A SPARSE FAST CHEBYSHEV TRANSFORM FOR HIGH-DIMENSIONAL APPROXIMATION

DALTON JONES[*], PIERRE-DAVID LETOURNEAU[†], MATTHEW J. MORSE[†], AND M. HARPER LANGSTON[†]

Abstract. We present the Fast Chebyshev Transform (FCT), a fast, randomized algorithm to compute a Chebyshev approximation of functions in high-dimensions from the knowledge of the location of its nonzero Chebyshev coefficients. Rather than sampling a full-resolution Chebyshev grid in each dimension, we randomly sample several grids with varied resolutions and solve a least-squares problem in coefficient space in order to compute a polynomial approximating the function of interest across all grids simultaneously. We theoretically and empirically show that the FCT exhibits quasi-linear scaling and high numerical accuracy on challenging and complex high-dimensional problems. We demonstrate the effectiveness of our approach compared to alternative Chebyshev approximation schemes. In particular, we highlight our algorithm's effectiveness in high dimensions, demonstrating significant speedups over commonly-used alternative techniques.

Key words. Chebyshev polynomial, high dimensional approximation, Discrete Cosine Transform, randomized sampling, least squares, fast algorithms

MSC codes.

1. Introduction. Polynomial interpolation is a fundamental building block of numerical analysis and scientific computing, often serving as the connection between continuous application domains and discrete computational domains. Further, Chebyshev polynomials are a popular basis for interpolation due to their favorable convergence properties for smooth and analytic functions, their numerical stability, their near-best approximation power [28], and a simple FFT-based interpolation procedure [42, 40, 33]. As a result, Chebyshev polynomial interpolation has been used in a variety of domains, such as solving partial differential equations [29, 40, 14], signal processing [35, 26] and optimization [11, 34], and they serve as the foundation for the popular numerical computing toolkit `chebfun` [3].

The standard approach for interpolating a function in D dimensions with a d degree polynomial is to sample the target function on a tensor product grid of $(d+1)^D$ points and solve a linear system for the coefficients of a polynomial $p(\boldsymbol{x})$; i.e.,

$$(1.1) \qquad p(\boldsymbol{x}) = \sum_{\boldsymbol{n=0}}^{\boldsymbol{D}} c_{\boldsymbol{n}} T_{\boldsymbol{n}}(\boldsymbol{x}),$$

where $\boldsymbol{n} = (n_1, \ldots, n_D)$ is a multi-index, $\boldsymbol{x} = (x_1, \ldots, x_D) \in \mathbb{R}^D$, $T_{\boldsymbol{n}}(\boldsymbol{x}) = \prod_{i=1}^{D} T_{n_i}(x_i)$, and $T_n(x)$ is the nth Chebyshev polynomial. However, as the dimension grows, the number of required samples and unknown coefficients grow exponentially, and the problem quickly becomes intractable.

Due to the importance of high-dimensional function approximation, many alternative methods have been developed to overcome this *curse of dimensionality*. A more thorough review of the literature can be found in [41], but we briefly summarize various approaches here. Low-rank approximations [5, 25, 38, 32, 18] try to reduce the problem to computing coefficients of a reduced set of separated functions that still approximate the target function well. The accuracy of low-rank approximations can be improved by increasing the number of terms in the approximation or by applying the method hierarchically on low-rank submatrices of the resulting system matrix [4, 24].

---

[*]Qualcomm AI Research. Qualcomm AI Research is an initiative of Qualcomm Technologies, Inc.

Sparse grids [8, 21, 22] reduce the degrees of freedom by using a reduced tensor-product basis formed from a progressively finer hierarchy of functions. After removing basis elements with mixed derivative terms below a threshold tolerance, the remaining coefficients decay rapidly, allowing for scaling independent of dimension (up to logarithmic factors). Adaptive methods have been proposed [12], leveraging the pseudospectral method combined with sparse grids and Smolyak's algorithm [37] to reach even higher dimensions.

Recently, deep neural networks have demonstrated their effectiveness at approximating and interpolating high-dimensional data [15]. This has become more relevant in light of the "double-descent" phenomenon [31, 6]: test accuracy actually continues to improve even after a network overfits the training data. Some progress has been made to determine the theoretical underpinning of this behavior [2, 43, 7]. Although designed for Gaussian process regression and limited to low dimensions, the most recent work that resembles our proposed approach is [20], though our appraoch is more broadly scoped and applied.

Standard tensor product representations for Chebyshev interpolation in high-dimensions have largely fallen by the wayside due to their exponential complexity. The most relevant improvement to the baseline algorithm, while still leveraging tensor product grids, is a faster DCT algorithm that reduces the constant runtime prefactor [10]. Recently, however, [41, 39] have shown that Chebyshev coefficients decay rapidly when approximating smooth functions, but anisotropically; that is, the normal tensor product grid underresolves the diagonal of the hypercube$[-1, 1]^D$ by a factor of $\sqrt{D}$. Interpolating with a bounded Euclidean degree instead of bounded total degree will address this anisotropy and recover the usual convergence behavior.

Another important observation from [41] is that, regardless of choosing a bounded total or Euclidean degree interpolant, the Chebyshev coefficients above machine precision are contained in a sphere of radius d under the s-norm ($s = 1$ for bounded total degree, $s = 2$ for Euclidean). An obvious consequence of this is that by increasing the radius of this sphere in coefficient-space, we increase the accuracy of the resulting Chebyshev interpolant. In other words, to achieve a given target accuracy, ϵ, we can truncate coefficients outside of this coefficient sphere of radius d_ϵ without losing accuracy. This is algorithmically important because there are exponentially fewer coefficients inside a sphere under the 1-or 2-norm than under the ∞-norm (i.e. bounded maximal degree) [36, 39].

In this paper, we describe an algorithm, the Fast Chebyshev Transform (FCT), whose runtime is proportional to the number of coefficients in this coefficient-space s-norm sphere rather than the size of a D-dimensional tensor-product grid. To do this, we estimate the number of significant coefficients N and sample L distinct tensor product Chebyshev grids with random resolutions in each dimension, such that the product of the resolutions roughly equals N. We then sample the target function at all of these grid points, perform separate D-dimensional DCTs on each grid, and solve a least-squares problem in coefficient space with the conjugate gradient (CG) method. Since the system matrix is well-conditioned, CG converges rapidly. More importantly, we can apply each matrix-vector multiplication in $O(N)$ time due to the sparsity pattern induced by an aliasing identity derived from Chebyshev polynomials. The FCT algorithm has an overall complexity of $O(N \log N)$. We limit our scope to Chebyshev approximation in this work, but FCT can be applied directly to high dimensional cosine approximation of smooth functions with minimal modifications.

The remainder of this paper is structured as follows: In Section 2, we discuss relevant notation and background information. In Section 3, we discuss the mathe-

matical formulation that underpins the Fast Chebyshev Transform and present the full algorithm. Subsection 3.4 discusses the algorithmic complexity and numerical behavior of FCT. Numerical results and comparisons are presented in Section 4. We conclude with Section 5.

2. Background and notation. We aim to approximate a given smooth function $f : \mathbb{R}^D \to \mathbb{R}$ by a tensor product Chebyshev polynomial in the form of Equation (1.1). Bold script will represent vector quantities, such as the real vector $\boldsymbol{x} = (x_1, \dots, x_D)$ and integer multi-indices $\boldsymbol{n} = (n_1, \dots, n_D)$. The nth Chebyshev polynomial is given by

$$(2.1) \qquad\qquad T_n(x) = \cos(n \arccos(x)).$$

Chebyshev polynomials are orthonormal with respect to measure $\frac{1}{\sqrt{1-x^2}}$, $x \in [-1, 1]$:

$$(2.2) \qquad\qquad \int_{-1}^{1} \frac{T_m(x)\, T_n(x)}{\sqrt{1 - x^2}} dx = \begin{cases} 0 & \text{if } n \neq m \\ 1 & \text{if } n = m \end{cases}$$

and obey the recurrence relation

$$(2.3) \qquad\qquad T_{n+1} = 2x T_n(x) - T_{n-1}(x).$$

A multi-dimensional Chebyshev polynomial defined on $\mathbb{R}^D$ is a tensor product of one dimensional Chebyshev polynomials:

$$(2.4) \qquad\qquad T_{\boldsymbol{n}}(\boldsymbol{x}) = \prod_{i=0}^{D} T_{n_i}(x_i).$$

When approximating a function in D dimensions by polynomials, we often consider the class of polynomials with bounded degree d with respect to some norm as approximation candidates; i.e. members of the set

$$(2.5) \qquad\qquad \mathbb{P}_{d,s}^{D} = \left\{ p(\boldsymbol{x}) = \sum_{\boldsymbol{n}} c_{\boldsymbol{n}} T_{\boldsymbol{n}}(\boldsymbol{x}) \in \mathbb{P}_d^D \;\middle|\; \|\boldsymbol{n}\|_s \leq d \right\},$$

where $\mathbb{P}_d^D$ is the space of polynomials $p : \mathbb{R}^D \to \mathbb{R}$ with maximum degree d, and $\|\boldsymbol{n}\|_s$ is the s-norm of the vector $\boldsymbol{n}$. We will refer to $\mathbb{P}_{d,1}^D$, $\mathbb{P}_{d,2}^D$ and $\mathbb{P}_d^D$, as polynomials with bounded total degree, Euclidean degree and maximum degree respectively.

Increasing the degree of a polynomial approximation of f increases the approximation accuracy [42]. This means that to approximate f by a polynomial p with accuracy ϵ, there is a degree $d = d_{\epsilon,s}$ such that $\|f - p\| \leq \epsilon$ if p is in $\mathbb{P}_{d,s}^D$. Choosing d determines the total number of coefficients of basis elements with bounded exponents in the s-norm, i.e. $\|\boldsymbol{n}\|_s \leq d$, which define p. We denote this set of indices by

$$(2.6) \qquad\qquad \mathcal{S}_{d,D}^s = \{\boldsymbol{n} \in \mathbb{N}^D \mid \|\boldsymbol{n}\|_s \leq d\}.$$

Known results tell us that

$$(2.7) \qquad\qquad |\mathcal{S}_{d,D}^1| = \binom{d + D}{d} \quad [30]$$

$$(2.8) \qquad\qquad |\mathcal{S}_{d,D}^2| = \frac{(\sqrt{\pi} d / 2)^D}{\Gamma\left(\frac{D}{2} + 1\right)}, \quad D \to \infty \quad [36]$$

$$(2.9) \qquad\qquad |\mathcal{S}_{d,D}^\infty| = d^D.$$

3

The important aspect of Equations (2.7) and (2.8) is that they grow much slower than Equation (2.9) as D increases. This makes them more efficient to work with directly than Equation (2.9) while still approximating f well. These values are used to discuss the computational complexity of this work; we typically choose $N = |\mathcal{S}_{d,D}^1|$ or $|\mathcal{S}_{d,D}^2|$.

Problem setup. One way to approximate a smooth function f is to sample it on a D-dimensional tensor product of 1D Chebyshev grids:

$$(2.10) \qquad x_k = \cos(\theta_k), \quad \theta_k = \frac{\left(k + \frac{1}{2}\right)\pi}{N+1}, \quad k = 0, \dots, N.$$

Then, apply a DCT in each dimension. This produces a polynomial interpolant of the function samples; since Chebyshev polynomials obey the following relation after a change of variables from $[-1, 1]$ to $[0, \pi]$

$$(2.11) \qquad T_n(x_k) = T_n(\cos(\theta_k)) = \cos(n\theta_k),$$

Equation (1.1) becomes a multi-dimensional cosine series. However, as mentioned above, the size of the tensor-product grid grows exponentially with D, limiting this approach's practicality to low dimensions.

Another common approach is to approximate f with a least-squares solution: find p that minimizes $\|f - p\|_2^2$. The standard way to solve this problem is sample f at a set of points, then form a matrix B with entries

$$(2.12) \qquad B_{i,j} = T_{\boldsymbol{n}^{(j)}}\left(\boldsymbol{x}^{(i)}\right),$$

where $\boldsymbol{n}^{(j)}$ and $\boldsymbol{x}^{(i)}$ imply an ordering on the multi-indices and sample points. After constructing B, we solve the minimization problem,

$$(2.13) \qquad \min_{\boldsymbol{c}} \|B\boldsymbol{c} - \boldsymbol{f}\|_2^2,$$

where $\boldsymbol{c}_i = c_{\boldsymbol{n}^{(i)}}$ and $\boldsymbol{f}_i = f(\boldsymbol{x}^{(i)})$. This can solved many ways, either with a direct approach, such as solving the normal equations, or by using the singular value decomposition (SVD), or with an iterative solver like the conjugate gradient method (CG) [27].

The advantage of a least-squares-based approach is that the complexity is in terms of the number of unknown coefficients N of Equations (2.7) and (2.8), rather than an exponentially growing grid size. A disadvantage of this approach is the implicit ambiguity in choosing sample point locations: poorly sampling f can produce a bad approximation or slow convergence, while provable convergence often requires exponentially many more samples [13]. Another drawback is the relatively large complexity; direct methods require $O(N^3)$ work, while the CG method still requires $O(N^2)$ dense matrix-vector multiplications.

3. Algorithm. To address these shortcomings, we propose an alternative approach: the Fast Chebyshev Transform (FCT). The core of the FCT algorithm is a consistent linear system of the form,

$$(3.1) \qquad \begin{bmatrix} F^{(1)} & 0 & \cdots & 0 \\ 0 & F^{(2)} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & F^{(L)} \end{bmatrix} \begin{bmatrix} \boldsymbol{f}^{(1)} \\ \boldsymbol{f}^{(2)} \\ \vdots \\ \boldsymbol{f}^{(L)} \end{bmatrix} = \begin{bmatrix} A^{(1)} \\ A^{(2)} \\ \vdots \\ A^{(L)} \end{bmatrix} \boldsymbol{c}$$

$$(3.2) \qquad \mathcal{F}\boldsymbol{f} = A\boldsymbol{c}.$$

Each matrix $A^{(\ell)}$ is a sparse $O(N) \times N$ matrix, which are concatenated into a large $O(NL) \times N$ matrix with L sparse blocks. The block matrices $F^{(\ell)}$ are $O(N) \times O(N)$ matrices corresponding to DCTs, $\boldsymbol{c} = [c_{\boldsymbol{n}^{(1)}}, \ldots, c_{\boldsymbol{n}^{(N)}}]^T$ is the vector containing non-zero coefficients of f's polynomial approximation, and $[\boldsymbol{f}^{(1)}, \ldots, \boldsymbol{f}^{(L)}]^T$ is a vector of f at sample points $\boldsymbol{x}^{(i)}$. To determine the sampling pattern of f and the sparsity pattern of A, we construct a collection of L distinct tensor product grids, each containing randomly selected resolutions in each dimension, but whose total number of points is roughly N. We call this collection of grids an L-*grid*; an example of one possible L-grid for $N = 90$ in three dimensions is shown in Figure 1-Left.
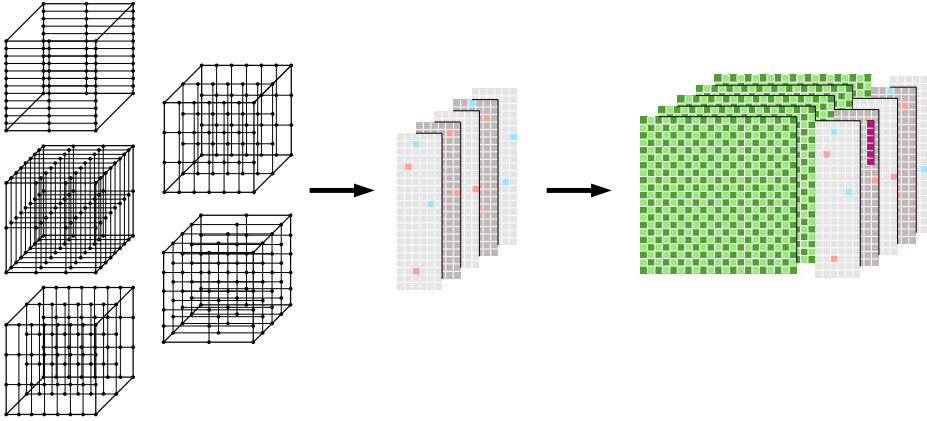

Fig. 1: Schematic of the matrix-vector multiplication step in the sparse Fast Chebyshev Transform. *Left*: First, we sample the high-dimensional input function on a collection of L grids with varied, randomly chosen discretizations in each dimension, the product of each of which roughly equals the estimated number of non-zero coefficients N. *Middle*: Next, for each sampled grid, we compute the non-zero entries (red and blue entries) of the linear system resulting from an interpolation problem on the grid. This is sparse due to aliasing effects of Chebyshev polynomials (see Lemma 3.1 and [42, Chapter 4]). *Right*: Finally, we multiply each sparse matrix with the coefficient vector (magenta) and apply a Discrete Cosine Transform (green matrix) to the result. To solve for unknown polynomial coefficients, we solve a sparse least-squares using an iterative method such as the Calgorithm.

Due to the aliasing properties of Chebyshev polynomials (Lemma 3.1, [42, Chapter 4]), the matrices $A^{(l)}$ are sparse and contain $O(N)$ non-zero entries. These properties, along with fast DCT algorithms [9, 1], allow us to compute both sides of Equation (3.1) with $O(LN \log(N))$ time complexity. When L is well-chosen, the matrix A has favorable conditioning properties, allowing an iterative solver such as the CG method [27], to converge rapidly to the least-squares solution. A schematic of the algorithm is shown in Figure 1.

In the remainder of this section, we first highlight the aliasing properties of Chebyshev polynomials on L-grids and how this leads to the sparsity of A in Subsection 3.1. In Subsection 3.3, we describe how to appropriately choose an L-grid. We combine each of these pieces in Subsection 3.4 to describe the full FCT algorithm.

3.1. Aliasing. Chebyshev polynomials evaluated on a Chebyshev grid exhibit a special property known as *aliasing*: a higher degree Chebyshev polynomial realizes the same values as a single lower degree Chebyshev polynomial on a fixed grid size

[42, Chapter 4]. When approximating functions with Chebyshev polynomials, this aliasing has another side effect: if a low degree Chebyshev polynomial is zero on a given Chebyshev grid, certain higher degree Chebyshev polynomials will also be zero. We leverage this to produce a sparse system matrix in Equation (3.1).

Aliasing in 1D. To highlight this relationship, consider the one-dimensional problem of interpolating a smooth function f by a degree N Chebyshev polynomial on $[-1, 1]$. We know that f can be expressed as a unique Chebyshev series on $[-1, 1]$:

$$(3.3) \qquad f(x) = \sum_{n=0}^{\infty} c_n T_n(x).$$

To approximate this, we want to find a set of M coefficients c_i defining a polynomial $p(x)$ such that

$$(3.4) \quad f(x) \approx p(x) = \sum_{n=0}^{M} c_n T_n(x), \quad x \in [-1, 1], \quad c_n = \frac{2 - \delta_0(n)}{\pi} \int_{-1}^{1} \frac{T_n(x) f(x)}{\sqrt{1 - x^2}} dx,$$

where $\delta_0(n)$ is the dirac delta function centered at 0. Specifically, we would like to approximate f by a truncated Chebyshev series. After applying a change of variables $x = \cos(\theta)$ from $[-1, 1]$ into $[0, \pi]$, the interpolation problem becomes

$$(3.5) \qquad f(\cos(\theta)) \approx p(\cos(\theta)) = \sum_{n=0}^{M} c_n \cos(n\theta), \quad \theta \in [0, \pi],$$

$$(3.6) \qquad c_n = \int_0^{\pi} \cos(n\theta) f(\cos(\theta)) d\theta.$$

Substituting the polynomial approximation of f into the expression for the nth Chebyshev coefficient in Equation (3.6), we obtain

$$(3.7) \qquad c_n = \int_0^{\pi} \cos(n\theta) \left(\sum_{m=0}^{M} c_m \cos(m\theta) \right) d\theta$$

$$(3.8) \qquad = \sum_{m=0}^{M} c_m \int_0^{\pi} \cos(n\theta) \cos(m\theta) d\theta$$

for $n \leq m$, thanks to the orthogonality of cosines over $[0, \pi]$. If we discretize the integral in Equation (3.8) using $N + 1$ equispaced points in $[0, \pi]$ (corresponding to a first-kind Chebyshev grid of $N + 1$ points in Equation (2.10)), the quadrature becomes
(3.9)
$$b_{n,N} := \sum_{m=0}^{M} c_m \left(\frac{1}{N+1} \sum_{k=0}^{N} \cos(n\theta_k) \cos(m\theta_k) \right) = \frac{1}{N+1} \sum_{k=0}^{N} \cos(n\theta_k) \, p\left(\cos(m\theta_k)\right).$$

In particular, note that if $N \geq M$, the trapezoidal rule is exact for the $(M+1)$-term cosine series in Equation (3.8) and $b_{n,N} = c_n$. However, this is not the case when $N \leq M$ due to *aliasing*. Such aliasing is quantified by the following lemma:

LEMMA 3.1. *For every positive integer N,*

$$(3.10) \qquad \frac{1}{N+1} \sum_{k=0}^{N} \cos\left(\frac{n(k+1/2)\pi}{N+1} \right) \cos\left(\frac{m(k+1/2)\pi}{N+1} \right)$$

$$(3.11) \qquad = \frac{1}{2}\Delta(m+n) + \frac{1}{2}\Delta(m-n),$$

where

$$(3.12) \qquad \Delta_N(\ell) = \begin{cases} 1 & \text{if } \ell \bmod (4(N+1)) = 0 \\ -1 & \text{if } \ell \bmod (4(N+1)) = 2(N+1) \\ 0 & o.w. \end{cases}$$

Substituting Equation (3.12) into Equation (3.9):

$$(3.13) \qquad b_{n,N} = \sum_{m=0}^{M} c_m \left(\frac{1}{2} \Delta_N(m+n) + \frac{1}{2} \Delta_N(m-n) \right)$$

$$(3.14) \qquad = \frac{1}{N+1} \sum_{k=0}^{N} \cos(n\theta_k) \, p \left(\cos(m\theta_k) \right).$$

$$(3.15)$$

This equation establishes a relationship between the aliased coefficients, the polynomial coefficients, and the polynomial values, which can be expressed as

$$(3.16) \qquad \boldsymbol{b}_N = A\,\boldsymbol{c} = F\,\boldsymbol{p},$$

where $\boldsymbol{c}$ is the vector of Chebyshev coefficients of the polynomial, $\boldsymbol{p}$ contains polynomial values samples on a regular grid, F corresponds to a size-$(N+1)$ Discrete Cosine Transform (DCT), and A is sparse with entries given by

$$(3.17) \qquad A_{n,m} = \frac{1}{2} \Delta_N(m+n) + \frac{1}{2} \Delta_N(m-n).$$

Higher dimensions. To see how this phenomenon impacts higher dimensions, we generalize the above relations from scalars in $[-1,1]$ to vectors $[-1,1]^D$. Chebyshev interpolation of a function $f : [-1,1]^D \to \mathbb{R}$ involves tensor-products of Chebyshev polynomials and replacing the single index n with multi-index $\boldsymbol{n} = (n_1, \ldots, n_D) \in \mathbb{N}^D$:

$$f(\boldsymbol{x}) \approx p(\boldsymbol{x}) = \sum_{\boldsymbol{n} \in \mathcal{N}} c_{\boldsymbol{n}} T_{\boldsymbol{n}}(\boldsymbol{x}), \quad \boldsymbol{x} \in [-1,1]^D,$$

$$c_{\boldsymbol{n}} = \frac{2 - \delta_0(n)}{\pi} \int_{[-1,1]^D} \frac{T_{\boldsymbol{n}}(\boldsymbol{x}) f(\boldsymbol{x})}{\sqrt{1 - \boldsymbol{x}^2}} d\boldsymbol{x},$$

where $T_{\boldsymbol{n}}(\boldsymbol{x}) = T_{n_1}(x_1) T_{n_2}(x_2) \ldots T_{n_d}(x_D)$, and $\mathcal{N}$ consists of the set of the multi-indices of the non-zero coefficients of the interpolant $p(\cdot)$. In general, for interpolation purposes, $\mathcal{N}$ corresponds to $\mathcal{S}_{d,D}^1$ or $\mathcal{S}_{d,D}^2$ (see Section 2).

We apply the same change of variables in each dimension as in the one-dimensional case, producing

$$(3.18) \qquad p(\cos(\theta_1), \ldots, \cos(\theta_D))) = \sum_{\boldsymbol{n} \in \mathcal{N}} c_{\boldsymbol{n}} \cos(\boldsymbol{n}\boldsymbol{\theta}), \quad \boldsymbol{\theta} \in [0, \pi]^D,$$

$$(3.19) \qquad c_{\boldsymbol{n}} = \int_{[0,\pi]^D} \cos(\boldsymbol{n}\boldsymbol{\theta}) f(\cos(\theta_1), \ldots, \cos(\theta_D)) d\boldsymbol{\theta}$$

with $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_D) \in [0, \pi]^D$ and $\cos(\boldsymbol{n}\boldsymbol{\theta}) = \cos(n_1\theta_1) \cos(n_2\theta_2) \ldots \cos(n_D\theta_D)$. Combining the two parts of Equation (3.19), we obtain

$$(3.20) \qquad c_{\boldsymbol{n}} = \sum_{\boldsymbol{m} \in \mathcal{N}} c_{\boldsymbol{m}} \prod_{i=0}^{D} \int_0^{\pi} \cos(n_i\theta_i) \cos(m_i\theta_i) d\theta_i.$$

7

Discretizing the integral in Equation (3.20), using $N_i + 1$ points in each dimension i, provides the multi-dimensional analogue of Equation (3.9):

(3.21)
$$b_{\boldsymbol{n},\boldsymbol{N}} = \sum_{\boldsymbol{m}\in\mathcal{N}} c_{\boldsymbol{m}} \prod_{i=0}^{D} \left(\frac{1}{N_i} \sum_{k_i=0}^{N_i} \cos(n_i\,\theta_{i,k_i}) \cos(m_i\,\theta_{i,k_i}) \right)$$

(3.22)
$$= \sum_{k_1=0}^{N_1} \cdots \sum_{k_D=0}^{N_D} \left(\frac{1}{\prod_{i=1}^{D} N_i} \prod_{i=0}^{D} \cos(n_i\theta_{i,k_i}) \right) p\left(\cos(\theta_{1,k_1}),\, ...,\, \cos(\theta_{D,k_D})\right).$$

In particular, note that if we sample on a grid where $N_i \geq \max_{n\in\mathcal{N}}(\max_i n_i)$ as is done traditionally, then $b_{\boldsymbol{n},\boldsymbol{N}} = c_{\boldsymbol{n}}$. However, as the number of samples increases exponentially in higher dimensions, this traditional approach becomes prohibitively expensive computationally. To alleviate this issue, we perform a form of under-sampling, leveraging the aliasing result of Lemma 3.1; i.e., we use grids with N_i points per dimension i, chosen in such a way that the total size $\prod_{i=1}^{D} N_i$ is roughly equal to $|\mathcal{N}|$ (see Subsection 3.3 for details). Following Lemma 3.1, this results in a linear system of the form

(3.23)
$$b_{\boldsymbol{n},\boldsymbol{N}} = \sum_{\boldsymbol{m}\in\mathcal{N}} c_{\boldsymbol{m}} \prod_{i=0}^{D} \left(\frac{1}{2}\Delta_{N_i}(m_i + n_i) + \frac{1}{2}\Delta_{N_i}(m_i - n_i) \right)$$

(3.24)
$$= \sum_{k_1=0}^{N_1} \cdots \sum_{k_D=0}^{N_D} \left(\prod_{i=0}^{D} \cos(n_i\theta_{i,k_i}) \right) p\left(\cos(\theta_{1,k_1}),\, ...,\, \cos(\theta_{D,k_D})\right),$$

which may be written as

(3.25)
$$b_{\boldsymbol{N}} = A_{\mathcal{N},\boldsymbol{N}} c = F_{\boldsymbol{N}} p_{\boldsymbol{N}}.$$

Here, $\boldsymbol{c}$ is the vector of Chebyshev coefficients of the polynomial with multi-index corresponding to elements of $\mathcal{N}$, $\boldsymbol{p}_{\boldsymbol{N}}$ contains polynomial values from the non-uniform grid with $N_i + 1$ samples per dimension, $F_{\boldsymbol{N}}$ corresponds to a non-uniform Discrete Cosine Transform (DCT) with $N_i + 1$ samples per dimension, and $A_{\mathcal{N},\boldsymbol{N}}$ is a size-$O(|\mathcal{N}|) \times |\mathcal{N}|$ matrix with rows corresponding to elements of the non-uniform grid and columns corresponding to elements of the set $\mathcal{N}$ with entries

(3.26)
$$A_{\boldsymbol{n},\boldsymbol{m}} = \prod_{i=1}^{D} A_{n_i,m_i}^{(i)} = \prod_{i=0}^{D} \left(\frac{1}{2}\Delta_{N_i}(m_i + n_i) + \frac{1}{2}\Delta_{N_i}(m_i - n_i) \right).$$

In particular, the size of this system is *linear* in the number of Chebyshev coefficients ($|\mathcal{N}|$), which leads to computational efficiency. Further, the matrix A can be computed efficiently by considering the products of non-zero elements of lower-dimensional sparse aliasing matrices only.

3.2. L-grids. With the system matrix on a single tensor product grids defined, we can apply the same construction to a collection of different grids. Let $\bar{\boldsymbol{N}}^{(\ell)} = (N_1^{(\ell)}, \ldots, N_D^{(\ell)})$ for $\ell = 1, \ldots, L$ be the ℓth choice of tensor-product discretization. The L discretizations resulting from discretizing each dimension via Equation (2.10) for each ℓ is called an *L-grid*, which is completely determined by the sampling rates $\bar{\boldsymbol{N}}^{(\ell)}$.

For each grid within a given L-grid, we compute the matrix $A^{(\ell)}$ using Equation (3.26) and concatenate them into a final full matrix A:

$$(3.27) \qquad A = \begin{bmatrix} A^{(1)} \\ A^{(2)} \\ \vdots \\ A^{(L)} \end{bmatrix}$$

Combining Equation (3.27) with Equation (3.25), we obtain the expression in Equation (3.1). From this, the FCT computes the DCT on the left side of Equation (3.1) using the function samples on the entire L-grid, followed by solving a least-squares problem for the unknown polynomial coefficients $\boldsymbol{c}$.

3.3. Sampling and Matrix Construction. With the system in Equation (3.1), we need to determine an L-grid, along with an appropriate value of L, such that Equation (3.1) is well-conditioned and invertible.

To ensure invertibility, we adopt a randomized approach by randomly assigning sampling rates along each dimension i in such a way that

$$(3.28) \qquad \prod_{i=1}^{D} N_i^{(\ell)} \approx N, \quad \ell = 1, \dots L$$

as detailed in Algorithm 3.1.

With our randomly selected $\bar{\boldsymbol{N}}$, we sample f on the associated L-grid and form the matrix A in Equation (3.1). The goal for a sufficiently large value of L is to sample enough points to produce a well-conditioned least-squares system to recover the Chebyshev coefficients. To determine L, we iteratively sample a new random grid, construct the corresponding matrix $A^{(\ell)}$, and append it to the matrix computed for the previous $\ell - 1$ grids. This process terminates when A is full-rank with a condition number lower than some provided bound κ; we typically choose $\kappa < 10^4$. This process is shown in Algorithm 3.2.

Algorithm 3.1 Sampling Rate Selection

1: INPUT: N (number of nonzero coefficients), D (problem dimension), d (degree)
2: OUTPUT: $(N_1, ..., N_D)$ (tuple of dimension-wise sampling rates)
3: Compute random permutation $(i_1, ..., i_D)$ of tuple $(1, ..., D)$.
4: **for** $j = 1, ..., D$ **do**
5:    Pick N_{i_k} uniformly at random in $\{1, 2, ..., d + 1\}$.
6:    **if** $\prod_{k=1}^{j} N_{i_j} > N$ **then**
7:      Set $N_{i_k} = 1$ for $k = j + 1, ..., D$.
8:      break
9:    **end if**
10: **end for**
11: RETURN: $(N_1, ..., N_D)$

We have performed a variety of numerical experiments with the sampling scheme of Algorithm 3.1. Empirically, choosing $L = cD$ for some $c \geq 2$ is sufficient to guarantee invertibility and well-conditioning of the matrix A of Equation (3.1).

Algorithm 3.2 Construction of Aliasing Matrix A

1: INPUT: $\mathcal{N}$ (list of multi-indices of non-zero coefficients), κ_0 (maximum condition number)
2: OUTPUT: Matrix A of Equation (3.1) (full-rank), number of sampled grids L , discretization pattern of L-grid $\mathfrak{N}$
3: Let $N = |\mathcal{N}|$, $A = \emptyset$, $\mathfrak{N} = \emptyset$, $L = 0$.
4: **while** $\kappa(A) > \kappa_0$ **do**
5:     $L \leftarrow L + 1$
6:     Compute $(N_1^{(L)}, ..., N_D^{(L)})$ using Algorithm 3.1
7:     Compute $A^{(L)}$ using aliasing identity in Equation (3.12) and $\mathcal{N}$
8:     $A \leftarrow [A; A^{(L)}]$
9:     $\mathfrak{N} \leftarrow [\mathfrak{N}, (N_1^{(L)}, ..., N_D^{(L)})]$
10: **end while**
11: RETURN: A, L, $\mathfrak{N}$

Precomputation of L-grids. Since Algorithm 3.1 depends only on $\mathcal{N}$ and is independent of the sample values, we can precompute and store the sampling rates $\bar{N}$ of an appropriate L-grid as well as the matrix A for various values of N, D, d and a given set of multi-indices $\mathcal{N}$ (e.g., $\mathcal{S}_{d,D}^1$ or $\mathcal{S}_{d,D}^2$). This makes Algorithm 3.2 a precomputation with no runtime cost in practical applications.

3.4. Full algorithm. We now give a description of the full FCT algorithm. Given a target function $f(\cdot)$, we first randomly sample an L-grid and construct the corresponding sparse matrix A using Algorithm 3.2 and Algorithm 3.1. Then, we sample the function $f(\cdot)$ at the discretization points on each grid in the chosen L-grid. We denote the vector containing these samples on a single grid $\boldsymbol{f}^{(\ell)}$ and the stacked vector of all values $\boldsymbol{f} = [\boldsymbol{f}^{(1)}; \ldots; \boldsymbol{f}^{(L)}]$. We then form the linear system in Equation (3.1) and solve for the Chebyshev coefficients $\boldsymbol{c}$ via least-squares:

$$(3.29) \qquad \boldsymbol{c}^* = \mathrm{argmin}_{\boldsymbol{c}} ||(F\boldsymbol{f} - A\boldsymbol{c})||_2^2.$$

We solve this least-squares problem with the CG method. Since we can efficiently compute Type-II DCTs with the Fast Fourier Transform (FFT) and can efficiently compute matrix-vector products $A\boldsymbol{c}$ due to the sparsity of A, we can solve this problem rapidly. Combined with the favorable conditioning of A, our approach allows the CG method to converge rapidly to a solution. The scheme is summarized in Algorithm 3.3.

3.5. Scaling & Complexity.
Precomputation complexity. The primary work of the precomputation phase of the FCT involves forming the matrix A using Algorithm 3.2. Computing the sampling rate for a given block of A, which calls Algorithm 3.1, requires $O(D)$ time to complete. For a given sampling rate $(N_1^{(\ell)}, N_2^{(\ell)}, \ldots, N_D^{(\ell)})$, we then compute the non-zero elements of $A^{(\ell)}$ by using the nonzeros of one dimensional aliasing matrices as in Equation (3.26); since each one dimensional aliasing matrix has $O(1)$ zeros per dimension per column, computing all of the zeros results in $O(ND)$ computational complexity. Further, because the loop in Algorithm 3.2 repeats L times, we can construct A in $O(NDL)$ time (the construction of the ℓth aliasing matrix dominates the cost of each loop iteration. It is also worth noting that, as the matrix A is sparse, the memory complexity is $O(NL)$.

Algorithm 3.3 Sparse Fast Chebyshev Transform

1: Input: target function f, degree d, dimension D, multi-index norm s
2: Output: Chebyshev coefficients $\boldsymbol{c}^*$ approximating f
3: $\kappa_0 \leftarrow 10^4$
4: $A, L, \mathfrak{N} \leftarrow$ Algorithm 3.2($\mathcal{S}_{d,D}^s$, κ_0)
5: **for** $\ell = 1, \ldots, L$ **do**
6: $\quad \boldsymbol{f}^{(\ell)} \leftarrow$ Sample f at $\mathfrak{N}_\ell$
7: $\quad \mathcal{F}^{(\ell)} = F^{(\ell)} \boldsymbol{f}^{(\ell)}$ (apply Type-II DCTs to $\boldsymbol{f}^{(\ell)}$)
8: **end for**
9: $\mathcal{F} = [\mathcal{F}^{(1)}; \ldots; \mathcal{F}^{(L)}]$
10: Solve least-squares problem: $\boldsymbol{c}^* = \text{argmin}_{\boldsymbol{c}} ||(\mathcal{F} - A\boldsymbol{c})||_2^2$ using CG.
11: **return** $\boldsymbol{c}^*$

FCT complexity. There are two main steps in Algorithm 3.3: (1) the Type-II DCTs applied to the function samples and (2) solving the least-squares problem with the CG method. The adjoint Type-II DCTs applied on on L different D-dimensional grids (containing $O(N)$) points can be computed in $O(NL\log(N))$ time using FFTs [1]. The least-squares problem can be solved with the CG method in n_{CG} iterations with each iteration requiring a sparse matrix-vector multiply in $O(NL)$ time, resulting in an overall computational complexity of $O(n_{CG}NL)$. Note that if the condition number of A is not too large in value, resulting in a *well-conditioned* system, n_{CG} is small [17, Chapter 10.2], and the CG method converges rapidly. We discuss this further in the following section, but in the case of a well-conditioned A,the complexity of FCT is $O(NL\log(N))$.

In summary, the precomputation phase computational complexity is $O(NDL)$, while applying FCT to f results in $O(NL\log(N))$ complexity.

4. Numerical Results. In this section, we show results in terms of scalability and accuracy for FCT. We begin with a brief background introduction on numerical approximation and error in Subsection 4.1, followed by a discussion on the methods used for comparison in Subsection 4.2 as well as an introduction to the different types of functions studied for scaling and accuracy in Subsection 4.3. We briefly note some details on the numerical implementation in Subsection 4.4. We complete our numerical discussions by showing the specific experiment results on scaling and accuracy in Subsection 4.5 for FCT against the methods in Subsection 4.2 on the functions in Subsection 4.3.

4.1. Numerical Approximation and Error Discussion. The error introduced from the polynomial approximation of a smooth function f evaluated at Chebyshev points is a well-studied problem [42]. On a tensor product Chebyshev grid in high dimensions, if f is analytic in an Bernstein ρ-hyperellipse containing $[-1,1]^D$, then an interpolating polynomial of Euclidean degree k will converge geometrically with rate $O(\rho^{-k})$ [39]. In the context of FCT, we are solving a least-squares problem for an approximating polynomial to f on an L-grid instead of interpolating f on a single D-dimensional grid. The quadrature rule used in Equation (3.22) to compute $c_{\boldsymbol{n}}$ in Equation (3.20) is exact due to the orthogonality of cosines; however, when f is underresolved in a certain dimension as a result of the randomized grid sub sampling, greater care must be taken. Suppose f is a polynomial with bounded total degree k, but whose leading order term has a multi-index $\boldsymbol{k} = (k_1, k_2, \ldots, k_D)$ with $\sum_{i=1}^{D} k_i \leq N$. We have observed empirically that as long as L scales on the order $O(D + d)$, where

D is the dimension of the ambient space and d is the maximum degree we wish to interpolate, the resulting aliasing matrix A has full column rank with high probability; indeed, experiments show this scaling for L holds for all numerical results considered below in the following sections. In this case, we can obtain an an accurate Chebyshev expansion approximating f using only a constrained set of polynomials in our basis from $\mathcal{N}$.

With this background going forward, we next discuss the methods we employ for evaluating the accuracy and scaling benefits of FCT in Subsection 4.2.

4.2. Methods for Numerical Comparison. We compare FCT with two standard approaches for performing Chebyshev interpolation: (1) the baseline DCT-based interpolation approach and (2) a least-squares approach based on a randomized interpolation matrix, which we call Randomized Least-Squares Interpolation (RLSI) with the following specific details:

- **DCT approach**: we sample the target function f on a D-dimensional tensor product Chebyshev grid and apply the DCT to compute the Chebyshev coefficients. This method has a complexity of $O(Dd^D \log d)$.
- **RLSI approach**: consider the general least squares approach as introduced in Section 2. For a Randomized Least Squares (RLSI) approach, we construct the matrix B in Equation (2.12) to solve Equation (2.13) by uniformly sampling CN points from a full uniform tensor-product grid of size d^D (without explicitly forming it). We further choose $C = 1.2$ to ensure that B is invertible with bounded condition number less than $\kappa_{\mathrm{RLSI}} = 10^4$; this mirrors the conditions imposed in Algorithm 3.2 for a fair comparison. The resulting least-squares problem is solved numerically using CG, available through the `Eigen` [23]. This method has complexity $O(|\mathcal{N}|^2)$.

Note that for consistency, we use a tolerance of 10^{-3} for both RLSI and FCT. We choose $L = 3D$ for FCT for all experiments unless otherwise specified.

To demonstrate the scalability and efficiency of the FCT algorithm versus the DCT-based and RLSI-based approaches, we compare against function samples for three different challenging families of functions as detailed in Subsection 4.3.

4.3. Numerical Experiments Function Families. For our numerical experiments, we used three families of functions as detailed below; these three function families are studied extensively in Subsection 4.5 against the methods as detailed above in Subsection 4.2. These function families are chosen for their challenging complexity in approximation, interpolation, and scalability by FCT, DCT-based, and RLSI methods, providing a clear and fair comparison of all three methods.

4.3.1. Function Class #1 (Runge Function). The first family $f_1(\cdot)$ can be parameterized as:

$$f_1(\boldsymbol{x}) = \frac{1}{1 + 10\|\boldsymbol{x}\|_2^2}.$$

This function is challenging to approximate accurately due to the nearby pole in the complex extension of f_1 from $\mathbb{R}^D$ to $\mathbb{C}^D$ located at $\pm\frac{i}{\sqrt{10}}\boldsymbol{e}_i$ for i^{th} unit vector, $\boldsymbol{e}_i$. Results for experiments using this function can be seen in Subsection 4.5.1.

4.3.2. Function Class #2 (Oscillatory Function). The second family of functions, $f_2(\cdot)$, are parameterized as:

(4.1) $$f_2(\boldsymbol{x}) = \sin(3\cos(3\exp(\|\boldsymbol{x}\|_2^2))) + \exp(\sin(3(\boldsymbol{x} \cdot \boldsymbol{1}))).$$

These functions are analytic, but they exhibit a complex behavior in the region $[-1,1]^D$. In this case, accurate interpolation requires high degree, and therefore many coefficients, making this family challenging computationally. Results for experiments using this function can be seen in Subsection 4.5.2.

4.3.3. Function Class #3 (Sparse Support Function). Finally, we consider a third family of functions $f_3(\cdot)$, which have *sparse support*; i.e., $\mathcal{N}$ has little structure and is a sparse subset of $S_{d,D}^\infty$. To generate elements of this family, we fix a dimension D and degree d and randomly select N multi-indices within $S_{d,D}^\infty$. We then assign random values in $[-1,1]$ to the coefficients corresponding to the chosen multi-indices. Results for experiments using this function can be seen in Subsection 4.5.3.

4.4. Numerical Implementation. We have implemented the FCT algorithm described in Subsection 3.4 in `C++`. We use the `fftw` package [16] for its optimized DCT implementation and `Eigen` [23] for its linear algebra routines, including its CG method. All experiments were performed with a single core on a machine with an AMD EPYC 7452 processor and 256GB of RAM.

4.5. Numerical Experiment Studies on Scaling and Accuracy. Using our FCT implementation, we provide the specific results of our numerical experiments against the methods detailed in Subsection 4.2 (i.e., DCT-based and RLSI) on the three function families described in Subsection 4.3. We begin with Function Family #1 (Runge Function $f_1(\cdot)$) as described in Subsection 4.3.1.

4.5.1. Runge Function ($f_1(\cdot)$). We first focus on Function Family #1 as described in Subsection 4.3.1 in order to study scaling of FCT against DCT-based and RLSI methods.

Scaling Comparison with $f_1(\cdot)$. To compare the scaling behaviors of each method, we continue to focus on $f_1(\cdot)$ and construct polynomials in $\mathbb{P}_{d,1}^D$ with random coefficients $c_{\boldsymbol{n}}$ in $[-1,1]^D$ and attempt to recover them with FCT, the DCT-based approach, and RLSI.

Each of these algorithms should recover the polynomials accurately with varying complexity. We test polynomials of degree $d = 3, 6$ and in dimensions $D = 2, \ldots, 25$ with $N = |\mathcal{S}_{d,D}^1| = \binom{D+d}{d}$ coefficients.

In Figure 2, we report the wall time of each method in terms of the number of nonzero coefficients N for a given value of d and D. We do not report timing results when the algorithm runs out of system memory. We see that the runtime of the DCT approach grows exponentially with D, quickly running out of system memory even for $d = 3$. The RLSI approach fares better, exhibiting roughly $O(N^2)$ scaling as expected; however, it still runs out of memory when $d = 6$ while FCT is able to recover the most polynomials in high dimensions. It is worth highlighting that, in low dimensions, RLSI and the DCT approach are in fact faster than FCT.

Accuracy Comparison. In Figure 3, we compare the approximation accuracy of the three methods by providing the mean ℓ_2 coefficient error: for a computed solution $\boldsymbol{c}$ and known solution $\tilde{\boldsymbol{c}}$, we report $\|\boldsymbol{c} - \tilde{\boldsymbol{c}}\|_2/N$. Tests that run out of system memory in this range are not plotted. The target function for interpolation in this case was generated by randomly selecting Chebyshev coefficients between $[-1,1]$ in the specified dimensions and degrees.

The error of each method is essentially flat (negative slopes are due to increasing N). The DCT-based approach exactly recovers the polynomial coefficients as expected, whereas the accuracy of RLSI is bounded by the CG error tolerance, since the system matrix is full-rank. The jump in error for FCT in the degree case is caused by a phase
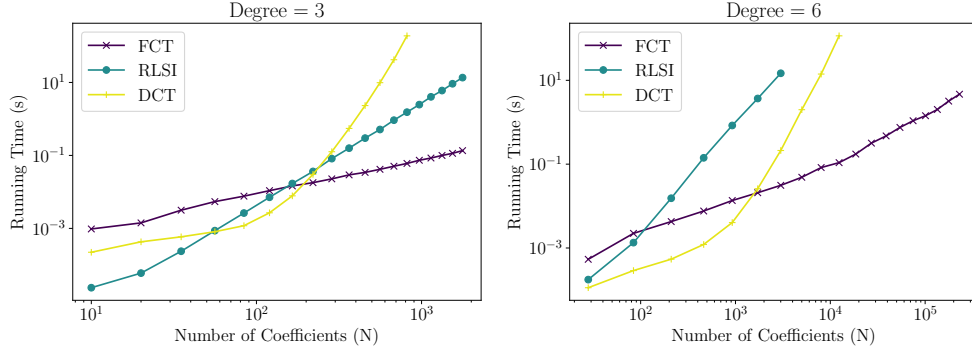
Fig. 2: Wall time versus number of coefficients $N = |\mathcal{S}_{d,D}^1| = \binom{D+d}{d}$ for the interpolation of polynomials with degree $d = 3$ (left) and at $d = 6$ (right), and varying dimension. The FCT significantly outperforms the DCT and RLSI with a crossover size of roughly $N = 2 \times 10^2$. Each point on each curve corresponds to dimension $D = 2, \ldots, 25$. Curves with fewer points indicate the algorithm in question ran out of memory.
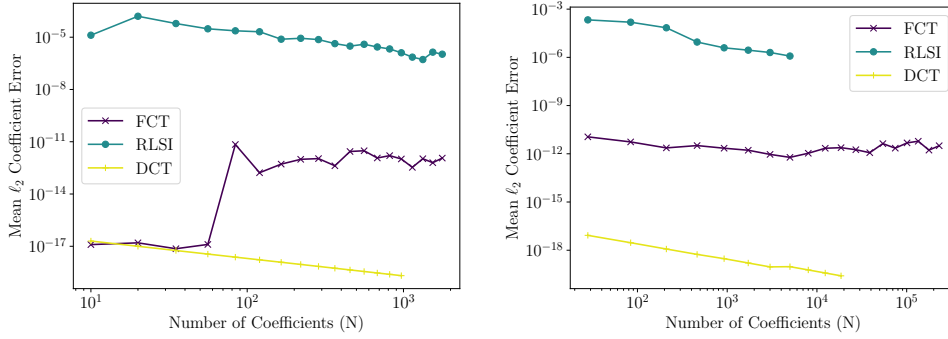

Fig. 3: Mean ℓ_2 coefficient error versus number of coefficients $N = |\mathcal{S}_{d,D}^1| = \binom{D+d}{d}$ for the interpolation of polynomials with degree $d = 3$ (left) and at $d = 6$ (right), and varying dimension. Each point on each curve corresponds to dimension $D = 2, \ldots, 25$.

change as we move from low condition number aliasing matrices for small problem instances, to ones with more moderate condition numbers. We observe that FCT becomes more efficient than the DCT-based approach and RLSI at $N \approx 2 \times 10^2$ for $d = 3$ and $N \approx 2 \times 10^3$ for $d = 6$, i.e., $D = 10$ for $d = 3$ and $D = 7$ for $d = 6$.

4.5.2. Oscillatory Function $(f_2(\cdot))$. To investigate the accuracy of function approximations produced with FCT, we consider the function $f_2(\cdot)$ as described in Subsection 4.3.2, where we choose $D = 5$. A 2D slice of f_2 through the origin can be seen in Figure 4-Right. We approximate f_2 with a polynomial of bounded Euclidean

degree d_E; i.e., $N = |\mathcal{S}_{d,D}^2|$ and compare the L^∞ absolute error.[1]. We choose a CG tolerance at machine precision to ensure that all error is solely due to FCT.
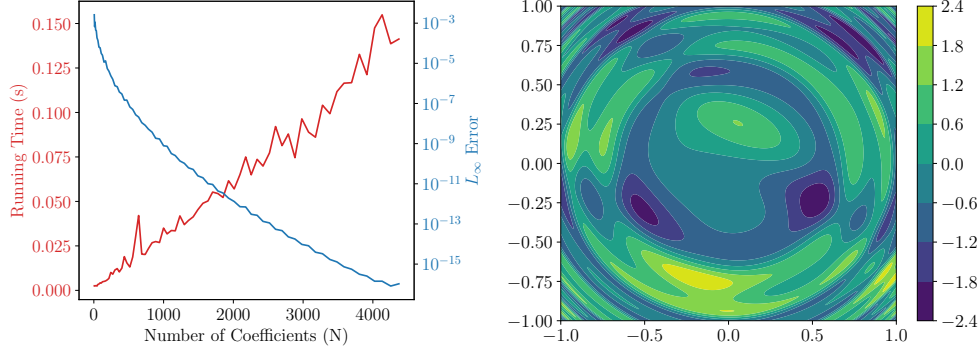

Fig. 4: *Left*: FCT wall time and L^∞ absolute error for f_2, with bounded Euclidean degree $d_E = 2, \ldots, 50$; *Right*: a horizontal slice through the origin of f_2. We observe rapid error decay with increasing Euclidean degree, while the computational cost scales quasi-linearly.

In Figure 4-Left, we plot the relative error and wall time of FCT in terms of the number of coefficients N, determined by a bounded Euclidean degree. We see that the L^∞ error decays rapidly with N, and therefore with d_E, as expected from [39]. By $d_E = 50$ ($N \approx 5000$), we have resolved f_2 to machine precision, which is a behavior consistent with a standard Chebyshev interpolation scheme. We also observe quasi-linear scaling in wall time with increasing N, requiring only a $1/10s$ to recover 5000 coefficients to machine precision.

4.5.3. Sparse Coefficient Recovery $(f_3(\cdot))$**.** We can use the FCT algorithm to approximate a function f using an expansion consisting of specific subset of Chebyshev coefficients as described in Subsection 4.3.3. Provided we know the location of the nonzero coefficients in the expansion of our function of interest, FCT will recover these rapidly and with high accuracy. Suppose we know that the target function f is a polynomial with N non-zero coefficients whose locations we know in advance. In other words, we know the multi-indices $\boldsymbol{n}^{(i_1)}, \boldsymbol{n}^{(i_2)}, \ldots, \boldsymbol{n}^{(i_N)}$ which have non-zero coefficients $c_{\boldsymbol{n}^{(i_j)}}$.

We compare the walltime for FCT and RLSI in Figure 5 in dimension $D = 100$. FCT recovers $N = 10^7$ Chebyshev coefficients in ~ 10 seconds, while RLSI runs out of memory at $N = 10^4$. Note that A is constructed via equation Equation (3.26), and only contains columns corresponding to the nonzero multi-indices (i.e., only N columns).

5. Conclusions. In this paper, we have introduced the Fast Chebyshev Transform (FCT), an algorithm for computing the Chebyshev coefficients of a polynomial from the knowledge of the location of its nonzero Chebyshev coefficients with applications to Chebyshev interpolation. We have theoretically and empirically demonstrated quasi-linear scaling and competitive numerical accuracy on high-dimensional problems. We have compared the performance of our method with that of commonly-used alternative

---

[1]To approximate the L^∞ error, we compute the maximum error over a random sampling of 5000 points in $[-1, 1]^D$.
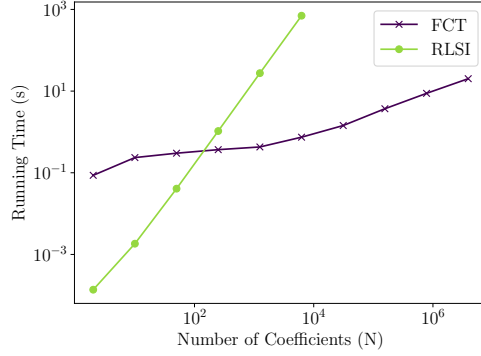
Fig. 5: Comparison of the computation time of the FCT and RLSI algorithms for the recovery of the Chebyshev coefficients of a polynomials in dimension $D = 100$ having an increasing number of sparse coefficients.

techniques based on least-squares (RLSI) and the Discrete Cosine Transform (DCT). We have further demonstrated significant speedups.

Moving forward, our most immediate next step is to improve the sampling scheme of FCT to further reduce the number of required function samples. We also plan to apply FCT to problems in polynomial optimization and apply the sampling scheme to more general transforms applied to smooth functions, such as the NUFFT (Non-Uniform Fast Fourier Transform) and Gaussian process regression [20, 19].

6. Data Availability Statement. The data produced for the redaction of this paper will be made available under reasonable requests satisfying the policy of Qualcomm Technologies Inc. for the public release of internal research material.

Appendix A. Proofs.

A.1. Proof of aliasing identity.

LEMMA A.1. *Let* $1 < N \in \mathbb{N}$. *Then,*

(A.1)
$$\frac{1}{N} \sum_{k=0}^{N-1} \cos\left(\frac{n(k+1/2)\pi}{N}\right) \cos\left(\frac{m(k+1/2)\pi}{N}\right) = \frac{1}{2}\Delta(m+n) + \frac{1}{2}\Delta(m-n)$$

where,

(A.2)
$$\Delta(l) = \begin{cases} 1 & \text{if } l \bmod (4N) \equiv 0 \\ -1 & \text{if } l \bmod (4N) \equiv 2N \\ 0 & \text{o.w.} \end{cases}$$

Proof. Consider,

(A.3) $$\frac{1}{N} \sum_{n=0}^{N-1} \cos\left(\frac{m(n+1/2)\pi}{N}\right) \cos\left(\frac{k(n+1/2)\pi}{N}\right)$$

(A.4) $$= \frac{1}{N} \sum_{n=0}^{N-1} \left[\frac{1}{2}\cos\left(\frac{(n+1/2)(m+k)\pi}{N}\right) + \frac{1}{2}\cos\left(\frac{(n+1/2)(m-k)\pi}{N}\right)\right]$$

16

Then, consider,

$$(A.5) \qquad \sum_{n=0}^{N-1} \cos\left(\frac{(n+1/2)l\pi}{N}\right) = \frac{e^{i\frac{\pi}{2N}l}}{2} \sum_{n=0}^{N-1} e^{i\frac{\pi}{N}nl} + \frac{e^{-i\frac{\pi}{2N}l}}{2} \sum_{n=0}^{N-1} e^{-i\frac{\pi}{N}nl}$$

$$(A.6) \qquad = \left(\frac{e^{i\frac{\pi}{2}l}}{2} + \frac{e^{-i\frac{\pi}{2}l}}{2}\right) \frac{\sin(\pi l/2)}{\sin(\pi l/(2N))}$$

$$(A.7) \qquad = \cos(\pi l/2) \frac{\sin(\pi l/2)}{\sin(\pi l/(2N))}$$

$$(A.8) \qquad = \frac{1}{2} \frac{\sin(\pi l)}{\sin(\pi l/(2N))}$$

since (Dirichlet kernel),

$$(A.9) \qquad \sum_{n=0}^{N-1} e^{inx} = e^{i\frac{N-1}{2}x} \frac{\sin(Nx/2)}{\sin(x/2)}$$

and since,

$$(A.10) \qquad \cos(\pi l/2)\sin(\pi l/2) = \frac{1}{2}\sin(\pi l) + \frac{1}{2}\sin(0) = \frac{1}{2}\sin(\pi l)$$

Therefore,

$$(A.11) \qquad \frac{1}{N} \sum_{n=0}^{N-1} \left[\frac{1}{2}\cos\left(\frac{(n+1/2)(m+k)\pi}{N}\right) + \frac{1}{2}\cos\left(\frac{(n+1/2)(m-k)\pi}{N}\right)\right]$$

$$(A.12) \qquad = \frac{1}{2}\left(\frac{1}{2N}\frac{\sin(\pi(m+k))}{\sin(\pi(m+k)/(2N))} + \frac{1}{2N}\frac{\sin(\pi(m-k))}{\sin(\pi(m-k)/(2N))}\right)$$

$$(A.13) \qquad = \frac{1}{2}\Delta(m+k) + \frac{1}{2}\Delta(m-k)$$

and the result follows after defining,

$$(A.14) \qquad \Delta(l) := \frac{1}{2N}\frac{\sin(\pi l)}{\sin(\pi l/(2N))} = \begin{cases} 1 & \text{if } l \bmod(4N) \equiv 0 \\ -1 & \text{if } l \bmod(4N) \equiv 2N \\ 0 & \text{o.w.} \end{cases}$$

for all $l \in \mathbb{Z}$. $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad$ $\square$

REFERENCES

[1] N. AHMED, T. NATARAJAN, AND K. R. RAO, *Discrete cosine transform*, IEEE transactions on Computers, 100 (1974), pp. 90–93.
[2] Z. ALLEN-ZHU, Y. LI, AND Z. SONG, *A convergence theory for deep learning via over-parameterization*, in International Conference on Machine Learning, PMLR, 2019, pp. 242–252.
[3] Z. BATTLES AND L. N. TREFETHEN, *An extension of matlab to continuous functions and operators*, SIAM Journal on Scientific Computing, 25 (2004), pp. 1743–1770.
[4] M. BEBENDORF, *Hierarchical matrices: a means to efficiently solve elliptic boundary value problems*, Lecture Notes in Computational Science and Engineering, (2007).
[5] M. BEBENDORF, *Adaptive cross approximation of multivariate functions*, Constructive approximation, 34 (2011), pp. 149–179.

[6] M. BELKIN, D. HSU, S. MA, AND S. MANDAL, *Reconciling modern machine-learning practice and the classical bias–variance trade-off*, Proceedings of the National Academy of Sciences, 116 (2019), pp. 15849–15854.

[7] M. BELKIN, A. RAKHLIN, AND A. B. TSYBAKOV, *Does data interpolation contradict statistical optimality?*, in The 22nd International Conference on Artificial Intelligence and Statistics, PMLR, 2019, pp. 1611–1619.

[8] H.-J. BUNGARTZ AND M. GRIEBEL, *Sparse grids*, Acta numerica, 13 (2004), pp. 147–269.

[9] W.-H. CHEN, C. SMITH, AND S. FRALICK, *A fast computational algorithm for the discrete cosine transform*, IEEE Transactions on communications, 25 (1977), pp. 1004–1009.

[10] X. CHEN, Q. DAI, AND C. LI, *A fast algorithm for computing multidimensional dct on certain small sizes*, IEEE transactions on signal processing, 51 (2003), pp. 213–220.

[11] H. CHENG, V. ROKHLIN, AND N. YARVIN, *Nonlinear optimization, quadrature, and interpolation*, SIAM Journal on Optimization, 9 (1999), pp. 901–923.

[12] P. R. CONRAD AND Y. M. MARZOUK, *Adaptive smolyak pseudospectral approximations*, SIAM Journal on Scientific Computing, 35 (2013), pp. A2643–A2670.

[13] L. DEMANET AND A. TOWNSEND, *Stable extrapolation of analytic functions*, Foundations of Computational Mathematics, 19 (2019), pp. 297–331.

[14] M. DEVILLE AND E. MUND, *Chebyshev pseudospectral solution of second-order elliptic equations with finite element preconditioning*, Journal of Computational Physics, 60 (1985), pp. 517–533.

[15] R. DEVORE, B. HANIN, AND G. PETROVA, *Neural network approximation*, Acta Numerica, 30 (2021), pp. 327–444.

[16] M. FRIGO AND S. G. JOHNSON, *Fftw: An adaptive software architecture for the fft*, in Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP'98 (Cat. No. 98CH36181), vol. 3, IEEE, 1998, pp. 1381–1384.

[17] G. H. GOLUB AND C. F. VAN LOAN, *Matrix computations*, JHU press, 2013.

[18] L. GRASEDYCK, D. KRESSNER, AND C. TOBLER, *A literature survey of low-rank tensor approximation techniques*, GAMM-Mitteilungen, 36 (2013), pp. 53–78.

[19] L. GREENGARD AND J.-Y. LEE, *Accelerating the nonuniform fast fourier transform*, SIAM review, 46 (2004), pp. 443–454.

[20] P. GREENGARD, M. RACHH, AND A. BARNETT, *Equispaced fourier representations for efficient gaussian process regression from a billion data points*, arXiv preprint arXiv:2210.10210, (2022).

[21] M. GRIEBEL AND J. HAMAEKERS, *Fast discrete fourier transform on generalized sparse grids*, in Sparse Grids and Applications-Munich 2012, Springer, 2014, pp. 75–107.

[22] M. GRIEBEL AND J. HAMAEKERS, *Generalized sparse grid interpolation based on the fast discrete fourier transform*, in Sparse Grids and Applications-Munich 2018, Springer, 2021, pp. 53–68.

[23] G. GUENNEBAUD, B. JACOB, ET AL., *Eigen v3*. http://eigen.tuxfamily.org, 2010.

[24] W. HACKBUSCH, *Tensor spaces and numerical tensor calculus*, vol. 42, Springer, 2012.

[25] B. HASHEMI AND L. N. TREFETHEN, *Chebfun in three dimensions*, SIAM Journal on Scientific Computing, 39 (2017), pp. C341–C363.

[26] P. KABAL AND R. P. RAMACHANDRAN, *The computation of line spectral frequencies using chebyshev polynomials*, IEEE Transactions on Acoustics, Speech, and Signal Processing, 34 (1986), pp. 1419–1426.

[27] D. S. KERSHAW, *The incomplete cholesky—conjugate gradient method for the iterative solution of systems of linear equations*, Journal of computational physics, 26 (1978), pp. 43–65.

[28] R.-C. LI, *Near optimality of chebyshev interpolation for elementary function computations*, IEEE Transactions on Computers, 53 (2004), pp. 678–687.

[29] J. MASON, *Chebyshev polynomial approximations for the l-membrane eigenvalue problem*, SIAM Journal on Applied Mathematics, 15 (1967), pp. 172–186.

[30] W. F. MOORE, M. ROGERS, S. SATHER-WAGSTAFF, ET AL., *Monomial ideals and their decompositions*, Springer, 2018.

[31] P. NAKKIRAN, G. KAPLUN, Y. BANSAL, T. YANG, B. BARAK, AND I. SUTSKEVER, *Deep double descent: Where bigger models and more data hurt*, Journal of Statistical Mechanics: Theory and Experiment, 2021 (2021), p. 124003.

[32] I. V. OSELEDETS, *Tensor-train decomposition*, SIAM Journal on Scientific Computing, 33 (2011), pp. 2295–2317.

[33] M. J. POWELL, *On the maximum errors of polynomial approximations defined by interpolation and by least squares criteria*, The Computer Journal, 9 (1967), pp. 404–407.

[34] H. D. SHERALI AND H. WANG, *Global optimization of nonconvex factorable programming problems*, Mathematical Programming, 89 (2001), pp. 459–478.

[35] D. I. SHUMAN, P. VANDERGHEYNST, AND P. FROSSARD, *Chebyshev polynomial approximation for*

distributed signal processing, in 2011 International Conference on Distributed Computing in Sensor Systems and Workshops (DCOSS), IEEE, 2011, pp. 1–8.

[36] D. J. SMITH AND M. K. VAMANAMURTHY, *How small is a unit ball?*, Mathematics Magazine, 62 (1989), pp. 101–107.

[37] S. A. SMOLYAK, *Quadrature and interpolation formulas for tensor products of certain classes of functions*, in Doklady Akademii Nauk, vol. 148, Russian Academy of Sciences, 1963, pp. 1042–1045.

[38] A. TOWNSEND AND L. N. TREFETHEN, *An extension of chebfun to two dimensions*, SIAM Journal on Scientific Computing, 35 (2013), pp. C495–C518.

[39] L. TREFETHEN, *Multivariate polynomial approximation in the hypercube*, Proceedings of the American Mathematical Society, 145 (2017), pp. 4837–4844.

[40] L. N. TREFETHEN, *Spectral methods in MATLAB*, SIAM, 2000.

[41] L. N. TREFETHEN, *Cubature, approximation, and isotropy in the hypercube*, SIAM Review, 59 (2017), pp. 469–491.

[42] L. N. TREFETHEN, *Approximation Theory and Approximation Practice, Extended Edition*, SIAM, 2019.

[43] Y. XIE, H.-H. CHOU, H. RAUHUT, AND R. WARD, *Overparameterization and generalization error: weighted trigonometric interpolation*, SIAM Journal on Mathematics of Data Science, 4 (2022), pp. 885–908.