2\right) \exp\left(-\frac{(x_2 - 2)^2}{2}\right),
\]

where

\[
m = \left\| \exp\left(-4(x_1 - 2)^2\right) \exp\left(-\frac{(x_2 - 2)^2}{2}\right) \right\|_{L^2(\Omega)}
\]
is a normalization constant.

We first ran one rank-adaptive tensor simulation in fixed Cartesian coordinates. We then ran two coordinate-adaptive simulations that use rank-reducing coordinate transformations during time integration. In the first coordinate-adaptive simulation we use Algorithm 3 with max rank = 15 and \( k_r = 0 \) to initialize coordinate transformations during time integration. For the Riemannian gradient descent algorithm that computes the rank reducing coordinate transformation we set step-size \( \Delta \epsilon = 10^{-4} \), maximum number of iterations \( M_{\text{iter}} = 2000 \) and stopping tolerance \( \eta = 10^{-1} \). In the second coordinate-adaptive simulation we use Algorithm 4 with \( M_{\text{iter}} = 1 \) and \( \Delta \epsilon = 10^{-3} \), i.e., the integrator performs one step of time integration followed by one step of the Riemannian gradient descent algorithm. In both coordinate-adaptive simulations we set the truncation threshold \( \delta = 10^{-6} \).

In Figure 7 we plot the solutions obtained from each of the three simulations at time \( t = 30 \). In order to check the accuracy of integrating the PDE solution in the low-rank coordinate system we mapped the transformed solution back to Cartesian coordinates using a two-dimensional trigonometric interpolant and compared with the solution computed in Cartesian coordinates. In both coordinate-adaptive simulations we found that the global \( L^\infty \) error is bounded by \( 8 \times 10^{-4} \), suggesting that the coordinate transformation does not affect accuracy significantly. In Figure 8 we plot the solution rank and the rank of the right hand side of the PDE (90) versus time for all three tensor simulations. We observe that in the coordinate-adaptive tensor ridge simulations the ranks of both the solution and the PDE right hand side are less than or equal to the corresponding ranks in Cartesian coordinates. We also observe that the adaptive simulation based on Algorithm 4 (denoted by “tensor ridge 2” in Figure 8) has significantly smaller rank than the adaptive simulation based on Algorithm 3 (denoted by “tensor ridge 1” in Figure 8).

Next, we demonstrate that linear coordinate transformations can be used to reduce the rank of a PDE operator and reduce the overall computational cost of temporal integration. To this end, consider again the two-dimensional advection equation (90) this time with advection coefficients

\[
f_1(x) = \exp\left(-a_1 (R_1 \cdot x)^2\right) \exp\left(-a_2 (R_2 \cdot x)^2\right),
\]

\[
f_2(x) = \exp\left(-b_1 (R_1 \cdot x)^2\right) \exp\left(-b_2 (R_2 \cdot x)^2\right),
\]

where \( R_i \) is the \( i \)th row of the matrix \( R \),

\[
R = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}.
\]

We set parameters \( \theta = \pi/4 \),

\[
a_1 = 1/20, \quad a_2 = 1/10, \quad b_1 = 1/10, \quad b_2 = 1/20,
\]

(97)
Figure 8: 2D Linear advection equation (90) with coefficients (93). Rank (number of singular values larger than $\delta = 10^{-6}$) of the PDE solution (a) and the right hand side the PDE (b). Tensor ridge 1 was computed using Algorithm 3 with a maximum solution rank threshold of 15. Tensor ridge 2 was computed using Algorithm 4, which performs coordinate corrections at each time step.

Figure 9: (a) Vector fields used as coefficients in the two-dimensional linear advection equation (90). The vector field defined in (93) is shown in (a) and the vector field defined in (95) is shown in (b).

and initial condition
\[ u_0(x) = \exp (-x_1^2/3) \exp (-x_2^2/3). \] (98)

In Figure 9(b) we plot the vector field $f(x) = (f_1(x), f_2(x))$ defined by (95). Note that the initial condition $u_0(x)$ is rank 1. The rank of the linear advection operator defined on the right hand side of (90) depends on the FTT truncation tolerance used to compress the multivariate functions $f_1(x)$ and $f_2(x)$. If we choose the coordinate transformation $y = \Gamma x$, where $\Gamma = R^{-1}$, then the initial condition remains rank 1, but the rank of the advection operator at the right hand side of (90) becomes 2, regardless of the FTT truncation tolerance used.

We ran two simulations of (90) with coefficients (95) using the step-truncation FTT integrator (49) based on Adams-Bashforth 3 with step-size $\Delta t = 10^{-3}$, truncation tolerance $\delta = 10^{-6}$, and final integration time $t = 5$. In the first simulation we solved the PDE with a step-truncation tensor method in fixed Cartesian coordinates. In the second simulation we solved the PDE in coordinates $y = R^{-1}x$, using the same step-truncation tensor method. In order to verify the accuracy of our FTT simulations we also computed a benchmark solution on a full tensor product grid in two dimensions. We then mapped the transformed solution back to Cartesian coordinates at each time step and compared it with the benchmark solution. We
Figure 10: Solution to the two-dimensional advection equation (90) with coefficients (95) at time $t = 5$. (a) FTT solution computed in Cartesian coordinates. (b) FTT-ridge computed low-rank coordinates.

found that the global $L^\infty$ error of both low-rank simulations is bounded by $8 \times 10^{-4}$. In Figure 10 we plot the FTT solution in Cartesian coordinates and the FTT-ridge solution in low-rank coordinates at time $t = 5$. We observe that the PDE operator expressed in Cartesian coordinates advects the solution at an angle relative to the underlying coordinate system while the operator expressed in coordinates $y = R^{-1}x$ advects the solution directly along a coordinate axis, hence the low rank dynamics. In Figure 12(a) we plot the solution ranks versus time. Note that even though the operator in Cartesian coordinates has significantly larger rank than the operator in low-rank coordinates, the solution ranks follow the same trend during temporal integration with the FTT-ridge rank only slightly smaller than the FTT rank.

**Computational cost.** The CPU-time of integrating the advection equation (90) with coefficients (95) from $t = 0$ to $t = 5$ is 72 seconds when computed with FTT in Cartesian coordinates and 31 seconds when computed with FTT-ridge in low-rank coordinates. Note that although the ranks of the low-rank solutions are similar at each time step (Figure 12(a)), the computational speed-up of the FTT-ridge simulation is due to the operator rank, which is 2 for FTT-ridge and 16 for FTT in Cartesian coordinates.

6.5. Allen-Cahn equation

Next we demonstrate coordinate-adaptive tensor integration on a simple nonlinear PDE. The Allen-Cahn equation is a reaction-diffusion PDE, which, in its simplest form includes a low-order polynomial non-linearity (reaction term) and a diffusion term

\[
\begin{align*}
\frac{\partial u(x, t)}{\partial t} &= \alpha \Delta u(x, t) + u(x, t) - u(x, t)^3, \\
\quad u(x, 0) &= u_0(x).
\end{align*}
\]

(99)

In two spatial dimensions the Laplacian in coordinates $y = \Gamma x$ is given by

\[
\Delta \mathbf{r} = (\Gamma_{11}^2 + \Gamma_{12}^2) \frac{\partial^2}{\partial y_1^2} + (\Gamma_{21}^2 + \Gamma_{22}^2) \frac{\partial^2}{\partial y_2^2} + 2(\Gamma_{11} \Gamma_{21} + \Gamma_{12} \Gamma_{22}) \frac{\partial^2}{\partial y_1 \partial y_2}
\]

(100)

which allows us to write the nonlinear PDE (99) in the coordinate system $y = \Gamma x$ with only a small increase in the rank of the Laplacian operator. The FTT rank of the cubic term appearing in the PDE operator of the
The Allen-Cahn equation (99) is determined by the rank of the FTT solution. Standard algorithms for multiplying two FTT tensors $u_{TT}$ and $v_{TT}$ with ranks $r_1$ and $r_2$ results in a FTT tensor with (non-optimal) rank equal to the Hadamard product of the two ranks $r_1 \circ r_2$. Hence, by reducing the solution rank with a coordinate transformation, we can reduce the computational cost of computing the nonlinear term in (99). We set the diffusion coefficient $\alpha = 0.2$ and the initial condition $u_0(x)$ as the rotated Gaussian from equation (25) with $\epsilon = \pi/3$.

We ran two FTT simulations of (99) for time $t \in [0, 5]$. The first simulation was computed in fixed Cartesian coordinates. In the second simulation we used the coordinate transformation $\Phi(\pi/3)^{-1}$ (which in this case we have available analytically) to transform the initial condition into a rank 1 FTT-ridge function. We integrated the rank 1 FTT-ridge initial condition forward in time using the corresponding transformed PDE, i.e., using the transformed Laplacian (100). In order to verify the accuracy of our low-rank simulations we also computed a benchmark solution on a full tensor product grid in two dimensions and mapped the transformed solution back to Cartesian coordinates at each time step. We found that the global $L^\infty$ error of both low-rank solutions is bounded $5 \times 10^{-5}$. In Figure 11 we plot the FTT solution in Cartesian coordinates and the FTT-ridge solution in low-rank coordinates at time $t = 5$. We observe that the profile of Gaussian functions are preserved as the solution moves from the unstable equilibrium at $u = 0$ to the stable equilibrium at $u = 1$. In Figure 12(b) we plot the solution ranks versus time. We observe that the FTT solution rank is larger than the FTT-ridge solution rank at each time.

**Computational cost.** The CPU-time of integrating the Allen-Cahn equation (99) from $t = 0$ to $t = 5$ is 279 seconds when computed using FTT in Cartesian coordinates and 72 seconds when computed using FTT-ridge in low-rank coordinates. The optimal coordinate transformation at time $t = 0$ is known analytically so we do not need to compute it, thus these computational timings only include the temporal integration, and do not account for any computational time of changing the coordinate system. A significant amount of computational time in computing the FTT solutions comes from computing the cubic nonlinearity appearing in the Allen-Cahn equation at each time step. The lower rank FTT-ridge solution allows for this term to be computed significantly faster than the FTT solution in Cartesian coordinates.
6.6. 3D and 5D linear advection equations

We also applied the rank-reducing coordinate-adaptive FTT integrators to the advection equation

\[
\begin{aligned}
\frac{\partial u(x,t)}{\partial t} &= f(x) \cdot \nabla u(x,t), \\
u(x,0) &= u_0(x),
\end{aligned}
\]  

(101)

in dimensions three and five with initial condition \(u_0(x)\) defined as a Gaussian mixture

\[
u_0(x) = \frac{1}{m} \sum_{i=1}^{N_\theta} \exp\left(-\sum_{j=1}^d \frac{1}{\beta_{ij}} \left(R^{(i)}_j \cdot x + t_{ij}\right)^2\right),
\]  

(102)

where \(R^{(i)}_j\) is the \(j\)-th row of a \(d \times d\) rotation matrix \(R^{(i)}\), \(\beta_i \geq 0\), \(t_{ij}\) are translations and \(m\) is the normalization factor

\[
m = \left\| \sum_{i=1}^{N_\theta} \exp\left(-\sum_{j=1}^d \frac{1}{\beta_{ij}} \left(R^{(i)}_j \cdot x + t_{ij}\right)^2\right) \right\|_{L^1(\mathbb{R}^d)}.
\]

6.6.1. Three-dimensional simulation results

First, we consider three spatial dimensions \((d = 3)\), and set the coefficients in (101) as

\[
f(x) = -\frac{1}{6} \begin{bmatrix} 2 \sin(x_2) \\ 3 \cos(x_3) \\ 3x_1 \end{bmatrix},
\]  

(103)

resulting in the linear operator
Figure 13: Volumetric plot of the FTT solutions to the 3D advection equation (101) at time $t = 0$ (left column) in Cartesian coordinates (top) and low-rank coordinates (bottom), and at time $t = 1$ (right column) in Cartesian coordinates (top) and low-rank coordinates (bottom).

$$f(x) \cdot \nabla = \frac{\sin(x_2)}{3} \frac{\partial}{\partial x_1} + \frac{\cos(x_3)}{2} \frac{\partial}{\partial x_2} + \frac{x_1}{2} \frac{\partial}{\partial x_3}. \quad (104)$$

In a previous work [12] we have demonstrated that variable coefficient advection problems with operators of the form (104) can have solutions with multilinear rank that grows significantly over time. We set the parameters in the initial condition (102): $N_q = 1$,

$$R^{(1)} = \exp \left( \frac{1}{28} \begin{bmatrix} 0 & 7\pi & 4\pi \\ -7\pi & 0 & 7\pi \\ -4\pi & -7\pi & 0 \end{bmatrix} \right), \quad \beta = \begin{bmatrix} 3 & 1/10 & 3 \end{bmatrix}, \quad (105)$$

and $t_{ij} = 0$ for all $i, j$.

We ran three FTT simulations up to time $t = 1$. The first simulation is computed in fixed Cartesian coordinates. We then tested the FTT integrator with rank-reducing coordinate transformation in two different simulation settings. In the first one, we performed a coordinate transformation only at time $t = 0$. Such a coordinate transformation is not done using the Riemannian gradient descent algorithm, since, in this case we have the optimal coordinate transformation available analytically and we can simply evaluate the FTT tensor on the low rank coordinates. In the second simulation we also performed a coordinate transformation at time $t = 0$ (once again the coordinate transformation at time $t = 0$ is not done using the Riemannian
gradient descent algorithm) and then used the coordinate-adaptive integration Algorithm 3 with max rank = 15 and \( k_r = 5 \). With these parameters the coordinate-adaptive algorithm triggers three additional coordinate transformations at times \( t \in \{0.25, 0.59, 0.9\} \) that are computed using the Riemannian gradient descent algorithm \( \mathcal{G} \) with step size \( \Delta \epsilon = 10^{-4} \) and stopping tolerance \( \eta = 10^{-1} \). In Figure 15(c) we plot the absolute value of the derivative of the cost function \( C(\mathbf{f}^T(\epsilon)) \) versus \( \epsilon \) for the instances of gradient descent at times \( t > 0 \). We observe that the rate of change of the cost function becomes smaller as we iterate the gradient descent routine, i.e., the cost function is decreasing less per iteration after several iterations. This indicates that the cost function is approaching a flatter region and our gradient descent method is becoming less effective for reducing the cost function. In Figure 14(a) we plot the 1-norm of the FTT solution rank vector versus time and in Figure 14(b) we plot the 1-norm of the FTT solution velocity (i.e., the PDE right hand side) for each FTT simulation. We observe that the FTT-ridge solutions and right hand side of the PDE have rank that is smaller than the corresponding ranks of the FTT solution in Cartesian coordinates. We also observe that the adaptive coordinate transformations performed at times \( t > 0 \) do not reduce the solution rank at the time of application, but they do slow the rank increase as time integration proceeds. In Figure 15(a)-(b) we plot the singular values of the FTT solutions at final time and note that both FTT-ridge solutions have singular values that decay significantly faster than the FTT solution in Cartesian coordinates. Moreover, the additional coordinate transformations performed by the coordinate-adaptive integrator causes the singular values of the FTT-ridge solution to decay faster than the other FTT-ridge solution that used only one coordinate transformation at \( t = 0 \). In Figure 13 we provide volumetric plots of the PDE solution in Cartesian coordinates and in the transformed coordinate system computed with the coordinate-adaptive algorithm 3 at time \( t = 1 \). We observe that the reduced rank tensor ridge solution appears to be more symmetrical with respect to the underlying coordinate axes than the solution in Cartesian coordinates.

It is important to note that the operator \( G(\mathbf{f}^T, \mathbf{a}) = f(\mathbf{a}) \cdot \nabla \) in (104) is a separable operator of rank \( g = [1, 3, 3, 1] \). For a general linear coordinate transformation \( \mathbf{f} \) the operator \( G_\mathbf{f} \) (see (82)) acting in the transformed coordinate system can be obtained using trigonometric identities with (non-optimal) separation ranks. A more efficient representation of \( G_\mathbf{f} \) can be obtained by FTT compression and then splitting \( G_\mathbf{f} \) into a sum of three operators

\[
G_\mathbf{f} = G_\mathbf{f}^{(1)} + G_\mathbf{f}^{(2)} + G_\mathbf{f}^{(3)},
\]

where \( G_\mathbf{f}^{(i)} \) have ranks \( g_\mathbf{f}^{(i)} = [1, 4, 3, 1] \) for each \( i = 1, 2, 3 \). Then we apply the operator \( G_\mathbf{f} \) to the FTT-ridge solution at each time using the procedure summarized in (78). In this case, since the PDE operator \( G \) in Cartesian coordinates is separable and low-rank, it is not surprising that a linear coordinate transformation

\[
\begin{align*}
\text{(a)} & \quad \text{Cartesian} \quad \text{1 Transformation} \quad \text{4 Transformations} \\
\text{(b)} & \quad \text{Cartesian} \quad \text{1 Transformation} \quad \text{4 Transformations} \\
\text{(c)} & \quad L^\infty \text{error of the FTT solutions relative to a benchmark solution.}
\end{align*}
\]

Figure 14: 3D advection equation (101): Multilinear rank of the PDE solution (a) and PDE right-hand-side (b) in Cartesian coordinates and transformed coordinates versus time. In (c) we plot the \( L^\infty \) error of the FTT solutions relative to a benchmark solution.
increases the PDE operator rank. However by splitting the operator such as (106) and performing a FTT truncation operation after applying each lower rank operator \( G_i \), we can mitigate the computational cost.

In Figure 14(c) we plot the \( L^\infty \) error between the transformed solutions and the FTT solution in Cartesian coordinates relative to the benchmark solution. We observe that the \( L^\infty \) error of our coordinate-adaptive solution is very close to the \( L^\infty \) error of the solution computed in Cartesian coordinates. This implies that the error incurred by transforming coordinates, integrating the PDE in the new coordinate system, and then transforming coordinates back is not significant.

Computational cost. The CPU-time of integrating the three-dimensional advection equation (101) from \( t = 0 \) to \( t = 1 \) is 413 seconds when computed using FTT in Cartesian coordinates, 173 seconds when computed using FTT-ridge in low-rank coordinates with one coordinate transformation at \( t = 0 \), and 299 seconds when computed using the coordinate-adaptive FTT Algorithm 3. These timings do not include the coordinate transformations at time \( t = 0 \) since they were not computed using Riemannian gradient descent. The timings do include the computation of the new coordinate systems at times \( t > 0 \).

6.6.2. Five-dimensional simulation results

Finally, we consider the advection equation (101) in dimension five \((d = 5)\) with coefficients

\[
f(x) = \begin{bmatrix} -x_2 \\ x_3 \\ x_5 \\ -x_2 \\ -x_3 \end{bmatrix},
\] (107)

This allows us to test our coordinate-adaptive algorithm for a case in which we know the optimal ridge matrix. In the initial condition (102) we set the following parameters: \( N_g = 2 \),

\[
R^{(1)} = R^{(2)} = I_{5 \times 5},
\]

\[
\beta = \begin{bmatrix} 1/2 & 2 & 1/2 & 3 & 1/2 \\ 1/3 & 2 & 1 & 1/2 \end{bmatrix},
\] (108)

and

\[
t = \begin{bmatrix} 1 & 1 & 1 & -1 & 1 \\ 0 & 3/2 & -1/2 & 1/2 \end{bmatrix},
\] (109)
which results in an initial condition with FTT rank $[1\ 2\ 2\ 2\ 2\ 1]$. Due to the choice of coefficients (107), the analytical solution to the PDE (101) can be written as a ridge function in terms of the PDE initial condition

$$u(x, t) = u_0(e^{tB}x),$$

where

$$B = \begin{bmatrix}
0 & -1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 \\
0 & -1 & 0 & 0 & 0 \\
0 & 0 & -1 & 0 & 0
\end{bmatrix}. \tag{111}$$

Thus (110) is a tensor ridge solution to the 5D advection equation (101) with the same rank as the initial condition, i.e., in this case there exists a tensor ridge solution at each time with FTT rank equal to $[1\ 2\ 2\ 2\ 2\ 1]$.

We ran two simulations of the PDE (101) up to $t = 1$. The first simulation is computed with a step-truncation method in fixed Cartesian coordinates. The second simulation is computed with the coordinate-adaptive FTT-ridge tensor method that performs coordinate corrections at each time step (Algorithm 4) with
\( \Delta \epsilon = 5 \times 10^{-4} \) and \( M_{\text{iter}} = 1 \). In Figure 16 we plot marginal PDFs of the FTT solution and the FTT-ridge solution at initial time \( t = 0 \) and final time \( t = 1 \). We observe that the reduced rank FTT-ridge solution appears to be more symmetrical with respect to the coordinate axes than the corresponding function on Cartesian coordinates. In Figure 18(a) we plot the 1-norm of the FTT solution rank versus time and in Figure 18(b) we plot the FTT rank of the PDE velocity vector (right hand side of the PDE) versus time for both FTT solutions. We observe that the FTT solution rank in Cartesian coordinates grows quickly compared to the FTT-ridge solution in Cartesian coordinates. This is expected due to the existence of a low-rank FTT-ridge solution \( (110) \). Note that the coordinate-adaptive algorithm \( \text{(4)} \) produces a FTT-ridge solution with ridge matrix that is different than \( e^B \) in \( (110) \). The reason can be traced back to the cost function we are minimizing, i.e., the Schauder norm (see section 4.1), and the fact that we do not fully determine the minimizer at each step, but rather perform only one \( \epsilon \)-step in the direction of the Riemannian gradient. Although the rank of the FTT-ridge solution computed with algorithm \( \text{(4)} \) is larger than the rank of the analytical solution \( (110) \), the algorithm still controls the FTT solution rank during time integration. In Figure 17 we plot the multilinear spectra of the two FTT solutions at time \( t = 1 \). We observe that the multilinear spectra of the FTT-ridge function decay significantly faster than the spectra of the FTT solution in Cartesian coordinates.

For this problem it is not straightforward to compute a benchmark solution on a full tensor product
grid. If we were to use the same resolution as the FTT solutions, i.e., 200 points in each dimension, then each time snapshot of the benchmark solution would be an array containing \(200^5 \approx 3.2 \times 10^{11}\) double precision floating point numbers. This requires 2.56 terabytes of memory storage per time snapshot. In lieu of comparing our FTT solutions with a benchmark solution, we compared the two FTT solutions with each other. To do so we mapped the FTT-ridge solution back to Cartesian coordinates every 250 time steps by solving a PDE of the form (20) numerically. We compared the \((x4, x5)\)-marginal PDFs of the two solutions and found that the global \(L^\infty\) norm of the difference of the two solution PDFs is bounded by \(6 \times 10^{-4}\).

**Computational cost.** The CPU-time of integrating the five-dimensional advection equation from \(t = 0\) to \(t = 1\) is 2046 seconds when computed using FTT in Cartesian coordinates and 2035 seconds when computed using FTT-ridge in low-rank coordinates with coordinate corrections at each time step.

**Acknowledgements**

This research was supported by the U.S. Air Force Office of Scientific Research grant FA9550-20-1-0174, and by the U.S. Army Research Office grant W911NF1810309.

**Appendix A. The Riemannian manifold of coordinate transformations**

We endow the search space \(\text{SL}_d(\mathbb{R})\) in (38) with a Riemannian manifold structure. To this end, we first notice that \(\text{SL}_d(\mathbb{R})\) is a matrix Lie group over \(\mathbb{R}\) and in particular is a smooth manifold. A point \(A \in \text{SL}_d(\mathbb{R})\) corresponds to a linear coordinate transformation of \(\mathbb{R}^d\) with determinant equal to 1. A smooth path \(\Theta(\epsilon)\) on the manifold \(\text{SL}_d(\mathbb{R})\) paramaterized by \(\epsilon \in (-\delta, \delta)\) is a collection of smoothly varying linear coordinate transformations with determinant equal to 1 for all \(\epsilon \in (-\delta, \delta)\). Denote by \(C^1((-\delta, \delta), \text{SL}_d(\mathbb{R}))\) the collection of all continuously differentiable paths \(\Theta(\epsilon)\) on the manifold \(\text{SL}_d(\mathbb{R})\) parameterized by \(\epsilon \in (-\delta, \delta)\). The tangent space of \(\text{SL}_d(\mathbb{R})\) at the point \(A \in \text{GL}_d(\mathbb{R})\) is defined to be the collection of equivalence classes of velocities associated to all possible curves on \(\text{SL}_d(\mathbb{R})\) passing through the point \(A\)

\[
T_A \text{SL}_d(\mathbb{R}) = \left\{ \frac{d\Theta(\epsilon)}{d\epsilon} \bigg|_{\epsilon=0} : \Theta \in C^1((-\delta, \delta), \text{SL}_d(\mathbb{R})), \Theta(0) = A \right\}.
\]  

(A.1)
It is well-known (e.g., [40]) that the tangent space of $\text{SL}_d(\mathbb{R})$ at the point $A$ is given by
\[
T_A\text{SL}_d(\mathbb{R}) = \text{sl}_d(\mathbb{R})A = \{NA : N \in \text{sl}_d(\mathbb{R})\},
\] (A.2)
where $\text{sl}_d(\mathbb{R})$ denotes the collection of all $d \times d$ real matrices with vanishing trace. We can easily verify that if $\Theta(\epsilon)$ is a smooth collection of matrices parameterized by $\epsilon \in (-\delta, \delta)$ with $\Theta(0) \in \text{SL}_d(\mathbb{R}) = \text{sl}_d(\mathbb{R})$ and $d\Theta(\epsilon)/d\epsilon \in T_{\Theta(\epsilon)}\text{SL}_d(\mathbb{R})$ for all $\epsilon$, then $\det(\Theta(\epsilon)) = 1$ for all $\epsilon$, i.e., $\Theta(\epsilon) \in \text{SL}_d(\mathbb{R})$ for all $\epsilon$. The proof of this result is a direct application of Jacobi’s formula [30] which states
\[
\frac{d}{d\epsilon}\det(\Theta(\epsilon)) = \det(\Theta(\epsilon))\text{trace}\left(\frac{d\Theta(\epsilon)}{d\epsilon}\Theta^{-1}(\epsilon)\right)
= \det(\Theta(\epsilon))\text{trace}(N\Theta(\epsilon)\Theta^{-1}(\epsilon))
= \text{det}(\Theta(\epsilon))\text{trace}(N)
= 0,
\] (A.3)
since $N \in \text{sl}_d(\mathbb{R})$. Thus the determinant of $\Theta(\epsilon)$ is constant and since $\det(\Theta(0)) = 1$ it follows that $\det(\Theta(\epsilon)) = 1$ for all $\epsilon \in (-\delta, \delta)$. In the language of abstract differential equations, $\text{sl}_d(\mathbb{R})$ is referred to as the Lie algebra associated with the Lie group $\text{SL}_d(\mathbb{R})$. In Figure A.19 a we provide an illustration of a path $\Theta(\epsilon)$ on the manifold $\text{SL}_d(\mathbb{R})$ passing through the point $A$ and the tangent space $T_A\text{SL}_d(\mathbb{R})$ of $\text{SL}_d(\mathbb{R})$ at $A$. Also depicted in Figure A.19 is a smooth function $f$ from $\text{SL}_d(\mathbb{R})$ to another smooth manifold $M$. The image of the path $\Theta(\epsilon)$ on $\text{SL}_d(\mathbb{R})$ under $f$ is a path $f(\Theta(\epsilon))$ on $M$. Under this mapping of curves, we can associate the tangent vector $d\Theta(\epsilon)/d\epsilon|_{\epsilon=0}$ in $T_A\text{SL}_d(\mathbb{R})$ with a tangent vector $df(\Theta(\epsilon))/d\epsilon|_{\epsilon=0}$ in $T_{f(A)}M$. This association gives rise to the notion of the directional derivative of a function $f : \text{SL}_d(\mathbb{R}) \to M$ which we now define.

**Definition A1.** Let $f$ be a smooth function from the Riemannian manifold $\text{SL}_d(\mathbb{R})$ to a smooth manifold $M$. The directional derivative of $f$ at the point $A \in \text{SL}_d(\mathbb{R})$ in the direction $V \in T_A\text{SL}_d(\mathbb{R})$ is defined as
\[
(d_Af)V = \left.\frac{df(\Theta(\epsilon))}{d\epsilon}\right|_{\epsilon=0},
\] (A.4)
where $\Theta(\epsilon)$ is a smooth curve on $\text{SL}_d(\mathbb{R})$ passing through the point $A$ at $\epsilon = 0$ with velocity $V$.

It is a standard exercise of differential geometry to verify that the directional derivative (A.4) is independent of the choice of curve $\Theta(\epsilon)$. The map $d_Af$ appearing in (A.4) is a linear map from $T_A\text{SL}_d(\mathbb{R})$ to $T_{f(A)}M$ known as the differential of $f$. In Figure A.19 we provide an illustration of the mapping $f$ and its differential. The differential and directional derivative allow us to understand the change in $f$ when moving from the point $A \in \text{SL}_d(\mathbb{R})$ in the direction of the tangent vector $V \in T_A\text{SL}_d(\mathbb{R})$. Next, we compute the Riemannian gradient of $f$, i.e., a specific tangent vector on $\text{SL}_d(\mathbb{R})$ that points in a direction which makes the function $f$ vary the most.

For functions defined on Euclidean space, the connection between directional derivative and gradient is understood by using the standard inner product defined for Euclidean spaces. A generalization of the Euclidean inner product for an abstract manifold such as $\text{SL}_d(\mathbb{R})$ is the Riemannian metric $(\cdot, \cdot)_A$, which is a collection of smoothly varying inner products on each tangent space $T_A\text{SL}_d(\mathbb{R})$. In particular, we define
\[
(V, W)_A = \text{trace}\left((VA^{-1})(WA^{-1})^\top\right), \quad \forall A \in \text{SL}_d(\mathbb{R}), \; \forall V, W \in T_A\text{SL}_d(\mathbb{R})
\] (A.5)
on $\text{SL}_d(\mathbb{R})$. With this Riemannian metric, we can define the Riemannian gradient.

**Definition A2.** Let $f$ be a smooth function from $\text{SL}_d(\mathbb{R})$ to a smooth manifold $M$. The Riemannian gradient of $f$ at the point $A \in \text{SL}_d(\mathbb{R})$ is the unique vector field $\text{grad } f(A)$ satisfying
\[
(d_Af)V = (\text{grad } f(A), V)_A, \quad \forall V \in T_A\text{SL}_d(\mathbb{R}).
\] (A.6)
Analogous to the Euclidean case, the Riemannian gradient points in the direction which $f$ increases the most, and, the negative gradient points the the direction which $f$ decreases most. With this Riemannian geometric structure, we proceed by constructing a path on $\text{SL}_d(\mathbb{R})$, known as a descent path, which converges to a local minimum of the cost function $S \circ E$ in (38).

**Appendix B. Theorems and Proofs**

First, we show that the tensor rank is invariant under translation of functions with compact support. This allows us to disregard translations when looking for rank reducing coordinate flows.

**Proposition B1.** Let $u_{TT} \in L^2_\mu(\Omega)$ ($\Omega \subseteq \mathbb{R}^d$) be a rank-$r$ FTT with $\text{supp}(u_{TT}) \subseteq \Omega$ and $C_t : \mathbb{R}^d \rightarrow \mathbb{R}^d$ a coordinate translation, i.e.,

$$C_t(x) = x + t,$$

where $t \in \mathbb{R}^d$, such that $\text{supp}(u_{TT}) \subseteq C_t(\Omega)$. Then $u_{TT}(C_t(x))$ is also a rank-$r$ FTT tensor.

**Proof.** Let

$$u_{TT}(x) = Q_{\leq i}(x_{\leq i}) \Sigma_i Q_{>i}(x_{>i})$$

be an orthogonalized expansion of the the rank-$r$ FTT $u_{TT}$ as in (28). Then by a simple change of variables in the integrals it is easy to verify that the translated cores $Q^T_{\leq i}(x_{\leq i} + t_{\leq i}), Q_{>i}(x_{>i} + t_{>i})$ also satisfy the orthogonality conditions

$$\langle Q^T_{\leq i}(x_{\leq i} + t_{\leq i})Q_{\leq i}(x_{\leq i} + t_{\leq i}) \rangle_{\leq i} = I_{r_i \times r_i},$$

$$\langle Q_{>i}(x_{>i} + t_{>i})Q^T_{>i}(x_{>i} + t_{>i}) \rangle_{>i} = I_{r_i \times r_i},$$

and thus

$$u_{TT}(C_t(x)) = Q_{\leq i}(x_{\leq i} + t_{\leq i}) \Sigma_i Q_{>i}(x_{>i} + t_{>i})$$

is an orthogonalized FTT tensor. Hence, $u_{TT} \circ C_t$ has the same multilinear rank $r$ as $u_{TT}$.

\[\square\]
Next, we provide a proof of Proposition 4.1. To do so we first provide the differentials of the maps $E$, $S$ and $C = S \circ E$ in the following lemmas.

**Lemma B1.** The differential of the evaluation map $E$ corresponding to $u_{tr}$ (see eqn. (36)) at the point $A \in \text{GL}_d(\mathbb{R})$ in the direction $V \in T_A \text{GL}_d(\mathbb{R})$ is

$$(d_A E) V = \nabla v(x) \cdot (V x).$$

**Proof.** This result is proven directly from the definitions. Let $\Theta(\epsilon)$ be a smooth curve on $\text{GL}_d(\mathbb{R})$ passing through $A$ with velocity $V$ at $\epsilon = 0$. Then

$$(d_A E) V = \frac{\partial}{\partial \epsilon} E(\Theta(\epsilon)) \bigg|_{\epsilon=0} = \frac{\partial}{\partial \epsilon} u_{tr}(\Theta(\epsilon)x) \bigg|_{\epsilon=0} = \nabla u_{tr}(Ax) \cdot (V x) = \nabla v(x) \cdot (V x).$$

$\square$

**Lemma B2.** The differential of $S$ (see (34)) at the point $v \in L^2_\mu(\Omega)$ in the direction $w \in T_v L^2_\mu(\Omega)$ is

$$(d_v S) w = \sum_{i=1}^{d-1} \int_\Omega Q_{\leq i} Q_{> i} w(x) d\mu(x)$$

where $Q_{\leq i}, Q_{> i}$ are FTT cores of $v$ as in eqn. (28) satisfying the orthogonality conditions (31).

**Proof.** Let $\gamma(\epsilon)$ be a smooth curve on $L^2_\mu(\Omega)$ passing through $v$ at $\epsilon = 0$ with velocity $w$. At each $\epsilon$ the function $\gamma(\epsilon)$ admits orthogonalizations of the form

$$\gamma(\epsilon) = Q_{\leq i}(\epsilon) \Sigma_i(\epsilon) Q_{> i}(\epsilon)$$

for each $i = 1, 2, \ldots, d-1$. Differentiating (B.8) with respect to $\epsilon$ we obtain

$$\frac{\partial \gamma}{\partial \epsilon} = \frac{\partial Q_{\leq i}}{\partial \epsilon} \Sigma_i Q_{> i} + Q_{\leq i} \frac{\partial \Sigma_i}{\partial \epsilon} Q_{> i} + Q_{\leq i} \Sigma_i \frac{\partial Q_{> i}}{\partial \epsilon}. $$

(B.9)

Multiplying on the left by $Q^T_{\leq i}$ and on the right by $Q^T_{> i}$, applying the operator $\langle \cdot \rangle_{\leq i} \leq i$, and evaluating at $\epsilon = 0$ we obtain

$$\frac{\partial \Sigma_i}{\partial \epsilon} = \langle Q^T_{\leq i} w(x) Q^T_{> i} \rangle_{\leq i} - \langle Q^T_{\leq i} \frac{\partial Q_{\leq i}}{\partial \epsilon} \rangle_{\leq i} \Sigma_i - \Sigma_i \langle \frac{\partial Q_{> i}}{\partial \epsilon} Q^T_{> i} \rangle_{> i},$$

(B.10)

where we used orthogonality of the FTT cores $Q_{\leq i}$ and $Q_{> i}$. Differentiating the orthogonality constraints (31) with respect to $\epsilon$ we obtain

$$\langle \frac{\partial Q^T_{\leq i}}{\partial \epsilon} Q_{\leq i} \rangle_{\leq i} = - \langle Q^T_{\leq i} \frac{\partial Q_{\leq i}}{\partial \epsilon} \rangle_{\leq i}, \quad \langle \frac{\partial Q_{> i}}{\partial \epsilon} Q^T_{> i} \rangle_{> i} = - \langle Q_{> i} \frac{\partial Q^T_{> i}}{\partial \epsilon} \rangle_{> i}, \quad \forall \epsilon, $$

(B.11)

which implies that the second two terms on the right hand side of (B.10) side are skew-symmetric and thus have zeros on the diagonal. Hence the diagonal entries of $\frac{\partial \Sigma_i}{\partial \epsilon}$ are the diagonal entries of the matrix $\langle Q^T_{\leq i} w(x) Q^T_{> i} \rangle_{\leq i} \leq i$, or written element-wise

$$\frac{\partial S_i(\alpha_i)}{\partial \epsilon} = \int q_{\leq i}(\alpha_i)w(x)q_{> i}(\alpha_i) d\mu(x).$$

(B.12)

36
Finally summing (B.12) over \( i = 1, 2, \ldots, d - 1 \) and \( \alpha_i = 1, 2, \ldots, r_i \) and using matrix product notation for the latter summation we obtain

\[
\sum_{i=1}^{d-1} \sum_{\alpha_i=1}^{r_i} \frac{\partial S_i(\alpha_i)}{\partial \epsilon} = \int_\Omega Q_{\leq i} Q_{>i} w(x) d\mu(x), \tag{B.13}
\]

proving the result.

Combining the results of Lemma B1 and Lemma B2 with a simple application of the chain rule for differentials we prove the following Lemma.

**Lemma B3.** The differential of the function \( C = S \circ E \) at the point \( A \in \text{GL}_d(\mathbb{R}) \) in the direction \( V \in T_A \text{GL}_d(\mathbb{R}) \) is

\[
d_A(S \circ E)V = \sum_{i=1}^{d-1} \int_\Omega Q_{\leq i} Q_{>i} \nabla v(x) \cdot (V x) \, d\mu(x), \tag{B.14}
\]

where \( Q_{\leq i}, Q_{>i} \) are orthogonal FTT cores of \( v(x) \).

Next we provide the Riemannian gradient of \( C = S \circ E \) when its domain is \( \text{GL}_d(\mathbb{R}) \).

**Proposition B2.** The Riemannian gradient of \( (S \circ E) : \text{GL}_d(\mathbb{R}) \rightarrow \mathbb{R} \) at the point \( A \in \text{GL}_d(\mathbb{R}) \) is given by

\[
\text{grad}(S \circ E)(A) = \hat{D}A, \tag{B.15}
\]

where

\[
\hat{D} = \sum_{i=1}^{d-1} \int_\Omega Q_{\leq i} Q_{>i} \nabla v(x) (A x)^T d\mu(x). \tag{B.16}
\]

**Proof.** To prove this result we check directly using the definition of Riemannian gradient. For any \( A \in \text{GL}_d(\mathbb{R}) \) and \( V \in T_A \text{GL}_d(\mathbb{R}) \) we have

\[
\left( \hat{D}A, V \right)_A = \left( \left[ \sum_{i=1}^{d-1} \int_\Omega Q_{\leq i} Q_{>i} \nabla v(x) (A x)^T d\mu(x) \right] A, V \right)_A
\]

\[
= \text{trace} \left( \sum_{i=1}^{d-1} \int_\Omega Q_{\leq i} Q_{>i} \nabla v(x) x^T A^T d\mu(x) A^{-T} V^T \right)
\]

\[
= \sum_{i=1}^{d-1} \int_\Omega Q_{\leq i} Q_{>i} \text{trace} (\nabla v(x) x^T V^T) d\mu(x)
\]

\[
= \sum_{i=1}^{d-1} \int_\Omega Q_{\leq i} Q_{>i} \nabla v(x) \cdot (V x) d\mu(x),
\]

where in the last equality we used the fact that \( \text{trace}(vw^T) = v \cdot w \) for any column vectors \( v, w \in \mathbb{R}^d \).
In general, the trace of $\hat{D}$ is not equal to zero and thus $\hat{DA}$ is not an element of the tangent space $T_A SL_d(\mathbb{R})$ (see eqn. (A.2)). In order to obtain the Riemannian gradient $DA$ of $S \circ E : SL_d(\mathbb{R}) \to \mathbb{R}$, we modify the diagonal entries of $\hat{D}$ to ensure that $DA$ satisfies the properties of Riemannian gradient and also belongs to the tangent space $T_A SL_d(\mathbb{R})$.

**Proof:** First we prove that $DA$ with $D$ defined in (41) is an element of $T_A SL_d(\mathbb{R})$, i.e., we prove that $\text{trace}(D) = 0$

$$\text{trace}(D) = \sum_{i=1}^{d-1} \int Q_{\leq i} Q_{> i} \left[ \text{trace} \left( \nabla v(x)(Ax)^T - \frac{\nabla v(x)^T Ax}{d} I_{d \times d} \right) \right] d\mu(x)$$

It is easy to verify that $\text{trace} \left( \nabla v(x)(Ax)^T \right) = \text{trace} \left( \frac{\nabla v(x)^T Ax}{d} I_{d \times d} \right)$ and hence $\text{trace}(D) = 0$. Next we show that $(DA, V)_A = DA(S \circ E)V$ for all $A \in SL_d(\mathbb{R})$ and $V \in T_A SL_d(\mathbb{R})$. Indeed, for any $A \in SL_d(\mathbb{R})$ and $V \in T_A SL_d(\mathbb{R})$ we have

$$(DA, V)_A = \sum_{i=1}^{d-1} \int Q_{\leq i} Q_{> i} \left[ \text{trace} \left( \nabla v(x)(Ax)^T - \frac{\nabla v(x)^T Ax}{d} I_{d \times d} \right) (VA^{-1})^T \right] d\mu(x)$$

Since $V \in T_A SL_d(\mathbb{R})$ we have that $V = WA$ for some real matrix $W$ with $\text{trace}(W) = 0$. Using this in the preceding equation we have

$$(DA, A, V)_A = \sum_{i=1}^{d-1} \int Q_{\leq i} Q_{> i} \left[ \text{trace} \left( \nabla v(x)x^TA^TVA^T - \frac{\nabla v(x)^T Ax}{d} \text{trace}(W^T) \right) \right] d\mu(x)$$

completing the proof.

**Lemma B4.** Let $\sigma_i(\alpha_i) \ (i = 1, 2, \ldots, d, \alpha_i = 1, 2, \ldots, r_i)$ be the multilinear spectrum of the FTT $v_{\sigma}(x) \approx u_{\sigma}(Ax)$ and assume that for each $i = 1, 2, \ldots, d$ the real numbers $\sigma_i(\alpha_i)$ are distinct for all $\alpha_i = 1, 2, \ldots, r_i$. Then the cost function $(S \circ E)$ is a differentiable at the point $A$.

**Proof.** Let $\Theta(\epsilon) \in C^1((-\delta, \delta), SL_d(\mathbb{R}))$ with $\Theta(0) = A$. The $\epsilon$-dependent multilinear spectrum $\sigma_i(\alpha_i; \epsilon)$ ($i = 1, 2, \ldots, d, \alpha_i = 1, 2, \ldots, r_i$) of $u_{\sigma}(\Theta(\epsilon)x)$ are given by the eigenvalues of an $\epsilon$-dependent self-adjoint compact Hermitian operator. In a neighborhood of $\epsilon = 0$ the eigenvalue $\sigma_i(\alpha_i; \epsilon)$ admits a series expansion [21, p. xx]

$$\sigma_i(\alpha_i; \epsilon) = \sigma_i(\alpha_i; 0) + \epsilon \tilde{\sigma}_i(\alpha_i),$$

(B.21)
and thus $\sigma_i(\alpha_i; \epsilon)$ is differentiable with respect to $\epsilon$ at $\epsilon = 0$. Hence the sum of all eigenvalues

$$\sum_{i=1}^{d-1} \sum_{\alpha_i=1}^{r_i} \sigma_i(\alpha_i; \epsilon)$$

is differentiable at $\epsilon = 0$ and thus the cost function $C$ is differentiable at $A \in \text{SL}_d(\mathbb{R})$.

References

[1] P.-A. Absil, R. Mahony, and R. Sepulchre. *Optimization algorithms on matrix manifolds*. Princeton University Press, Princeton, NJ, 2008.

[2] H. Al-Daas, G. Ballard, and P. Benner. Parallel algorithms for tensor train arithmetic. *SIAM J. Sci. Comput.*, 44(1):C25–C53, 2022.

[3] R. Aris. *Vectors, tensors and the basic equations of fluid mechanics*. Dover publications, 1989.

[4] M. Bachmayr, R. Schneider, and A. Uschmajew. Tensor networks and hierarchical tensors for the solution of high-dimensional partial differential equations. *Found. Comput. Math.*, 16(6), 2016.

[5] J. A. Bengua, H. N. Phien, H. D. Tua, and N. M. Do. Efficient tensor completion for color image and video recovery: low-rank tensor train. *IEEE Trans. Image Process.*, 26(5):2466–2479, 2017.

[6] D. Bigoni, A. P. Engsig-Karup, and Y. M. Marzouk. Spectral tensor-train decomposition. *SIAM J. Sci. Comput.*, 38(4):A2405–A2439, 2016.

[7] S. Blanes, F. Casas, J. A. Oteo, and J. Ros. The Magnus expansion and some of its applications. *Phys. Rep.*, 470(5-6):151–238, 2009.

[8] H. Cho, D. Venturi, and G. E. Karniadakis. Numerical methods for high-dimensional probability density function equation. *J. Comput. Phys.*, 315:817–837, 2016.

[9] P. G. Constantine, E. Dow, and Q. Wang. Active subspace methods in theory and practice: applications to kriging surfaces. *SIAM J. Sci. Comput.*, 36(4):A1500–A1524, 2014.

[10] H. A. Daas, G. Ballard, P. Cazeaux, E. Hallman, A. Międlar, M. Pasha, T. W. Reid, and A. K. Saibaba. Randomized algorithms for rounding in the tensor-train format. *SIAM Journal on Scientific Computing*, 45(1):A74–A95, 2023.

[11] A. Dektor, A. Rodgers, and D. Venturi. Rank-adaptive tensor methods for high-dimensional nonlinear PDEs. *J. Sci. Comput.*, 88(36):1–27, 2021.

[12] A. Dektor and D. Venturi. Dynamically orthogonal tensor methods for high-dimensional nonlinear PDEs. *J. Comput. Phys.*, 404:109125, 2020.

[13] A. Dektor and D. Venturi. Dynamic tensor approximation of high-dimensional nonlinear PDEs. *J. Comput. Phys.*, 437:110295, 2021.

[14] A. Falcó, W. Hackbusch, and A. Nouy. Geometric structures in tensor representations. *arXiv:1505.03027*, pages 1–50, 2015.
[15] L. Grasedyck. Hierarchical singular value decomposition of tensors. *SIAM J. Matrix Anal. Appl.*, 31(4):2029–2054, 2009/10.

[16] L. Grasedyck and C. Löbbert. Distributed hierarchical SVD in the hierarchical Tucker format. *Numer. Linear Algebra Appl.*, 25(6):e2174, 2018.

[17] J. Heng, A. Doucet, and Y. Pokern. Gibbs flow for approximate transport with applications to Bayesian computation. *J. R. Stat. Soc. Series B*, 83:156–187, 2021.

[18] J. S. Hesthaven, S. Gottlieb, and D. Gottlieb. *Spectral methods for time-dependent problems*, volume 21 of *Cambridge Monographs on Applied and Computational Mathematics*. Cambridge University Press, Cambridge, 2007.

[19] G. E. Karniadakis and S. Sherwin. *Spectral/hp element methods for computational fluid dynamics*. Oxford University Press, second edition, 2005.

[20] A.-K Kassam and L. N. Trefethen. Fourth-order time-stepping for stiff pdes. *SIAM Journal on Scientific Computing*, 26(4):1214–1233, 2005.

[21] T. Kato. *Perturbation theory for linear operators*. Classics in Mathematics. Springer-Verlag, Berlin, 1995. Reprint of the 1980 edition.

[22] B. N. Khoromskij. Tensor numerical methods for multidimensional PDEs: theoretical analysis and initial applications. In *CEMRECS 2013—modelling and simulation of complex systems: stochastic and deterministic approaches*, volume 48 of *ESAIM Proc. Surveys*, pages 1–28. EDP Sci., Les Ulis, 2015.

[23] E. Kieri and B. Vandereycken. Projection methods for dynamical low-rank approximation of high-dimensional problems. *Comput. Methods Appl. Math.*, 19(1):73–92, 2019.

[24] O. Koch and C. Lubich. Dynamical tensor approximation. *SIAM J. Matrix Anal. Appl.*, 31(5):2360–2375, 2010.

[25] T. Kolda and B. W. Bader. Tensor decompositions and applications. *SIREV*, 51:455–500, 2009.

[26] C. Krumnow, L. Veis, Ö. Legeza, and J. Eisert. Fermionic orbital optimization in tensor network states. *Phys. Rev. Lett.*, 117:210402, 2016.

[27] C. Lu, J. Tang, S. Yan, and Z. Lin. Nonconvex nonsmooth low rank minimization via iteratively reweighted nuclear norm. *IEEE Trans. Image Process.*, 25(2):829–839, 2016.

[28] C. Lubich, I. V. Oseledets, and B. Vandereycken. Time integration of tensor trains. *SIAM J. Numer. Anal.*, 53(2):917–941, 2015.

[29] H. Luo and T. R. Bewley. On the contravariant form of the Navier-Stokes equations in time-dependent curvilinear coordinate systems. *J. Comput. Phys.*, 199:355–375, 2004.

[30] J. R. Magnus and H. Neudecker. *Matrix differential calculus with applications in statistics and econometrics*. John Wiley & Sons, 2019.

[31] C. Hubig I. P. McCulloch and U. Schollwöck. Generic construction of efficient matrix product operators. *Phys. Rev. B*, 95:035129, 2017.
[32] I. Oseledets and E. Tyrtyshnikov. TT-cross approximation for multidimensional arrays. *Linear Algebra Appl.*, 432(1):70–88, 2010.

[33] I. V. Oseledets. Tensor-train decomposition. *SIAM J. Sci. Comput.*, 33(5):2295—2317, 2011.

[34] A. Pinkus. *Ridge Functions*. Cambridge University Press, 2015.

[35] B. Recht, M. Fazel, and P. A. Parrilo. Guaranteed minimum-rank solutions of linear matrix equations via nuclear norm minimization. *SIAM Review*, 52(3):471–501, 2010.

[36] A. Rodgers, A. Dektor, and D. Venturi. Adaptive integration of nonlinear evolution equations on tensor manifolds. *J. Sci. Comput.*, 92(39):1–31, 2022.

[37] A. Rodgers and D. Venturi. Stability analysis of hierarchical tensor methods for time-dependent PDEs. *J. Comput. Phys.*, 409:109341, 2020.

[38] A. Rodgers and D. Venturi. Implicit integration of nonlinear evolution equations on tensor manifolds. *arXiv:2207.01962*, pages 1–23, 2023.

[39] M. Rosenblatt. Remarks on a multivariate transformation. *Ann. Math. Stat.*, 23:470–472, 1952.

[40] T. Schulte-Herbrüggen, S. J. Glaser, G. Dirr, and U. Helmke. Gradient flows for optimization in quantum information and quantum dynamics: foundations and applications. *Rev. Math. Phys.*, 22(6):597–667, 2010.

[41] T. Shi, M. Ruth, and A. Townsend. Parallel algorithms for computing the tensor-train decomposition. *arXiv:2111.10448*, pages 1–23, 2021.

[42] A. Spantini, D. Bigoni, and Y. Marzouk. Inference via low-dimensional couplings. *J. Mach. Learn. Res.*, 19:1–71, 2018.

[43] A. Uschmajew and B. Vandereycken. The geometry of algorithms using hierarchical tensors. *Linear Algebra Appl.*, 439(1):133–166, 2013.

[44] D. Venturi. Conjugate flow action functionals. *J. Math. Phys.*, (54):113502(1–19), 2013.

[45] D. Venturi. The numerical approximation of nonlinear functionals and functional differential equations. *Physics Reports*, 732:1–102, 2018.

[46] D. Venturi, M. Choi, and G.E. Karniadakis. Supercritical quasi-conduction states in stochastic Rayleigh-Bénard convection. *International Journal of Heat and Mass Transfer*, 55(13):3732–3743, 2012.

[47] D. Venturi and A. Dektor. Spectral methods for nonlinear functionals and functional differential equations. *Res. Math. Sci.*, 8(27):1–39, 2021.