Communications on Analysis and Computation
Vol. 1, No. 2, June 2023, pp. 135-156
doi:10.3934/cac.2023008

DENOISING CONVOLUTION ALGORITHMS AND APPLICATIONS TO SAR SIGNAL PROCESSING

Alina Chertock[1], Chris Leonard[1], Semyon Tsynkov[*1] and Sergey Utyuzhnikov[2]

[1]Department of Mathematics, North Carolina State University, Raleigh, NC, USA

[2]Department of Mechanical, Aerospace & Civil Engineering
University of Manchester, Manchester, UK

(Communicated by Hongyu Liu)

Abstract. Convolutions are one of the most important operations in signal processing. They often involve large arrays and require significant computing time. Moreover, in practice, the signal data to be processed by convolution may be corrupted by noise. In this paper, we introduce a new method for computing the convolutions in the quantized tensor train (QTT) format and removing noise from data using the QTT decomposition. We demonstrate the performance of our method using a common mathematical model for synthetic aperture radar (SAR) processing that involves a sinc kernel and present the entire cost of decomposing the original data array, computing the convolutions, and then reformatting the data back into full arrays.

1. **Introduction.** Convolution operations are used in different practical applications. They often involve large arrays of data and require optimization with respect to memory and computational cost. While input data are usually available only in a discrete form, the standard realization based on a vector-matrix representation is not often efficient since it leads to using sparse matrices. On the other hand, a tensor decomposition looks very attractive because it might reduce the volume of data very drastically, minimizing the number of zero elements. In addition, arithmetic operations between tensors can be implemented efficiently.

There are different forms of tensor decomposition. The most popular approach is based on the canonical decomposition [12] where a multidimensional array is represented (might be approximate) via a sum of outer products of vectors. For matrices, such decomposition is reduced to skeleton decomposition. However, it is known to be unstable in the cases of multiple tensor dimensions, also referred to as tensor modes. The Tucker decomposition [27] represents a natural stable generalization of the canonical decomposition and can provide a high compression rate. The main drawback of the Tucker decomposition is related to the so-called curse of dimensionality; that is, the algorithm's complexity grows exponentially with the number of tensor modes. A way to overcome these difficulties is to use the Tensor Train

2020 *Mathematics Subject Classification.* Primary: 15A69, 65F55, 86A22; Secondary: 68W20.
Key words and phrases. Tensor train decomposition (TT), randomized algorithm, quantized tensor train decomposition (QTT), low rank representation, synthetic aperture radar (SAR), ground reflectivity, imaging kernel, generalized ambiguity function (GAF), SAR resolution.

*Corresponding author: Semyon Tsynkov.

(TT) decomposition, which was originally introduced in [23, 24]. Effectively, the TT decomposition represents a generalization of the classical SVD decomposition to the case of multiple modes. It can also be interpreted as a hierarchical Tucker decomposition [10].

Computing the TT decomposition fully can be very expensive if we use the standard TT-SVD algorithm given, e.g., by Algorithm 1 below. Therefore, many modifications to this algorithm were proposed in the literature to help speed it up. One such improvement was presented in [18], where results comparable to those obtained by the TT-SVD algorithm were produced in a fraction of the time for sparse tensor data. Another algorithm that uses the column space of the unfolding tensors was designed to compute the TT cores in parallel; see [26]. The most popular approach to efficiently compute the TT decomposition is based on using a randomized algorithm; see, e.g., [1, 5, 13].

Maximal compression with the TT decomposition can be reached with matrices whose dimensions are powers of two, as proposed in the so-called Quantized TT (QTT) algorithm [20]. As shown in [15], the convolution realized for multilevel Toeplitz matrices via QTT has a logarithmic complexity with respect to the number of elements in each mode, N, and is proportional to the number of modes. It is proven that the result cannot be asymptotically improved. However, this algorithm is improved for finite and practically important $N \sim 10^4$ in [25] thanks to the cross-convolution in the Fourier (image) space. The improvement is demonstrated for convolutions with three modes with Newton's potential. It is to be noted that QTT can also be applied to the Fast Fourier Transform (FTT) to decrease its complexity, as shown in [3]. This super-fast FFT (QTT-FFT) beats the standard FFT for extremely large N such as $N \sim 2^{60}$ for one mode tensors and $N \sim 2^{20}$ for tensors with three modes.

For practical applications, a critical issue is denoising. Real-life data, such as radar signals, are typically contaminated with noise. Denoising is not addressed in the papers we have cited previously. However, TT decomposition itself potentially has the property of denoising, owing to the SVD incorporated in the algorithm [4,8]. In the current work, we propose and implement the low-rank modifications for the previously developed TT-SVD algorithm of [21]. These modifications speed up the computations. We also demonstrate the denoising capacity of numerical convolutions computed using the QTT decomposition. Specifically, we employ a common model for synthetic aperture radar (SAR) signal processing based on the convolution with a sinc imaging kernel (called the generalized ambiguity function) [6, Chapter 2] and show that when a convolution with this kernel is evaluated in the QTT format, the noise level in the resulting image is substantially reduced compared to that in the original data.

It should be observed that most papers on tensor convolution only consider the run time cost of the convolution after the tensor decomposition has been applied to the objective function and the kernel function and either ignores the cost of the actual tensor decompositions or puts it as a side note. In this paper, we consider every step of computing the convolution using the QTT-FFT algorithm, including the decomposition of the arrays into the QTT format using the TT-SVD algorithm (see Algorithm 1), computation of the QTT-FFT algorithm once in that format, and then extracting the data back after the computation is conducted (Section 5). As the QTT decomposition is computationally expensive, we consider several approaches

to speed up the decomposition run time. Without these modifications to the TT-SVD decomposition algorithm, the convolution can take a long to compute and is not a practical approach. We provide more detail in section 7.1.

The methods we use to speed up our TT decompositions are based on truncating SVD ranks in the decomposition algorithm (Algorithm 1) and lead to a significant noise reduction in the data (Section 6). Thus, in Chapter 6, we present algorithms to compute convolutions in a reasonable time while significantly reducing the noise in the data at the same time. Our contribution includes developing and analyzing new approaches to speeding up the tensor train decomposition, see Section 5. In Section 6, we consider the effects of convolutions on removing noise in data. Finally, in Section 7, we show numerical examples and compare our results with other approaches to computing convolutions.

2. Convolution.

The convolution operation is widely used in different applications in signal processing, data imaging, physics, and probability, to name a few. This operation is a way to combine two signals, usually represented as functions, and produce a third signal with meaningful information. The D-dimensional convolution between two functions f and g is defined as

$$I(\boldsymbol{x}) = [f * g](\boldsymbol{x}) = \int_{\mathbb{R}^D} f(\boldsymbol{y}) g(\boldsymbol{x} - \boldsymbol{y})\, d\boldsymbol{y}, \quad \forall \boldsymbol{x} \in \mathbb{R}^D. \tag{1}$$

Often to compute the convolution numerically, we assume the support of f and g, denoted $\text{supp}(f)$ and $\text{supp}(g)$ respectively, are compact. For simplicity, in this paper, we assume $\text{supp}(f) = \text{supp}(g) = [-L, L]^D$ for some $L \in \mathbb{R}$. Next, we discretize the domain $[-L, L]^D$ uniformly into N^D points such that

$$\boldsymbol{x_j} = (x_{j_1}, \ldots, x_{j_D}),$$
$$x_{j_d} = -L + \frac{\Delta x}{2} + j_d \Delta x, \quad j_d = 0, \ldots, N-1, \quad d = 1, \ldots, D,$$

where $\Delta x = \frac{2L}{N}$ and $\boldsymbol{j} = (j_1, \ldots, j_D)$. We then let $\boldsymbol{f}$ and $\boldsymbol{g}$ be D-dimensional arrays such that

$$\boldsymbol{f_j} = f(\boldsymbol{x_j}), \quad \boldsymbol{g_j} = g(\boldsymbol{x_j})$$

for all $\boldsymbol{j}$. This leads to the discrete convolution $\boldsymbol{I}$ such that

$$\boldsymbol{I_j} \coloneqq (\Delta x)^D \sum_{\boldsymbol{i}} \boldsymbol{f_i} \boldsymbol{g_{j-i+(\frac{N}{2}-1)\mathbf{1}}} \approx I(\boldsymbol{x_j}), \tag{2}$$

where $\mathbf{1} = (1, \ldots, 1)$ and the sums are over all indices $\boldsymbol{i} = (i_1, \ldots, i_D)$ that lead to legal subscripts. This Riemann sum approximation (2) to the integral (1) uses the midpoint rule, thus having $\mathcal{O}(\Delta x^2)$ accuracy.

Remark 2.1. The convolution defined in (2) is equivalent to Matlab's **convn** function with the optional shape input set to 'same' and then multiplied by $(\Delta x)^D$.

To compute this convolution directly takes $\mathcal{O}(N^{2D})$ operations, but it can be reduced to $\mathcal{O}(N^D \log(N^D))$ by using the fast Fourier transform (FFT) and the discrete convolution theorem. The FFT algorithm is an efficient algorithm used to compute the D-dimensional discrete Fourier transforms (DFT) of $\boldsymbol{\mathcal{V}} \in \mathbb{R}^{N \times \ldots \times N}$,

$$\hat{\boldsymbol{\mathcal{V}}}_{\boldsymbol{\alpha}} \coloneqq DFT(\boldsymbol{\mathcal{V}}) = \sum_{\boldsymbol{j}=\mathbf{0}}^{\boldsymbol{N-1}} \boldsymbol{\mathcal{V}}_{\boldsymbol{j}} \omega_N^{\boldsymbol{j} \cdot \boldsymbol{\alpha}}$$

where the sum is over the multi-indexed array $\boldsymbol{j}$,

$$\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_D), \quad \alpha_d = 0, \dots, N-1, \quad d = 1, \dots, D,$$
$$\boldsymbol{N} = (N, \dots, N), \quad \boldsymbol{0} = (0, \dots, 0),$$

and $\omega_N = e^{-\frac{2\pi\hat{\imath}}{N}}$, where $\hat{\imath} = \sqrt{-1}$ is the imaginary unit. Similarly, the D-dimensional inverse discrete Fourier transform (IDFT), such that

$$\boldsymbol{\mathcal{V}} = IDFT(DFT(\boldsymbol{\mathcal{V}})),$$

of the array $\hat{\boldsymbol{\mathcal{V}}} \in \mathbb{R}^{N\times\dots\times N}$ is given by

$$\boldsymbol{\mathcal{V}_j} = \frac{1}{N^D} \sum_{\boldsymbol{\alpha}=\boldsymbol{0}}^{\boldsymbol{N-1}} \hat{\boldsymbol{\mathcal{V}}}_{\boldsymbol{\alpha}} \omega_N^{-\boldsymbol{j}\cdot\boldsymbol{\alpha}}.$$

Using the discrete Fourier transform, we can compute the circular convolution $\boldsymbol{I}^c = (\boldsymbol{\mathcal{V}} \circledast \boldsymbol{\mathcal{W}})$ defined as

$$\boldsymbol{I}^c_{\boldsymbol{j}} = \sum_{\boldsymbol{i}=\boldsymbol{0}}^{\boldsymbol{N-1}} \boldsymbol{\mathcal{V}}_{\boldsymbol{i}} \bar{\boldsymbol{\mathcal{W}}}_{\boldsymbol{j-i}}$$
$$\bar{\boldsymbol{\mathcal{W}}}_{i_1,\dots,i_D} = \boldsymbol{\mathcal{W}}_{j_1,\dots,j_D}, \quad i_d \equiv j_d \bmod(N), \quad d = 1, \dots, D,$$

by taking the DFT of $\boldsymbol{\mathcal{W}}$ and $\boldsymbol{\mathcal{V}}$, multiplying the results together, and then taking the IDFT of the given result. Thus, we have

$$\boldsymbol{I}^c = IDFT(DFT(\boldsymbol{\mathcal{W}}) \odot DFT(\boldsymbol{\mathcal{V}}))$$

where $\odot$ is Hadamard product (element-wise product) of D-dimensional arrays. The circular convolution is the same as the convolution of two periodic functions (up to a constant scaling), thus to obtain the convolution given in (2) (also known as a linear convolution), we need to pad the vectors $\boldsymbol{f}$ and $\boldsymbol{g}$ with at least $N-1$ zeros in each dimension. For example, given the vectors $\boldsymbol{f}^0, \boldsymbol{g}^0 \in \mathbb{R}^{2N-1}$ with

$$\boldsymbol{f}^0_j = \begin{cases} \boldsymbol{f}_j & 0 \le j \le N-1 \\ 0 & j > N-1 \end{cases}, \quad \text{and} \quad \boldsymbol{g}^0_j = \begin{cases} \boldsymbol{g}_j & 0 \le j \le N-1 \\ 0 & j > N-1 \end{cases},$$

and $\boldsymbol{I}^c = (\boldsymbol{f}^0 \circledast \boldsymbol{g}^0)$ as the circular convolution between them, the linear convolution $\boldsymbol{I}$ in (2) is given by

$$\boldsymbol{I}_j = \Delta x \boldsymbol{I}^c_{j+\frac{N-1}{2}}, \quad j = 0, \dots, N-1.$$

In this paper, we let g be a predefined kernel, such as the SAR generalized ambiguity function (GAF) (see Section 3 and [6, Chapter 2] for detail) and f be a smooth gradually varying function contaminated with white noise. To compute the convolution, we use the QTT decomposition [16] and the QTT-FFT algorithm [3]. The QTT decomposition is a particular case of the more general TT decomposition (see Section 4 and [21] for detail).

3. **Synthetic aperture radar (SAR).** SAR is a coherent remote sensing technology capable of producing two-dimensional images of the Earth's surface from overhead platforms (airborne or spaceborne). SAR illuminates the chosen area on the surface of the Earth with microwaves (specially modulated pulses) and generates the image by digitally processing the returns (i.e., reflected signals). SAR processing involves the application of the matched filter and summation along the synthetic array, which is a collection of successive locations of the SAR antenna along the flight path. Matched filtering yields the image in the direction normal to

the platform flight trajectory or orbit (called cross-track or range), while summation along the array yields the image in the direction parallel to the trajectory or orbit (along-the-track or azimuth).

Mathematically, each of the two signal processing stages can be interpreted as the convolution of the signal received by the SAR antenna with a known function. Equivalently, it can be represented as a convolution of the ground reflectivity function, which is the unknown quantity that SAR aims to reconstruct the imaging kernel or generalized ambiguity function (GAF). The advantage of this equivalent representation is that it leads to a very convenient partition: the GAF depends on the imaging system's characteristics, whereas the target's properties determine the ground reflectivity function. Moreover, image representation via GAF allows one to see clearly how signal compression (a property that pertains to SAR interrogating waveforms) enables SAR resolution, i.e., the capacity of the sensor to distinguish between closely located targets.

In the simplest possible imaging scenario, when the propagation of radar signals between the antenna and the target is assumed unobstructed, and several additional assumptions also hold; the GAF in either range or azimuthal direction is given by the sinc (or spherical Bessel) function:

$$g(x) = A\,\mathrm{sinc}\left(\pi\frac{x}{\Delta_x}\right) \equiv A\frac{\sin\left(\pi\frac{x}{\Delta_x}\right)}{\pi\frac{x}{\Delta_x}}, \tag{3}$$

where the constant A is determined by normalization, x denotes a given direction, and the quantity Δ_x is the resolution in this direction. From the formula (3), we see that the resolution is defined as half-width of the sinc main lobe, i.e., the distance from is central maximum to the first zero. When x is the range direction (cross-track), the resolution Δ_x is inversely proportional to the SAR signal bandwidth, see [6, Section 2.4.4]. When x is the azimuthal direction (along-the-track), the resolution is inversely proportional to the length of the synthetic array, i.e., synthetic aperture, see [6, Section 2.4.3]. Note that lower values of Δ_x correspond to better resolution because SAR can tell between the targets located closer to one another. It can also be shown that as $\Delta_x \to 0$ the GAF given by (3) converges to the δ-function in the sense of distributions [7, Section 3.3]. In this case, the image, which is a convolution of the ground reflectivity with the GAF, coincides with ground reflectivity. This would be ideal because the image would reconstruct the unknown ground reflectivity exactly. This situation, however, is never realized in practice because having $\Delta_x \to 0$ requires either the SAR bandwidth (range direction) or synthetic aperture (azimuthal direction) to become infinitely large, which is not possible.

The literature on SAR imaging is vast. Among the more mathematical sources, we mention the monographs [2] and [6].

4. **Tensor train decomposition.** Consider the K-mode, tensor $\boldsymbol{\mathcal{A}} \in \mathbb{C}^{M_1\times\ldots\times M_K}$ such that

$$\boldsymbol{\mathcal{A}} = a(i_1,\ldots,i_K), \quad i_k = 0,\ldots,M_k-1, \quad k = 1,\ldots,K,$$

where M_k is the size of each mode, and $a(i_1,\ldots,i_K) \in \mathbb{C}$ are the elements of the tensor $\boldsymbol{\mathcal{A}}$ for all $i_k = 0,\ldots,M_k-1$ and $k = 1,\ldots,K$. The tensor train format of $\boldsymbol{\mathcal{A}}$ decomposes the tensor into K cores $\boldsymbol{\mathcal{A}}^{(k)} \in \mathbb{C}^{r_{k-1}\times M_k\times r_k}$ such that

$$a(i_1,\ldots,i_K) = \boldsymbol{A}^{(1)}_{i_1}\boldsymbol{A}^{(2)}_{i_2}\cdots\boldsymbol{A}^{(K)}_{i_K},$$

where the matrices $\boldsymbol{\mathcal{A}}^{(k)}(:,i_k,:) = \boldsymbol{A}_{i_k}^{(k)} \in \mathbb{C}^{r_{k-1}\times r_k}$, for all $i_k = 0,\dots,M_k-1$, $k = 1,\dots,K$ (In Matlab notation, $\boldsymbol{A}_{i_k}^{(k)} = \text{squeeze}(\boldsymbol{\mathcal{A}}^{(k)}(:,i_k,:))$, where squeez() is used to convert the $\mathbb{C}^{r_{k-1}\times 1\times r_k}$ tensor into a $\mathbb{C}^{r_{k-1}\times r_k}$ matrix). The matrix dimensions r_k, $k = 1,\dots,K$, are referred to as the TT-ranks of the tensor decomposition, and the 3−mode tensors $\boldsymbol{\mathcal{A}}^{(k)}$ are the TT-cores. Since we are interested in the case when $a(i_1,\dots,i_K) \in \mathbb{C}$, we impose the condition $r_0 = r_K = 1$. Let $M = \max_{1\le k\le K} M_k$ and $r = \max_{1\le k\le K-1} r_k$, then the the tensor $\boldsymbol{\mathcal{A}}$, which has $\mathcal{O}(M^K)$ elements, can be represented with $\mathcal{O}(MKr^2)$ elements in the TT format.

We can also represent the TT decomposition as the product of tensor contraction operators. Define the tensor contraction between the tensors $\boldsymbol{\mathcal{A}} \in \mathbb{C}^{M_1\times\dots\times M_K}$ and $\boldsymbol{\mathcal{B}} \in \mathbb{C}^{M_K\times\dots\times M_{\tilde{K}}}$ (note that the first dimension size of $\boldsymbol{\mathcal{B}}$ equals the last dimension size of $\boldsymbol{\mathcal{A}}$) as $\boldsymbol{\mathcal{C}} = \boldsymbol{\mathcal{A}}\circ\boldsymbol{\mathcal{B}} \in \mathbb{C}^{M_1\times\dots\times M_{K-1}\times M_{K+1}\times\dots\times M_{\tilde{K}}}$ where

$$\boldsymbol{\mathcal{C}}(i_1,\dots,i_{K-1},i_{K+1},\dots,i_{\tilde{K}}) = \sum_{p=0}^{M_K-1} \boldsymbol{\mathcal{A}}(i_1,\dots,p)\boldsymbol{\mathcal{B}}(p,\dots,i_{\tilde{K}}).$$

Then the TT format of $\boldsymbol{\mathcal{A}}$ can be represented as

$$\boldsymbol{\mathcal{A}} = \boldsymbol{\mathcal{A}}^{(1)}\circ\dots\circ\boldsymbol{\mathcal{A}}^{(K)}.$$

Before we show how to find the TT-cores, we first need to define a few properties of tensors. First, let the matrix $\boldsymbol{A}^{\{k\}}$ be the k-th unfolding of the tensor $\boldsymbol{\mathcal{A}}$ such that

$$\boldsymbol{A}^{\{k\}}(\alpha,\beta) = a(i_1,\dots,i_K),$$
$$\alpha = i_1 + i_2M_1 + \dots + i_k\Pi_{l=1}^{k-1}M_l, \quad \beta = i_{k+1} + i_{k+2}M_{k+1} + \dots + i_K\Pi_{l=k+1}^{K-1}M_l.$$

Thus, we have that $\boldsymbol{A}^{\{k\}} \in \mathbb{C}^{M_1M_2\dots M_k\times M_{k+1}M_{k+2}\dots M_K}$ which we write as

$$\boldsymbol{A}^{\{k\}} = a(i_1\dots i_k, i_{k+1}\dots i_K).$$

We denote the process of unfolding a tensor $\boldsymbol{\mathcal{A}}$ into a matrix $\boldsymbol{A}^{\{k\}} \in \mathbb{C}^{M_1M_2\dots M_k\times M_{k+1}M_{k+2}\dots M_K}$ as

$$\boldsymbol{A}^{\{k\}} = \text{reshape}(\boldsymbol{\mathcal{A}}, [M_1M_2\dots M_k, M_{k+1}M_{k+2}\dots M_K])$$

and folding a matrix into a tensor $\boldsymbol{\mathcal{A}} \in \mathbb{C}^{M_1\times\dots\times M_K}$ as

$$\boldsymbol{\mathcal{A}} = \text{reshape}(\boldsymbol{A}^{\{k\}}, [M_1, M_2, \dots, M_K]).$$

(Note this is to be consistent with the Matlab function reshape()).

From [21] it can be shown that there exist a TT-decomposition of $\boldsymbol{\mathcal{A}}$ such that

$$r_k = \text{rank}(A^{\{k\}}), \quad k = 1,\dots,K.$$

Denote the Frobenius norm of a tensor $\boldsymbol{\mathcal{A}} \in \mathbb{C}^{M_1\times\dots\times M_K}$ as

$$\|\boldsymbol{\mathcal{A}}\|_F = \sqrt{\sum_{i_1=0}^{M_1-1}\dots\sum_{i_K=0}^{M_K-1} |a(i_1,\dots,i_K)|^2},$$

and the ε_k-rank of the matrix $\boldsymbol{A}^{\{k\}}$ as

$$\text{rank}_{\varepsilon_k}(\boldsymbol{A}^{\{k\}}) := \min\{\text{rank}(\boldsymbol{B}) : \|\boldsymbol{A}^{\{k\}} - \boldsymbol{B}\|_F \le \varepsilon_k\}.$$

Given a set $\{\varepsilon_k\}_{k=1}^K$, we can approximate the tensor $\boldsymbol{\mathcal{A}}$ with a tensor $\tilde{\boldsymbol{\mathcal{A}}}$ in the TT format such that it has TT- ranks $\tilde{r}_k \le \text{rank}_{\varepsilon_k}(\boldsymbol{A}^{\{k\}})$ and

$$\|\boldsymbol{\mathcal{A}} - \tilde{\boldsymbol{\mathcal{A}}}\|_F \le \varepsilon, \quad \varepsilon^2 = \varepsilon_1^2 + \dots + \varepsilon_{K-1}^2.$$

In Algorithm 1, we present the TT-SVD algorithm [21], which computes a TT-decomposition of a tensor $\boldsymbol{\mathcal{A}}$ with a prescribed accuracy ε. In Section 5, we present some modifications to this algorithm that relax the prescribed tolerance and allow us to compute an approximate decomposition faster. For a tensor $\boldsymbol{\mathcal{A}} \in \mathbb{C}^{M_1 \times \ldots \times M_K}$, define

$$|\boldsymbol{\mathcal{A}}| = \text{number of elements in } \boldsymbol{\mathcal{A}} = M_1 M_2 \ldots M_K.$$

Algorithm 1: TT-SVD

input : $\boldsymbol{\mathcal{A}}$, ε
output: TT-Cores: $\boldsymbol{\mathcal{A}}^{(1)}, \boldsymbol{\mathcal{A}}^{(2)}, ..., \boldsymbol{\mathcal{A}}^{(K)}$
$\tau \coloneqq \frac{\varepsilon}{\sqrt{M-1}} \|\boldsymbol{\mathcal{A}}\|_F$
$r_0 \coloneqq 1$;
for *k=1,...,K-1* **do**
  $\boldsymbol{A}^{\{k\}} \coloneqq \text{reshape}(\boldsymbol{\mathcal{A}}, [M_k r_{k-1}, \frac{|\boldsymbol{\mathcal{A}}|}{M_k r_{k-1}}])$
  Compute truncated SVD: $\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^* + \boldsymbol{E} = \boldsymbol{A}^{\{k\}}$ such that $\|\boldsymbol{E}\|_F \leq \tau$
  $r_k \coloneqq \text{rank}(\boldsymbol{\Sigma}) = \text{rank}_\tau(\boldsymbol{A}^{\{k\}})$
  $\boldsymbol{\mathcal{A}}^{(k)} \coloneqq \text{reshape}(\boldsymbol{U}, [r_{k-1}, M_k, r_k])$
  $\boldsymbol{\mathcal{A}} \coloneqq \boldsymbol{\Sigma}\boldsymbol{V}^*$
end
$\boldsymbol{\mathcal{A}}^{(K)} \coloneqq \boldsymbol{\mathcal{A}}$

The TT-decomposition can also be applied to tensors with a small number of modes by using the quantized tensor train decomposition (QTT). For instance, let $\boldsymbol{v} \in \mathbb{C}^{2^K}$ be a vector (1-mode tensor). To apply the QTT-decomposition of $\boldsymbol{v}$, we reshape it into the K-mode tensor $\boldsymbol{\mathcal{V}} \in \mathbb{C}^{2 \times \ldots \times 2}$ such that

$$\boldsymbol{\mathcal{V}}(i_1, i_2, \ldots, i_K) = \boldsymbol{v}(i),$$

where

$$i = \sum_{k=1}^{K} i_k 2^{k-1}, \quad i_k = 0, 1,$$

then compute the TT-decomposition of the tensor $\boldsymbol{\mathcal{V}}$ (you can think of $i_K \ldots i_1$ as the binary representation of i). Extending the QTT-decomposition to matrices (2-mode tensors) $\boldsymbol{V} \in \mathbb{C}^{2^K \times 2^K}$ can be done similarly by reshaping them into $2K$-mode tensors $\boldsymbol{\mathcal{V}} \in \mathbb{C}^{2 \times \ldots \times 2}$, then computing the TT-decomposition of $\boldsymbol{\mathcal{V}}$.

We can approximate the discrete Fourier transform of a vector $\boldsymbol{v} \in \mathbb{R}^{2^K}$ (or 2D discrete Fourier transform of a matrix $\boldsymbol{V} \in \mathbb{R}^{2^K \times 2^K}$) in the QTT format using what is known as the QTT-FFT approximation algorithm [3]. Let $\hat{\boldsymbol{v}} = DFT(\boldsymbol{v})$ be the discrete Fourier transform of $\boldsymbol{v}$ and let $\boldsymbol{\mathcal{V}}$ and $\hat{\boldsymbol{\mathcal{V}}}$ be the tensors in the QTT-format that represent the vectors $\boldsymbol{v}$ and $\hat{\boldsymbol{v}}$ respectively. Given $\boldsymbol{\mathcal{V}}$, the QTT- FFT approximation algorithm can approximate $\hat{\boldsymbol{\mathcal{V}}}$ with a tensor $\tilde{\boldsymbol{\mathcal{V}}}$ such that

$$\|\tilde{\boldsymbol{\mathcal{V}}} - \hat{\boldsymbol{\mathcal{V}}}\|_F \leq \varepsilon \tag{4}$$

for some given tolerance ε. Similarly, we could prescribe some maximum TT-rank, $\hat{R}_{\max}$, for the QTT-FFT algorithm such that $\tilde{r}_k \leq \hat{R}_{\max}$ for all TT-ranks of $\tilde{\boldsymbol{\mathcal{V}}}$, $\{\tilde{r}_k\}_{k=0}^K$. The QTT-FFT algorithm can easily be modified to the inverse Fourier transform of a vector (or matrix) in the QTT format, which we denote as the QTT-iFFT algorithm.

5. **Computing the convolution with QTT decomposition.** In practice, we often need to compute the convolution (1), where f is the function of interest and g is a given kernel, but f is not given explicitly. Instead, we are given noisy data

$$(\boldsymbol{f}_\xi)_{\boldsymbol{j}} = f(\boldsymbol{x}_{\boldsymbol{j}}) + \xi_{\boldsymbol{j}} \tag{5}$$

at discrete points $\boldsymbol{x}_{\boldsymbol{j}}$, $\boldsymbol{j} = (j_1, \dots, j_D)$. In particular, representing the ground reflectivity function for SAR reconstruction in the form (5) helps one model the noise in the received data. We assume that $\xi_{\boldsymbol{j}}$ is white noise from a normal distribution with the standard deviation σ, i.e., $\xi_{\boldsymbol{j}} \sim \mathcal{N}(0, \sigma^2)$.

Since the kernel function g is known, we can discretize it as

$$\boldsymbol{g}_{\boldsymbol{j}} = g(\boldsymbol{x}_{\boldsymbol{j}}),$$

for the same $\boldsymbol{x}_{\boldsymbol{j}}$ values as in (5). We assume the D-dimensional spatial domain is uniformly discretized into N^D points where $N = 2^{K-1} - 1$, see (2). To compute the discrete convolution (2), we propose using the quantized tensor train (QTT) decomposition. To represent the arrays in the QTT format, we pad them with zeros such that the new arrays are D-mode tensors in $\mathbb{R}^{2^K \times \dots \times 2^K}$. We can relax the condition on the size N, but to compute the convolution with an FFT algorithm, we need to zero-pad each dimension with at least $N-1$ extra zeros (see Section 2). Also, for the QTT decomposition, we need each dimension to be of size 2^K for some $K \in \mathbb{N}$. Let $\boldsymbol{\mathcal{F}}_\xi, \boldsymbol{\mathcal{G}}$ be the zero-padded tensors representing $\boldsymbol{f}_\xi$ and $\boldsymbol{g}$ respectively in the QTT format. Here, we assume that the discretization of f, $\boldsymbol{f}$, has a low, but not exactly known, TT-rank in the QTT-format. This is motivated by the fact that many standard piecewise smooth functions naturally have a low TT-rank, see [9, 16, 22].

To find approximations of these tensors in the TT-format, we modify the original TT-SVD algorithm. This is because with the full TT-SVD algorithm, if the tolerance ε is small, see equation (4), the TT-decomposition has close to full rank. Not only does it take a very long time to compute these decompositions, but most of the noise is still present. However, if ε is too large, the TT-SVD algorithm loses too much information about the true function f. For these reasons, we present slight modifications to the TT-SVD algorithm. They are needed to significantly reduce the computing time, as illustrated by the example in Section 7.1.

We consider three different modifications to the TT-SVD algorithm. These modifications are as follows:

(1) Set some max rank $R_{\max}$ and truncate the SVD in Algorithm 1 with ranks less than or equal to this threshold. Denote this method as the **max rank TT-SVD** algorithm.
(2) Set some max rank $R_{\max}$ and replace the SVD in Algorithm 1 with a randomized SVD (RSVD) given in [11] with max ranks set to $R_{\max}$ (see Appendix A). Denote this method as the **max rank TT-RSVD** algorithm. Note that for this algorithm, we also need to prescribe an oversampling parameter p. We could choose from several randomized SVD algorithms, but due to simplicity and effectiveness, we use the approach described in Appendix A. This algorithm implements the direct SVD.
(3) Truncate the SVD in Algorithm 1 based on when there is a relative drop in singular values, i.e., if $\frac{\sigma_{k+1}}{\sigma_k} < \delta$ $(0 < \delta < 1)$ for a given threshold δ, then truncate the singular values less than σ_k. Denote this method as the **SV drop off TT-SVD** algorithm.

For the **max rank TT-RSVD**, if the unfolding matrices $\boldsymbol{A}^{\{k\}} \in \mathbb{R}^{m_k \times n_k}$, where $\min(m_k, n_k) \leq R_{\max} + p$, then we revert to the **max rank TT-SVD** algorithm (without the randomized SVD).

We can modify the QTT-FFT and QTT-iFFT algorithms similarly to our modifications of the TT-SVD algorithms to get a low-rank approximation to the discrete Fourier transform representations of $\boldsymbol{\mathcal{F}}_\xi$ and $\boldsymbol{\mathcal{G}}$. For this, we replace the SVD in the QTT-FFT algorithm (QTT-iFFT) with the truncated SVD algorithms (1)-(3) given above, but with possibly a different max rank which we denote $\hat{R}_{max}$ for (1) and (2), or different threshold $\hat{\delta}$ for (3). For the examples in Section 7, we distinguish between R_{max} and $\hat{R}_{max}$. However, we use the same threshold for δ in the TT-SVD algorithm and the QTT-FFT algorithm. Thus, we do not distinguish between the two. Note that using the threshold (1) in the QTT-FFT algorithm is not new and is mentioned in [3].

With these above modifications to the TT-SVD algorithm and QTT-FFT (QTT-iFFT) algorithms, we propose the following algorithm (Algorithm 2) to approximate the convolution between the D-dimensional arrays $\boldsymbol{f}$ and $\boldsymbol{g}$. For this algorithm, we denote

- $QFFT_{\hat{R}_{\max}(\hat{\delta})}$: QTT-FFT algorithm with a max rank of $\hat{R}_{\max}$ (or threshold $\hat{\delta}$),
- $QiFFT_{\hat{R}_{\max}(\hat{\delta})}$: QTT-iFFT algorithm with a max rank of $\hat{R}_{\max}$ (or threshold $\hat{\delta}$).

Algorithm 2: QTT convolution

input : $\boldsymbol{f}_\xi$, $\boldsymbol{g}$
output : $\boldsymbol{I}$
Step 1: $\boldsymbol{\mathcal{F}}_\xi = \text{reshape}(\boldsymbol{f}_\xi, [2, \ldots, 2])$, $\boldsymbol{\mathcal{G}} = \text{reshape}(\boldsymbol{g}, [2, \ldots, 2])$
Step 2: Decompose $\boldsymbol{\mathcal{F}}_\xi$ and $\boldsymbol{\mathcal{G}}$ into the QTT format using one of the modified TT-SVD algorithms.
Step 3: $\boldsymbol{\mathcal{I}} = QiFFT_{\hat{R}_{max}(\hat{\delta})}(QFFT_{\hat{R}_{max}(\hat{\delta})}(\boldsymbol{\mathcal{F}}_\xi) \odot QFFT_{\hat{R}_{max}(\hat{\delta})}(\boldsymbol{\mathcal{G}}))$.
Step 4: Retrieve $\boldsymbol{I}$ from $\boldsymbol{\mathcal{I}}$. (see Algorithm 3)

In Theorem 5.2, we show the asymptotic run time behavior of computing a convolution in one spatial dimension ($D = 1$) with the **max rank TT-SVD** algorithm. First, we prove an auxiliary result about the size of the unfolding matrices for this algorithm; see Lemma 5.1. For Theorem 5.2, we consider the whole process of converting the vector into the QTT-format, computing the convolution, then converting the convolution in the QTT format back into a vector, as is demonstrated in Algorithm 2. For the last step, to convert a tensor in the TT-format back into the standard format, we use the 'full' algorithm from the Matlab toolbox **oseledets/TT-Toolbox**. This is given in Algorithm 3. We then reshape this tensor into a vector with a bit of run time.

Lemma 5.1. *Let $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{2\times\ldots\times 2}$ be a K-mode tensor. Let $\{\boldsymbol{A}^{\{k\}}\}_{k=1}^{K-1}$ be the unfolding matrices of $\boldsymbol{\mathcal{A}}$ in the max rank TT-SVD algorithm with a max rank of $R_{\max}$ and with each $\boldsymbol{A}^{\{k\}} \in \mathbb{C}^{m_k \times n_k}$. Then*

$$m_k = 2r_{k-1} \leq 2R_{\max} \quad \textit{and} \quad n_k = 2^{K-k}.$$

Proof. Since $M_k = 2$ for all k, the proof for $m_k = 2r_{k-1} \leq 2R_{\max}$ is trivial by the first line inside the for loop in Algorithm 1. For n_k, we do a proof by induction.

Algorithm 3: Full

input : $\boldsymbol{\mathcal{A}}^{(1)}, \boldsymbol{\mathcal{A}}^{(2)}, ..., \boldsymbol{\mathcal{A}}^{(K)}$, and size of output tensor $[M_1, \ldots, M_k]$
output : $\boldsymbol{\mathcal{A}} \in \mathbb{C}^{M_1 \times \ldots \times M_k}$
Let $\boldsymbol{A} = \boldsymbol{\mathcal{A}}^{(1)}$
for k=2,...,K **do**
  $\boldsymbol{A} = \text{reshape}(\boldsymbol{A}, [\frac{(|\boldsymbol{A}|)}{r_{k-1}}, r_{k-1}])$
  $\boldsymbol{B} = \text{reshape}(\boldsymbol{\mathcal{A}}^{(k)}, [r_{k-1}, 2r_k])$
  $\boldsymbol{A} = \boldsymbol{AB}$
end
$\boldsymbol{\mathcal{A}} = \text{reshape}(\boldsymbol{A}, [M_1, \ldots, M_k])$

First, note that $|\boldsymbol{A}^{\{1\}}| = 2^K$ and $r_0 = 1$, thus

$$n_1 = \frac{|\boldsymbol{A}^{\{1\}}|}{2r_0} = \frac{2^K}{2} = 2^{K-1}.$$

Assume $n_\ell = 2^{K-\ell}$ for all $1 \le \ell \le k-1$. Then,

$$n_k = \frac{|\boldsymbol{A}^{\{k\}}|}{2r_{k-1}} = \frac{|\boldsymbol{\Sigma}_{k-1}\boldsymbol{V}_{k-1}^*|}{2r_{k-1}} = \frac{r_{k-1}n_{k-1}}{2r_{k-1}} = \frac{n_{k-1}}{2} = \frac{2^{K-(k-1)}}{2} = 2^{K-k}.$$

Thus, we get

$$m_k = 2r_{k-1} \le 2R_{\max} \quad \text{and} \quad n_k = 2^{K-k}.$$

□

Theorem 5.2. *Let $\boldsymbol{f}_\xi, \boldsymbol{g} \in \mathbb{R}^{2^{K-1}-1}$ for some positive integer K. Then the computational complexity, $C_{QTT\text{-}conv}$, of approximating the convolution $\boldsymbol{f}_\xi * \boldsymbol{g}$ with the max rank TT-SVD and max rank QTT-SVD algorithms described above is*

$$C_{QTT\text{-}conv} \le \mathcal{O}(R_{\max}^2 2^K),$$

where $R_{\max}$ is the prescribed max rank for both the TT-SVD algorithms and the QTT-FFT algorithm.

Proof. We show that the computational complexity is dominated asymptotically by the max rank TT-SVD algorithms and the full tensor algorithm. First, let C_{svd} be the computational cost of the SVD in big $\mathcal{O}$ notation. Then, for a matrix $\boldsymbol{A} \in \mathbb{C}^{m\times n}$, $C_{svd}(\boldsymbol{A}) = \mathcal{O}(mn\min(m,n))$. Note that, in Algorithm 1 (as well as in our max rank modifications), the computational complexity is dominated by the SVD algorithm. Denote the unfolding matrices at the kth iterations as $\boldsymbol{A}^{\{k\}} \in \mathbb{C}^{m_k \times n_k}$. Hence, the computational cost of the max rank TT-SVD algorithm is

$$\begin{aligned}
\sum_{k=1}^{K-1} C_{\text{svd}}(\boldsymbol{A}^{\{k\}}) &= \sum_{k=1}^{K-1} \mathcal{O}(m_k n_k \min(m_k, n_k)) \\
&\le \sum_{k=1}^{K-1} \mathcal{O}((2R_{\max})^2 2^{K-k}) \\
&= 4R_{\max}^2 \sum_{k=1}^{K-1} \mathcal{O}(2^k) \\
&= 4R_{\max}^2 \mathcal{O}(2^K - 2) \\
&= \mathcal{O}(R_{\max}^2 2^K).
\end{aligned}$$

From [3], we have that for the QTT-FFT and QTT-iFFT algorithms, the computational complexity is $\mathcal{O}(K^2R_{\max}^3)$. In Algorithm 3, the computational complexity comes from the multiplication $\boldsymbol{AB}$ in every loop. For the kth loop, $\boldsymbol{A} \in \mathbb{C}^{2^{k-1}\times r_{k-1}}$ and $\boldsymbol{B} \in \mathbb{R}^{r_{k-1}\times 2r_k}$ for $k = 2,\dots,K$, thus the computational complexity is proportional to the cost of multiplying $\boldsymbol{A}$ by $\boldsymbol{B}$, i.e.,

$$\begin{aligned} C_{\text{full}} &= \sum_{k=2}^{K} \mathcal{O}(2^{k-1} r_{k-1} 2 r_k) \\ &\le R_{\max}^2 \sum_{k=2}^{K} \mathcal{O}(2^k) \\ &= R_{\max}^2 \mathcal{O}(2^{K+1} - 4) \\ &= \mathcal{O}(R_{\max}^2 2^K). \end{aligned}$$

Hence, the total computational complexity is

$$\mathcal{O}(R_{\max}^2 2^K) + \mathcal{O}(K^2 R_{\max}^3) + \mathcal{O}(R_{\max}^2 2^K) = \mathcal{O}(R_{\max}^2 2^K).$$

□

For the randomized SVD, we have the computational complexity $C_{rsvd}(A^{\{k\}}) = \mathcal{O}(m_k n_k (R_{\max} + p)) = \mathcal{O}(2^{K-k} R_{\max}(R_{\max} + p))$. Thus, the run time for the convolution with a max rank TT-RSVD is similar when p is small. In D spatial dimensions, we can obtain a similar result but by replacing K with DK in the max rank TT-SVD algorithm and the full tensor algorithm, and the QTT-FFT algorithm is $\mathcal{O}(DK^2 R_{\max}^3)$. Hence, the total run time complexity in D spatial dimensions is $\mathcal{O}(R_{\max}^2 2^{DK})$.

6. **Denoising.** It is well known that the SVD can remove noise from matrix data, as seen in [4, 14], but little research has been done in denoising with tensor decompositions. In [17] and [19], the Tucker decomposition was used to help remove noise from point cloud data and electron holograms, respectively. In [8], it was shown that the TT-decomposition might have some advantages to denoising as opposed to the Tucker decomposition. This is because a low-rank Tucker matrix guarantees a low TT-rank for the data. However, the converse statement is not always true.

Let $\boldsymbol{\mathcal{F}}$ be the low TT-rank tensor representing $\boldsymbol{f}$ in the QTT format. Then for some core tensors $\boldsymbol{\mathcal{F}}^{(k)} \in \mathbb{R}^{r_{k-1}\times 2\times r_k}$ with tensor slices $\boldsymbol{\mathcal{F}}^{(k)}(:, i_k, :) = \boldsymbol{f}_{i_k}^{(k)} \in \mathbb{R}^{r_{k-1}\times r_k}$, $i_k = 0, 1$. Each element of $\boldsymbol{\mathcal{F}}$ can be represented in the TT format as

$$\boldsymbol{\mathcal{F}}(i_1, \dots, i_k) = \boldsymbol{f}_{i_1}^{(1)} \dots \boldsymbol{f}_{i_K}^{(K)}, \quad i_k = 0, 1, \quad k = 1, \dots, K,$$

where each $\boldsymbol{f}_{i_k}^{(k)}$ is a low rank matrix. In practice, it is unlikely the data collected has a low-rank TT decomposition since almost all real radar data has noise due to hardware limitations or other signals interfering with the data. Instead, we have the noisy data $\boldsymbol{f}_\xi$ whose tensor representation is

$$\boldsymbol{\mathcal{F}}_\xi = \boldsymbol{\mathcal{F}} + \boldsymbol{\xi},$$

where $\boldsymbol{\xi}$ is the realization of the random noise in the TT format. The tensor $\boldsymbol{\mathcal{F}}_\xi$ almost surely has full TT-rank when represented exactly in the QTT format. Ideally, we would like to be able to find an approximate TT decomposition $\tilde{\boldsymbol{\mathcal{F}}}$ with TT-cores $\tilde{\boldsymbol{\mathcal{F}}}^{(k)}$, $k = 1, \dots, K$, using the noisy data such that $\tilde{\boldsymbol{\mathcal{F}}}^{(k)} \approx \boldsymbol{\mathcal{F}}^{(k)}$. However, it is hard to guarantee any bound on this. We argue, though, that by using our

proposed methods when given the noisy data $\boldsymbol{\mathcal{F}}_\xi$, we can find a TT decomposition $\tilde{\boldsymbol{\mathcal{F}}}$ with low rank such that $\tilde{\boldsymbol{\mathcal{F}}} \approx \boldsymbol{\mathcal{F}}$.

Consider the first iteration of the for loop of algorithm 1, with $\boldsymbol{\mathcal{A}} = \boldsymbol{\mathcal{A}}_0 + \boldsymbol{\mathcal{A}}_\xi$ as the sum of a smooth tensor ($\boldsymbol{\mathcal{A}}_0$) and a noisy tensor ($\boldsymbol{\mathcal{A}}_\xi$). Then, after it is reshaped, we obtain the matrix

$$\boldsymbol{A}^{\{1\}} = \boldsymbol{A}_0^{\{1\}} + \boldsymbol{A}_\xi^{\{1\}},$$

where $\boldsymbol{A}_0^{\{1\}}$ is a low rank matrix and $\boldsymbol{A}_\xi^{\{1\}}$ is added noise. Let $\boldsymbol{A}^{\{1\}} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^* + \boldsymbol{E}$ be the truncated SVD of $\boldsymbol{A}^{\{1\}}$ and $\boldsymbol{A}_0^{\{1\}} = \boldsymbol{U}_0\boldsymbol{\Sigma}_0\boldsymbol{V}_0^*$ be the SVD of $\boldsymbol{A}_0^{\{1\}}$. Note that $\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^* \approx \boldsymbol{A}_0^{\{1\}}$ does not imply that $\boldsymbol{U} \approx \boldsymbol{U}_0$, and thus the TT-core $\boldsymbol{\mathcal{A}}^{(1)}$ is not guaranteed to be approximately equal to $\boldsymbol{\mathcal{A}}_0^{(1)}$, where $\boldsymbol{\mathcal{A}}_0^{(1)}$ is the first TT-core of $\boldsymbol{\mathcal{A}}_0$. However, if we let $\boldsymbol{\mathcal{A}}^2 = \boldsymbol{\mathcal{A}}$ on the second iteration of the loop in Algorithm 1 (and similarly for $\boldsymbol{\mathcal{A}}_0$), we do get that the elements of the tensor contraction $\boldsymbol{\mathcal{A}}^{(1)} \circ \boldsymbol{\mathcal{A}}^2 \approx \boldsymbol{\mathcal{A}}_0^{(1)} \circ \boldsymbol{\mathcal{A}}_0^2$. Similarly, if we can approximate the noise-free component on every iteration of the for loop, we obtain an approximation for the tensor $\boldsymbol{\mathcal{A}}_0$. While we do not have a theoretical bound on this error, our experiments in Section 7 show that this method works well at removing the noise. Since our method computes multiple SVDs, it can reduce a lot more noise than if we just did a single SVD and can do so without excessive smearing.

7. Numerical simulations.

This section presents some examples in one and two spatial dimensions. The original code for the TT-decompositions and the QTT-FFT algorithms comes from the Matlab toolbox **oseledets/TT-Toolbox**. We have modified it accordingly for the **max rank TT-SVD**, **max rank TT-RSVD**, and **SV drop off TT-SVD** algorithm, as discussed in Section 5. For all our examples, we compare the run time and errors of computing the convolution (1) using several methods. The error for every example is the l_2 relative error

$$E_2(\boldsymbol{I}) = \frac{\|\boldsymbol{I} - \boldsymbol{I}_{\text{ref}}\|_2}{\|\boldsymbol{I}_{\text{ref}}\|_2}, \tag{6}$$

where in D spatial dimensions

$$\|\boldsymbol{I}\|_2 = \sqrt{\frac{1}{N^D} \sum_{\boldsymbol{j}=\boldsymbol{0}}^{\boldsymbol{N-1}} |\boldsymbol{I_j}|^2}.$$

The reference solution, $\boldsymbol{I}_{\text{ref}}$, is the discrete convolution (2) computed without any noise. In all of the examples, we compare our methods against computing the convolution with the randomized TT-SVD algorithm from [13], as well as computing the true noisy convolution with FFT. In two space dimensions, we also approximate the convolution using a low matrix rank approximation to the noisy data $\boldsymbol{f}_\xi$, where the truncated rank is determined by the actual matrix rank of $\boldsymbol{f}$.

For all of these examples, we use the normalized sinc imaging kernel that corresponds to the GAF (3) truncated to a sufficiently large interval $[-L, L]$:

$$g(x) = \frac{\operatorname{sinc}(\pi \frac{x}{\Delta_x})}{\int_{-L}^{L} \operatorname{sinc}(\pi \frac{x}{\Delta_x})\, dx}, \quad x \in [-L, L] \tag{7}$$

for $D = 1$, and

$$g(x,y) = \frac{\operatorname{sinc}(\pi \frac{x}{\Delta_x})\operatorname{sinc}(\pi \frac{y}{\Delta_y})}{\int\int_{-L}^{L} \operatorname{sinc}(\pi \frac{x}{\Delta_x})\operatorname{sinc}(\pi \frac{y}{\Delta_y})\, dxdy}, \quad (x,y) \in [-L,L] \times [-L,L] \tag{8}$$

for $D = 2$, where the resolution Δ_x in (7) and $\Delta_x = \Delta_y$ in (8) is a given parameter. The one- dimensional kernel (7) for $\Delta_x = 0.04\pi$ is shown in Figure 1.

In Table 1, we present the relative error for each example for $K = 20$ when $D = 1$, and $K = 10$ when $D = 2$. In this table, the convolution $\boldsymbol{f}_\xi * \boldsymbol{g}$ is denoted by $\boldsymbol{I}_\xi$ and computed using the FFT algorithm, the QTT-convolution computed with the **max rank TT-SVD** algorithm is denoted by $\boldsymbol{I}_{QTT_0}$, the QTT-convolution computed with the **max rank TT-RSVD** is denoted by $\boldsymbol{I}_{QTT_r}$, and the convolution computed using the **SV drop off TT-SVD** algorithm is denoted by I_δ. In turn, the convolutions computed using the randomized TT-decomposition are denoted by $\boldsymbol{I}_{RTT}$, and in two dimensions, the convolution computed using low-rank approximations of $\boldsymbol{f}$ is denoted by $\boldsymbol{I}_{lr}$. For I_δ and $\boldsymbol{I}_{lr}$, we also denote what parameter δ and truncation matrix rank R are used, respectively, for each example using a subscript of the error.

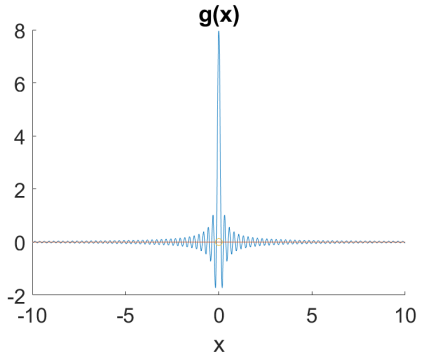

FIGURE 1. Kernel function (7) with $\Delta_x = 0.04\pi$.

For each example, we show the TT-ranks of the original function without noise, $\boldsymbol{f}$, in the QTT format given by the tensor $\boldsymbol{\mathcal{F}}$. This QTT approximation is computed with Algorithm 1 with the tolerance $\varepsilon = 10^{-10}$. We compute these TT-ranks for $K = 20$ when $D = 1$ and $K = 10$ when $D = 2$. However, it is worth noting that these TT-ranks do not change much for any data size. Notice that the max TT-ranks we choose for our algorithms are less than the TT-ranks of $\boldsymbol{f}$ from Algorithm 1, yet still provide a reasonable estimate.

TABLE 1. l_2-norm relative error for $K = 20$ for examples 1 and 2, and $K = 10$ for example 3.

example	$\boldsymbol{I}_\xi$	$\boldsymbol{I}_{QTT_0}$	$\boldsymbol{I}_{QTT_r}$	$\boldsymbol{I}_\delta$	$\boldsymbol{I}_{RTT}$	$\boldsymbol{I}_{lr}$
1	0.0383	**0.0028**	0.0102	$0.0280_{\delta=0.02}$	0.0430	-
2	0.0131	**0.0011**	0.0075	$0.0068_{\delta=0.01}$	0.0201	-
3	0.1142	**0.0151**	0.0447	$0.1650_{\delta=0.09}$	0.1534	$0.0470_{rank=23}$

7.1. **Example 1.** For this example, let

$$f(x) = e^{-(\frac{3x}{10})^2}(0.4\sin(8\pi x) - 0.7\cos(6\pi x)), \quad x \in [-10, 10],$$

and

$$(f_\xi)_j = f(x_j) + \xi_j,$$
$$x_j = -10 + \frac{\Delta x}{2} + j\Delta x, \quad j = 0, \dots, N-1, \quad \Delta x = \frac{20}{N}, \quad N = 2^{K-1} - 1,$$

with $\xi_j \sim \mathcal{N}(0, 0.02)$. We also set the resolution $\Delta_x = 4\Delta x$ in (7), where Δx is the size of the spatial discretization. Thus, the width of the main lobe of the sinc is $8\Delta x$ on the x-axis.

As we can see in Figure 2, the FFT-QTT algorithm removed much of the noise in the data compared to the true convolution. For $K = 20$, we also tried computing the convolution using the original TT-SVD algorithm given in Algorithm 1 with multiple values of ε. The smallest error, as defined in (6), occurred when $\varepsilon = 0.01$ and gave the relative error of $E_2(\boldsymbol{I}) = 0.03202$. This is close to the error of the true convolution of the noisy data and took over 100 seconds to compute. However, as we can see in Table 2, the run times for all of our methods on the same grid took less than a second. This indicates that the original TT-SVD algorithm is practically unsuitable for removing data noise.

The max TT-rank of the discretization of $f(x)$ in the QTT format, $\boldsymbol{\mathcal{F}}$, is 17, yet we were able to achieve our approximation using a max rank of $R_{max} = 10$ for the **max rank TT-SVD** and **max rank TT-RSVD** algorithms and $\hat{R}_{max} = 15$ for the QTT-FFT algorithm. Thus, even if we do not know the exact TT-rank, we can still compute a good approximation.

Table 2 shows run times for different grid sizes for each method. We can see that computing the convolution with FFT is faster than our methods for these values of K. However, the convolution with our QTT methods gets closer to the FFT run time as K increases. This is shown in the last column of Table 2 where we see the ratio of the **max rank TT-SVD** convolution method to the FFT convolution method is getting smaller as K grows. This helps verify our theoretical result that for some constant max rank R_{max} (and $\hat{R}_{max}$), the **max rank TT-SVD** convolution method is asymptotically faster than computing the convolution with FFT. The amount of data needed for our method to outperform the FFT method may be impractical for most real-world applications in 1-2 spatial dimensions.

TABLE 2. Run time (seconds): Example 1 convolutions.

K	$\boldsymbol{I}_\xi$	$\boldsymbol{I}_{QTT_0}$	$\boldsymbol{I}_{QTT_r}$	$\boldsymbol{I}_\delta$	$\boldsymbol{I}_{RTT}$	$\boldsymbol{I}_{QTT_0}/\boldsymbol{I}_\xi$
16	**0.005**	0.325	0.369	$1.485_{\delta=0.02}$	0.414	65
20	**0.067**	0.653	0.694	$0.479_{\delta=0.02}$	0.609	9.7463
24	**1.21**	4.41	4.86	$3.10_{\delta=0.02}$	2.80	3.6446
26	**5.77**	17.75	19.68	$13.93_{\delta=0.02}$	10.97	3.0763
28	66.6	95.5	108.7	$79.0_{\delta=0.02}$	**62.49**	1.4339

Table 3 shows the number of elements to represent the data $\boldsymbol{f}_\xi$ fully versus how many elements are required to store the data in the QTT-format with a prescribed max rank of $R_{max} = 10$, $\boldsymbol{\mathcal{F}}_\xi$, in Example 1. As we can see, storing all the elements takes a lot of data and grows exponentially in K, while storing the elements in the

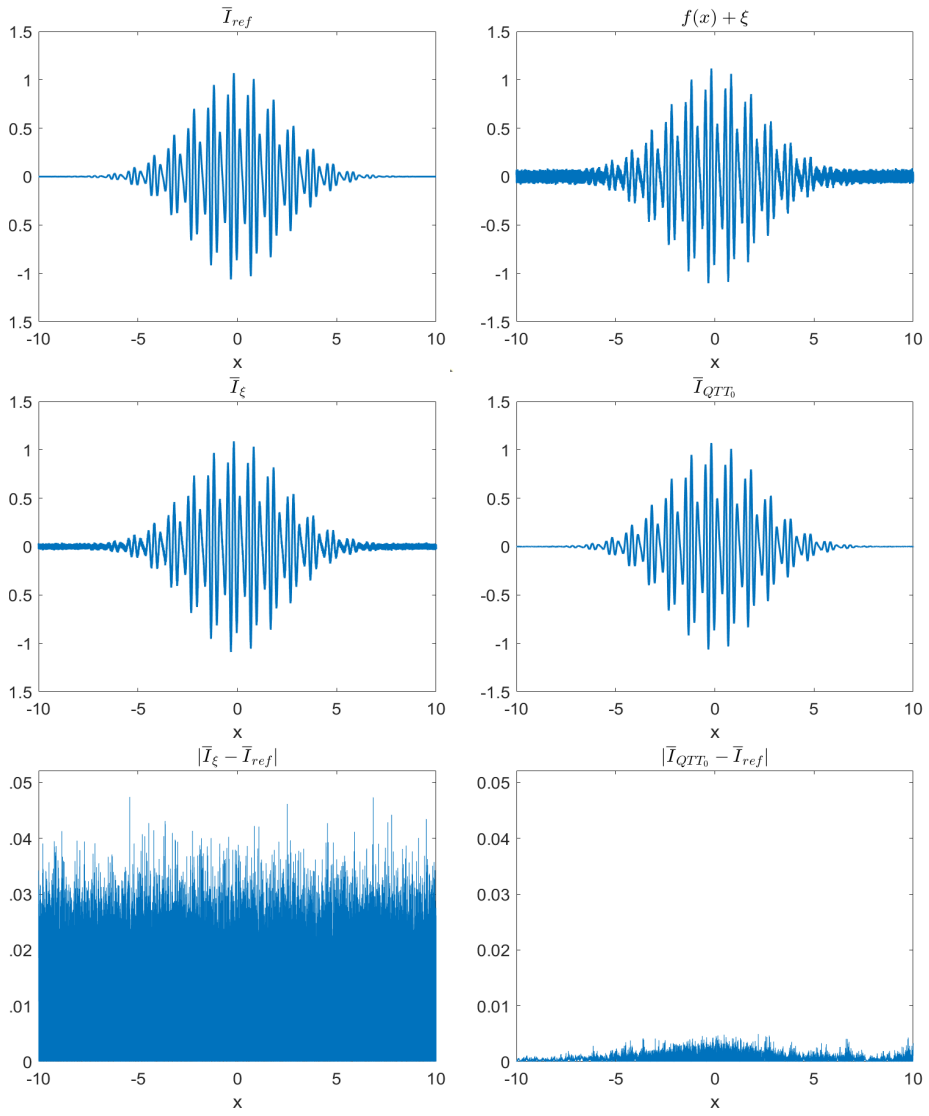

FIGURE 2. *Top Left:* True convolution of data without noise, $\boldsymbol{I}$. *Top Right:* Function data with noise, $\boldsymbol{f}_{\xi}$. *Middle Left:* True convolution of data with noise, $\boldsymbol{I}_{\xi}$. *Middle Right:* Convolution using the **max rank TT-SVD** algorithm, $\boldsymbol{I}_{QTT_0}$. *Bottom Left:* Absolute error of $\boldsymbol{I}_{\xi}$. *Bottom Right:* Absolute error of $\boldsymbol{I}_{QTT_0}$.

QTT format takes a lot less data and only grows linearly in K. These values for the QTT-data storage can be found by looking at the size of the core tensors. For the tensor $\boldsymbol{\mathcal{F}}_{\xi}$ in the QTT format and with a max TT-rank of $R_{max} = 10$, we have the TT-cores

$$\boldsymbol{\mathcal{F}}_{\xi}^{(1)}, \boldsymbol{\mathcal{F}}_{\xi}^{(K)} \in \mathbb{R}^{1\times 2\times 2},$$
$$\boldsymbol{\mathcal{F}}_{\xi}^{(2)}, \boldsymbol{\mathcal{F}}_{\xi}^{(K-1)} \in \mathbb{R}^{2\times 2\times 4},$$

TABLE 3. Data storage for Example 1.

K	f_ξ	$\boldsymbol{\mathcal{F}}_\xi$
16	65,536	2088
20	1,048,576	2888
24	16,777,216	3688
26	67,108,864	4088
28	268,435,456	4488

$$\boldsymbol{\mathcal{F}}_\xi^{(3)}, \boldsymbol{\mathcal{F}}_\xi^{(K-2)} \in \mathbb{R}^{4\times 2\times 8},$$
$$\boldsymbol{\mathcal{F}}_\xi^{(4)}, \boldsymbol{\mathcal{F}}_\xi^{(K-3)} \in \mathbb{R}^{8\times 2\times 10},$$
$$\boldsymbol{\mathcal{F}}_\xi^{(k)} \in \mathbb{R}^{10\times 2\times 10}, \quad k = 5, \dots, K-4.$$

Thus, the number of elements, N_{el}, that make up this QTT tensors is:

$$N_{el} = 2(1\times 2\times 2) + 2(2\times 2\times 4) + 2(4\times 2\times 8) + 2(8\times 2\times 10) + (K-8)(10\times 2\times 10).$$

The **max rank TT-RSVD** algorithm is not able to produce results as good as the **max rank TT-SVD** (see Table 1 for relative error comparison and Table 2 for a run time comparison) but is still able to produce a reasonably low error. While the run time for the **max rank TT-SVD** is faster than the **max rank TT-RSVD** for all of our methods, the **max rank TT-RSVD** can be faster for tensors with larger mode sizes. This is due to the SVD in **max rank TT-SVD** algorithm with mode sizes, M_k, may be computed on a matrix with $m_k = M_k R_{\max}$ rows. In contrast, for the **max rank TT-RSVD** algorithm, the SVD is computed on a matrix with $m_k = R_{\max} + p$ rows when $M_k = 2$ (such as for the QTT decomposition). The difference in the sizes of m_k does not make up for the extra amount of work the RSVD algorithm does. Although this paper focuses on the QTT-decomposition and thus $M_k = 2$, we believe this is important to note as the **max rank TT-RSVD** algorithm can speed up the TT-decomposition for higher mode tensor data and still produce accurate approximations. We verify this by computing the **max rank TT-SVD** algorithm and the **max rank TT-RSVD** algorithm on a tensor with $K = 8$ modes with each mode of size $M_k = 10$, $k = 1, \dots, K$. Each element of this tensor is taken from the uniform distribution $\mathcal{U}[0, 1)$. The max rank TT-SVD algorithm took 9.57 seconds, and the **max rank TT-RSVD** algorithm only took 5.12 seconds, almost half the time of the **max rank TT-SVD** algorithm.

7.2. **Example 2.** If we were to choose a coarser resolution for the example of Section 7.1 (i.e., a wider sinc function), we could reduce the noise using the standard convolution at the cost of smoothing out the solution's peaks. Doing this gives similar results for the true convolution and with our methods (Section 5). In this section, we show an example where the ground reflectivity is very oscillatory. Here, the resolution Δ_x determined by the GAF must be small (i.e., the sinc function must be "skinny"). Otherwise, if the sinc window is close to or larger than the characteristic scale of variation of the ground reflectivity, then the convolution can smooth out the actual oscillations instead of just the noise, losing most of the information in f.

We choose the ground reflectivity as

$$f(x) = e^{-(3x)^2}\left(0.9\sin\left(\frac{2x\pi}{5\Delta x}\right) + 1.4\cos\left(\frac{x\pi}{3\Delta x}\right)\right), \quad x \in [-1, 1],$$

and

$$(f_\xi)_j = f(x_j) + \xi_j,$$
$$x_j = -1 + \frac{\Delta x}{2} + j\Delta x, \quad j = 0, \ldots, N-1, \quad \Delta x = \frac{2}{N}, \quad N = 2^{K-1} - 1,$$

with $\xi_j \sim \mathcal{N}(0, .01)$. We use $\Delta_x = 2\Delta x$ and the max TT-rank of the discretization of the smooth function $f(x)$ in the QTT-format is 26.

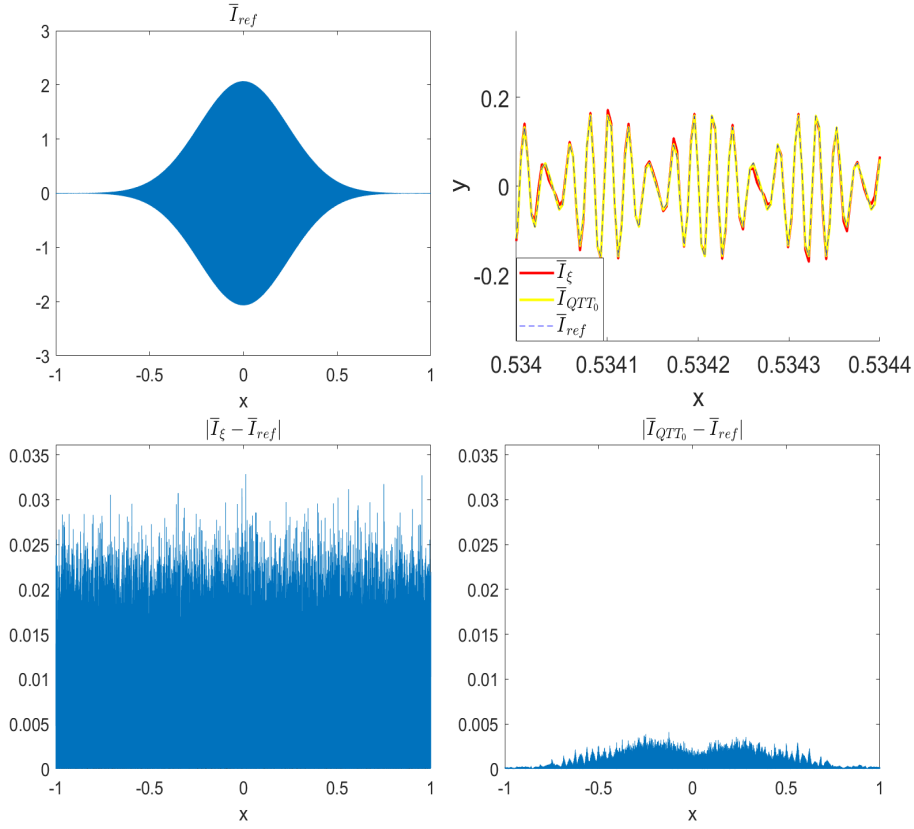

FIGURE 3. *Top Left:* True convolution of data without noise, $\boldsymbol{I}$. *Top Right:* Zoomed in graph of $\boldsymbol{I}_\xi$, $\boldsymbol{I}_{QTT_0}$, and $\boldsymbol{I}_{ref}$. *Bottom Left:* Absolute error of $\boldsymbol{I}_\xi$. *Bottom Right:* Absolute error of $\boldsymbol{I}_{QTT_0}$.

Again, we use the max ranks of $R_{max} = 10$ and $\hat{R}_{max} = 15$ for the **max rank TT-SVD** and QTT-SVD algorithms, respectively. As the function is too oscillatory to see a lot of helpful information in the full graph (see top left of Figure 3), we show a zoomed-in plot of the graph of $\boldsymbol{I}_\xi$, $\boldsymbol{I}_{QTT_0}$, and $\boldsymbol{I}_{\text{ref}}$ (see top right of Figure 3). While there is some error, the QTT-FFT convolution agrees with the true convolution $\boldsymbol{I}_{ref}$ very well, whereas $\boldsymbol{I}_\xi$ has a more considerable noticeable difference. This is verified by the graphs of the absolute error given in Figure 3, where the bottom left shows the error for $\boldsymbol{I}_\xi$, and the bottom right shows the error for $\boldsymbol{I}_{QTT_0}$.

7.3. **Example 3.** Let

$$f(x, y) = e^{-((2x)^2 + (2y)^2)}(\sin(2\pi x) - \cos(7\pi y) + \cos(4\pi xy) - \sin(3\pi xy),$$

$$(x, y) \in [-1, 1] \times [-1, 1]$$

and

$$(\boldsymbol{f}_\xi)_{j,k} = f(x_j, y_k) + \xi_{j,k},$$
$$x_j = -1 + \frac{\Delta x}{2} + j\Delta x, \quad y_k = -1 + \frac{\Delta y}{2} + k\Delta y,$$
$$j, k = 0, \ldots, N-1, \quad \Delta x = \Delta y = \frac{2}{N}, \quad N = 2^{K-1} - 1,$$

with $\xi_{j,k} \sim \mathcal{N}(0, 0.1)$. We have $\Delta_x = \Delta_y = 2\Delta x$. Thus, the diameter of the main lobe of the sinc is $4\Delta x$ on the xy-plane. Here, we show a 2D example whose discretization of a smooth function f has a matrix rank of 23 and a TT-rank of 26 when represented in the QTT format. We still use the ranks $R_{max} = 10$ and $\hat{R}_{max} = 15$ for our **max rank TT-SVD** (**max rank TT-RSVD**) and max rank QTT-FFT. Thus, our TT-ranks are much smaller than the true TT-ranks. In Figure 4 and Table 1, notice that our method can still capture the shape of the original function with an error that is an order of magnitude smaller than the error from the true convolution using FFT. The plots on the bottom of Figure 4 are a side view of the error graphs, as it is easier to compare the errors in this view. The 2D examples are similar to the previous test case. Thus, it is reasonable to assume our method works about the same regardless of the spatial dimension.

In Table 4, we compare the run times for the different methods of computing the convolution in two spatial dimensions. We get similar results as the one-dimensional case, where the fastest run time is from the convolution with FFT, but with the **max rank TT-SVD** and **max rank TT-RSVD** methods approaching its run time asymptotically. In 2D, when there is the same amount of data as in the 1D case (for example, $2^{14\times 14}$ in 2D compared to 2^{28} in 1D), the 2D examples do not run as fast as the 1D example. This is due to the extra work in the 2D QTT-FFT algorithm from [3].

TABLE 4. Run times (seconds): Example 3 convolutions.

K	$\boldsymbol{I}_\xi$	$\boldsymbol{I}_{QTT_0}$	$\boldsymbol{I}_{QTT_r}$	$\boldsymbol{I}_\delta$	$\boldsymbol{I}_{RTT}$	$\boldsymbol{I}_{lr}$	$\boldsymbol{I}_{QTT_0}/\boldsymbol{I}_\xi$
8	**0.0034**	0.2213	0.310	$7.102_{\delta=0.04}$	0.297	$0.0090_{rank=2}$	65.088
10	**0.0629**	0.9209	1.428	$0.670_{\delta=0.04}$	1.321	$0.154_{rank=2}$	14.651
12	**0.948**	8.6462	11.258	$22.79_{\delta=0.04}$	10.27	$5.15_{rank=2}$	9.121
14	**58.67**	147.8	151.05	$1087_{\delta=0.04}$	164.5	$286.2_{rank=2}$	2.519

Again we compare the amount of data stored in the full format versus in the QTT-format. Note that the spatial dimension of the original function does not matter in how much storage it takes, just the dimensionality of the data. For example, it takes just as much data to store a vector in $\mathbb{R}^{2^{20}}$ as it does to store a matrix in $\mathbb{R}^{2^{10}\times 2^{10}}$ in the QTT format with a max rank of R_{max}.

8. **Conclusions.** In this paper, we have shown that the QTT decomposition, along with the QTT-FFT algorithm, can effectively remove noise from signals with full TT-ranks when the true signal is of low rank. As we have seen in the numerical examples, we could drastically remove the amount of noise from the signal compared to if we took the convolution in the traditional way of using the FFT algorithm. This comes at the cost of run time, but our methods still run at a reasonable speed

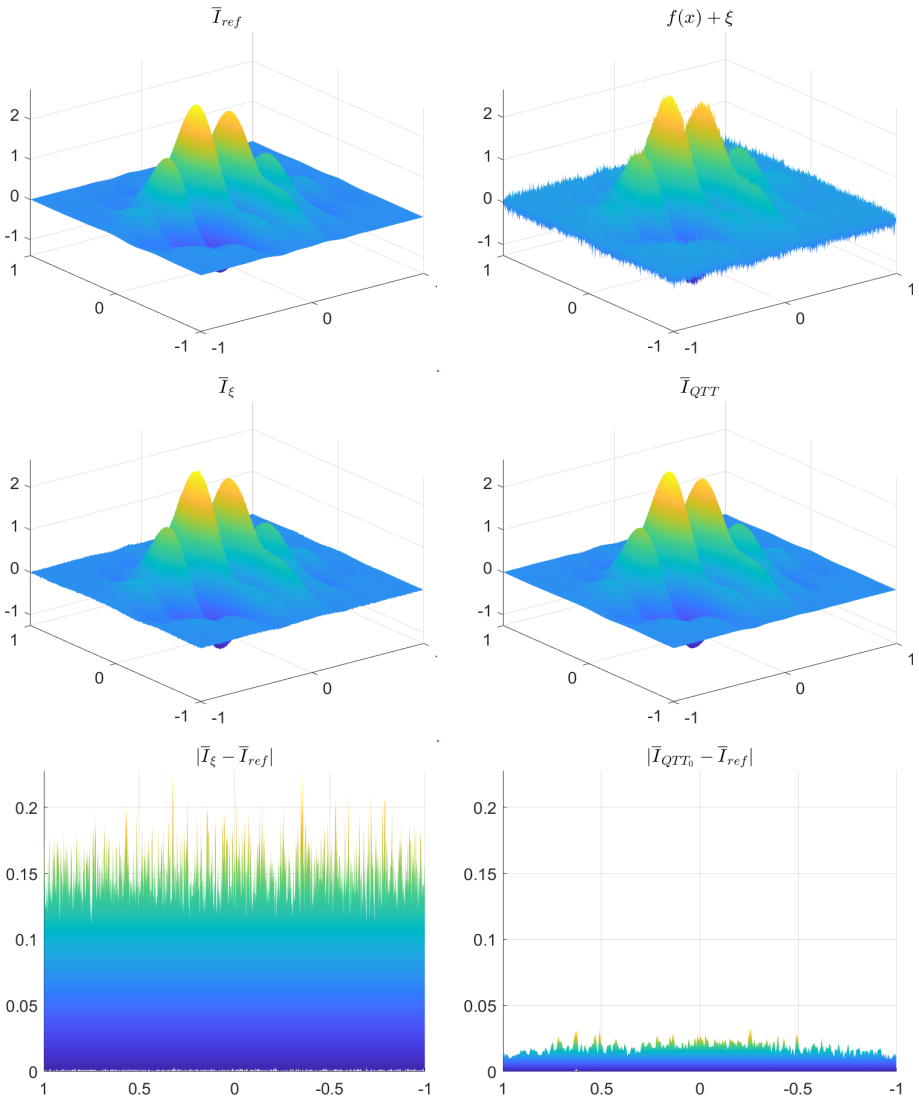

FIGURE 4. *Top Left:* True convolution of data without noise, $\boldsymbol{I}$. *Top Right:* Function data with noise, $\boldsymbol{f}_{\xi}$. *Middle Left:* True convolution of data with noise, $\boldsymbol{I}_{\xi}$. *Middle Right:* Convolution using the **max rank TT-SVD** algorithm, $\boldsymbol{I}_{QTT_0}$. *Bottom Left:* Absolute error of $\boldsymbol{I}_{\xi}$. *Bottom Right:* Absolute error of $\boldsymbol{I}_{QTT_0}$.

which got closer to the FFT run time as the dimensionality of the data increased. This is demonstrated by three different examples, two in one spatial dimension and one in two spatial dimensions. We are even able to show that our method works on very oscillatory data where it is required to have a sinc kernel with a narrow main lobe. Using approximate TT-ranks smaller than the TT-ranks of the actual signal data, we are able to recover most of the signal. This indicates that as long as the signal is reasonably smooth, the QTT decomposition can effectively be used for noise reduction for high-dimensional data, even if the true TT-rank is unknown.

TABLE 5. Data storage for Example 3.

K	$\boldsymbol{f}_\xi$	$\boldsymbol{\mathcal{F}}_\xi$
8	65,536	2088
10	1,048,576	2888
12	16,777,216	3688
14	268,435,456	4488

From our three new approaches, the **max rank TT-SVD** convolution algorithm works much better than the **max rank TT-RSVD** and the **SV drop off TT-SVD** convolution algorithms. As was stated before, the **max rank TT-RSVD** can outperform the **max rank TT-SVD** algorithm when there are larger mode sizes that are used in this paper. For this reason, we present this algorithm, as we have not seen it in the literature elsewhere. The **SV drop off TT-SVD** convolution algorithms do not produce as accurate of a method as the **max rank TT-SVD** or the **max rank TT-RSVD** algorithm; however, in some cases, it does run faster, and this method may give a higher degree of confidence that the truncated singular values are of little importance. Unfortunately, this method can also lead to long run times, as is seen in Table 4 when $K = 14$.

Appendix A. **Randomized SVD.** Here, we give a brief overview of the randomized SVD (RSVD) decomposition from [11]. To compute the RSVD of the matrix $A \in \mathbb{R}^{m\times n}$, the first step is to find $Q \in \mathbb{R}^{m\times(k+p)}$ such that

$$\boldsymbol{A} \approx \boldsymbol{Q}\boldsymbol{Q}^*\boldsymbol{A}$$

where $\boldsymbol{Q}$ has orthonormal columns and whose columns are approximations for the range of $\boldsymbol{A}$. Here, k is the number of singular values that we want in our approximation to be close to the singular values of $\boldsymbol{A}$, and p is what is known as an oversampling parameter. To find $\boldsymbol{Q}$, we use the following Algorithm 4.

Algorithm 4: Solving the Fixed-Rank Problem

input : $\boldsymbol{A}$, k, p
output : $\boldsymbol{Q}$
Draw random matrix $\boldsymbol{\Omega} \in \mathbb{R}^{n\times(k+p)}$ such that $\boldsymbol{\Omega}_{i,j} \sim \mathcal{N}(0,1)$.
Let $\boldsymbol{Y} = \boldsymbol{A}\boldsymbol{\Omega}$.
Compute QR factorization $\boldsymbol{Q}\boldsymbol{R} = \boldsymbol{Y}$.

Once we have obtained $\boldsymbol{Q}$, we can compute the low-rank RSVD using Algorithm 5 (Algorithm 5.1 in [11]).

Algorithm 5: RSVD

input : $\boldsymbol{A}$, $\boldsymbol{Q}$, k
output : $\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^*$
1. Let $\boldsymbol{B} = \boldsymbol{Q}^*\boldsymbol{A}$.
2. Compute SVD: $\tilde{\boldsymbol{U}}\boldsymbol{\Sigma}\boldsymbol{V}^* = \boldsymbol{B}$.
3. Let $\boldsymbol{U} = \boldsymbol{Q}\tilde{\boldsymbol{U}}$.

With these algorithms, we obtain an approximation $\tilde{\boldsymbol{A}}$ to $\boldsymbol{A}$ such that

$$\|\boldsymbol{A} - \tilde{\boldsymbol{A}}\| \le (1 + 11\sqrt{k+p}\sqrt{\min(m,n)})\sigma_{k+1}, \tag{9}$$

with probability $1 - 6p^{-p}$. If we truncate the SVD to only the leading k singular values in Algorithm 5, then the error on the left-hand side of (9) only increases

by at most σ_{k+1}. The computational complexity for each step of this algorithm is given as

- $\mathcal{O}(mn(k+p))$
- $\mathcal{O}((k+p)^2 n)$
- $\mathcal{O}((k+p)^2 m)$,

Thus, for $k+p < \min(m,n)$, the overall algorithm requires $\mathcal{O}(mn(k+p))$ operations.

Acknowledgments. The work of A. Chertock was supported in part by NSF grants DMS-1818684 and DMS-2208438. The work of C. Leonard was supported in part by NSF grant DMS-1818684. The work of S. Tsynkov was supported in part by US Air Force Office of Scientific Research (AFOSR) grant # FA9550-21-1-0086.

REFERENCES

[1] M. Che and Y. Wei, Randomized algorithms for the approximations of Tucker and the tensor train decompositions, *Adv Comput Math*, **45** (2019), 395-428.
[2] M. Cheney and B. Borden, *Fundamentals of Radar Imaging*, vol. 79 of CBMS-NSF Regional Conference Series in Applied Mathematics, SIAM, Philadelphia, 2009.
[3] S. Dolgov, B. Khoromskij and D. Savostyanov, Superfast Fourier transform using QTT approximation, *J. Fourier Anal. Appl.*, **18** (2012), 915-953.
[4] B. P. Epps and E. M. Krivitzky, Singular value decomposition of noisy data: Noise filtering, *Experiments in Fluids*, **60** (2019), 1-23.
[5] K. Fonałand R. Zdunek, Distributed and randomized tensor train decomposition for feature extraction, in *2019 International Joint Conference on Neural Networks (IJCNN)*, 2019, 1-8.
[6] M. Gilman, E. Smith and S. Tsynkov, *Transionospheric Synthetic Aperture Imaging*, Applied and Numerical Harmonic Analysis, Birkhäuser/Springer, Cham, Switzerland, 2017.
[7] M. Gilman and S. Tsynkov, A mathematical perspective on radar interferometry, *Inverse Problems & Imaging*, **16** (2022), 119-152.
[8] X. Gong, W. Chen, J. Chen and B. Ai, Tensor denoising using low-rank tensor train decomposition, *IEEE Signal Processing Letters*, **27** (2020), 1685-1689.
[9] L. Grasedyck, *Polynomial Approximation in Hierarchical Tucker Format by Vector–Tensorization*, Technical Report 308, Institut für Geometrie und Praktische Mathematik RWTH Aachen, 2010.
[10] W. Hackbusch and S. Kühn, A new scheme for the tensor representation, *J. Fourier Anal. Appl.*, **15** (2009), 706-722.
[11] N. Halko, P. G. Martinsson and J. A. Tropp, Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions, *SIAM Rev.*, **53** (2011), 217-288.
[12] R. Harshman, Foundations of the PARAFAC procedure: Models and conditions for an explanatory multimodal factor analysis, *UCLA Working Papers in Phonetics*, **16** (1970), 1-84.
[13] B. Huber, R. Schneider and S. Wolf, A randomized tensor train singular value decomposition, in *Compressed Sensing and its Applications*, Appl. Numer. Harmon. Anal., Birkhäuser/Springer, Cham, 2017, 261-290.
[14] S. K. Jha and R. D. S. Yadava, Denoising by singular value decomposition and its application to electronic nose data processing, *IEEE Sensors Journal*, **11** (2011), 35-44.
[15] V. A. Kazeev, B. N. Khoromskij and E. E. Tyrtyshnikov, Multilevel Toeplitz matrices generated by tensor-structured vectors and convolution with logarithmic complexity, *SIAM J. Sci. Comput.*, **35** (2013), A1511-A1536.
[16] B. N. Khoromskij, $O(d \log N)$-quantics approximation of N-d tensors in high-dimensional numerical modeling, *Constr. Approx.*, **34** (2011), 257-280.
[17] J. Li, X.-P. Zhang and T. Tran, Point cloud denoising based on tensor tucker decomposition, in *2019 IEEE International Conference on Image Processing (ICIP)*, (2019), 4375-4379.
[18] L. Li, W. Yu and K. Batselier, Faster tensor train decomposition for sparse data, *Journal of Computational and Applied Mathematics*, **405** (2022), 113972.
[19] Y. Nomura, K. Yamamoto, S. Anada, T. Hirayama, E. Igaki and K. Saitoh, Denoising of series electron holograms using tensor decomposition, *Microscopy*, **70** (2020), 255-264.

[20] I. V. Oseledets, Approximation of $2^d \times 2^d$ matrices using tensor decomposition, *SIAM J. Matrix Anal. Appl.*, **31** (2009/10), 2130-2145.
[21] I. V. Oseledets, Tensor-train decomposition, *SIAM Journal on Scientific Computing*, **33** (2011), 2295-2317.
[22] I. V. Oseledets, Constructive representation of functions in low-rank tensor formats, *Constr. Approx.*, **37** (2013), 1-18.
[23] I. V. Oseledets and E. E. Tyrtyshnikov, Breaking the curse of dimensionality, or how to use SVD in many dimensions, *SIAM J. Sci. Comput.*, **31** (2009), 3744-3759.
[24] I. Oseledets and E. Tyrtyshnikov, TT-cross approximation for multidimensional arrays, *Linear Algebra Appl.*, **432** (2010), 70-88.
[25] M. V. Rakhuba and I. V. Oseledets, Fast multidimensional convolution in low-rank tensor formats via cross approximation, *SIAM J. Sci. Comput.*, **37** (2015), A565-A582.
[26] T. Shi, M. Ruth and A. Townsend, Parallel algorithms for computing the tensor-train decomposition, 2021, https://arxiv.org/abs/2111.10448.
[27] L. Tucker, Some mathematical notes on three-mode factor analysis, *Psychometrika*, **31** (1966), 279-311.

Received January 2023; revised March 2023; early access March 2023.